

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Odabrane teme iz područja računalne grafike
Tehnička dokumentacija
Verzija <1.0>

Studentski tim: Helena Hrženjak
Dana Dodigović
Adrian Komadina
Tin Srnić
Marin Hrkec
Filip Pavletić
Luka Radivoj

Nastavnik: Prof. dr. sc. Željka Mihajlović

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Sadržaj

| | |
|---|-----------|
| 1. Uvod | 4 |
| 2. Proceduralno generiranje terena algoritmom pokretnih kocki | 4 |
| 2.1 Opis razvijenog proizvoda | 4 |
| 2.1.1 Algoritam pokretnih kocki | 4 |
| <i>Slika 2.1.1.7: Prikaz generiranog terena</i> | 8 |
| 2.1.2 Proceduralno generiranje | 8 |
| 2.1.3 Alati za promjenu parametara | 9 |
| 2.2 Tehničke značajke | 9 |
| 2.3 Upute za korištenje | 10 |
| 2.4 Zaključak | 10 |
| 2.5 Literatura | 10 |
| 3. 3D reprezentacija dijela svijeta temeljena na stvarnim podacima | 10 |
| 3.1 Opis razvijenog proizvoda | 10 |
| 3.2 Tehničke značajke | 12 |
| 3.3 Upute za korištenje | 19 |
| 3.4 Literatura | 21 |
| 4. Implementacija računalne igre Agar korištenjem tehnologije OpenGL | |
| inačice 3.3+ | 23 |
| 4.1 Opis razvijenog proizvoda | 23 |
| 4.2 Tehničke značajke | 24 |
| 4.3 Upute za korištenje | 27 |
| 4.4 Literatura | 28 |
| 5. Sustav animacije u OpenGL-u | 28 |
| 5.1 Opis razvijenog proizvoda | 28 |
| 5.2 Tehničke značajke | 29 |
| 5.3 Upute za korištenje | 32 |

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

| | |
|--|-----------|
| 5.4 Literatura | 33 |
| 6. Generiranje prikaza virtualne scene algoritmima praćenja zrake i praćenja puta | 33 |
| 6.1 Opis razvijenog proizvoda | 33 |
| 6.2 Tehničke specifikacije | 35 |
| 6.2.1 Ray Tracer | 36 |
| 6.2.2 Progressive Sampling Shader | 40 |
| 6.2.3 Ray Tracer Master Node | 41 |
| 6.2.4 Triangle Mesh Material | 42 |
| 6.3 Upute za korištenje | 43 |
| 6.4 Literatura | 46 |
| 7. Vizualizacija i simulacija ekosustava građenog genetskim algoritmom | 46 |
| 8. Izračun i vizualizacija podataka dobivenih skeniranjem ultrazvučnom sondom | 51 |
| 8.1 Opis razvijenog proizvoda | 51 |
| 8.2 Tehničke značajke | 51 |
| 8.3 Upute za korištenje | 53 |
| 8.4 Literatura | 55 |

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Tehnička dokumentacija

1. Uvod

Ovaj projekt je skupina manjih projekata vezanih uz računalnu grafiku i njezine primjene. Popis pojedinačnih projekata je sjedeći:

- Proceduralno generiranje terena algoritmom pokretnih kocki – **Tin Srnić**
- 3D reprezentacija dijela svijeta temeljena na stvarnim podacima – **Adrian Komadina**
- Implementacija računalne igre Agar korištenjem tehnologije OpenGL inačice 3.3+ - **Dana Dodigović**
- Sustav animacije u OpenGL-u – **Marin Hrkec**
- Generiranje prikaza virtualne scene algoritmima praćenja zrake i praćenja puta – **Luka Radivoj**
- Vizualizacija i simulacija ekosustava građenog genetskim algoritmom – **Filip Pavletić**
- Izračun i vizualizacija podataka dobivenih skeniranjem ultrazvučnom sondom – **Helena Hrženjak**

2. Proceduralno generiranje terena algoritmom pokretnih kocki

2.1 Opis razvijenog proizvoda

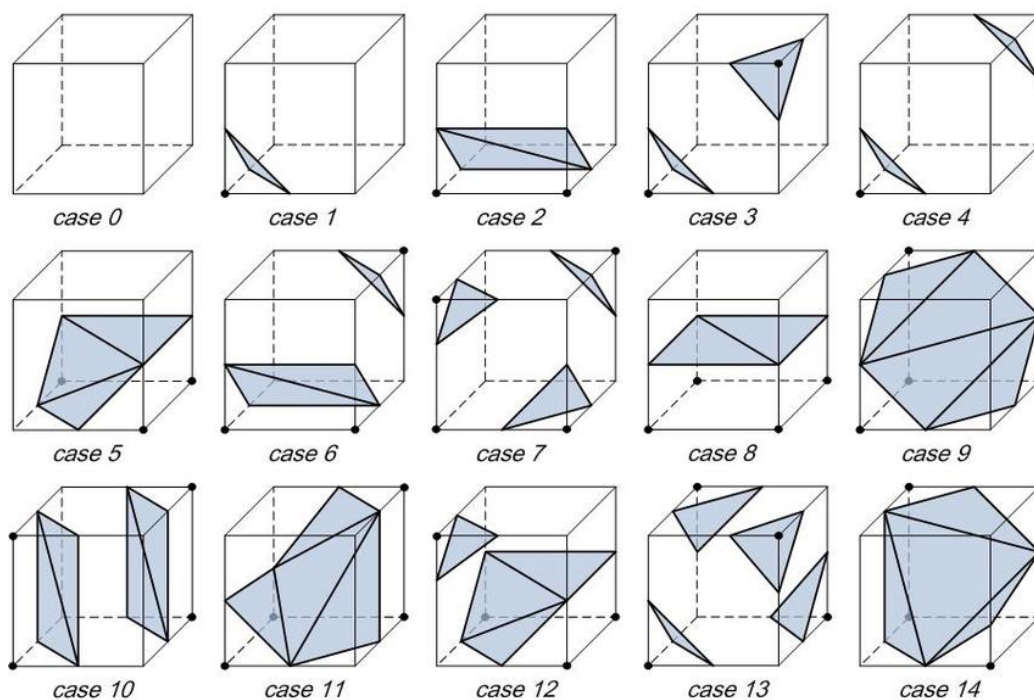
Ovaj projekt se fokusira na proceduralnu generaciju terena koja stvara teren oko igrača tijekom njegovog kretanja kroz virtualni svijet igre. Generiranje se obavlja pomoću algoritma pokretnih kocki koji pruža mogućnost stvaranja 3D terena s podzemnim i nadzemnim elementima.

2.1.1 Algoritam pokretnih kocki

Algoritmom pokretnih kocki generiraju se 3D tereni podjelom prostora na mnoštvo manjih kocaka. Na temelju vrijednosti dodijeljenih vrhovima kocki određuje se oblik

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

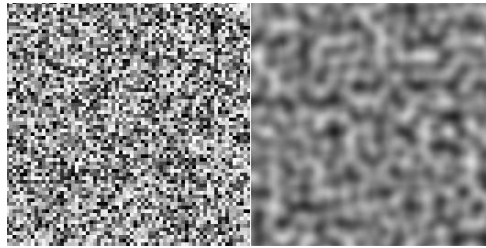
generiranog terena u danoj kocki. Ukoliko je zadana vrijednost ispod neke prethodno definirane granice, taj će se vrh kocke proglašiti „ugašenim“, a u obrnutom slučaju „upaljenim“ [1]. Na osnovi stanja svih vrhova unutar pojedinačne kocke algoritam bira oblik terena unutar iste. Postoji 15 jedinstvenih oblika prikazanih slikom 2.1.1.1 te se temeljem stanja vrhova kocke mijenja njihova orijentacija. Na taj se način može dobiti 2^8 različitih konačnih izgleda terena [2].



Slika 2.1.1.1: Prikaz 15 jedinstvenih slučajeva.

Vrijednosti za vrhove biramo raznim algoritmima, no u ovom je projektu odabran Perlinov šum modificiran za rad u 3 dimenzije. Algoritam Perlinovog šuma generira pseudo slučajne vrijednosti, ali na način da prijelazi između susjednih vrijednosti ne budu nagli te stoga i nerealistični. Taj efekt se postiže tako da se prvo generiraju nasumične vrijednosti u svim vrhovima te se one potom izgladuju kao što je prikazano na slici 2.1.1.2 [3].

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |



Slika 2.1.1.2: Izgladivanje nasumičnih vrijednosti pomoću Perlinovog šuma

Generiranje vrijednosti započinje stvaranjem pseudo slučajne vrijednosti pomoću funkcije čiji je kod prikazan na slici 2.1.1.3. Funkcija može biti proizvoljna, no njeno bitno svojstvo je da za iste vrijednosti x , y i z uvijek vraća istu vrijednost.

```
float GetNoise(int x, int y, int z) const {
    int n = x * 347 + z * 59 + y * 919 + m_seed;
    n = (n << 13) ^ n;
    return (1.0f - ((n * ((n * n * 15731) + 789221) + 1376312589) & 0x7fffffff) / 1073741824.0f);
}
```

Slika 2.1.1.3: Kod generatora pseudo slučajnih brojeva

Daljnji koraci algoritma fokusiraju se na izgladivanje vrijednosti koje generiramo funkcijom `GetNoise`. Prvi korak izgladivanja uzima u obzir ne samo vrh koji trenutno promatramo, nego još dodatno i njegovih 26 susjeda. Vidljivo u kodu na slici 2.1.1.4, funkcija `GetSmoothNoise` generira vrijednosti za sve susjedne vrhove. Svaki susjedni vrh doprinosi konačnoj vrijednosti proporcionalno njegovoj udaljenosti od promatranog vrha.

```
float GetSmoothNoise(int x, int y, int z) const {
    float totalNoise = 0.0f;
    for (int i = -1; i <= 1; ++i) {
        for (int j = -1; j <= 1; ++j) {
            for (int k = -1; k <= 1; ++k) {
                int distance = abs(i) + abs(j) + abs(k);
                if (distance == 0) totalNoise += GetNoise(x + i, y + j, z + k) / 4.0f;
                else if (distance == 1) totalNoise += GetNoise(x + i, y + j, z + k) / 8.0f;
                else if (distance == 2) totalNoise += GetNoise(x + i, y + j, z + k) / 16.0f;
                else if (distance == 3) totalNoise += GetNoise(x + i, y + j, z + k) / 32.0f;
            }
        }
    }
    return totalNoise;
}
```

Slika 2.1.1.4: Kod prve faze izgladivanja

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Prelazimo na drugu fazu izgladivanja koristeći kosinusnu interpolaciju. Cilj ove faze, čiji kod je vidljiv na slici 2.1.1.5, je dodatno izgladiti teren kako bi eliminirali sve nagle prijelaze. Za razliku od prethodne dvije funkcije, ulazi x, y i z su tipa *float* kako bi mogli iskoristiti svojstva funkcije kosinus. Koristeći već navedenu funkciju *GetSmoothNoise* generiramo vrijednosti vrhova za jednu kocku. 8 vrijednosti kocke svodimo na 1 konačnu vrijednost tako da kosinusnom interpolacijom 8 vrhova svodimo na 4, 4 na 2 i konačno 2 na 1 [4].

```
float GetInterpolatedNoise(float x, float y, float z) const {
    int intX = (int)x;
    int intY = (int)y;
    int intZ = (int)z;
    float fracX = x - intX;
    float fracY = y - intY;
    float fracZ = z - intZ;

    float v1 = GetSmoothNoise(intX + 0, intY + 0, intZ + 0);
    float v2 = GetSmoothNoise(intX + 1, intY + 0, intZ + 0);
    float v3 = GetSmoothNoise(intX + 0, intY + 0, intZ + 1);
    float v4 = GetSmoothNoise(intX + 1, intY + 0, intZ + 1);
    float v5 = GetSmoothNoise(intX + 0, intY + 1, intZ + 0);
    float v6 = GetSmoothNoise(intX + 1, intY + 1, intZ + 0);
    float v7 = GetSmoothNoise(intX + 0, intY + 1, intZ + 1);
    float v8 = GetSmoothNoise(intX + 1, intY + 1, intZ + 1);

    float i1 = Interpolate(v1, v2, fracX);
    float i2 = Interpolate(v3, v4, fracX);
    float i3 = Interpolate(v5, v6, fracX);
    float i4 = Interpolate(v7, v8, fracX);

    float i5 = Interpolate(i1, i3, fracY);
    float i6 = Interpolate(i2, i4, fracY);

    return Interpolate(i5, i6, fracZ);
}

float Interpolate(float a, float b, float blend) const {
    double theta = blend * M_PI;
    float f = (float)(1.0f - cos(theta)) * 0.5f;
    return a * (1.0f - f) + b * f;
}
```

Slika 2.1.1.5: Kod interpoliranja

Završna faza generiranja, čiji kod je vidljiv na slici 2.1.1.6, dodaje detalje terenu. Funkcija započinje pomicanjem koordinata x, y i z za neki veliki broj kako bi izbjegli kasnije probleme s dijeljenjem i dobivanjem premalih brojeva. Svaka iteracija petlje u funkciji generira teren sa određenom sinusnom funkcijom neke frekvencije i amplitude. Sinusi s nižim frekvencijama imat će veće amplitude kako bi u konačnom terenu dobili izražene uspone i padove, dok sinusi s višim frekvencijama dobivaju manje amplitude kako teren ne bi bio posve gladak, nego imao blage izbočine na površini. Na slici

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

2.1.1.7 je konačni prikaz isječka generiranog terena.

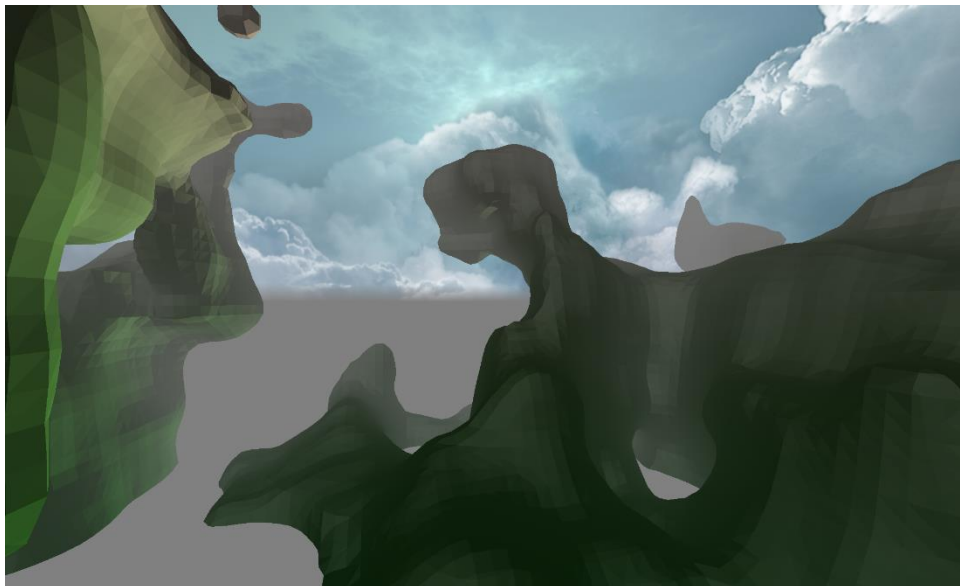
```
float GenerateHeight(int x, int y, int z) const {
    int xPos = x + INT_MAX / 500;
    int yPos = y + INT_MAX / 500;
    int zPos = z + INT_MAX / 500;

    float total = 0;

    for (int x = 0; x < m_octaves; x++) {
        float freq = (float) (pow(2, x) / m_baseSmoothness);
        float amp = (float) pow(m_roughness, x) * m_amplitude;
        total += GetInterpolatedNoise(xPos * freq, yPos * freq, zPos * freq) * amp;
    }

    return total;
}
```

Slika 2.1.1.6: Završna funkcija generiranja vrijednosti vrha

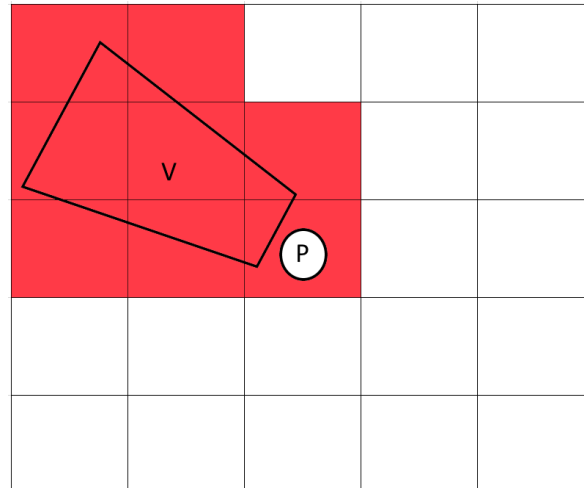


Slika 2.1.1.7: Prikaz generiranog terena

2.1.2 Proceduralno generiranje

Proceduralno generiranje terena je proces prilikom kojeg se oko igrača stvara teren kako se on kreće kroz igru. Teren je podijeljen na dijelove od kojih se samo oni najbliži igraču kratkotrajno pohranjuju u memoriju. Od dijelova terena pohranjenih u memoriji na ekran se ne iscrtavaju svi, nego samo oni koji su u vidnom polju igrača.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |



Slika 2.1.2.1: Prikaz dijelova terena koje treba iscrtati

Na slici 2.1.2.1 obojano crvenom su oni dijelovi terena koji se iscrtavaju, slovom P označen je igrač, a slovom V njegovo vidno polje. Kako se igrač kreće unutra igre, dijelovi terena kojima se primiče se učitavaju u memoriju, dok oni od kojih se udaljava brišu.

