

Univerzitet u Beogradu – Elektrotehnički fakultet

**Računarska grafika 2
Seminarski rad
Tehnike za crtanje vodenih površi**

Đukić Jovan, 3016/2017

Apstrakt

U ovom radu biće predstavljene tehnike za crtanje vodenih površi, problemi sa kojima se susrećemo prilikom crtanja kao i tehnike koje koristimo za prevazilaženje ovih problema. Problemi će biti izloženi u dve kategorije, a one su: simulacija kretanja vodene površi i optički efekti do kojih dolazi prilikom interakcije vode sa zracima svetla. Od ovih efekata biće pokrivene refleksija i refrakcija, kao dva najbitnija efekta koji doprinose realističnom izgledu vode. Pored ovih, takođe će biti pokrivena kaustika kao treći efekat koji doprinosi realistično prikazu vodene površi, doduše ne u tolikoj meri kao prethodna dva. Na kraju ovog rada biće dat kratak opis jedne moguće implementacije vodene površi koja pokriva sve navedene probleme.

Sadržaj

1. Uvod.....	1
2. Problem.....	2
2.1. Simulacija.....	2
2.2. Optika.....	2
2.2.1. Refleksija i refrakcija.....	2
2.2.2. Fresnel-ov efekat.....	3
2.2.3. Kaustika.....	4
3. Pregled postojećih rešenja.....	5
3.1. Simulacija.....	5
3.1.1. 2D jednačina talasa.....	5
3.1.2. Aproksimacija pomoću sume sinusa.....	8
3.1.3. Gerstner-ovi talasi.....	10
3.1.4. Perlin-ov šum.....	11
3.1.5. Brze Fouier-ove transformacije (FFT).....	12
3.2. Optika.....	13
3.2.1. Refleksija i refrakcija.....	14
3.2.2. Fresnel-ov efekat.....	16
3.2.3. Kaustika.....	17
4. Implementacija.....	19
4.1. Refleksija i refrakcija.....	19
4.2. Simulacija.....	22
4.3. Fresnel-ov efekat.....	24
4.4. Spekularno osvetljenje.....	27
4.5. Kaustika.....	28
5. Zaključak.....	33
Literatura.....	34
Slike.....	35
Listinzi.....	36
Dodatak A - Kod <i>vertex shader-a</i> za implementaciju vodene površi.....	37
Dodatak B - Kod <i>fragment shader-a</i> za implementaciju vodene površi.....	38
Dodatak C - Kod <i>vertex shader-a</i> za implementaciju površi ispod vodene površi.....	40
Dodatak D - Kod <i>fragment shader-a</i> za implementaciju površi ispod vodene površi.....	41

1. Uvod

Kako je rasla moć grafičkih procesora, tako je rasla potreba za što realnijim prikazivanjem stvarnosti na računarima. Prikazivanje prirodnih fenomena igra važnu ulogu kod prikazivanja stvarnosti. Međutim, i dalje nismo u mogućnosti da sve fizičke fenomene prikažemo onakve kakvi oni jesu u prirodi. Od svih fizičkih pojava koje se prikazuju, vodene površi zahtevaju najviše napora i truda. Rezultat ovih napora se može videti kako u video igricama tako i u filmovima.

Uprkos današnjem hardveru, priliko crtanja vodenih površi ne možemo prikazati vodu onaku kakva je u prirodi. Dva glavna razloga za to su samo kretanje vode, koje može biti dosta nepredvidivo i zavisi od raznoraznih faktora kao što su atmosferske prilike, dubina itd. i interakcija vode sa svetлом u koju spadaju efekti kao što su refleksija, refrakcija, kaustika itd. Danas je skoro nemoguće predstaviti vodu poštujući njen ponasanje u prirodi i postići dovoljnu brzinu crtanja tako da posmatrač ne primeti kašnjenja ili seckanja prilikom prikazivanja. Zato se dosta često pribegava aproksimacijama koje, iako ponekad mogu da budu jako grube, daju prilično zadovoljavajuće rezultate. U ovom radu ću dati pregled načina na koji se može pristupiti ovim problemima koji uzimaju u obzir i realnost i brzinu crtanja.

Kao što je navedeno u [Fle2007], probleme sa kojima se susrećemo prilikom crtanja vode možemo podeliti u dve kategorije koje odgovaraju gore navedenim problemima:

1. simulacija vodenih površi i njihovog kretanja
2. poboljšanje realizma same površi dodavanjem interakcije sa svetlosnim zracima

Ova dva problem mogu, u nekoj meri, da se rešavaju nezavisno. Kažem u nekoj meri jer se ponekad dešava da produkti rešavanja prvog problema pomažu prilikom rešavanja drugog problema. U ovom radu čitaocu će biti predviđeni neki od načina na koje se mogu prevazići ovi problemi. Neki od njih su jednostavniji i uključuju dosta grubih aproksimacija, ali ipak daju zadovoljavajuće rezultate. Neki od njih su komplikovani i oslanjaju se na matematički zahtevne metode. Koji od ovih ćemo koristiti zavisi od naših potreba za realnošću i brzinom.

U drugom poglavlju ovog rada biće detaljno prestavljeni problemi sa kojima se susrećemo prilikom crtanja vodenih površi. Kao što je navedeno, ovi problemi će biti podeljeni u dve kategorije. U trećem poglavlju biće opisana neka od postojećih rešenja kojima se prevazilaze gore navedeni problemi. Četvrtog poglavlje će sadržati opis programa, implementiranog pomoću OpenGL grafičke biblioteke, koji pokazuje jedno jednostavno rešenje navedenih problema. Poslednje, peto, poglavlje predstavlja zaključak u kome će biti dat kratak osvrt na sam sadržaj rada kao i pravci u kojima treba nastaviti dalje istraživanje ove teme.

2. Problem

U ovom poglavlju biće predstavljeni problemi na koje nailazimo prilikom crtanja vodenih površina. Kao što je navedeno u uvodu ta dva problema su simulacija kretanja vodene površine i interakcija vode sa svetlosnim zracima.

2.1. Simulacija

Kretanje fluida je opisano Navier-Stokes jednačinama[WikiA]. Kada primenimo ove jednačine na volumetrijsku predstavu vode dobijemo fizički tačnu predstavu. Međutim, ove jednačine zahtevaju ogroman napor računara i primenjuju se samo u simulacijama za naučne potrebe. Zato se veoma često koriste aproksimacije. Izbor samog metoda simulacije vodene površi zavisi od toga koju vodenu površ predstavljamo. Način ponašanja jezera, reka i okeana nije isti i za svaki postoji dijapazon jednostavnijih jednačine kojima možemo opisati kretanje vode.

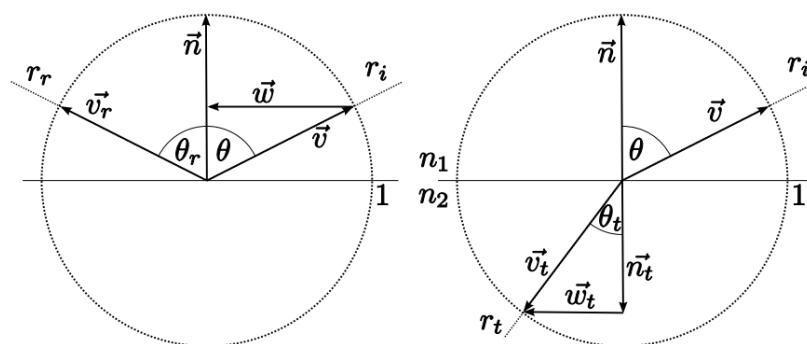
Pored samog izbora načina simulacije postoji i problem modela same vodene površi. Najčešće je to mreža trouglova. Međutim, tu se postavlja pitanje koliki broj trouglova ta mreža treba da sadrži. Naravno, što je data mreža gušća to je i sama vodena površ realističnija ali kao što je navedeno u [Tur2014] takva mreža zahteva ogromne količine memorije na grafičkoj kartici jer je potrebno za svaku tačku date mreže čuvati njene koordinate, a to nije uvek moguće.

2.2. Optika

Postoji mnogo pojava koje se dešavaju usled interakcije vode sa svetlosnim zracima i sve one doprinose realističnom izgledu vode. U ovom odeljku će biti spomenuti samo oni efekti koji imaju najveći uticaj na realnost vodene površi.

2.2.1. Refleksija i refrakcija

Da bi uopšte mogli da krenemo u razmatranje interakcija vodene površine sa svetlosni zracima potrebno je da znamo kako one funkcionišu. Dve najbitnije pojave su refleksija i refrakcija. Za njihov proračun potreban nam je jedinični vektor normale u svakoj tački vodene površi. Ovo generalno nije problem jer ukoliko su nam poznate koordinate tačaka vodene površi lako možemo nekom matematičkom metodom doći do vektora normala vodene površi u datim tačkama. Način delovanja refleksije i refrakcije dat je na slici 1:



Slika 1. Refleksija (levo) i refrakcija (desno)[Fle2007]

Najpre ćemo analizirati refleksiju, odnosno levi deo slike 1. Neka su vektori $\vec{v}, \vec{n}, \vec{v}_r, \vec{w}$ sa levog dela slike 1 jedinični vektori i to inverzni vektor svetlosnog zraka, vektor normale, vektor reflektovanog zraka i pomoći vektor, respektivno. Na osnovu činjenice da su updati ugao θ i reflektujući ugao θ_r jednaki možemo izračunati reflektovani vektor na sledeći način:

$$\begin{aligned}\cos(\theta) &= \vec{n} \cdot \vec{v} \\ \vec{w} &= \cos(\theta) \cdot \vec{n} - \vec{v} \\ \vec{v}_r &= \vec{v} + 2 \cdot \vec{w}\end{aligned}$$

Kao što se može videte, proračun koji zahteva refleksija je jednostavan. Sada ćemo analizirati refrakciju, odnosno desni deo slike 1. Za razliku od refleksije, refrakcija zahteva dva dodatna podatka a oni su indeks prelamanja vode i indeks prelamanja medijuma sa kojim naša vodena površ dolazi u kontakt. Najčešće je to vazduh. Neka su vektori $\vec{v}, \vec{n}, \vec{v}_t, \vec{w}_t$ sa desnog dela slike 1. jedinični vektori i to inverzni vektor svetla, vektor normale, vektor refraktovanog zraka i pomoći vektor, respektivno. Odnos upadnog ugla θ i ugla refrakcije θ_t možemo dobiti iz Snell-ovog zakona[WikiB]:

$$\frac{\sin(\theta)}{\sin(\theta_t)} = \frac{n_2}{n_1}$$

Dalje, koristeći Snell-ov zakon možemo dobiti jedinični vektor refraktovanog zraka.

$$\begin{aligned}\cos^2(\theta_t) &= 1 - \sin^2(\theta_t) = 1 - \left(\frac{n_1}{n_2}\right)^2 \sin^2(\theta) = 1 - \left(\frac{n_1}{n_2}\right)^2 (1 - \cos^2(\theta)) \\ \cos(\theta_t) &= \sqrt{\cos^2(\theta_t)} \\ \vec{n}_t &= -\vec{n} \cos(\theta_t) \\ \vec{w}_t &= \frac{\vec{w}}{\|\vec{w}\|} \sin(\theta_t) = \vec{w} \frac{\sin(\theta_t)}{\sin(\theta)} = \vec{w} \frac{n_1}{n_2} \\ \vec{v}_t &= \frac{n_1}{n_2} \vec{w} + \vec{n}_t\end{aligned}$$

Kao i za refleksiju, proračun koji zahteva refrakcija je jednostavan.

2.2.2. Fresnel-ov efekat

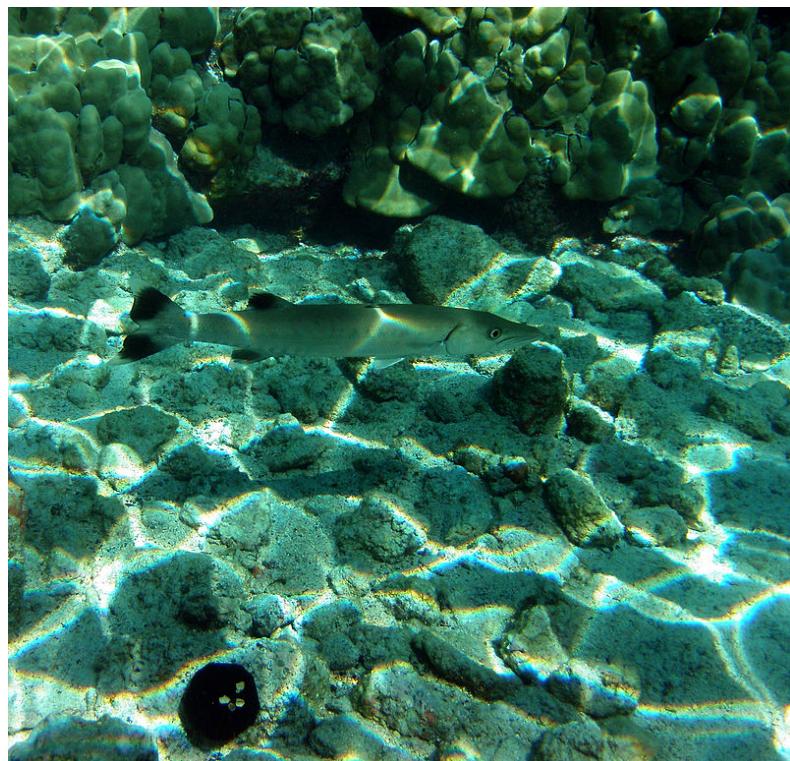
Još jedna jako bitna pojava koja se javlja prilikom interakcije vode sa zracima jeste Fresnel-ov efekat koji nam govori koji deo upadanog zraka se reflektovao a koji deo se refraktovao. Pored toga što zavisi od upadnog ugla i samih indeksa refrakcije, takođe zavisi od toga da li je svetlost s-polarizovana ili p-polarizovana[WikiC]. Jednačine po kojima se dobijaju refleksioni koeficijenti za s-polarizovanu i p-polarizovanu svetlost su sledeće:

$$\begin{aligned}R_s &= \left(\frac{\sin(\theta_t - \theta)}{\sin(\theta_t + \theta)} \right)^2 = \left(\frac{n_1 \cdot \cos(\theta) - n_2 \cdot \cos(\theta_t)}{n_1 \cdot \cos(\theta) + n_2 \cdot \cos(\theta_t)} \right)^2 \\ R_p &= \left(\frac{\tan(\theta_t - \theta)}{\tan(\theta_t + \theta)} \right)^2 = \left(\frac{n_1 \cdot \cos(\theta_t) - n_2 \cdot \cos(\theta)}{n_1 \cdot \cos(\theta_t) + n_2 \cdot \cos(\theta)} \right)^2\end{aligned}$$

gde su R_s i R_p refleksioni koeficijenti za s-polarizovanu i p-polarizovanu svetlost, respektivno. Refraktujući (transmisioni) koeficijenti se dobija iz jednakosti $T_s = 1 - R_s$ i $T_p = 1 - R_p$, gde su T_s i T_p refraktujući (transmisioni) koeficijenti za s-polarizovanu i p-polarizovanu svetlost, respektivno.

