

**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 koji se javljaju prilikom crtanja kao i tehnike koje se mogu koristiti za rešavanje ovih problema. Problemi će biti izloženi u dve kategorije, a one su: simulacija kretanja vodene površi i optički efekti do kojih dolazi prilikom interakcije vodene površi i svetlosnih zraka. Od ovih efekata biće pokrivena refleksija i refrakcija, kao dva najbitnija efekta koji doprinose realističnom izgledu vode. Pored ovih, takođe će biti pokriven efekat kaustike 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 ilustruje rešenja na 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.....	4
2.2.1. Kaustika.....	4
3. Pregled postojećih rešenja.....	5
3.1. Simulacija.....	5
3.1.1. Simulacije pomoću aproksimacija.....	5
3.1.1.1. Aproksimacija pomoću sume sinusa.....	5
3.1.1.2. Gerstner-ovi talasi.....	7
3.1.2. Simulacija pomoću mapa visina.....	9
3.1.2.1. 2D jednačina talasa.....	11
3.1.2.2. Perlin-ov šum.....	14
3.1.2.3. Brze Fouier-ove transformacije (FFT).....	16
3.2. Optika.....	17
3.2.1. Refleksija i refrakcija.....	17
3.2.2. Fresnel-ov efekat.....	21
3.2.3. Kaustika.....	24
4. Implementacija.....	26
4.1. Refleksija i refrakcija.....	26
4.2. Simulacija.....	29
4.3. Fresnel-ov efekat.....	31
4.4. Spekularno osvetljenje.....	35
4.5. Kaustika.....	36
5. Zaključak.....	41
Literatura.....	42
Slike.....	43
Listinzi.....	44
Dodatak A - Kod <i>vertex shader-a</i> za implementaciju vodene površi.....	45
Dodatak B - Kod <i>fragment shader-a</i> za implementaciju vodene površi.....	46
Dodatak C - Kod <i>vertex shader-a</i> za implementaciju površi ispod vodene površi.....	48
Dodatak D - Kod <i>fragment shader-a</i> za implementaciju površi ispod vodene površi.....	49

# 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 nije moguće prikazati ih onakve kakvi oni jesu u prirodi. Od svih fenomena koji se obrađuju, oni koji daju vodenim površima njihov izgled zahtevaju najviše napora i truda. Rezultat ovih napora se može videti kako u video igramu tako i u filmovima.

Uprkos današnjem hardveru i naporima programera, prilikom crtanja vodenih površi se ne može postići isti rezultat kakav se može videti u prirodi. Dva glavna razloga za to su samo kretanje vode, koje može biti veoma nepredvidivo i zavisi od raznoraznih faktora kao što su atmosferske prilike, dubina itd, i interakcija vodene površi sa svetlosnim zracima, odnosno optički efekti kao što su refleksija, refrakcija, kaustika itd. Danas je skoro nemoguće predstaviti vodenu površ poštujući njen ponašanje u prirodi i postići dovoljnu brzinu crtanja tako da posmatrač ne primeti kašnjenja ili seckanja prilikom prikazivanja.

Kao što je navedeno u [Fle2007], problemi koji se javljaju prilikom crtanja vodene površi se mogu podeliti u dve kategorije:

1. Simulacija vodenih površi i njihovog kretanja
2. Poboljšanje izgleda vodene površi dodavanjem optičkih efekata, odnosno dodavanjem interakcije vodene površi sa svetlosnim zracima

U ovom radu čitaocu će biti predviđeni neki od načina na koje se mogu prevazići ovi problemi. Prilikom njihovog razmatranja uzeto je obzir koliko oni doprinose realističnom izgledu vodene površi, ali i to kako utiču na brzinu crtanja. Neki od njih su jednostavni i uključuju dosta grubih aproksimacija, ali ipak daju zadovoljavajuće rezultate, dok su drugi komplikovani i oslanjaju se na matematički zahtevne metode. Koji od ovih će se koristiti zavisi od potreba za realnošću i performansama.

U drugom poglavlju ovog rada detaljno su predstavljeni problemi na koje se nailazi prilikom crtanja vodenih površi. Kao što je navedeno, ovi problemi će biti podeljeni u dve kategorije. U trećem poglavlju opisana su neka od postojećih rešenja pomoću kojih se rešavaju gore navedeni problemi. Četvrto poglavlje sadrži opis programa, implementiranog pomoću OpenGL grafičke biblioteke, koji pokazuje jednu moguću realizaciju vodene površi baziranu na navedenim rešenjima. Poslednje, peto, poglavlje predstavlja zaključak u kome je dat kratak osvrt na sam sadržaj rada kao i pravci u kojima treba nastaviti dalje istraživanje ove teme.

## 2. Problem

U ovom poglavljiju biće predstavljena dva glavna problema koji se javljaju prilikom crtanja vodenih površina. Kao što je navedeno u uvodu, ta dva problema su simulacija kretanja vodene površine i interakcija vodene površi sa svetlosnim zracima, odnosno optički efekti.

### 2.1. Simulacija

Kretanje fluida je opisano Navier-Stokes jednačinama[WikiA]. Kada se ove jednačine primene na volumetrijsku predstavu vode dobije se fizički tačna predstava vodene površi. Međutim, ove jednačine zahtevaju ogromno zauzeće računarskih resursa i primenjuju se samo u simulacijama za naučne potrebe. Stoga se prilikom prikazivanja vodene površi u računarskoj grafici često koriste razne aproksimacije njenog kretanja. Izbor samog metoda simulacije vodene površi zavisi od toga kakvu vodenu površ je potrebno predstaviti. Način ponašanja jezera, reka i okeana nije isti i za svaki postoji dijapazon jednostavnijih jednačine kojima se može opisati kretanje vodene površi.

