Izrada modela za simulaciju širenja vatre po terenu Tehnička dokumentacija

Verzija 1.0

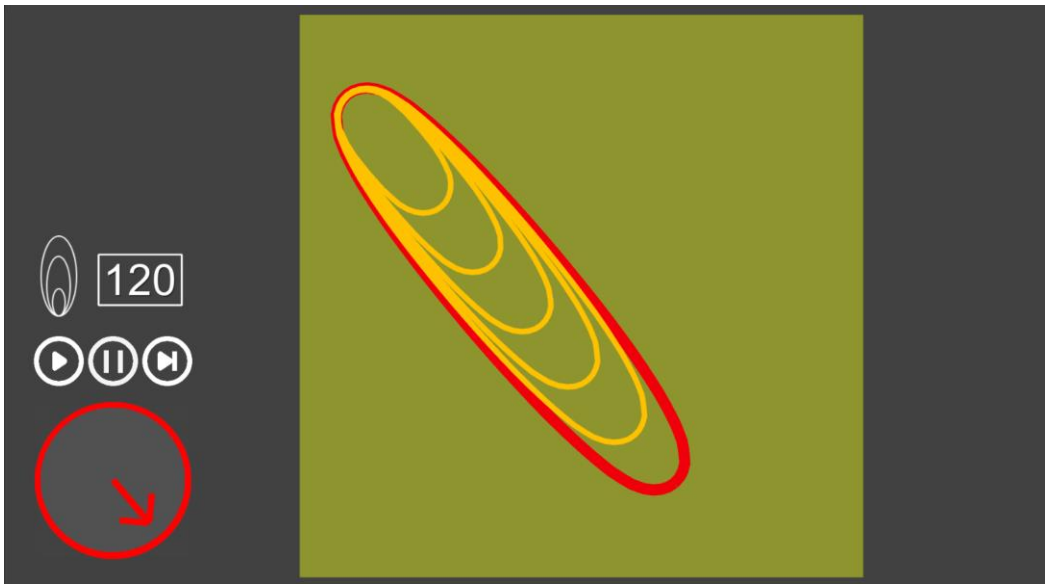
Studentski tim: **Bruno Banek**

Nastavnik: **Prof. dr.sc. Željka Mihajlović**

Sadržaj

1. Opis razvijenog proizvoda

Unutar projekta razvijen je sustav za simulaciju površinskog širenja vatre zasnovan na Rothermelovom modelu. Sustav je razvijen unutar programskog alata Unity. Na ulazu sustava postavljaju se vegetacijski biomi nad kojima će se odvijati simulacija površinskog širenja vatre, te se ili automatski generira tekstura koja će sadržavati sve biome, ili se ručno označe regije pojedinih bioma. Nakon toga se postavlja jačina i smjer puhanja vjetra, te pokreće simulacija. Simulacija se odvija dok ju korisnik ne prekine. Za simulaciju površinskog širenja vatre koristi se implementirani Rothermelov model zasnovan na Huygens-ovom principu.

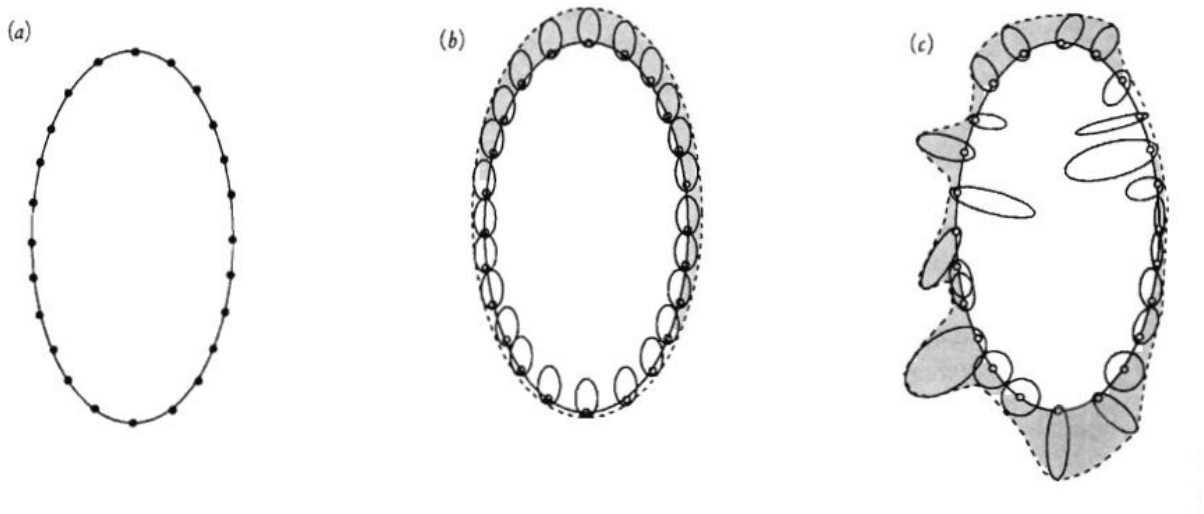


Slika 1. Prikaz simulacije širenja vatre

2. Tehničke značajke

2.1. Propagacija vatrenog fronta

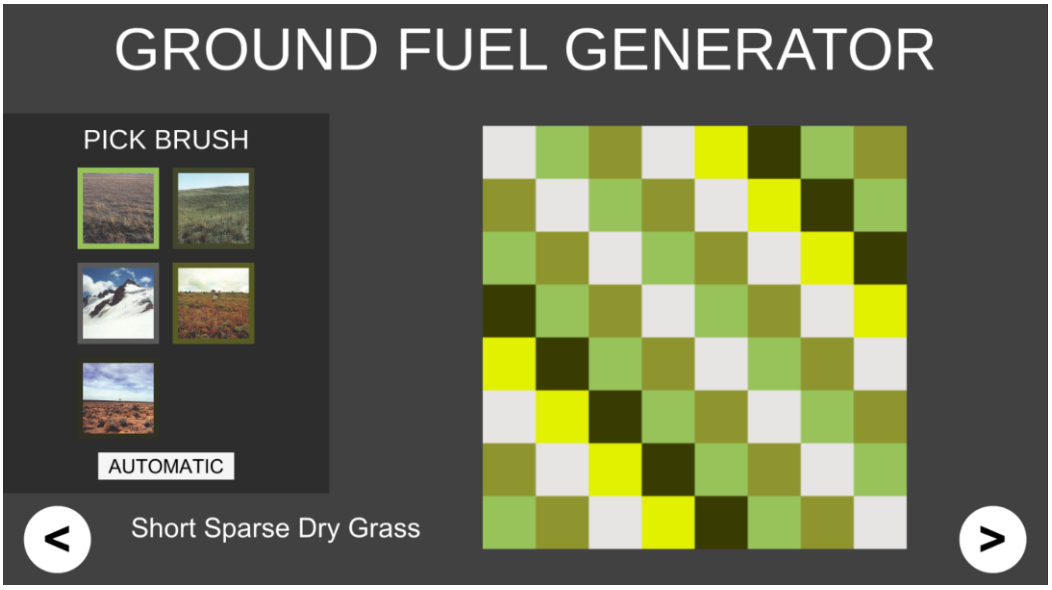
Na početku simulacije postavlja se točka zapaljenja vatre, iz koje se model počinje širiti. Osnovni oblik s kojim se započinje širenje jest elipsa čiji je kut sa horizontalnom osi, te omjer male i velike osi određen početnim postavkama vjetra. Na elipsi se određuje N početnih točaka koje su međusobno na bliskim udaljenostima, koje predstavljaju vatrenu frontu u određenom vremenskom trenutku. Vegetacijski parametri uzorkuju se iz teksture za svaku pojedinu točku, te se propagacija vatrenog fronta vrši za svaku točku na fronti, poštujući Huygens-ov princip. Huygens-ov princip nalaže kako se valni front može propagirati koristeći bilo koju točku na njegovom rubu kao neovisan izvor novog dijela vala (*engl. wavelet*). Novi dio vala predstavlja elipsu veličine određene vremenskim intervalom simulacije i brzinom širenja u toj točki. Orijentacija elipse određena je maksimalnim smjerom širenja vatre, koji se određuje preko vektora smjera puhanja vjetra i vektora nagiba terena u toj točki širenja (trenutno u implementiranom simulatoru nagib terena nije uzet u izračun pošto se simulacija odvija u dvodimenzionalnom prostoru). Omjer velike i male osi elipse određen je funkcijom brzine širenja koja ovisi o jačini vjetra izmjerenog na pola visine vatrene fronte. Nakon izračuna za svaku točku vatrenog fronta, kao nova fronta uzima se omotnica elipsi izračunatih propagacijom.



