

Simulacija simbioze bespilotne letjelice i jedrilice
Tehnička dokumentacija
Verzija 1.0

Studentski tim: Bruno Bijelić
Nastavnik: prof. dr. sc. Željka Mihajlović

Sadržaj

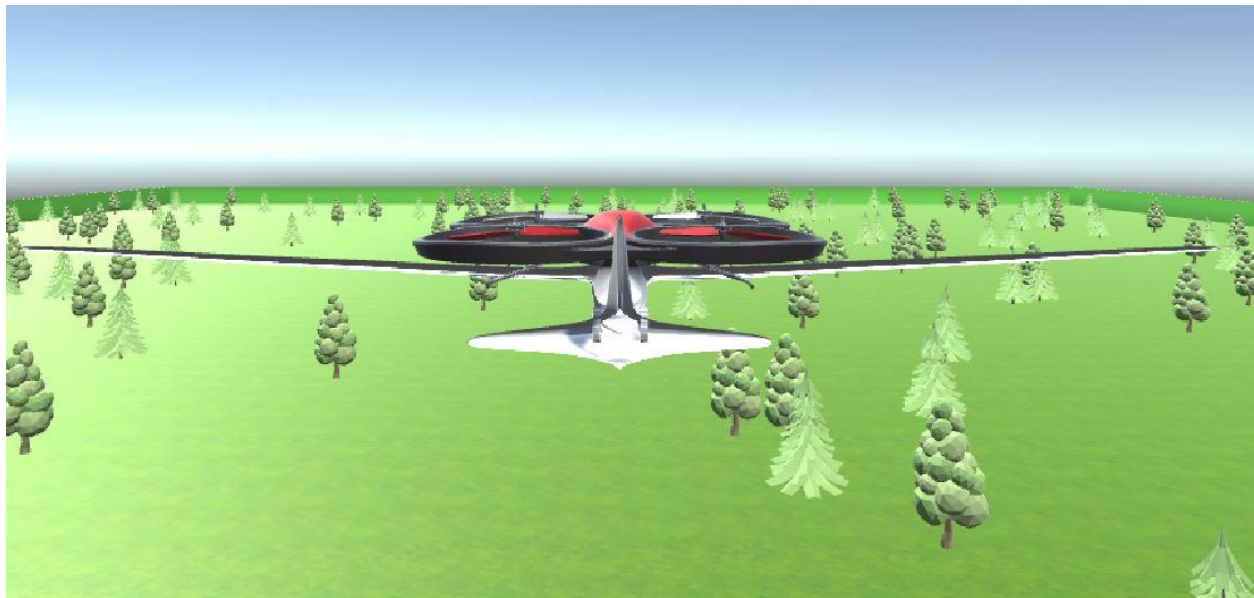
1. Opis razvijenog proizvoda	10
2. Tehničke značajke	4
3. Upute za korištenje	14
4. Literatura	16

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Tehnička dokumentacija

1. Opis razvijenog proizvoda

Cilj projekta je bio napraviti simulaciju bespilotne letjelice i jedrilice te njihovu suradnju. I bespilotnom letjelicom i jedrilicom treba biti moguće upravljati, međutim, jedrilica nema pogon kojim si može mijenjati akceleraciju, stoga je ideja dodati i suradnju između jedrilice i letjelice. Naime, letjelica treba imati mogućnost nositi jedrilicu te joj tako dati visinu i brzinu. U bilo kojem trenutku letjelica može pustiti jedrilicu i u tom trenutku jedrilica može manevrirati zrakom. Također, kada su odvojeni, letjelica može ponovno pokupiti jedrilicu u zraku ako se međusobno uspiju poravnati. Simulacije je bila rađena u Unity-u. Najvažniji dio simulacije je bio implementirati simuliranu letjelicu, jedrilicu te mehanizme pomoću kojih se oni mogu spajati i odvajati. Na slici 1 može se vidjeti završni izgled simulacije.



Slika 1. Konačni izgled simulacije

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

2. Tehničke značajke

2.1 Simulacija bespilotne letjelice

Bespilotna letjelica je simulirana pomoću Unity-evog sustava fizike. Letjelici se mijenjaju brzina i rotacija ovisno o tome što je korisnik simulacije pritisnuo. Naime, na letjelicu se može dodati relativna sila u 6 smjerova (ravno, iza, lijevo, desno, gore, dolje). Također, letjelicu je moguće rotirati oko njegove vertikalne osi. To ne mijenja akceleraciju letjelice, već samo njegovu rotaciju. Međutim, već spomenutih 6 sila mijenjaju i rotaciju i akceleraciju letjelice. Naime, ako letjelica ima vrlo malu masu, ona bi mogla ubrzavati i pritom se vrlo malo rotirati. Međutim, ako je masa letjelice nešto veća (nekoliko kilograma), puno je teže akcelerirati bez rotacije, pogotovo ako želimo dati letjelici veliku akceleraciju [1]. Pošto želimo da ova letjelica može nositi jedrilicu, pretpostavit ćemo da ova letjelica dobiva rotaciju prilikom akceleriranja. Stoga, kada akcelerira, letjelica se također malo nagne u stranu prema kojoj akcelerira. Dodatno, letjelica se kreće nešto sporije kada nosi jedrilicu sa sobom. Sve te postavke, kao što su akceleracija letjelice po smjerovima, nagnjanje letjelice prilikom akceleracije, brzina rotacije letjelice, maksimalna brzina letjelice te faktor s kojim se množi maksimalna brzina letjelice kada je jedrilica spojena na nju, mogu se vrlo lako podešavati u *Unity Editor-u* bez ikakvih promjena u kodu. Na slici 2 možemo vidjeti model letjelice.

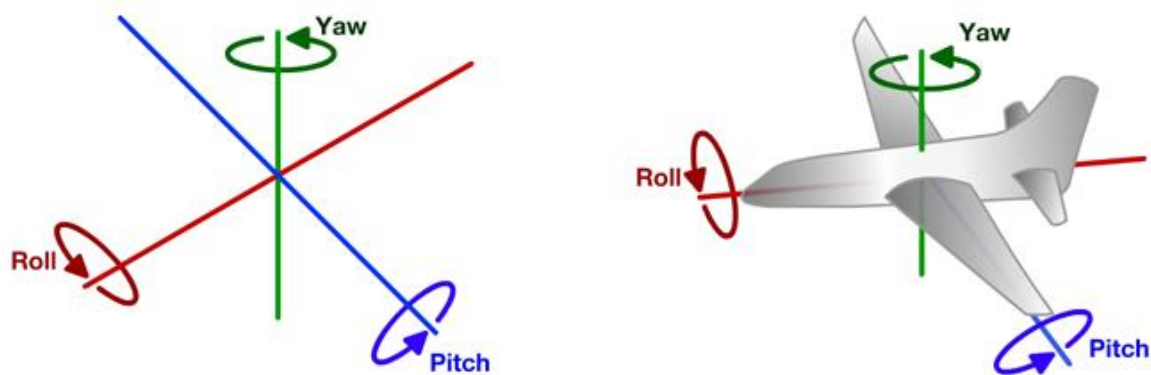


Slika 2. Model bespilotne letjelice

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

2.2 Simulacija jedrilice

Za jedrilicu je također korišten Unity-ev sustav fizike. Jedrilica je modelirana po uzoru na avion, jedino je razlika to što jedrilica nema nikakav pogon, stoga jedrilica svoju akceleraciju dobiva od dinamičkog uzgona. Također, na jedrilicu cijelo vrijeme djeluje gravitacijska sila, stoga će se jedrilica u jednom trenutku spustiti na tlo. Međutim, jedrilice i avioni imaju veliku sposobnost manevriranja u zraku. Oni mogu kontrolirati smjer svoje brzine. Također su vrlo aerodinamični, stoga se jedrilica dosta dugo može boriti protiv gravitacije, iako nema nikakav pogon. Jedrilica se upravlja sa 6 tipki (po 2 za svaki način rotiranja). 3 načina rotiranja su rotiranja oko sljedećih osi (engleski nazivi): *yaw*, *pitch*, *roll* [2]. Na slici 3 se može točno vidjeti što koja os označava. U trenutku kada letjelica pusti jedrilicu, oni imaju jednaku brzinu, stoga jedrilica ima početnu brzinu koju može preusmjeravati. Također, na jedrilicu konstantno djeluje gravitacijska sila, stoga i nju treba simulirati. Kao i kod letjelice, i za jedrilicu u *Unity Editor-u* mogu se uređivati postavke bez promjena u kodu. Neke od postavki su: gravitacijska sila, otpor zraka (stopa usporavanja jedrilice) te brzine rotiranja oko *yaw*, *pitch* i *roll* osi.



Slika 3. Prikaz *yaw*, *pitch* i *roll* osi

2.3 Spajanje i odvajanje letjelice i jedrilice

Zadnja važna stvar za implementaciju je mogućnost spajanja i odvajanja letjelice i jedrilice. Kada simulacija tek krene, oni su spojeni. Odvajanje nije previše komplicirano. Tijekom odvajanja samo treba paziti na to da u trenutku odvajanja letjelica i jedrilica moraju imati jednako brzinu i rotaciju. Također je dodana sila izbačaja u trenutku odvajanja. Naime, prilikom odvajanja letjelica i jedrilica imaju veliki rizik da se odmah sudare. Stoga je dodana sila izbačaja, to jest u trenutku odvajanja letjelica „odgurne“ jedrilicu od sebe tako da ne bi bili preblizu. Što se tiče spajanja, kao što je već rečeno, želimo da letjelica može u zraku „uloviti“ jedrilicu ako su poravnati. U trenutku kada su letjelica i jedrilica odvojeni, osoba koja upravlja letjelicom može stisnuti tipku „*space*“ koja simulaciji govori da korisnik simulacije želi da se letjelica i jedrilica spoje. Simulacija u tom trenutku provjeri poziciju i rotaciju letjelice i jedrilice te ako su one dovoljno slične, letjelica i jedrilica će se spojiti. Nakon toga, letjelica može malo nositi jedrilicu te ju opet pustiti u bilo kojem trenutku.

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

3. Upute za korištenje

Ako korisnik ima instaliran *Unity* programski paket, može u *Unity* aplikaciji otvoriti ovaj projekt te može pokretati igricu. Također, u *Unity* aplikaciji korisnik može podešavati postavke igrice kao što su maksimalna brzina bespilotne letjelice. Alternativni način za pokretanje simulacije je dvoklikom na *build-anu* verziju aplikacije. Za to nije potreban *Unity* programski paket, već je dovoljan samo dvoklik na izvršivu datoteku.