2.2.3. Kaustika

Kaustika je pojava koja se javlja usled refrakcije. Neki zraci se refraktuju tako da nakon prelaska u vodu dolazi do njihove interferencije u određenim tačkama površi koja se nalazi ispod vodene površi i dolazi do neproporcionalnog osvetljenja tog jednog dela površi u odnosu na ostale. Na slici 2 je prikazan efekat kaustike.



Slika 2. Kaustika

3. Pregled postojećih rešenja

U ovom poglavlju biće dat pregled postojećih rešenja problema navedenih u prethodnom poglavlju. Kao i u prethodnom poglavlju, rešenja će biti podeljena u dve grupe. Prva grupa se bavi problemima simulacije kretanja vodene površine, a druga grupa se bavi problemima koji nastaju usled interkacije vodene površi sa svetlosnim zracima.

3.1. Simulacija

U ovom delu će biti predstavljene tehnike koje se koriste za simularanje kretanja vodene površi. Svaka od ovih tehnika podrazumeva da našu vodenu površ modelujemo kao mrežu trouglova određene gustine. Od same gustine trouglova nam zavisi koliko će biti nivo detalja naše vodene površi. Pored računanja samih pozicija čvorova mreže trouglova, moram izračunati i normale u svakom čvoru koje će nam kasnije trebati za implementaciju optičkih efekata.

3.1.1. 2D jednačina talasa

Detaljan opis ove metode dat je u [Zel2004]. Kao što je navedeno u ovom dokumentu ova metoda u maloj meri smanjuje nivo realizma ali obezbeđuje veću brzinu. Pored toga sama metoda je dosta jednostavna za implementaciju. **2D jednačina talasa** glasi:

$$\frac{\partial^2 y}{\partial t^2} = c^2 \cdot \left(\frac{\partial^2 y}{\partial x^2} + \frac{\partial^2 y}{\partial z^2} \right)$$

Data jednačina menja y koordinatu tako što ona osciluje gore dole, gde c predstavlja brzinu datog talasa. Gledajući datu jednačinu intuitivno možemo zaključiti da se y koordinata menja u skladu sa **menjanje nagibom** vodene površi. Međutim da bi dobili pozicije čvorova potrebno je naći integral leve strane jednačine po promenljivoj t kao i naći izvode na desnoj strani jednačine. Zato ćemo ovu jednačinu rešiti numeričkom metodom.

Najpre ćemo, umesto integracije leve strane, naći jednačinu koja opisuje kretanje određenog čvora. Prvo moramo uvesti pretpostavku da se vreme menja u jednakim koracima, odnosno da za taj korak h važi sledeće $h = t_1 - t_0 = t_2 - t_1 = t_{n+1} - t_n$. Dalje, ukoliko imamo poziciju u trenutku $t=0$ ($p(t_0)$), poziciju u trenutku $t=1$ ($p(t_1)$) i ubrzanje u trenutku $t=1$ ($a(t_1)$), možemo doći do pozicije u trenutku $t=2$ ($p(t_2)$) na sledeći način:

$$\begin{aligned} v(t_1) &= \frac{p(t_1) - p(t_0)}{t_1 - t_0} \\ p(t_2) &= p(t_1) + v(t_1) \cdot (t_2 - t_1) + \frac{1}{2} \cdot a(t_1) \cdot (t_2 - t_1)^2 \\ h &= (t_2 - t_1) = (t_1 - t_0) \\ p(t_2) &= 2 \cdot p(t_1) - p(t_0) + \frac{1}{2} \cdot a(t_1) \cdot h^2 \end{aligned}$$

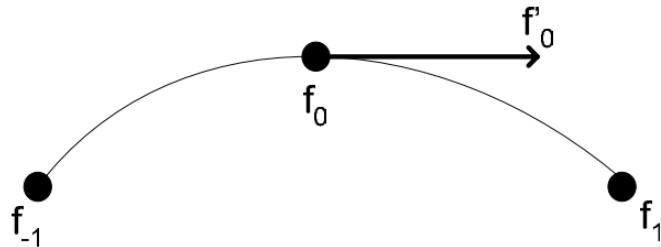
Međutim, zbog greške koja se ispoljava prilikom primenjivanja ove metode dolazi do podrhtavanja vodene površi. Ovo možemo rešiti tako što ćemo smanjiti brzinu pomoću nekog konstantog faktora. Označićemo ga sa α . Nakon toga data jednačina izgleda ovako:

$$\begin{aligned} p(t_2) &= p(t_1) + \alpha \cdot v(t_1) \cdot h + \frac{1}{2} \cdot a(t_1) \cdot h^2 \\ p(t_2) &= p(t_1) + \alpha \cdot (p(t_1) - p(t_0)) + \frac{1}{2} \cdot a(t_1) \cdot h^2 \\ p(t_2) &= (1 + \alpha) \cdot p(t_1) - \alpha \cdot p(t_0) + \frac{1}{2} \cdot a(t_1) \cdot h^2 \end{aligned}$$

Sledeći korak je rešavanja parcijalnih izvoda na desnoj strani jednačine. Najpre ćemo našu vodenu površinu aproksimirati kubnom krivom. Na ovaj način možemo izjednačiti naše druge parcijalne izvode sa drugi izvodom datog kubne krive. Data kubnu krivu ćemo posmatrati u 2D zbog pojednostavljenog problema. Funkcija koja opisuje ovaku krivu kao i njeni izvodi izgledaju ovako [WikiD]:

$$\begin{aligned} f(x) &= c_0 \cdot x^3 + c_1 \cdot x^2 + c_2 \cdot x + c_3 \\ f'(x) &= 3 \cdot c_0 \cdot x^2 + 2 \cdot c_1 \cdot x + c_2 \\ f''(x) &= 3 \cdot c_0 \cdot x + 2 \cdot c_1 \end{aligned}$$

Datu krivu možemo prikazati na sledeći način:



Slika 3. Kubna kriva [Zel2004]

Na slici 3 data je kriva kroz tri tačke koje su na odstojanju dužine 1. Nama je potreban drugi izvod u tački nula (na slici 3 je to tačka u sredini). Njega dobijamo na sledeći način:

$$\begin{aligned} f(-1) &= -c_0 + c_1 - c_2 + c_3 = f_{-1} \\ f(0) &= c_3 \\ f(1) &= c_0 + c_1 + c_2 + c_3 = f_1 \\ f''(0) &= c_2 \\ f_1 + f_{-1} &= 2 \cdot c_1 + 2 \cdot c_3 \\ c_1 &= \frac{1}{2} \cdot (f_1 + f_{-1}) - c_3 \\ f''(0) &= c_1 = \frac{1}{2} \cdot (f_1 + f_{-1}) - c_3 = \frac{1}{2} \cdot (f_1 + f_{-1}) - f_0 \end{aligned}$$

Na osnovu ovoga možemo naći parcijalne izvode na desnoj strani jednačine 2D talasa. Nakon toga dobićemo kompletну desnu stranu i ona glasi ovako:

$$c^2 \cdot \left(\frac{\partial^2 y}{\partial x^2} + \frac{\partial^2 y}{\partial z^2} \right) = c^2 \cdot ((y_{-1,0} + y_{1,0} - 2 \cdot y_{0,0}) + (y_{0,-1} + y_{0,1} - 2 \cdot y_{0,0}))$$

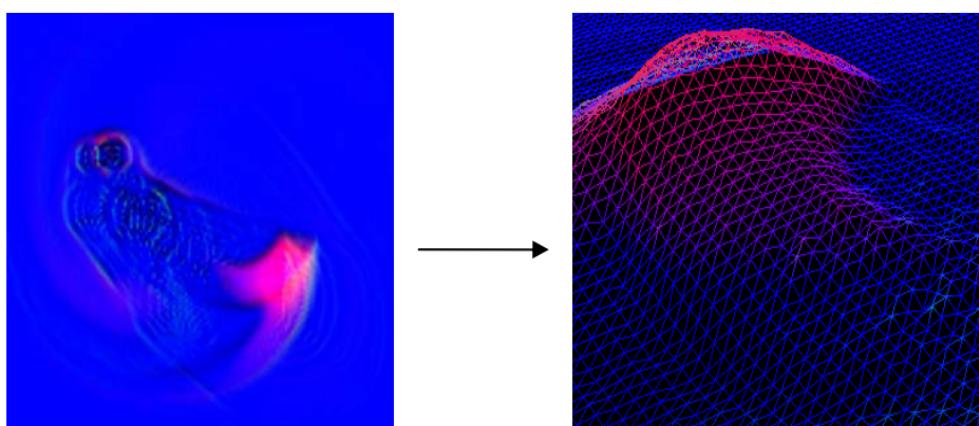
$$c^2 \cdot \left(\frac{\partial^2 y}{\partial x^2} + \frac{\partial^2 y}{\partial z^2} \right) = c^2 \cdot (y_{-1,0} + y_{1,0} + y_{0,-1} + y_{0,1} - 4 \cdot y_{0,0})$$

Ovo ćemo implementirati korišćenjem tekstura. Teksturu ćemo koristiti za smeštanje visina čvorova mreže i prilikom crtanja vodene površine pristupićemo dajoj teksturi sa (x, z) koordinatama čvorova. Nakon tога ćemo pročitanu vrednost iz teksture koristiti kao y kordinatu datog čvora. Naravno, pre svakog crtanja ćemo ažurirati datu teksturu koristeći gore navede jednačine. Prilikom ažuriranja koristićemo tehniku koja se zove "ping-pong" teksture (engl. *ping-pong textures*). Odnosno, nećemo imati jednu već dve tekture. Jedna će čuvati visine iz prethodne iteracije dok ćemo u drugu smeštati nove visine čvorova. Ukoliko sa p označimo (x, z) koordinate čvora i ukoliko sa w i h označimo širinu i visinu naše teksture, kod na HLSL (engl. *High Level Shading Language*) kod bi izgledao kao na listingu 1:

```
dx = 1 / w;
dy = 1 / h;
heightP = sampleTexture(currentHeightTexture, p);
heightPL = sampleTexture(currentHeightTexture, p + offset(-dx, 0));
heightPR = sampleTexture(currentHeightTexture, p + offset(dx, 0));
heightPU = sampleTexture(currentHeightTexture, p + offset(0, dy));
heightPD = sampleTexture(currentHeightTexture, p + offset(dx - dy));
previousHeightP = sampleTexture(previousHeightTexture, p);
accelartion = c * c * (heightPL + heightPR + heightPU + heightPD - 4 * heightP);
newHeightP = 2 * heightP - previousHeightP + 0.5 * acceleration * dt * dt;
```

Listing 1. Kod za generisanje mape visina pomoću 2D jednačine talasa

Naravno treba voditi računa o tome da dve koršćene tekture budu odgovarajućeg tipa, odnosno da celije datih tekstura mogu da smeste brojive sa odgovarajućom preciznošću. Rezultat ovog koda dat je na slici 4:



Slika 4. 2D jednačina talasa [Zel2004]

Naravno pored računanja samih pozicije čvorova mreže trouglova potrebno je i sračunati vektore normala u datim čvorovima. Najlakši način da to odradimo jeste da nađemo dva vektora koji počinju u tekućem čvoru i da njihovim vektorskim proizvodom

dobijemo vektor normale. Ova dva vektora možemo dobiti iz pozicija tekućeg čvora i nekih susednih čvorova. Naravno moramo voditi računa o tome da vektor normale mora da bude jedinični vektor. Kod na HLSL (engl. *High Level Shading Language*) koji računa vektor normale dat je u listingu 2:

```
vectorRight = pRight - p;
vectorUp = pUp - p;
normalP = normalize(cross(vectorRight, vectorUp));
```

Listing 2. Kod za računanje vektora normale

Naravno za postizanje veće preciznost možemo izračunati sve četiri normale koristeći sva četiri suseda i nakon toga interpolacijom dobiti konačnu normalu. Velika pogodnost ovog pristupa je ta što i visine i normale možemo čuvati u jednoj teksturi RGBA tipa gde bi na primer alfa kanal bio odvojen za visinu našeg čvora a crveni, zeleni i plavi kanal za x , y i z komponentu vektora normale u datom čvoru.

3.1.2. Aproksimacija pomoću sume sinusa

Kao što je navedeno u [FerKir2004], da bi aproksimirali kretanje naše vodene površine kao sumu sinusa treba nam određeni skup parametara koji će opisati ponašanje naših talasa. Oni su sledeći:

1. talasna dužina L koja je povezana sa frekvencijom preko formule $w = \frac{2\pi}{L}$
2. amplituda A , odnosno visina od ravni vode do vrha talasa
3. brzina S kojom se talas kreće, međutim za našu implementaciju izrazićemo brzinu kao faznu konstantu $\varphi = \frac{2\pi S t}{L}$
4. pravac D izražen kao horizontalan vektor normalan na čelo talasa

Stanje svakog talasa izrazićemo funkcijom koja zavisi od horizontalnog položaja, odnosno kordinata x i z i vremena t i ona glasi:

$$W_i(x, z, t) = A_i \cdot \sin(D_i \cdot (x, z) \cdot w_i + t \cdot \varphi_i)$$

A konačna visina čvora mreže je izraženo kao suma svih talasa i data je sledećom formulom:

$$H(x, z, t) = \sum (A_i \cdot \sin(D_i \cdot (x, z) \cdot w_i + t \cdot \varphi_i))$$

Parametre samih talasa je najlakše generisati pomoću nekog pseudoslučajnog generatora, naravno vodeći računa o tome da postoje određena ograničenja. Tokom simulacije ćemo kontinuirano izbacivati talas kada dođe do kraja naše mreže i ponovo ga ubacivati sa drugim skupom parametara. Ova tehnika je pogodna zato što nam je ponašanje talasa opisano konkretno funkcijom pa izračunavanje normala u čvorovima mreže nije komplikovano. Za izračunavanje normale u čvoru mreže trouglova potrebni su vektor binormale i vektor tangente u datom čvoru, koji se dobijaju kao parcijalni izvodi naše funkcije u x i z pravcu. Pozicija nekog čvora sa kordinatama (x, z) u horizontalnoj ravni se u određenom trenutku može izraziti kao:

$$P(x, z, t) = (x, H(x, z, t), z)$$

Kao što je navedeno vektor binormale u čvoru se dobija kao parcijalni izvod u x pravcu i on glasi:

$$B(x, z) = \left(\frac{\partial x}{\partial x}, \frac{\partial}{\partial x} H(x, z, t), \frac{\partial z}{\partial x} \right) = \left(1, \frac{\partial}{\partial x} \cdot H(x, z, t), 0 \right)$$

