

Univerzitet u Beogradu – Elektrotehnički fakultet

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

Đukić Jovan, 3016/2017

Apstrakt

U ovom radu biće predstavljeno crtanje vodenih površina, 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 površine i optički efekti do kojih dolazi prilikom interakcije vode sa zracima svetla. Od ovih efekata biće pokrivena refleksija i refrakcija, kao dva najbitnija efekta koji doprinose realistično izgledu vode. Pored ovih, takođe će biti pokrivena kaustika kao treći efekat koji doprinosi realistično prikazu vodene površine, doduše ne u tolikoj meri kao prethodna dva. Na kraju ovog rada biće dat kratak opis jedne moguće implementacije vodene površine 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.Progled 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.....	11
3.1.4.Perlin-ov šum.....	12
3.1.5.Brze Fouier-ove transformacije (FFT).....	13
3.2.Optika.....	14
3.2.1.Refleksija i refrakcija.....	14
3.2.2.Fresnel-ov efekat.....	17
3.2.3.Kaustika.....	18
4.Implementacija.....	20
4.1.Refleksija i refrakcija.....	20
4.2.Simulacija.....	23
4.3.Fresnel-ov efekat.....	24
4.4.Spekulativno osvetljenje.....	28
4.5.Kaustika.....	29
5.Zaključak.....	32
Literatura.....	33
Slike.....	34

1. Uvod

Kako je moć grafičkih kartica rasla 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 fenome prikažemo onakve kakvi oni jesu u prirodi. Od svih fizičkih pojava koje se prikazuju, vodene površine zahtevaju najviše napora i truda. Rezultat ovih napora se može videti kako u video igricama tako i u filmovima.

Uprkos današnjem hardveru, prilično crtanja vodenih površina 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, dubine itd. I interakcija vode sa svetлом u koju spadaju efekti kao što su refleksija, refrakcija, kaustika, "božji zraci" itd. Danas je skoro nemoguće predstaviti vodu poštujući njeno ponašanje 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 prilično crtanja vode možemo podeliti u dve kategorije koje odgovaraju gore navedenim problemima:

1. Simulacija vodenih površina 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 predočeni 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.

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 načne potrebe. Zato se ~~dosta~~ često koriste ~~raznorazne~~ 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.

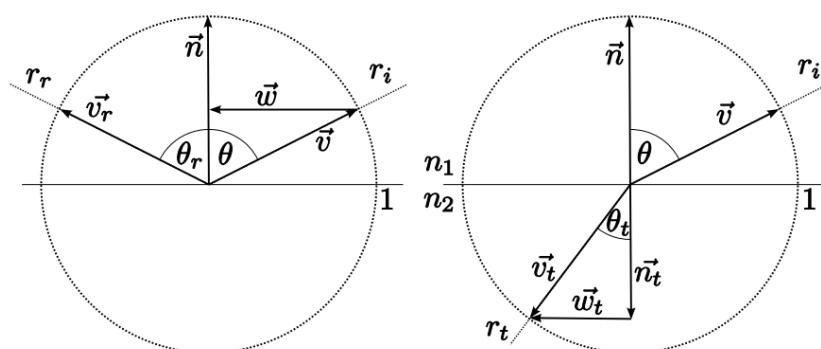
Pored samog izbora načina simulacije postoji i problem modela same vodene površine. Najčešće je to mreža trouglova, međutim tu se postavlja pitanje koliki broj trouglova ta mreža treba da sadrži. Naravno, što je data mreža gušća to je i sama vodena površ realističnija ali kao što je navedeno u [Tur2014]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

2.2. Optika

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

2.2.1. Refleksija i refrakcija

Da bi uopšte mogli da krenemo u razmatranje interakcije 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 površine. Ovo generalno nije problem jer kada znam pozicije tačaka vodene površine lako ćemo nekom matematičkom metodom dobiti i normale u tim tačkama. Način delovanja refleksije i refrakcije dat je na sledećoj slici:



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

Najpre ćemo analizirati refleksiju, odnosno levi deo prethodne slike. Neka su vektori $\vec{v}, \vec{n}, \vec{v}_r$ jedinični vektori I to inverzni vektor svetla, vektor normale I vektor reflektovanog zraka respektivno. Na osnovu činjenice da su updani ugao θ i reflektujući ugao θ_r jednaki možemo izračunati reflektovani vektor na sledeći način:

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

gde je vektor \vec{w} pomoći vektor. Kao što se može videte proračun koji zahteva refleksija je jednostavan i ne zahteva ništa od dodatnih informacija sem normale u dатој таčki. Što se tiče refleksije tu su potrebana dva dodatna podatka a oni su indeksi prelamanja vazduha i vode, zato što se voda najčešće predstavlja tako da jedino sa vazduhom dolazi u kontakt.

Sad ćemo analizirati refrakciju, odnosno desni deo prethodne slike. Neka su vektori $\vec{v}, \vec{n}, \vec{v}_t$ jedinični vektori I to inverzni vektor svetla, vektor normale I vektor refraktovanog zraka respektivno. Odnos upadnog ugla θ i reflektujući ugla 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.

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

gde je vektor \vec{w}_t pomoći vektor. Kao i za refleksiju, proračun za refrakciju je jednostavan i od dodatnih informacija zahteva jedino indekse prelamanja vode i vazduha.

2.2.2. Fresnel-ov efekat

Još jedna jako bitna pojava koja se javlja prilikom interakcije vode sa zracima jeste Fresnel-ov efekat koji na govori koji deo upadnog 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:

$$R_s = \left(\frac{\sin(\theta_t - \theta)}{\sin(\theta_t + \theta)} \right)^2 = \left(\frac{n_1 \cdot \cos(\theta) - n_2 \cdot \cos(\theta_t)}{n_1 \cdot \cos(\theta) + n_2 \cdot \cos(\theta_t)} \right)^2$$

$$R_p = \left(\frac{\tan(\theta_t - \theta)}{\tan(\theta_t + \theta)} \right)^2 = \left(\frac{n_1 \cdot \cos(\theta_t) - n_2 \cdot \cos(\theta)}{n_1 \cdot \cos(\theta_t) + n_2 \cdot \cos(\theta)} \right)^2$$

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 refraktojući 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 vode i dolazi do neproporcionalnog osvetljenja tog jednog dela površi u odnosu na ostale. Na slici 2. je prikazan efekat kaustike.



Slika 2. kaustika

3. Progled 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 optičkih efekata.

3.1. Simulacija

U ovom delu će biti predstavljene tehnike koje se koriste prilikom simularanja kretanja vodene površine. Svaka od ovih tehnika podrazumeva da našu vodenu površinu modelujemo kao mrežu trouglova određene gustine. Od same gustine trouglova nam zavisi koliko će biti nivo detalja naše vodene površine. Pored računanja samih pozicija čvorova naše mreže, računamo 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 kordinatu y kordinatu 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 naše 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, ali problem je što grafička kartica ne razume ni jedno ni drugo. Zato ćemo ovu jednačinu rešiti numerički.

Najpre ćemo, umesto integracije leve strani, koristiti znanje iz fizike da nadjemo jednačinu koja opisuje kretanje određenog čvora. Prvo moramo uvesti prepostavku 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 trenuntku $t=0$ ($p(t_0)$), poziciju u trenutku $t=1$ ($p(t_1)$) i ubrzanje u trenutku $t=1$ ($a(t_1)$), možemo doći do pozicije u trenutku $t=2$ ($p(t_2)$).

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

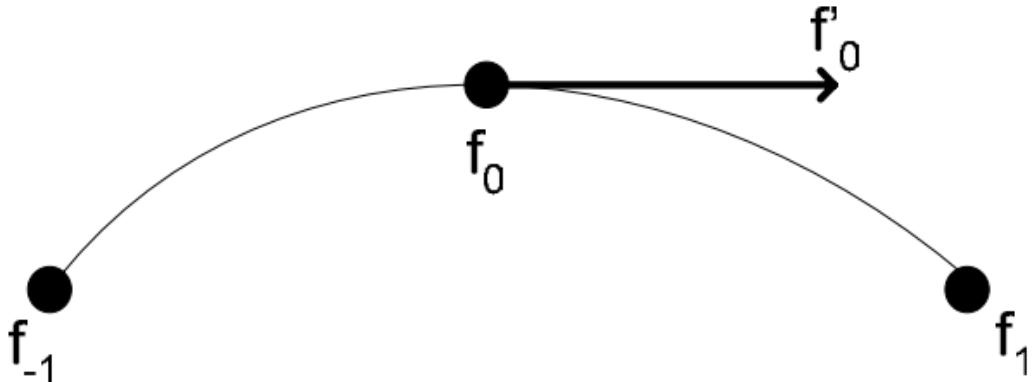
Međutim postoji određeni višak energije koji se dobija zbog greške prilikom primenjivanja ove metode. Ovo možemo rešiti tako što ćemo smanjiti brzinu pomoću nekog konstantog faktora. Označićemo ga sa α . Nakon toga data jednačina izgleda ovako.