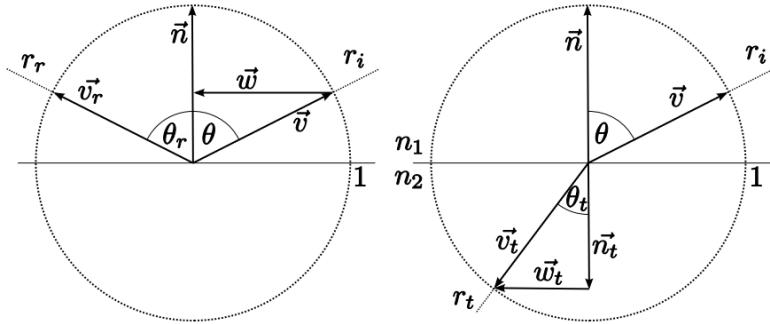
Pored samog izbora načina simulacije postoji i problem modela same vodene površi. Najčešće je to mreža trouglova. Međutim, tu se postavlja pitanje koliki broj trouglova ta mreža treba da sadrži. Naravno, što je data mreža gušća to je i sama vodena površ realističnija ali, kao što je navedeno u [Tur2014], takva mreža zahteva velike količine memorije na grafičkom procesoru jer je potrebno za svaki čvor date mreže čuvati njegove koordinate.

### 2.2. Optika

Postoji mnogo efekata koji se javljaju usled interakcije vodene površi sa svetlosnim zracima i svi oni doprinose realističnom njenom izgledu. Od svih efekata, u ovom odeljku biće spomenute refleksija, refrakcija i kaustika.

#### 2.2.1. Refleksija i refrakcija

Dva najbitnija optička efekta koja se javljaju usled interakcije vodenom površi sa svetlosnim zracima su refleksija i refrakcija. Međutim, da bi se uopšte krenulo u njihovo razmatranje potrebno je znati kako one funkcionišu. Za njihov proračun potreban je jedinični vektor normale u svakoj tački vodene površi. Ovo ne predstavlja problem zato što ukoliko su poznate koordinate tačaka vodene površi lako se može nekom matematičkom metodom doći do vektora normala u datim tačkama vodene površi. Način delovanja refleksije i refrakcije dat je na slici 1.



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

Najpre će biti analizirana refleksija, odnosno levi deo slike 1. Neka su vektori  $\vec{v}, \vec{n}, \vec{v}_r$  i  $\vec{w}$  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 updani ugao  $\theta$  i reflektujući ugao  $\theta_r$  jednaki, vektor reflektovanog zraka se može izračunati 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 će se 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 vodena površ dolazi u kontakt. Neka su vektori  $\vec{v}, \vec{n}, \vec{v}_t$  i  $\vec{w}_t$  sa desnog dela slike 1 jedinični vektori i to inverzni vektor svetlosnog zraka, vektor normale, vektor refraktovanog zraka i pomoći vektor, respektivno. Osim toga, na ovom delu slike sa  $n_1$  i  $n_2$  su označeni indeksi refrakcije medijuma sa kojim vodena površ dolazi u kontakt i vodene površi, repsektivno. Odnos upadnog ugla  $\theta$  i ugla refrakcije  $\theta_t$  se može izračunati iz Snell-ovog zakona[WikiB]:

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

Dalje, koristeći Snell-ov zakon vektor refraktovanog zraka se može dobiti na sledeći način:

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

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 svetlosnim zracima je Fresnel-ov efekat na osnovu kojeg se računa koji deo upadanog svetlosnog zraka se reflektovao, a koji deo se refraktovao. Pored toga što zavisi od upadnog ugla i indeksa refrakcije, zavisi i od toga da li je svetlost s-polarizovana ili p-polarizovana [WikiC]. Jednačine po kojoj se dobija koji deo zraka se reflektovao, odnosno refleksioni koeficijent, 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 je  $\theta$  ugao koji zaklapaju vektor upadnog zraka i vektor noramale,  $\theta_t$  ugao koji zaklapaju vektor refraktovanog zraka i vektor normale,  $n_1$  indeks refrakcije prvog medijuma,  $n_2$  indeks refrakcije drugog medijuma,  $R_s$  refleksioni koeficijeti za s-polarizovanu i  $R_p$  refleksioni koeficijeti za p-polarizovanu svetlost. Refrakcioni (transmisioni) koeficijenti se dobijaju iz jednakosti  $T_s = 1 - R_s$  i  $T_p = 1 - R_p$ , gde su  $T_s$  i  $T_p$  refrakcioni (transmisioni) koeficijenti za s-polarizovanu i p-polarizovanu svetlost, respektivno. Za nepolarizovanu svetlost, koja sadrži jednak deo s-polarizovane i p-polarizovane svetlosti, koeficijenti su dati sledećim jednačinama:

$$R = \frac{R_s + R_p}{2}$$
$$T = \frac{T_s + T_p}{2}$$

## 2.2.1. Kaustika

Efekat kaustike se javlja kao posledica refrakcije. Naime, deo upadnih zraka 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 dela površi u odnosu na ostale. Na slici 2 je prikazan efekat kaustike.



Slika 2. Efekat kaustike

### **3. Pregled postojećih rešenja**

U ovom poglavlju dat je pregled postojećih rešenja problema navedenih u prethodnom poglavlju. Kao problemi u prethodnom poglavlju, i rešenja su podeljena u dve grupe. Prva grupa se bavi rešenjima problema simulacije kretanja vodene površi, a druga grupa se bavi rešenjima problema optičkih efekata, odnosno problema koji nastaju usled interakcije vodene površi sa svetlosnim zracima.

#### **3.1. Simulacija**

U ovom delu biće predstavljene tehnike koje se koriste za simularanje kretanja vodene površi. Svaka od navedenih tehnika podrazumeva da se vodenu površ modeluje mrežom trouglova određene gustine. Od same gustine mreže trouglova zavisi koliki će biti nivo detalja vodene površi. Pored računanja samih pozicija čvorova mreže trouglova, moraju se izračunati i vektori normala u svakom čvoru koji će se kasnije koristiti za implementaciju optičkih efekata.

Načelno, prilikom simulacije kretanja vodene površi postoje dva pristupa. Prvi pristup podrazumeva da se prilikom svakog crtanja vodene površi koriste jednačine, dobijene određenim aproksimacijama, da bi se izračunale pozicije čvorova mreže trouglova, kao i svi drugi podaci neophodni dalje za crtanje. Drugi pristup podrazumeva da se svi neophodni podaci izračunaju i smeste u teksturu koja će kasnije biti korišćena prilikom crtanja. Ove teksture se u računarskoj grafici nazivaju mape visina o kojima će biti reči kasnije.