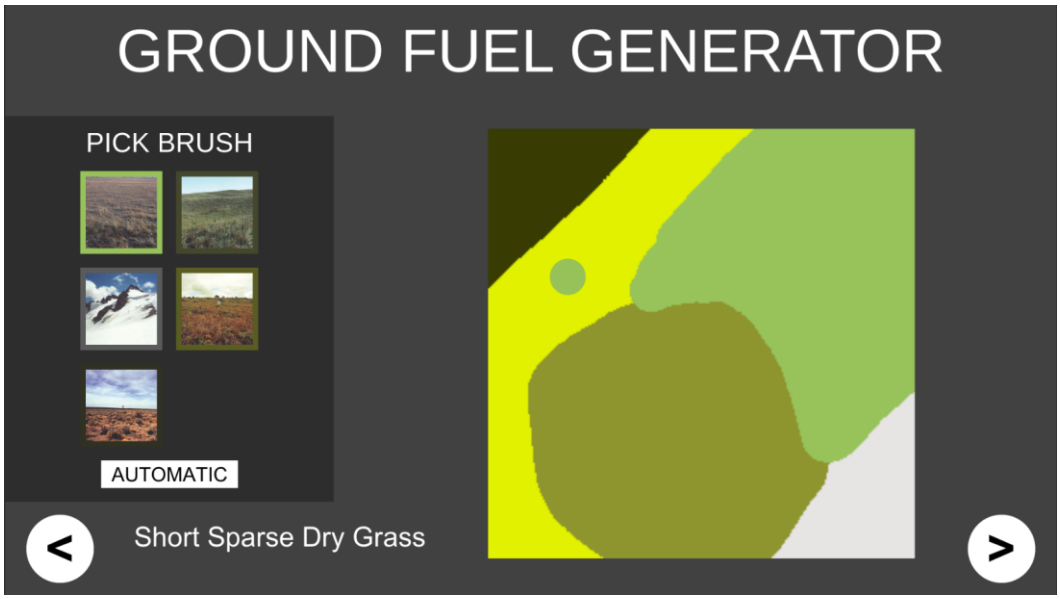
Slika 2. Prikaz propagacije za početnu frontu a), uz uniformne uvjete b) i neuniformne uvjete c)

2.2 Generiranje teksture vegetacijskih značajki

Za simulaciju širenja vatre potrebne su vegetacijske značajke terena po kojem se vatra širi, u svakoj točki računanja propagacije. Značajke se uzorkuju iz teksture vegetacijskih značajki koju korisnici mogu konstruirati unutar proizvedene aplikacije nakon odabira vegetacijskih bioma koji će se koristiti unutar simulacije. Tekstura se može generirati automatski, čime nastaje tekstura oblika šahovnice, koja podjednako zastupljuje svaku vrstu vegetacije. Dodatno, korisnici mogu izraditi teksturu odabirom željenog tipa vegetacije kao kista, kojim se zatim pritiskom miša obilježava predio na teksturi gdje će odabrana vegetacija postojati.



Slika 3. Prikaz automatske teksture vegetacije



Slika 4. Prikaz ručno generirane teksture vegetacije

3. Upute za korištenje

Nakon pokretanja aplikacije, prikazuje se zaslon sa vegetacijskim biomima koji se mogu koristiti pri simulaciji. Korisnik odabire vegetacijske biome koje želi koristiti pritiskom miša na ikonu istog. Nakon pritiska ikona će dobiti bijeli obrub čime će označavati da je uspješno označen biom, ponovnim klikom ikona će izgubiti bijeli obrub, te će vegetacijski biom biti odbačen. Nakon odabira bioma, pritiskom na strelicu ">" otvara se novi zaslon gdje se izrađuje tekstura vegetacije opisana u poglavlju 2.2. Pritiskom na strelicu ">" otvara se posljednji zaslon gdje se odvija simulacija širenja vatre. Postavljaju se početni uvjeti smjera i jačine vjetra pritiskom mišem unutar crvenog kruga. Iznad kruga nalaze se instrukcije za zaustavljanje, pokretanje i odrađivanje jednog koraka simulacije. Iznad njih nalazi se polje unutar kojeg se unosi broj koji označava koliko simulacijskih koraka treba proći između vizualizacije položaja vatrene fronte. Pritiskom miša bilo gdje na teksturu vegetacije pokreće se simulacija širenja vatrenog fronta sa izvorom postavljenim na mjestu pritiska miša.

4. Literatura

[1] Finney, Mark A. 1998. FARSITE: Fire Area Simulator—Model development and evaluation. Res. Pap. RMRS-RP-4. Ogden, UT: U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station. 47 p.

[2] G. Richards, An elliptical growth model of forest fire fronts and its numerical solution, Int. J. Num. Meth. Engineering, 30, 1163-1179 (1990).

[3] Barber, Bose, Bourlioux, Braun, Brunelle, Garcia, Hillen, Lam, Ong, PROMETHEUS: Canada's Wildfire Growth Simulator.

[4] Joe H. Scott, Robert E. Burgan, Standard Fire Behaviour Fuel Models: A Comprehensive Set for Use with Rothermel's Surface Fire Spread Model, General Techincal Report, (2005).

Simulacija vatre metodom sustava čestica

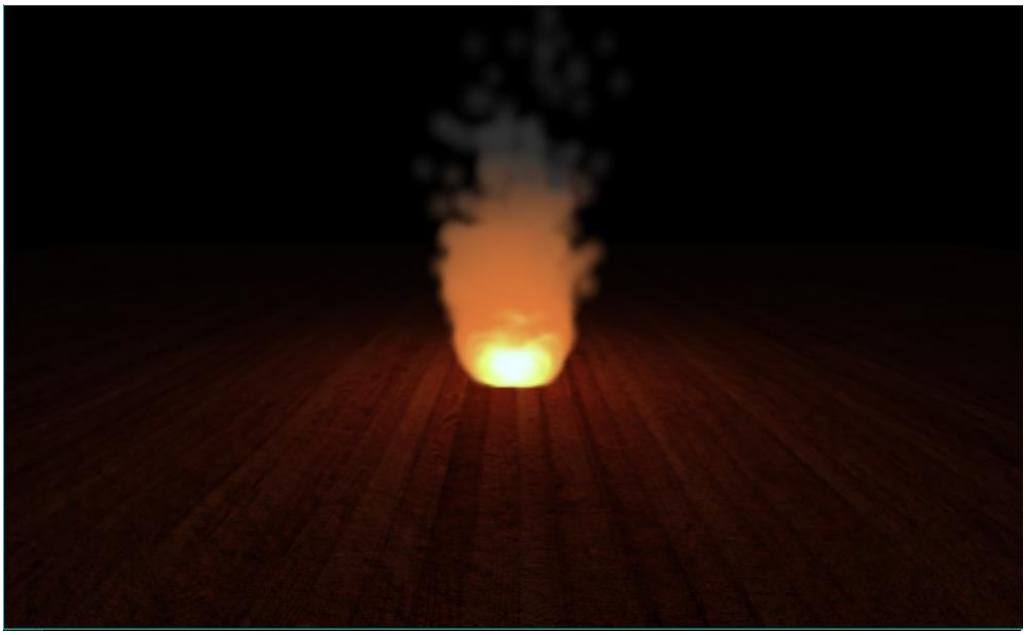
Studentski tim: <Anteo Ivankov>

Nastavnik: <Željka Mihajlović>

Sadržaj

1. Opis razvijenog proizvoda

Cilj projekta bio je prikaz simulacije vatre metodom sustava čestica. U sklopu projekta napravljena je jednostavna scena koja prikazuje simulaciju vatre. Programu je za rad potrebna tekstura čestice vatre te tri programa za sjenčanje. Projekt je napisan u programskom jeziku C++ uz korištenje grafičke knjižnice OpenGL verzije 4.1. Korištene knjižnice su SDL (Simple DirectMedia Layer) koja služi za kreiranje prozora te za upravljanje događajima kao što su pritisak tipke na tipkovnici ili pomak miša, GLM (OpenGL Mathematics) koja olakšava korištenje matematičkih konstrukata poput vektora i matrica, SDL_image za učitavanje tekstura i GLEW (OpenGL Extension Wrangler Library) koji služi za lakše korištenje OpenGL funkcionalnosti. Trenutno su u programu zapisane sve vrijednosti potrebne za izvođenje kao što su broj čestica koji se prikazuje na ekranu, tekstura koja se koristi za prikaz vatre te programi koji se koriste za sjenčanje.



Slika 1 Prikaz scene s vatrom

2.Tehničke značajke

Sustav čestica

Sustav čestica je tehnika kojom se modeliraju različite pojave koje se teško prikazuju klasičnim tehnikama iscrtavanja. Metoda sustava čestica pogodna je za prikaz pojava poput vatre, snijega, dima, iskri, oblaka i slično. Tipična implementacija uključuje postojanje izvora iz kojeg nastaju čestice. Svaka čestica opisana je određenim brojem parametara kao što su pozicija, trenutna brzina, vrijeme koje je preostalo prije nego čestica nestane te mnogi drugi. U svakom koraku svakoj čestici se osvježe njeni parametri te se potom čestice šalju na iscrtavanje. Najčešće se svaka čestica prikazuje kao plakat u obliku četverokuta (engl. Billboarded quad) zajedno s pripadajućom teksturom.

Iscrtavanje čestice

Svaka čestica u programu modelirana je primjerkom klase koja sadrži informaciju o poziciji čestice, trenutnoj brzini čestice te koliko vremena još treba proći da čestica nestane. Prije iscrtavanja čestice potrebno je postavljanje matrice pogleda, modela, projekcijske matrice kao i odgovarajućih tekstura.. Prvi korak prilikom iscrtavanja je izvođenje programa sjenčara vrhova. U ovom slučaju radi se o vrlo jednostavnoj implementaciji sjenčara vrhova u kojoj se matrica pogleda i modela množe s pozicijom čestice te se rezultat prosljeđuje u sjenčar geometrije. S obzirom da su primitivi koji su korišteni prilikom poziva funkcije za iscrtavanje postavljeni na točke, Zadatak sjenčara geometrije je od točke napraviti novu geometriju kako bi se čestica prikazala. Dakle, u sljedećem koraku sjenčar geometrije iz svake primljene točke emitira četiri nove točke. Te nove točke se računaju na način da se točka koja je dobivena u sjenčaru geometrije interpretira kao središte kvadrata a nove točke onda predstavljaju vrhove tog kvadrata. Na taj način se dobila geometrija kvadrata koja se dalje šalje sjenčaru fragmenta. U zadnjem koraku sjenčar fragmenta na temelju uzorkovanja teksture te udaljenosti od izvora računa konačnu boju.

Određivanje parametara čestice

akceleracija svake čestice izračunava se na način da se za svaku česticu u obzir uzimaju sve okolne čestice te ukoliko je njihov broj veći od nekog predodređenog onda se akceleracija povećava u smjeru suprotnom od njih a u protivnom prema njima. Osim toga, svakoj čestici se na početku pridoda određeni vektor akceleracije u smjeru y-osi kako bi čestice išle prema gore. Kako bi se odredile susjedne čestice koristi se struktura podataka k-stabla koja se svaki put nanovo izgrađuje. Iz izračunate akceleracije se potom određuju ostali parametri čestice poput vektora brzine i novog položaja čestice u prostoru.