Slično ovome vektor tangente u čvoru se dobija kao parcijalni izvod u z pravcu i on glasi:

$$T(x, z) = \left(\frac{\partial x}{\partial z}, \frac{\partial}{\partial z} H(x, z, t), \frac{\partial z}{\partial z} \right) = \left(0, \frac{\partial}{\partial z} H(x, y, t), 1 \right)$$

Vektor normale u čvoru dobija kao vektorski proizvod vektora binormale i vektora tangente istog i on glasi:

$$N(x, z) = T(x, z) \times B(x, z) = \left(-\frac{\partial}{\partial x} H(x, z, t), 1, -\frac{\partial}{\partial z} H(x, z, t) \right)$$

Naravno nakon ovoga je potrebno normalizovati vektor normale ukoliko on nije jedinični vektor. Dodatna pogodnost ovog metoda je to što je nalaženje parcijalnih izvoda jednostavno zato što je izvod sume jednak sumi izvoda i to je navedeno u sledećoj formuli:

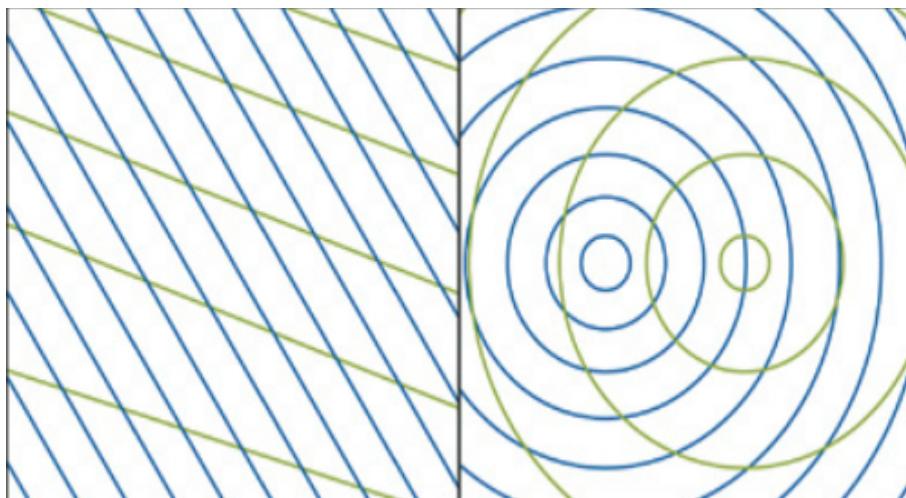
$$\frac{\partial}{\partial x} \cdot H(x, z, t) = \sum \frac{\partial}{\partial x} \cdot W_i(x, z, t) = \sum w_i \cdot D_i \cdot x \cdot A_i \cdot \cos(D_i \cdot (x, z) \cdot w_i + t \cdot \varphi_i)$$

Međutim, postoji jedna primedba navedena u [FerKir2004] a ona je da ovi talasi nemaju oštretre vrhove kao što to imaju pravi talasi. Ovaj problem se lako rešava tako što sinus pomerimo tako da on bude nenegativan i dat sinus podignema na neki stepen k . Sama funkcija talasa i njen parcijalni izvod u x pravcu izgledaju ovako:

$$W_i(x, z, t) = 2 \cdot A_i \cdot \left(\frac{\sin(D_i \cdot (x, z) \cdot w_i + t \cdot \varphi_i) + 1}{2} \right)^k$$

$$\frac{\partial}{\partial x} W_i(x, z, t) = k \cdot D_i \cdot x \cdot w_i \cdot A_i \cdot \left(\frac{\sin(D_i \cdot (x, z) \cdot w_i + t \cdot \varphi_i) + 1}{2} \right)^{k-1} \cdot \cos(D_i \cdot (x, z) \cdot w_i + t \cdot \varphi_i)$$

Poslednja stvar koju treba odraditi jeste odabir između usmerenih ili kružnih talasa koji su prikazani na slici 5:



Slika 5. Usmereni (levo) i kružni (desno) talasi [FerKir2004]

Za usmerene talase vektor D_i je konstantan za svaki čvor. Za kružne talase ovaj vektor se računa svaki put i tako da on ide od centra kružnog talasa C_i do čvora naše mreže i on glasi:

$$D_i(x, z) = \frac{(x, z) - C_i}{\|(x, z) - C_i\|}$$

3.1.3. Gerstner-ovi talasi

Ukoliko želimo da naši talasi imaju oštije vrhove možemo koristiti Gerstner-ovu jednačinu talasa. Kao što je navedeno u [FerKir2004] ona glasi:

$$P(x, z, t) = \begin{cases} x + \sum (Q_i \cdot A_i \cdot D_i \cdot x \cdot \cos(w_i \cdot D_i \cdot (x, z) + \varphi_i \cdot t)) \\ \quad \sum (A_i \cdot \sin(w_i \cdot D_i \cdot (x, z) + \varphi_i \cdot t)) \\ z + \sum (Q_i \cdot A_i \cdot D_i \cdot z \cdot \cos(w_i \cdot D_i \cdot (x, z) + \varphi_i \cdot t)) \end{cases}$$

U ovim jednačinama Q_i je parametar koji kontroliše strminu talasa. Za svaki talas vrednost parametra Q_i jednaka 0 daje daje uobičajne okrugle sinusne talase kakve smo videli u prethodnom poglavlju i vrednost parametra $Q_i = \frac{1}{w_i \cdot A_i}$ daje oštar vrh. Veće vrednosti ovog parametra treba izbegavati jer će to prouzrokovati formiranje petlji na vrhovima talasa što ne želimo. Iako se data funkcija razlikuje od one iz prethodnog poglavlja i dalje je lako naći njene parcijalne izvod i dobiti vektor binormale, vektor tangente i vektor normale u nekom čvoru. Oni su dati jednačinama:

$$\begin{aligned} B(x, z) &= \begin{cases} 1 - \sum (Q_i \cdot D_i \cdot x^2 \cdot w_i \cdot A_i \cdot \sin(w_i \cdot D_i \cdot P + \varphi_i \cdot t)) \\ \quad \sum (D_i \cdot x \cdot w_i \cdot A_i \cdot \cos(w_i \cdot D_i \cdot P + \varphi_i \cdot t)) \\ - \sum (Q_i \cdot D_i \cdot x \cdot D_i \cdot z \cdot w_i \cdot A_i \cdot \sin(w_i \cdot D_i \cdot P + \varphi_i \cdot t)) \end{cases} \\ T(x, z) &= \begin{cases} - \sum (Q_i \cdot D_i \cdot x \cdot D_i \cdot z \cdot w_i \cdot A_i \cdot \sin(w_i \cdot D_i \cdot P + \varphi_i \cdot t)) \\ \quad \sum (D_i \cdot z \cdot w_i \cdot A_i \cdot \cos(w_i \cdot D_i \cdot P + \varphi_i \cdot t)) \\ 1 - \sum (Q_i \cdot D_i \cdot z^2 \cdot w_i \cdot A_i \cdot \sin(w_i \cdot D_i \cdot P + \varphi_i \cdot t))S \end{cases} \\ N(x, z) &= \begin{cases} - \sum (D_i \cdot x \cdot w_i \cdot A_i \cdot \cos(w_i \cdot D_i \cdot P + \varphi_i \cdot t)) \\ 1 - \sum (Q_i \cdot w_i \cdot A_i \cdot \sin(w_i \cdot D_i \cdot P + \varphi_i \cdot t)) \\ - \sum (D_i \cdot z \cdot w_i \cdot A_i \cdot \cos(w_i \cdot D_i \cdot P + \varphi_i \cdot t)) \end{cases} \end{aligned}$$

Ove jednačine nisu "lepe" kao one iz prethodnog poglavlja ali su i dalje jednostavne za izračunavanje. Uslov za izbegavanje petlji na vrhovima talasa se može videti iz y komponente vektora normale. Sve dok je suma manja od 1 petlje se neće formirati na vrhovima talasa, otuda i uslov naveden u prethodnom pasusu. Kao i u prethodnom modelu, ukoliko vektori nisu jedinični potrebno ih je normalizovati.

Odabir parametara zavisi od toga kakvi su nam talasi potrebni, odnosno kakvi su atmosferski uslovi. Na primer za talasnu dužinu možemo izabrati neku minimalnu vrednost koja odgovara sunčanom i mirnom vremenu i neku maksimalnu koja odgovara olujnom vremenu. Nakon toga ćemo nasumično birati talasne dužine iz tog opsega. Možemo koristiti relaciju disperzije navedenu u [Tes2001] da dobijemo brzinu talasa. Ona glasi:

$$w = \sqrt{\frac{g \cdot 2 \cdot \pi}{L}}$$

Ostale parametre možemo izabrati po istom principu iz nekog predefinisanog opsega prema atmosferskim uslovima.

3.1.4. Perlin-ov šum

Šumovi se koriste u grafici za simulaciju prirodnih fenomena koji su ili prezahtevni za precizno izračunavanje i simuliranje ili su jednostavno i sami nasumični. Postoje razne tehnike za generisanje šumova i jedna od jednostavnijih je tehnika koju je preložio Ken Perlin. Međutim, danas postoji nadogradnja originalne tehnike i ona je detaljno opisana u [Per2002].

Ovu tehniku ćemo koristiti da generešimo nešto što je u računarskoj grafici poznato kao mapa visina (engl. *height map*). Ove mape se često koriste u računarskoj grafici i najčešće se pomoću njih generiše teren. Međutim, možemo je koristiti i za crtanje vodenih površi. Ova mapa predstavlja teksturu u kojoj je šum sačuvan kao paletu sivih tonova. Svaki piksel date teksture će sadržati vrednost između 0 (crna boja) i 1 (bela boja). Prilikom crtanja vodene površi, odnosno mreže trouglova, uzorkovaćemo datu teksturu koristeći (x, z) koordinate čvora i vrednost koju pročitamo koristićemo kao visinu čvora u odnosu na horizontalnu ravan. Naravno pre same upotrebe pročitane vrednosti, ona mora biti skalira tako da odgovara izabranoj amplitudi talasa.

Da bi simulirali kretanje talasa potrebno je dodati pomeraj koji će zavisiti od vremena. Ovo je najlakše implementirati tako što se prilikom svakog crtanja teksturne kordinate čvora pomeraju za određeni broj piksela u bilo kom pravcu. Na taj način je obezbeđeno kretanje vodene površi.

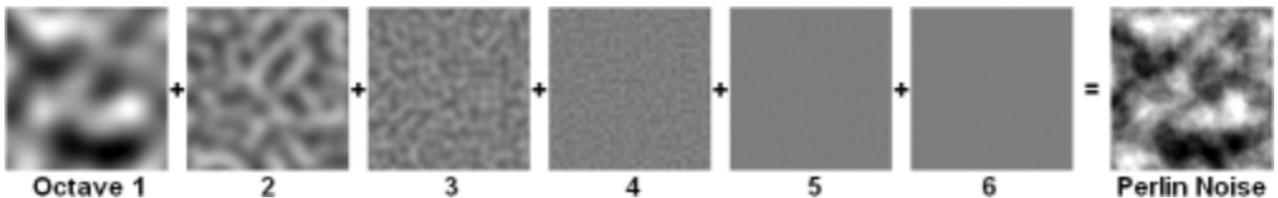
Veoma velika pogodnost ove tehnike je ta što je datu mapu visina potrebno generisati samo jednom pre početka same simulacije vodene površi. Takođe, data tekstura u koju će biti smeštena mapa visina ne mora da bude istih dimenzija kao mreža trouglova već može biti manja, a kasnije prilikom crtanja više mapa visinaće biti naslagano, jedna do druge, da bi dobili jednu veliku mapu visina koja je istih dimenzija kao i mreže trouglova. Jedina manja ove tehnike je ta što se njom mogu simulirati samo usmereni talasi.

Naravno, kao i kod svakog drugog šuma, jedna oktava daje mnogo velike oscilacije između susednih tačaka. Jedna oktava šuma data je na slici 6:



Slika 6. Perlin-ov šum

Ovaj problem se lako prevaziđa tako što saberemo više oktava ovog šuma. Tada dobijamo šum prikazan na slici 7:



Slika 7. Suma 6 oktava Perlin-ovog šuma

3.1.5. Brze Fouier-ove transformacije (FFT)

Najrealističnije vodene površi daju statistički modeli koji jedan talas dekomponuju na sumu sinusoida. Na računarima se to postiže brzim Fouier-ovim transformacijama. Metoda je dosta komplikovana i neće biti detaljno pokrivena ovde, a njen kompletan opis se može naći u radovima [Tes2001] i [Flü2017]. Naime, visina svakog čvora mreže se može izraziti kao suma sinusoida sa kompleksnim i vremenski zavisnim amplitudama. Visina čvora je data formulom:

$$h(\vec{X}, t) = \sum_{\vec{k}} \tilde{h}(\vec{k}, t) \cdot e^{i \cdot \vec{k} \cdot \vec{X}}$$

Gde je vektor \vec{X} horizontalni položaj čvora izražen kao $\vec{X} = (x, z)$, a \vec{k} je dvodimenzionalan vektor izražen kao $\vec{k} = (k_x, k_z)$ i predstavlja vektor pravca datog talasa. Komponente k_x i k_z date su sledećim jednačinama:

$$\begin{aligned} k_x &= \frac{2\pi n}{L_x}, -\frac{N}{2} \leq n \leq \frac{N}{2} \\ k_z &= \frac{2\pi m}{L_z}, -\frac{M}{2} \leq m \leq \frac{M}{2} \end{aligned}$$

Gde su M i N dimenzije mape visina, odnosno teksture, a L_x i L_z su dimenzije dela naše mreže na koji će bit primenjena data mapa visina jer, kao što je navedeno u prethodnom odeljku, mapa visina ne mora da bude istih dimenzija kao i sama mreža trouglova, već je moguće generisati jednu manju mapu visina i kasnije spojiti više istih da bi se dobila mapa visina istih dimenzija kao i mreža trouglova. Amplitude talasa

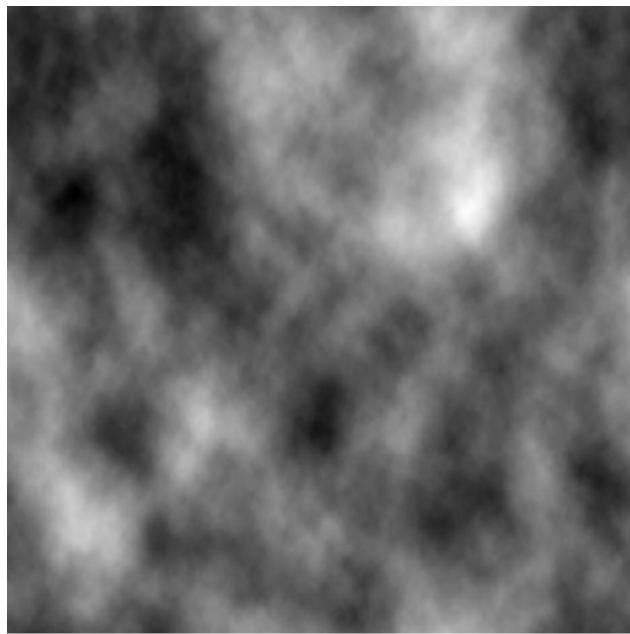
$\tilde{h}(\vec{k}, t)$ su određene strukturom vodene površi i više o tome se može naći u [Tes2001]. Jedina stvar koja nam ostaje jeste da nađemo vektore normala u čvorovima mreže trouglova. Ova tehnika pruža način da se normala precizno odredi ali nam najpre treba odrediti nagib talasa ϵ i on se može izračunati sledećom formulom:

$$\epsilon(\vec{X}, t) = \nabla h(\vec{X}, t) = \sum_{\vec{k}} i \cdot \vec{k} \cdot \tilde{h}(\vec{k}, t) \cdot e^{i \cdot \vec{k} \cdot \vec{X}}$$

Nako ovoga, vektor normale možemo dobiti na sledeći način:

$$\vec{N}(\vec{X}, t) = (0, 1, 0) - (\epsilon_x(\vec{X}, t), 0, \epsilon_z(\vec{X}, t)) = (-\epsilon_x(\vec{X}, t), 0, -\epsilon_z(\vec{X}, t))$$

Na slici 8 je dat izlgled mape visina dobijene pomoću ove metode.