##### **3.1.1. Simulacije pomoću aproksimacija**

U ovom odeljku biće opisane metode simulacije kretanja vodene površi korišćenjem jednačina dobijenih pomoću raznih aproksimacija. Za razliku od pristupa kod kojeg se koriste mape visina i kod kojeg je potreban dodatni memoriski prostor za smeštanje te mape, kod ovog pristupa se prilikom svakog crtanja računaju pozicije čvorova mreže pa samim tim i nema dodatnog memoriskog zauzeća na grafičkom procesoru. Međutim, sama računica može biti kompleksna pa sa kao posledica toga može javiti veliko zauzeće drugih resursa grafičkog procesora.

U ovom odeljku biće objašnjene dve jednostavne aproksimacije pomoću kojih se postiže simulacija kretanja vodene površi i one su: aproksimacije talasa sumom sinusa i simulacije kretanja vodene površi pomoću Gerstner-ovih talasa.

###### **3.1.1.1. Aproksimacija pomoću sume sinusa**

Kao što je navedeno u [FerKir2004], da bi aproksimirali kretanje vodene površi pomoću talasa predstavljenih kao sume sinusa potreban je određeni skup parametara koji opisuje ponašanje talasa. Ti parametri su sledeći:

1. Talasna dužina  $L$  koja je povezana sa frekvencijom preko formule  $w = \frac{2}{L}$
2. Amplituda  $A$ , odnosno elevacija talasa od ravni vodene površi

3. Brzina  $S$  kojom se talas kreće pomoću koje će se izraziti fazna konstanta

$$\varphi = \frac{2 \cdot S}{L}$$

4. Pravac  $D$  izražen kao horizontalan vektor normalan na čelo talasa

Stanje talasa se može izraziti funkcijom koja zavisi od horizontalnog položaja, odnosno  $x$  i  $z$  koordinata čvora mreže, i vremena  $t$  i u kojoj figuriraju prethodno navedeni parametri. Ona glasi:

$$W(x, z, t) = A \cdot \sin(D \cdot (x, z) \cdot w + t \cdot \varphi)$$

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

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

Parametre samih talasa je najlakše odrediti pomoću generatora pseudoslučajnih brojeva, naravno vodeći računa o tome da postoje određena ograničenja. Tokom simulacije, talasi će kontinuirano biti izbacivani kada dođu do kraja mreže trouglova i ponovo ubacivani sa drugim skupom parametara. Ova tehnika je pogodna zato što je ponašanje talasa opisano konkretno funkcijom pa računanje vektora normale u čvorovima mreže nije komplikovano. Za računanje vektora normale u čvoru mreže trouglova potrebni su vektor binormale i vektor tangente u datom čvoru, koji se dobijaju kao parcijalni izvodi funkcije talasa  $H(x, z, t)$  u  $x$  i  $z$  pravcu, respektivno. Pozicija nekog čvora sa koordinatama  $(x, z)$  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:

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

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

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

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

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

Naravno, nakon ovoga je potrebno normalizovati vektor normale ukoliko on nije jedinični vektor. Dodatna pogodnost ovog metoda je ta što je nalaženje parcijalnih izvoda jednostavno zbog osobine izvoda koja se ispoljava prilikom računanja izvoda sume tj. važi da je izvod sume jednak sumi izvoda i to je navedeno u sledećoj formuli:

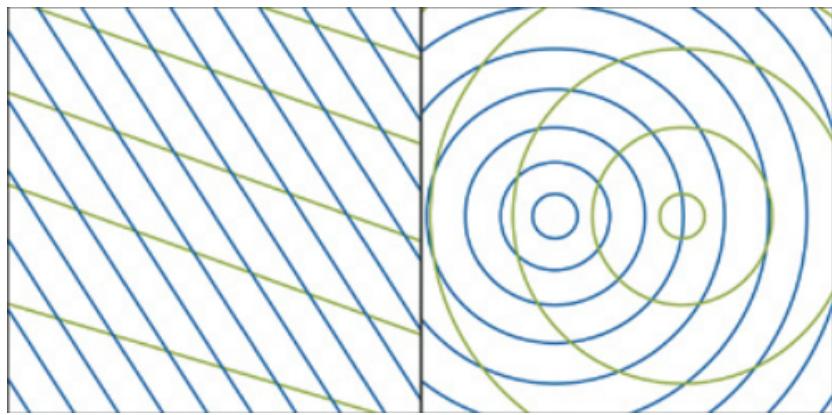
$$\frac{\partial}{\partial x} \cdot H(x, z, t) = \sum \frac{\partial}{\partial x} \cdot W_i(x, z, t) = \sum w_i D_i \cdot x \cdot A_i \cdot \cos(D_i \cdot (x, z) \cdot w_i + t \cdot \varphi_i)$$

Međutim, postoji jedna primedba navedena u [FerKir2004] i ona je da ovi talasi nemaju oštре vrhove kao što je slučaj sa talasima u prirodi. Ovaj problem se lako rešava malom modifikacijom prethodne formule talasa. Modifikacija se ogleda u promeni sinusne funkcije tako što joj se doda pomeraj od 1 tako da ona bude nenegativna i podigne se na stepen  $k$ , gde ovaj stepen određuje koliko oštри će biti vrhovi talasa. Funkcija talasa i njen parcijalni izvod u  $x$  pravcu nakon prethodne modifikacije izgledaju ovako:

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

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

Poslednja stvar koju treba odraditi je odabir između usmerenih i kružnih talasa koji su prikazani na slici 3.



Slika 3. Usmereni (levo) i kružni (desno) talasi [FerKir2004]

Razlika između ova dva tipa talasa se ogleda u vektoru pravca. Za usmerene talase vektor pravca  $D_i$  je konstantan za svaki čvor. Za kružne talase ovaj vektor se računa svaki put i to tako da on ide od centra kružnog talasa  $C_i$  do čvora mreže i on glasi:

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

### 3.1.1.2. Gerstner-ovi talasi

Ukoliko je neophodno da talasi imaju oštije vrhove potrebno je 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 uobičajne okrugle sinusne talase opisane u prethodnom odeljku, a vrednost parametra  $Q_i = \frac{1}{w_i \cdot A_i}$  daje talase sa oštrim vrhom. Veće

vrednosti ovog parametra treba izbegavati jer one mogu prouzrokovati formiranje petlji na vrhovima talasa. Ostale promenljive imaju isto značenje kao u prethodnom odeljku.