Utjecaj broja čestica na brzinu izvođenja

Trenutno je u programu predefinirana konstanta koja određuje najveći broj čestica na ekranu. Kada nekoj čestici njeno preostalo vrijeme života padne na nula, ona se briše te se stvara određen broj novih čestica kako bi broj čestica na ekranu uvijek ostao isti. Naravno, veći broj čestica utječe na performanse sustava. Na vlastitom laptopu s grafičkom karticom Nvidia GeForce GTX 850M scena sa do 1 000 000 čestica izvršava se u šezdeset sličica u sekundi. Ako se broj čestica poveća, brzina izvođenja počinje padati. Primjerice sa 10 000 000 čestica brzina izvođenja padne na oko 15 sličica po sekundi.

Upute za korištenje

Program se pokreće izvršnom datotekom. Nakon pokretanja otvara se prozor u kojem se prikazuje scena. Pritiskom strelica lijevo ili desno moguće je rotirati kameru oko vatre. Za korektan rad programa potrebna je tekstura čestice koja se treba nalaziti u istom direktoriju kao i izvršna datoteka kao i pripadajući tekst programa za sjenčanje.

3.Literatura

[1] John Kessenich, Graham Sellers, and Dave Shreiner (2016). *The OpenGL Programming Guide 9th Edition*.

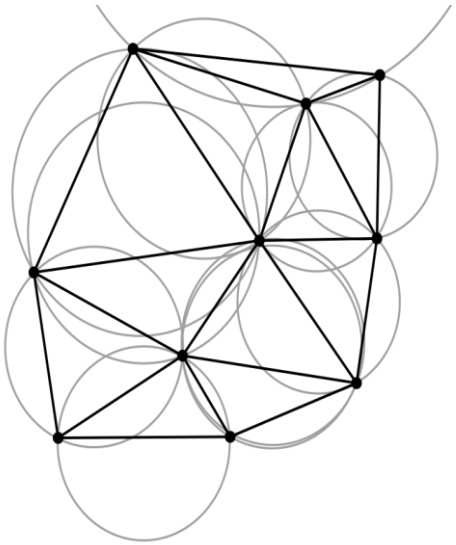
Deformacija 2D objekta

Studentski tim: Josip Sito

Nastavnik: Prof.dr.sc. Željka Mihajlović

Sadržaj

Opis razvijenog proizvoda



Slika 1. Primjer Delaunayeve

Cilj projekta je bio istražiti i implementirati deformaciju 2D objekta tako da korisnik sustava mijenjanjem vrijednosti različitih parametara mijenja stanje deformabilnog objekta. Projekt je izrađen u grafičkom alatu Unity, a izvorni kod je napisan u C# programskom jeziku. Deformacija objekta sastoji se od dvije faze: priprema geometrije i animiranje deformacije. Priprema geometrije se odvija pomoću Delaunayeve triangulacije, a animiranje deformacije se temelji na osnovi sustava masa i opruga i semi-implicitne Eulerove metode. Za potrebe testiranja i demonstracije sustava dodana je mogućnost jednostavne detekcije kolizije.

1. Tehničke značajke

1.1 Delaunayeva triangulacija

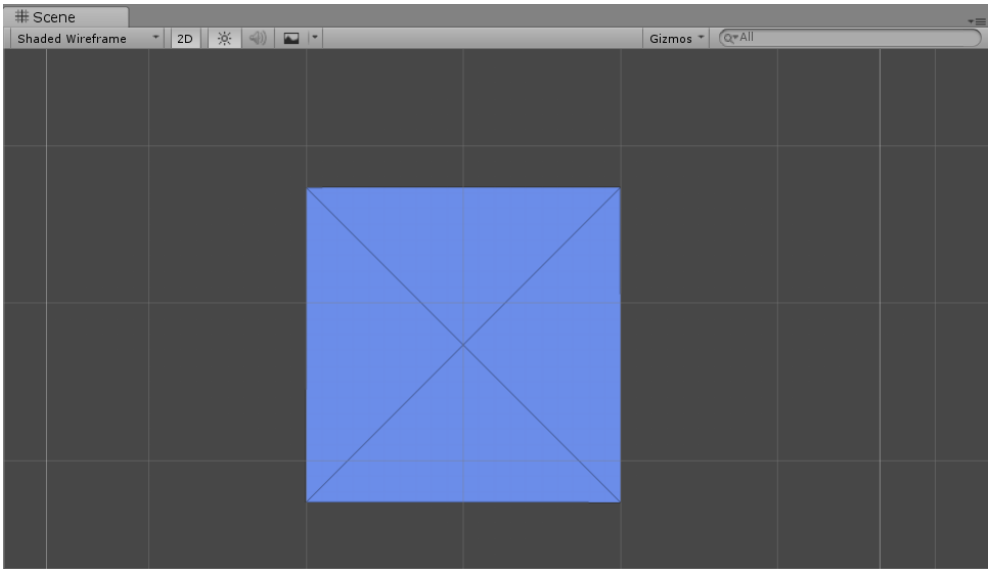
Za pripremu geometrije korištena je Delaunayeva triangulacija koja predstavlja jedinstvenu podjelu planarnog objekta na trokuta tako da se nijedna točka ne nalazi u opisanoj kružnici bilo kojeg trokuta u triangulaciji (Slika 1).

Kao implementacija Delaunay-ove triangulacije korišten je inkrementalan Bowyer-Watson algoritam koji funkcionira tako da dodaje jednu po jednu točku u triangulaciju te je pseudokod algoritma dan u nastavku.

```
 function BowyerWatson (pointList)
      // pointList is a set of coordinates defining the points to be triangulated
      triangulation := empty triangle mesh data structure
      add super-triangle to triangulation // must be large enough to completely
contain all the points in pointList
      for each point in pointList do // add all the points one at a time to the
triangulation
         badTriangles := empty set
         for each triangle in triangulation do // first find all the triangles that
are no longer valid due to the insertion
            if point is inside circumcircle of triangle
               add triangle to badTriangles
         polygon := empty set
         for each triangle in badTriangles do // find the boundary of the polygonal
hole
            for each edge in triangle do
               if edge is not shared by any other triangles in badTriangles
                  add edge to polygon
         for each triangle in badTriangles do // remove them from the data structure
            remove triangle from triangulation
         for each edge in polygon do // re-triangulate the polygonal hole
            newTri := form a triangle from edge to point
            add newTri to triangulation
      for each triangle in triangulation // done inserting points, now clean up
         if triangle contains a vertex from original super-triangle
            remove triangle from triangulation
      return triangulation
```

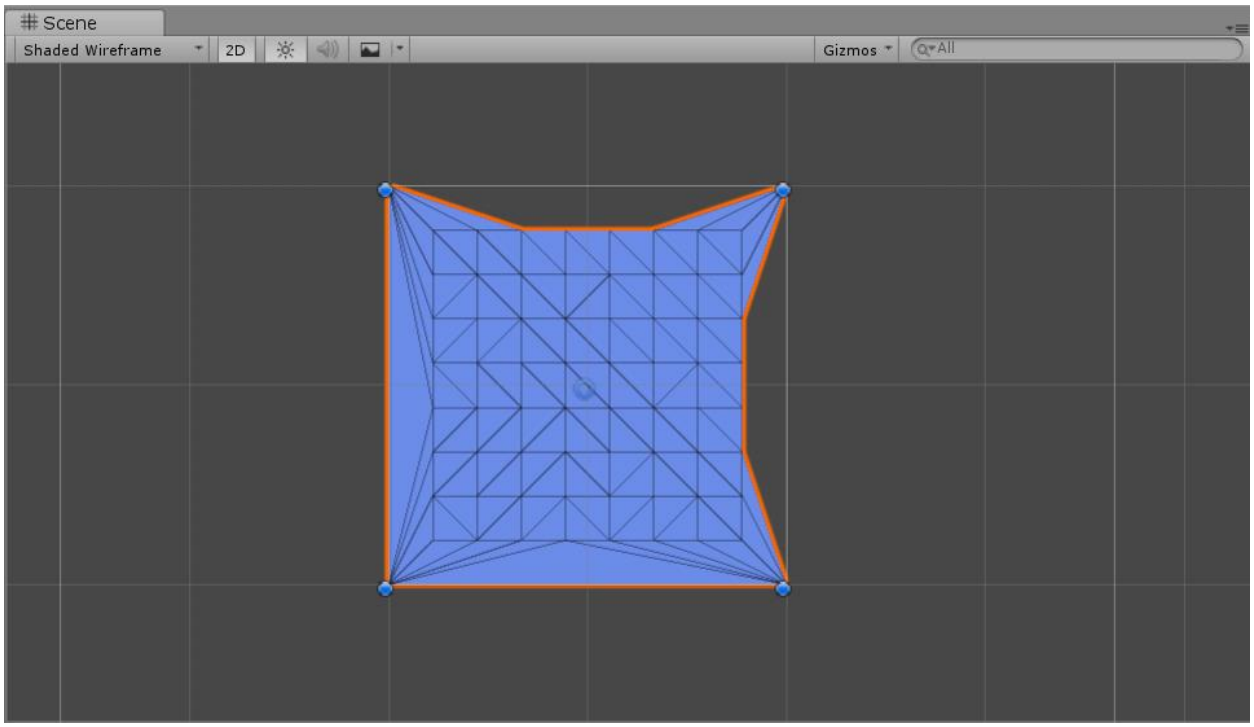
Izvor: https://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm

Sustav nudi mogućnost stvaranja objekta u obliku kvadra ili objekta pomoću teksture. Kvadar se stvara na način da se zadaju četiri inicijalne točke koje ga definiraju, a zatim se deterministički dodaju točke unutar njega ovisno o odabranoj razini detalja, odnosno za *n* razinu detalja uniformno će se dodati n^2 točaka unutar kvadra (slika 2 i slika 3). Na slici 3 zbog načina na koji su točke početne točke