2.1.3 Alati za promjenu parametara

Za jednostavniju konfiguraciju terena postavljeni su sljedeći parametri:

Granična vrijednost vrhova terena – vrijednost koja određuje koji su vrhovi „upaljeni“, a koji „ugašeni“

Amplituda – maksimalna dopuštena amplituda terena

Oktave – povećanjem oktava povećava se razina detalja terena

Glatkoća – koliko je teren gladak

Grubost – izraženost nepravilnosti na terenu

2.2 Tehničke značajke

Program je napisan u C++ programskom jeziku koristeći GLFW i GLEW razvojne okvire. Računalo na kojem je program pisan i testiran odlikuju sljedeće značajke: procesor intel i7 pete generacije, grafička kartica Nvidia GeForce GTX 770 i 16GB RAM. Program ostvaruje 30 FPS.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

2.3 Upute za korištenje

Datoteke je potrebno kopirati na računalo i pokrenuti. Kada je aplikacija pokrenuta, prema želji se mogu mijenjati parametri opisani u poglavlju 1.3.

2.4 Zaključak

Algoritam pokretnih kocki je veoma pogodan za generiranje terena i njegovu daljnju modifikaciju. Jedna od njegovih glavnih prednosti je brzina s obzirom na povećanje volumena. Može se koristiti u prikazivanju medicinskih podataka poput CT-a i MR-a. Fleksibilan je u svojoj primjeni zbog toga što je moguće postići različite oblike složenih geometrija.

2.5 Literatura

[1] Vizualizacija medicinskih podataka putem Interneta
<http://www.zemris.fer.hr/predmeti/rg/diplomski/05Blazona/vizualizacija.htm> Sječanj 2021.

[2] Polygonising a scalar field <http://paulbourke.net/geometry/polygonise/> Prosinac 2020.

[3] Understanding Perlin Noise <https://adrianb.io/2014/08/09/perlinnoise.html> Prosinac 2020.

[4] Value Noise and Procedural Pattern
<https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/procedural-patterns-noise-part-1/creating-simple-1D-noise> Prosinac 2020.

3. 3D reprezentacija dijela svijeta temeljena na stvarnim podacima

3.1 Opis razvijenog proizvoda

Iz dana u dan sve više raste broj dostupnih informacija koje je moguće slobodno preuzeti s interneta o nekom području. Ti izvori podataka se protežu od nekih općenitih kao što su urbanistički podaci do onih konkretnih kao što su popis svih javnih stabala

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

nekoj grad. Sve ove podatke moguće je povezati i prikazati na određeni način. U sklopu ovog diplomskog projekta cilj je bio razviti aplikaciju koja će stvoriti 3D reprezentaciju određenog područja na temelju stvarnih preuzetih podataka.

Od stvarnih podataka koji se mogu naći uzeta su tri glavna izvora podataka. Prvi su Open Street Map (OSM) podaci iz kojih ćemo izvući informacije o putevima i građevinama. Drugi izvor je Google Elevation API preko kojeg dobivamo informaciju o stvarnoj visini neke točke na zemlji određene koordinatama. Zadnji izvor je katastar zelenila grada Zagreba preko kojeg dolazimo do informacija o poziciji javnih stabala u gradu Zagrebu.

Na samom početku aplikacije korisniku je omogućeno postavljanje geografskih parametara koji određuju za koje će se područje generirati 3D prikaz. Također korisnik može i ručno podešavati veličinu generiranog prikaza, a da pri tome budu sačuvane stvarne proporcije iz realnog svijeta. Aplikacija se spaja na internet te se pomoću HTTP protokola dohvaća jedan dio podataka vezanih visinu terena i vegetaciju, dok se drugi dio izvlači iz OSM podataka danih uz samu aplikaciju.

Stvoreni prikaz se sastoji od 3D terena čije dimenzije odgovaraju stvarnoj veličini u odabranom mjerilu kojim su pridjeljenje visinske razlike na temelju podataka dobivenih preko Google Elevation API-ja. Terenu su pridijeljene i teksture na različitim dijelovima terena koje odgovaraju poziciji putova na tom području. Putovi se razlikuju po veličini i površini od kojih su rađeni što prikazujemo drugačijim teksturama. Na terenu bit će stvoreni 3D modeli građevina sastavljeni jednostavno od 2D ploha odnosno zidova koji čine građevinu. Uz građevine bit će prikazani i 3D modeli stabala na mjestima gdje su ona javno vlasništvo grada Zagreba. Aplikacija je ograničena na prikaze područja koji spadaju unutar teritorija Republike Hrvatske uz napomenu da su podaci o vegetaciji dostupni samo za grad Zagreb pa će na drugim područjima ona izostati.

Nakon generiranja 3D prikaza korisnik se može kretati slobodno u zadanom prostoru. Tada on u svakom trenutku može napustiti aplikaciju ili se vratiti u izbornik i odabrati novo područje za koje želi generirati 3D prikaz.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

3.2 Tehničke značajke

Aplikacija je izrađena u 3D pokretaču Unity 2019.3.13. Skripte korištene u pokretaču pisane su u jeziku C#. Aplikacija u svome radu koristi još Osmosis alat koji je zapravo Java aplikacija namijenjena za procesiranje OSM podataka. Za potrebe stvaranja 3D modela stabla korišten je program Blender. Sve potrebne datoteke priložene su uz izvršni kod same aplikacije.

Tijek rada aplikacije može se razložiti u šest koraka. U prvom koraku korisnik unosi parametre potrebne za rad aplikacije, te je on opisan u poglavlju 3. Potom se stvara teren sa svojim visinskim obilježjima. U trećem koraku procesiramo OSM podatke i parsiramo podatke o vegetaciji. Nadalje iscrtavamo putove u obliku različitih tekstura koje primjenjujemo na terenu. Nakon toga se stvaraju 3D objekti koji predstavljaju građevine na tom području. U zadnjem se koraku stvaraju 3D objekti koji predstavljaju stabla.

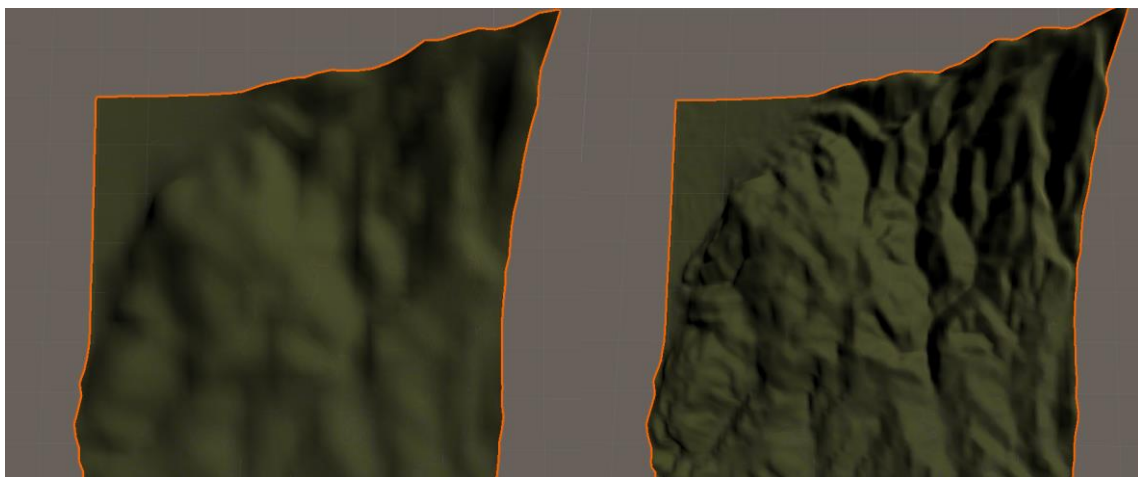
| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Generiranje terena je podijeljeno u dvije faze. U prvoj fazi instancira se objekt terena te ga se skalira na odgovarajuću veličinu temeljem unesenih koordinata zadanog područja i faktora skaliranja terena. To se radi tako da se najprije izračuna udaljenost dvije točke zadane geografskom širinom i dužinom u metrima te se onda pomnoži sa zadanim faktorom skaliranja. U drugoj fazi pozivamo preko HTTP protokola Google Elevation API. Google Elevation API je API u sklopu Google Maps platforme koji nam omogućuje da za zadanu točku u prostoru određenu geografskom širinom i dužinom dobijemo njezinu visinu u metrima. Izgled jednog odgovora dobivenog slanjem HTTP zahtjeva na Google Elevation API vidi se na slici 3.2.1. Važno je naglasiti kako kod objekta tipa terena u Unity-u nije moguće za proizvoljnu njegovu točku postaviti visinu već je unaprijed potrebno postaviti mrežu predefiniране rezolucije te se tek onda za svaku točku te mreže može postaviti visina. Postavku o rezoluciji te mreže moguće je vidjeti promjenom atributa Heightmap resolution. Kako Google Elevations API nije u potpunosti besplatan već je njegovo korištenje određeno limitom broja poslanih HTTP zahtjeva, u sklopu ovog projekta zadržat ćemo se na najmanjoj rezoluciji od 65x65 koja sama po sebi daje dovoljno dobre rezultate u odnosu na više rezolucije. Rezultat ove dvije faze generiranja terena vidljiv je na slici 3.2.2. uz usporedbu dviju različito odabranih rezolucija. Na lijevom dijelu slike moguće je vidjeti teren generiran uz rezoluciju visinske mape od 65x65 što uzrokuje trošak od 65 HTTP zahtjeva, dok je na desnom dijelu vidljiv teren s rezolucijom od 257x257 s troškom od 257 zahtjeva.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

```
{
  "results" : [
    {
      "elevation" : 4411.941894531250,
      "location" : {
        "lat" : 36.5785810,
        "lng" : -118.2919940
      },
      "resolution" : 19.08790397644043
    },
    {
      "elevation" : 1381.861694335938,
      "location" : {
        "lat" : 36.41150289067028,
        "lng" : -117.5602607523847
      },
      "resolution" : 19.08790397644043
    },
    {
      "elevation" : -84.61699676513672,
      "location" : {
        "lat" : 36.239980,
        "lng" : -116.831710
      },
      "resolution" : 19.08790397644043
    }
  ],
  "status" : "OK"
}
```

Slika 3.2.1: HTTP odgovor Google Elevation API-ja



Slika 3.2.2: Rezultat generiranja terena uz usporedbu rezolucije visinske mape

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Glavni skup podataka kojeg koristimo je Open Street Map (OSM) podaci o Hrvatskoj. OpenStreetMap je karta svijeta stvorena od raznih autora te su njezini podaci slobodni za korištenje. Preuzetu datoteku s podacima o Hrvatskoj stavit ćemo u poseban direktorij kojemu će aplikacija moći pristupiti. Ta datoteka u svom izvornom obliku nije lako čitljiva pa je njezino parsiranje teško za postići, u tu svrhu koristit ćemo Osmosis Java aplikaciju koja će nam služiti upravo za pretprocesiranje tih podataka. U Unity skripti ćemo postići to da pozovemo Windowsovu komandni interpreter (CMD), te pomoću njega pokrenemo java aplikaciju sa zadanim parametrima. Pozivanje osmosis aplikacije bi konkretno ovako izgledalo „osmosis --read-pbf croatia.osm.pbf --bounding-box top=[maxLatitude] left=[minLongitude] bottom=[minLatitude] right=[maxLongitude] --write-xml OSMFile“. Pozivom tog programa dobit ćemo ograničen OSM file za zadano područje zapisan u XML oblika pogodan za daljnje parsiranje. Dobivena XML datoteka sadrži listu čvorova s oznakom „node“ s njihovom geografskom širinom i dužinom u EPSG4326 sustavu koordinata, te listu spojnica s oznakom „way“. Svaka spojnica označava nešto u stvarnom svijetu bila to cesta ili građevina ili nešto treće označeno posebnim imenom, te listom čvorova od kojih se sastoji i popisom atributa koji ju opisuju. Isječak izgleda XML datoteke moguće je vidjeti na slici 3.2.3., na lijevom dijelu slike nalaze se čvorovi, a na desnom spojnice. Parsiranjem XML datoteke dobit ćemo listu objekata čvorova, listu objekata građevina i listu objekata cesta koje ćemo kasnije koristiti za generiranje građevina i cesta.

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version="0.6" generator="Osmosis 0.48.3">
  <bounds minlon="15.83220" minlat="45.81780" maxlon="15.93600" maxlat="45.88930" origin="0.48.3"/>
  <node id="20969180" version="7" timestamp="2016-01-20T07:38:24Z" uid="0" user="" lat="45.8272785" lon="15.9312911"/>
  <node id="26045016" version="10" timestamp="2019-09-13T21:44:04Z" uid="0" user="" lat="45.8190276" lon="15.8703149"/>
  <node id="26045020" version="3" timestamp="2019-12-18T15:53:53Z" uid="0" user="" lat="45.8193721" lon="15.8768129"/>
  <node id="26045024" version="5" timestamp="2020-06-16T08:10:13Z" uid="0" user="" lat="45.8178189" lon="15.8496579"/>
  <node id="26045025" version="8" timestamp="2020-06-16T08:10:13Z" uid="0" user="" lat="45.8182414" lon="15.8494424"/>
  <node id="26045027" version="5" timestamp="2020-06-16T08:10:13Z" uid="0" user="" lat="45.8189227" lon="15.8490652"/>
  <node id="26045028" version="5" timestamp="2020-06-16T08:10:13Z" uid="0" user="" lat="45.819214" lon="15.8489675"/>
  <node id="26045030" version="7" timestamp="2020-06-16T08:10:13Z" uid="0" user="" lat="45.8198169" lon="15.8490276"/>
  <node id="26045032" version="6" timestamp="2020-06-16T08:10:13Z" uid="0" user="" lat="45.8202425" lon="15.8493693"/>
  <node id="26045033" version="5" timestamp="2020-06-16T08:10:13Z" uid="0" user="" lat="45.8207595" lon="15.8494114"/>
  <node id="26045034" version="4" timestamp="2020-02-03T17:20:16Z" uid="0" user="" lat="45.8209276" lon="15.8497984"/>
  <node id="26045035" version="5" timestamp="2020-06-16T08:10:13Z" uid="0" user="" lat="45.8212072" lon="15.8492944"/>
  <node id="26045036" version="4" timestamp="2020-06-16T08:10:13Z" uid="0" user="" lat="45.8215133" lon="15.8490722"/>
  <node id="26045039" version="5" timestamp="2020-06-16T08:10:13Z" uid="0" user="" lat="45.8221788" lon="15.8484156"/>
  <node id="26045043" version="2" timestamp="2017-01-20T22:18:30Z" uid="0" user="" lat="45.8235615" lon="15.8476998"/>
  <node id="26045044" version="2" timestamp="2009-11-30T14:04:48Z" uid="0" user="" lat="45.8238788" lon="15.8471471"/>
  <node id="26045045" version="4" timestamp="2017-06-22T20:22:33Z" uid="0" user="" lat="45.8232271" lon="15.8478219"/>
  </way>
  <way id="200470902" version="1" timestamp="2013-01-10T18:35:09Z" uid="0" user="">
    <nd ref="2104599032"/>
    <nd ref="2104599001"/>
    <tag k="highway" v="service"/>
    <tag k="service" v="driveway"/>
  </way>
  <way id="200470904" version="1" timestamp="2013-01-10T18:35:09Z" uid="0" user="">
    <nd ref="2104589913"/>
    <nd ref="2104589911"/>
    <tag k="highway" v="service"/>
    <tag k="service" v="driveway"/>
  </way>
  <way id="200470906" version="1" timestamp="2013-01-10T18:35:09Z" uid="0" user="">
    <nd ref="2104589925"/>
    <nd ref="2104589920"/>
    <tag k="highway" v="service"/>
    <tag k="service" v="driveway"/>
  </way>
  <way id="200470907" version="1" timestamp="2013-01-10T18:35:09Z" uid="0" user="">
    <nd ref="2104590186"/>
    <nd ref="2104590187"/>
    <nd ref="2104590188"/>
    <nd ref="2104590189"/>
    <tag k="highway" v="track"/>
  </way>
  <way id="200477343" version="2" timestamp="2017-11-24T08:16:08Z" uid="0" user="">
    <nd ref="2104609558"/>
    <nd ref="2104609559"/>
    <nd ref="2104609561"/>
    <nd ref="2104609576"/>
  </way>
</osm>
```

Slika 3.2.3: Isječak izgleda XML datoteke sa OSM podacima

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Osim OSM podataka kako bi generirali vegetaciju skup podataka koji ćemo koristiti je onaj s web stranice Zrinjevca. Na web stranici <https://gis.zrinjevac.hr/> moguće je pristupiti katastru zelenila u obliku interaktivne karte. Na istu web lokaciju je moguće poslati HTTP zahtjev sa zadanim parametrima preko čijeg odgovora možemo dobiti popis stabala za neko područje. Taj zahtjev bi izgledao ovako „[https://gis.zrinjevac.hr/stabla_geom.php?bbox=\[minBoundX\],\[minBoundY\],\[maxBoundX\],\[maxBoundY\]&srId=3857](https://gis.zrinjevac.hr/stabla_geom.php?bbox=[minBoundX],[minBoundY],[maxBoundX],[maxBoundY]&srId=3857)“. Važno je uočiti kako korištene koordinate na toj web usluzi su u sustavu EPSG3857 pa je potrebno ostvariti pretvorbu iz jednog sustava u drugi. Dobiveni HTTP odgovor (slika 3.2.4.) je potrebno najprije parsirati pa spremati kao listu objekata stabala.