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

4. Literatura

[1] Mathematical Modelling of Multirotor UAV, International Journal of Theoretical and Applied Mechanics, (2016)

[2] The University of Melbourne, *How do airplanes fly*, (2017),

<https://blogs.unimelb.edu.au/sciencecommunication/2017/10/18/how-do-airplanes-fly-physics-behind-the-navigation-of-aircraft/>

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Proceduralno generiranje terena

Tehnička dokumentacija

Verzija 1.0

Studentski tim: Mate Buljan

Nastavnik: prof.dr.sc. Željka Mihajlović

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

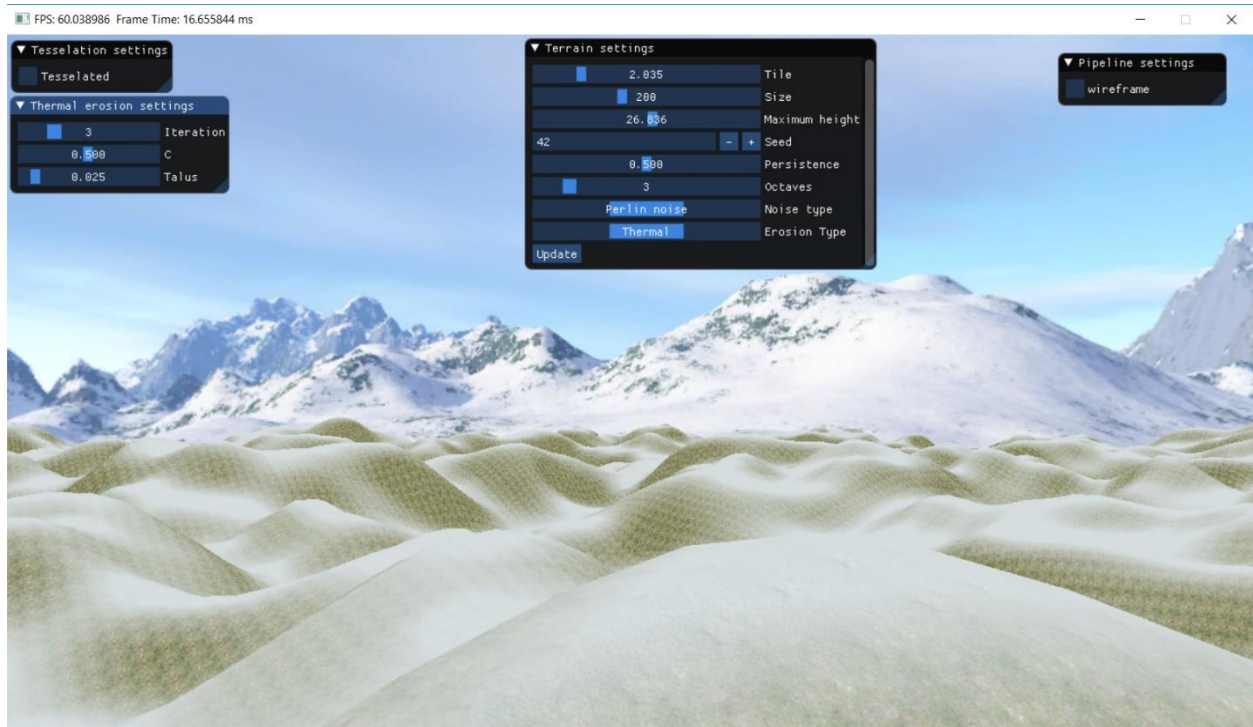
Sadržaj

1. Opis razvijenog proizvoda	10
2. Tehničke značajke	4
3. Upute za korištenje	14
4. Literatura	16

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

5. Opis razvijenog proizvoda

Kao rezultat ovog projekta je napravljena aplikacija koristeći C++/DirectX11 pomoću kojeg korisnik može generirati proceduralni teren mjenjajući razne parametre kroz korisničko sučelje (napravljeno s ImGui-jem). Parametri koje korisnik može postaviti su objašnjeni u **3. Upute za korištenje**. Slika 5-1 prikazuje aplikaciju u nekom trenutku.



Slika 5-1 Prikaz aplikacije u nekom trenutku

6. Tehničke značajke

Generiranje terena

Teren se generira prvo kao obična mreža kvadratića čija je visina jednaka 0. Zatim se iskonstruira objekt klase Heightmap koji iz parametara izgenerira vrijednosti visinu u 2D polje (vektor vektora). Parametar koji je predan objektu Heightmap pri konstrukciji je struct HEIGHTMAP_DESC:

```
struct HEIGHTMAP_DESC
{
    int size;
    float maxY;
    int seed;
```

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

```

float persistence;
int octaves;
NOISE_TYPE noise;
};

```

Te vrijednosti su dobivene ili početnim postavljanjem, ili eventualno kasnije korisnikovim mjenjanjem parametara. S obzirom da je proces računanja matrice visina trivijalno paralelan implementirana je i paralelna verzija.

NOISE_TYPE noise (enum) diktira koja se vrsta šuma koristi za generiranje visina. Tri su mogućnosti:

1. VALUE – najjednostavnija i najbrža mogućnost od moguće tri. Iako konceptualno drugačija (interpolirani bijeli šum), ljudi ju vrlo često miješaju s gradijentim šumovima poput Perlinovog šuma i Simplex šuma
2. PERLIN – najpoznatija i najkorištenija vrsta šuma, koristi se poboljšana inačica Perlinovog šuma iz 2002., <https://mrl.nyu.edu/~perlin/noise/>
3. SIMPLEX – metoda za konstruiranje n-dimenzionalne funkcije šuma slična Perlinovom šumu (oboje su gradijentni šumovi) ali s nekim prednostima poput veće brzine za više dimenzije. S obzirom da se ovdje koristi samo 2-dimenzionalni šum, to ne predstavlja neki faktor kod biranja šuma.

Zbrajanjem slojeva šuma na različitim frekvencijama i s različitim amplitudama (oktave određuju koliko puta sumiramo) koja se skalira na interval [0,1] i potom množi s konstantom maxY dobivamo vrijednost visine. Zbrajanje frekvencija maloprije opisano simulira se takozvani ružičasti ili 1/f šum.

Te vrijednosti se pridruže y vrijednostima točkama u mreži te se nakon toga računaju normale koje su potrebne za teksturiranje terena. Ovisno o nagibu terena, tj. o normali, teren se sjenča s teksturom snijega, trave ili njihovom linearnom interpolacijom.

Erozija terena

Fraktalni oblici stvarnih terena su uzrokovani erozijom. Kako bi se dobio nešto realističniji prikaz,

implementirane su dvije vrste erozije: termalna i hidraulička (Olsen, 2004).

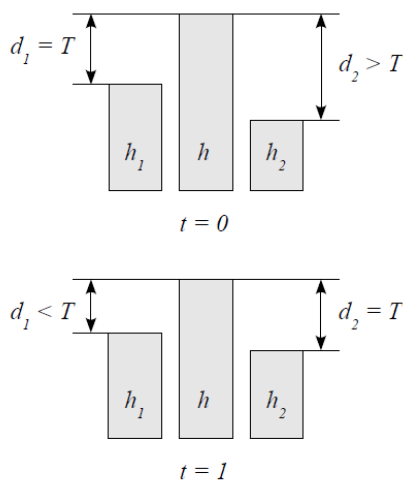
Termalna erozija simulira pucanje materijala i klizanje niz padine te nakupljanje na dnu terena.

Ovisno o visini terena, obavlja se transfer sedimenta s više točke na nižu pomoću pomoću formule:

$h_i = h_i + c(d_i - T)$ ako je $d_i > T$ inače nema promjene. Slika 6-1 ilustrira taj slučaj. T je skraćena od „talus angle“.

To je pojednostavljena verzija u slučaju da promatramo samo dva susjeda, a jednom od njih prenosimo sediment. U najčešćem slučaju promatramo ih osam. Tada je formula malo složenija.

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

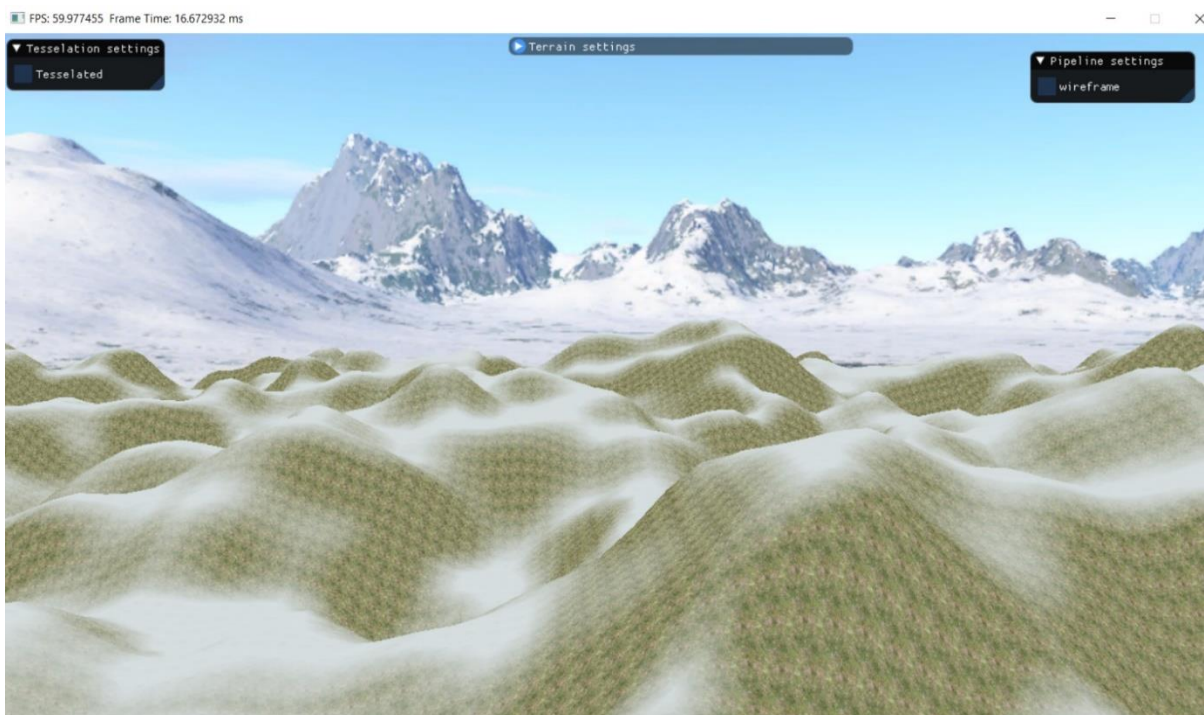


Slika 6-1 pojednostavljeni primjer za koncept termalne erozije (Olsen,2004.)

Hidraulička erozija simulira promjene terena uzrokovane tokom vode koja topi materijal, prenosi taj rastopljeni materijal s jednog mjesta i ostavlja ga na drugom mjestu. Algoritmi se mogu naći u [1] i [3].

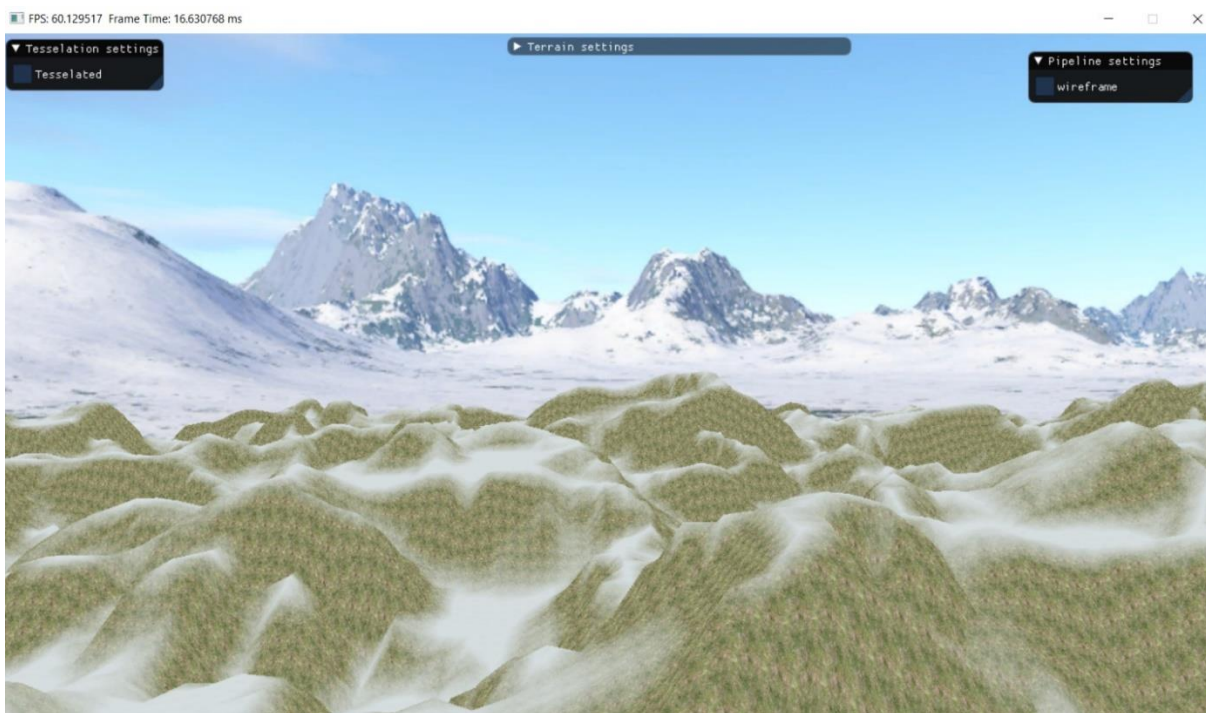
Slika 6-2 prikazuje jedan proceduralni teren bez utjecaja erozije, dok slike Slika 6-3 i

Slika 6-4 prikazuju utjecaj redom hidrauličke i termalne erozije na taj isti teren.