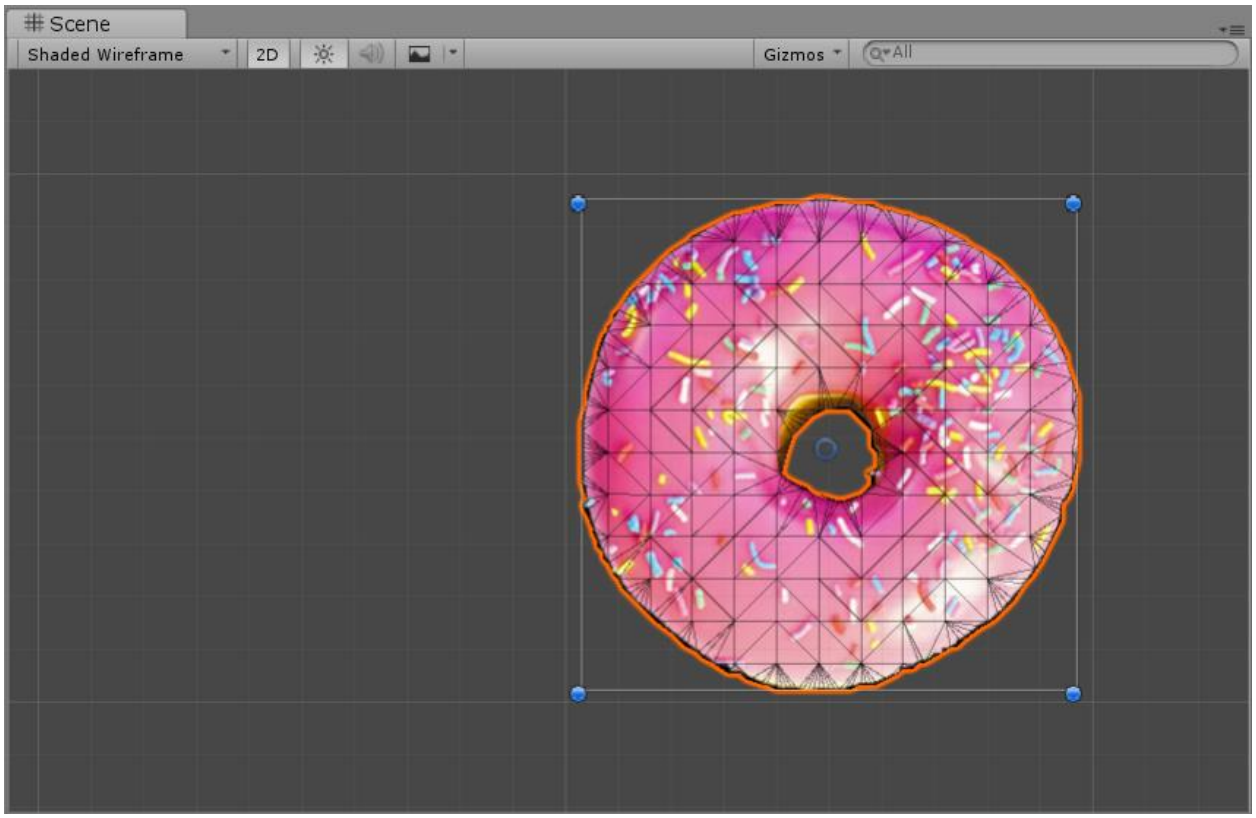
Slika 2. Kvadar sa razinom detalja 1

postavljene objekt koji je stvoren neće imati pravilni izgled kvadra.



Slika 3. Kvadar sa razinom detalja 8

Objekt kreiran pomoću teksture stvara se na način da se po slici prati piksel po piksel te ukoliko piksel nije crne boje, onda se doda nova točka na tom položaju. Kako se nebi stvorilo previše točaka zadaje se određeni cijeli broj *n* na temelju kojeg se preskače svakih *n* piksela. Od tekstura može se odabrati slika zeca, krafne ili armadila (slika 4).



Slika 4. Objekt stvoren pomoću teksture krafne

1.2 Sustav masa i opruga

Sustav masa i opruga jedan je od jednostavnijih fizičkih deformacijskih modela. Temeljen je na Hookeovom zakonu $F = -k\Delta x$, a u sustavu je svakoj točki dodana masa. Sila opruge u točki *i* računa se za svaku točku *j* s kojom je točka *i* povezana na temelju sljedećih formula:

$$f_{ij} = k_s(|x_{ij}| - l_{ij})\frac{x_{ij}}{|x_{ij}|} + k_d \cdot v_{ij}$$

k_s- konstanta opruge

x_{ij}- vektor između točke *i* i *j* $(x_j - x_i)$

l_{ij}- početna duljina između točke *i* i *j*

k_d- konstanta prigušenja

v_{ij}-razlika projeciranih brzina *i* i *j* točke na vektor duljine opruge $(v_j - v_i)$

Ukupna unutarnja sila u točki *i* računa se prema:

$$F_i^{int} = \sum_{i=j_1}^{j_q} f_{ij}$$

Ukupna sila koja djeluje na točku *i*:

$$F_i = F_i^{int} + F_i^{ext}$$

Gdje vanjska sila može biti:

$$F_i^{ext} = m \cdot g$$

1.3 Detekcija sudara

Nakon računanja sile provodi se detektiranje kolizije. Implementiran je jednostavan sustav detekcije kolizije gdje se za svaku točku računa sudara li se ona s objektom u obliku pravokutnika. Za detekciju kolizije pravokutnika s točkom korišten je sljedeći pseudokod:

```
function IsColliding (P)

        // points of rectangle are A, B, C, D
        AB := vector edge from A to B
        BC := vector edge from B to C
        AP := vector from A to P
        BP := vector from B to P
        return 0 <= dot(AB, AP) && dot(AB, AP) <= dot(AB, AB) && 0 <= dot(BC, BP) &&

        dot(BC, BP) <= dot(BC, BC)
```

Nakon što se uspostavi da je došlo do sudara točka se vrati na položaj prije sudara, dohvati se normala ruba pravokutnika kod kojeg je došlo do sudara te se brzina u smjeru suprotnom od dohvaćene normale postavi na nulu. Također sve dok je točka u koliziji sa pravokutnikom na nju djeluje sila trenja u smjeru suprotnom od smjera gibanja te se brzina točke smanjuje ovisno o parametrima za statičko i dinamičko trenje.

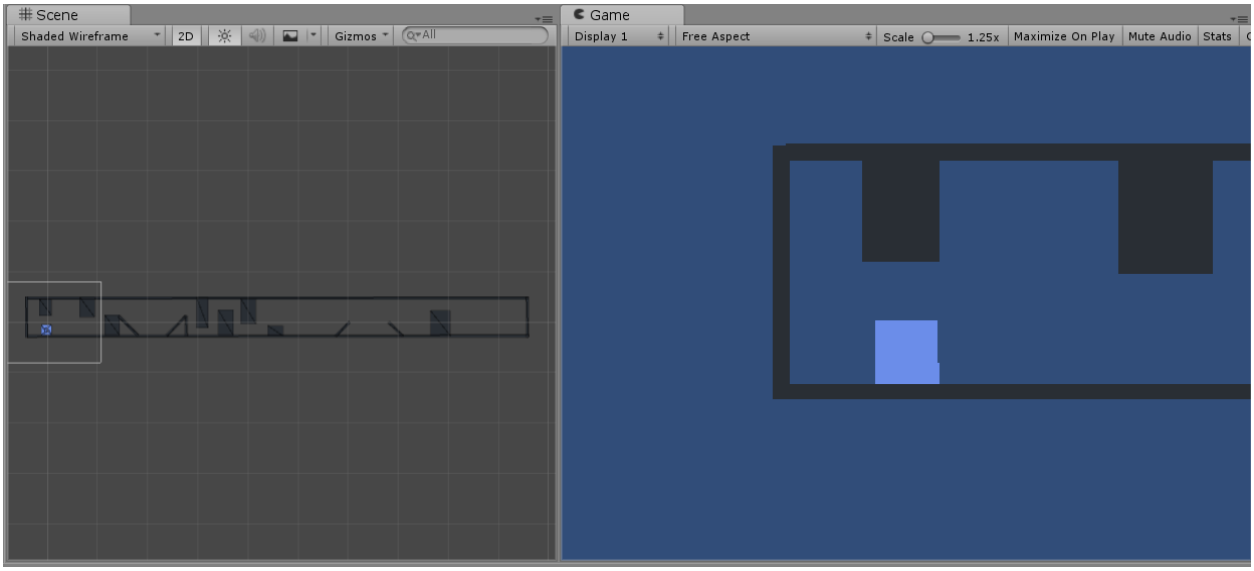
1.4 Vremenska integracija

Kako bi se deformacija ispravno prikazala potrebno ju je animirati uvođenjem vremenske dimenzije u sustav. Pri uvođenju vremenske integracije zbog svoje jednostavnosti i stabilnosti korištena je semi-implicitna Eulerova metoda te se pomoću nje izračuna pomak svih točaka za sljedeći vremenski trenutak.