Slika 8. FFT mapa visina[Tes2001]

3.2. Optika

U ovom poglavlju biće prestavljene tehnike koje se koriste za simulaciju efekata do kojih dolazi usled interakcije vodene površi i svetlosnih zraka. Ovi efekti igraju mnogo veću ulogu u postizanju realističnog izgleda vodene površi nego sama simulacija. Međutim njihova implementacija je zahtevnija i to ne samo računarski zahtevna, već zahteva i mnogo veći napor programera. Iz toga razloga se često pribegava aproksimacijama datih efekata koji ipak postižu zadovoljavajuće rezultate. Od ovih efekata ovde će biti pokriveni refleksija, refrakcija i kaustika kao tri efekta koji najviše doprinose izgledu same vodene površi.

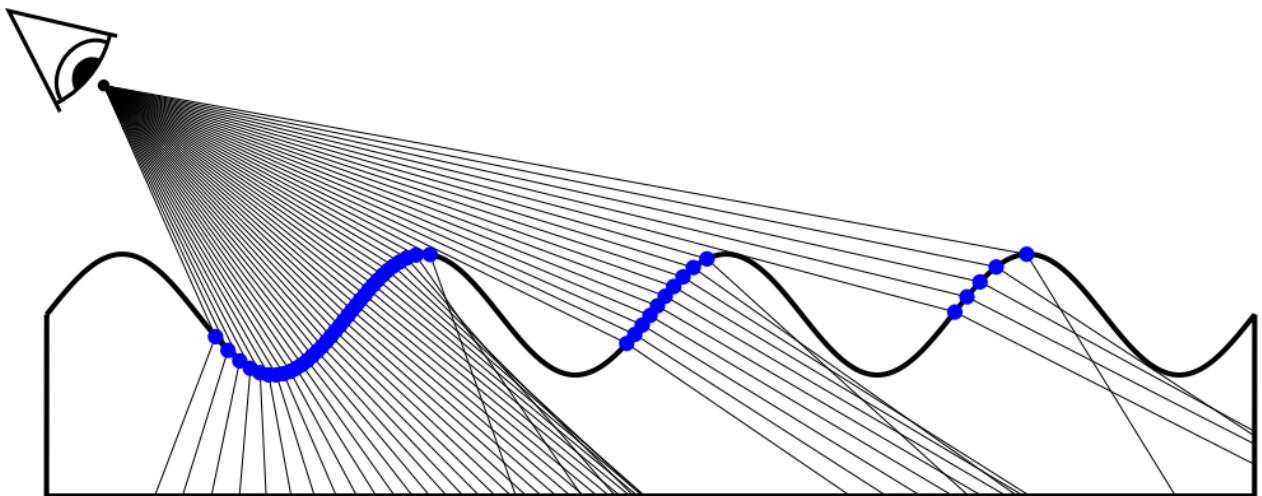
3.2.1. Refleksija i refrakcija

Refleksija i refrakcija su pojave kod kojih se zraci reflektuju i refraktuju prilikom interakcije sa vodenom površinom. Kao što smo videli u poglavlju [2.21](#) za proračun reflektovanog i refraktovanog zraka potreban je vektor normale u dатој тачки. Iz prethodnog poglavlja smo videli da proračun vektora normale ide uporedo sa самом simulacijom kretanja vodene površi. Nakon interakcije sa vodenom površi, zraci nastavljaju dalje, odnosno reflektovani se odbija, a refraktovani prolazi kroz vodenu površ i u nekom trenuntku se ponovo sudsaraju sa objektima u sceni. Boje ovih objekata određuju boju naše vodene površi. Međutim, takvih zraka ima previše da bi se sproveo fizički tačan proračun i dobio realan rezultat. Ovde će biti izložene dve metode pomoću kojih možemo jednostavnije odrediti boju naše vodene površi.

Prvi metod se zove praćenje zraka (engl. *ray tracing*[WikiE]). Koristi se veoma često kada postoji potreba za realističnim rezultatima. U osnovi radi tako što emituje zrake od kamere ka vodenoj površi gde se ti zraci reflektuju i refraktuju, kreirajući nove zrake. Zatim prati putanju novih zraka koji ili udare u neki objekat ili ponovo dođu do vodene površine gde ponovo dolazi do refleksije ili refrakcije i kreiranja novih zraka. Postupak se ponavlja onoliko puta koliko nam je potrebno. Naravno, kao što se može i zaključiti iz opisa, ovaj metod previše zahtevan za crtanje u realnom vremenu i danas ne postoji

grafički hardver koji može u potpunosti da podrži ovaj algoritam, tako da se prilikom njegove implementacije uzimaju određena ograničenja da bi se postigle bolje performanse.

U [GalChi2006] je izložena jedna varijanta ovog algoritma. Ograničenje koje je uvedeno kod ovog pristupa je da se analizira samo jedan nivo rekurzije. Odnosno, nakon interakcije zraka sa vodenom površinom, novonastali zraci se prate bez dalje refleksije ili refrakcije. Logički prikaz ovog pristupa dat je na slici 9:



Slika 9. Algoritam praćenja zraka sa jednim nivoom rekurzije[GalChi2006]

Međutim, i pored ovog ograničenja ovaj metod daje dovoljno dobre rezultate. Kod ovog pristupa početna pretpostavka je da su i voda i teren predstavljeni pomoću mapa visina kojima možemo pristupiti prilikom sprovođenja algoritma. Algoritam radi tako što prati zrak sa namerom da dođe do tačke interakcije zraka sa vodom ili terenom. Ovo se može odrediti binarnom pretragom za datom tačkom ili nekom sličnom metodom zato što na osnovu visinskih mapa vode i terena možemo odrediti poziciju svake njihove tačke. Ukoliko je došlo do interakcije sa terenom na datom pikselu se crta osvetljeni deo terena. Ukoliko je došlo do interakcije sa vodenom površi dolazi do refleksije i refrakcije i novodobijeni zraci se dalje prate sa istom namerom kao originalni zrak. Kada nađemo tačke preseka novih zraka sa ostatkom sredine, odnosno terenom, boje datih tačaka interakcije se kombinujemo da bi se dobila boja vodene površi, naravno poštujući Fresnel-ovu jednačinu. Pseudokod datog algoritma dat je na slici 10:

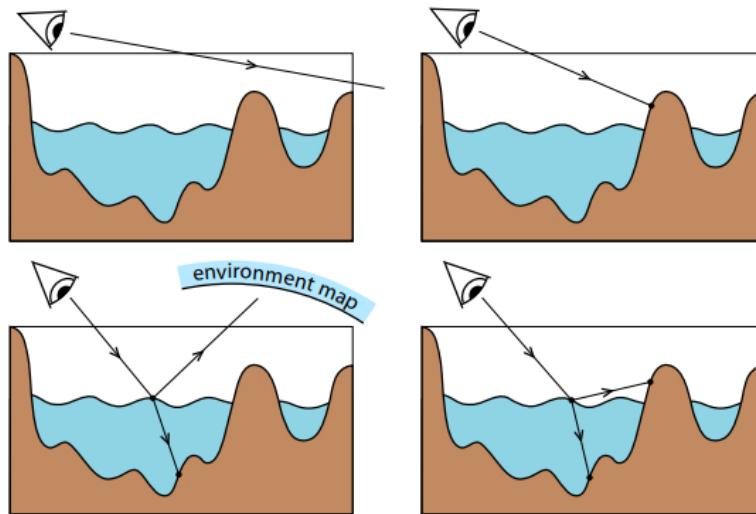
```

 $v \leftarrow \text{viewpoint}$ 
for each screen pixel
     $d \leftarrow \text{direction of the corresponding viewing ray}$ 
     $p_g \leftarrow \text{intersection}(\text{ground}, \text{ray}(v, d))$ 
     $p_w \leftarrow \text{intersection}(\text{water}, \text{ray}(v, d))$ 
    if ( $p_g$  undefined and  $p_w$  undefined)
        do nothing (discard fragment)
    else if ( $p_g$  before  $p_w$ )
        pixel  $\leftarrow \text{lightedGroundColor}(p_g, d)$ 
    else
         $d_t \leftarrow \text{refractedDirection}(d, n(p_w), \eta)$ 
         $d_r \leftarrow \text{reflectedDirection}(d, n(p_w))$ 
         $p_g \leftarrow \text{intersection}(\text{ground}, \text{ray}(p_w, d_t))$ 
         $p_e \leftarrow \text{intersection}(\text{ground}, \text{ray}(p_w, d_r))$ 
         $C_t \leftarrow \text{lightedGroundColor}(p_g, d_t)$ 
         $C_r \leftarrow \text{if } (p_e \text{ undefined})$ 
            envMap( $d_r$ )
        else
            lightedGroundColor( $p_e, d_r$ )
         $F \leftarrow \text{fresnelReflectivity}(d, n(p_w), \eta)$ 
        pixel  $\leftarrow (1 - F) \times C_t + F \times C_r$ 

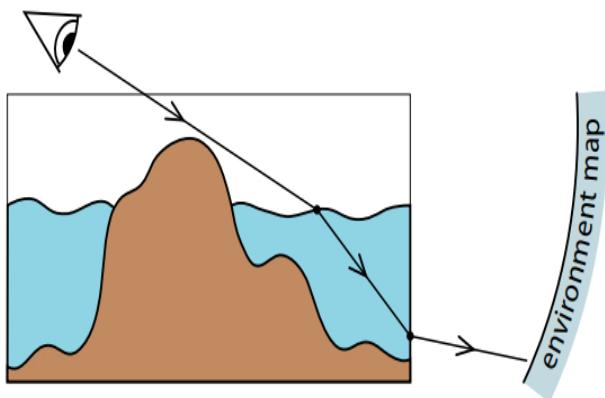
```

Slika 10. Pseudo kod algoritma praćenja zraka GalChi2006

Naravno postoje situacije kada dati zrak izlazi van predela koji mi crtamo. To se može desiti i prilikom refleksije (kada odbijeni zrak ode u "nebo") i prilikom refrakcije kada izađe izvan granica našeg terena. Oba slučaja se mogu rešiti uvođenjem statički definisane teksture koja predstavlja "okolinu" koja nije deo predela na kom je nacrtana vodena površ. Na osnovu putanje zraka možemo odrediti na kojem mestu uzorkujemo tu teksturu. U načelu, algoritam pokriva slučajeve date na slikama 11 i 12:



Slika 11. Različite putanje zraka [GalChi2006]



Slika 12. Zrak izlazi van granica nakon refrakcije[GalChi2006]

Nasuprot ovom algoritmu, koji iako računarski zahtevan daje odlične rezultate, postoji i pristup koji korišćen u [Tur2014] i [Dha2016]. Ova tehnika podrazumeva da u toku svakog crtanja pravimo dve teksture, jednu reflektujuću i jednu refraktujuću. Odnosno, da prilikom svakog crtanja, pre nego što krenemo krenemo u samo crtanje vodene površi, prvo crtamo deo scene koji može da se vidi nakon refleksije i deo scene koji može da se vidi nakon refrakcije. Deo scene koji može da se vidi nakon refleksije je sve što se nalazi iznad vodene površine, međutim kao što možemo videti u prirodi, odraz nečega u vodi je invertovan. Tako pre samog formiranja reflektujuće teksture moramo primeniti refleksionu transformaciju koja će svet invertovati kao odraz u ogledalu, gde je naše ogledalu zapravo horizontalna ravan u visini vodene površi. Ova transformacija je ništa drugo nego skaliranje sa faktorom -1 po y osi teksture. Refraktujuću teksturu crtamo na uobičajeni način. Nakon što generišemo ove teksture možemo preći na crtanje vodene površi. Prilikom crtanja koristićemo projektujuće teksturiranje, tj datu teksturu ćemo projektovati na vodenu površinu kao što projektor prikazuje sliku na zidu. Sam "projektor" uglavnom ima istu poziciju kao i kamera.

3.2.2. Fresnel-ov efekat

Kao što je navedeno u poglavljju [2.2.2](#), Fresnel-ova jednačina nam govori u kojoj meri se zrak svetla reflektuje, odnosno refraktuje od vodene površine. Nakon što nađemo boje tačaka terena u koje su naši zraci svetlosti udarili nakon refleksije, odnosno refrakcije, ova jednačina će nam dati odnos date dve boje. U osnovi, ova jednačina nam govori da što je ugao između vektora pogleda i vektora normale vodene površi u tački interakcije manji to je dominantnija boja dobijena od refraktovanog zraka, a što je ugao između vektora pogleda i vektora normale vodene površine u tački interakcije veći to je dominantnija boja dobijena od reflektovanog zraka. Sama jednačina zahteva značajnu upotrebu računarsih resursa grafičke kartice pa se zato često koriste aproksimacije. Jedna takva aproksimacija je navedena u [Fle2007] i ona računa odnos između boja na sledeći način:

$$F = \frac{1}{1 + \cos \theta^8}$$

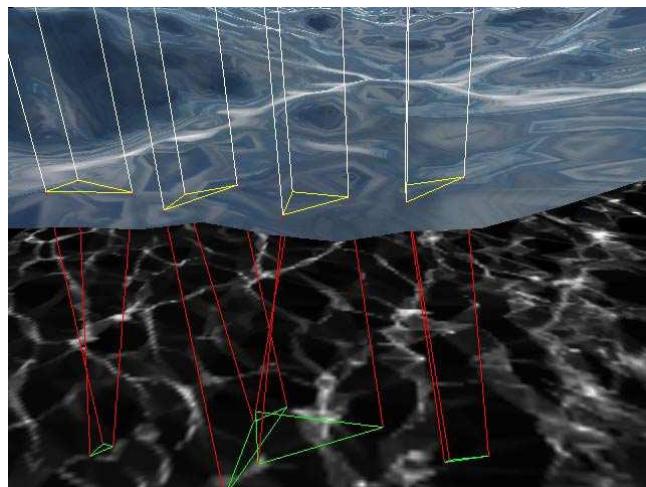
U gore navedenoj jednačini θ predstavlja ugao između vektora pogleda i vektora normale, a F koeficijent koji određuje odnos boje dobijene od reflektujućeg zraka i boje dobijene od refraktujućeg zraka.