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

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 pojednostavljenja 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 datoј slici data je kriva kroz tri tačke koje su na odstojanju dužine 1. Nama je potreban drugi izvod u tački nula (na slici 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:

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

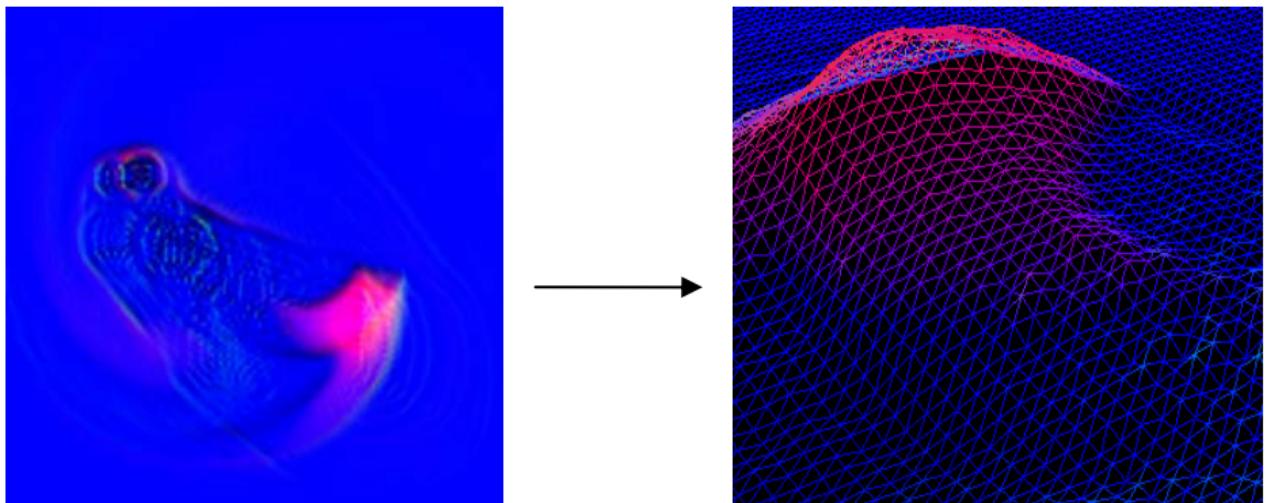
Ovo ćemo implementirati korišćenjem tekstura. Teksturu ćemo koristiti za smeštanje visina čvorova mreže i prilikom crtanja vodene površine pristupićemo datoj teksturi sa kordinatama određenim unapred. Nakon toga ćemo pročitanu vrednost iz teksture koristiti kao y kordinatu datog čvora. Naravno, pre svakog crtanja ćemo ažurirati datu teksturu koristeći gorenavede jednačine. Prilikom ažuriranja koristićemo tehniku koja se zove "ping-pong" teksture (engl. *ping-pong textures*). Odnosno nećemo imati jednu već dve teksture. Jedna će čuvati visine iz prethodne iteracije dok ćemo u drugu smeštati nove visine čvorova. Ukoliko sa p označimo kordinate teksture koje odgovaraju našem čvoru i ukoliko sa w i h označimo širinu i visinu naše teksture, kod na HLSL bi izgledao ovako:

```

dx = 1 / w;
dy = 1 / h;
heightP = sampleTexture(currentHeightTexture, p);
heightPLeft = sampleTexture(currentHeightTexture, p + offset(-dx, 0));
heightPRight = sampleTexture(currentHeightTexture, p + offset(dx, 0));
heightPUp = sampleTexture(currentHeightTexture, p + offset(0, dy));
heightPDown = sampleTexture(currentHeightTexture, p + offset(dx - dy));
previousHeightP = sampleTexture(previousHeightTexture, p);
acceleration = c * c * (heightPLeft + heightPRight + heightPUp + heightPDown - 4 * heightP);
newHeightP = 2 * heightP - previousHeightP + 0.5 * acceleration * dt * dt;

```

Naravno treba voditi računa o tome da dve koršćene teksture budu odgovarajućeg tipa, odnosno da budu dovoljno velike da mogu da smeste brojeve sa odgovarajućom preciznošću. Rezultat ovog koda dat je na sledećoj slici:



Slika 4. 2D jednačina talasa

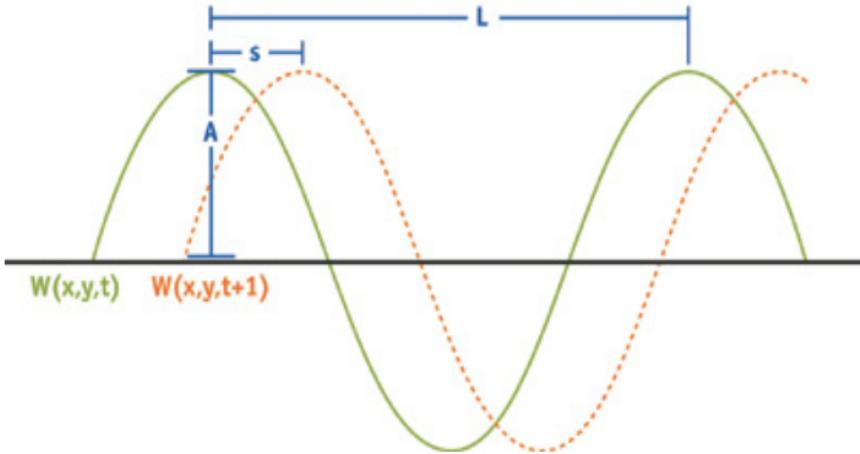
Naravno pored računanja same pozicije talasa potrebno je i sračunati normale. Najlakši način da to odradimo jeste da nađemo dva vektora koji počinju u našoj tekućoj tački i da njihovim vektorskim proizvodom dobijemo normalu. Ova dva vektora možemo dobiti iz pozicije naše tačke i nekih suseda. Naravno moramo voditi računa o tome da normala mora da bude jedinični vektor. Kod na HLSL za ovo bi izgledao ovako:

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

Naravno za veću 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 tiba gde bi na primer alfa kanal bio odvojen za visinu našeg čvora a crveni, zeleni i plavi kanal za x, y i z komponentu naše normale.

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. Ti parametri se mogu vide na sledećoj slici:



Slika 5. parametri sinusne funkcije[FerKir2004]

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 $\varphi = \frac{2\pi}{L} \cdot t$
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 y i vremena t i ona glasi:

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

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

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

Parametre samih talasa je najlakše izgenerisati 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 drugi skupom parametara. Ova tehnika je pogodna zato što nam je ponašanje talasa opisano konkretno funkcijom pa izračunavanje normala u čvorovima mreže nije komplikovano. Za ovo će nam biti potrebni vektor binormale i vektor tangente, koji se dobijaju kao parcijalni izvodi naše funkcije u x i y pravcu. Pozicija nekog čvora sa kordinatama (x, y) u horizontalnoj ravni se u određenom trenutku može izraziti kao:

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

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

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

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

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

Vektor normale dobija kao vektorski proizvod vektora binormal i vektora tangente i on glasi:

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

Naravno nakon ovoga je potrebno normalizovati dati vektor normal 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:

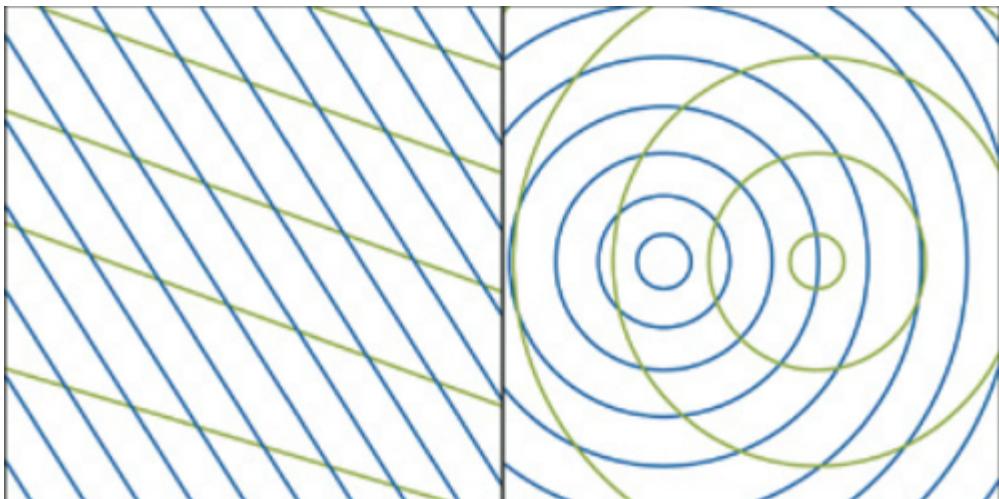
$$\frac{\partial}{\partial x} \cdot H(x, y, t) = \sum \frac{\partial}{\partial x} \cdot W_i(x, y, t) = \sum w_i \cdot D_i \cdot x \cdot A_i \cdot \cos(D_i \cdot (x, y) \cdot w_i + t \cdot \varphi_i)$$

Međutim, postoji jedna primedba navedena u [FerKir2004] a ona je da ovi talasi nemaju oštре vrhove kao što to imaju pravi talas. 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:

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

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

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



Slika 6. 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:

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

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:

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

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, tangente i normali. Oni su dati jednačinama:

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

Ove jednačine nisu "lepe" kao one iz prethodnog poglavlja ali su i dalje jednostavne za izračunavanje. Uslov za izbegavanje petlji na vrhovima talasa se zapravo može videti iz z komponente normale. Sve dok je suma manja od 1 petlje se neće formirati na vrhovima talasa, otuda i uslov naveden u prethodnom pasusu.

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

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

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

3.1.4. Perlin-ov šum

Šumovi se koriste u grafici za simulaciju prirodnih fenoma koji su ili prezahtevni za precizno izračunavanje i simuliranje ili su jednostavno i sam 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 ~~izgenerēšimo~~ nešto što je u grafici poznato kao ~~visinska~~ mapa (engl. *height map*). Ove mape se ~~dosta~~ koriste u grafici i najčešće se pomoću njih generiše teren. Međutim možemo je koristiti i za crtanje vodenih površina. Ova mapa predstavlja teksturu u kojoj je šum sačuvan kao tekstura koja predstavlja paletu sivih tonova. Svaki piksel date teksture će sadržati vrednost između 0 (crna boja) i 1 (siva boja). Prilikom crtanja naše vode površine, odnosno naše mreže trouglova uzorkovaćemo datu teksturu i vrednost koju pročitamo zapravo predstavlja visinu našeg čvora. Naravno pre same upotrebe pročitane vrednosti mi ćemo je skalirati pre izabranoj amplitudi talasa.