```

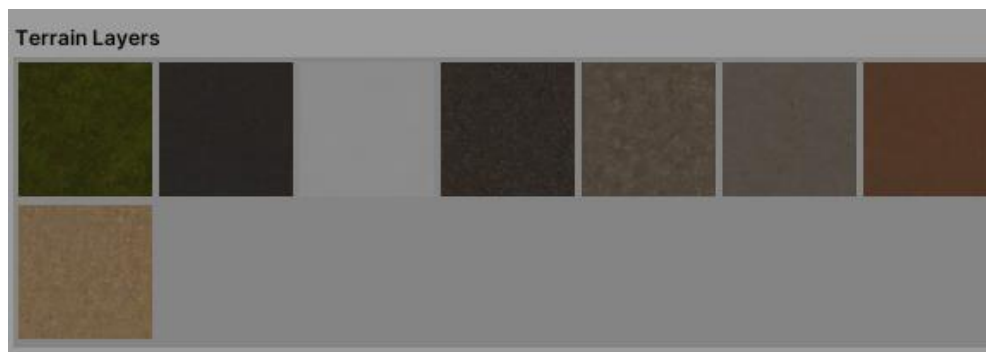
▼ features: [{"type": "Feature", geometry: {"type": "Point", coordinates: [1778477.48446792, 5749587.26467285]},...}]
▼ [0 - 99]
▼ 0: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778477.48446792, 5749587.26467285]},...}
  ► geometry: {"type": "Point", coordinates: [1778477.48446792, 5749587.26467285]}
  ► properties: {"sifra": 20509, file: "stabla.php"}
  type: "Feature"
▼ 1: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778474.83980943, 5749621.14052852]},...}
  ► geometry: {"type": "Point", coordinates: [1778474.83980943, 5749621.14052852]}
  ► properties: {"sifra": 20512, file: "stabla.php"}
  type: "Feature"
▼ 2: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778473.04154562, 5749644.9389754]},...}
  ► geometry: {"type": "Point", coordinates: [1778473.04154562, 5749644.9389754]}
  ► properties: {"sifra": 20514, file: "stabla.php"}
  type: "Feature"
▼ 3: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778472.17511722, 5749656.5452987]},...}
  ► geometry: {"type": "Point", coordinates: [1778472.17511722, 5749656.5452987]}
  ► properties: {"sifra": 20515, file: "stabla.php"}
  type: "Feature"
▼ 4: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778470.9138938, 5749674.86968749]},...}
  ► geometry: {"type": "Point", coordinates: [1778470.9138938, 5749674.86968749]}
  ► properties: {"sifra": 20531, file: "stabla.php"}
  type: "Feature"
▼ 5: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778468.80880854, 5749700.37618954]},...}
  ► geometry: {"type": "Point", coordinates: [1778468.80880854, 5749700.37618954]}
  ► properties: {"sifra": 20533, file: "stabla.php"}
  type: "Feature"
▼ 6: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778466.0308293, 5749733.91252092]},...}
▼ 7: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778467.5692716, 5749711.74308417]},...}
▼ 8: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778467.44611837, 5749722.92041769]},...}
▼ 9: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778463.91286027, 5749761.69675634]},...}
▼ 10: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778463.26829654, 5749774.41690591]},...}
▼ 11: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778397.39501567, 5749563.61373074]},...}
▼ 12: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778409.45735716, 5749564.18195371]},...}
▼ 13: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778456.08716783, 5749565.79109384]},...}
▼ 14: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778297.82229921, 5749559.29582072]},...}
▼ 15: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778281.76023388, 5749555.53317252]},...}
▼ 16: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778190.1709884, 5749555.23926061]},...}
▼ 17: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778178.72300721, 5749554.65301307]},...}
▼ 18: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778119.46229994, 5749552.15349376]},...}
▼ 19: {"type": "Feature", geometry: {"type": "Point", coordinates: [1778073.15938622, 5749550.54925612]},...}
▼ 20: {"type": "Feature", geometry: {"type": "Point", coordinates: [1777975.14912624, 5749546.407221681]},...}

```

Slika 3.2.4: Isječak HTTP odgovora sa web usluge [gis.zrinjevac](https://gis.zrinjevac.hr/)

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Stvaranje putova je implementirano kao bojanje terena pripadnom teksturom koja odgovara vrsti puta. U objekt terena koji smo instancirali ugrađene su teksture od kojih svaka odgovara pojedinoj vrsti podloge puta. Izgled ugrađenih tekstura u teren prikazan je na slici 3.2.5. Ako pak na tom dijelu terena ne postoji put tada će se taj dio obijati teksturom trave. Kao i kod postavljanja visina na teren tako i kod postavljanja tekstura postoji teksturalna mreža te možemo samo po njezinim točkama mijenjati teksturu tako da je potrebno projicirati poziciju svakog čvora na mrežu rezolucije 4096x4096. Bojanje terena odvijat će se tako da ćemo iterirati po listi objekata cesta u kojem se nalaze pojedini čvorovi s njihovim koordinatama. Između svaka dva čvora pomoću Bresenhamovog algoritma generirat ćemo međučvorove, te ćemo poziciju na terenu svakog čvora i međučvora pobožati pripadnom teksturom. Bojanje terena je ostvareno tako da svakoj od ugrađenih tekstura pridodamo jačinu utjecaja (decimalni broj od 0 do 1) za neku točku na mreži ovisno o vrsti ceste koja se nalazi na toj točki. Također se ceste razlikuju i po veličini pa ćemo tako ovisno o važnosti pobožati još par okolnih područja lijevo i desno od svakog čvora kako bi cesta dobila na širini. Također za gradske ceste čvorove koji su u sredini pobožat ćemo bijelom teksturom kako bi dobili efekt linije na cesti između dvije trake.

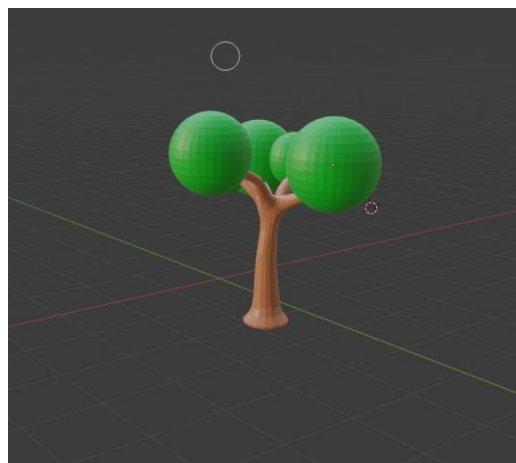


Slika 3.2.5: Izgled tekstura ugrađenih u teren

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Stvaranje građevina je izvedeno tako da se iz liste objekata koji predstavljaju građevine stvore Mesh objekti koji predstavljaju pod, strop i vanjske zidove građevine, u suštini dvodimenzionalne plohe. Trenutno posjedujemo za svaku građevinu samo listu čvorova koji tvore njen tlocrt. Za početak ćemo te čvorove pretvoriti u listu parova koordinata u prostoru koje predstavljaju vrhove poligona koji opisuje tlocrt. Zatim ćemo pozvati funkciju `Triangulator` čiji je kod dostupan na web stranici <https://wiki.unity3d.com/index.php/Triangulator> koji će za proizvoljnu listu vrhova stvoriti listu indeksa koji određuju kako će se trokuti iscrtavati kako bi dobili traženi poligon. Nakon toga uz pomoć liste vrhova i liste indeksa možemo u Unity-u stvoriti Mesh koji predstavlja pod odnosno s određenim odmakom na y-osi strop. Za stvaranje bočnih zidova iterirat ćemo se kroz isti skup vrhova koje smo koristili za pod, ali ćemo uzimati dvije po dvije uzastopne točke (tako da se jedna uvijek preklapa) i dodati još dvije iste takve samo na određenom odmaku po y-osi čime ćemo dobiti poligon koji čini jedan zid i tako dok ne iscrtamo sve zidove. Za svaku ćemo građevinu pretpostaviti da ima 3 kata odnosno da je visoka 9 metara (1 kat = 3 metra).

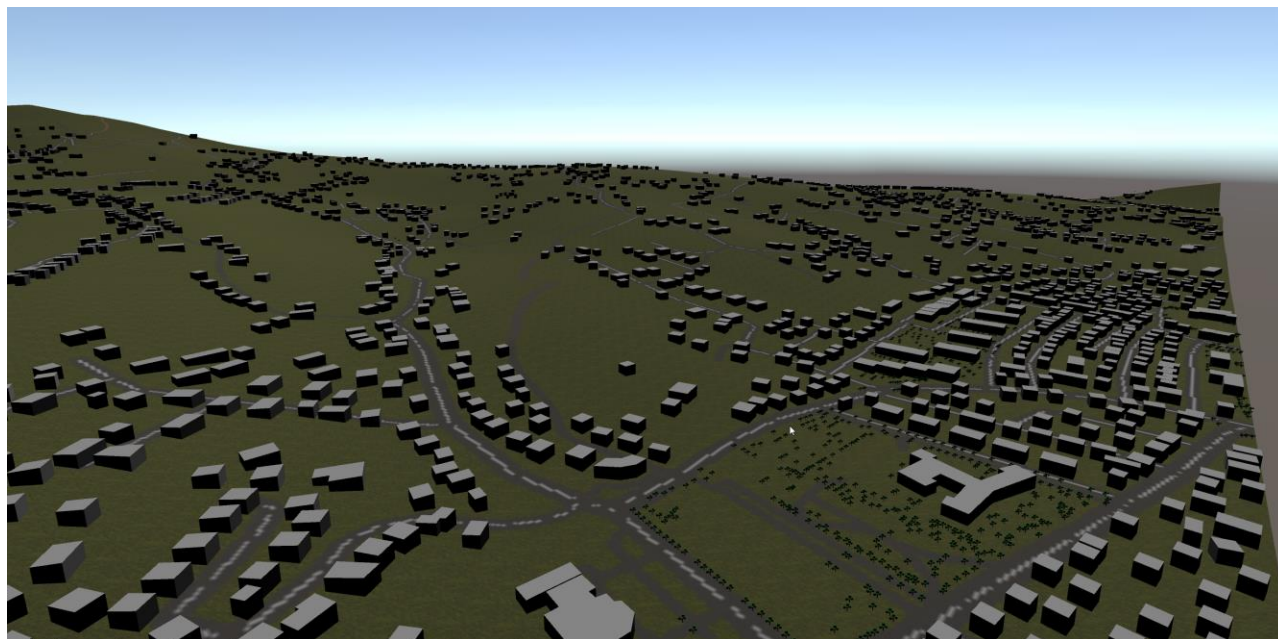
Stvaranje vegetacije odnosno stabala izvedeno je vrlo jednostavno. Iterirali smo kroz listu objekata stabala, te smo na poziciju na terenu instancirali objekt koji koristimo za reprezentaciju stabla. Objekt koji predstavlja stablo modelirali smo u programu Blender te ga uvezli u Unity. Izgled modeliranog stabla vidljiv je na slici 3.2.6.



Slika 3.2.6: 3D model stabla

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Na slici 3.2.7. moguće je vidjeti kako izgleda konačni 3D prikaz nekog područja stvoren od strane aplikacije.



Slika 3.2.7: Izgled 3D reprezentacije područja stvorenog aplikacijom

3.3 Upute za korištenje

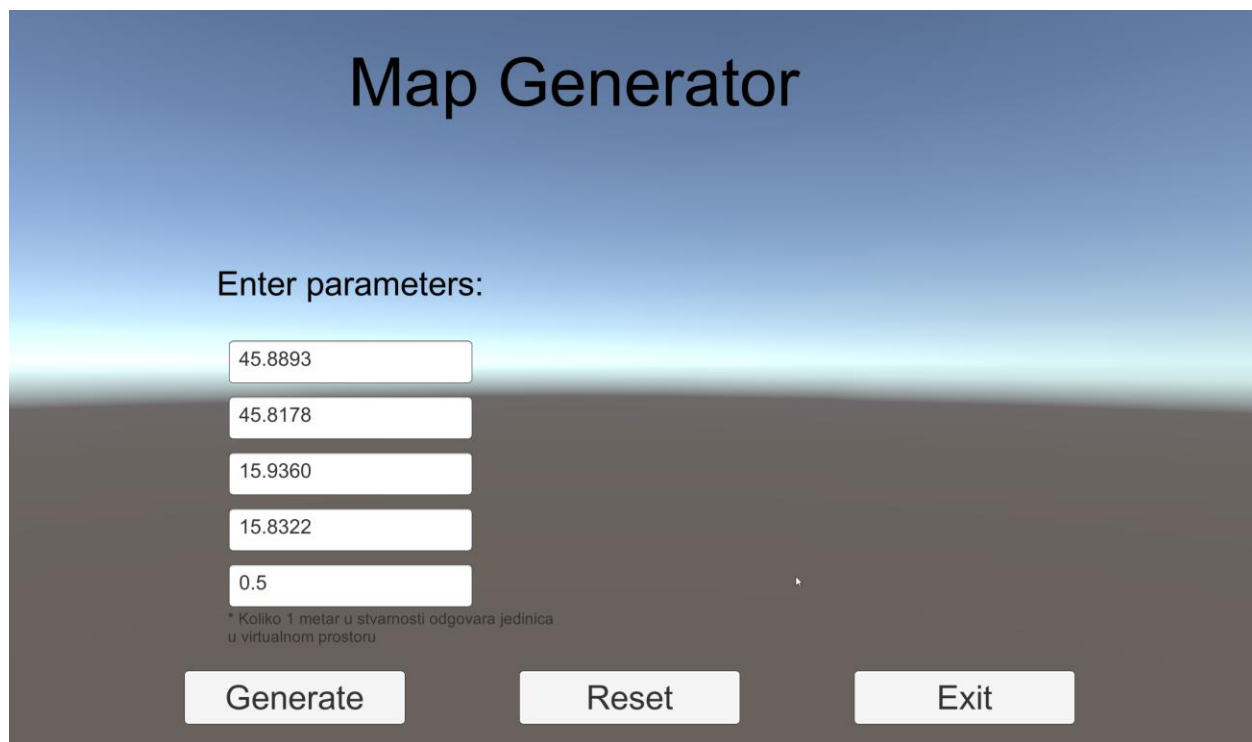
Aplikaciju je jednostavno moguće pokrenuti otvaranjem izvršne datoteke MapGenerator.exe priložene uz diplomski projekt. Priložena izvršna datoteka prilagođena je za Windows operacijski sustav, te su uz nju dostupne sve ostale datoteke potrebne za njen uspješan rad. Važno je naglasiti kako aplikacija zahtijeva povezanost na internet jer se služi HTTP protokolom kako bi dobila informacije o vegetaciji na nekom području i visini tog područja.

Pri otvaranju aplikacije dočekat će vas početni izbornik u koji je moguće unijeti parametre potrebne za izvođenje aplikacije. Prva četiri parametra redom označavaju maksimalnu geografsku širinu, minimalnu geografsku širinu, maksimalnu geografsku dužinu te minimalnu geografsku dužinu. Navedene parametre je potrebno unijeti kao decimalne brojeve (npr. 45.6178). Kako ne postoji nikakva validacija njihove ispravnosti, potrebno je paziti kako su navedeni parametri ispravno zadani inače aplikacije neće raditi. Do problema može doći i zadavanjem geografskih koordinata koje se ne nalaze

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

na teritoriju Republike Hrvatske ili u slučaju kada je zadana minimalna koordinata veća ili jednaka maksimalnoj koordinati.

Posljednji parametar koji je potrebno unijeti je faktor skaliranja terena koji određuju veličinu stvorene karte, preciznije govori o tome koliko jedan metar u stvarnosti odgovara jedinica u virtualnom prostoru (npr. ako je faktor skaliranja 0.5 tada će 1 jedinica u virtualnom prostoru predstavljati 2 metra u stvarnosti). Taj je broj također potrebno unijeti kao decimalni te se ne vrši nikakva validacija njegove ispravnosti. Izgled izbornika prikazan je na slici 3.3.1.



Map Generator

Enter parameters:

45.8893

45.8178

15.9360

15.8322

0.5

* Koliko 1 metar u stvarnosti odgovara jedinica u virtualnom prostoru

Generate Reset Exit

Slika 3.3.1: Izgled izbornika

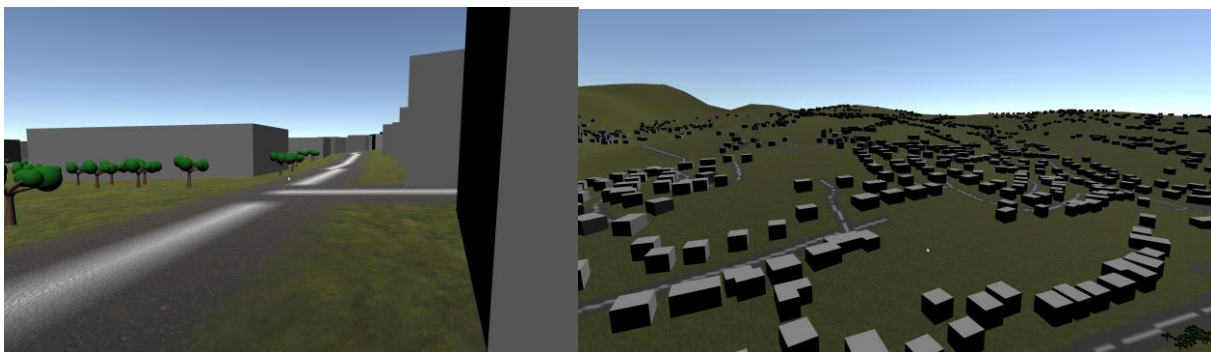
Nakon što smo unijeli sve parametre pritiskom na gumb Generate puštamo aplikaciju neka stvori mapu u zadanim koordinatama i mjerilu. Gumbom Reset možemo izbrisati trenutno stanje aplikacije odnosno spremljene objekte i podatke kako bismo pripremili aplikaciju za novi skup parametara i novu kartu. Klikom na gumb Exit izlazimo iz aplikacije.

Ovisno o veličini odabranog područja aplikaciji će biti potrebno manje odnosno više vremena da dohvati i obradi sve podatke pa je nekad potrebno malo pričekati dok se ne

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

prikaže karta.

Nakon uspješno procesiranih podataka i stvaranja karte kamera će biti pozicionirana na njezino središte te će prikaz biti iz prvog lica. Po karti se moguće kretati korištenjem tipki W, A, S, D na tipkovnici za promjenu pozicije i mišem za promjenu smjera pogleda. Kako je taj prikaz nekada neprikladan za brže kretanje po karti pritiskom na tipku C na tipkovnici moguće je promijeniti kameru kojom je moguće lebdjeti iznad mape i brže doći do nekog dijela. Ako želimo ubrzati kretanje moguće je držati tipku SHIFT za vrijeme kretanja. Isto tako ponovnim pritiskom na tipku C moguće se vratiti na kameru iz prvog lica na onom mjestu na koje smo trenutno došli. Na slici 3.3.2. prikazana su dva različita pogleda kamera, na lijevoj strani je prikaz kamere iz prvog lica, a na desnoj lebdeće kamere.



Slika 3.3.2. *Različiti pogledi kamere*

Ako pak želimo pauzirati aplikaciju to možemo napraviti pritiskom tipke TAB na tipkovnici. Tada je aplikacija privremeno pauzirana te se otvara ponovno početni izbornik. Tamo je moguće promijeniti granične koordinate i faktor skaliranja kako bi promijenili područje koje želimo prikazati. Kada smo promijenili parametre važno je prvo pritisnuti gumb Reset, a tek onda Generate kako bi se stara karta obrisala iz memorije. U svakom je trenutku moguće stisnuti tipku ESC na tipkovnici kako bi izašli iz aplikacije.

3.4 Literatura

[1] Unity User Manual, 2020., Datum pristupa: 28.12.2020.
<https://docs.unity3d.com/Manual/index.html>

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

[2] Osmosis aplikacija, 2020., Datum pristupa: 28.12.2020.
<https://wiki.openstreetmap.org/wiki/Osmosis>

[3] Google Elevation API dokumentacija, 2020., Datum pristupa: 28.12.2020.
<https://developers.google.com/maps/documentation/elevation/start>

[4] Triangulator metoda, 2020., Datum pristupa: 28.12.2020.
<https://wiki.unity3d.com/index.php/Triangulator>

[5] Blender 2.91 Reference Manual, 2020., Datum pristupa: 28.12.2020.
<https://docs.blender.org/manual/en/latest/>

[6] OSM podaci za Republiku Hrvatsku, 2020, Datum pristupa: 28.12.2020.,
<https://download.geofabrik.de/europe/croatia.html>

[7] Katastar zelenila grada Zagreba, 2020, Datum pristupa: 28.12.2020.,
<https://gis.zrinjevac.hr/>

[8] Joling A. Open Data Sources for 3D Data Visualisation - Generating 3D Worlds based on OpenStreetMaps Data, 2017., <https://www.semanticscholar.org/paper/Open-Data-Sources-for-3D-Data-Visualisation-3D-on-Joling/5844b2c8c689141c81b61b29c782a1278a808ba7>

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

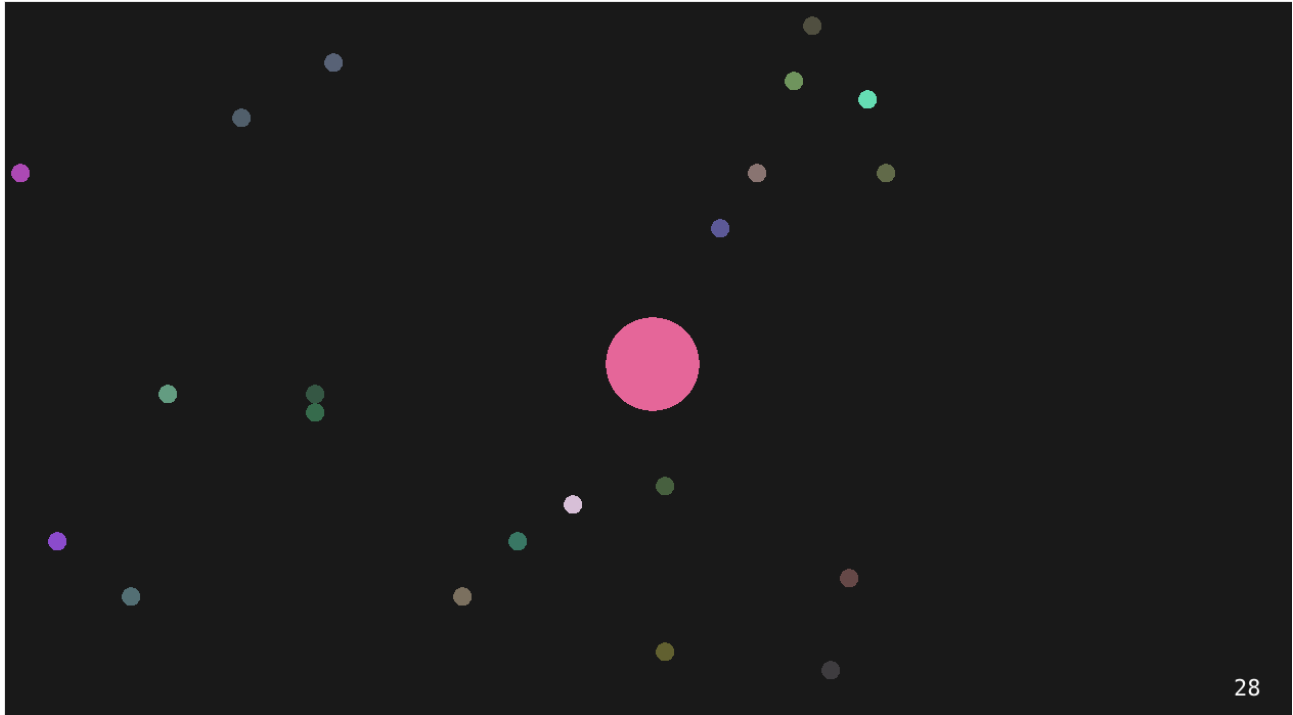
4. Implementacija računalne igre Agar korištenjem tehnologije OpenGL inačice 3.3+

4.1 Opis razvijenog proizvoda

U sklopu projekta je razvijena pojednostavljena verzija računalne igru Agar (slika 4.1.1) korištenjem modernog **OpenGL**-a i programskog jezika **C++17**. Za implementaciju sjenčara je korišten jezik GLSL, a za dodatan grafički prikaz knjižnica Dear ImGui. Kako je opseg ove igre za trajanje projekta prevelik, implementiran je samo dio igre koje će se dodatno razvijati kroz diplomski rad.

Implementacija projekta uključuje grafički prikaz cijele scene, kretanje i stvaranje objekata, detekciju kolizije objekata u sceni (pri tome treba uzeti u obzir odnos veličina objekata u sceni - nije dovoljno provjeriti dodiruju li se objekti, jer to ne rezultira uvijek istim ishodom), kontinuirano kretanje na zaslonu (linearno usporeenje u odnosu na rast objekta, inercija), kretanje kamere i grupno prikazivanje (*engl. batch rendering*) koje omogućava efikasnije korištenje grafičkog procesora.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |



Slika 4.1.1: *Prikaz scene*

U sklopu ovog projekta nije implementirana mogućnost dodavanja više igrača tako da ovaj proizvod predstavlja kvalitetnu bazu za sljedeće radove.

4.2 Tehničke značajke

Za cijeli grafički prikaz i logiku igre je implementirano niz funkcionalnosti koje omogućavaju lakše rukovanje događajima u sustavu (*engl. event system*), jednostavan prikaz elemenata (*engl. renderer*), rukovanje različitim sjenčarima (*engl. shader*) i kretanje kamere u sustavu (translacija i rotacija). Ovi elementi mogu poslužiti u razvoju bilo koje igre i predstavljaju temelj za razvoj vlastitog pokretača igara (*engl. game engine*). Iako je prikaz igre jednostavan, trenutne funkcionalnosti omogućuju laganu proširivost na kompleksnije scene.

Grafički prikaz scene je vrlo jednostavan i sastoji se od igrača i hrane koja se generiraju u određenom rasponu na zaslonu. Obje komponente su prikazane pomoću krugova koji se iscrtavaju na temelju pravila iz sjenčara fragmenata čime se postiže efikasniji prikaz nego aproksimacijom kruga pomoću niza trokuta.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