Drugi pristup za određivanju ovog koeficijenta jeste da koristimo unapred izračunate kofecijente mešanja boje. Za smeštanje ovih vrednosti biće korišćena heš mapu koja će biti popunjena u nekom od inicijalizacionih koraka. Ključ ove heš mapi će biti ugao između vektora pogleda i vektora normale, a vrednost će biti koeficijent mešanja reflektujuće i refraktojuće boje dobijen iz Fresnel-ove jednačine. Naravno broj uglova koji zaklapaju vektor pogleda i vektor normale je beskonačan, pa će heš mapa sadržati vrednosti za samo određen podskup ovih uglova. Sama heš mapa biće implementirana kao 1D tekstura gde koordinata sa kojom uzorkujem teksturu predstavlja ugao između vektora normale i vektora pogleda, a pročitana vrednost predstavlja odnos mešanja boja. Kasnije ćemo prilikom crtanja pristupati ovoj teksturi, odnosno heš mapi, na osnovu ugla između vektora pogleda i vektora normalne površi u dатој тачки površi da bi pročitali odgovarajuću vrednost i odredili odnos boja.

3.2.3. Kaustika

Kaustika je jedan od najinteresantnijih efekata koje možemo prikazati prilikom crtanja vode, međutim ujedno i jedan od najsloženijih. Kao što je navedeno u poglavlju [2.2.3](#), kaustika predstavlja interferenciju refraktovanih zraka koja čini određene predele ispod vodene površine svetlijim. Kaustiku je nemoguće crtati bez primene tehnike praćenja zraka i to se najčešće primjenjuje obrnuta tehnika praćenja zraka, odnosno izvor zraka nije iz kamere već iz svetlosnog izvora.

Primer ove tehnike je dat u [Fle2007]. Pretpostavka ove tehnike je da je vodena površ predstavljena kao mreža trouglova i da je zemlja ispod vodene površi paralelna vodenoj površi. Za svaki čvor naše mreže ćemo izračunati upadni zrak svetla i odgovarajući refraktovani zrak. Refraktovani zrak će se u nekom trenutku sudsariti sa zemljom ispod vodene površi pa je takođe potrebno naći tačku u kojoj se ovo dešava. Ovo zapravo rezultuje u projektovanju trouglava na zemlju ispod vodene površi i to je prikazano na slici 13:



Slika 13. Projektovani trouglovi [Fle2007]

Intenzitet refraktovanog zraka se može proračunati prema sledećoj formuli:

$$I_c = N \cdot L \cdot \frac{a_s}{a_c}$$

gde je N vektor normale vodene površi u odgovarajućoj tački, L je vektor upadnog zraka koji se prostire od čvora mreže trouglova do svetlosnog izvora, a a_c i a_s su površine projektovanog i površinskog trougla. Sa obzirom na to da je zemlja paralelna vodenoj površi ovi trouglovi se mogu lako iscrtati u teksturu, na koju moramo primeniti algoritme protiv nazupčenja (engl. *antialiasing*) da bi dobili vizuelno zadovoljavajuće efekte. Nakon ovoga se data tekstura paralelno projektuje na zemlju sa visine vodene površine u pravcu izvora svetlosti.

Malo komplikovaniji pristup naveden u [GalChi2006] koristi tehniku mapiranja fotona (engl. *photon mapping*). U ovom radu neće biti izloženi detalje ove tehnike. Načelno, ova tehnika funkcioniše tako što emituje zrake iz izvora svetla ka vodenoj površini i nakon svake interakcije računa refraktovani zrak. Nakon ovoga, refraktovani zrak se prati i ukoliko se dođe do preske sa zemljom ispod vodene površi kordinate preseka kao i doprinos datog fotona celokupnom osvetljenju tog dela zemlje će biti smešteni u određenu keš memoriju, koja je načešće implementirana jedna tekstura. Nakon toga moram primeniti odgovarajuće filtere tako da izoštrimo ovu teksturu jer u prirodi vidimo da je kaustika efekat koji prouzrukuje veome oštре oblike. Poslednji korak jeste da ovako dobijenu teksturu primenimo prilikom iscrtavanja zemlje ispod vodene površi.

4. Implementacija

U ovom poglavlju biće predstavljeno jedno rešenja datog problema kao i njegova implementacija. Rešenje će biti podeljene po malo drugačijem principu nego prethodna poglavlja. Najpre će biti opisana implementacija refleksije i refrakcije, zatim simulacija kretanje vodene površi, zatim Fresnel-ov efekat, zatim spekularno osvetljenje na vodenoj površi i na kraju implementacija kaustike. Razlog za to je taj što je u ovom rešenju vodena površ samo jedan pravougaonik, a ne mreža trouglova. Nema nikakvih simulacija poput onih opisanih poglavlju [3.1](#) već se na određeni način postiže prividno kretanje vode, a to kretanje nije vidljivo sve dok se ne implementiraju efekti refleksije i refrakcije.

Implementacija je odrađena na programskom jeziku Java. Za implementaciju je korišćena biblioteka OpenGL verzije 4.2, tačnije omotačka biblioteka JOGL koja omogućava pristup OpenGL iz koda napisanog u Javi.

4.1. Refleksija i refrakcija

Refleksija i refrakcija su u ovom rešenju implementirane na veoma jednostavan način. Pre samog crtanja vodene površi formirane su reflektujuću i refraktujuću tekstura tako što je u njih crtan određeni deo scene bez vodene površi. Najpre je generisana reflektujuću teksturu, odnosno deo scene iznad vodene površi je crtan u datu tekstuру. Međutim, pre samog crtanja, dati deo scena je invertovan, odnosno primenjena je transformacija "preslikavanja u ogledalu", gde je ogledalo xz ravan. Ovo se može postići transformacijom skaliranja sa faktorom -1 u pravcu y ose, ali je ovde odrađeno na jednostavniji način i to pomeranjem pozicije kamere pre crtanja i vraćanje u originalni položaj nakon crtanja. Kamere je pomerena u negativnom smeru y ose za duplu visinsku razliku između vodene površi i kamere. Pored toga invertovan je nagib kamere, odnosno invertovan je ugao rotacije oko x ose.

Nakon generisanja reflektujuće tekture, generisana je refraktujuća tekstura. Ovo je postignuto crtanjem dela scene ispod vodene površi u datu tekstuру. Za razliku od reflektujuće teksture, prilikom generisanja refraktujuće teksture scene nije transformisana ni na koji način.

Za implementaciju gore navedenih tekstura korišćeni su baferi slike (engl. *framebuffer*[[OpenGL Wiki A](#)]). Naime, pre samog prikazivanja scene na ekranu cela scena se najpre crta u bafer slike, pa se potom sadržaj ovog bafera prikazuje na ekranu. Sam bafer slike ne postoji fizički u memoriji, već prestavlja određenu apstrakciju. On se sastoji iz više komponenti, koje postoje fizički u memoriji, od kojih su dve glavne bafer dubine (engl. *depth buffer*) i bafer boje (engl. *color buffer*).

OpenGL biblioteka pruža opciju da pored glavnog bafera slike, u kojem je smeštena scena koja će biti prikazana na ekranu, kreiramo određen broj novih bafera slike (broj ovih bafera je ograničen karakteristikama grafičkog procesora). Sadržaj novokreiranih bafera slike neće biti prikazan na ekranu i oni se koriste za implementaciju raznih efekata. Novi baferi slike mogu sadržati sve komponente koje poseduje glavni bafer slike. Šta više, možemo koristiti tekture kao komponente ovih bafera. Efekat ovoga jeste da ćemo scenu crtati u neku tekstuру, a ne u glavni bafer slike. Kasnije, u toku prikazivanja scene na ekranu, odnosno crtanja scene u glavni bafer slike, možemo koristite ove tekture na isti način kao i do sada.

U ovoj implementaciji biće kreirana dva nova bafera slike. Jedan ćemo koristiti za generisanje reflektujuće teksture, a drugi za generisanje refraktujuće teksture. Najpre je potrebno stvoriti date baferne slike i oni se, kao i svi ostali objekti u OpenGL biblioteci, stvaraju pozivom *glGen* funkcije, konkretno funkcijom *glGenFrameBuffer* koja vraća celobrojni identifikator novokreiranog bafera slike. Nakon toga je potrebno dodati datom baferu slike bafer boje i bafer dubine, odnosno teksture koje će se koristiti kao bafer boje i bafer dubine. Ovo postižemo funkcijom *glFrameBufferTexture2D* gde kao parametre prosleđujemo celobrojni identifikator tekture koje će nam služiti kao bafer dubine, odnosno bafer boje. Nakon ovoga možemo koristite ove baferne za crtanje scene. Pre samog crtanja potrebno je da specificiramo u koji bafer slike želimo da smestimo scenu. To radimo pozivom funkcije *glBindFramebuffer* kojoj kao parametar prosleđujemo identifikator bafera slike u koji želimo da crtamo scenu. Sve što se su bude crtalo nakon ovog poziva biće smešteno u odabrani bafer slike.

Međutim, neće se cela scena reflektovati na vodenoj površini već samo deo iznad nje. Takodje, neće se cela scena refraktovati na vodenoj površi već samo deo ispod nje. Stoga je potrebno na neki način ukoliniti deo scene ispod vodene površi prilikom generisanja reflektujuće tekture, odnosno deo scene iznad vodene površi prilikom generisanja refraktujuće tekture. Srećom, OpenGL pruža jednostavan način za uklanjanja delova scene u vidu odsecajućih ravnih (engl. *clipping plane*). Naime, moguće je specificirati ravan tako da deo scene koji se nalazi ispod ove ravni neće biti prikazan na ekranu. U OpenGL biblioteci ovo je implementirano pomoću ugrađene nizovne promenljive u *vertex shader*-u. Ovaj niz se zove *gl_ClipDistance* i njegova dužina zavisi od karakteristika grafičkog procesora. Svaki element ovog niza odgovara jednoj odsecajućoj ravni. Ovaj niz se koristi tako što se u *vertex shader*-u prilikom obrade svake tačke smešta njeno rastojanje od odsecajuće ravni u odgovarajući element niza. Ukoliko je to rastojanje pozitivno data tačka će biti prikazana na ekranu, u suprotnom biće eliminisana iz data scene.

Pre samog korišćenja odsecajućih ravnih potrebno je da se one omoguće. Ovo se postiže pozivom funkcije *glEnable* kojoj se kao parametar prosleđuje identifikator odsecajuća ravnih koja će biti korišćena. U ovom rešenju korišćena je samo jedna odsecajuća ravan, i to prva odsecajuća ravan čiji je identifikator dat kao konstanta *GL_CLIP_DISTANCE0*. Sledeći korak je prosleđivanje jednačine kojom je opisana data ravan *vertex shader*-u. Ravan je opisana jednačinom $A \cdot x + B \cdot z + C \cdot y + D = 0$ gde su

A, B, C komponenti vektora normale ravni, a D rastojanje ravni od koordinatnog početka. Ova jednačina će biti prosleđena kao uniformna promenljiva *vec4* tipa koja je u *vertex shader*-u nazvana *clippingPlane0*. Da bi dobili rastojanje tačke od odsecajuće ravni potrebno je samo da pomnožimo poziciju naše tačke u koordinatnom sistemu sveta sa četiri koeficijenta iz jednačine odsecajuće ravni. Na kraju je potrebno da izračunato rastojanje upišem u element niza *gl_ClipDistance* koji odgovara korišćenoj odsecajućoj ravni. Kod *vertex shader*-a kojim se postiže odsecanje dat je na listingu 3:

```
vec4 worldPosition = transform * vec4(vertexPosition, 1.0);
gl_ClipDistance[0] = dot(worldPosition, clippingPlane0);
```

Listing 3. Kod za implementaciju odsecajuće ravni

Cela scena biće crtana tri puta. Prvi put će biti crtan deo scene iznad vodene površine u prvi novokreirani bafer slike. Drugi put će biti crtan deo scene ispod vodene površine u drugi novokreirani bafer slike. Za treće crtanje biće korišćen glavni bafer slike

čiji se sadržaj prikazuje na ekranu, a prilikom ovog crtanja baferi boje dva novokreirana bafera slike će biti korišćeni kao reflektujuća tekstura i refraktujuća tekstura. Reflektujuća i refraktujuća tekstura su date na slikama 14 i 15:



Slika 14. Reflektujuća tekstura



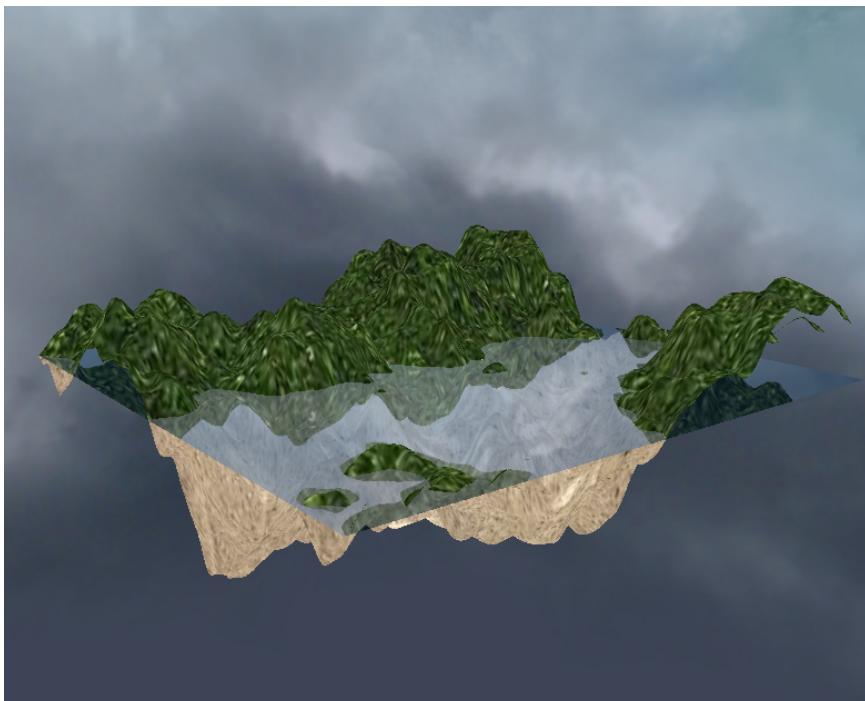
Slika 15. Refraktujuća tekstura

Prilikom crtanja vodene površi biće korišćeno projektujuće teksturiranje, odnosno teksture će biti uzorkovane tako da se dobije efekat projektoru koji prikazuje datu teksturu, gde je pozicija datog projektoru ista kao pozicija kamere. Da bi postigli ovo potrebno je da prosledimo koordinate iz prostora odsecanja (engl. *clip space*) iz *vertex shader-a* u *fragment shader*. Nakon perspektivnog deljenja sa w kordinatom naše tačkećemo koristiti dobijene x i y koordinate za uzorkovanje teksture. Ovako dobijene kordinate su u opsegu $[-1,1]$, tako da ih pre upotrebe potrebno prebaciti u opseg $[0,1]$ tako [što ćemo ih prvo podeliti sa 2 i na tu vrednost dodati 0.5]. Poslednje o čemu treba voditi računa jeste to da je u prirodi prikaz nekog objekta na vodenoj površini prilikom refleksije invertovan tako da y koordinata mora biti invertovana. Nakon toga potrebno je da boje dobijene uzorkovanjem ove dve teksture kombinujemo uz eventualno dodavanje plave boje da bi se dobio realističniji prikaz vodene površi. Na ovaj način će se na vodenoj površini prikazivati i reflektujuća i refraktujuća tekstura. Kod za ovo dat je na listingu 4:

```
vec2 coordinates = clipSpaceCoordinates.xy / clipSpaceCoordinates.w;
coordinates = coordinates / 2 + 0.5;
vec2 refractionTextureCoordinates = coordinates;
vec2 reflectionTextureCoordinates = (coordinates.x, 1 - coordinates.y);
vec2 reflectionColor = texture(reflectioTexture, reflectionTextureCoordinates);
vec2 refractionColor = texture(refractioTexture, refractionTextureCoordinates);
outColor = mix(reflectionColor, refractionColor, 0.5);
outColor = mix(outColor, vec4(0, 0.2, 0.5, 0.2), 0.2);
```

Listing 4. Kod za implementaciju refleksije i refrakcije

Rezultat ovog koda da je na slici 16:



Slika 16. Efekat refleksije i refrakcije

4.2. Simulacija

Za simulaciju kretanja vodene površine nije primenjen nijedan fizički ili statistički model navede u poglavlju [3.1](#) već se koristila metoda koja samo daje privid kretanje vodene površi. Za implementaciju simulacije korišćena je dUDV teksturu. Ova tekstura je RGBA teksturu koja koristi samo crveni i zeleni kanal. Vrednosti iz ovih kanala korišćene su za ubacivanje distorzije prilikom uzorkovanja reflektujuće i refraktujuće teksture. Međutim, sve vrednosti koje se nalaze u ovim kanalima su pozitivne, a nama je potreban i negativan pomeraj. Tako da ćemo pre konačnog korišćena ove vrednosti prvo pomnožiti sa 2 i oduzeti 1 da bi ih prebacili iz opsega $[0,1]$ u opseg $[-1,1]$.

Naravno, da bi se postigao efekat kretanja ova distorzija mora da se menja u toku vremena tako da je u tu svrhu ubaćena uniformna promenljiva tipa *float* nazvana *moveFactor* koje se menja prilikom crtanja svake slike i koju koristimo kao pomeraj prilikom uzorkovanje dUDV teksture. Sama distorzija smeštena u ovoj teksturu je mnogo velika, odnosno vrednosti u crvenom i zelenom kanalu su mnogo velike. Da bi se ovo resilo ubaćena je dodatana uniformna promenljiva tipa *float* koja predstavlja faktor za skaliranje distorzije i koja u suštini određuje jačinu prividnih talasa pa se zato i zove *waveStrength*. Kod za implementaciju ovog efekta dat je na listingu 5:

```

vec2 distortedTextureCoords = texture(
    dudvTexture,
    vec2(textureCoordinates.x + moveFactor, textureCoordinates.y)
).rg * distortionStrength;
distortedTextureCoords = textureCoordinates + vec2(
    distortedTextureCoords.x,
    distortedTextureCoords.y + moveFactor
);
vec2 totalDistortion =
(texture(dudvTexture, distortedTextureCoords).rg * 2.0 - 1.0) * waveStrength;

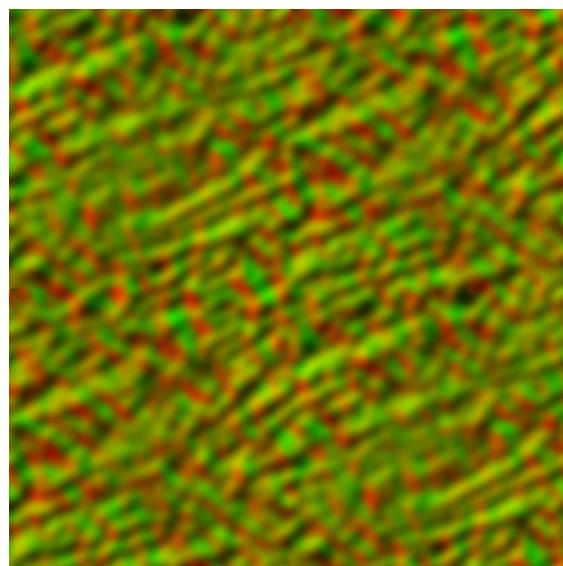
vec2 refractionTextureCoordinates =
    (clipSpaceCoordinates.xy / clipSpaceCoordinates.w) / 2 + 0.5;
refractionTextureCoordinates += totalDistortion;

vec2 reflectionTextureCoordinates = vec2(
    refractionTextureCoordinates.x,
    1 - refractionTextureCoordinates.y
);
reflectionTextureCoordinates += totalDistortion;

vec4 reflectionColor = texture(
    reflectionTexture,
    reflectionTextureCoordinates
);
vec4 refractionColor = texture(
    refractionTexture,
    refractionTextureCoordinates
);

```

*Listing 5. Kod za implementaciju prividnog kretanja vodene površi
Na slici 17 prikazana je konkretna dUDV tekstura:*



Slika 17. dUDV tekstura

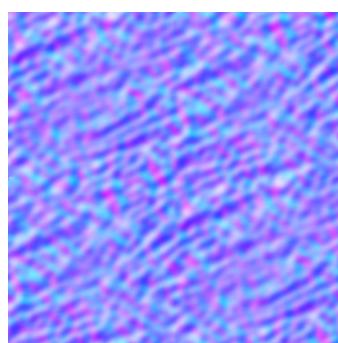
Na slici 18 prikazan je rezultat dobijen ovom tehnikom:



Slika 18. Konačan efekat nakon primenjivanja distorzije na reflektujuću i refraktujuću teksturu

4.3. Fresnel-ov efekat

Sledeća stvar koju sam dodao bila je Fresnel-ov efekat. Kao što je navedeno u poglavljiju [3.2.2](#) Fresnel-ov efekat je komplikovan za egzaktnu računica zato se veoma često koriste aproksimacije koje zavise sam od ugla pogleda. U ovoj implementaciji joe korišćena slična aproksimacija. Ona zavisi od ugla između vektor pogleda i vektora normale u dатој тачки. Ukoliko su oba vektora jedinični taj ugao je lako dobiti i računa se samo jednim pozivom *dot* funkcije коју pruža OpenGL biblioteka. Što je ugao između ova dva vektora manji to je dominantnija boja dobijena od refraktujuće текстуре, а што je ugao između ova dva vektora veći to je dominantnija boja dobijena iz reflektujuće текстуре. Naravno, за implementaciju ovoga potreban je vektor normale u dатој тачки. Za dobijanje vektora normale u određenoj тачки koristiće mapu normala (engl. *normal map*). Ona je data na slici 19:



Slika 19. Mapa normala

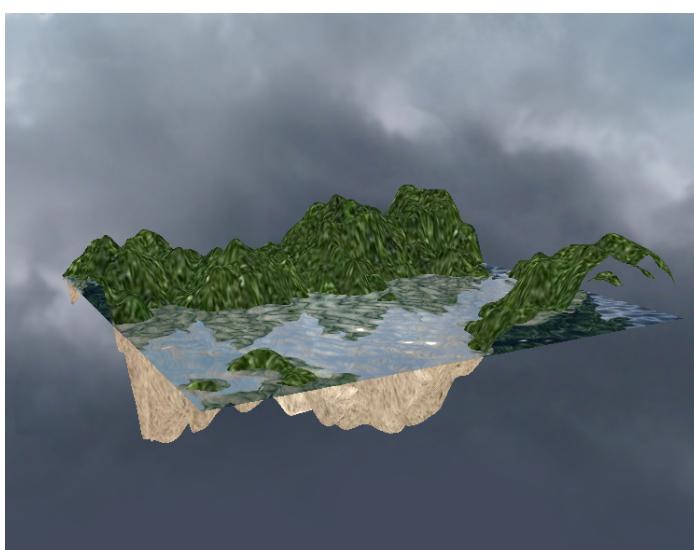
Mapa normala je RGBA текстура, где црвени, зелени и плави канал сваког пикселя представљају x, z и y компоненте вектора нормале, респективно. Као што можемо приметити ова текстура је плавичаста што и има смисла јер су све нормале усмерене на горе

pa je plavi kanal, koji predstavlja y komponentu vektora normale, dominantan. Međutim, sve tri komponente jednog piksela su uvek pozitivne, a naša normala može imati i negativne komponente pa je potrebno da izvršimo određenu konverziju da bi postigli ovo. To radimo tako što vrednosti pročitane vrednosti iz crvenog i zelenog kanala množimo sa 2 i umanjujemo za 1 i na taj način vrednosti prebacujemo iz opsega $[0,1]$ u opseg $[-1,1]$. Ovako dobijeni vektori normale ne moraju da budu jednični vektori. Zato moraju biti normalizovati pre upotrebe.

Još jedna stvar o kojoj moramo da vodimo računa jeste uzorkovanje same mape normala. Mapa prikazana na prethodnoj slici odgovara dUDV teksturi iz prethodnog odeljka, odnosno odgovara teksturi pomoću koje su simulirani talasi. Tako da vektor normale moramo uzorkovati sa istim koordinatama kao i dUDV teksturu, naravno uz odgovarajući pomeraj u toku vremena, da bi dobili što realističnije efekte. Kod za uzorkovanje i normalizaciju vektora normale dat je na listingu 5:

```
vec2 distortedTextureCoords = texture(
    dudvTexture,
    vec2(textureCoordinates.x + moveFactor, textureCoordinates.y)
).rg * distortionStrength;
distortedTextureCoords = textureCoordinates + vec2(
    distortedTextureCoords.x,
    distortedTextureCoords.y + moveFactor
);
vec4 normalMapColor = texture(normalMapTexture, distortedTextureCoords);
vec3 normal = vec3(
    normalMapColor.r * 2 - 1,
    normalMapColor.b,
    normalMapColor.g * 2 - 1
);
normal = normalize(normal);
```

Listing 6. Kod za uzorkovanje i normalizaciju vektora normale
Efekat koji dobijamo nakon ovoga dat je na slici 20:



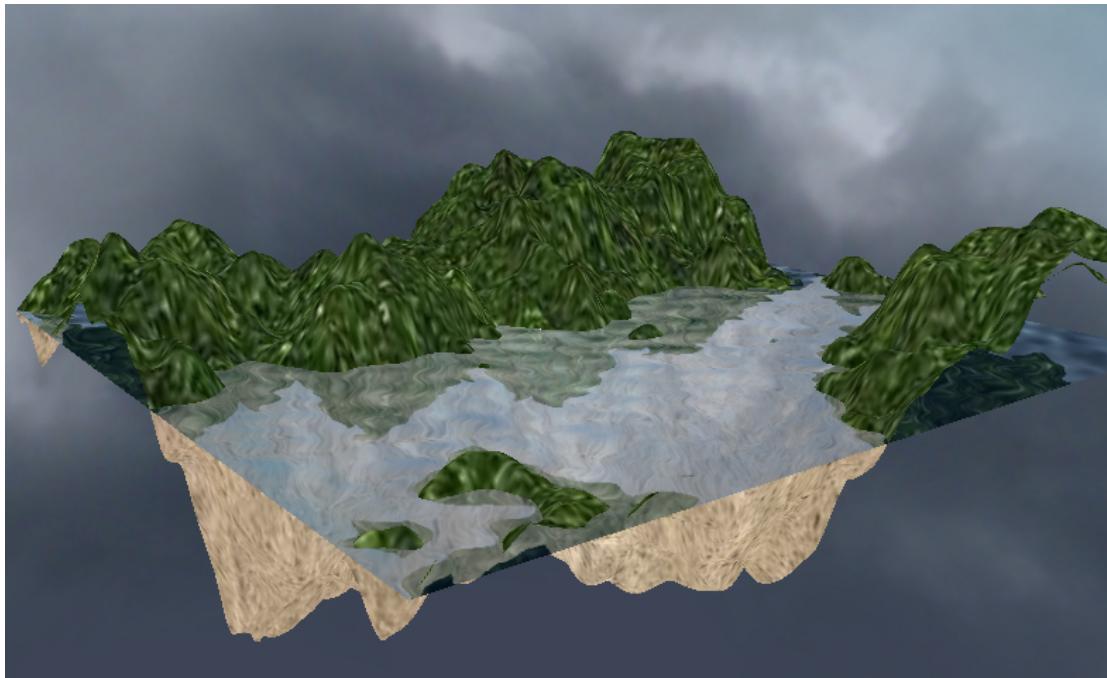
Slika 20. Fresnel-ov efekat

Kao što se može videti normale, iako sve uperene na gore, su raštrkane i zbog toga nije dobijen realističan prikaz vodene površi. To možemo popraviti tako što ćemo pomnožiti y komponentu naše normale sa određenim faktorom tipa *float* koji je nazvan *normalEqualizationFactor*. Nakon toga, kada budemo normalizovali vektor normala, povećana y komponenta će doprineti tome da vektori normala ne budu toliko raštrkane već većim delom usmereni ka gore. Kod kojim se postiže ovo da je na listingu 7:

```
vec4 normalMapColor = texture(normalMapTexture, distortedTextureCoords);
vec3 normal = vec3(
    normalMapColor.r * 2 - 1,
    normalMapColor.b * normalEqualizationFactor,
    normalMapColor.g * 2 - 1
);
normal = normalize(normal);
```

Listing 7. Kod za popravljanje normala

Konačni efekat dobijen sa ovakvim računanjem vektora normala prikazan je na slici 21:



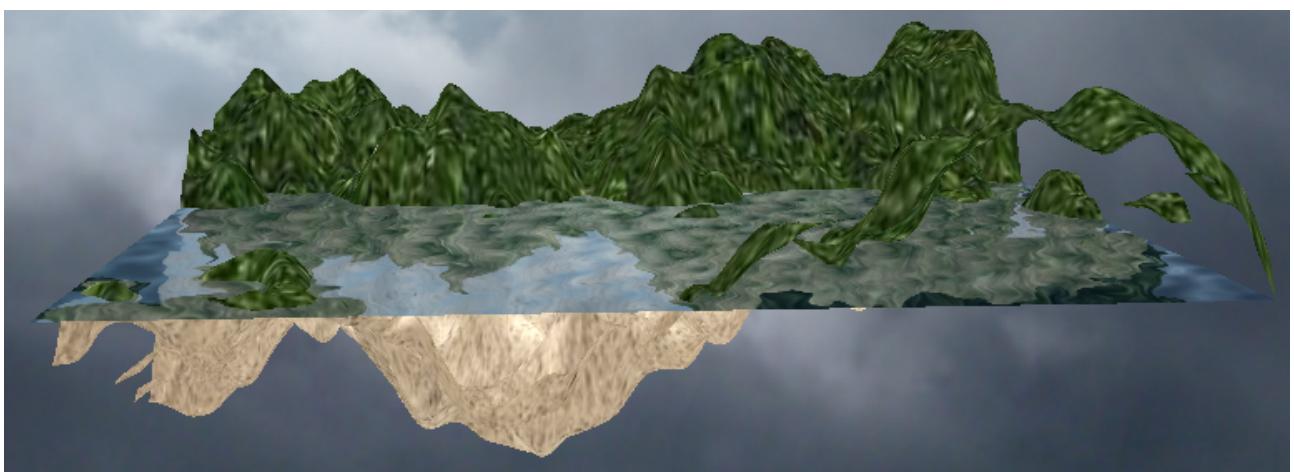
Slika 21. Popravljeni Fresnel-ov efekat

Pravi efekti Fresnel-ove jednačine se mogu videti kada promenimo ugao pogleda. Ukoliko je ugao između vektor pogleda i vektora normale veći biće jači efekat refrakcije kao što se vidi na slici 22:



Slika 22. Refrakcija sa Fresnel-ovom jednačinom

Ukoliko je ugao između vektor pogleda i vektora normale veći biće jači efekat refleksije što se vidi na slici 23:



Slika 23. Refleksija sa Fresnel-ovom jednačinom

4.4. Spekularno osvetljenje

U cilju što realističnijeg izgleda, dodato je i spekularno osvetljenje vodene površi. Po uzoru na Phong-ov model senčenja, za svaki piksel vodene površi pronađen je vektor upadnog zraka. Nakon toga, na osnovu vektora normale u datoј tački, izračunat je vektor reflektovanog zraka. Na kraju je izračunat ugao između vektora reflektovanog zraka i vektora pogleda, koji je usmeren od date tačke ka kameri. Ovo je jednostavno odraditi jer su oba vektora jedinični vektori i ugao između njih možemo dobiti jednostavnim proizvodom. Nakon toga je taj ugao podignut na određeni stepen i pomnožen sa korigujućim faktorom. Obe vrednosti su uniformne promenljive tipa *float* nazvane *shineDamper* i *lightReflectivity*, respektivno. Iako one zavise od karakteristika vodene

površi, do ovih faktora se došlo probom. Kod kojim je implementirano spekularno osvetljenja dat je na listingu 8:

```
vec3 reflectedLight = reflect(normalize(fromLightVector), normal);
float specular = max(
    dot(normalizedToCameraVector, reflectedLight),
    0
);
vec3 specularHighlights = lightColor.rgb * pow(
    specular,
    shineDamper
) * lightReflectivity;
```

Listing 8. Kod za implementaciju spekularnog osvetljenja

Nakon ovog proračuna, *specularHighlights* vektor je jednostavno dodat boji koja je dobijena mešanjem vrednosti pročitanih iz reflektujuće i refraktujuće teksture. Efekat ovog se može videti na slici 24:



Slika 24. Efekat spekularnog osvetljenja

4.5. Kaustika

Kao što je navedeno u odeljku [3.2.3](#) kaustika je efekat koji je veoma teško implementirati i gotovo je nemoguće postići to bez neke vrste praćenja zraka. U ovom rešenju upotrebljena je ideja koja je slična onoj opisanoj u [FerKir2004]. Međutim prilikom implementacije ove ideje usvojena je jako gruba pretpostavka, a to je da na kaustiku utiču samo zraci refraktovani pod pravim uglom. Na ovaj način sa svakog piksela površi ispod vodene površi možemo pratiti zrak do vodene površi i odrediti originalni upadni zrak na osnovu refraktovanog zraka. Ovaj zrak ne potiče od svetlosnog izvora pa je nakon toga potrebno odrediti ugao između vektora upadnog zraka i zraka koji potiče od svetlosnog izvora. Što je ovaj ugao manji to je piksel svetlijiji.

Vektor upadnog zraka se dobija iz Fresnel-ove jednačine. Posmatrajmo sliku 1, odnosno refrakciju na slici 1. Zamislimo da postoji još jedan vektor \vec{n}_t koji je normalan u odnosu na vektor normale vodene površi i prostire se u ravni upadnog zraka i refraktovanog zraka i to u pravcu refraktovanog zraka. Nije teško da na osnovu tih informacija izvedeo jednačine za vektor upadnog zraka i vektor refraktovanog zraka i one glase:

$$(1) \vec{v} = \cos(\theta) \cdot \vec{n} - \sin(\theta) \cdot \vec{n}_t$$

$$(2) \vec{v}_t = -\cos(\theta_t) \cdot \vec{n} + \sin(\theta_t) \cdot \vec{n}_t$$

Na osnovu ovoga možemo odrediti jednačinu vektora \vec{n}_t u prvom i u drugom slučaju i izjednačavanjem te dve jednačine odrediti jednačinu vektora upadnog zraka. Postupak je sledeći:

$$(1) \vec{n}_t = \cot(\theta) \cdot \vec{n} - \frac{1}{\sin(\theta)} \cdot \vec{v}$$

$$(2) \vec{n}_t = \frac{1}{\sin(\theta_t)} \cdot \vec{v}_t + \cot(\theta_t) \cdot \vec{n}$$

$$(1) = (2)$$

$$\cot(\theta) \cdot \vec{n} - \frac{1}{\sin(\theta)} \cdot \vec{v} = \frac{1}{\sin(\theta_t)} \cdot \vec{v}_t + \cot(\theta_t) \cdot \vec{n}$$

$$\vec{v} = -\frac{\sin(\theta)}{\sin(\theta_t)} \cdot \vec{v}_t + \cos(\theta) \cdot \vec{n} - \frac{\cos(\theta_t) \cdot \sin(\theta)}{\sin(\theta)} \cdot \vec{n}$$

Dalje korišćenjem Snell-ovog zakona i trigonometrijskog identiteta možemo doći do sledeće formule za vektor upadnog zraka:

$$\vec{v} = -\frac{n_2}{n_1} \cdot \vec{v}_t - \vec{n} \cdot \left(\frac{n_2}{n_1} \cdot \cos(\theta_t) - \cos(\theta) \right)$$

$$\vec{v} = -\frac{n_2}{n_1} \cdot \vec{v}_t - \vec{n} \cdot \left(\frac{n_2}{n_1} \cdot \cos(\theta_t) - \sqrt{(1 - \sin^2(\theta))} \right)$$

$$\vec{v} = -\frac{n_2}{n_1} \cdot \vec{v}_t - \vec{n} \cdot \left(\frac{n_2}{n_1} \cdot \cos(\theta_t) - \sqrt{\left(1 - \left(\frac{n_2}{n_1}\right)^2 \cdot \sin^2(\theta_t)\right)} \right)$$

$$\vec{v} = -\frac{n_2}{n_1} \cdot \vec{v}_t - \vec{n} \cdot \left(\frac{n_2}{n_1} \cdot \cos(\theta_t) - \sqrt{\left(1 - \left(\frac{n_2}{n_1}\right)^2 \cdot (1 - \cos^2(\theta_t))\right)} \right)$$

Ukoliko su vektor normale i vektor refraktovanog zraka jedinični vektori, ugao koji zaklapaju ova dva vektora se dobija jednostavnim množenje ta dva vektor. Naravno, treba voditi računa da vektor upadnog zraka dobijen ovom računicom nije jedinični vektor pa ga je potrebno normalizovati. Nakon što nađemo ugao između ova dva vektora, možemo ga iskoristimo na isti način na kod spekularnog osvetljenja, odnosno da određenim faktorima postignemo vizuelno zadovoljavajući efekat. Jedna razlika u kodu u odnosu na prethodne jednačine jeste početna prepostavka smera vektora upadnog zraka, odnosno prepostavljeno je da vektor upadnog zraka ima suprotan smer od onog na slici 1.

Za implementaciju kaustike potrebna je mapa normala, a uz to i dUDV tekstura koja je korišćena prilikom uzorkovanja mape normala. Uzorkovanje se radi po istom principu kao što je navedeno u odeljku [4.3](#). Kod za implementaciju kaustike dat je na listingu 9:

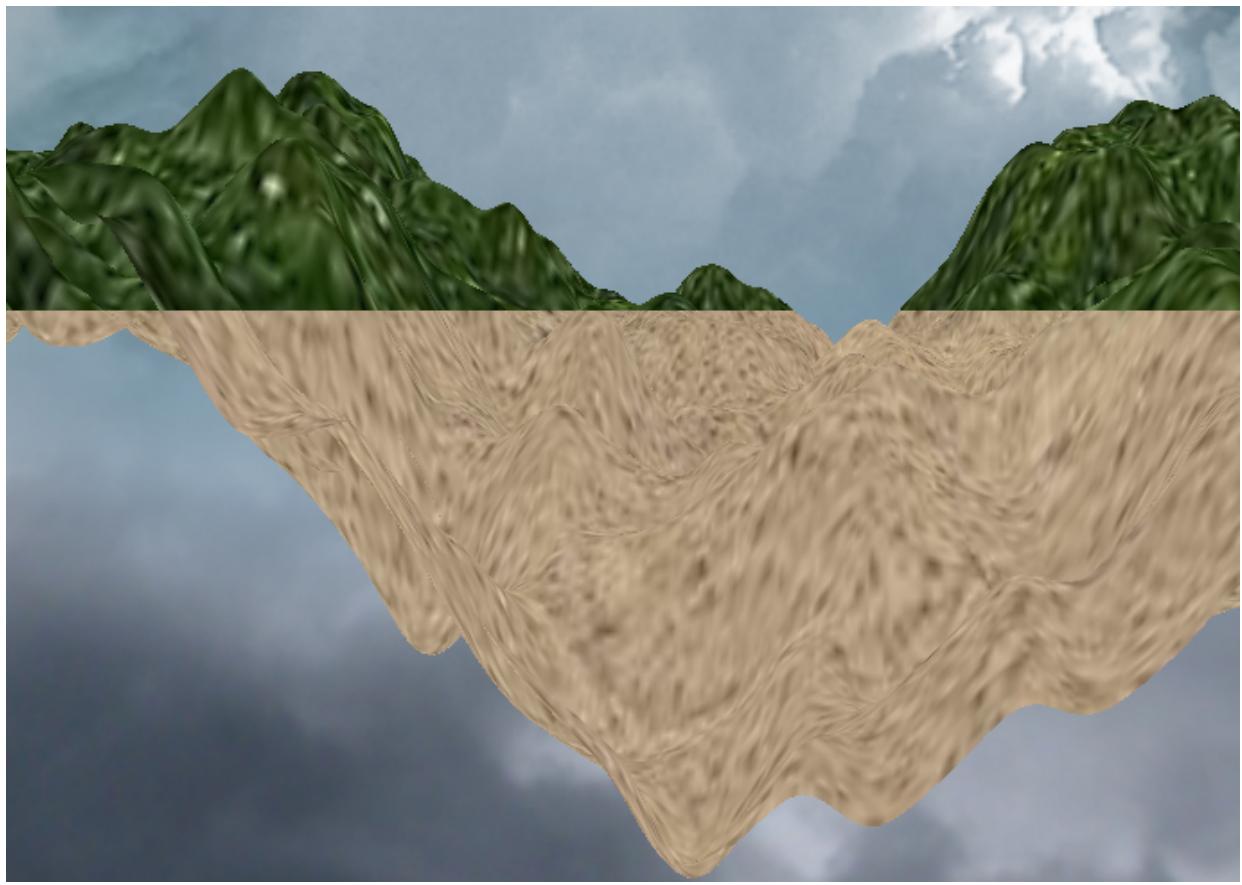
```

vec2 distortedTextureCoords = texture(
    dudvTexture,
    vec2(textureCoordinates.x + moveFactor, textureCoordinates.y)
).rg * distortionStrength;
distortedTextureCoords = textureCoordinates + vec2(
    distortedTextureCoords.x,
    distortedTextureCoords.y + moveFactor
);
vec4 normalMapColor = texture(normalMapTexture, distortedTextureCoords);
vec3 normal = vec3(
    normalMapColor.r * 2 - 1,
    normalMapColor.b,
    normalMapColor.g * 2 - 1
);
normal = normalize(normal);
vec3 refractedRay = vec3(0, -1, 0);
float theta = dot(-normal, refractedRay);
vec3 incidentRay = eta * refractedRay + normal * (eta * theta - eta * sqrt(
1 - eta * eta * (1 - theta * theta)
));
incidentRay = normalize(-incidentRay);
vec3 normalizedToLightVector = normalize(
lightPosition - waterSurfaceCoordinates
);
float specular = max(dot(normalizedToLightVector, incidentRay), 0);
specular = pow(specular, shineDamper);
vec3 specularHighlights = lightColor.rgb * specular * lightReflectivity;
outColor = texture(terrainTexture, textureCoordinates) +vec4(
    specularHighlights,
    0
);

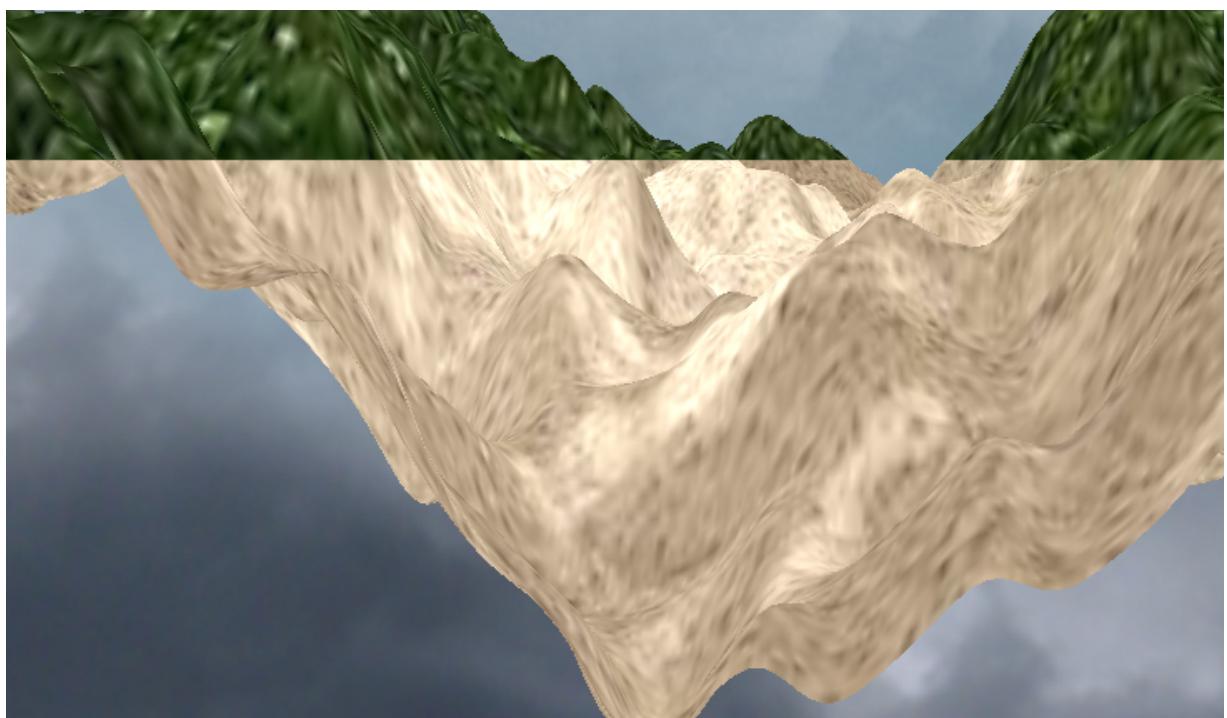
```