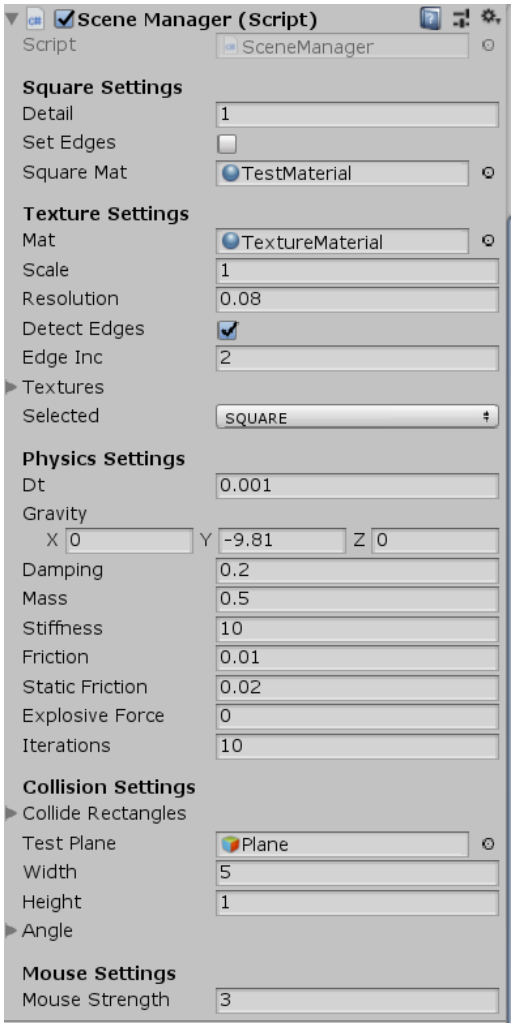
$$v(t + \Delta t) = v(t) + \Delta t\, F(v(t), x(t), t)$$
$$x(t + \Delta t) = x(t) + \Delta t\, v(t + \Delta t)$$

2. Upute za korištenje

Sustav sadrži dvije scene: *SampleScene* i *DemoScene*. *SampleScene* je inicijalna scena u kojoj se vršilo početno testiranje, a *DemoScene* je scena koja prije svega služi za demonstraciju sustava, ali se može koristiti i za testiranje (slika 5). U sceni se nalazi objekt *ObjectRenderer* koji sadrži skriptu *SceneManager* koja predstavlja glavnu skriptu za pokretanje sustava i preko koje se mijenjanjem i unošenjem različitih parametara mijenja stanje deformabilnog objekta. Postoje dva načina preko kojih se može stvoriti pravokutnik za kolizije. Prvi način je da se u *Editoru* doda položaj točke, visina, širina i kut te se na temelju tih parametara stvori pravokutnik, a drugi način je da se može kreirati novi *GameObject* u obliku pravokutnika sa tagom *CollisionObject.* Parametre koje je moguće promijeniti i unijeti vidljivi su u *Editoru* označavanjem *ObjectRenderer* objekta (slika 6), no potrebno je pažljivo mijenjati parametre pošto simulacija deformacije za određene vrijednosti neće biti stabilna.



Slika 5. Prikaz scene DemoScene



Slika 6. Prikaz parametara u Editoru

Square Settings

- Detail – razina detalja za kvadar, za cijeli broj *n* uniformno se unosi n^2 točaka unutar kvadra
- Set Edges – označavanjem se unose točke i na rubove kvadra
- Square Mat – materijal kvadra

Texture Settings

- Mat – materijal objekta stvorenog pomoću teksture
- Scale – razmjer objekta
- Resolution – određuje koliko će se točaka stvarati na teksturi, što je broj manji to će se više točaka stvoriti na teksturi
- Detect Edges – označavanjem će se detektirati rubovi na teksturi što znači da će model biti

precizniji izgledu teksturi, ali će biti potrebno više vremena jer se izrađuje puno točaka
- Edge Inc – broj na temelju kojega se preskaču pikseli kako bi dobili manje točaka i kako bi se ubrzala triangulacija, ali će time biti smanjena preciznost
- Textures – niz tekstura koje određuju izgled objekta
- Selected – odabrani objekt, može biti: DONUT, ARMADILLO, BUNNY, SQUARE

Physics Settings

- Dt – vremenski razmak, što je niži, to će simulacija biti preciznija, ali će zahtijevati više procesorske snage.
- Gravity – sila gravitacije prikazana kao vektor
- Damping – konstanta prigušenja
- Mass – ukupna masa objekta, masa pojedine točke će biti mass / broj_točaka
- Stiffness – konstanta opruge, parametar koji određuje čvrstoću tijela
- Static Friction – koeficijent statičkog trenja
- Friction – koeficijent dinamičkog trenja
- Explosive Force – parametar koji dodaje brzinu u pozitivnom smjeru osi y kada se stisne tipka *space*

Collision Settings

- Collide Rectangles – niz položaja sudarajućih pravokutnika
- Test Plane – pravokutni objekt koji inicijalno služi za testiranje i koristi se u sceni *SampleScene*
- Width – širina pravokutnika
- Height – visina pravokutnika
- Angle – niz kutova pravokutnika

Mouse Strength

- Mouse Strength – jačina sile kojom će objekt biti pokrenut u smjeru pokazivača miša kada se klikne na njega

3. Literatura

[1] Z. Huangfu, L. Yan, X. Liu, 20.11.2013, An Improved Mass-spring Model for Simulation of Soft Tissue Deformation, Department of Information Engineering, North China University of Water Resources and Electric

Power, China

[2] Y. Wang, Y. Xiong, K. Xu, K. Tan, G. Guo, A Mass-Spring Model for Surface Mesh Deformation Based on Shape Matching, School of Computer Science, National University of Defense Technology, Changsha, 410073, China, Center for Technique Education, PLA General Hospital, Beijing, 100853, China, Siječanj 2006.

[3] Nealen A., Müller M., Keiser R., Boxerman E., Carlson M., *Physically Based Deformable Models in Computer Graphics*, EuroGraphics 2005, Dublin, Republika Irska, 2005., stranice 1 – 6

[4] Torres L. M., *Fracture Modeling in Computer Graphics*, Universitat de Girona 2011., stranice 13-14

[5] Dinamika, Računalna grafika, Zagreb, FER, stranica 9

[6] Delaunay Triangulation, https://en.wikipedia.org/wiki/Delaunay_triangulation, 19.1.2019.

Magic LEAP ONE – Implementacija racunalne igre

Studentski tim: Filip Matijević

Nastavnik: prof. dr. sc. Željka Mihajlović

Sadržaj

Opis razvijenog proizvoda

Magic LEAP

Razvijeni proizvod može se opisati u dvije cjeline. Kako je cilj ovoga projekta bio proučiti Magic LEAP uređaj, njegove značajke i mogućnosti te na temelju njih razviti računalnu igru koja koristi neke od mogućnosti uređaja.

Uređaj Magic LEAP usporediv je sa već poznatim uređajem Microsoft HoloLens uz pokušaj eliminacije Hololensovih mana. Prva stavka jest težina i kompaktnost uređaja. Bazira se na naglavnim naočala povezanih sa računalnom jedinicom koja se drži prikačena oko struka ili na službeni nosač.

Uređaj posjeduje 6 DOF kontroler kojim se obavlja većina interakcija sa korisničkim sučelja unutar LuminOS operacijskog sustava. Dodatne interakcije obavljaju se korištenjem gesti ruku koje prepoznaje RGB kamera, ali taj način unosa nije izražen unutar operacijskog sustava nego je izložen razvijačima aplikacija kao mogućnost.

Mapiranje prostor vrši se pomoću tehnologije koja nije javno dostupna, ali tvorci Magic LEAPa nazivaju ju „photon field“ (hrv. Fotonsko polje).

Pokraj samog mapiranja prostora, razvijačima aplikacija dostupan je filtar ravnina kako bi se odabrale isključivo plohe određenih karakteristika (horizontalne, vertikalne, nagnute… itd). Također je dostupan filtar zidova, stropova i podova.

Kontroler Magic LEAP-a sa naočalama komunicira pomoću elektromagnetnih valova i bluetoothom. Kontroler odašilje tri ortogonalna signala koji se ponašaju kao trodimenzionalni koordinatni sustav. Inverznim operacijama nad jačinom ta tri signala, naglavni uređaj može pouzdano procijeniti poziciju kontrolera u odnosu na sebe.

Slika 5 - magic leap logo

Magic Realm

Aplikacija se bazira na logici igre kamen škare papir gdje se protivnici stvaraju oko igraća, a igrač ih pokušava pobijediti koristeći magiju iz jednog od tri čarobna stapica koji su mu na raspolaganju.

Implementirano je stanje aplikacije koje traži pod kako bi sve modele prilagodio na točnu visinu. Čarobni stapic se izmjenjuje tako da se lijevom rukom napravi simbol koji karakterizira određeni element.

Cilj igre je poraziti zadani broj neprijatelja.

U nastavku dokumentacije paralelno ću opisivati komponente Magic LEAP-a zajedno sa njihovom primjenom u računalnoj igri Magic Realm

Slika 6 - magical realm logo

Tehnicke znacajke

Slika 7 - magic leap uredaj

Pozicija uredaja u prostoru

Magic leap uređaj svoju referentnu točku u stvarnome svijetu uzima prilikom pokretanja (paljenje uređaja, a ne same aplikacije). Proces pronalaženja pravilne visine uređaja može potrajati nekoliko sekundi jer prilikom svakog gašenja, gubi mu se mapa stvarnog svijeta pa mora ponovo pokrenuti proces pronalaženja tla kako bi procijenio visinu na kojoj se naočale nalaze.

Prilikom Pokretanja aplikacije Magical Realm, potrebno je pristupiti stanju za postavljanje elemenata u prostor. Stanje radi na način da ishodiste prostora za igru postavi ispod igrača. Ono je označeno zelenim krugom. Ukoliko taj zeleni krug nije u potpunosti u ravnini s tlom, potrebno je ponovo pozvati izbornik dok uređaj ne pronađe dovoljno veliku ravnu plohu na kojoj se može odvijati igra.

Kontroler

Magic leap kontroler s naočalama komunicira ortogonalnim elektromagnetnim valovima i bluetoothom kako je to opisano u uvodu. Uredaj je sposoban očitati izmjenu rotacije i translacije kontrolera. Sam kontroler posjeduje plohu osjetljivu na dodir i četiri tipke na koje se mogu dodijeliti akcije.

Standardni način interakcije s proširenim svijetom Magic LEAPa je preko kontrolera koristeći bacanje zraka u smjeru kontrolera.

Kako je Magical Realm implementiran koristeći Unity3D pokretač, bilo je iznimno jednostavno dodijeliti skup čarobnih stapica objektu LEAP controllera kojeg je dll pomicao po sceni kako se kontroler pomicao u stvarnome svijetu.

U sklopu interakcije s okolinom, stražnji gumb na kontroleru sluzi za ispaljivanje magije iz stapica. Magic LEAP SDK omogućuje pretplatu na akcije koje izvodi pritisak gumba sa silom potrebnom da se akcija izvede. To je ostvareno jer kontroler ima nekoliko stotina razine osjetljivosti pritiska.