```
void main()
{
    color = distance(v_coordinate, v_center) < v_radius ? v_color : vec4(0.0);
}
```

Slika 4.2.1: Glavni dio sjenčara fragmenata

Pomak igrača

Kako bi se osiguralo kontinuirano gibanje igrača kroz scenu koje je neovisno o brzini procesora, svaki pomak se računa na temelju proteklog vremena od zadnjeg poziva funkcije za ažuriranje pozicije. Pomak se računa na temelju željene brzine igrača i pozicije miša. Igrač se pomiče u smjeru miša, a brzina ovisi i o njihovoj udaljenosti.

Nakon pomaka potrebno je dodatno provjeriti nalazi li se igrač na granicama dopuštenog raspona i onemogućiti mu daljnje kretanje u tu stranu u slučaju da se nalazi.

```
void PlayerController::on_update(Timestep timestep) noexcept
{
    constexpr float player_translation_speed = 0.01f;

    auto [x_m, y_m] = input.get_mouse_position();
    glm::vec2 mouse_position = translate_to_center({x_m, y_m});

    glm::vec2 delta = player_translation_speed * timestep * mouse_position;
    move_player(delta);
}
```

Slika 4.2.2. Ažuriranje pozicije igrača

Nakon što igrač interno ažurira svoje stanje, to je potrebno proslijediti i komponenti koja je zadužena za prikaz igrača. To je prikazano na slici 4.2.3. gdje se ažurira i veličina igrača u ovisnosti o sakupljenim bodovima.

```
void Game::update_player() noexcept
{
    constexpr static float increase_step = 0.03;
    player.radius += n_eaten * increase_step;
    player.score += n_eaten; // Update score.
    n_eaten = 0;
}
```

Slika 4.2.3: Ažuriranje bodovnog stanja igrača

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Kretanje kamere

Kretanje kamere je oblikovano na sličan način kao i pomak igrača gdje se na temelju pritisnute tipke na tipkovnici računala nova pozicija, a dodatno je omogućeno i rotiranje kamere korištenjem miša.

```
void OrthographicCameraController::on_event(Event& event) noexcept
{
    EventDispatcher dispatcher(event);
    dispatcher.dispatch<MouseScrolledEvent>(
        [this](MouseScrolledEvent& event) { return on_mouse_scroll(event); });
    dispatcher.dispatch<WindowResizeEvent>(
        [this](WindowResizeEvent& event) { return on_window_resize(event); });
}
```

Slika 4.2.4. Pridruživanje funkcija pojedinim događajima

Grupno prikazivanje (engl. batch rendering)

U početnoj inačici igre je za crtanje svakog elementa bio pozivan zaseban OpenGL poziv. Takav pristup se pokazao neefikasan. Ilustrativno, na prikazu 6400 kvadrata je s grupom veličine 10000 u odnosu na grupu veličine 1 postignuto ubrzanje od 20 puta, a to ubrzanje se samo povećava kako se povećava i broj poziva za crtanje.

Grupno prikazivanje je implementirano premještanjem dodatnih informacija iz uniform-a u spremnik vrhova (*engl.* vertex buffer). Na ovaj način se spremaju informacije o objektima u međuspremnik koji se tada prazni pri pozivu koji dolazi kada se međuspremnik napuni ili kada su pohranjeni svi objekti scene. Pri tome je potrebno grupirati samo one objekte koji koriste iste sjenčare.

Slika 6. prikazuje inicijalizaciju takvog sustava s potrebnim atributima.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

```

void Renderer::initialize_batch_renderer() noexcept
{
    // Allocate memory for batch on the CPU side.
    batch_data.quad_buffer.resize(max_vertex_count);
    batch_data.quad_buffer_it = batch_data.quad_buffer.begin();

    // Allocate memory for batch on the GPU side.
    VertexBuffer vb(nullptr,
                    max_vertex_count * sizeof(Vertex) / sizeof(float),
                    {{ShaderDataType::Float3, "position"},
                    {ShaderDataType::Float3, "center"},
                    {ShaderDataType::Float, "radius"},
                    {ShaderDataType::Float4, "color"}}});
    vertex_array.add_vertex_buffer(std::move(vb));