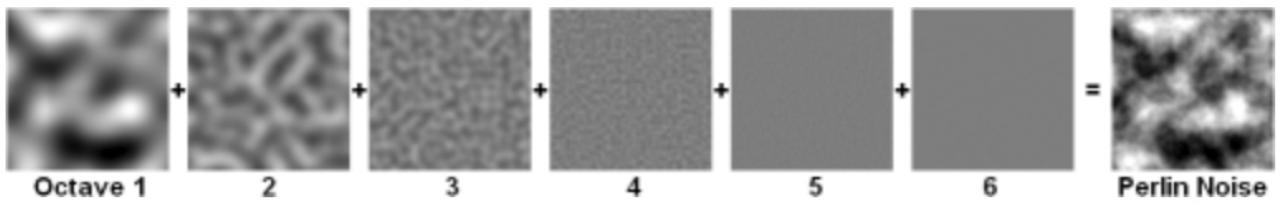
Da bi simulirali kretanje talasa ubacićemo pomeraj koji će zavisi od vremena. Ovo je najlakše implementirati tako što prilikom svakog ~~isrtavanja~~ mi pomeramo teksturne kordinate čvora za određeni broj piksela u bilo kom pravcu. Na taj način obezbeđujemo kretanje naše vodene površine.

Veoma velika pogodnost ove tehnike je ta što datu visinsku mapu treba ~~izgenerisati samo jednom pre početka same simulacije vodene površine~~. Takođe, data tekstura u koju će da smeštamo visinsku mapu ne mora da bude istih dimenzija kao naša mreža trouglova već može biti manja, a kasnije prilikom ~~iscrtavanja~~ naslagaćemo više visinskih mapa da bi dobili jednu veliku visinsku mapu koja je dimenzija naše mreže trouglova. Naravno, kao i kod svakog drugog šuma, jedna oktava daje mnogo velike oscilacije između susednih tačaka. Ovako izgleda jedna oktava šuma:



Slika 7. Perlin-ov šum

Ovaj problem se lako prevaziđa ukoliko saberemo više oktava ovog šuma. Tada dobijamo nešto ovako:



Slika 8. Suma 6 oktava Perlin-ovog šuma

3.1.5. Brze Fouier-ove transformacije (FFT)

Najrealističnije vodene površine 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 svog čvora naše mreže sa se može izraziti kao suma sinusoida sa kompleksnim i vremenski zavisnim amplitudama. Visina čvora je data formulom:

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

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 naše visinske mape, odnosno teksture, a L_x i L_z su dimenzije dela naše mreže na koji će da se primeni data visinska mapa jer, kao što sam naveo u prethodnom poglavlju, visinska mapa ne mora da bude istih dimenzija kao i sama mreža već možemo izgenerisati jednu manju visinsku mapu i kasnije spojiti više istih da dobije visinsku mapu dimenzija naše mreže. Visinska amplituda komponentu našeg talasa

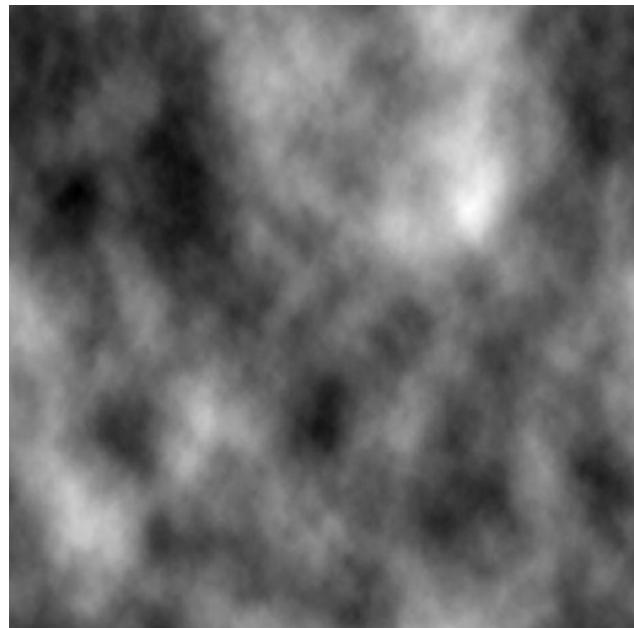
$\tilde{h}(\vec{k}, t)$ određena je strukturom naše vodene površine i više o tome se može naći u [Tes2001]. Jedina stvar koja nam ostaje jeste da nađemo normale u čvorovima naše mreže. Ova tehnika pruža način da se normala precizno odredi ali nam najpre treba odrediti nagib ϵ pomoću formule:

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

Nako ovoga normalu datog čvora naše mreže možemo dobiti kao:

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

Na sledećoj slici je dat izlgled visinske mape dobijene pomoću ove metode.



Slika 9. FFT visinska mapa

3.2. Optika

U ovom poglavlju biće prestavljene tehnike koje se koriste za simulaciju efekata do kojih dolazi usled interakcije vodene površine i svetlosnih zraka. Ovi efekti igraju mnogo veću ulogu u postizanju realističnog izgleda vode nego sama simulacija. Međutim njihova implementacija je ~~dosta~~ zahtevnija i to ne samo računarski zahtevna, već zahteva i mnogo veći napor programera. Iz toga razloga se dosta 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 površine.

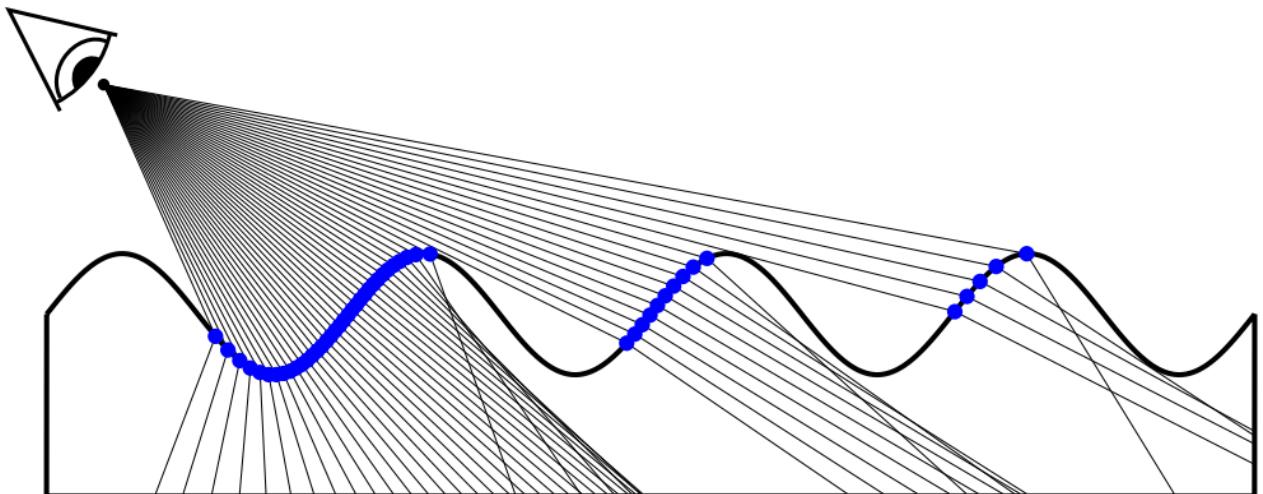
3.2.1. Refleksija i refrakcija

Refleksija i refrakcija su pojave kod kojih se zraci prelamaju prilikom interakcije sa vodenom površinom. Kao što smo videli u poglavlju 2.21 za proračun reflektovanog i refraktovanog zraka potrebna nam je normala u dатој тачки. Iz prethodnog poglavlja smo видели да прораčун нормала иде упоредо са самом simulација кретања водене површи. Nakon ovoga zraci nastavljaju dalje, односно reflektovani se odbijaju a refraktovani prolaze kroz водену површину и у неком trenuntku se ponovo sudaraju са objektima u sceni. Boje ових objekata određuju боју наше водене површи. Međutim, takvih zraka има previše да би се спровела физички тачна рачуница и добио реалан резултат. Овде ће бити изложена два метода помоћу којих можемо једноставније одредити боју наше водене површи.

Prvi метод се зове праћење зрака (engl. *ray tracing*[WikiE]). Користи се ~~dosta~~ често када ћелимо да постignemo реалистичније резултате. У основи ради тако што емитује зраке од камере ка воденој површи где се ти зracи reflektuju i refraktuju, kreirajući нове зраке. Затим прати путању нових зрака који или удара у неки објекат или поново дођу до водене површине где поново долази до refleksije ili refrakcije i kreiranja novih zraka. Postupak се понавља онолико puta koliko нам је потребно. Наравно, као што се може i zaključiti из описа, овај

metod previše zahtevan za grafičke kartice tako da se prilikom implementacije ovog algoritma uzimaju određena ograničenja.