Slika 8 - magic leap kontroler

Rekonstrukcija prostora

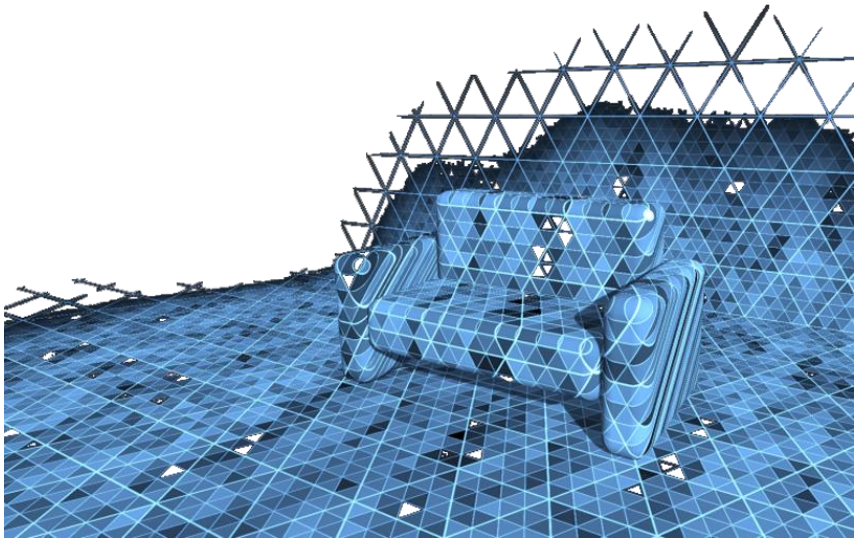
Uredaj ima mogućnost rekonstrukcije prostora u nekoliko razina preciznosti. Povećanjem razine preciznosti performanse aplikacija nažalost opadaju ali se prostor iznimno vjerno preslikava u virtualni.

Prilikom razvojne faze Magical Realm aplikacije korišteno je potpuno mapiranje prostora ali se pokazalo previse zahtjevno za ovu primjenu jer mi je jedina bitna stavka bila postavljanje objekata na ravnu plohu te da ti isti objekti nisu u vidljivoj koliziji s velikim predmetima. To je ostvareno ručnim postavljanjem ploha i definiranjem lokacija na kojem se neprijatelj može pojaviti

Prilikom skeniranja, senzori mogu kompenzirati za mjesta nad kojima se prostor nije dobro očitao ili mjesta na kojima je vidljiv sum. Takva područja uredaj samostalno izravna ukoliko razvijač aplikacije to zatraži.

Također se mapiranje prostora može ograničiti na fiksan radijus oko korisnika. Iako LuminOS drži čitavu mapu svega sto je LEAP registrirao tijekom svog perioda dok je bio upaljen, Unity3D aplikacija može zatražiti tek dio tog prostora kako bi se smanjio broj nepotrebnih poligona u prostoru.

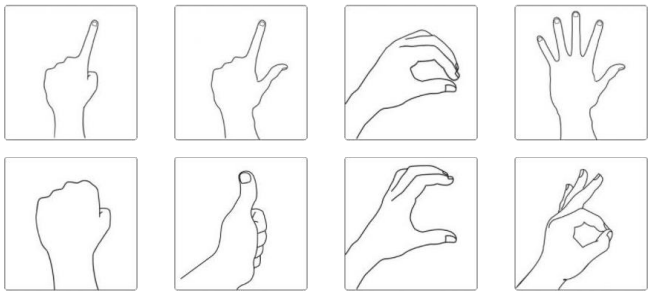
Kako je obnavljanje mesha u svakome frameu skupa operacija, dobro je zaustaviti taj proces nakon potpuno skenirane sobe ako se ovakav scenarij treba implementirati u igru. Mana ovoga je izmjena okoline usred igranja (pomicanje namještaja i slično)



Slika 9 - interpretacija mapiranja prostora

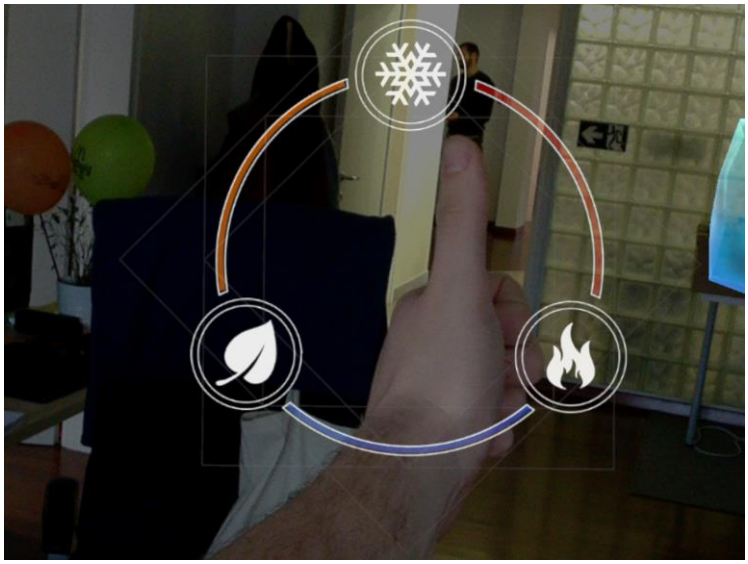
Prepoznavanje Gesti

Prednja kamera Magic LEAP sposobna je prepoznati obje ruke korisnika i gestu koja je izvedena. Postoji ukupno 8 gesti koja je izvediva na svakoj od ruku, a Magic LEAP ima mogućnost razlikovanja lijeve i desne ruke sto razvijaču aplikacije daje mogućnosti implementacije 64 različite akcije ako se ne koristi kontroler.



Slika 10 - dostupne geste

Magical Realm aplikacija koristi prepoznavanje gesti kako bi omogućila korisniku odabir željenog čarobnog stapica. Stapic se bira tako da se lijevom rukom napravi simbol koji je povezan za odabranim element.



Slika 11 - pogled kroz leap (prepoznata gesta)

5. Upute za korištenje

Nakon sto upogonite uredaj Magic LEAP, potrebno je odabrati aplikaciju Magical Realm. Nakon otvaranja aplikacije, u blizini korisnika se može naci kamen s natpisom „START“.

Prije početka igre, dobro je uvjeriti se da je prostor dobro mapiran. To se radi pritiskom na „home“ tipku na kontroleru kojom se otvara stanje za postavljane igre u prostor. Ako se zeleni krug nalazi ispod igrača, pomicanjem prsta po površini za praćenje postavite rotaciju terena. Teren je prikazan crvenim krugovima koji govore gdje će se neprijatelji stvoriti prilikom pokretanja igre.

Kada je sve postavljeno, korisnik treba ponovo pritisnuti home tipku za izlazak is stanja za postavljanje te vrhom čarobnog stapica treba dotaknuti kamen.

Daljnje upute opisane su unutar aplikacije, a sastoje se od poruka korisniku koje govore koliko neprijatelja mora poraziti kako bi završio razinu.

6. Literatura

[1] https://docs.unity3d.com/Manual/index.html

[2] https://www.magicleap.com/creator

Interaktivna vizualizacija tokova koristeći web tehnologije, LIC i metoda čestica
Tehnička dokumentacija

Verzija 1.0

Studentski tim: David Emanuel Lukšić

Nastavnik: Prof. dr. sc. Željka Mihajlović

Sadržaj

1. Opis razvijenog proizvoda

Cilj projekta je ostvariti python biblioteku za interaktivni prikaz 2D vektorskih polja (tokova). Biblioteka je razvijena kao dodatak na postojeću biblioteku „Bokeh“, a koristi suvremenu WebGL tehnologiju kao grafički pogon u internet pregledniku. Ostvareni načini prikaza su „ Particle“ (čestice) i LIC (Line integral convolution). Korištenje biblioteke se sastoji u kopiranju nekoliko datoteka u lokalni repozitorij i uključivanju (import) određenih modela u vlastiti kod. Detaljnije u nastavku.

2. Tehničke značajke

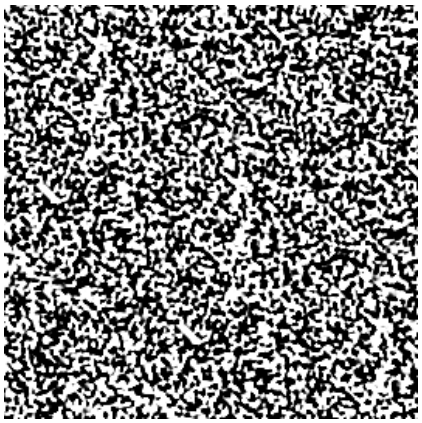
2.1 Metoda „Line Integral Convolution“, LIC

Vrlo jednostavna metoda, uz nekoliko implementacijskih detalja. Sam algoritam je vrlo jednostavan:

- počnemo s teksturom šuma
- teksturu šuma „zamuljamo“ u smjeru polja
- obojamo zamuljanu teksturu bojom kako bi dodali informaciju
- animacija smjera

2.1.1 Tekstura šuma

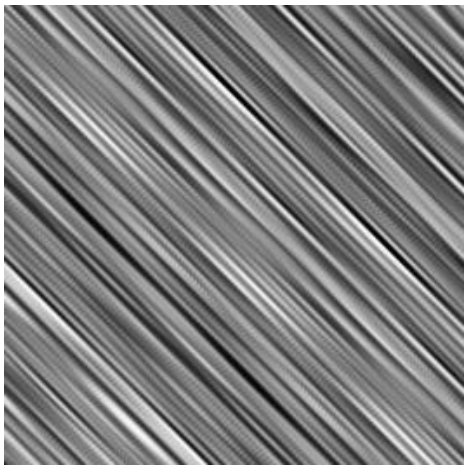
Tekstura šuma direktno se računa korištenjem sjenčara. Originalni algoritam „Simplex 3D noise“ napisao je Ian McEwan, a može se pronaći na linku (https://github.com/ashima/webgl-noise/blob/master/src/noise3D.glsl). Odabrao sam ovaj šum jer implementira šum u ovisnosti o 3D koordinati, što je korisno za prikaz u tri dimenzije.