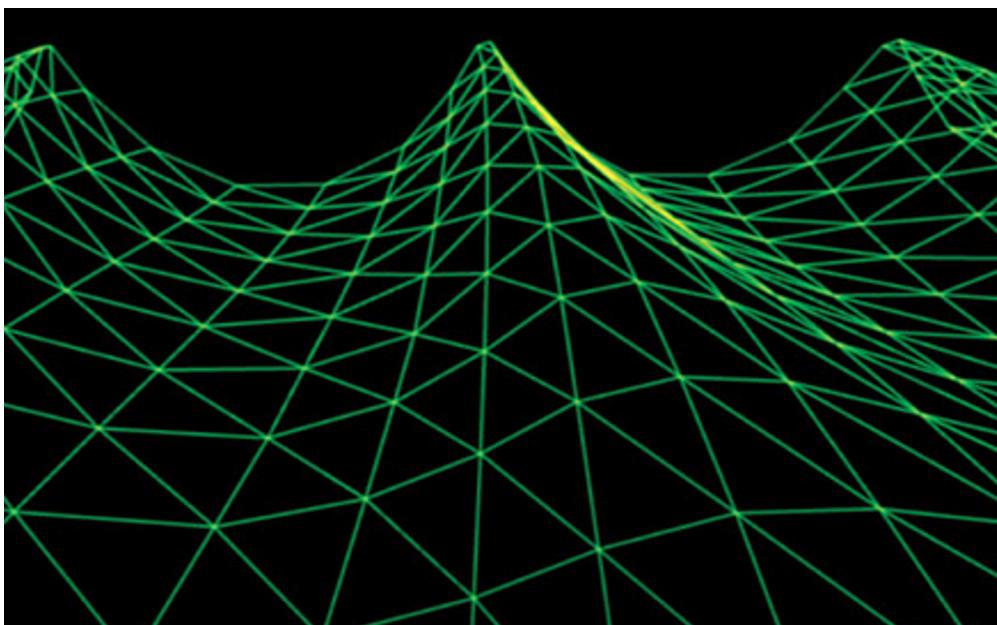
Vektor binormale i vektor tangente se dobijaju parcijalnim izvodom pozicije čvora  $P(x, z, t)$  u  $x$  i  $z$  pravcu, respektivno, a vektor normale se dobija vektorski proizvodom vektora binormale i vektora tangente. Ovi vektori su dati sledećim jednačinama:

$$B(x, z, t) = \begin{cases} 1 - \sum (Q_i \cdot D_i \cdot x^2 \cdot w_i \cdot A_i \cdot \sin(w_i \dot{D}_i \cdot P + \varphi_i \cdot t)) \\ \sum (D_i \cdot x \cdot w_i \cdot A_i \cdot \cos(w_i \dot{D}_i \cdot P + \varphi_i \cdot t)) \\ -\sum (Q_i \cdot D_i \cdot x \cdot D_i \cdot z \cdot w_i \cdot A_i \cdot \sin(w_i \dot{D}_i \cdot P + \varphi_i \cdot t)) \end{cases}$$

$$T(x, z, t) = \begin{cases} -\sum (Q_i \cdot D_i \cdot x \cdot D_i \cdot z \cdot w_i \cdot A_i \cdot \sin(w_i \dot{D}_i \cdot P + \varphi_i \cdot t)) \\ \sum (D_i \cdot z \cdot w_i \cdot A_i \cdot \cos(w_i \dot{D}_i \cdot P + \varphi_i \cdot t)) \\ 1 - \sum (Q_i \cdot D_i \cdot z^2 \cdot w_i \cdot A_i \cdot \sin(w_i \dot{D}_i \cdot P + \varphi_i \cdot t)) S \end{cases}$$

$$N(x, z, t) = \begin{cases} -\sum (D_i \cdot x \cdot w_i \cdot A_i \cdot \cos(w_i \dot{D}_i \cdot P + \varphi_i \cdot t)) \\ 1 - \sum (Q_i \cdot w_i \cdot A_i \cdot \sin(w_i \dot{D}_i \cdot P + \varphi_i \cdot t)) \\ -\sum (D_i \cdot z \cdot w_i \cdot A_i \cdot \cos(w_i \dot{D}_i \cdot P + \varphi_i \cdot t)) \end{cases}$$

Ove jednačine nisu "lepe" kao one iz prethodnog poglavlja, ali su i dalje jednostavne za računanje. Uslov za izbegavanje petlji na vrhovima talasa se može videti iz  $y$  komponente vektora normale. Sve dok je u jednačini  $y$  komponente vektora normale 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. Izled talasa dobijenih ovom jednačinom dat je na slici 4.



Slika 4. Gerstner-ovi talasi

Kao i kod prethodnog pristupa, odabir parametara jednačine zavisi od toga kakvi su talasi potrebni, odnosno kakvi su atmosferski uslovi. Na primer, za talasnu dužinu se može izabrati neka minimalna vrednost koja odgovara sunčanom i mirnom vremenu i neka maksimalna vrednost koja odgovara olujnom vremenu. Nakon toga talasne dužine se

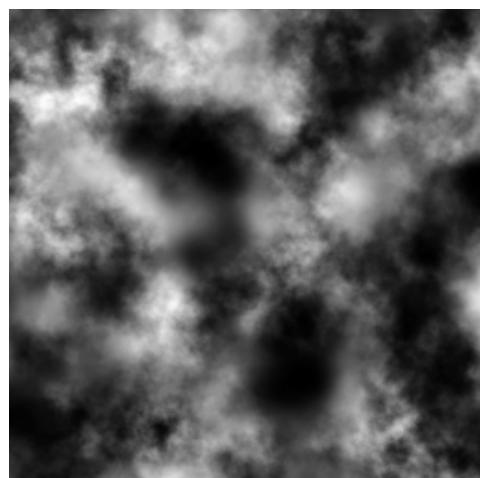
nasumično biraju iz ovog opsega. Može se koristiti relaciju disperzije navedena u [Tes2001] da bi se izračunala brzina talasa. Ona glasi:

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

Ostali parametri se mogu izabrati po istom principu, odnosno iz nekog predefinisanog opsega prema atmosferskim uslovima.