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 bi izgledao ovako:



Slika 10. Algoritam praćenja zraka sa jednim nivoom rekurzije GalChi2006

Razlog za uvođenje ovog ograničenja je taj što grafičke kartice ne podržavaju rekurziju. Međutim, i pored ovog ograničenja ovaj metod daje dovoljno dobre rezultate. Kod ovog pristupa početna prepostavka je da su i voda i teren predstavljeni pomoću visinskih mapa 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 vodom 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 kombinujemo da bi dobili boju vodene površine, naravno poštujući Fresnel-ovu jednačinu. Pseudokod datog algoritma bi izgledao ovako:

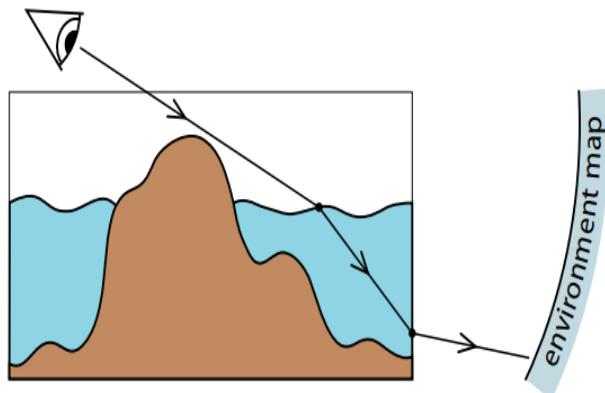
```

 $v \leftarrow \text{viewpoint}$ 
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  $p_w$ )
         $\text{pixel} \leftarrow \text{lightedGroundColor}(p_g, d)$ 
    else
         $d_t \leftarrow \text{refractedDirection}(d, n(p_w), \eta)$ 
         $d_r \leftarrow \text{reflectedDirection}(d, n(p_w))$ 
         $p_g \leftarrow \text{intersection}(\text{ground}, \text{ray}(p_w, d_t))$ 
         $p_e \leftarrow \text{intersection}(\text{ground}, \text{ray}(p_w, d_r))$ 
         $C_t \leftarrow \text{lightedGroundColor}(p_g, d_t)$ 
         $C_r \leftarrow \text{if } (p_e \text{ undefined})$ 
             $\text{envMap}(d_r)$ 
        else
             $\text{lightedGroundColor}(p_e, d_r)$ 
         $F \leftarrow \text{fresnelReflectivity}(d, n(p_w), \eta)$ 
         $\text{pixel} \leftarrow (1 - F) \times C_t + F \times C_r$ 

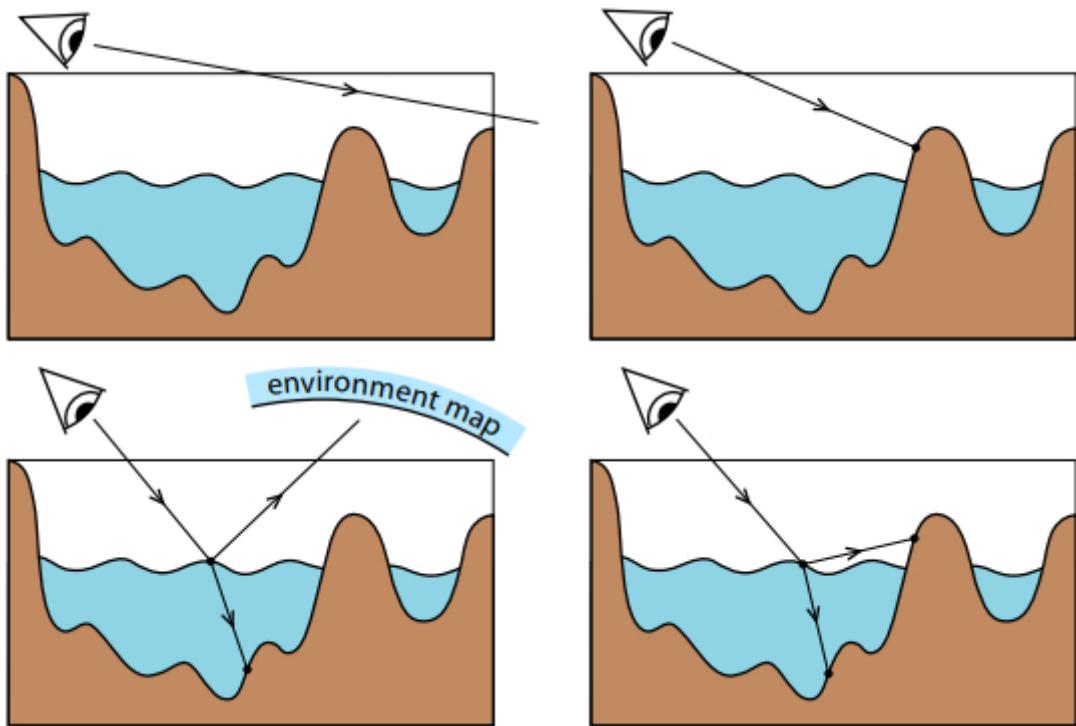
```

Slika 11. 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 možemo rešiti tako što ćemo imati neku statički definisanu teksturu koja će prestavljati "okolinu" koja nije deo našeg predela. Na osnovu putanje zraka možemo odrediti na kojem mestu uzorkujemo tu teksturu. U načelu algoritam pokriva sledeće slučajeve:



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



Slika 13. Različite putanje zraka [GalChi2006]

Nasuprot ovom algoritmu, koji iako računarski zahtevan daje odlične rezultate, postoji i pristup koji korišćen u [Tur2014] i [Dha]. Ova tehnika podrazumeva da u toku svakog iscrtavanja pravimo dve teksture, jednu reflektujuću i jednu refraktujuću. Odnosno da prilikom svakog iscrtavanja, pre nego što krenemo u iscrtavanje naše vodene površine, prvo iscrtamo 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 reflektujeće teksture moramo primeniti refleksionu transformaciju koja će svet invertovati kao odraz u ogledalu, gde je naše ogledalu zapravo horizontalna ravan u visini vodene površine. Ova transformacija je ništa drugo nego skaliranje sa faktorom -1 . Refraktujuću teksturu crtamo na uobičajeni način. Nakon što izgenerišemo ove teksture možemo preći na crtanje vode. Prilikom iscrtavanja koristićemo projektujuće teksturiranje, tj. datu teksturu ćemo projektovati na vodenu površinu kao što projektor prikazuje sliku na zidu. Samo "projektor" uglavnom ima istu poziciju kao i sama kamera.

3.2.2. Fresnel-ov efekat

Kao što je navedeno u poglavlju 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 normale vodene površi u tački interakcije manji to je dominantnija boja boja dobijena od refruktovanog zraka, a što je ugao između vektora pogleda i normali vodene površine u tački interakcije veći to je dominantnija boja dobijena

od reflektovanog zraka. Sama jednačina ~~oduzima~~ dosta računarske snage grafičke kartice pa se zato često koriste aproksimacije. Jedna takva aproksimacija je navedena u [Fle2007] i ona računa odnos između boja na sledeći način:

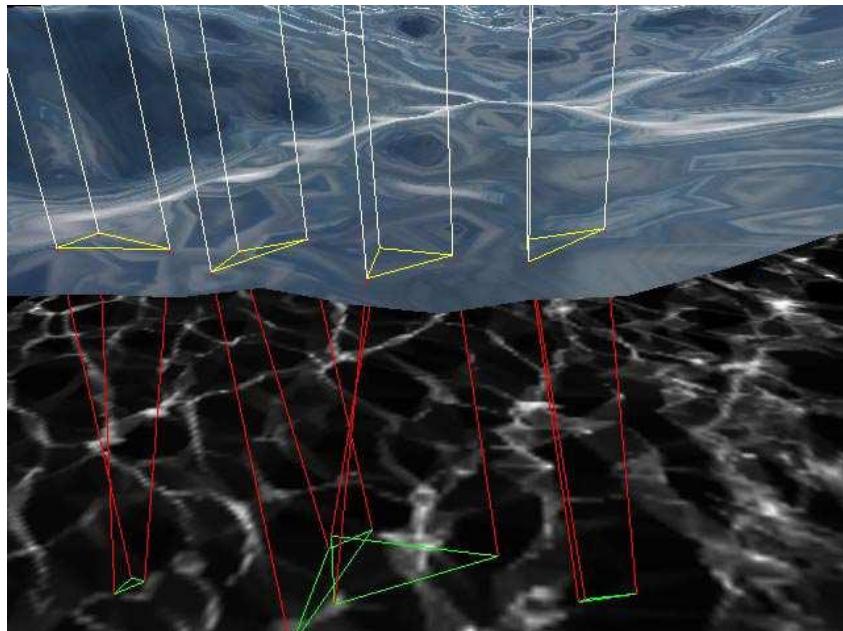
$$F = \frac{1}{1 + \cos \theta^8}$$

Drugi pristup određivanju ovog odnosa **jeste imamo 1D teksturu** u koju ćemo u nekom od inicijalizacionih koraka izračunati ovaj odnos za određeni broj uglova. Ove vrednosti ćemo u datu teksturu smeštati kao u heš tabelu gde će ključ biti sam ugao. Kasnije prilikom iscrtavanja ćemo pristupati ovo heš tabeli na osnovu ugla između vektora pogleda i normal vodene površine u dатој tački da bi pročitali odgovarajuću vrednost.