    // Initialize index buffer.
    std::vector<unsigned> indices(max_index_count);
    unsigned offset = 0;
    for (std::size_t i = 0; i < max_index_count; i += 6) {
        indices[i + 0] = 0 + offset;
        indices[i + 1] = 1 + offset;
        indices[i + 2] = 2 + offset;

        indices[i + 3] = 2 + offset;
        indices[i + 4] = 3 + offset;
        indices[i + 5] = 0 + offset;

        offset += 4;
    }
    IndexBuffer ib(indices.data(), max_index_count);
    vertex_array.set_index_buffer(std::move(ib));
}

```

Slika 4.2.5: Inicijalizacija grupnog prikazivanja

4.3 Upute za korištenje

Računalna igra je implementirana za operacijski sustav Linux, no u slučaju proširivanja na druge operacijske sustave, potrebne su minimalne preinake.

Nakon što se igra pokrene, s igračem se upravlja korištenjem miša. Igrač se tada kreće u smjeru miša i njegova brzina se smanjuje u ovisnosti u udaljenosti od strelice miša. Nakon što igrač prođe preko hrane, njegova se veličina povećava, a inercija se povećava. Time se i povećava ukupan broj bodova.

Kao što je na početku rada napomenuto, ovo ne predstavlja punu funkcionalnost igre, ali može se funkcionalno koristiti i u ovom stadiju. Daljnje preinake će omogućiti dodavanje više igrača i igranje preko mreže, isto kao i zvučne efekte.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

4.4 Literatura

[1] Marek Krzeminski, OpenGL Batch Rendering, 2014

<https://www.gamedev.net/tutorials/programming/graphics/opengl-batch-rendering-r3900/>

[2] Joey de Vries, Camera, 2015

<https://learnopengl.com/Getting-started/Camera>

[3] The Chernobyl, Game Engine Series, 2019-

https://www.youtube.com/watch?v=JxIZbV_XjAs&list=PLlrATfBNZ98dC-V-N3m0Go4deliWHPFwT

[4] The Chernobyl, Batch Rendering, 2020

<https://www.youtube.com/watch?v=Th4huqR77rl&list=PLlrATfBNZ98f5vZ8nJ6UengEkZUMC4fy5>

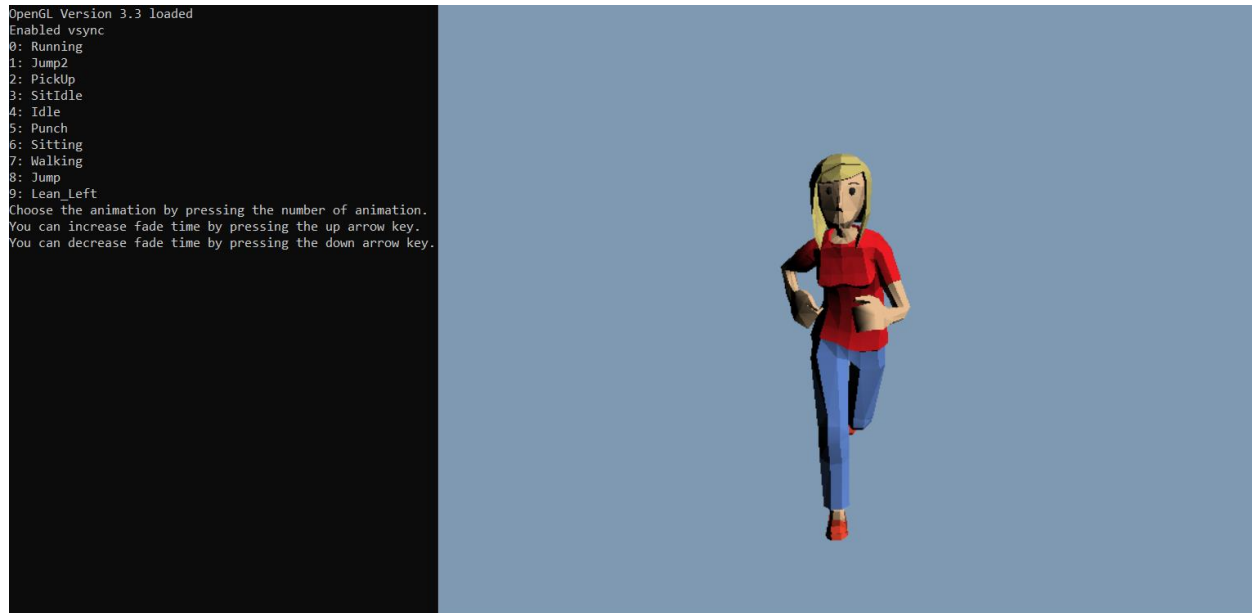
[5] Niko Savas, Nomad Game Engine: Part 7— The Event System, 2015

5. Sustav animacije u OpenGL-u

5.1 Opis razvijenog proizvoda

Kao rezultat ovog projekta izrađen je funkcionalan sustav animacije koji prikazuje model čovjeka s raznim animacijama – trčanje, hodanje, skakanje i sl. Izrađen je u OpenGL-u koristeći programski jezik C++ za operacijski sustav Windows 10. Model čovjeka i animacije se učitavaju iz *glTF* datoteke uz korištenje biblioteke *cgltf*. Za učitavanje teksture koristi se biblioteka *stb_image*. Implementiran je postupak *mesh skinning* koji se izvodi na grafičkoj kartici (engl. *Graphics processing unit, GPU*). Korisniku je omogućen odabir animacije i podešavanje vremena prijelaza iz jedne animacije u drugu koristeći tipkovnicu. Na slici 1. prikazana je aplikacija tijekom izvođenja.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |



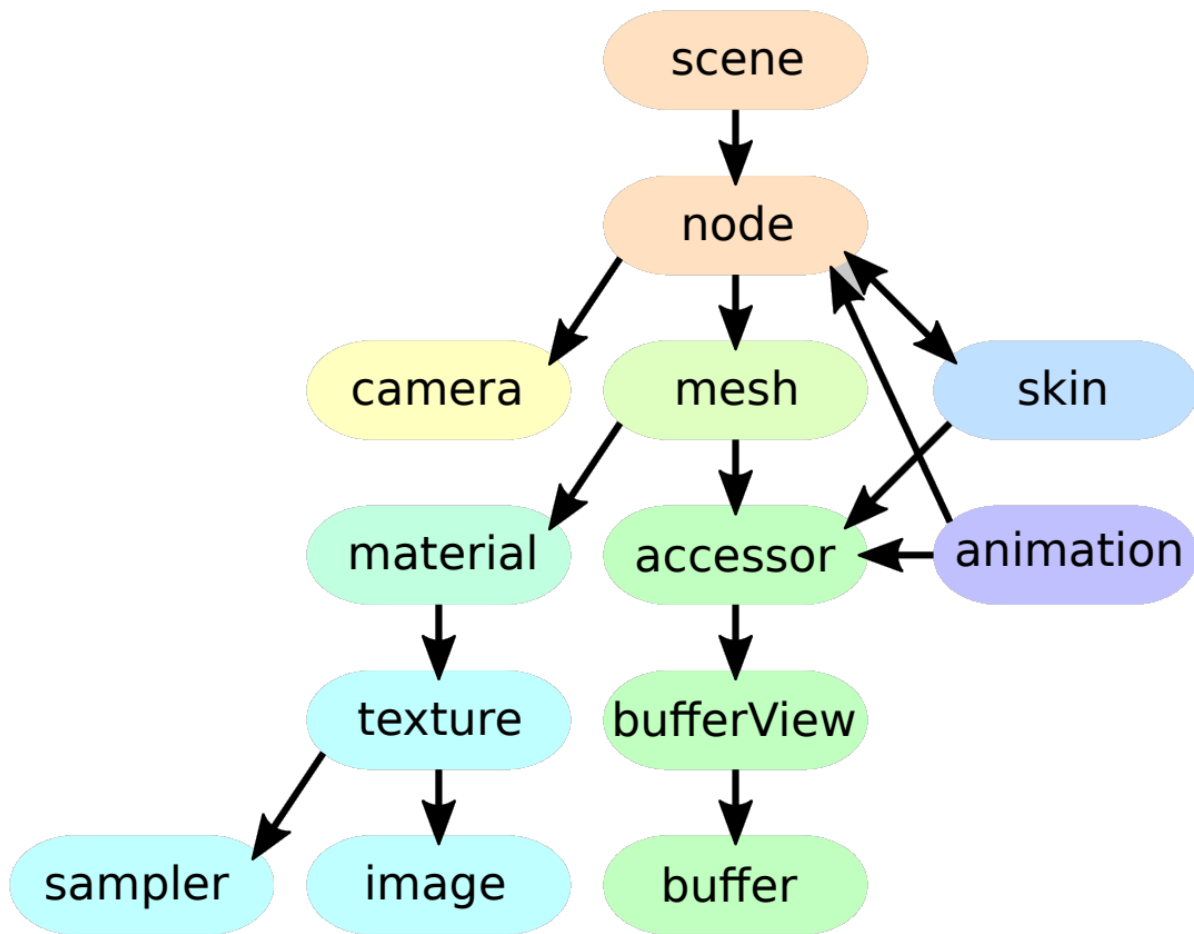
Slika 5.1.1: Prikaz aplikacije u nekom trenutku

5.2 Tehničke značajke

Projekt je izrađen u OpenGL-u koristeći programski jezik C++. Za prikaz na ekran i animiranje potrebno je izraditi pomoćne matematičke elemente – vektore, matrice i kvaternione. Implementirane su sve potrebne operacije za sve elemente. Matrice su vrlo korisne u grafičkom protočnom sustavu i olakšavaju prikazivanje objekata na ekran. Matricom se može zapisati linearna transformacija iz jednog koordinatnog sustava u drugi. Kvaternioni su proširenje kompleksnih brojeva i koriste se za zapis rotacija.

Datoteke formata glTF se mogu spremati na više načina. Prvi način je u obliku čistog teksta s ekstenzijom .gltf, a drugi u kompaktnijem obliku u binarnoj datoteci s ekstenzijom .glb. Ovaj projekt radi s datotekama s ekstenzijom .gltf. Datoteke formata glTF se koriste za spremanje cijele trodimenzionalne scene, a ne pojedinog modela. Korijen (engl. *root*) glTF datoteke je scena, a može sadržavati jednu ili više scena. Scena sadrži jedan ili više čvorova (engl. *node*). Sadržaj glTF datoteke prikazan je na slici 2. Budući da datoteka opisuje cijelu scenu, dio podataka nije potreban.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |



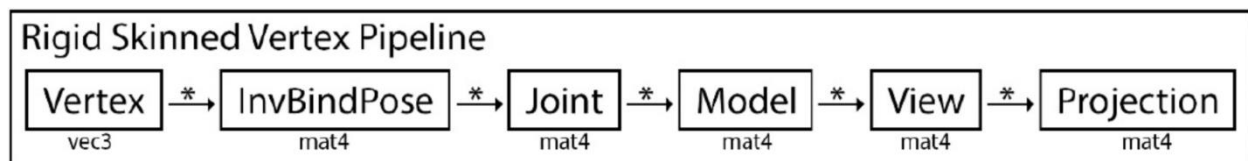
Slika 5.2.1: Sadržaj glTF datoteke

Podaci potrebni za učitavanje animiranog modela su *scene*, *nodes*, *meshes* i *skins*. Za olakšavanje učitavanja glTF datoteka koristi se biblioteka *cglTF*.

Animacijski isječak animira kolekciju zglobova (kostiju) čovjeka kroz vrijeme. Poza je kolekcija svih zglobova koji se animiraju u nekom vremenu. Svaki zglob može imati roditelja pa animacija svakog zgloba utječe na svu njegovu djecu. Postoje tri uobičajene poze svakog animiranog lika – *rest pose*, *bind pose* i *current pose*. *Rest pose* je poza lika u mirovanju, tj. zadana konfiguracija svih zglobova. Uzorkovanjem animacije u vremenu se dolazi do trenutne poze (*current pose*). *Bind pose* je zadana poza kostura modela prije nego je animiran. *Bind* i *rest* poza su često jednake.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Mesh skinning je postupak deformiranja mreže vrhova (engl. *mesh*) tako da odgovara animiranoj pozi. Može se izvoditi na procesoru pomoću C++ kôda (engl. *CPU skinning*) ili pomoću sjenčara na grafičkoj kartici (engl. *GPU skinning*). To je zapravo proces određivanja koji zglob/kost utječe na deformaciju pojedinog vrha. Na jedan vrh može utjecati više zglobova. U postupku *rigid skinning*, na svaki vrh utječe samo jedna kost. Za takav postupak, prikazan je protočni sustav kroz koji mora proći svaki vrh na slici 3. Nakon učitavanja mreže vrhova, svi vrhovi su u koordinatnom sustavu modela. Vrh se prvo množi inverznom matricom *bind* poze zgloba kojem pripada (*InvBindPose*). Zatim se množi matricom trenutne poze zgloba kojem pripada (*Joint*) da se vrati u sustav modela. Nakon toga slijedi standardno množenje s matricom modela (*Model*) koja ga stavlja u globalni sustav (sustav scene), zatim s matricom pogleda (*View*) koja ga stavlja u sustav kamere i konačno množenje s matricom projekcije (*Projection*) koja stavlja vrh u sustav projekcije za prikaz na ekranu.



Slika 5.2.2: Protočni sustav za *rigid skinning* postupak

U postupku *smooth skinning*, na svaki vrh utječe više kostiju. Takav postupak je danas standard u animacijama. U većini igara, uzima se da četiri kosti utječu na svaki vrh i takav postupak je implementiran u ovom projektu. Da bi takav postupak radio, potrebno je pamtit i koje kosti i kojom težinom utječu na svaki vrh. U strukturu vrha se zato dodaju dva vektora s četiri dimenzije – vektor koji pamti indekse kostiju koje utječu na njega i vektor težina kojom svaka od tih kosti utječe na vrh.

Prijelaz iz jedne animacije u drugu može biti izgledati neskladno, posebice u slučaju kad lik u trenu promijeni animaciju. Taj problem se rješava generiranjem prijelaznih okvira animacije koju su prosjek dviju animacija. Na taj način prijelaz animacije iz jedne u drugu izgleda skladnije i prirodnije. Taj prijelaz je uglavnom kratak – traje oko četvrtine sekunde. U projektu je zato moguće podešavati prijelazno vrijeme između animacija na vrijednost između 0 i 1 sekunde.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

5.3 Upute za korištenje

Projekt se pokreće otvaranjem izvršne (engl. *executable*) datoteke na operacijskom sustavu Windows 10. Korisnik aplikacije bira animaciju koju želi prikazati pritiskanjem broja animacije koristeći tipkovnicu:

0. Trčanje
1. Skakanje
2. Skupljanje s poda
3. Mirno sjedenje
4. Mirno stajanje
5. Udaranje
6. Sjedenje
7. Hodanje
8. Skakanje 2
9. Naginjanje ulijevo

Korisnik ima mogućnost podešavanja vremena prijelaza između jedne animacije u drugu (engl. *fade time*) koristeći tipkovnicu:

- Strelica prema dolje: smanjuje *fade time* za 0.05s
- Strelica prema gore: povećava *fade time* za 0.05s

Fade time je moguće postaviti na vrijednost iz intervala [0, 1].

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

5.4 Literatura

- [1] Szauer, G. (2020) *Hands-On C++ Game Animation Programming*. Birmingham: Packt Publishing
- [2] Khronos *glTF 2.0 Quick Reference Guide*. Dostupno na: <https://www.khronos.org/files/glTF20-reference-guide.pdf>
- [3] Quaternius Assets. Dostupno na: <http://quaternius.com/assets.html>
- [4] KhronosGroup *glTF Tutorial – Animations*. Dostupno na: https://github.com/KhronosGroup/glTF-Tutorials/blob/master/glTFTutorial/glTFTutorial_007_Animations.md

6. Generiranje prikaza virtualne scene algoritmima praćenja zrake i praćenja puta

6.1 Opis razvijenog proizvoda

U sklopu ovog projekta razvijen je sustav za generiranje prikaza virtualnih scena koji se zasniva na varijanti algoritma praćenja zrake (eng. *Ray Tracing*), algoritmu praćenja puta (eng. *Path Tracing*). Algoritam praćenja puta je u svojoj suštini stohastička inačica algoritma praćenja zrake. Stohastička priroda ovog algoritma omogućava mu prikaz fizikalne fenomenologije vezane uz širenje svjetlosti kroz prostor, poput mekih sjena, ambijentalnog zasjenjenja, globalnog difuznog osvjetljenja i slično. Ideja algoritma praćenja puta je za određeni slikovni element u scenu poslati ne samo jednu zraku, nego više njih u različitim smjerovima, te pratiti njihov put kroz scenu. Konačna boja piksela je kombinacija rezultata dobivenih od svih zraka ispaljenih za određeni slikovni element. Odnos kvalitete dobivenog prikaza i broja uzoraka po slikovnom elementu je proporcionalan, odnosno što je više uzoraka po slikovnom elementu, to je veća kvaliteta dobivenog prikaza. Implementirani sustav nastoji riješiti takozvanu jednadžbu iscrtavanja (eng. *Rendering equation*) (1) [1].

$$L(x, \vec{\omega}_0) = L_e(x, \vec{\omega}_0) + \int f_r(x, \vec{\omega}_i, \vec{\omega}_0) (\vec{\omega}_i \cdot \vec{n}) L(x, \vec{\omega}_i) d\vec{\omega}_i \quad (1)$$

Pojednostavljeno govoreći, jednadžba govori da je količina svjetla koja dolazi iz neke

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

točke u određenom smjeru ovisna o svjetlosti koju u prostor daje sama točka u slučaju emisivnih materijala, kao i o dolaznoj svjetlosti iz svih smjerova u hemisferi oko normale u toj točki. Količina svjetlosti koja dolazi iz tih smjerova je opet dana ovom jednadžbom te stoga dobivamo beskonačno rekurzivne jednadžbe koje je potrebno riješiti. Integral koji sumira svu dolaznu svjetlost u određenoj točki nema opći oblik rješenja, nego ga se pokušava riješiti nekom od numeričkih metoda. Numerička metoda integracije implementirana u ovom sustavu je Monte Carlo metoda. Opći izgled metode je dan formulom (2) [1].

$$F_N \approx \frac{1}{N} \sum_{n=0}^N \frac{f(x_n)}{p(x_n)} \quad (2)$$

Metoda računa vrijednost funkcije u nasumičnom smjeru te tu vrijednost dijeli s vjerojatnošću pojavljivanja tog smjera. Time se postiže da česti uzorci imaju manji doprinos od onih koje rijetko bilježimo. Nadalje, Monte-Carlo metoda ima dokazanu konvergenciju. Za BDRF funkcije (eng. *Bidirectional reflectance function*) odabrane su Lambertova difuzna BDRF funkcija (3) [1] koja opisuje kako materijal generira difuznu komponentu osvjetljenja i Phongova spekularna BDRF funkcija (4) [1] koja opisuje kako materijal generira spekularnu komponentu osvjetljenja.

$$f_r(x, \vec{\omega}_i, \vec{\omega}_0) = \frac{k_d}{\pi} \quad (3)$$

$$f_r(x, \vec{\omega}_i, \vec{\omega}_0) = k_s \frac{\alpha+2}{2\pi} (\vec{\omega}_r \cdot \vec{\omega}_0)^\alpha \quad (4)$$

Sustav je implementiran u sklopu pokretača za računalne igre *Unity*. Sustav podržava sljedeću prirodnu fenomenologiju:

- Tvrde i meke sjene
- Prirodno globalno difuzno osvjetljenje
- Izvore svjetlosti koji ne moraju biti točkasti
- Prirodno ambijentalno zasjenjenje
- Refleksije predmeta koji se trenutno ne nalaze u vidnom polju kamere

Važno je napomenuti da je navedenu fizikalnu fenomenologiju izrazito teško vjerno simulirati klasičnim rasterizacijskim postupcima. Sustav doduše ima određena ograničenja, od kojih su neka sljedeća:

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

- Osvjetljenje dolazi samo od teksture neba i od poligonalnih mreža koje su sačinjene od emisivnih materijala
- Sustav podržava samo neprozirne materijale
- Objekti reprezentirani mrežom poligona su podržani, no kompleksnost mreža mora biti malena, oko par stotina trokuta ukupno u sceni
- Reprezentativni rezultati se ne dobivaju u stvarnome vremenu, već je potrebno pričekati konvergenciju
- Poligonalne mreže se sjenčaju konstantnim sjenčanjem

Kako bi se koristio sustav, potrebno je imati instalirani pokretač *Unity*. Performanse sustava su direktno ovisne o grafičkom procesoru pomoću kojeg se pokreće pokretač *Unity*.

6.2 Tehničke specifikacije