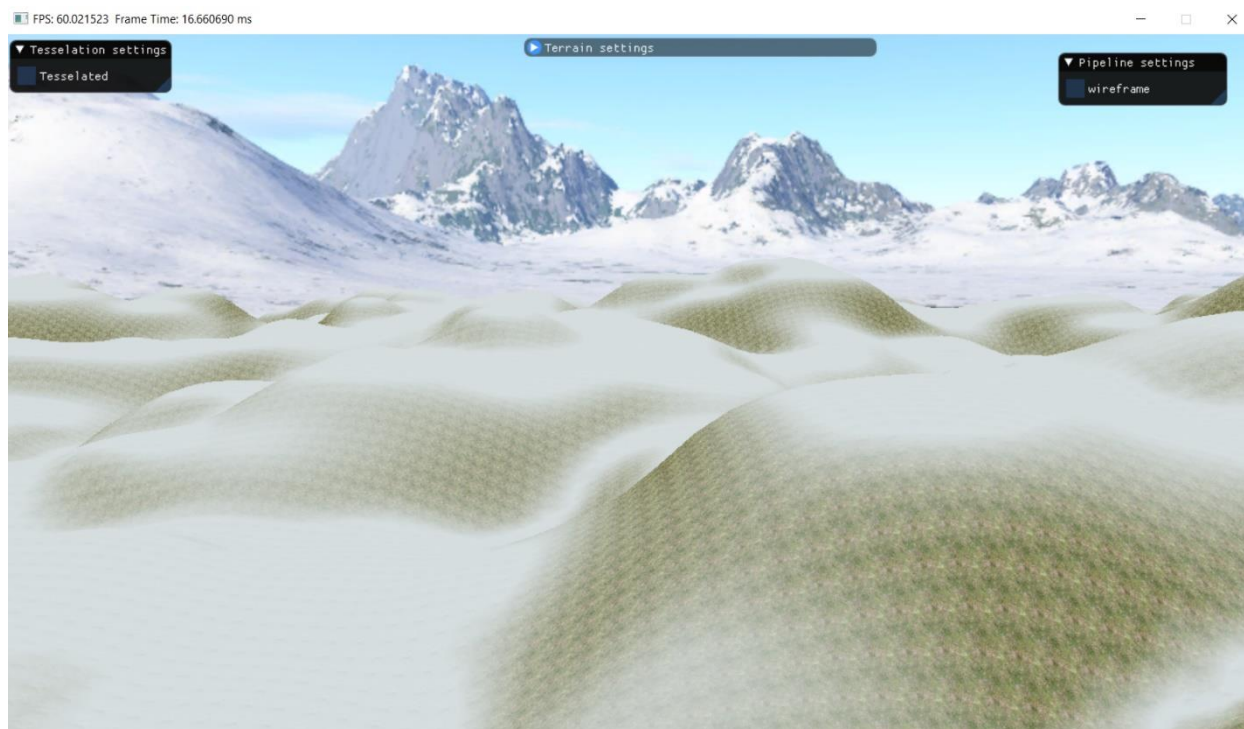


<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Slika 6-2 Generirani teren bez utjecaja erozije



Slika 6-3 Utjecaj hidrauličke erozije na teren



Slika 6-4 Utjecaj termalne erozije na teren

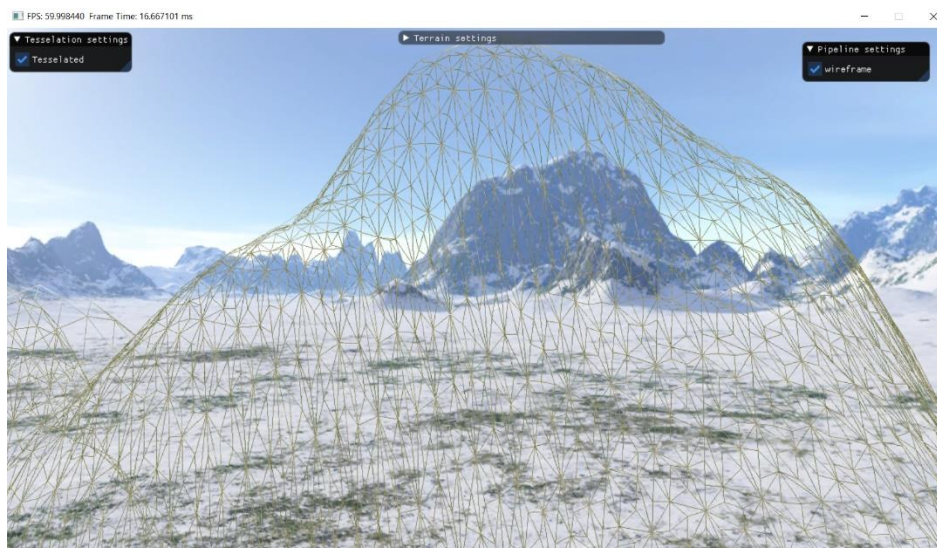
<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Adaptivna teselacija

Implementirana je adaptivna teselacija koristeći teselatorske sjenčare u DirectX11 – Hull Shader i Domain Shader. (U OpenGL-u se nazivaju Tessellation Control i Tessellation Evaluation Shader). Teselacija podrazumijeva generiranje dodatne geometrije povrh početne geometrije (one koja je poslana na GPU). U ovom kontekstu adaptivna označava da se teselacija prilagođava udaljenosti kamere od terena – što je kamera bliža, teselacija je veća. Iako nije nužno da se mora poslati vrhovi geometrije nego to relativno često budu i kontrolne točke kao za npr. Bezierove površine, u ovom slučaju Hull Shader propušta početnu geometriju te računa faktore teselacije koji će odrediti koeficijente pomoću kojih se, u Domain Shaderu računa novonastala geometrija. Na ovaj način je kreiran LOD sustav za teren.



Slika 6-5 Udaljenost između kamere i terena je dovoljno velika da ne dođe do teselacije



Slika 6-6 Teselacija trokuta zbog približavanja kamere

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

7. Upute za korištenje

Korisnik kontrolira kameru sa:

W – kretanje unaprijed

A – kretanje ulijevo

S – kretanje unazad

D – kretanje udesno

Q – Zoom in

E – Zoom out

Pomicanjem miša korisnik rotira kameru oko x i y osi (pitch and yaw). Kako bi korisnik izbjegao pomicanje kamere dok s mišem postavlja parametre, tada može držati tipku SPACE. Za to vrijeme rotiranje kamere je zaustavljeno.

Sva ostala interakcija se obavlja preko GUI-ja napravljenog pomoću lagane knjižnice ImGui.

Postavke za teren:

- Tile : Veličina jednog kvadratića mreže.
- Size: Broj kvadratića mreže u jednoj dimenziji, odnosno teren se prostire po mreži od Size x Size kvadratića. S obzirom da je teren uvijek centriran oko nule, tada su i x i z koordinate terena iz intervala $[-Size * Tile/2, Size * Tile/2]$.
- Maximum height: vrijednost s kojom se skalira vrijednost dobivenog šuma za tu točku
- Seed: vrijednost koju koristi samo Value Noise za generiranje terena
- Persistence: vrijednost koja određuje koliko će više oktave utjecati na konačnu visinu (ovisnost je proporcionalna)
- Octaves: Broj oktava koje koristi šum (obično nakon 6 ili 7 oktave ne utječu zamjetno za visinu terena osim ako se ne postavi relativno visok persistence).
- Noise Type: tri tipa šuma između kojih korisnik može birati su Value Noise, Perlin Noise i Simplex Noise.
- Erosion Type: tip erozije koju korisnik želi primijeniti na teren. Moguće opcije su bez erozije, termalna erozija i hidraulička erozija. Ovisno što korisnik odabere, stvorit će se prozor za postavljanje parametara za odabran tip erozije.
- Nakon što korisnik postavi parametre po želji, klikom na tipku Update se iscrtava novi teren generiran pomoću postavljenih parametara.

Dodatne dvije opcije su: Adaptivna teselacija terena i prikaz terena u žičanom obliku.

Adaptivna teselacija podrazumijeva generiranje više geometrije (teselacijom) od početne geometrije u slučaju da se korisnik, točnije kamera, približi geometriji terena. Kako bi se teselacija mogla bolje promatrati, dodana je mogućnost žičanog prikaza geometrije terena. Ove dvije opcije djeluju trenutno, tj. nije potrebno pritisnuti tipku Update.

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

8. Literatura

- [1] Olsen, J. (2004). *Realtime Procedural Terrain Generation*. Department of Mathematics and Computer Science, IMADA, University of Southern Denmark.,
- [2] Mei, Decaudin, Hu (2007) *Fast Hydraulic Erosion Simulation and Visualization on GPU*, 15th Pacific Conference on Computer Graphics and Applications
- [3] <https://softologyblog.wordpress.com/2016/12/09/eroding-fractal-terrains-with-virtual-raindrops/>
- [4] Luna, F. (2012), *Introduction to 3D GAME PROGRAMMING WITH DIRECTX® 11*, Mercury
- [5] Zink J., Pettineo M., Hoxley J. (2011), *Practical Rendering and Computation with Direct3D 11*, CRC Press

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

AR Potraga za blagom Tehnička dokumentacija Verzija 1.0

Studentski tim: Dino Ehman

Nastavnik: prof. dr. sc. Željka Mihajlović

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Sadržaj

1. Opis razvijenog proizvoda	19
2. Tehničke značajke i izrada	19
2.1 Izrada	19
3. Upute za korištenje	28
4. Literatura	30

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Tehnička dokumentacija

9. Opis razvijenog proizvoda

Projekt AR Potraga za blagom će istražiti korištenje tehnologije proširene stvarnosti (eng. Augmented Reality, AR) u klasičnoj igri potrage za blagom. Na razini grada će se omogućiti dodavanje lokacija, zadataka, tragova i ostalih stvari potrebnih kako bi igrači uspješno došli do cilja/pronašli blago. Prednost proširene stvarnosti je u tome što će neovisno o ostalim igračima, vremenskim uvjetima i ostalim raznim promjenama svi igrači moći doći do istih tragova i zadataka. Projekt će biti napravljen u game engine-u Unity te razvijen za sustav Android.