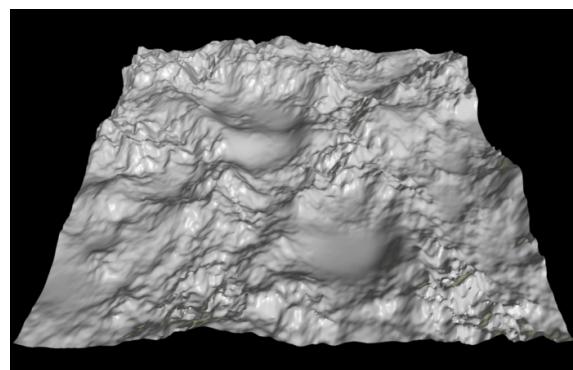
### 3.1.2. Simulacija pomoću mapa visina

Kod ovog pristupa se prilikom inicijalizacije ili pre svakog crtanja računaju elevacije čvorova mreže i dobijene vrednosti se smeštaju u teksturu. Ove tekture se u računarskoj grafici nazivaju mape visina (engl. *heightmap*[WikiD]). Primer jedne takve mape dat je na slici 5.



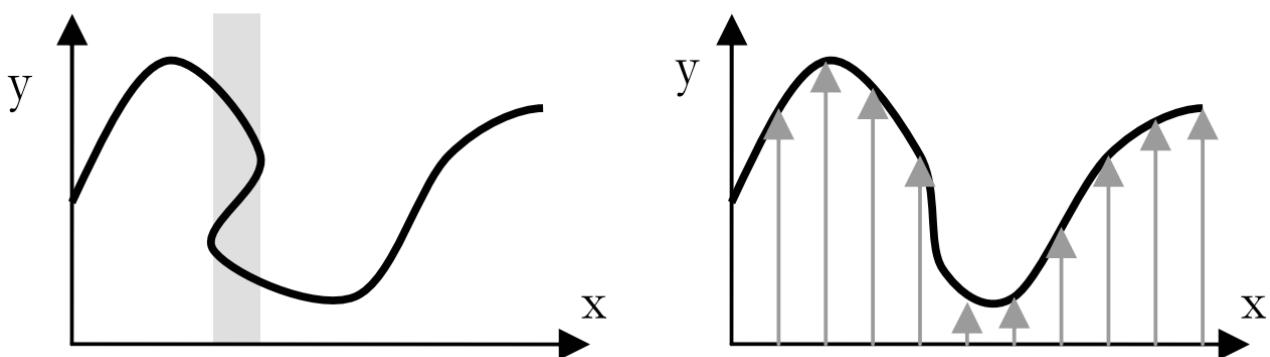
Slika 5. Mapa visina[WikiD]

Ove mape predstavljaju paletu sivih tonova. Svaka ćelija tekture čuva informacije o visini, odnosno elevaciji, čvora od neke referente ravni. Crne ćelije predstavljaju minimalnu elevaciju, dok bele ćelije predstavljaju maksimalnu elevaciju. Prilikom crtanja, mapa visina se uzorkuje i pročitane vrednosti se pretvaraju u pomjeraj u odnosu na neku referentnu ravan. Primer korišćenja mape visina dat je na slici 6.



Slika 6. Teren dobijen pomoću mape visina[WikiD]

Na slici 6 nacrtan je teren. Prilikom crtanja, za modelovanje ovog terena korišćena je mreža trouglova, a visina svakog čvora dobijena je uzorkovanjem mape visina date na slici 5. Naravno, mape visina mogu biti korišćene za dobijanje elevacije talasa u odnosu na ravan vodene površi. Prilikom crtanja potrebno je koristiti  $(x, z)$  koordinate čvora za uzorkovanje mape visina, a pročitana vrednost biće tretirana kao elevacija čvora, odnosno  $y$  koordinata, u odnosu na horizontalnu ravan vodene površi. Međutim, ovaj pristup ima jedno ograničenje. Naime, elevacija čvora, odnosno  $y$  koordinata, je sada funkcija zavisna od  $x$  i  $z$  koordinata tj. važi  $y=f(x, z)$ , pa se ne može desiti da postoje dva ili više čvorova mreže različitih visina, a istih  $x$  i  $z$  koordinata. Ovime je izgubljena mogućnost crtanja talasa koji se "lome". Ilustracija ovog ograničenja data je na slici 7.



Slika 7. Ograničenje tehniku koje koriste mape visina [Joh2004]

Tehnikama koje koriste mape visina moguće je postići talase prikazane na desnom delu slike 7.

Za realističan prikaz vodene površi potrebno je takođe postići njeno kretanje. Ovo se može realizovati na dva načina. Prvi je da se mapa visina generiše prilikom svakog crtanja, ali sa drugačijim parametrima. Drugi je da se prilikom crtanja pomoći vremenski zavisnog faktora odredi pomeraj koji će biti dodat  $(x, z)$  koordinatama čvora pomoći kojih se uzorkuje mapu visina.

Prednost ove tehnike je da programi za crtanje (engl. *shader*) sada ne sadrže kod za računanje visina korišćenjem jednačina dobijenih pomoći neke aproksimacije koje mogu biti veoma kompleksne kao što se moglo videti iz prethodnog odeljka. Pored toga, kao što je navedeno u prethodnom pasusu, postoje tehnike kod kojih je potrebno generisati mapu visinu samo jednom, najčešće u nekom od inicijalizacionih koraka, a ipak postići efekat kretanja vodene površi.

Međutim, glavni nedostatak ove tehnike je veličina teksture koja se koristi kao mapa visina. U prirodi su talasi "glatki" i da bi se ta glatkoća postigla u računarskoj grafici potrebno je da razlika visina susednih čvorova bude minimalna, odnosno da vodena površ ne sadrži "stepenike". Ovo zahteva da ćelije teksture budu dovoljno velike da bi se eliminisale greške koje nastaju usled računice, tačnije greške koje se javljaju zbog zaokruživanja usled ograničene širine ćelija teksture. Treba pažljivo birati veličinu ćelija jer veća ćelija može dati realističniji izgled, ali takođe može uticati i na performanse zato što će u tom slučaju tekstura zauzimati mnogo više memorijskog prostora na grafičkom procesoru. Međutim, ovaj nedostatak se može prevazići tako što će se generisati mapa

visina sa većom širinom celija čije su dimenzije manje od dimenzija mreže trouglova, a kasnije, tokom crtanja, će više manjih mapa visina biti naslagane da se dobila jedna mapa visina dovoljno velika da pokrije celokupnu mrežu trouglova.