3.2.3. Kaustika

Kaustika je jedan od najinteresantnijih efekata koje možemo prikazati prilikom crtanja vode, međutim ~~jedan od najtežih~~. Kao što je navedeno u poglavlju [2.2.3](#) kaustika predstavlja interferenciju refraktovanih zraka koji čini određene predele ispod vodene površine svetlijim. Kaustika je nemoguće iscrtati 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 površina predstavljena kao mreža trouglova i da je zemlja ispod vodene površine paralelna vodenoj površini. Za svaki čvor naše mreže računamo upadni zrak svetla i odgovarajući refraktovani zrak. Refraktovani zrak će se u nekom trenutku sudsariti sa zemljom ispod vodene površine pa je takođe potrebno naći tačku u kojoj se ovo dešava. Ovo zapravo rezultuje u projektovanju trouglava nešte mreže na zemlju ispod vodene površine i to je prikazano na sledećoj slici:



Slika 14. 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 normala nše vodene površi u odgovarajućoj tački, L je vektor od čvor mreže trouglova koja predstavlja vodenu površinu, a a_c i a_s su površine projektovanog i površinskog trougla. Sa obzirom na to da je zemlja paralelna vodenoj površini 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 neću ulaziti u detalje ove tehnike. Načelno, emituje zrake iz izvora svetla ka vodenoj površini i nakon svake interakcije računamo refraktovan zrak. Nakon ovoga pratimo refraktovan zrak i ukoliko se on preseca sa zemljom ispod vodene površi snimimo kordinate preseka kao i doprinos datog fotona celokupnom osvetljenju tog dela zemlje u određeni keš, koji je načešće jedna tekstura. Nakon toga moram primeniti odgovarajuće filtere tako da izoštimo 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 našeg terena.

4. Implementacija

U ovom poglavlju ču predstaviti moje rešenja datog problema kao i njegovu implementaciju. Rešenje će biti podeljene po malo drugačijem principu nego prethodna poglavlja. Najpre ču opisati kako sam implementirao refleksiju i refrakciju, zatim kako sam simulirao kretanje vodene površine, zatim fresnel-ov efekat, spekulativno osvetljenje na površini vode i na kraju kako sam implementirao efekat kaustike. Razlog za to je taj što je u mom rešenju vodena površina samo jedan pravougaonik, a ne mreže 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 odrđena na programskom jeziku Java. Za implementaciju biblioteka korišćena je biblioteka OpenGL verzije 4.2, tačnije omotačka biblioteka JOGL koja omogućava pristup OpenGL is koda napisanog u Javi.

4.1. Refleksija i refrakcija

Refleksija i refrakcija su u ovom rešenju implementirani na jednostavan način. Prilikom iscrtavanja svake slike a pre samog crtanja vodene površine prvo formiramo dve nove teksture, odnosno reflektujuću i refraktujuću tako što u njih iscrtavamo kompletну scenu bez same vodene površine. Najpre crtamo reflektujuću ali tako pre samog iscrtavanja invertujumu celu scenu, odnosno primenimo transformaciju "preslikavanja u ogledalu" gde je ogledalo naša xz ravan. Ovo može da se odradi transformacijom skaliranja ali je ovde odrđeno na jednostavniji način i to pomeranjem pozicije kamere pre crtanja i vraćanje u originalni položaj nakon crtanja. Poziciju kamere pomeramo tako što samnjimo njenu y kordinatu za duplu visinsku razliku između vode i kamere i invertujemo "pitch" kamere, odnosno invertujem ugao rotacije oko x ose.

Za iscrtavanje glavne slike koja se nama prikazuje na ekranu koristi se bafer slike (engl. *framebuffer*[OpenGLWikiA]). OpenGL pruža opciju da pored glavnog bafera koji čuva ono što se prikazuje nama na ekranu kreiramo određen broj novih (broj ovih bafera je ograničen karakteristikama grafičke kartice) i da crtamo našu sliku u njih umesto u glavni. Ovaj bafer slike je zapravo apstrakcija. On se sastoji iz više komponenti od kojih su dve glavne bafer dubine (engl. *depth buffer*) i bafer boje (engl. *color buffer*). Pre samog iscrtavanja na ekran, cela scena se najpre crta u ova dva bafera pa tek onda na ekran. Takodje, nasi novi baferi slike mogu imati ova dva bafera. Sta vise, umesto ova dva bafera možemo koristiti teksture koje kasnije možemo koristiti na drugim mestima u našem programu. To ćemo i raditi u našem program. Kreiraćemo nova dva bafera slike i jedan ćemo koristiti za iscrtavanje reflektujuće a drugi za iscrtavanje refraktujuće teksture. Scenu iscrtavamo na isti način kao i do sada bez ikakvih dodatnih izmena. Kasnije ćemo ove dve teksture koristiti prilikom implementacije refleksije i refrakcije zraka.

Najpre je potrebno stvoriti dati bafer slike i on se kao i svi ostali objekti u OpenGL biblioteci stvara pozivom *glGen* metode i to *glGenFrameBuffer* koja vraća celobrojni identifikator novokreiranog bafera slike. Nakon toga je potrebno dodati datom baferu slike bafer boje i bafer dubine sa metodama *glFrameBufferTexture2D* gde kao parametre prosleđujemo celobrojni identifikator tekstura koje će da nam služe kao bafer boje i bafer dubine. Sada je naš bafer kompletan, odnosno može da se koristi za iscrtavanje i potrebno je da pre svakog iscrtavanja kažemo OpenGL biblioteci da hoćemo da sve što se bude

iscrtavalo bude smešteno u novi bafer slike. To radimo pozivom *glBindFramebuffer* kojog prosleđujemo identifikator našeg bafera slike. Sve što se su bude iscrtavalo nakon ovog poziva biće smešteno u ovaj bafer slike.

Pored ovoga je potrebno odraditi ješ jedno stvar a to je odsecanje određenog dela scene. Naime, neće se cela scena reflektovati na vodenoj površini već samo deo iznad nje. Takodje, neće se cela scena refraktovati na vodenoj površini već samo deo ispod nje. Tako da je potrebno na neki način ukloniti deo scene koji nam ne treba. Srećom, OpenGL pruža jednostavan način za postizanje ovog odsecanja u vidu odsecajućih ravni (engl. *clipping plane*). Da budem precizniji ono što nam OpenGL pruža je niz ugrađenih promenljivi koji se zove *gl_ClipDistance*. Ova promenljiva je dostupna u *vertex shader* programu i nju smeštamo rastojanje tačke od odsecajuće ravni. Ukoliko je rastojanje negativno tačka se neće iscrtati, ukoliko je pozitivno hoće. Sada jedino što je potrebno da se odradi jeste da se data mogućnost OpenGL-a omogući pozivom metode *glEnable* sa parametrom *GL_CLIP_DISTANCE0* (ovime omogućavamo prvu odsecajuću ravan) i da se data ravan prosledi *shader* programu kao uniformna varijabla. 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 centra kordinatnog sistema. Da bi dobili rastojanje date tačke od odsecajuće ravni potrebno je samo da pomnožimo poziciju naše tačke u kordinatnom sistemu sveta sa ova 4 koeficijenta ravni, koje prosleđujemo kao *vec4* uniformnu promenljivu. Kod *shader-a* za ovo bi izgledao ovako:

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

Slika 15. Kod za implementaciju odsecajuće ravni.

Sve ostalo je isto kako kad bismo normalno crtali našu scenu. Kao rezultat dobije reflektujuću teksturu gde je sve iznad vodene površine iscrtano i refraktujuću teksturu gde je sve ispod vodene površine iscrtano. One bi izgledale ovako:



Slika 16. Reflektujuća tekstura



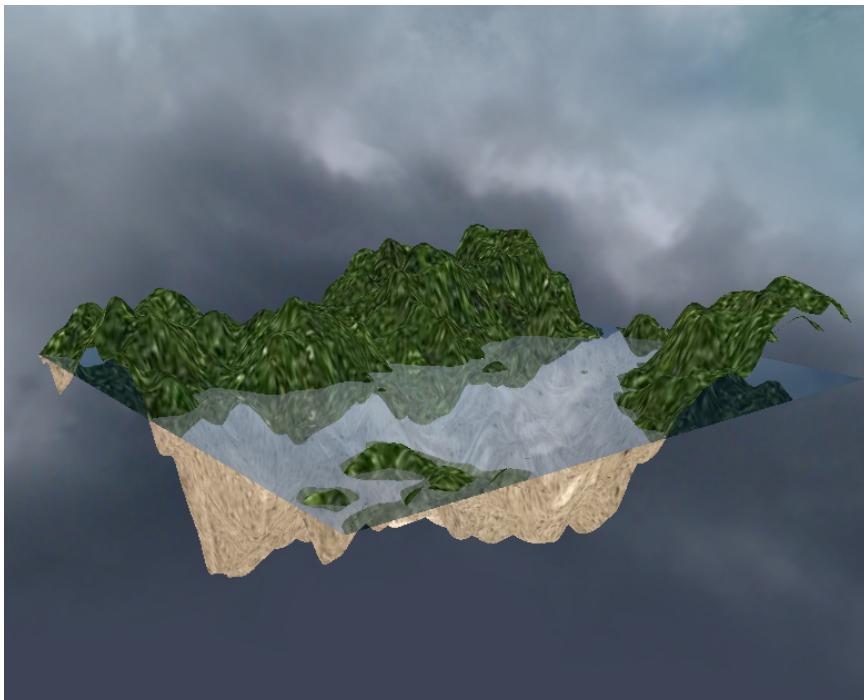
Slika 17. Refraktujuća tekstura

Nakon ovoga je potrebno ove dve teksture primeniti prilikom crtanja vode da bi se dobio efekat refleksije i refrakcije. Primjenjujemo ih tako što koristimo projektujuće teksturiranje odnosno uzorkujemo teksture tako da dobije efekat projektor-a koji prikazuje datu teksturu, gde je pozicija datog projektor-a ista kao pozicija kamere. To je u stvari jako jednostavno jer sve što treba da uradimo jeste da prosledimo *clip space* kordinate iz *vertex shader-a* u *fragment shader* i da sami odradimo perspektivno deljenje sa *w* kordinatom naše tačke. Kasnije ćemo koristiti *x* i *y* kordinate dobijene perspektivnim deljenjem za uzorkovanje teksture. Jedino o čemu treba voditi računa jeste to da je u prirodi prikaz nekog objekta na vodenoj površini prilikom refleksije invertovan pa stoga i mi moramo invertovati *y* kordinatu kojom će da uzorkujemo našu teksturu. Nakon toga potrebno je date boje kombinovati uz eventualno dodavanje plave boje da bi se dobio realističan prikaz vodene površine. Na ovaj način će se na vodenoj površini prikazivati i refleksiona i refrakciona tekstura. Kod za ovo bi izgledao ovako:

```
vec2 refractionTextureCoordinates = (clipSpaceCoordinates.xy / clipSpaceCoordinates.w) / 2 + 0.5;
vec2 reflectionTextureCoordinates = vec2(refractionTextureCoordinates.x, 1 - refractionTextureCoordinates.y);

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

Slika 18. Kod za perspektivno deljenje i uzorkovanje teksture
Rezultat izgleda ovako:



Slika 19. Refleksiona i refrakciona tekstura

4.2. Simulacija

Za simulaciju kretanja vodene površine nije primjenjen nijedan fizički ili statistički model navede u poglavlju [3.1](#) već se koristila metoda koja samo daje privid kretanje vodene površine. Za implementaciju simulacije sam koristio dUDV teksturu, odnosno RGBA tekstuру koja koristi samo crveni i zeleni kanal. Vrednosti iz ovih kanala sam koristio da bi ubacio distorziju priliko 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 bih 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 nazvana *moveFactor* koje se menja prilikom iscrtavanja svake slike i koju koristim kao pomeraj prilikom uzorkovanje dUDV tekstuure. Na kraju, sama distorzija smeštena u ovoj tekstuuri je mnogo velika, odnosno vrednosti u crvenom i zelenom kanalu su mnogo velike. Zato sam ubacio dodatanu uniformnu promenljivu koja predstavlja faktor za skaliranje distorzije i koja u suštini određuje jačinu prividnih talasa pa se yato i zove *waveStrength*. Sledeća slika prikazuje kod za implementaciju ovog efekta:

```

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;

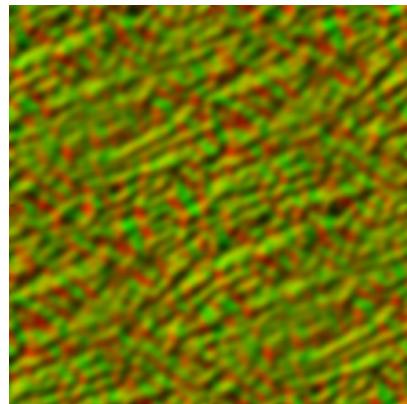
vec2 reflectionTextureCoordinates = vec2(refractionTextureCoordinates.x, 1 - refractionTextureCoordinates.y);
reflectionTextureCoordinates += totalDistortion;

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

```

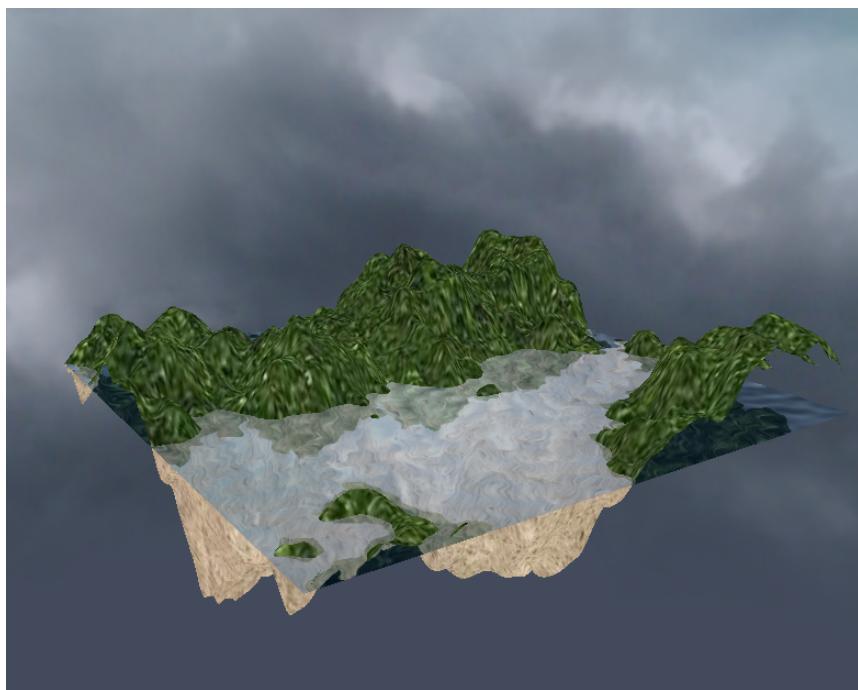
Slika 20. Kod za ubacivanje distorzije kod uzorkovanja reflektujuće i refraktujuće tekstuure

Naredna slika prikazuje samu dUdV teksturu koja je korišćena prilikom ove implementacije:



Slika 21. dUdV tekstura

Narednal slika prikazuje konačan efekat:

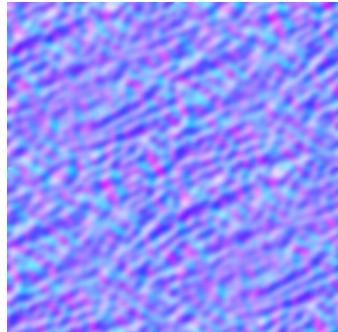


Slika 22. Konačan efekat nakon primenjivanja distorzije na reflektujuću i refraktujuću teksturu

4.3. Fresnel-ov efekat

Sledeća stvar koju sam dodao bila je Fresnel-ov efekat. Kao što je navedeno u poglavlju [3.2.2](#) Fresnel-ov efekat je komplikovan za ekgzaktnu računica zato se dosta često koriste aproksimacije koje zavise sam od ugla pogleda. U mojoj implementaciji sam koristio sličnu aproksimaciju. Ona zavisi od ugla između vektor pogleda i vektora normal u dатој тачки. Ukoliko su oba vektora jediničни тaj ugao je lako dobiti i računa se само jedним pozivom *dot* funkcije коју pruža OpenGL. Што је угао између ова два vektora manji то је

dominantnija boja dobijena od refraktojuće teksture a što je ugao između ova dva vektora veći to je dominantnija boja dobijena iz reflektujuće teksture. Naravno, za ovo nam je potrebna normala u datoj tački. Za dobijanje normale u određenoj tački koristiće mapu normala (engl. *normal map*). Ona je data na sledećoj slici:



Slika 23. Mapa normala

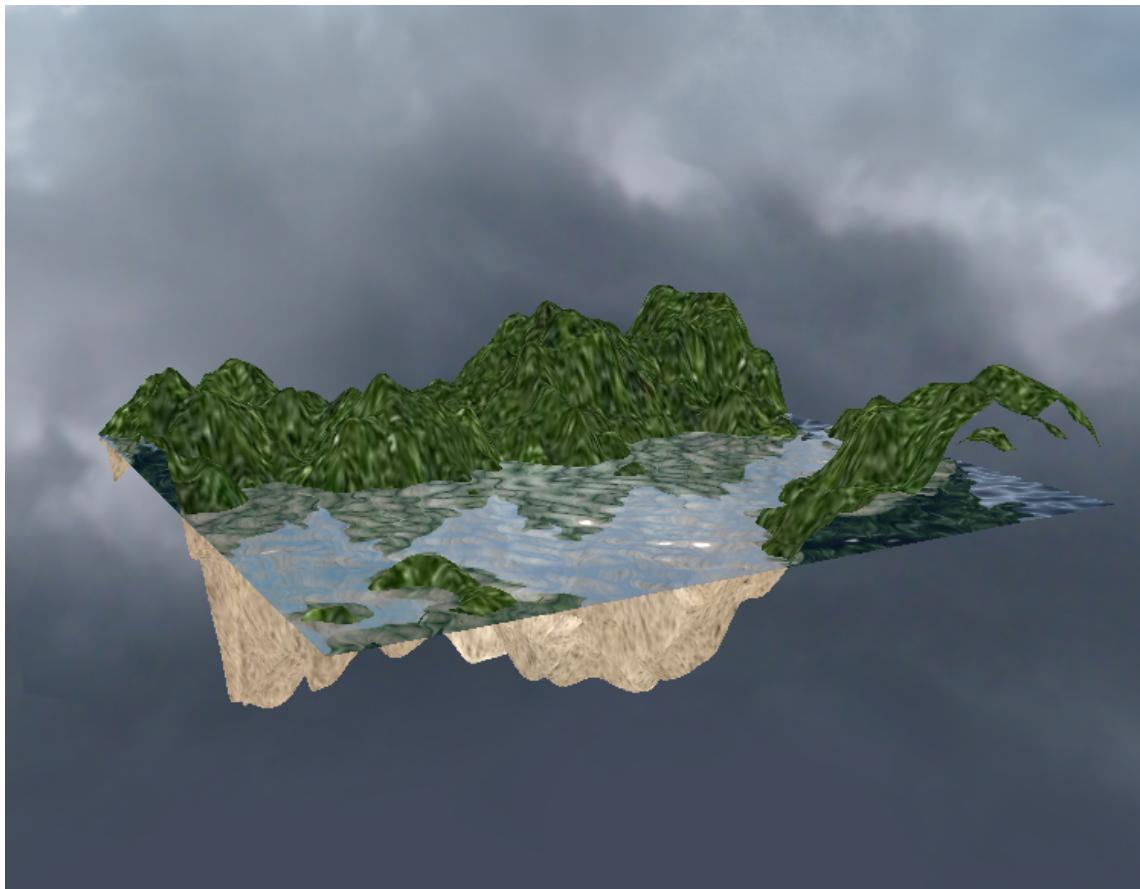
Ta mapa je zapravo RGBA tekstura, gde crveni, zeleni i plavi kanal svakog predstavlja x , z i y komponente vektora normale, respektivno. Kao što možemo primetiti ova tekstura je plavičasta što i ima smisla jer su sve normale usmerene na gore pa je plavi kanal, koji predstavlja y komponentu, dominantan. 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 $[-1, 1]$. Međutim tu nailazimo na sledeći problem. Ovako dobijene normale ne moraju da budu jednični vektori što je nama neophodno. Tako da ćemo pre upotrebe normalizovati našu normalu da bi dobili jednični vektor.

Još jedna stvar o kojoj moram da vodimo računa jeste uzorkovanje same mape normala. Mapa prikazana na prethodnoj slici odgovara dUDV teksturi iz prethodnog poglavlja, odnosno odgovara teksturi pomoću koje simuliramo talasa. Tako da normalu moramo uzorkovati na istim mestima kao i dUDV teksturu, naravno uz odgovarajući pomeraj u toku vremena, da bi dobili što realističnije efekte. Ovo se može videti iz sledećeg dela koda koda:

```
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);
```

Slika 24. Kod za korišćenje mape normala

Efekat koji dobijamo nakon ovoga dat je na sledećoj slici:



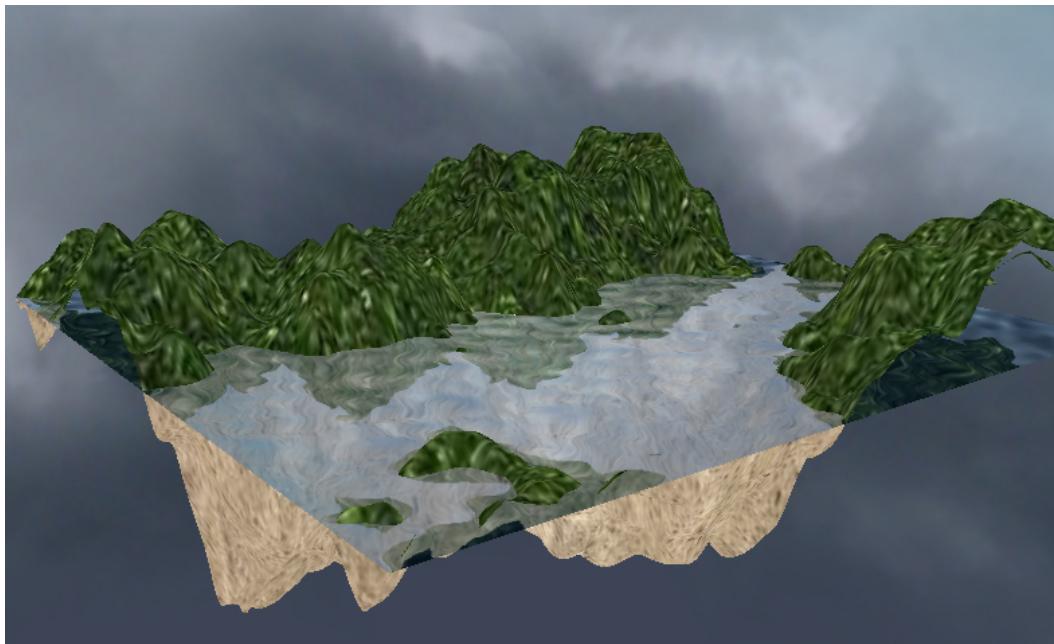
Slika 25. Fresnel-ov efekat

Ovakva slika ne predstavlja realističan prikaz vode. Kao što se može videti normale, iako sve uperene na gore, su raštrkane. To možemo popraviti tako što ćemo pomnožiti y komponentu naše normale sa određenim faktorom koji sam u kodu nazva *normaEqualizationFactor*. Nakon toga kada budemo normalizovali naš vektor normal povećana y komponenta će doprineti tome da normale ne budu toliko raštrkane već većim delom usmerene ka gore. Kod koji postiže ovo izgleda ovako:

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

Slika 26. Popravljanje normala

Konačni efekat dobijen sa ovakvim računanje normala prikazan je na sledećoj slici:



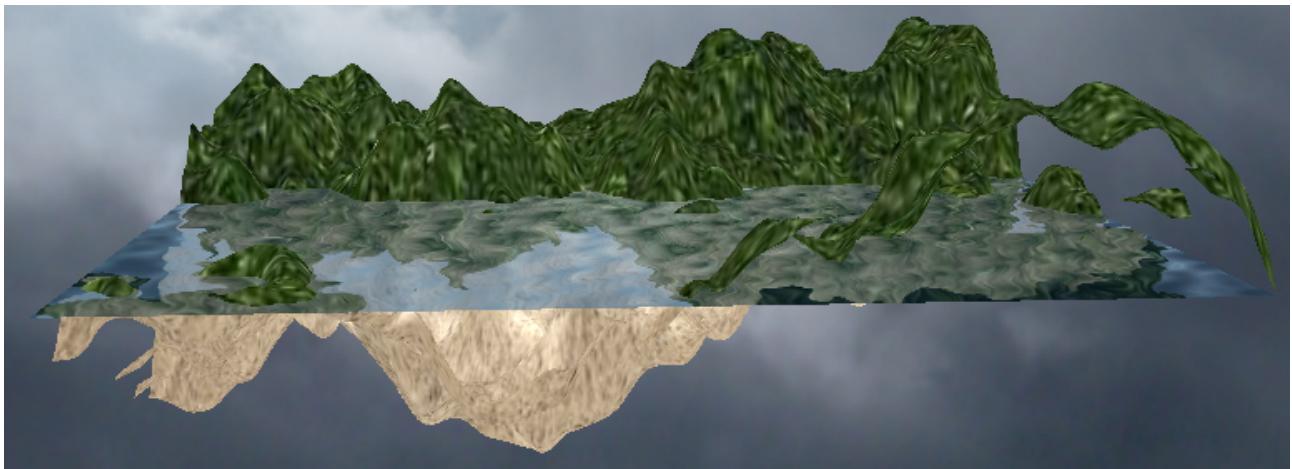
Slika 27. Popravljeni fresnel-ov efekat

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



Slika 28. Refrakcija sa Fresnel-ovom jednačinom

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



Slika 29. Refleksija sa Fresnel-ovom jednačinom

4.4. Spekulativno osvetljenje

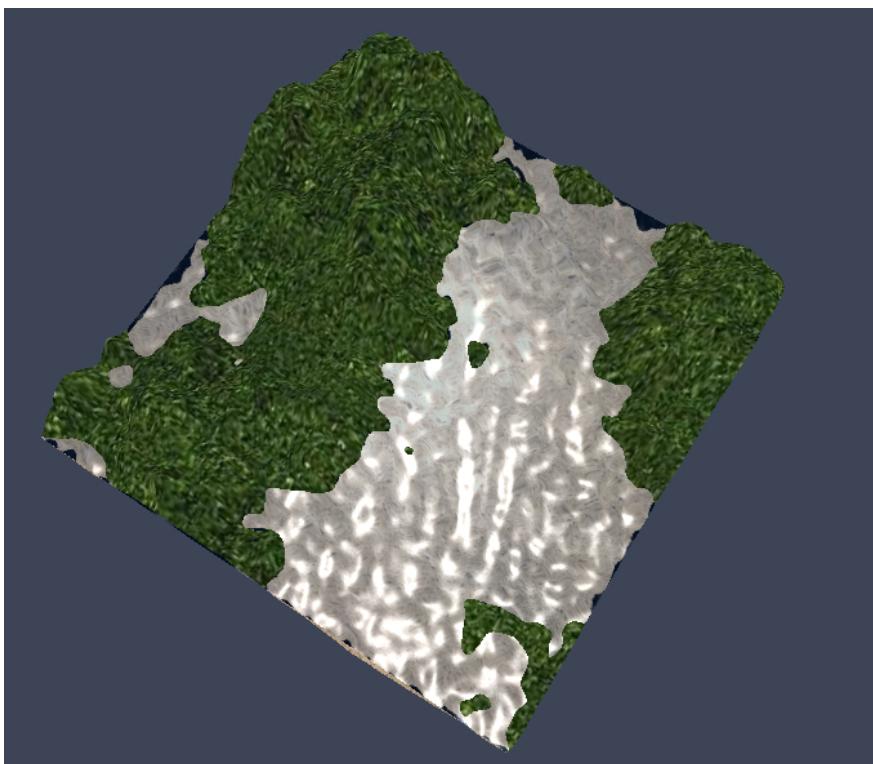
U cilju što realističnijeg izgleda, dodao sam i spekulativno osvetljenje vodene površine. Po uzoru na Phong-ov model senčenja, za svaki piksel vodene površine nađen je upadni svetla koji je u toj tački reflektovan na osnovu normale u toj tački. Nakon toga sam izračunao ugao između reflektovanog zraka i vektora pogleda. Ovo je jednostavno odraditi jer oba vektora možemo svesti na jedinične vektore. Nakon toga sam taj ugao podigao na određeni stepen i pomnoži sa korigujućim faktorom gde su obe vrednosti

određene samom karakteristikom veden površine. Međutim, do ovih faktora sam došao probom. Kod koji ilustruje primenu spekulativnog osvetljenja dat je na sledećoj slici:

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

Slika 30. Kod za spekulativno osvetljenje

Nakon ovog proračuna, *specularHighlights* vektor je jednostavno dodat pomešanoj boji reflektujuće i refraktojuće teksture. Efekat ovog se može videti na sledećoj slici:



Slika 31. Efekat spukularnog osvetljenja

4.5. Kaustika

Kao što je navedeno u poglavlju [3.2.3](#) kaustika je efekat koji jekao teško implementirati i gotovo je nemoguće postići to bez neke vrste praćenja zraka. U mom rešenju sam upotrebio ideju koja je slična onoj opisanoj u [FerKir2004]. Međutim ova ideja pravi jednu jako grubu pretpostavku a to je da na kaustiku utiču zraci koji se refraktuju pod pravim uglom. Na ovaj način sa svakog piksela zemlje ispod vodene površine možemo pratiti zrak do površine i odrediti originalni upadni zrak na osnovu refraktovanog. Ovaj zrak ne potiče od svetlosnog ivora pa je nakon toga je potrebno odrediti ugao između vektora upadnog zraka i zraka koji potiče od svetlosnog izvora. Što je ovaj ugao manji to 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 na

normalu vodene površine 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 refraktovanih 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. Postupa 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 upadni vektor:

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

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

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

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

Ukoliko su vektor normale i refraktovani vektor jedinični vektori kosinus ugla između njih se dobija jednostavnim množenje ta dva vektor. Naravno treba voditi računa da upadni vektor dobijen ovom računicom nije jedinični vektor pa ga je potrebno normalizovati. Sledeća stvar koju treba odraditi jeste naći ugao između vektora upadnog zraka i vektora koji se prostire od date tačke do izvora svetla. Ponovo, ukoliko su oba vektora jedinični ovaj ugao je lako naći. Nakon toga to možemo da iskoristimo na isti način na kod spekularnog osvetljenja, odnosno da određenim faktorima postignemo vizuelno zadovoljavajući efekat. Jedna razlika u kodu je početna pretpostavka smera vektora upadnog zraka, odnosno pretpostavljen je da upadni vektor ima suprotan smer od onog na slici 1. Kod za računanje kaustike dat je na sledećoj slici:

```

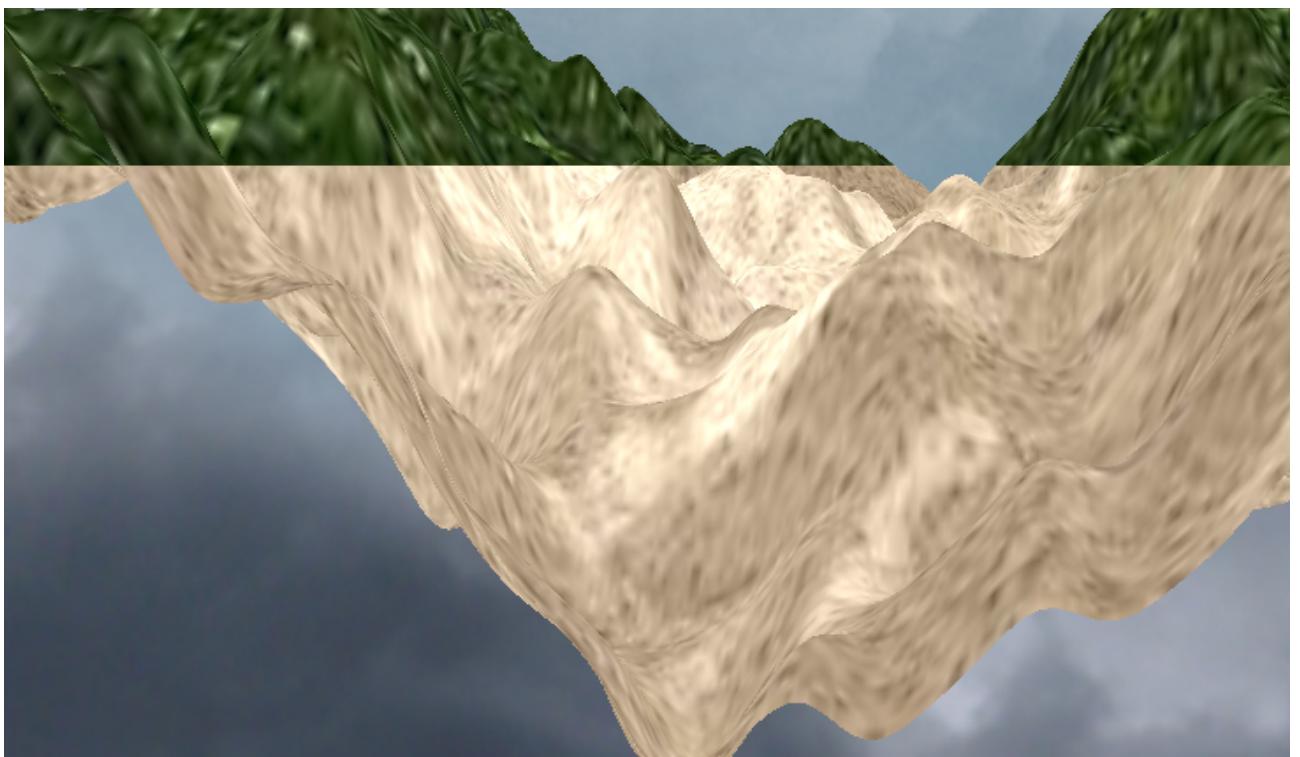
vec3 refractedRay = vec3(0, -1, 0);
float theta = dot(-normal, refractedRay);
vec3 incidentRay = eta * refractedRay + normal * (eta * theta - eta * sqrt(1 - eta * eta * (1 - theta * theta)));
incidentRay = normalize(-incidentRay);

vec3 normalizedToLightVector = normalize(lightPosition - waterSurfaceCoordinates);
float specular = max(dot(normalizedToLightVector, incidentRay), 0);
specular = pow(specular, shineDamper);
vec3 specularHighlights = lightColor.rgb * specular * lightReflectivity;

```

Slika 32. Kod za implemtaciju kaustike

Konstanta *eta* predstavlja odnos indeksa refrakcije vode i vazduha i ovde ima konstantu vrednost 0.75. Efekat koji se postiže ovim kodom dat je na sledećoj slici:



Slika 33. Kaustika

Kao što se može videti sa slike efekat nije savršen ali je metoda dosta jednostavna i brza.

5. Zaključak

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

Glavni problemi na koje nailazimo su simulacija kretanje vodene površine i implementacija optičkih efekata kod kojih refleksija i refrakcija najviše doprinose realistično izgledu. Pored ova dva efekta dodata je kaustika kao još jedan efekat koji doprinosi realistično izgledu, međutim ne tako mnogo kao prva dva. Pored problema opisane su razne tehnike za njihovo rešavanje koje ide lakših ka težim, odnosno od manje računarski zahtevnih ka više računarski zahtevim. Na kraju je data jedna implementacija koja ilustruje sve te probleme i neke od načina na koji se oni rešavaju.

Međutim ovo nije celokupna priča jer crtanje vodenih površina je jako zahtevan posao i svakim danom nailazimo na nove tehnike kojim doprinosimo realistično izgledu vode. Sledeći korak u daljem istraživanju bi bio da se detaljnije istraže tehnika praćenja zrakova i njeno korišćenje prilikom implementacije optičkih efekata. Pored same tehnike potrebno je istražiti i to koliko je ona primenjiva jer, kao što sam naveo u radu, ona je jako zahtevna i današnjih 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 "božji zraci", pena 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%E2%80%93Stokes_equations, 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 Universität 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

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

[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.....	8
Slika 5. parametri sinusne funkcije[FerKir2004].....	9
Slika 6. Usmereni (levo) i kružni (desno) talasi[FerKir2004].....	10
Slika 7. Perlin-ov šum.....	12
Slika 8. Suma 6 oktava Perlin-ovog šuma.....	13
Slika 9. FFT visinska mapa.....	14
Slika 10. Algoritam praćenja zraka sa jednim nivoom rekurzijeGalChi2006.....	15
Slika 11. Pseudo kod algoritma praćenja zrakaGalChi2006.....	16
Slika 12. Zrak izlazi van ranica nakon refrakcije[GalChi2006].....	16
Slika 13. Različite putanje zraka[GalChi2006].....	17
Slika 14. Projektovani trouglovi[Fle2007].....	18
Slika 15. Kod za implementaciju odsecajuće ravni.....	21
Slika 16. Reflektujuća tekstura.....	21
Slika 17. Refraktujuća tekstura.....	22
Slika 18. Kod za perspektivno deljenje i uzorkovanje teksture.....	22
Slika 19. Refleksiona i refrakcionala tekstura.....	23
Slika 20. Kod za ubacivanje distorzije kod uzorkovanja reflektujuće i refraktujuće teksture.....	23
Slika 21. dUDV tekstura.....	24
Slika 22. Konačan efekat nakon primenjivanja distorzije na reflektujuću i refraktujuću teksturu...24	24
Slika 23. Mapa normala.....	25
Slika 24. Kod za korišćenje mape normala.....	25
Slika 25. Fresnel-ov efekat.....	26
Slika 26. Popravljanje normala.....	26
Slika 27. Popravljeni fresnel-ov efekat.....	27
Slika 28. Refrakcija sa Fresnel-ovom jednačinom.....	27
Slika 29. Refleksija sa Fresnel-ovom jednačinom.....	28
Slika 30. Kod za spekulativno osvetljenje.....	28
Slika 31. Efekat spukularnog osvetljenja.....	29
Slika 32. Kod za implementaciju kaustike.....	30
Slika 33. Kaustika.....	31