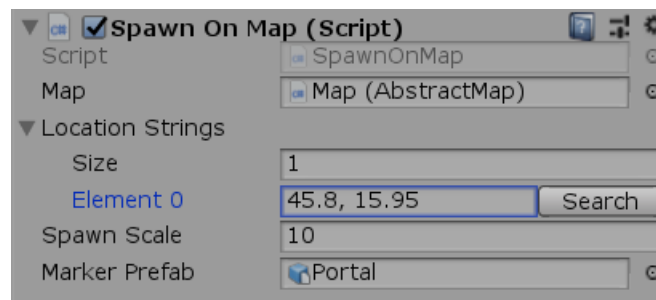
10. Tehničke značajke i izrada

Koristio sam game engine Unity 2019 za razvoj svoje aplikacije proširene stvarnosti. Koristio sam platformu Mapbox koja je open source platforma za dizajniranje mapa te koristeći Mapbox API i Mapbox SDK kao građevne blokove integrirao sam lokaciju u stvarnom svijetu unutar mobilne aplikacije. Mapa je jedan aspekt projekta, drugi aspekt je proširena stvarnost. Koristio sam Google ARCore SDK i Vuforia Engine razvojne alate koji omogućuju ugrađivanje funkcionalnosti proširene stvarnosti u aplikaciju. Potrebno je instalirati aplikaciju Google Play Services for AR prije korištenja aplikacije.

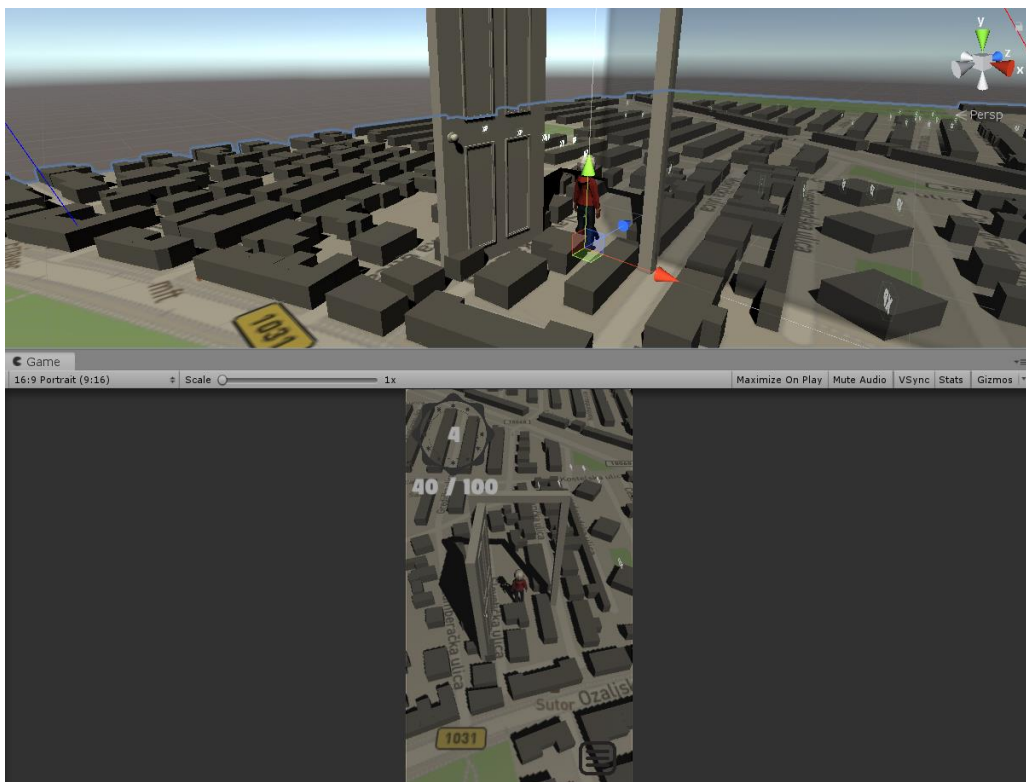
10.1 Izrada

Prvi korak je bio instalirati Mapbox SDK za Unity te ga uvesti u naš Unity projekt [3][5]. Jedna od prednosti korištenja Mapbox platforme je uređivanje vlastitog izgleda mape što možemo vidjeti na slici 1. [4].

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>



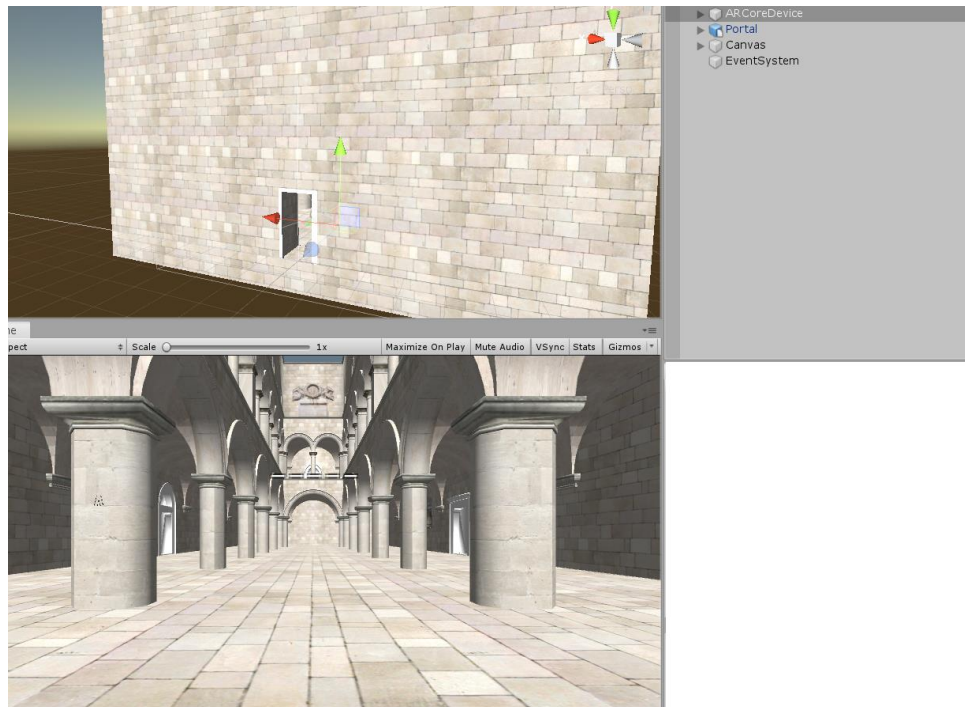
Slika 7. Skripta *SpawnOnMap* omogućava dodavanje virtualnih objekata na mapu



Slika 8. Prikaz vrata portala na lokaciji (45.8 N, 15.95 E) – Zagreb

Što se tiče dijela proširene stvarnosti, za početak je potrebno preuzeti Google ARCore SDK za Unity te ga uvesti u Unity projekt [2]. U scenu je potrebno dodati elemente ARCoreDevice koji predstavlja kameru te naš model koji želimo prikazati pomoću proširene stvarnosti (Slika 5).

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>



Slika 9. Prikaz kamere i modela u sceni

Problem s kojim se susrećemo je taj što će kamera na mobitelu prikazivati cijeli object izvana što ne želimo jer cilj je da postignemo efekt portal [7]. Ovaj problem možemo riješiti na sljedeći način. Stvorimo skriptu PortalManager koja će ovisno o tome je li kamera unutar zgrade ili ne omogućiti stencil test ili onemogućiti ga.

```

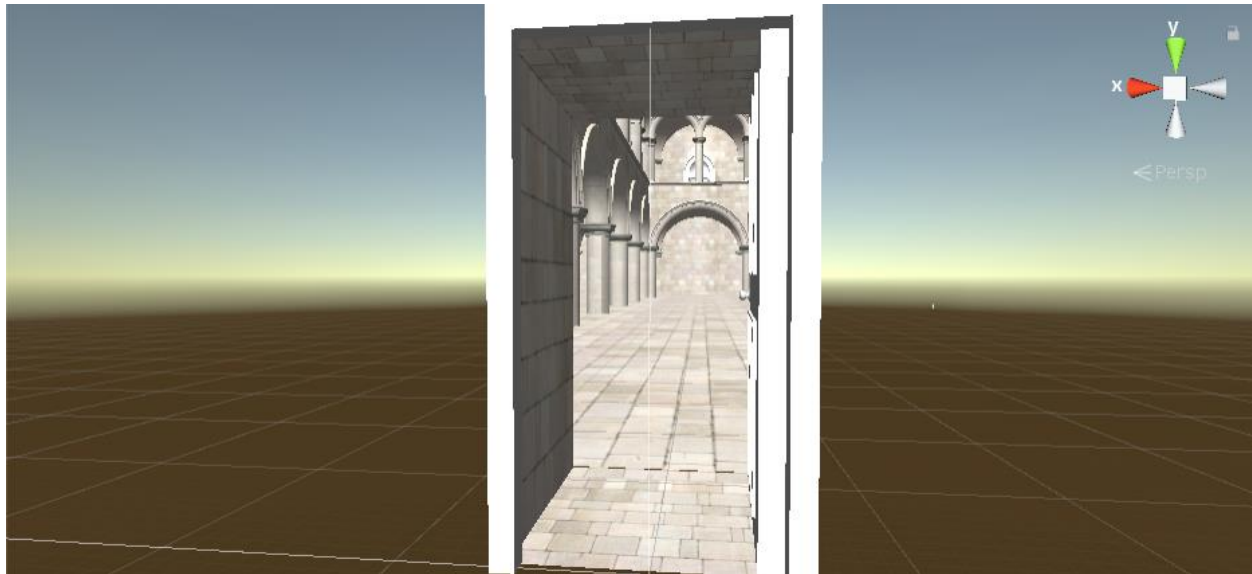
void OnTriggerStay(Collider collider)
{
    Vector3 camPositionInPortalSpace =
transform.InverseTransformPoint(MainCamera.transform.position);

    if(camPositionInPortalSpace.y < 0.5f)
    {
        for (int i = 0; i < SponzaMaterials.Length; i++)
        {
            SponzaMaterials[i].SetInt("_StencilComp", (int)CompareFunction.Always);
        }
    }
    else
    {
        for (int i = 0; i < SponzaMaterials.Length; i++)
        {
            SponzaMaterials[i].SetInt("_StencilComp", (int)CompareFunction.Equal);
        }
    }
}

```

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Nakon dodane skripte, portal se ponaša na odgovarajući način (Slika 6 i 7).



Slika 10. Objekt kada je kamera izvan portala



Slika 11. Objekt kada je kamera unutar portala

Unutar objekta sam zatim rasporedio 5 slova koje igrač treba pronaći kako bi mogao nastaviti dalje potragu (Slika 8).