Postoji mnogo načina na koje se može doći do mape visina. Najjednostavniji je upotreba nekog generatora mapa visina dostupnog na internetu. Međutim, postoje i načini da se ova mapa generiše u samom programu. U ovom odeljku biće predstavljena tri načina pomoću kojih možemo generisati mape visina u samom programu. Prvi koristi 2D jednačinu talasa, drugi tehniku generisanja Perlin-ovog šuma, a treći brze Fourier-ove transformacije.

### 3.1.2.1. 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:

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

U gorenavedenoj jednačini ravan  $XoZ$  predstavlja horizontalnu ravan,  $x$  i  $z$  su koordinate čvora u horizontalnoj ravni,  $y$  je vertikalna koordinata čvora, odnosno elevacija u odnosu na horizontalnu ravan, a  $t$  je proteklo vreme. Data jednačina menja  $y$  koordinatu tako da ona osciluje gore dole, gde  $c$  predstavlja brzinu datog talasa. Gledajući datu jednačinu intuitivno se može zaključiti da se  $y$  koordinata menja u skladu sa menjanjem nagiba talasa, odnosno nagiba vodene površi. Međutim, da bi se dobole pozicije čvorova potrebno je naći integral leve strane jednačine po promenljivoj  $t$ , kao i izvode na desnoj strani jednačine. U tu svrhu, ova jednačina će se rešiti korišćenjem numeričke metode.

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

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

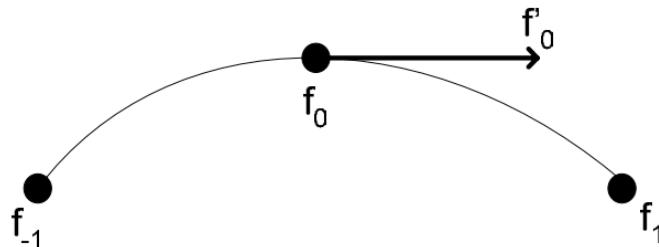
Međutim, zbog greške koja se ispoljava prilikom primene ove metode dolazi do podrhtavanja vodene površi. Ovo je moguće rešiti smanjivanjem brzine pomoću nekog konstantog faktora. Ovaj faktor će se označiti sa  $\alpha$ . Nakon toga data jednačina izgleda ovako:

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

Sledeći korak je rešavanja parcijalnih izvoda na desnoj strani jednačine. Najpre je potrebno aproksimirati vodenu površ kubnom krivom. Na ovaj način se drugi parcijalni izvod iz jednačine talasa može izjednačiti sa drugi izvodom kubne krive. Datu kubnu krivu je posmatrana u 2D umesto u 3D zbog pojednostavljenja problema. Funkcija koja opisuje kubnu krivu, kao i njeni izvodi, opisani su sledećim jednačinama [WikiE]:

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

Datu krivu se može prikazati na sledeći način:



*Slika 8. Kubna kriva [Zel/2004]*

Na slici 8 data je kriva koja prolazi kroz tri tačke koje su na odstojanju dužine 1. Potreban je drugi izvod u tački nula (na slici 8 je to tačka u sredini). On se može izračunati 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 mogu se izračunati parcijalni izvodi na desnoj strani jednačine 2D talasa. Nakon toga moguće je izračunati kompletну desnu stranu i ona glasi ovako:

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

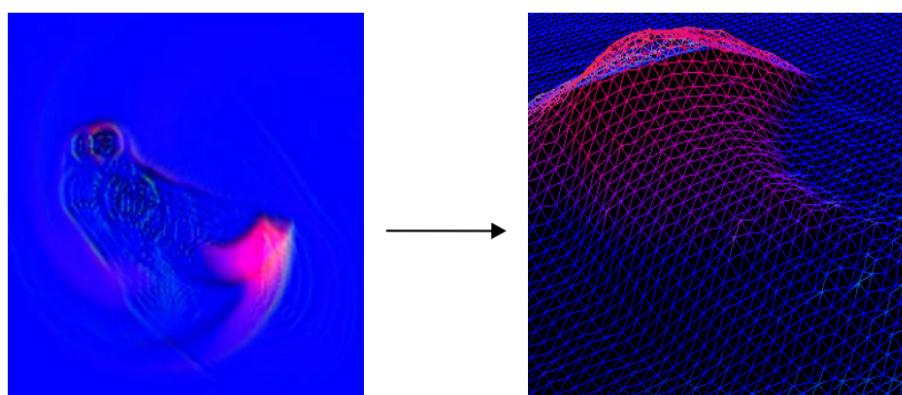
$$c^2 \cdot \left( \frac{\partial^2 y}{\partial x^2} + \frac{\partial^2 y}{\partial z^2} \right) = c^2 \cdot (y_{-1,0} + y_{1,0} + y_{0,-1} + y_{0,1} - 4 \cdot y_{0,0})$$

Na osnovu navedenih jednačina se može generisati potrebna mapa visina. Naravno, pre svakog crtanja neophodno je ažurirati, odnosno generisati novu mapu visina, da bi se postiglo kretanje vodene površine. Prilikom ažuriranja koristiće se tehniku koja se zove "ping-pong" teksture (engl. *ping-pong textures*). Odnosno, nećemo imati jednu već dve teksture. U prvoj će biti smeštene visine iz prethodnog crtanja, dok će se u drugoj smeštati nove visine čvorova koje će se koristiti u tekućem crtanju. Ukoliko se sa  $p$  označe  $(x, z)$  koordinate čvora i ukoliko se sa  $w$  i  $h$  označe širina i visina tekture, kod na HLSL (engl. *High Level Shading Language*) bi izgledao kao na listingu 1.

```
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);
accelartion = c * c * (heightPLeft + heightPRight + heightPUp + heightPDown - 4 * heightP);
newHeightP = 2 * heightP - previousHeightP + 0.5 * acceleration * dt * dt;
```

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

Rezultat ovog koda dat je na slici 9.