Slika 12 Tekstura šuma

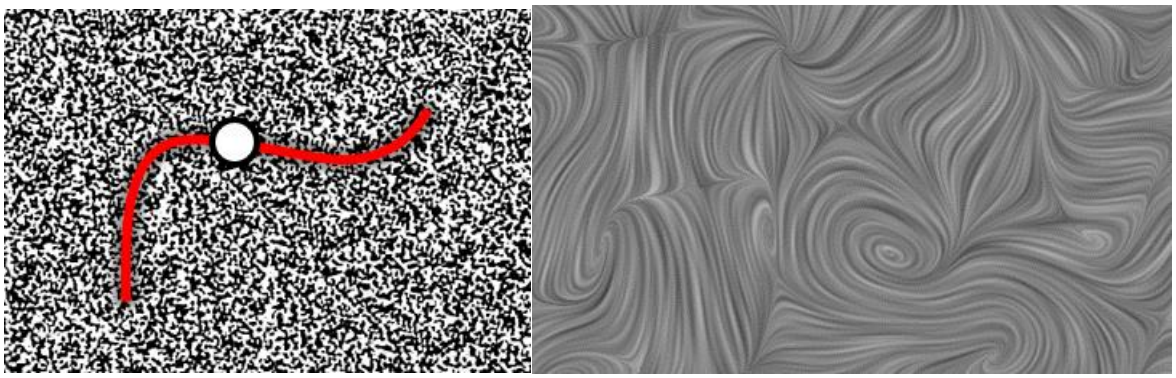
2.1.2 Linijski integral

Kako bismo dobili intuiciju o tome što dobivamo muljanjem šuma, pogledajmo sliku:



Slika 13 Zamuljana tekstura šuma

Vidimo da muljanjem u smjeru polja, možemo reći da korelacija vrijednosti piksela u smjeru polja se povećava, dok korelacija okomita na smjer polja ostaje nepromijenjena, odnosno nula. Time smo dobili silnice. Zatim želimo to napraviti u svim smjerovima, ne samo u jednom smjeru. To radimo linijskim integralom, odnosno jednostavnije rečeno, za svaki piksel u sjenčru uzimamo prosjek svih piksela koji se nalaze na strujnici istog.

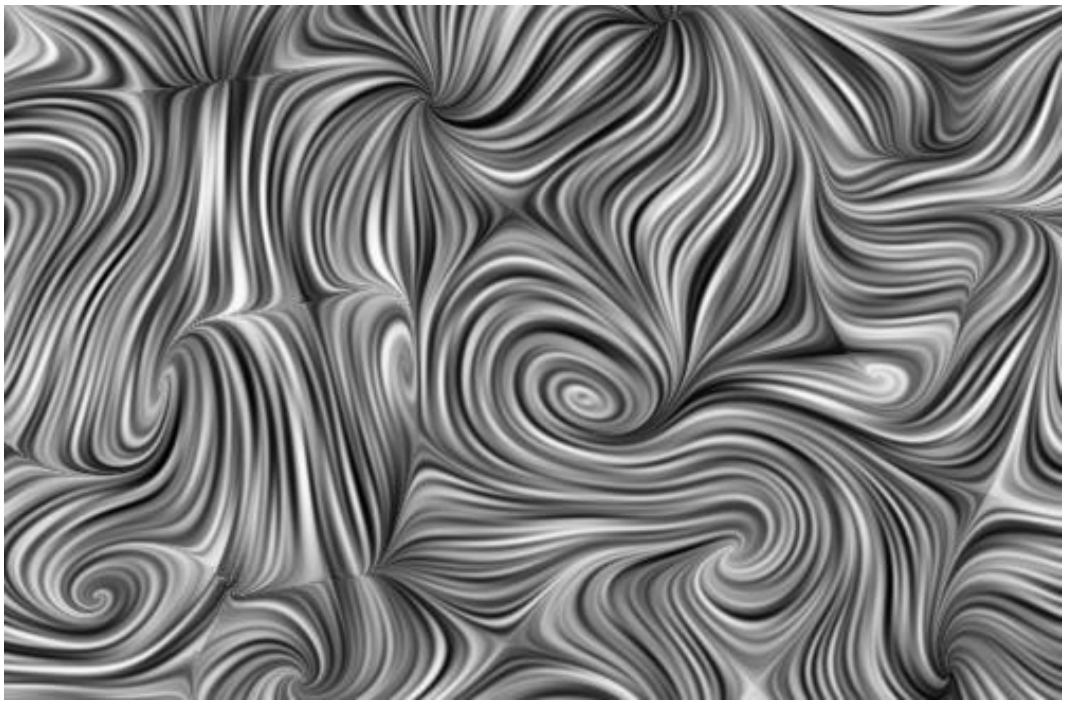


Slika 14 Ilustracija linijskog integrala i rezultat LIC operacije

Sama strujnica računa se jednostavnom Eulerovom integracijom 20 piksela naprijed i nazad od zadanog.

2.1.3 Poboljšanje LIC metode

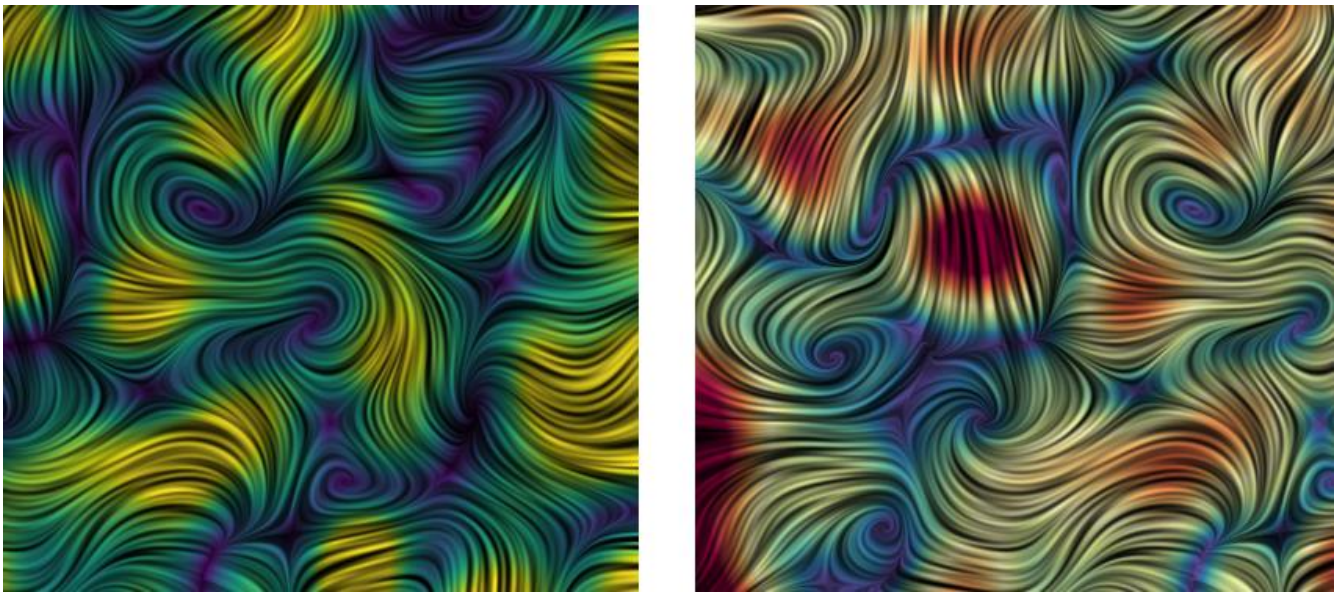
Kako bismo poboljšali kvalitetu novonastale teksture, možemo pojačati kontrast, korištenjem glsl-ove „smoothstep“ funkcije. Time rastegnemo vrijednosti sa intervala [0.4, 0.6] natrag na [0, 1]. No, još bolji kontrast možemo dobiti ako napravimo još jedan LIC prolaz, tako da novonastalu teksturu tretiramo kao ulaz u drugi LIC prolaz:



Slika 15 Poboljšana LIC metoda

2.1.4 Bojanje pseudo bojom

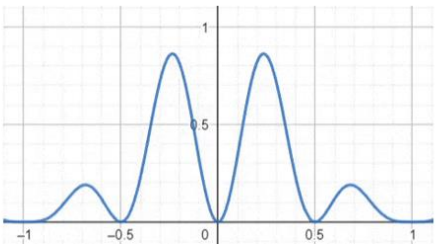
Pogledamo li sliku koju daje LIC metoda, primijetit ćemo da na njoj ne vidimo niti smjer niti iznos samog vektorskog polja. Ovu informaciju možemo dodati korištenjem boje. Najčešće se odabire iznos vektorskog polja u svrhu bojanja:



Slika 16 Obojana LIC metoda

2.1.5 Animacija smjera

LIC metoda ima nedostatak da može prikazati samo nagib, ali ne i smjer polja u statičnom okruženju. No, ukoliko možemo prikazati LIC metodu na nekom ekranu, postoji mogućnost animiranja dodavanjem jezgrenih funkcija (kernel). Takva se zatim funkcija koristi kako bismo izračunali težinski prosjek piksela na strujnici, a ne obični. Konkretno, u implementiranoj biblioteci, koristi se sljedeća funkcija:




```
float kernel(float p){
    float a = sin(PI*(kernelcoef*p + time));
    float b = cos(PI_2*p);
    return a*a*b*b;
}
```