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>



Slika 12. Slova unutar objekta koje igrač treba skupiti

Sljedeći korak zagonetke je bilo složiti anagram od slova koje je igrač skupio unutar portala. Unity elementi grafičkog sučelja (Canvas, Image, Button) se koriste za inicijaliziranje gumbova koji predstavljaju slova [6]. Skripta WordScramble omogućava da nakon 2 klika se zamijeni poredak slova te nakon što igrač točno odgonetne riječ pokaže sljedeći trag (Slika 9 i 10).

```

void RepositionObject ()
{
    if(charObjects.Count == 0)
    {
        return;
    }

    float center = (charObjects.Count - 1) / 2;

    for(int i=0;i<charObjects.Count;i++)
    {
        charObjects[i].rectTransform.anchoredPosition =
        Vector2.Lerp(charObjects[i].rectTransform.anchoredPosition,
            new Vector2((i - center) * space, 0), lerpSpeed * Time.deltaTime);

        charObjects[i].index = i;
    }
}

public void Swap(int indexA, int indexB)
{
    CharObject tmpA = charObjects[indexA];

    charObjects[indexA] = charObjects[indexB];
    charObjects[indexB] = tmpA;

    charObjects[indexA].transform.SetAsLastSibling();
}

```


<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

```

        charObjects[indexB].transform.SetAsLastSibling();
    }
    if (CheckWord())
    {
        Debug.Log("POBJEDA");
    }
}
public bool CheckWord()
{
    string word = "";
    foreach(CharObject charobject in charObjects)
    {
        word += charobject.character;
    }

    if(word == words[currentWord].word)
    {
        StartCoroutine(waiter());
        return true;
    }

    return false;
}
IEnumerator waiter()
{
    yield return new WaitForSeconds(0.5f);

    foreach (CharObject c in charObjects)
    {
        c.Win();
    }

    win.gameObject.SetActive(true);
}

```




Slika 13. Početak anagrama

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>



Slika 14. Uspješno pronađena tražena riječ

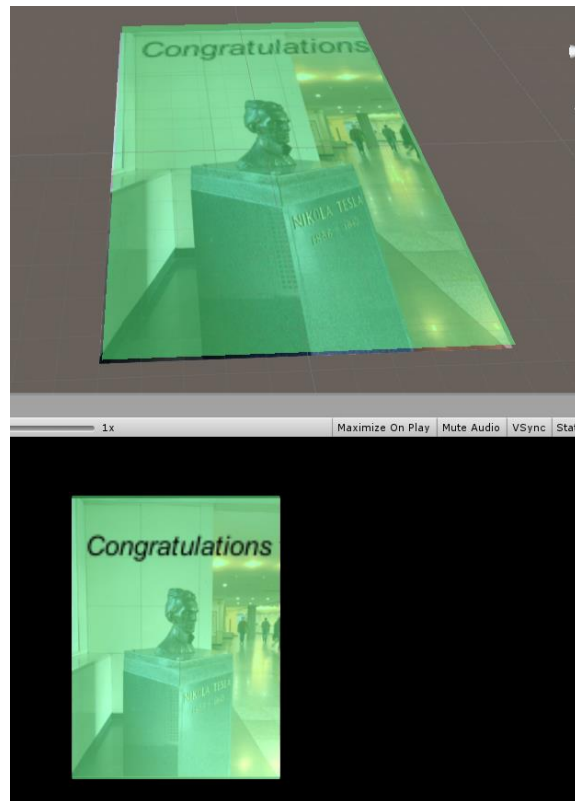
Igrač u ovom trenutku treba pronaći kip Tesle na FER-u te usmjeriti kameru prema njemu i koristeći proširenu stvarnost će se prikazati poruka, ključ i novi trag. Za ovaj dio proširene stvarnosti sam iskoristio Vuforia SDK [1]. Vuforia paket je već unutar samog Unity-a pa ga je potrebno samo uključiti u projekt. Također, na stranicama Vuforie je potrebno staviti marker koji želimo da se prati (Slika 11).

<input type="checkbox"/> Target Name	Type	Rating ⓘ	Status ▾
<input type="checkbox"/>  tesla	Single Image	★★★★★	Active

Slika 15. Slika Tesle koju Vuforia prati

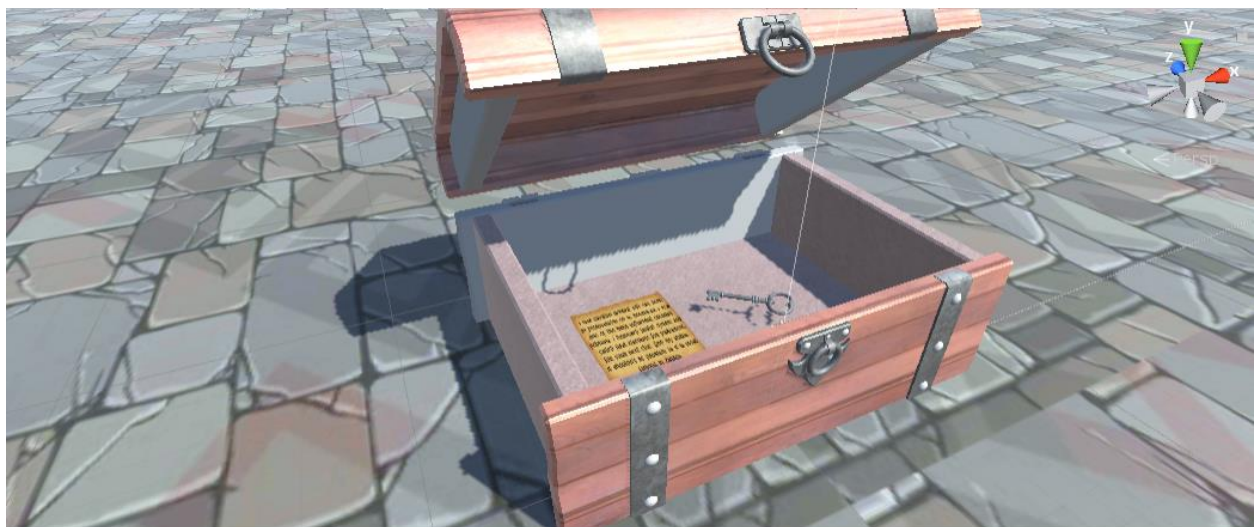
Unutar scene potrebno je dodati ARCamera objekt koji će pratiti traženu sliku, ImageTarget objekt koji predstavlja traženu sliku i ako ju Vuforia uspije prepoznati, dodat će virtualne objekte na zaslon uređaja (Slika 12).

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>



Slika 16. Prikaz slike u sceni

Nakon uspješnog pronalaska ciljane slike, igraču se pušta animacija škrinje u kojoj su ključ i sljedeći trag. Scena je animacija gdje nakon par sekundi korisnik može kliknuti na ključ i trag kako bi mogao nastaviti dalje u potrazi za konačnim blagom (Slika 13).

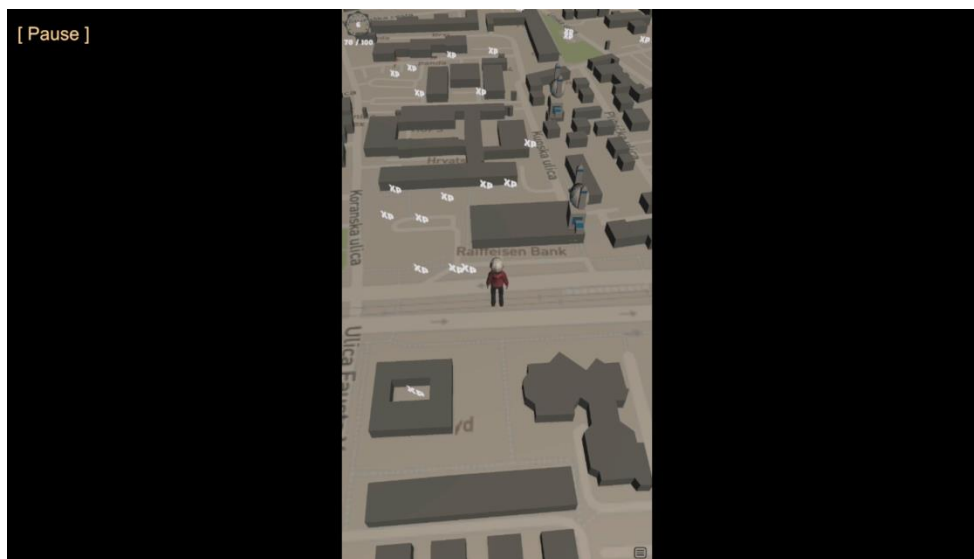


Slika 17. Škrinja s ključem i sljedećim tragom

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

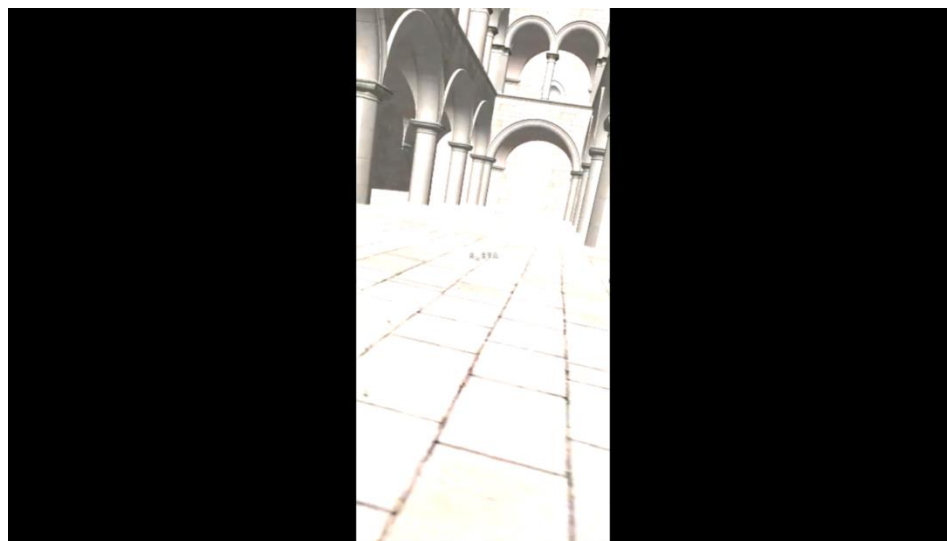
11. Upute za korištenje

Aplikacija se pokrene i korisnik se kreće u stvarnom svijetu dok se avatar u aplikaciji ažurira s obzirom na GPS korisnika (Slika 14).



Slika 18. Prikaz igrača na mapi

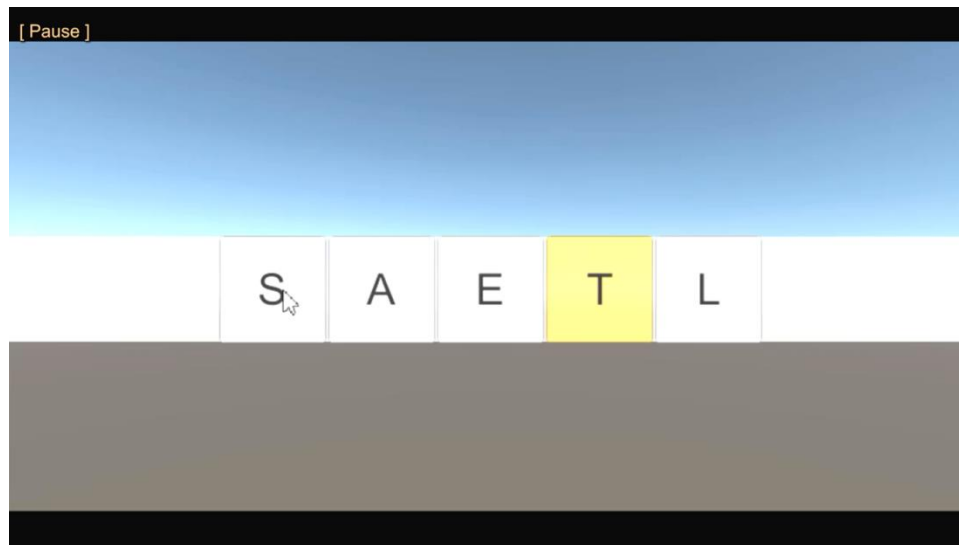
Na mapi igrač može pronaći razne zagonetke, jedna od njih je portal gdje je cilj skupiti određena slova kako bi nastavio dalje s potragom (Slika 15).