*Slika 9. 2D jednačina talasa [Zel2004]*

Pored računanja samih pozicije čvorova mreže trouglova potrebno je i sračunati vektore normala u datim čvorovima. Najlakši način da se to odradi je da se nađu dva vektora koji počinju u tekućem čvoru i prostiru se ka njegovim susedima i da se njihovim vektorskim proizvodom dobije vektor normale. Naravno, kao i do sada, potrebno je 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*

Za postizanje veće preciznost moguće je izračunati više vektora normale koristeći sva četiri suseda i nakon toga interpolacijom dobiti konačan vektor normale.

Kao što je navedeno na početku ovog odeljka, najveća pogodnost ovog pristupa je njegova brzina i činjenicu da u maloj meri smanjuje realnost vodene površi. Pored toga, još jedna pogodnost ovog pristupa je ta što se i visine i normale mogu čuvati u jednoj teksturi RGBA tipa gde bi na primer alfa kanal bio odvojen za elevaciju čvora mreže, a crveni, zeleni i plavi kanal za  $x, y$  i  $z$  komponentu vektora normale u datom čvoru, respektivno. Ova tekstura bi se kasnije koristila prilikom crtanja vodene površi, i to dva puta. Prvi put bi se pristupalo prilikom obrade čvorova mreže trouglova kada bi se tekstura uzorkovala sa  $(x, z)$  koordinata trenutno obrađivanog čvora da bi se pročitala njegova elevacija u odnosu na horizontalnu ravan i tako dobila konačna pozicija čvora. Drugi put bi se pristupalo prilikom implementacije optičkih efekata kada bi se tekstura uzorkovala sa  $(x, z)$  koordinatama čvora da bi se pročitale komponente vektora normale koja je neophodna za realizaciju velikog broja ovih efekata.

### 3.1.2.2. Perlin-ov šum

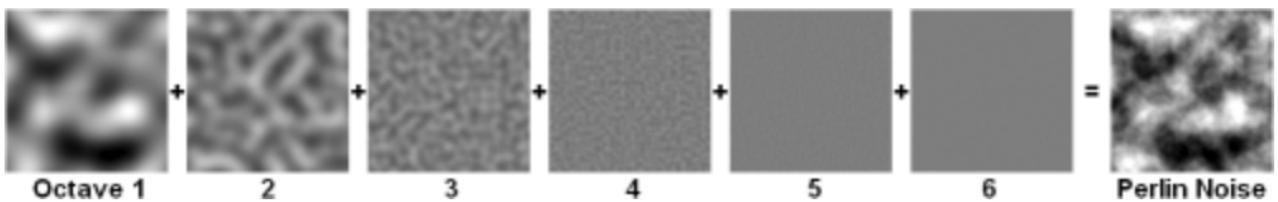
Šumovi se u računarskoj grafici koriste za simulaciju prirodnih fenomena koji su ili prezahtevni za preciznu simulaciju ili su jednostavno i sami nasumični. Postoje razne tehnike za generisanje šumova i jedna od jednostavnijih je tehnika koju je preložio Ken Perlin. Međutim, danas postoji nadogradnja originalne tehnike i ona je detaljno opisana u [Per2002]. Detalji implementacije ovog šuma neće biti razmatrani u ovom radu.

Naravno, kao i kod svakog drugog šuma, jedna oktava daje mnogo velike oscilacije između susednih tačaka, pa se time i gubi "glatkoća" talasa koju možemo videti u prirodi. Jedna oktava šuma prikazana je na slici 10.



*Slika 10. Perlin-ov šum*

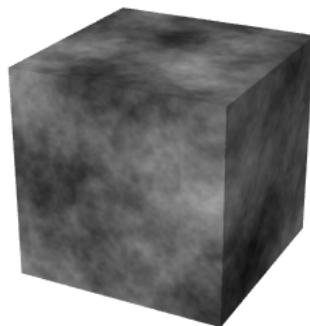
Jedno rešenje ovog problema je povećanje širina ćelija teksture. Međutim, ovaj problem se može lako prevazići i tako što se više oktava šuma saberi. Tada dobijamo šum prikazan na slici 11.



Slika 11. Suma 6 oktava Perlin-ovog šuma

Kretanje vodene površi se može postići na dva načina, kao što je navedeno u uvodu ovog odeljka. Jedan zahteva korišćenje vremenski zavisnog faktora na osnovu kojeg će se izračunati pomeraj (koji se menja u toku vremena) i dodati na  $(x, z)$  koordinate čvora mreže kojima se uzorkuje mapa visina. Drugi zahteva da se prilikom svakog crtanja generiše nova mapa visina, odnosno novi šum, sa drugačijim parametrima.

Međutim, Perlin-on šum pruža još jedan pristup. Ovaj šum je u osnovi 3D šum, odnosno vrednost šuma se dobija na osnovu tri koordinate. Ovakav šum se grafički može prikazati kao kocka što se može videte na slici 12.



Slika 12. 3D Perlin-ov šum

Prilikom inicijalizacije programa generisće se jedan ovakav 3D šum. Kasnije, prilikom crtanja vodene površi, ovako dobijen šum će se uzorkovati da bi se dobole elevacije čvorova mreže u odnosu na ravan vodene površi. Međutim, u ovom slučaju prilikom uzorkovanja potrebno je koristiti tri koordinate. Prve dve će biti  $(x, z)$  koordinate čvora mreže, kao i do sada, dok će treća koordinata biti neki vremenski zavistan faktor. Zbog uključivanja vremenski zavisnog faktora postignuta je promena elevacije čvora u toku vremena, odnosno kretanje vodene površi.

Ovakav način simuliranja kretanja vodene površi je mnogo bolji nego prethodno opisani pristupi. Prvi opisani pristup kod kojeg koristimo vremenski zavistan faktor za računanje pomeraja prilikom uzorkovanja mape visina može se koristiti za simuliranje samo usmerenih talasa. Naime, kod ovog pristupa prilikom crtanja vodene površi koordinate kojima se uzorkuje mapa visina će biti pomerene u istom pravcu za isti intenzitet. Efekat ovoga je privid da se vodena površ kreće samo u jednom pravcu, što ne mora nužno biti slučaj u prirodi. Korišćenjem 3D šuma izbegavamo ovaj pomeraj, a samim tim i privid da se vodena površ kreće samo u jednom pravcu. Međutim, ovaj problem se mogao i rešiti tako što će se pre svakog crtanja, korišćenjem tehnike za generisanje Perlin-ovog šuma, generisati nova mapa visina sa drugačijim parametrima. Kod ovog pristupa bi se takođe eliminisao efekat usmerenog kretanja vodene površi, ali ovakav pristup zahteva više računarskih resursa jer se mapu visina sada mora generisati pre

svakog crtanja. 3D šum predstavljen na slici 12 se može posmatrati kao više naslaganih 2D šumova tako da svi oni čine jednu kocku, odnosno jedan 3D šum. Sa obzirom na to da se 3D šum generiše pri inicijalizaciji sistema, efektivno je postignuto generisanje više 2D šumova, ali se sada ti šumovi ne generiše pre svakog crtanja već samo jednom, pa je zauzeće računarskih resursa manje.