Slika 17 Jezgrena funkcija korištena kako bi se dobila animacija toka

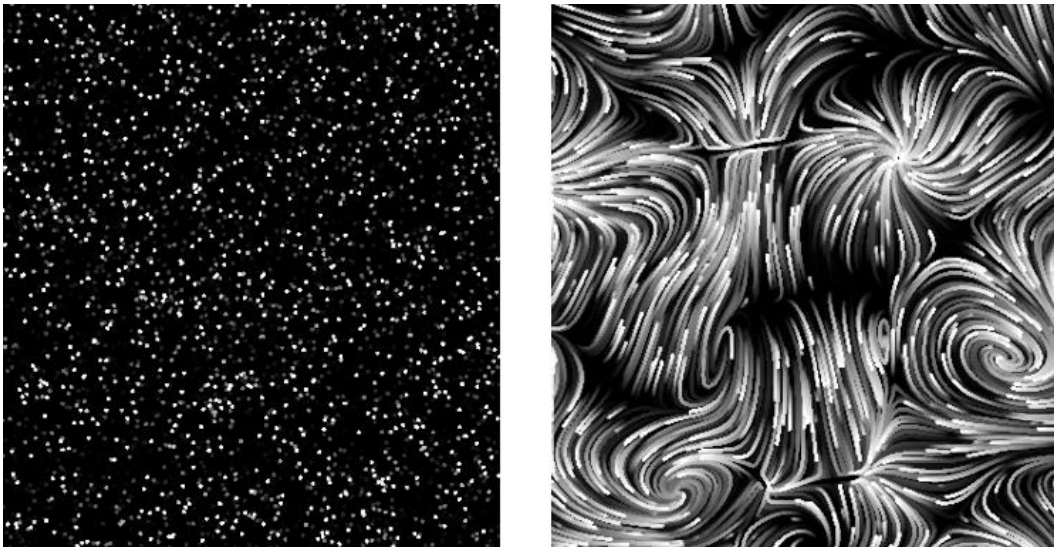
2.2 Metoda čestica, „Particles“

Prikaz toka česticama klasična je metoda prikaza, koja koristi advekciju čestica koje ostavljaju trag. Same čestice su implementirane koristeći sjenčar vrhova i piksela. Informacija o česticama zapisana je u kvadratnoj teksturi. Svaki piksel te teksture označava jednu česticu:

- crveni kanal - x položaj
- zeleni kanal - y položaj
- plavi kanal - preostali život čestice

2.2.1 Iscrtavanje čestica

Kada želimo iscrtati čestice, u spremnik vrhova upisujemo 3D vektore čije XY koordinate označavaju gdje se u teksturi nalazi informacija o zadanoj čestici. Zatim sjenčar vrhova kopira koordinatu čestice iz zadane teksture te proslijedi sjenčaru piksela na iscrtavanje točki. Za praćenje tragova koristi se zaseban spremnik, koji prije iscrtavanja novih položaja čestica treba izblijediti (množeći sve s nekom konstantom manjom od 1):



Slika 18 Slučajno inicijalizirane čestice i spremnik tragova

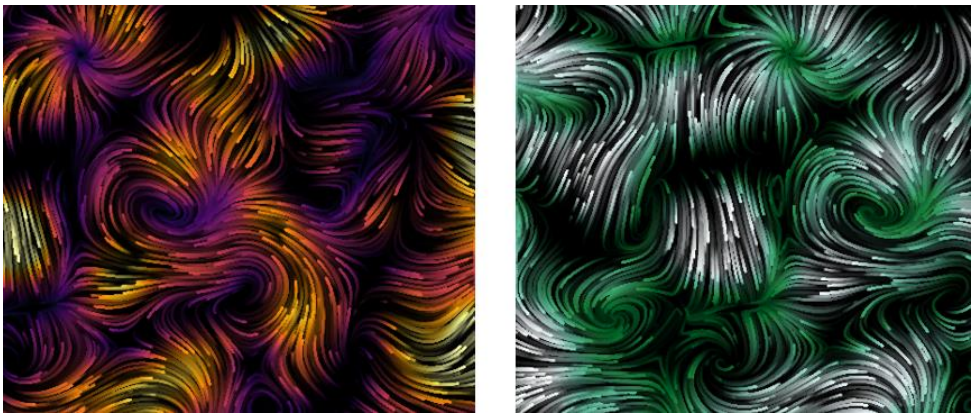
2.2.2 Advekcija čestica

Kod LIC metode, Eulerova metoda advekcije strujnice bila nam je dovoljna, jer udaljenost koju je trebalo izračunati je bila oko 20 piksela. Kod čestica, linearna pogreška nam je prevelika jer čestice proputuju velike udaljenosti. Zato se u implementaciji koristi „midpoint“ metoda koja ima kvadratnu ovisnost pogreške o koraku. Algoritam je relativno jednostavan:

- napraviti polu-korak u smjeru trenutnog polja
- pročitati smjer polja na tom mjestu
- napraviti puni korak u pročitanom smjeru iz originalnog položaja

2.2.3 Bojanje čestica

Rezultirajući spremnik tragova možemo obojati, baš kao i LIC teksturu:

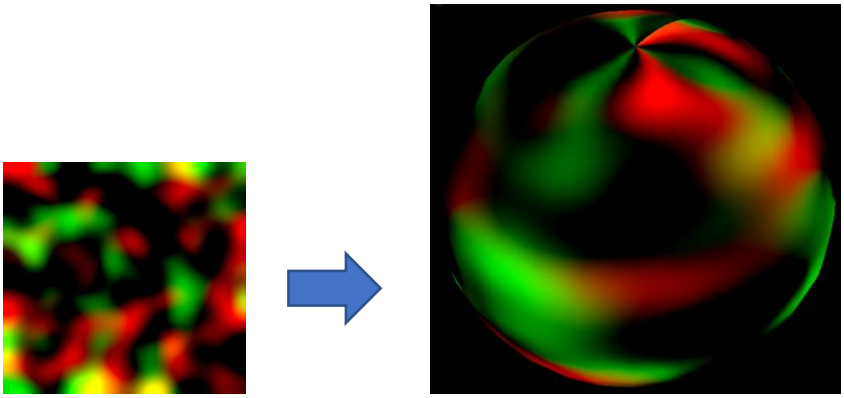


Slika 19 Čestice obojane iznosom vektorskog polja

3. Proširenje u tri dimenzije

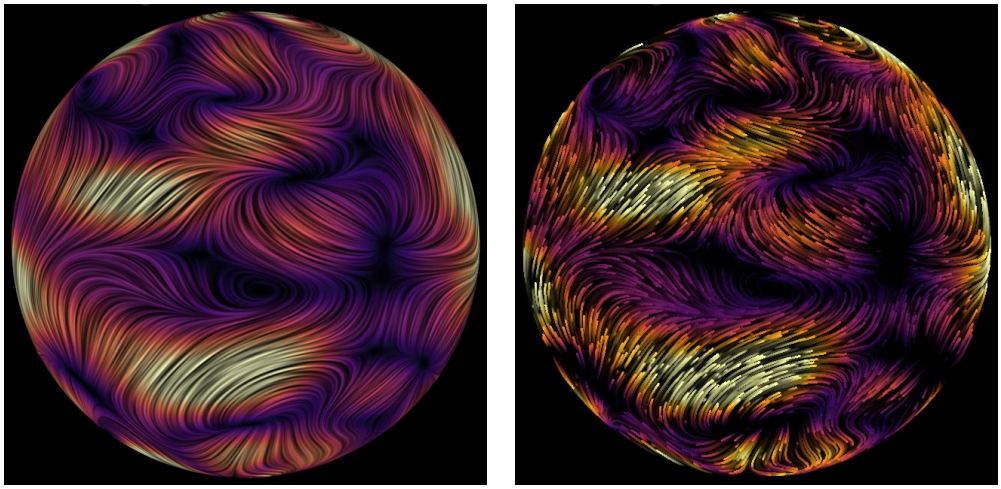
Obje metode, čestice i LIC, su originalno dvodimenzionalne. No, ukoliko želimo prikazati tok po površini nekog trodimenzionalnog predmeta, možemo koristiti sljedeći trik:

- projiciramo uv-mapiranu teksturu vektorskog polja na predmet



Slika 20 Sfera iscrtana pomoću projekcije vektorskog polja

- vizualiziramo tok kao da je dvodimenzionalan



Slika 21 LIC i čestice animirane nad projiciranim vektorskim poljem

4. Upute za korištenje

4.1 Korištenje biblioteke u python okruženju

Biblioteka je napisana tako da se na vrlo jednostavan način može koristiti u raznim python okruženjima. Ovdje ćemo pokazati kako koristiti biblioteku u Jupyter Notebook okruženju.

Cijela biblioteka može se dohvatiti u obliku .zip datoteke. Ona sadržava sljedeće:

- public/build/flowvis.min.js
- python/
  - __init__.py
  - flowvis.json
  - flowvis.py
  - flowvis.ts
  - loader.py

Prije korištenja treba imati instalirane python biblioteke:

- numpy
- bokeh
- jinja2
- jupyter

Ovu .zip datoteku treba raspakirati negdje. Zatim, u njoj (ili negdje blizu nje) pokrenuti jupyter notebook razvojno okruženje naredbom „jupyter notebook .“ Napraviti novu bilježnicu sa sljedećim kodom:

```
import os, sys, numpy as np

from bokeh.io import output_notebook
from bokeh.plotting import show
from bokeh.models import ColumnDataSource

sys.path.append(os.path.realpath('python'))
from flowvis import Quiver2DModel
import loader

flow = np.ones((200, 200, 2))

# Turn data into Bokeh Data Source
flow_field_source = ColumnDataSource(data=dict(
    dx=flow[..., 0],
    dy=flow[..., 1]
))

quiver = Quiver2DModel(
    show_param_gui=True,
    width=512, height=512,
    flow_field_width=200,
    flow_field_height=200,
    flow_field_source=flow_field_source,
    col_x='dx', col_y='dy')

output_notebook()
show(quiver)
```

4.2 Pokretanje lokalnog razvojnog poslužitelja

U korijenskoj datoteci projekta pokrenuti:

- npm install
- npm start

Ovo će pokrenuti razvojni poslužitelj na „http://localhost:8080“. Ukoliko pogledate što se nalazi na toj adresi pomoću Internet preglednika, vidjet ćete rezultat pokretanja testnog koda unutar „src/index.ts“ datoteke.

Kako biste kompilirali paket za distribuciju nakon mijenjanja izvornog koda, potrebno je pokrenuti „build.sh“ skriptu unutar nekog „bash“ okruženja.

5. Literatura

[1] Cabral, Brian, and Leith Casey Leedom. "Imaging vector fields using line integral convolution." *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. ACM, 1993.

[2] Dirksen, Jos. *Learning Three. js: the JavaScript 3D library for WebGL*. Packt Publishing Ltd, 2013.

[3] Bokeh Development Team. "Bokeh: Python library for interactive visualization." (2014).

[4] https://blog.mapbox.com/how-i-built-a-wind-map-with-webgl-b63022b5537f