Slika 19. Prikaz portala gdje je igrač skupio par slova

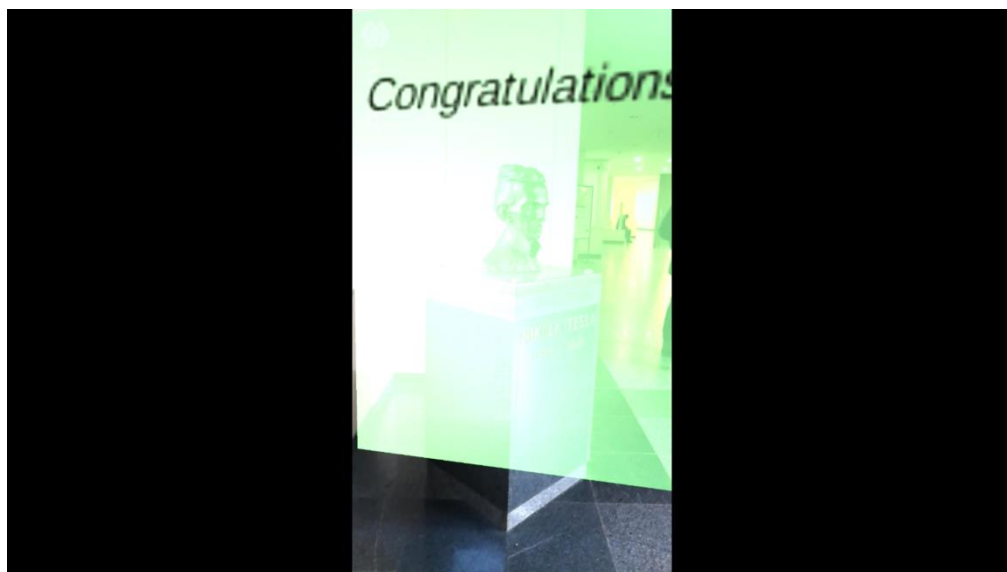
<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Nakon što je skupio sva slova u portalu, korisnika dočeka zaslon sa slovima gdje je potrebno odgonetnuti anagram riječi (Slika 16).



Slika 20. Anagram koji igrač treba odgonetnuti

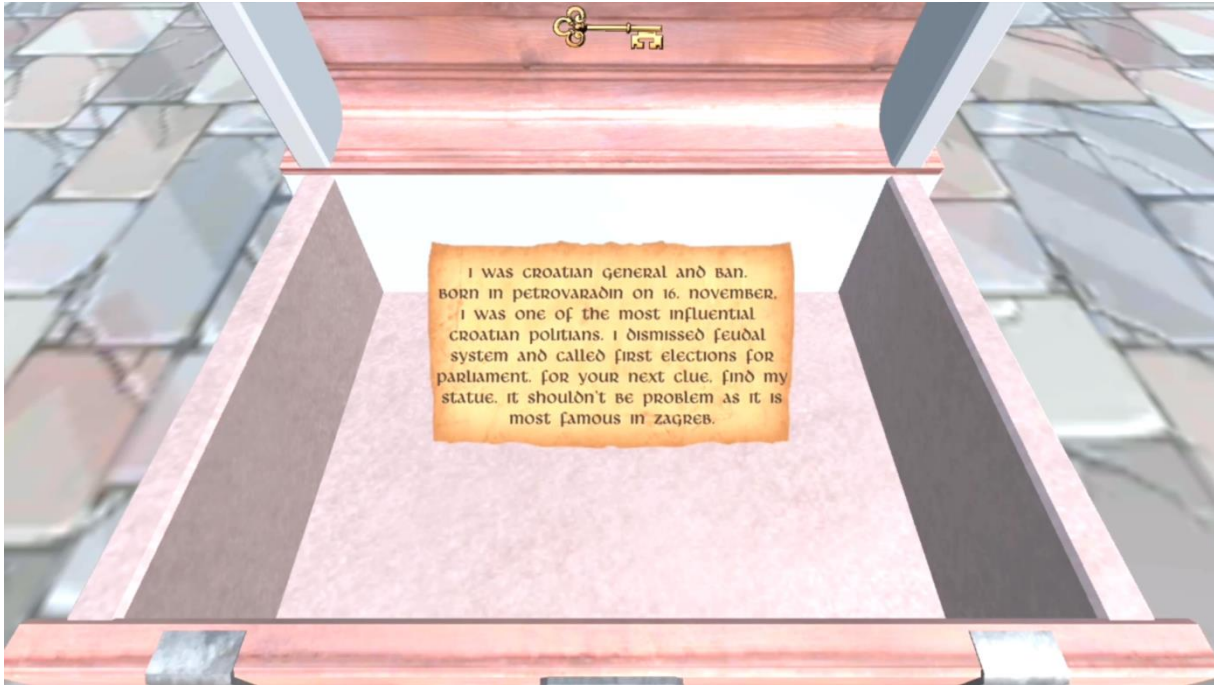
Uspješno premetanje riječi dovodi korisnika do poruke „Now go and find me“ gdje mora otići do kipa Nikole Tesle na FER-u i sa aplikacijom onda uspješno otići po sljedeću zagonetku (Slika 17).



Slika 21. Prikaz virtualnih objekata nakon što je igrač pronašao Teslu

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Konačni trag ovog mini zadatka pokazuje animaciju škrinje korisniku s ključem i papirom na kojoj je novi trag (Slika 18).



Slika 22. Škrinja s ključem i sljedećim tragom

12. Literatura

- [1] Vuforia - <https://developer.vuforia.com/>
- [2] Google ARCore - <https://developers.google.com/ar>
- [3] Mapbox - <https://www.mapbox.com/>
- [4] Getting to know Mapbox - <https://www.youtube.com/watch?v=5YfmnTbVnS0>
<https://www.youtube.com/watch?v=RhG1kfDBhgM>
- [5] Unity - <https://unity.com/>
- [6] Word Scramble - https://www.youtube.com/watch?v=KDjY_FMwIRY
- [7] ARCore Portal - <https://www.youtube.com/watch?v=g78hQB8UKEM>

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Površinska napetost u simulaciji fluida tehnikom SPH
Tehnička dokumentacija
Verzija 1.0

Studentski tim: Jurij Kos

Nastavnik: prof. dr. sc. Željka Mihajlović

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Sadržaj

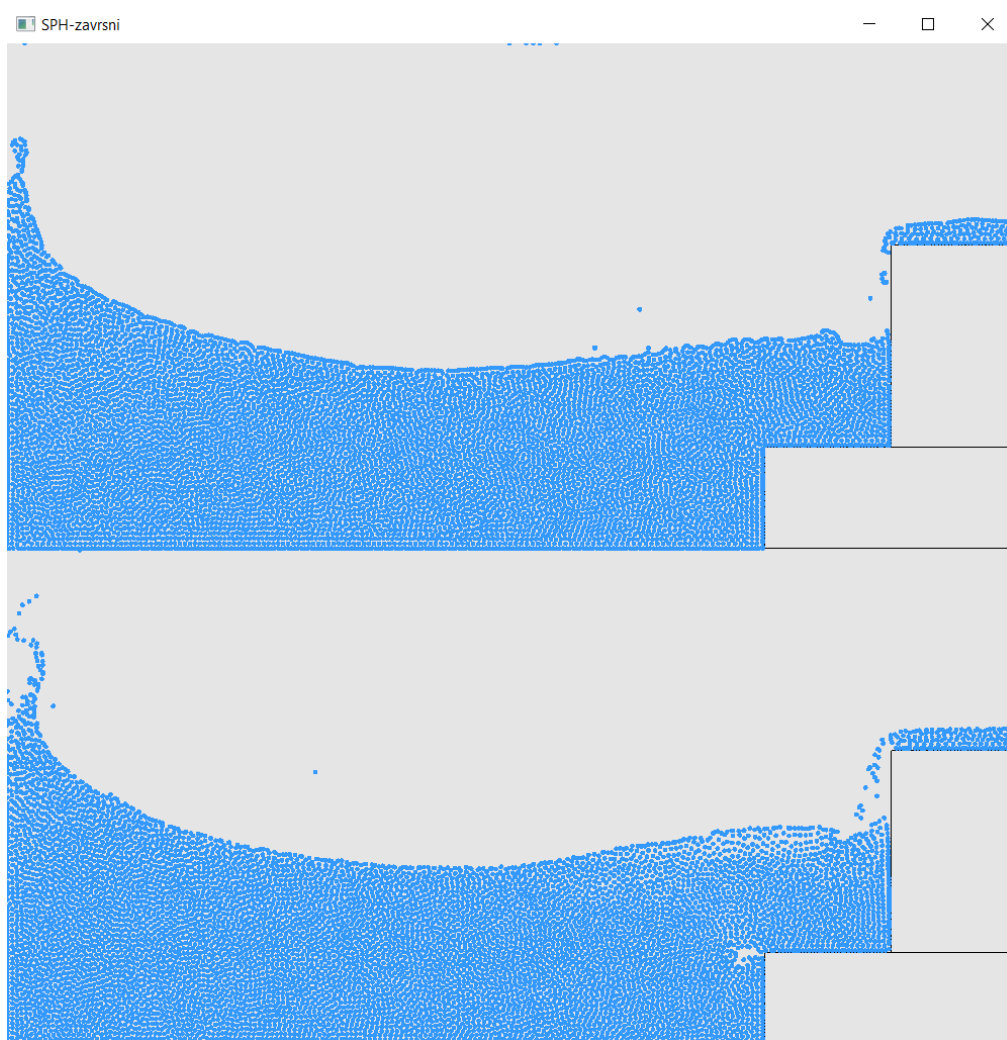
1. Opis razvijenog proizvoda	10
2. Tehničke značajke	4
3. Upute za korištenje	14
4. Literatura	16

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Tehnička dokumentacija

13. Opis razvijenog proizvoda

Cilj projekta je bio istražiti kako se modelira površinska napetost u fluidima simuliranim tehnikom SPH (engl. Smoothed particle hydrodynamics). Za završni rad sam implementirao jednostavan program za simulaciju fluida tehnikom SPH u 2D. U odnosu na završni rad implementacija je proširena modelom površinske napetosti, sudarima s jednostavnim tijelima, te mogućnošću da prvo provedemo računanje simulacije i spremamo pozicije čestica te kasnije prikazemo gibanje fluida koje odgovara brzini gibanja u stvarnosti. Uvedeno je i nekoliko poboljšanja starog modela: korištenje jezgrene funkcije koja je bolje prilagođena 2D, promjenjivi vremenski korak kod integracije koji ovisi o trenutnom stanju sustava i preciznija jednačba stanja pomoću kojeg se računa tlak u fluidu. Program je napisan u programskom jeziku c++ i koriste se biblioteke OpenGL, GLFW, imgui i glm.