Sustav za generiranje prikaza virtualne scene pomoću algoritma praćenja puta implementiran je pomoću pokretača *Unity*. Projekt je izrađen u verziji pokretača *Unity 2020.2.1f1*. Algoritam praćenja puta je implementiran pomoću sjenčara za računanje (eng. *Compute Shader*). Ovi procesori za sjenčanje su dio pokretača *Unity* i u općem slučaju služe za komplekse izračune koje je moguće paralelizirati, te je samim time lakše i brže provesti navedene izračune na grafičkom procesoru nego na procesoru opće namjene. Implementirani sustav je u grubo moguće podijeliti u 3 cjeline:

- **Ray Tracer**
 - sjenčar za računanje (eng. *Compute Shader*)
 - pomoću ovog procesora za sjenčanje je implementiran algoritam praćenja puta
- **Progressive Sampling Shader**
 - procesor za sjenčanje efekata slike (eng. *Image Effect Shader*)
 - pomoću ovog procesora za sjenčanje je implementirana logika miješanja više uzoraka kako bi se dobio konačan prikaz
- **Ray Tracer Master Node**
 - C# skripta

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

- ova skripta je odgovorna za postavljanje parametara oba ranije navedena procesora za sjenčanje, njihovo pokretanje, kreiranje i punjenje strukturiranih međuspremnik (eng. *Structured Buffer*), kao i za nasumično generiranje i slaganje sfera u sceni

Navedene tri komponente zajedno surađuju kako bi se dobio konačan prikaz virtualne scene. Pogledajmo u nastavku поближе svaku od komponenata.

6.2.1 Ray Tracer

Ray Tracer je sjenčar za računanje unutar kojeg se implementira algoritam praćenja puta. Procesor za sjenčanje ima određeni skup parametara pomoću koji je moguće kontrolirati njegov rad. Parametri koji su dio ovog procesora za sjenčanje su sljedeći:

- **RWTexture2D<float4> Result** – tekstura u koju se zapisuje rezultat rada procesora za sjenčanje
- **float 4x4 CameraToWorld** – transformacijska matrica koja transformira točke iz koordinatnog sustava kamere u globalni koordinatni sustav
- **float4x4 CameraInverseProjection** – transformacijska matrica koja transformira točke iz prostora projekcije nazad u koordinatni sustav kamere
- **Texture2D<float4> SkyBoxTexture** – tekstura neba
- **SamplerState samplerSkyBoxTexture** – služi za uzorkovanje teksture neba
- **static const float PI** – broj π
- **float2 PixelOffset** – nasumičan pomak u prostoru jednog slikovnog elementa
- **float2 Pixel** – pomoćna vrijednost za generiranje nasumičnih brojeva posebnih za svaki slikovni element
- **float RandomSeed** – parametar za generiranje pseudo-slučajnih brojeva
- **static const float EPSILON** – pomoćna vrijednost koja se koristi prilikom izračuna presjeka zrake i trokuta
- **float SkyBoxIntensity** – određuje jačinu svijetla koje dolazi s teksture neba
- **float3 YPlaneAlbedo** – parametar materijala neograničene plohe koji se koristi pri izračunu difuzne komponente osvjetljenja
- **float3 YPlaneSpecular** – parametar materijala neograničene plohe koji se koristi pri izračunu spekularne komponente osvjetljenja

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

- **float YPlaneSmoothness** – parametar materijala neograničene plohe koji određuje zrcalnost plohe
- **float3 YPlaneEmission** – parametar materijala neograničene plohe koji određuje je li materijal od kojeg je sačinjena ploha emisivan ili ne, i ako je, u kojoj mjeri za koju boju
- **StructuredBuffer<Sphere> Spheres** – strukturirani međuspremnik u kojem se nalaze podaci o sferama koje se nalaze u sceni
- **StructuredBuffer<TriangleMesh> TriangleMeshesBuffer** – strukturirani međuspremnik u kojem se nalaze podaci o poligonalnim mrežama koje se nalaze u sceni
- **StructuredBuffer<float3> VertexBuffer** – strukturirani međuspremnik u kojem se nalazi popis svih vrhova svih poligonalnih mreža u sceni
- **StructuredBuffer<int> IndexBuffer** – strukturirani međuspremnik u kojem se nalaze podaci o svim indeksima vrhova svih poligonalnih mreža koje se nalaze u sceni

Strukture koje se koriste u procesoru za sjenčanje za implementaciju algoritma su sljedeće:

- **Ray** – ova struktura predstavlja zraku koju ispaljujemo u scenu, te u sebi sadrži informacije o početnoj točki zrake, njezinom vektoru smjera, kao i o preostaloj energiji
- **Hit** – ova struktura pohranjuje informacije o presjeku zrake s nekim od objekata u sceni, te sadrži informacije o poziciji točke sudara, udaljenosti te točke, normala na površinu u točki sudara, te o parametrima materijala predmeta s kojim se zraka sudarila
- **Sphere** – struktura koja predstavlja jednu sferu koja se nalazi u sceni i sadrži informacije o samoj sferi, poput njezinog središta, radijusa, te o parametrima materijala od kojeg je napravljena sfera
- **TriangleMesh** – struktura koja predstavlja poligonalnu mrežu sastavljenu od trokuta, te pohranjuje informacije o mreži poput transformacijske matrice koja transformira vrhove iz koordinatnog sustava objekta u globalni koordinatni

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

sustav, svojstvima materijala od kojeg se mreža sastoji, broju vrhova i pomaku vrhova u globalnom međuspremniku vrhova

Funkcije koje se nalaze u procesoru za sjenčanje **Ray Tracer** i pomoću kojih se implementira algoritam praćenja puta su:

- **RandomValue**
 - funkcija generira i vraća nasumičan broj u interval $[0,1]$
- **CreateRay**
 - funkcija stvara i vraća instancu strukture Ray
- **RayHitCreation**
 - funkcija stvara i vraća instancu strukture Hit
- **TangentSpaceToWorld**
 - funkcija prima normalu i vraća transformacijsku matricu koja transformira točke iz koordinatnog sustava tangente dobivene normale u globalni koordinatni sustav
- **HemisphereSampling**
 - Funkcija prima normalu površine i realan broj
 - Funkcija vraća novi vektor smjera odbijene zrake
 - Novi smjer se računa pomoću kosinusovog uzorkovanja (eng. *Cosine Sampling*) na hemisferi oko normale
- **CreateCameraRay**
 - Funkcija prima koordinate slikovnog elementa u sustavu ekrana te na temelju njih kreira zraku za taj slikovni element
- **YPlaneIntersection**
 - Funkcija prima strukture Ray i Hit te računa sječe li se dobivena zraka s ravninom $y=0$
 - Ako presjek postoji, te je točka presjeka bliža od trenutno najbliže zabilježene u strukturi Hit, strukturu Hit postavljamo na vrijednosti vezane uz ravninu $y=0$
- **SphereIntersection**

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

- Funkcija prima instance strukture Ray, Hit i Sphere te određuje je li došlo do presjeka zrake i sfere
- Takav presjek postoji, te je točka presjeka bliža od trenutno najbliže zabilježene u strukturi Hit, strukturu Hit postavljamo na vrijednosti vezane uz sferu
- **TriangleIntersection**
 - Funkcija prima instancu strukture Ray, vrhove trokuta V_0 , V_1 , V_2 , te parametre t , u i v koje je potrebno izračunati
 - Funkcija računa sječe li zraka trokut pomoću algoritma koji su predstavili Tomas Moller i Ben Trumbore [2], te ako sjecište postoji, računa njegove baricentrične koordinate u i v , te udaljenost točke sjecišta od početka zrake t
- **TriangleMeshIntersection**
 - Funkcija prima strukture Ray, Hit i TriangleMesh te računa sječe li zraka poligonalnu mrežu, odnosno sječe li neki od trokuta od kojih se mreža sastoji
 - Ako sjecište postoji, te je trenutno najbliže, parametri strukture Hit se postavljaju na parametre ovog sjecišta i parametre materijala od kojeg je sačinjen trokut
- **TraceRay**
 - Funkcija prima strukturu Ray, te određuje najbližu točku presjeka zrake s nekim od objekata u sceni
- **ShadePoint**
 - Funkcija koja prima strukture Ray i Hit te računa boju slikovnog elementa kojoj zraka pripada
 - Ako je udaljenost zapisana u strukturi Hit beskonačnost, zraka se terminira te se uzorkuje tekstura neba
 - Ako je udaljenost pak manja od beskonačnosti, zraka se odbija i šalje dalje u scenu

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

- Prilikom toga, računa ili difuzno ili spekularno osvjetljenje točke presjeka, ovisno o vjerojatnosti koja se računa pomoću parametara materijala
- Ukupno osvjetljenje se oslanja na miješanje dobivenih konačnih uzoraka
- Odbijanje se također temelji na odabranom sjenčanju te koristi princip uzorkovanja po važnosti (eng. *Importance Sampling*), koji nastoji odabrati one smjerove kod kojih BDRF funkcija ima velike vrijednosti, odnosno, drugim riječima, ako je odabrano difuzno sjenčanje (Lambert BDRF), novi smjer se dobiva uniformnim uzorkovanjem hemisfere oko normale, dok se za spekularno sjenčanje (Phong BDRF) odabiru oni smjerovi u kojima je BDRF velik, odnosno oni smjerovi bliski smjeru koji se dobiva zakonom refleksije
- **CSMain**
 - Glavna funkcija u koju se prvo ulazi prilikom pokretanja procesora za sjenčanje
 - Funkcija kreira zraku, pušta ju scenu i prati njezin put, te rezultat zapisuje u izlaznu teksturu

6.2.2 Progressive Sampling Shader

Progressive Sampling Shader je procesor za sjenčanje efekata slike koji implementira logiku miješanja uzoraka koji se dobivaju kao izlaz sjenčara za računanje **Ray Tracer**. Naime, za jedan slikovni okvir, procesor za sjenčanje **Ray Tracer** za svaki slikovni element istražuje samo jednu od svih mogućih putanja zrake samo jednog od početnih usmjerenja. Rezultat je takav da nakon samo jednog prolaza, dobivamo izrazito zrnatu sliku. S druge strane, ako kamera i objekti u sceni miruju, sljedeći slikovni okvir će istražiti neke druge moguće puteve zraka svakog od slikovnih elemenata. Kako bi mogli kombinirati navedene slikovne okvire, koristimo miješanje po prozirnosti (eng. *Alpha Blending*). Time miješamo slikovne okvire i dobivamo sve čišću i čišću sliku što više vremena je u sceni sve statično. Prilikom pomicanja kamere ili nekog drugog objekta, miješanje kreće od početka što dovodi do toga da je ponovno prvi prikaz izrazito pun šuma.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

6.2.3 Ray Tracer Master Node

Ray Tracer Master Node je C# skripta koja je zadužena za stvaranje struktura pomoću kojih se objekti u sceni prenose na grafički procesor, za nasumično generiranje sfera, za postavljanje parametara procesora za sjenčanje **Ray Tracer**, te finalno za njegovo pokretanje i prikazivanje rezultata zapisanih u željenoj teksturi. Kako bi se moglo upravljati izgledom i rasporedom scene, skripta u prozoru *Inspector* nudi razne parametre koje je moguće mijenjati kako bi mogli dobiti različite scene. Ti parametri su:

- **Ray Tracer – Obavezan** – referenca na sjenčar za računanje koji implementira algoritam praćenja puta
- **Sky Box Texture – Obavezan** – referenca na teksturu neba koja se nalaz u sceni
- **Sky Box Intensity – Neobavezan** – određuje jačinu svjetla koje dolazi od teksture neba
- **Plane Albedo, Plane Specular, Plane Emission, Plane Smoothness – Neobavezni** – parametri materijala od kojeg je sačinjena neograničena ploha koja se nalazi u sceni
- **Draw Spheres – Neobavezan** – određuje generira li se i prikazuje li se skup sfera u sceni
- **Sphere Seed – Neobavezan** – predstavlja sjeme za pseudo-slučajne brojeve
- **Sphere Radius – Neobavezan** - interval iz kojeg se uzimaju vrijednosti za radijus sfera
- **Max Number Of Spheres – Neobavezan** - maksimalan broj sfera koje se mogu nalaziti u sceni
- **Sphere Placement Radius – Neobavezan** - radijus unutar kojega se nalaze sve sfere u sceni
- **Scene Meshes – Neobavezan** – lista svih poligonalnih mreža koje se nalaze u sceni, te je za svaku potrebno da pripadajući objekt ima komponente tipa *Mesh Renderer*, *Mesh Filter* i *Triangle Mesh Material*

Zadaće koje obavlja ova C# skripta su uglavnom grupirane u funkcije i one su sljedeće:

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

- Inicijalizacija teksture za prikaz rezultata i teksture za akumulaciju rezultata procesora za sjenčanje **Ray Tracer**
- Inicijalizacija materijala koji se koristi za miješanje rezultata u teksturi za akumulaciju pomoću procesora za sjenčanje **Progressive Sampling Shader**
- Nasumično generiranje sfera
- Izgradnja strukturiranih međuspremnik (eng. *Structured Buffer*) pomoću kojih se informacije o sferama i poligonalnim mrežama prenose na grafički procesor
- Postavljanje parametara procesora za sjenčanje **Ray Tracer** i njegovo pokretanje
- Detektiranje promjena položaja kamere i resetiranje materijala koji miješa rezultate

Skripta je dizajnirana da se uvijek postavlja na glavnu kameru scene.

6.2.4 Triangle Mesh Material

Ova skripta opisuje parametre pomoću kojih se određuje vrsta materijala od kojeg je sačinjena određena poligonalna mreža. Ova skripta je dizajnirana tako da se dodaje kao komponenta objektu koji sadrži komponente *Mesh Filter* i *Mesh Renderer*. Bez obzira na ovo, opis parametara koji je dan u nastavku primjenjiv je na sve materijale od kojih su sačinjeni objekti u sceni. Parametri pomoću kojih se definiraju materijali su sljedeći:

- **Refleksna komponenta boje materijala** (eng. *Albedo*) – određuje kako materijal reagira na difuznu komponentu osvjetljenja
- **Zrcalna komponenta boje materijala** (eng. *Specular*) – određuje kako materijal reagira na spekularnu komponentu osvjetljenja
- **Zrcalnost materijala** (eng. *Smoothness*) – predstavlja eksponent α u Phongovoj BDRF funkciji i određuje koliko je materijal reflektivan
- **Emisivnost materijala** (eng. *Emission*) – određuje je li materijal emisivan ili nije, i ako je, uolikoj mjeri u prostor pušta koju komponentu boje

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

6.3 Upute za korištenje

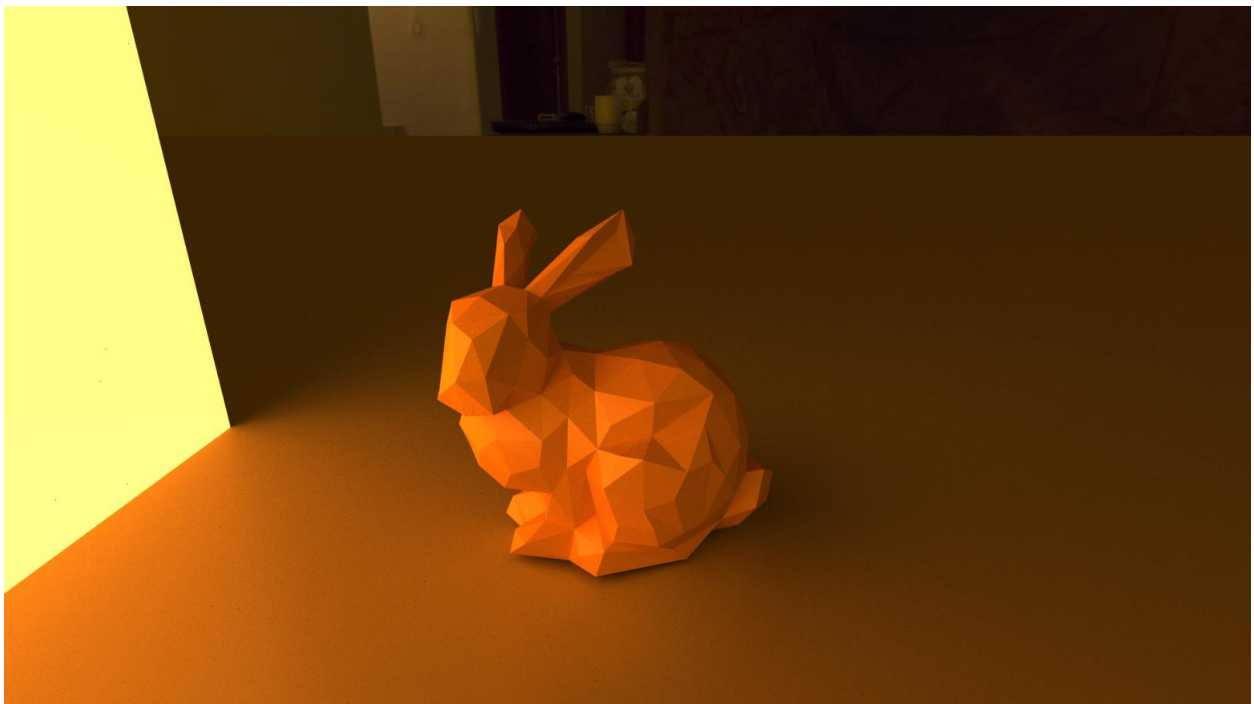
Za korištenje razvijenog sustava potrebno je na računalu imati instalirani pokretač *Unity*. Sustav je izrađen pomoću verzije *Unity 2020.2.1f1*, no bilo koja instalacija pokretača *Unity 2020* bi trebala biti dostatna. Kada se projekt pokrene pomoću navedenog pokretača, moguće je izraditi vlastitu scenu ili pokrenuti jednu od 4 unaprijed definirane scene. Unaprijed definirane scenu su namijenjene jednostavnom prikazivanju različitih fizikalnih efekata koje je sustav u mogućnosti prikazati. Unaprijed definirane scene su:

- **BunnyFireplace** – slika 6.3.1 - scena u kojoj se nalazi pojednostavljena poligonalna mreža Stanford zečića sačinjenog od primarno difuznog materijala te se pored njega nalazi ploha s emisivnim materijalom, sve to u zamračenom ambijentu koji nastoji demonstrirati difuzne efekte osvjetljenja poput mekih sjena i ambijentalnog osvjetljenja, kao i osvjetljenja koje dolazi od emisivnog materijala
- **BunnyScene** – slika 6.3.2 - scena u kojoj se nalazi pojednostavljena poligonalna mreža Stanford zečića koji se napravljen od zlata, dvije mreže koje predstavljaju travke, neograničena ploha napravljena od veoma reflektivnog materijala, koja nastoji prikazati izrazito reflektivne materijale i njihove međusobne odnose
- **SpheresAndMeshesDay** – slika 6.3.3 - scena koja se sastoji od nasumično generiranih sfera, jednog cilindra, koji je sastavljen od zelenog reflektivnog materijala, i jedne kocke, koja je sastavljena od difuznog, emisivnog materijala, te koja nastoji prikazati elemente difuznog osvjetljenja, kao i različite refleksije u dobro osvjetljenom okruženju
- **SpheresNight** – slika 6.3.4 - scena koja se sastoji od nasumično generiranih sfera, od kojih svaka ima nasumično generiran materijal, u veoma slabo osvjetljenom okruženju, te koja nastoji prikazati osvjetljenje koje pružaju emisivni materijali, te međusoban odnos raznih vrsta materijala u ovakvom okruženju

Kako bi se dobio prikaz scene, potrebno je odabrati jednu od navedenih scena iz mape *Scenes*, te pritisnuti opciju *Play* u sklopu *Unity Editor*a. Kao što je ranije spomenuto, moguće je definirati i vlastitu scenu. Kako bi definirali svoju scenu potrebno je odabrati opciju *Assets -> Create -> Scene*. U novoj sceni je moguće obrisati objekt tipa *Directional Light*. Na objekt *Main Camera* potrebno je dodati

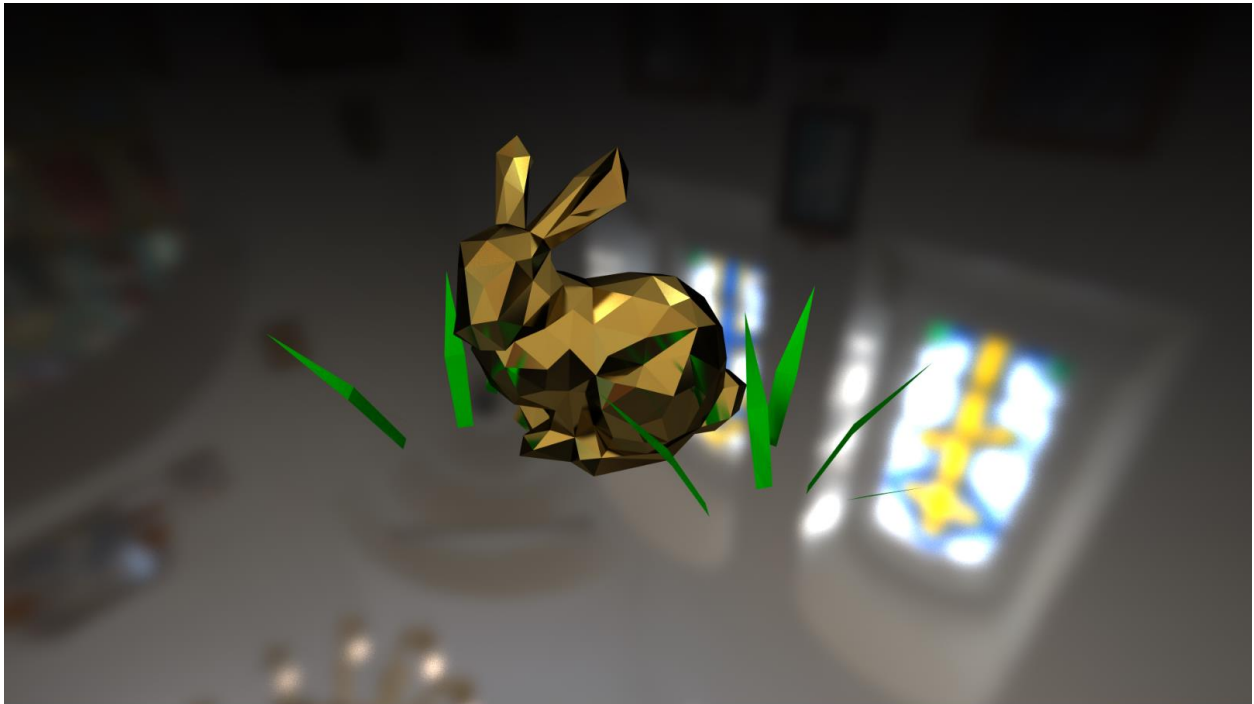
| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

komponentu **Ray Tracing Master Node**. Nakon dodavanja, potrebno je odrediti njezine parametre. Parametri **Ray Tracing Master Node** komponente su navedeni ranije u poglavlju 2. Važno je napomenuti da bez obaveznih polja nije moguće generirati prikaz scene. Kako bi dodali poligonalnu mrežu u skup objekata koje će se is crtavati, njoj pripadajući objekt mora imati sljedeće komponente: *Mesh Filter*, *Mesh Renderer* i *Triangle Mesh Material*. Pozicija ovakvih objekata u prikazu je ekvivalentna njihovoj poziciji u prikazu scene. Važno je napomenuti da je sve parametre potrebno definirati prije pokretanja opcije *Play*. Kada se pokrene opcija *Play*, moguće je pomoću prozora *Inspector* pomicati kameru u prostoru.

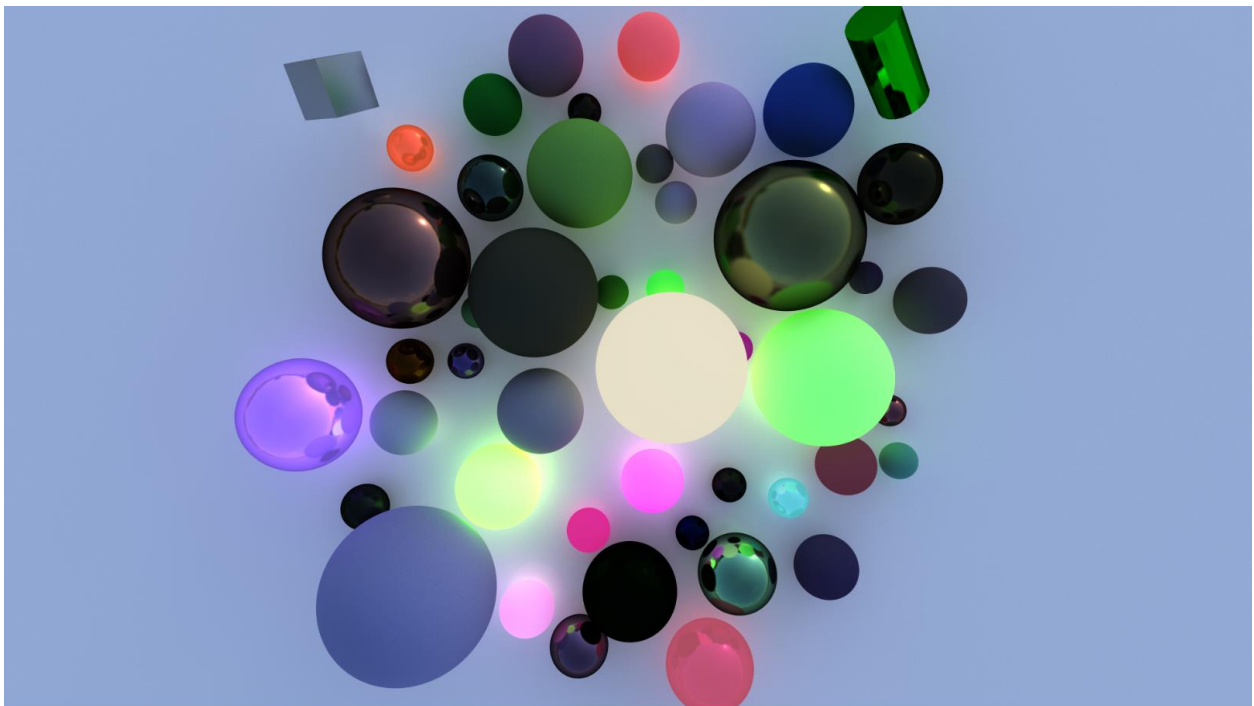


Slika 6.3.1: *Prikaz BunnyFireplace scene*

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

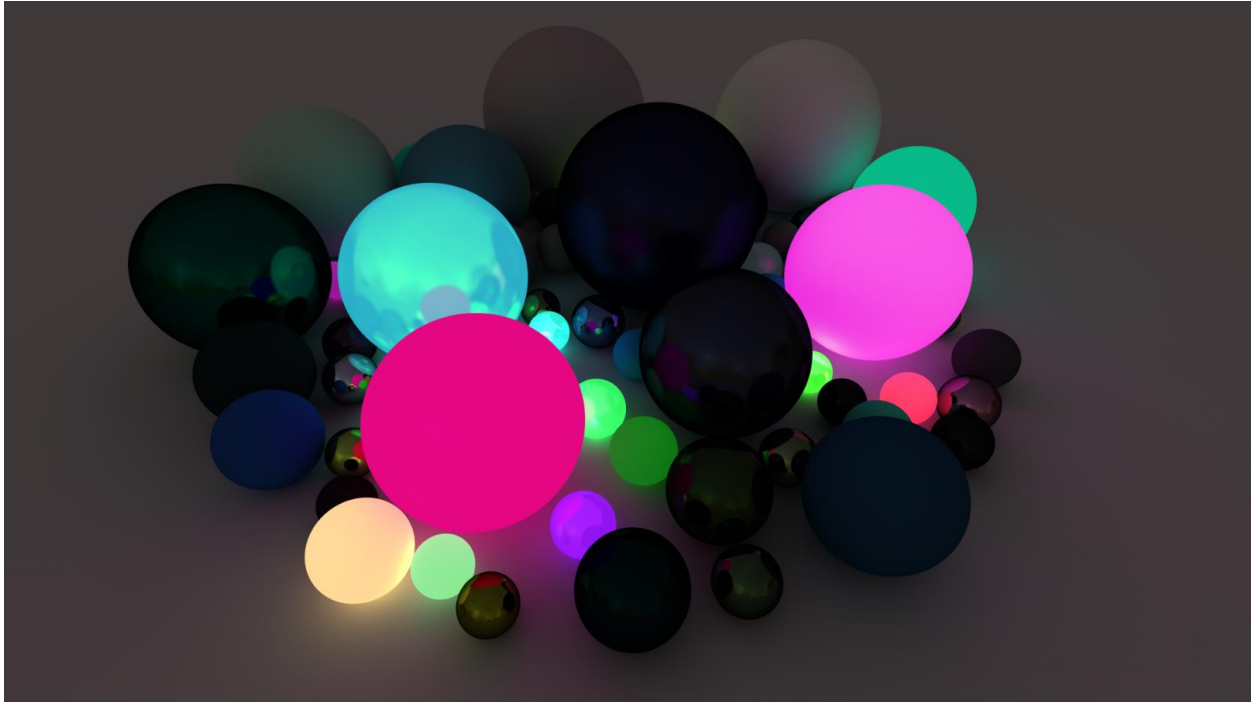


Slika 6.3.2: *Prikaz BunnyScene scene*



Slika 6.3.3: *Prikaz SpheresAndMeshesDay scene*

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |



Slika 6.3.4: *Prikaz SpheresNight scene*

6.4 Literatura

[1] David Kuri, GPU Path Tracing in Unity – Part 2, s Interneta, <http://three-eyed-games.com/2018/05/12/gpu-path-tracing-in-unity-part-2/>

[2] Thomas Möller, Ben Trumbore, Fast, Minimum Storage Ray/Triangle Intersection, s Interneta, https://fileadmin.cs.lth.se/cs/Personal/Tomas_Akenine-Moller/pubs/raytri_tam.pdf

7. Vizualizacija i simulacija ekosustava građenog genetskim algoritmom

7.1 Opis razvijenog proizvoda

Razvijen je genetski algoritam koji svakom generacijom pokušava poboljšati sposobnost vrste za opstanak.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Za spremanje gena odabrano je polje decimalnih brojeva. Transformacija gena u značajke jedinice je bijekcijska. Geni redom predstavljaju prosječni životni vijek, maksimalnu brzinu, akceleraciju, okretnost, izdržljivost, brzina oporavka nakon iscrpljujućih aktivnosti i snagu. Kako se generalno radi o parametrima koji bi u optimalnoj vrsti jednostavno bili podignuti na maksimalnu vrijednost svaki od njih ima pripadnu energetska cijenu po jedinici vremena. Ta cijena oduzima se iz energetskih zaliha jedinice. Energetskih zaliha potrebnih da bi se jedinice razmnožavale.

Kako bi uveli raznolikost među vrstama, energetska cijena svake od značajki za pojedinačne vrste blago je slučajno podešena. Željeni cilj ove promjene je nesimetričan razvoj različitih vrsta.

Zbog toga što radimo sa lokalnim, statičnim sustavom zanemarujemo mogućnost stvaranja novih vrsta te se bavimo isključivo specijalizacijom trenutnih vrsta u postojećem okolišu.

Selekcijski postupak obavlja se simulacijom ponašanja samih jedinki. Kao zadatak svaka jedinka ima prikupljanje energije te, kada uspije skupiti dovoljno energije za uzdržati samu sebe i stvaranje potomka, traženje druge jedinice iste vrste za razmnožavanje. To što je jedinka uspjela doći do situacije u kojoj stvara potomka smatra se dovoljnim dokazom da je jedinka vrijedna selekcije. Ovaj postupak mogli bi usporediti s klasičnim eliminacijskim postupkom. Razlika je u tome što u ovom slučaju ne ubijamo direktno jedinice koje su izgubile turnir ili na neki drugi način bile određene kao loše već računamo s time da će bolje jedinice svojim postojanjem onemogućiti lošijim dolazak do resursa potrebnih za prokreaciju. Iznimno loše jedinice tako će se eliminirati veoma brzo. One blago lošije od prosjeka uspjeti će stvoriti određen broj potomaka što nam omogućuje istraživanje evolucije u smjerovima van onih trenutno smatranih optimalnima.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Za križanje koristimo slučajni odabir između simulacije binarnog križanja i križanja s jednom točkom prekida. Simulacija binarnog križanja opisana je jednadžbom (1). Vjerojatnost korištenja simulacije binarnog križanja puno je veća od vjerojatnosti križanja s jednom točkom prekida. Simulacija binarnog križanja jedan je od alata za iterativni napredak populacije. Križanje s jednom točkom prekida opisano je jednadžbom (2). Ono služi za isprobavanje kombinacija pojedinih karakteristika različitih uspješnih jedinki. Jedan je od načina za bijeg iz lokalnog minimuma.

$$a = \text{random}(0, 1)$$

$$\text{chromosome}[i] = (1 - a) * \text{parent1}[i] + a * \text{parent2}[i] \quad (1)$$

$$a = \text{random}(0, \text{chromosomeCount})$$

$$\text{chromosome}[i] = i \leq a ? \text{parent1}[i] : \text{parent2}[i] \quad (2)$$

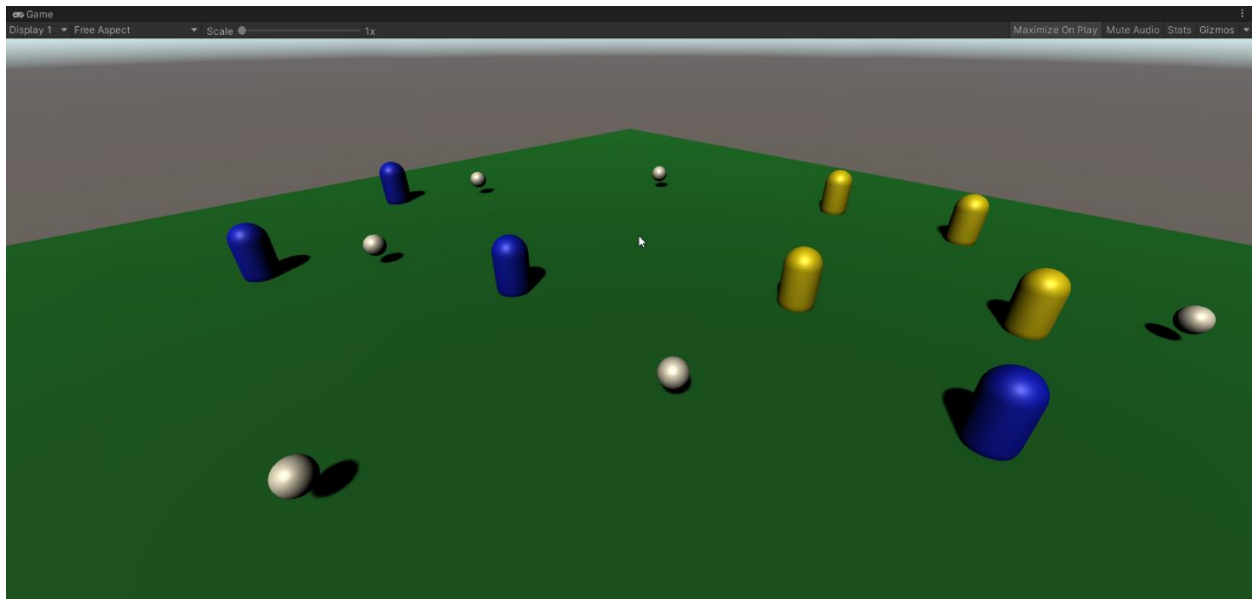
Implementirana mutacija također sadrži slučajnu komponentu. Radimo s dva tipa mutacije. U prvom (3), uz zadovoljenu vjerojatnost, gen povećavamo za $N(0, \sigma)$. U drugom (4) gen zamjenjujemo s $N(0, \sigma)$. U konačnu mutaciju ugrađujemo dvije mutacije prvog tipa i jednu mutaciju drugog tipa (M3). Jedna (M1) od mutacija prvog tipa imati će manju, a druga (M2) veću σ . U konačnici vjerojatnost odabira mutacija postavljamo tako da $p(M1) > p(M2) > p(M3)$. Taj omjer odabiremo zbog toga što u većini slučajeva želimo male iterativne pomake u dobrom smjeru (M1), ponekad želimo probati veći korak kako bi brže došli do optimuma (M2), a u iznimnim situacijama želimo ponovno pokrenuti traženje optimalne vrijednosti gena (M3).

$$\text{chromosome}[i] += N(0, \sigma) \quad (3)$$

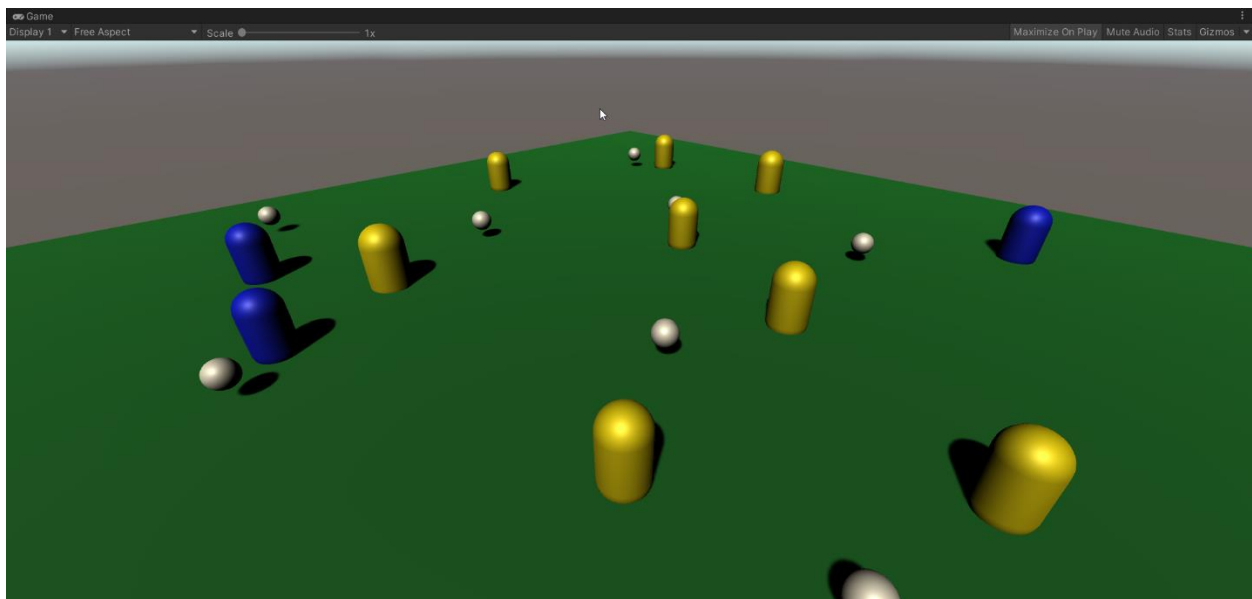
$$\text{chromosome}[i] = N(0, \sigma) \quad (4)$$

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Uz sam algoritam razvijena je i jednostavna programska podrška za vizualizaciju procesa evolucije u stvarnom vremenu. Jedinke su prikazane kapsulama. Veličina kapsule simbolizira količinu pohranjene energije. Jedinke iste vrste obojane su u iste boje. Kugle koje se pojavljuju na mapi simboliziraju hranu skupljanjem koje jedinke pune svoje zalihe energije.



Slika 7.1.1: Početak simulacije s dvije vrste



Slika 7.1.2: Simulacija nakon određenog vremena izvođenja

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

Na slici 7.1.1 možemo vidjeti snimku zaslona pri početku izvođenja simulacije. U simulaciji su dvije vrste. Plava i žuta. Po veličini možemo vidjeti kako su neke od jedinki već počele prikupljati energiju. Na slici 7.1.2 možemo vidjeti tu istu simulaciju u odmakloj fazi. Sustav je ovaj put ispao poprilično nebalansiran. Plava vrsta je dosta blizu izumiranja.

7.2 Tehničke značajke

Implementacija algoritma ostvarena je koristeći programski jezik C#. Za vizualizaciju je korišten grafički pogonu Unity. Svi korišteni algoritmi su implementirani od strane autora. Nisu korištene vanjske biblioteke van onih uključenih u C# i Unity.

7.3 Upute za korištenje

Projekt se nalazi u git repozitoriju na adresi <https://github.com/fpavletic/Svega>. Program se na operativnom sustavu Windows pokreće koristeći priloženu .exe datoteku. Pokretanje na Linux operativnom sustavu moguće je no morati ćete sami kompajlati projekt. Upute u nastavku.

Modifikacija simulacije promjenom hiperparametara moguća je korištenjem init datoteke. Datoteka je naziva Svega.init i može se naći u StreamingAssets direktoriju. Očekivani format datoteke i detalji o svakom od parametara mogu se naći u gore navedenom git repozitoriju.

Program se kompajla koristeći alat Unity. Provjereno je uspješno prevođenje korištenjem verzije 2019.4 ili novije. Nakon otvaranja projekta u Unityu držite tipke LCtrl i LShift te pritisnite B. U prozoru koji se otvorio Target Platform namjestite na željeni operativni sustav te kliknite Build. Unity će vas zatražiti da odaberete direktorij u kojem će build biti pohranjen. Kada ga odaberete Unity će izgraditi projekt i pozicionirati vas u odabran direktorij.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

7.4 Literatura

1. Genetic Algorithms, 23. kolovoza 2018., <https://www.geeksforgeeks.org/genetic-algorithms/>, 15. prosinca. 2020.
2. Marin Golub, Genetski Algoritam, 5. svibnja 2010., http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf, 15. prosinca 2020.
3. Darwin, C. On The Origin of Species, 1859. London
4. Huxley, J Evolution: The Modern Synthesis, 1942. London

8. Izračun i vizualizacija podataka dobivenih skeniranjem ultrazvučnom sondom

8.1 Opis razvijenog proizvoda

Cilj projekta je bio vizualizirati zraku ultrazvučne sonde i prikazati ju na 3D modelu. Dostupni podaci su u obliku 2D tekstura te je jedan od problema bio prikazati 2D podatke u trodimenzionalnom prostoru. Prikaz je ostvaren tehnikom marširanja zraka (*engl. ray marching*). Projekt je izrađen u pokretaču igre Unity, dio je napisan u programskom jeziku C#, dok je dio potreban za sjenčare (*engl. shader*) napisan u jeziku HLSL. Pri izradi je korištena struktura podataka za 3D teksture. Dodane su i odrezujuće (*engl. clipping*) plohe kako bi se mogli vidjeti presjeci unutar objekta.

8.2 Tehničke značajke

Prvo je potrebno pripremiti podatke, podaci su dostupni u obliku skupa slika. Korišteni su podaci dobiveni INETEC-ovom sondom. Iz podataka se zatim pomoću skripte generira 3D tekstura i sprema u odgovarajuću strukturu podataka, ustvari se 2D teksture slika spremaju poredane po z osi 3D teksture.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

U sjenčaru za iscrtavanje koristimo tehniku marširanja zraka, koja je slična tehnici bacanja zraka, kako bi smo vizualizirali 2D teksture u 3D prostoru. Korišten je Unitijev sjenčar površine (*engl. surface shader*) sa dodatnom modifikacijom izlazne boje piksela koju smo napravili dodajući liniju:

```
#pragma surface surf Lambert finalcolor:mycolor
```

Gdje je mycolor funkcija koju smo definirali.

Boju određujemo tako da krećemo od pozicije kamere i odašiljemo zraku u scenu, pomičemo se po zraci od prednje prema stražnjoj strani određenom duljinom koraka, na tom putu „skupljamo“ boju i prozirnost po formuli te u konačnici dobivamo boju piksela. Kako program ne bi bio prezahtjevan i predugo se izvodio, ograničimo maksimalan broj koraka i volumen u kojem želimo prikaz podataka.

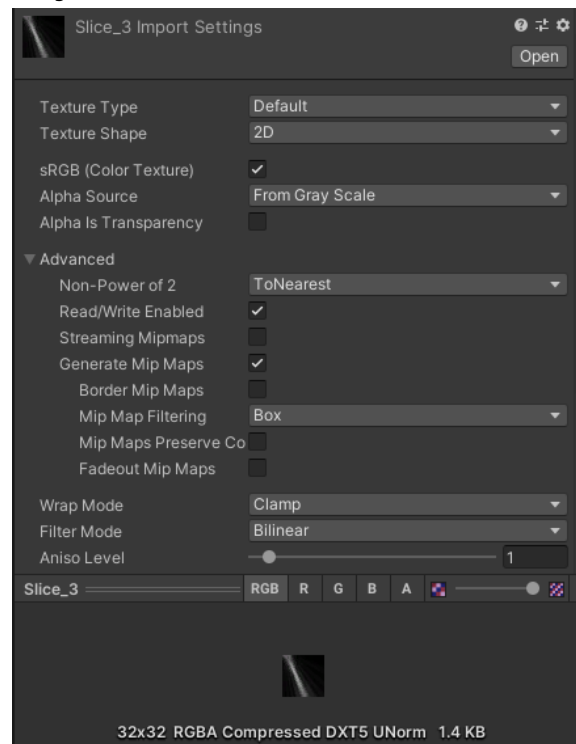
$$C^{out} = C^{in} + (1 - \alpha^{in}) \alpha C \quad , \quad \alpha^{out} = \alpha^{in} + (1 - \alpha^{in}) \alpha$$

Slika 8.2.1: Formula za boju i prozirnost ^[1]

Za ostvarenje funkcionalnosti odrezujuće plohe moramo za svaku točku provjeriti s koje strane plohe se nalazi, što računamo skalarnim produktom točke i ravnine, te odbacujemo točku ovisno o rezultatu. Ako je točka ispod ravnine (s obzirom na normalu) onda je produkt manji od 0 te možemo koristiti funkciju u sjenčaru clip(x) koja odbacuje i ne iscrta točku ako je x manje od 0. Odbacivanje točaka ostvarujemo unutar funkcije surf.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

8.3 Upute za korištenje



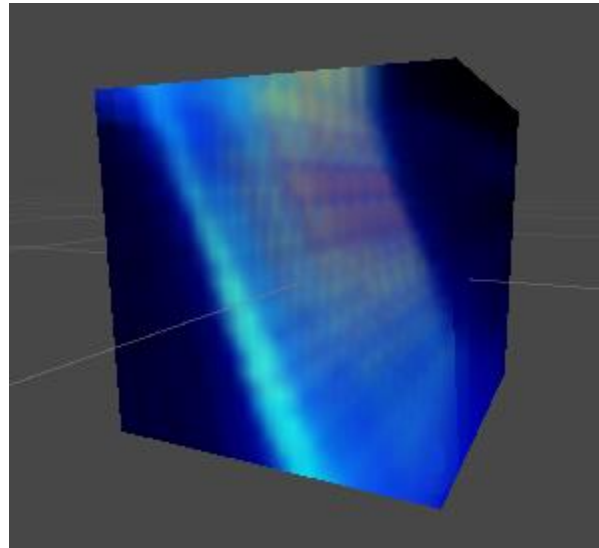
Slika 8.3.1: Prikaz potrebnih postavki u Unity editoru

Potrebno je prvo u direktorij učitati podatke u obliku skupa slika, što je više slika to će rezultati biti detaljniji. Unutar Unityja u izborniku Inspector je potrebno omogućiti opciju read/write i postaviti Wrap Mode na clamp.

Zatim se u posebno kreiranom izborniku CreateExamples odabere željena opcija za generiranje 3D teksture pomoću skripte.

Potrebno je kreirati željeni model u ishodištu postaviti materijal sa dobivenom teksturom na model. Moguće je proizvoljno pomicati položaj odrezujuće plohe kako bi se vidjeli unutrašnji presjeci objekta.

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |



Slika 8.3.2: Rezultati dobiveni korištenjem INETEC-ovih podataka



Slika 8.3.3: Rezultati dobiveni korištenjem MRI podataka sa poveznice <https://www.osirix-viewer.com/resources/dicom-image-library/>

| | |
|---|-------------------|
| Odabrane teme iz područja računalne grafike | Verzija: 1.0 |
| Tehnička dokumentacija | Datum: 20/01/2021 |

8.4 Literatura

- [1] ETH Zürich, Direct Volume Rendering
https://cgl.ethz.ch/teaching/former/scivis_07/Notes/stuff/StuttgartCourse/VIS-Modules-06-Direct_Volume_Rendering.pdf
- [2] N. D. Duong, X. Liang, and J. Tian, University of Washington: Ray Marching for VR Medical Imaging
https://courses.cs.washington.edu/courses/cse490v/20wi/public/report_09.pdf
- [3] M. Meißner, H. Pfister, R. Westermann, C.M. Wittenbrin: Volume Visualization and Volume Rendering Techniques, Eurographics 2000
<https://www.labri.fr/perso/preuter/imageSynthesis/02-03/papers/volvistut.pdf>