Listing 9. Kod za implementaciju kaustike

Konstanta **eta** predstavlja odnos indeksa refrakcije vode i indeksa refrakcije vazduha i ima vrednost 0.75. Na slikama 25 i 26 dati su izgledi površi bez i sa efektom kaustike, respektivno:



Slika 25. Površ bez efekta kaustike



Slika 26. Površ sa efektom kaustike

Kao što se može videti sa slike 26 efekat nije savršen. Zraci ne prave oštре figure kao što se to dešava u prirodi. Međutim, metoda je dosta jednostavna i brza zato što ne zahteva implementaciju praćenja zraka. Odnosno, ne zahteva povećano korišćenje resursa grafičkog procesora zato što u implementacije ove metode nema rekurzivnog praćenja upadnog zraka i određivanja da li on dolazi od izvora svetla ili ne, što je potrebno odraditi ukoliko želimo da implementiramo pravi algoritam praćenja zraka.

5. Zaključak

Cilj ovog seminarског rada je bio da se istraži na koje sve probleme nailazimo prilikom crtanja vodenih površi i koje su to tehnike koje se koriste za prevazilaženje opisanih problema. Pored toga, data je jedna jednostavna implementacija vodene površi. Sama implementacija je bazirana na idejama predstavljenim u ovom radu.

Glavni problemi na koje nailazimo su simulacija kretanje vodene površi i implementacija optičkih efekata, od kojih refleksija i refrakcija najviše doprinose realističnom izgledu vodene površi. Pored ova dva efekta dodata je kaustika kao još jedan efekat koji doprinosi realističnom izgledu celokupne scene, međutim ne u toj meri kao prva dva. Pored problema, opisane su razne tehnike za njihovo rešavanje koje idu od lakših ka težim. Na kraju je data jedna implementacija koja ilustruje sve te probleme i neke od načina na koji se oni rešavaju.

Međutim ovo nije kraj jer crtanje vodenih površina je jako zahtevan posao i svakim danom nailazimo na nove tehnike koje doprinosimo realističnom izgledu vodenih površi. Sledeći korak u daljem istraživanju bi bio da se detaljnije istraže tehnika praćenja zrakova i njeni korišćenje prilikom implementacije optičkih efekata. Pored same tehnike potrebno je istražiti i to koliko je ona primenjiva jer, kao što je navedeno u radu, ona je jako zahtevna i današnji grafički hardver nije opremljen da sprovodi celokupan algoritam već se moraju uvesti dodatna ograničenja. Pored ovoga, takođe je potrebno istražiti druge efekte koji doprinose realistično izgledu kao što su pena, mehurići, simulacija sudaranja talasa i sl.

Literatura

[Fle2007] Bernhard Fleck, "Real-Time Rendering of Water in Computer Graphics", 2007

[WikiA] "Navier–Stokes equations", <https://en.wikipedia.org/wiki/Navier%20%26%20%28%20%29>, 02.03.2018

[Tur2014] Aharon Turpie, "Real time rendering of optical effects of water", 2014

[WikiB] "Snell's law", https://en.wikipedia.org/wiki/Snell%27s_law, 02.03.2018

[WikiC] "Polarization (waves)", [https://en.wikipedia.org/wiki/Polarization_\(waves\)](https://en.wikipedia.org/wiki/Polarization_(waves)), 03.03.2018

[Zel2004] Jeremy Zelsnack, "SDK White Paper, Vertex Texture Fetch Water ", NVIDIA Corporation , 2004

[WikiD] "Curve fitting", https://en.wikipedia.org/wiki/Curve_fitting, 04.03.2018

[FerKir2004] Randima Fernando, David Kirk, "GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics", Addison-Wesley Professional, 2004

[Tes2001] Jerry Tessendorf, "Simulating Ocean Water", 2001

[Per2002] Ken Perlin, "Improving Noise ", 2002

[Flü2017] Flynn-Jorin Flügge, "Realtime GPGPU FFT, Ocean Water Simulation", Hamburg Universitz of Technologz, Institute of Embedded Systems, 2017

[WikiE] "Ray tracing (graphics)", [https://en.wikipedia.org/wiki/Ray_tracing_\(graphics\)](https://en.wikipedia.org/wiki/Ray_tracing_(graphics)), 06.03.2018

[GalChi2006] E. Galin, N. Chiba, "Realistic Water Volumes in Real-Time", 2006

[Dha2016] Jagjeet Singh Dhaliwal, "Real-time water simulation",, , 2016

[OpenGLWikiA] "Framebuffer Object", https://www.khronos.org/opengl/wiki/Framebuffer_Object, 07.03.2018

Slike

Slika 1. Refleksija (levo) i refrakcija (desno)[Fle2007].....	2
Slika 2. Kaustika.....	4
Slika 3. Kubna kriva[Zel2004].....	6
Slika 4. 2D jednačina talasa[Zel2004].....	7
Slika 5. Usmereni (levo) i kružni (desno) talasi[FerKir2004].....	10
Slika 6. Perlin-ov šum.....	12
Slika 7. Suma 6 oktava Perlin-ovog šuma.....	12
Slika 8. FFT mapa visina[Tes2001].....	13
Slika 9. Algoritam praćenja zraka sa jednim nivoom rekurzije[<i>GalChi2006</i>].....	14
Slika 10. Pseudo kod algoritma praćenja zrakaGalChi2006.....	15
Slika 11. Različite putanje zraka[<i>GalChi2006</i>].....	15
Slika 12. Zrak izlazi van granica nakon refrakcije[<i>GalChi2006</i>].....	16
Slika 13. Projektovani trouglovi[Fle2007].....	17
Slika 14. Reflektujuća tekstura.....	21
Slika 15. Refraktujuća tekstura.....	21
Slika 16. Efekat refleksije i refrakcije.....	22
Slika 17. dUdV tekstura.....	23
Slika 18. Konačan efekat nakon primenjivanja distorzije na reflektujuću i refraktujuću teksturu ..	24
Slika 19. Mapa normala.....	24
Slika 20. Fresnel-ov efekat.....	25
Slika 21. Popravljeni Fresnel-ov efekat.....	26
Slika 22. Refrakcija sa Fresnel-ovom jednačinom.....	27
Slika 23. Refleksija sa Fresnel-ovom jednačinom.....	27
Slika 24. Efekat spekularnog osvetljenja.....	28
Slika 25. Površ bez efekta kaustike.....	31
Slika 26. Površ sa efektom kaustike.....	31

Listinzi

Listing 1. Kod za generisanje mape visina pomoću 2D jednačine talasa.....	7
Listing 2. Kod za računanje vektora normale.....	8
Listing 3. Kod za implementaciju odsecajuće ravnii.....	20
Listing 4. Kod za implementaciju refleksije i refrakcije.....	21
Listing 5. Kod za implementaciju prividnog kretanja vodene površi.....	23
Listing 6. Kod za uzorkovanje i normalizaciju vektora normale.....	25
Listing 7. Kod za popravljanje normala.....	26
Listing 8. Kod za implementaciju spekularnog osvetljenja.....	28
Listing 9. Kod za implementaciju kaustike.....	30

Dodatak A - Kod *vertex shader-a* za implementaciju vodene površi

```
#version 420 core
layout (location = 0) in vec3 vertexCoordinates;
layout (location = 1) in vec2 inTexelCoordinate;

out vec4 clipSpaceCoordinates;
out vec2 textureCoordinates;
out vec3 toCameraVector;
out vec3 fromLightVector;

uniform mat4 transform, projection, view;
uniform vec3 cameraPosition;
uniform vec3 lightPosition;

void main() {
    textureCoordinates = inTexelCoordinate;
    vec4 worldPosition = transform * vec4(vertexCoordinates, 1);
    clipSpaceCoordinates = projection * view * worldPosition;
    gl_Position = clipSpaceCoordinates;
    toCameraVector = cameraPosition - worldPosition.xyz;
    fromLightVector = worldPosition.xyz - lightPosition;
}
```

Dodatak B - Kod *fragment shader-a* za implementaciju vodene površi

```
#version 420

in vec4 clipSpaceCoordinates;
in vec2 textureCoordinates;
in vec3 toCameraVector;
in vec3 fromLightVector;

out vec4 outColor;

uniform sampler2D reflectionTexture, refractionTexture, dudvTexture,
           normalMapTexture;
uniform float waveStrength, moveFactor, distortionStrength, waterReflectivity,
           shineDamper, lightReflectivity, normalEqualizationFactor;
uniform vec4 lightColor;

void main() {
    vec2 distortedTextureCoords = texture(
        dudvTexture,
        vec2(textureCoordinates.x + moveFactor, textureCoordinates.y)
    ).rg * distortionStrength;
    distortedTextureCoords = textureCoordinates + vec2(
        distortedTextureCoords.x,
        distortedTextureCoords.y + moveFactor
    );
    vec2 totalDistortion = texture(dudvTexture, distortedTextureCoords).rg;
    totalDistortion = (totalDistortion * 2.0 - 1.0) * waveStrength;
    vec2 refractionTextureCoordinates =
        (clipSpaceCoordinates.xy / clipSpaceCoordinates.w) / 2 + 0.5;
    refractionTextureCoordinates += totalDistortion;
    vec2 reflectionTextureCoordinates = vec2(
        refractionTextureCoordinates.x, 1 -
        refractionTextureCoordinates.y
    );
    reflectionTextureCoordinates += totalDistortion;
    vec4 reflectionColor = texture(
        reflectionTexture,
        reflectionTextureCoordinates
    );
    vec4 refractionColor = texture(
        refractionTexture,
        refractionTextureCoordinates
    );
    vec4 normalMapColor = texture(normalMapTexture, distortedTextureCoords);
    vec3 normal = vec3(
        normalMapColor.r * 2 - 1,
        normalMapColor.b * normalEqualizationFactor,
        normalMapColor.g * 2 - 1
    );
    normal = normalize(normal);
    vec3 normalizedToCameraVector = normalize(toCameraVector);
    float refractiveFactor = dot(normalizedToCameraVector, normal);
    vec3 reflectedLight = reflect(normalize(fromLightVector), normal);
```

```
float specular = max(dot(normalizedToCameraVector, reflectedLight), 0);
vec3 specularHighlights =
    lightColor.rgb * pow(specular, shineDamper) * lightReflectivity;
outColor = mix(
    reflectionColor,
    refractionColor,
    pow(refractiveFactor, waterReflectivity)
);
outColor = mix(outColor, vec4(0, 0.2, 0.5, 0.2), 0.2);
outColor = outColor + vec4(specularHighlights, 0);
}
```

Dodatak C - Kod *vertex shader-a* za implementaciju površi ispod vodene površi

```
#version 420 core

layout (location = 0) in vec3 vertexPosition;
layout (location = 1) in vec2 inTextureCoordinates;

uniform mat4 transform, projection, view;
uniform vec4 clippingPlane0;
uniform float waterHeight;

out vec2 textureCoordinates;
out vec3 waterSurfaceCoordinates;

void main() {
    textureCoordinates = inTextureCoordinates;
    vec4 worldPosition = transform * vec4(vertexPosition, 1.0);
    waterSurfaceCoordinates = vec3(
        worldPosition.x,
        waterHeight,
        worldPosition.z
    );
    gl_ClipDistance[0] = dot(worldPosition, clippingPlane0);
    gl_Position = projection * view * worldPosition;
}
```

Dodatak D - Kod *fragment shader-a* za implementaciju površi ispod vodene površi

```
#version 420

#define eta 0.75

in vec2 textureCoordinates;
in vec3 waterSurfaceCoordinates;

out vec4 outColor;

uniform sampler2D terrainTexture;
uniform sampler2D dudvTexture, normalMapTexture;
uniform float moveFactor, distortionStrength, shineDamper, lightReflectivity,
           normalEqualizationFactor;
uniform vec3 lightPosition;
uniform vec4 lightColor;

void main() {
    vec2 distortedTextureCoords = texture(
        dudvTexture,
        vec2(textureCoordinates.x + moveFactor, textureCoordinates.y)
    ).rg * distortionStrength;
    distortedTextureCoords = textureCoordinates + vec2(
        distortedTextureCoords.x,
        distortedTextureCoords.y + moveFactor
    );
    vec4 normalMapColor = texture(normalMapTexture, distortedTextureCoords);
    vec3 normal = vec3(
        normalMapColor.r * 2 - 1,
        normalMapColor.b,
        normalMapColor.g * 2 - 1
    );
    normal = normalize(normal);
    vec3 refractedRay = vec3(0, -1, 0);
    float theta = dot(-normal, refractedRay);
    vec3 incidentRay = eta * refractedRay + normal * (eta * theta - eta *
        sqrt(1 - eta * eta * (1 - theta * theta)));
    incidentRay = normalize(-incidentRay);
    vec3 normalizedToLightVector = normalize(
        lightPosition - waterSurfaceCoordinates
    );
    float specular = max(dot(normalizedToLightVector, incidentRay), 0);
    specular = pow(specular, shineDamper);
    vec3 specularHighlights = lightColor.rgb * specular * lightReflectivity;
    outColor = texture(terrainTexture, textureCoordinates) + vec4(
        specularHighlights, 0
    );
}
```