Slika 1. Usporedba fluida s površinskom napetosti i bez površinske napetosti

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

14. Tehničke značajke

2.1 Model površinske napetosti

Površinska napetost je implementirana prema modelu koji je opisan u radu „Versatile Surface Tension and Adhesion for SPH Fluids”. Površinska napetost se sastoji od dvije komponente: kohezije i sile koja smanjuje površinu vanjskog ruba fluida. Kohezija je privlačna međumolekularna sila koja djeluje između susjednih čestica i računamo je prema izrazu

$$F_{cohesion\ i \rightarrow j} = -\gamma m_i m_j C(|\mathbf{r}_i - \mathbf{r}_j|) \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|}$$

gdje C izračunavamo prema

$$C(r) = \frac{32}{\pi h^9} \begin{cases} (h-r)^3 r^3, & 2r > h \text{ and } r \leq h \\ 2(h-r)^3 r^3 - \frac{h^6}{64}, & r > 0 \text{ and } 2r \leq h \\ 0, & \text{otherwise} \end{cases}$$

γ predstavlja konstantu kojom opisujemo površinsku napetost u ovisnosti o vrsti fluida, s m označavamo mase čestica a r je vektor koji označava poziciju čestica. Izraz za izračunavanje sile koja smanjuje površinu vanjskog ruba računamo prema formulama

$$F_{curvature\ i \rightarrow j} = -\gamma m_i (\mathbf{n}_i - \mathbf{n}_j)$$

$$\mathbf{n}_i = k \Sigma \frac{m_j}{\rho_j} \nabla W(|\mathbf{x}_i - \mathbf{x}_j|)$$

\mathbf{n}_i predstavlja normalu u čestici i. Koriste se još i sljedeće formule:

$$K_{ij} = \frac{2\rho_0}{\rho_i + \rho_j}$$

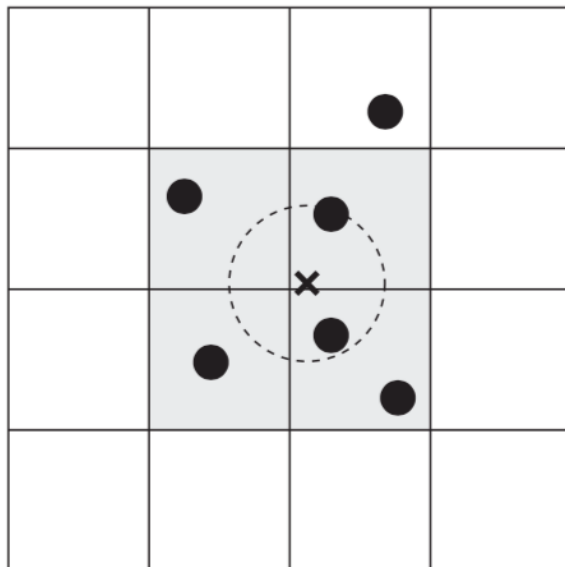
$$F_{st\ i \rightarrow j} = K_{ij} (F_{cohesion\ i \rightarrow j} + F_{curvature\ i \rightarrow j})$$

ρ_0 je gustoća fluida u mirovanju, a ρ_i i ρ_j su gustoće susjednih čestica. K je korekcijski faktor kojim postižemo simetriju sila te povećavamo silu na rubovima s obzirom na to da rubne čestice imaju gustoću manju od gustoće u mirovanju.

2.2 Pronalaženje susjednih čestica

Najjednostavniji način pronalaska susjeda je za svaku česticu proći kroz sve čestice i provjeriti nalaze li se ono unutar radijusa zaglađivanja, ali je to istovremeno i najneefikasniji način kod kojega vrijeme pronalaska susjeda raste kvadratno s povećanjem broja čestica. Pronalaženje susjeda implementirano je tako da u prostor u kojemu simuliramo fluid postavimo rešetku podijeljenu na polja koja imaju stranice jednake duljini radijusa zaglađivanja. Na početku svakog koraka upišemo čestice u odgovarajuća polja. Kako je veličina polja jednaka radijusu zaglađivanja dovoljno je provjeriti samo čestice koje se nalaze unutar 4 polja kao što je to prikazano na slici 2.

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>



Slika 2. Traženje čestica unutar radijusa zaglađivanja

15. Upute za korištenje

Rješenje je podijeljeno u dva odvojena programa te prevođenjem dobivamo dvije izvršne datoteke. Prva se koristi za kreiranje različitih scena preko komandne linije (različite početne pozicije fluida i prepreka te parametara koji određuju ponašanje fluida). Nakon što prvi program završi imamo po jedan tekstualni file za svaku sliku koju prikazujemo. Nakon toga je potrebno pokrenuti drugi program koji čita pozicije čestica iz file-ova i prikazuje ih na ekran u 60 fps-a.

16. Literatura

- [1] Nadir Akinici, Gizem Akinici, Matthias Teschner, Versatile Surface Tension and Adhesion for SPH Fluids, 2014.
- [2] Kim Doyub, Fluid Engine Development, 2016.
- [3] Markus Becker, Matthias Teschner Weakly compressible SPH for free surface flows, 2007.
- [4] Matthias Müller, David Charypar and Markus Gross, Particle-Based Fluid Simulation for Interactive Applications, 2004.

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Simulacija vodene površine uporabom FFT tehnike
Tehnička dokumentacija
Verzija 1.0

Studentski tim: Nikola Nađ

Nastavnik: prof. dr. sc. Željka Mihajlović

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

Sadržaj

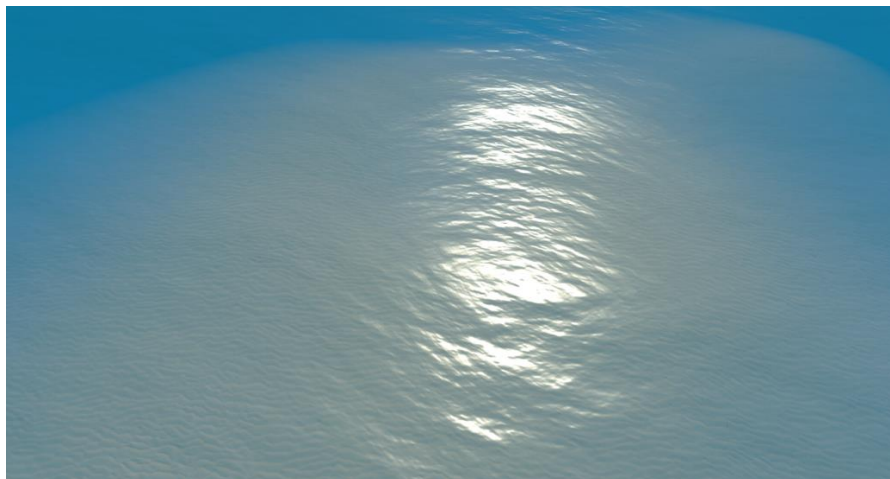
1. Opis razvijenog proizvoda	3
2. Tehničke značajke	4
3. Upute za korištenje	8
4. Literatura	9

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

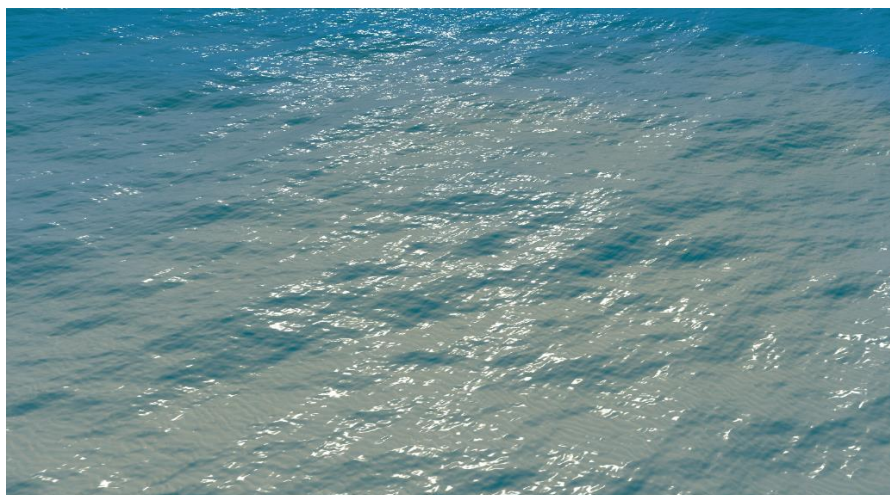
Tehnička dokumentacija

17. Opis razvijenog proizvoda

Cilj projekta bio je napraviti simulaciju otvorene vodene površine (oceana) koja bi se mogla izvoditi u stvarnom vremenu, uporabom brze Fourierove transformacije (engl. Fast Fourier Transform, FFT). FFT je korišten za generiranje visinske mape na temelju Phillipsovog spektra te se izvodi na grafičkoj kartici kako bi se poboljšale performanse. Na temelju visinske mape generira se i mapa normala, koja se koristi u daljnjem sjenčanju vodene površine. Projekt je u cijelosti izrađen u Unity-ju; skripte su pisane u programskom jeziku C#, računski sjenčari (engl. Compute Shader) su pisani u jeziku HLSL (engl. High-Level Shading Language), dok je sjenčar vode napravljen u Shader Graph-u, Unity-jevom vizualnom alatu za izradu sjenčara. Primjer uporabe razvijenog proizvoda su video igre, s obzirom da je u njima važno izvođenje u stvarnom vremenu.



Slika 23 Prikaz simulirane vode



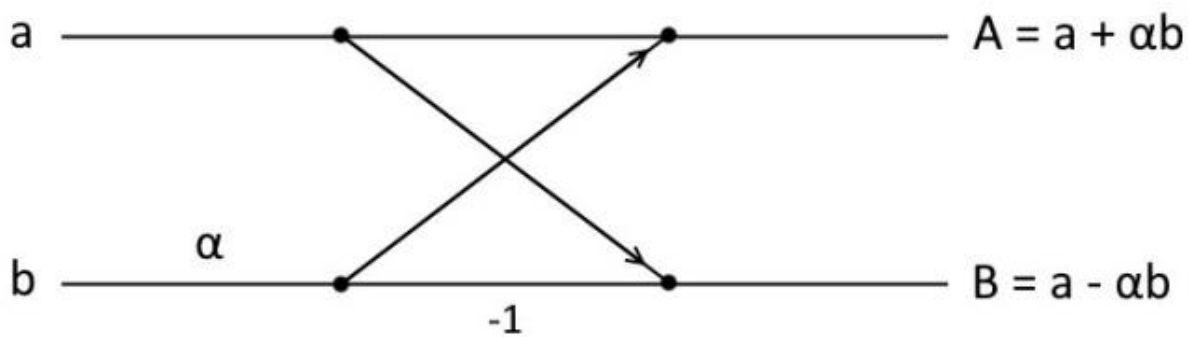
Slika 24 Prikaz simulirane vode s izraženijim valovima

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

18. Tehničke značajke

18.1 Brza Fourierova transformacija (Fast Fourier Transform, FFT)

Osnovni koncept brze Fourierove transformacije je repozicioniranje sumanada standardne diskretne Fourierove transformacije (Discrete Fourier Transform, DFT) kako bi se međurezultati mogli višestruko iskoristiti, dok se u DFT-u ti međurezultati u svakom koraku ponovno računaju. Složenost DFT postupka je $O(N^2)$, dok je složenost FFT postupka $O(N * \log_2(N))$, što predstavlja značajnu razliku za velik broj uzoraka N . Korišten je algoritam Cooley-Tukey, temeljen na strategiji podijeli pa vladaj (engl. Divide and Conquer, gdje se Fourierova transformacija dijeli na veći broj manjih DFT-ova za proračun međurezultata. Algoritam se izvodi u $\log_2(N)$ faza, a izlaz svake faze predstavlja ulaz za sljedeću. Algoritam se može zorno prikazati tzv. leptir dijagramom (engl. butterfly diagram) (Slika 25). Ime je dobio po obliku, koji podsjeća na leptirova krila. Operacija ima dva ulaza, a i b , te dva izlaza, A i B , koji su dobiveni sumom ulaza pomnoženih faktorima koji su naznačeni na bridovima (npr. da bi se dobio izlaz A , ulazu a se pribraja ulaz b pomnožen faktorom α).



Slika 25 Leptir dijagram

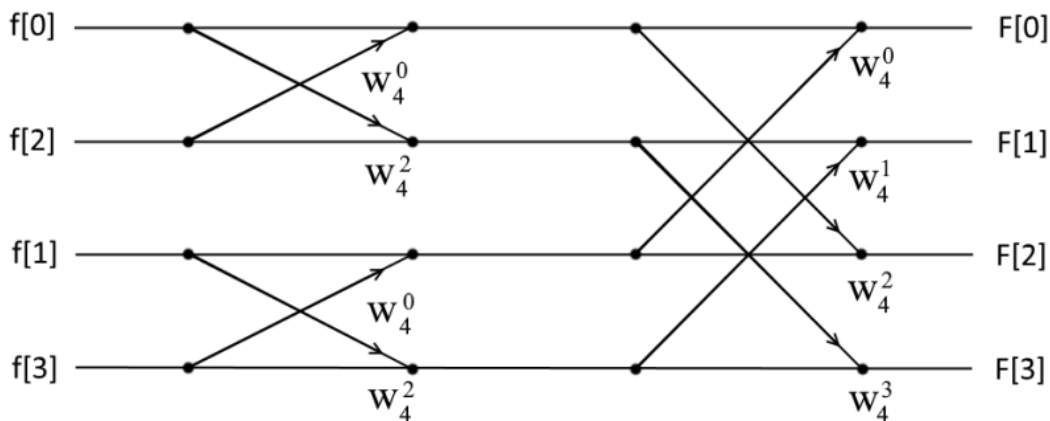
Primjer proračuna FFT-a prikazan je na slici (Slika 26) za $N = 4$. Vektor f sadrži vrijednosti funkcije f u N točaka, a F predstavlja spektar te funkcije. Pritom je

$$W_N = e^{-i\frac{2\pi}{N}},$$

a taj faktor proizlazi iz formule za diskretnu Fourierovu transformaciju:

$$F(k) = \sum_{n=0}^{N-1} f(n) * e^{\frac{-2\pi i k n}{N}}.$$

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>



Slika 26 Prikaz brze Fourierove transformacije leptir dijagramom za $N = 4$

18.2 Generiranje visinske mape

Polazna ideja bila je implementirati visinu vodene površine $h(\mathbf{x}, t)$ u točki $\mathbf{x} = (x, z)$ kao sumu sinusoida s kompleksnom, vremenski ovisnom amplitudom:

$$h(\mathbf{x}, t) = \sum_{\mathbf{k}} \tilde{h}(\mathbf{k}, t) * e^{i\mathbf{k} \cdot \mathbf{x}}$$

Ovdje je t vrijeme, a $\mathbf{k} = (k_x, k_z)$ dvodimenzionalni vektor koji određuje smjer propagacije vala. Pritom je

$$k_x = \frac{2\pi n}{L}$$

i

$$k_z = \frac{2\pi m}{L}$$

dok su m i n iz intervala

$$-\frac{N}{2} \leq m, n \leq \frac{N}{2}.$$

FFT postupak na temelju sinusoida generira visinu vodene površine u diskretnim točkama $\mathbf{x} = (nL/N, mL/N)$. $\tilde{h}(\mathbf{k}, t)$ su Fourierove komponente koje predstavljaju amplitude te one određuju konačnu visinu vodene površine. Ovime se dobivaju visine točaka unutar prostora dimenzija $L \times L$, a izvan tog područja visine su periodičke, pa je moguće definirati vodenu površinu proizvoljne veličine bez vidljivih šavova (iako je moguće primijetiti periodičnost ako se površina pogleda iz dovoljno velike daljine).

Oceanografsko istraživanje pokazalo je da su amplitude $\tilde{h}(\mathbf{k}, t)$ približno nezavisne i statistički stacionarne Gaussove fluktuacije s Phillipsovim spektrom $P_h(\mathbf{k})$:

$$P_h(\mathbf{k}) = A \frac{e^{-\frac{1}{(kL)^2}}}{k^4} |\hat{\mathbf{k}} \cdot \hat{\mathbf{w}}|^2,$$

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

gdje $L = V^2/g$ predstavlja najveće moguće valove uzrokovane konstantnim vjetrom brzine V , g je gravitacijska konstanta, k je norma vektora \mathbf{k} , \mathbf{w} je smjer vjetra, a A je konstanta. Faktor $|\hat{\mathbf{k}} \cdot \hat{\mathbf{w}}|^2$ prigušuje valove koji su približno okomiti na smjer vjetra, a povećanjem eksponenta to prigušenje se može pojačati. Polazne Fourierove amplitude računaju se sljedećim izrazom:

$$\tilde{h}_0(\mathbf{k}) = \frac{1}{\sqrt{2}}(\xi_r + i\xi_i)\sqrt{P_h(\mathbf{k})},$$

gdje su ξ_r i ξ_i slučajne varijable sa standardnom normalnom razdiobom (centrirane oko 0 sa standardnom devijacijom 1). Na temelju polaznih amplituda i relacije disperzije $\omega(k)$ dobivaju se Fourierove amplitude u trenutku t :

$$\tilde{h}(\mathbf{k}, t) = \tilde{h}_0(\mathbf{k})e^{i\omega(k)t} + \tilde{h}_0^*(-\mathbf{k})e^{-i\omega(k)t}.$$

Relacija disperzije predstavlja vezu između norme valnog vektora \mathbf{k} (engl. wavevector) i valne frekvencije, a za duboku vodu ona glasi $\omega(k) = \sqrt{gk}$.

Konačna jednadžba pomoću koje se dobiva visinska mapa je inverzna dvodimenzionalna Fourierova transformacija vremenski ovisnih kompleksnih amplituda $\tilde{h}(\mathbf{k}, t)$ te ona glasi (nakon sređivanja):

$$h(n, m, t) = \frac{1}{N^2}(-1)^n \sum_{l=0}^{N-1} \left[(-1)^m \sum_{j=0}^{N-1} \tilde{h}(l, j, t) e^{i\frac{2\pi m j}{N}} \right] e^{i\frac{2\pi n l}{N}}$$

Ovdje su uvedene varijable l i j s rasponom $0 < l, j < N - 1$ te je vektor \mathbf{k} redefiniran njima na sljedeći način:

$$\mathbf{k} = \left(\frac{2\pi l - \pi N}{L}, \frac{2\pi j - \pi N}{L} \right).$$

Generiranje visinske mape implementacijski je izvedeno u 5 koraka: prvi korak je proračun polaznih komponenata $\tilde{h}_0(\mathbf{k})$ i $\tilde{h}_0^*(-\mathbf{k})$. Ovaj korak izvršava se jedanput na početku izvođenja programa. Drugi korak je proračun amplituda $\tilde{h}(\mathbf{k}, t)$ na temelju polaznih komponenata, što se izvodi za svaku sličicu/okvir (engl. frame). Treći korak je proračun N horizontalnih 1D brzih Fourierovih transformacija gdje su ulaz amplitude iz prethodnog koraka. Četvrti korak je proračun N vertikalnih 1D brzih Fourierovih transformacija gdje se kao ulaz koristi izlaz prethodnog koraka, a peti korak radi konačnu multiplikaciju s $(-1)^{m+n}$ i $1/N^2$ kako bi se dobila konačna visinska mapa.

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

```

Initial Spectrum  $\tilde{h}_0(\mathbf{k})$  and  $\tilde{h}_0(-\mathbf{k})$  shaderpass;
while isRunning do
    | Fourier components  $\tilde{h}(\mathbf{k}, t)$  shaderpass;
    | pingpong := 0;
    | for  $i=0$  to  $i < \log_2 N$  do
    | | horizontal butterfly shaderpass for stage  $i$ ;
    | | pingpong := pingpong++ mod 2;
    | end
    | for  $i=0$  to  $i < \log_2 N$  do
    | | vertical butterfly shaderpass for stage  $i$ ;
    | | pingpong := pingpong++ mod 2;
    | end
    | Inversion and permutation shaderpass;
end

```

Slika 27 Pseudokod algoritma za generiranje visinske mape

18.3 Generiranje mapa normala

Mapa normala generira se na temelju visinske mape dobivene postupkom opisanim u prethodnom potpoglavlju uporabom Sobelovog filtera. Sobelov filter često se koristi u obradi slika i u računalnom vidu za detektiranje rubova na slici, odnosno dijelova slike na kojem se javljaju velike promjene u boji. Radi se o konvoluciji s dvije jezgre dimenzija 3x3 pomoću kojih se računa aproksimacija gradijenta intenziteta slike u horizontalnom i vertikalnom smjeru. Korištene jezgre prikazane su na slici (Slika 28). Pritom jezgra A služi za aproksimaciju gradijenta u horizontalnom smjeru, a jezgra B u vertikalnom smjeru. Konvolucija jezgrom A rezultira x komponentom normale, dok konvolucija jezgrom B daje y komponentu. Z komponenta umjetno se dodaje u proizvoljnom iznosu, čime se može kontrolirati izraženost normale – ako je Z komponenta velika, smanjit će se relativni doprinos x i y komponentata nakon normalizacije, te će normala biti pretežno usmjerena prema gore.

$$A = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad B = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Slika 28 Jezgre Sobelovog filtra

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

18.4 Sjenčar vode

Sjenčar zadužen za vizualni prikaz vode izrađen je u Shader Graph-u, vizualnom Unity-jevom alatu za definiranje sjenčara pomoću grafova. Svaki čvor u grafu može predstavljati ulazni podatak (npr. vektor, teksturu), operaciju s određenim ulazima i izlazima (npr. množenje vektora) ili krajnji čvor. Sjenčar kao osnovne ulazne podatke koristi prethodno opisanu visinsku mapu i mapu normala. Visinska mapa koristi se za pomicanje vrhova objekta po Y osi, dok se normala očitana iz mape normala predaje krajnjem čvoru. Pritom se koriste globalne UV koordinate, odnosno koordinate dobivene na temelju x i z koordinata vrha objekta. Normale se koriste i za jednostavan efekt refrakcije (Slika 29), na način da se uzme boja ekrana na poziciji pomaknutoj u smjeru normale za određeni faktor. Taj faktor se povećava s dubinom vode, pa efekt gotovo nije primjetljiv u plitkoj vodi, dok dolazi do izražaja u dubljoj vodi. Faktor dubine također se koristi i kao faktor kojim se miješaju boja dobivena refrakcijom i boja vode definirana kao nijansa plave boje, pa je voda u plićim dijelovima prozirna, dok je na dovoljnoj dubini neprozirna i u potpunosti plava.



Slika 29 Demonstracija refrakcije na crvenoj kocki u plitkoj (lijeva slika) i dubokoj vodi (desna slika). Također se može primijetiti kako je na većoj dubini voda manje prozirna te je izraženija plava boja.

<Naziv projekta>	Verzija: <1.0>
Tehnička dokumentacija	Datum: <dd/mm/yy>

19. Upute za korištenje

U projektu se nalazi scena „WaterSimulationScene“ u kojoj se nalaze kamera, izvor svjetlosti, vodena površina, morsko dno i HeightMapGenerator objekt na kojem se nalazi glavna skripta za proračun visinske mape i mape normala. Selekcijom tog objekta moguće je podešavati parametre generatora. Na vodenu površinu primijenjen je materijal „Water“, koji također ima podešive parametre. Da bi se vodena površina koristila u proizvoljnoj sceni, potrebno je na plohu postaviti materijal „Water“ te u scenu dodati objekt HeightMapGenerator.

Literatura

- [1] Jerry Tessendorf, Simulating Ocean Water, 2001.
- [2] Fynn-Jorin Flügge, Realtime GPGPU FFT Ocean Water Simulation, 2017.
- [3] Keith Lantz, Ocean simulation using the fast Fourier transform, 2011.
<https://www.keithlantz.net/2011/11/ocean-simulation-part-two-using-the-fast-fourier-transform/>