### 3.1.2.3. Brze Fouier-ove transformacije (FFT)

Najrealističnije vodene površi daju statistički modeli koji jedan talas dekomponuju na sumu sinusoida. Prednost ovakve dekompozicije jesu matematičke osobine sinusnih funkcija zbog kojih je modelovanje vodenih površi jednostavnije. Na računarima se ova dekompozija postiže brzim Fouier-ovim transformacijama. Metoda je dosta komplikovana i neće biti detaljno pokrivena u ovom radu, 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. Data je sledećom formulom:

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

U prethodnoj jednačini vektor  $\vec{X}$  je 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:

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

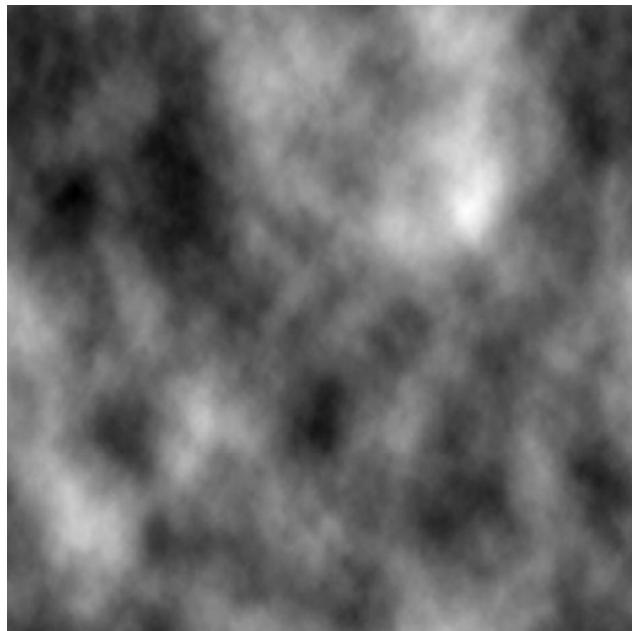
U prethodnoj jednačini  $M$  i  $N$  su dimenzije mape visina, odnosno teksture, a  $L_x$  i  $L_z$  su dimenzije dela mreže trouglova na koji će biti primenjena data mapa visina jer, kao što je navedeno u uvodu ovog odeljku, mapa visina ne mora da bude istih dimenzija kao i mreža trouglova, već je moguće 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 ostaje je pronaleženje vektora normale u čvorovima mreže trouglova. Ova tehnika pruža način da se vektor normale precizno odredi. Najpre je potrebno odrediti nagib talasa  $\epsilon$  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:

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

Opisane jednačine koriste se za generisanje mape visina koja se koristi za simulaciju vodene površi. Na slici 13 je dat izlgled mape visina dobijene korišćenjem ove metode.



Slika 13. FFT mapa visina[Tes2001]

Kod ovog pristupa mapa visina se generiše prilikom svakog crtanja i na taj način se postiže kretanje vodene površi. Stoga, prilikom uzorkovanja potrebne su samo ( $x, z$ ) koordinate čvora mreže trouglova.

## 3.2. Optika

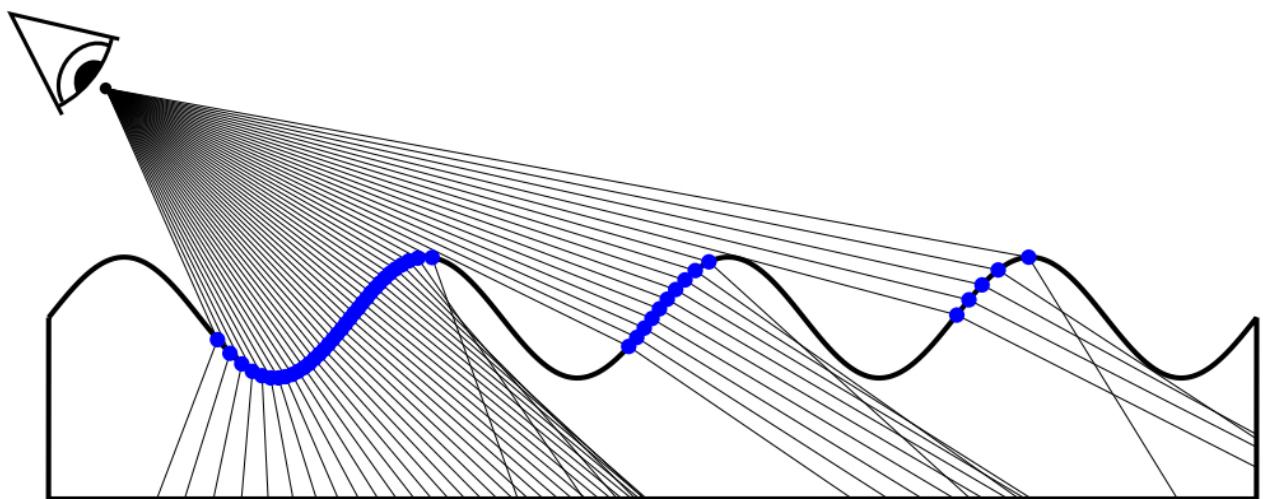
U ovom poglavlju biće predstavljene tehnike koje se koriste za implementaciju optičkih efekata do kojih dolazi usled interakcije vodene površi i svetlosnih zraka. Ovi efekti igraju mnogo veću ulogu u postizanju realističnog izgleda vodene površi nego sama simulacija njenog kretanja. Međutim, njihova 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 koje, iako ponekad grube, ipak postižu zadovoljavajuće rezultate. Od ovih efekata ovde će biti pokriveni refleksija i refrakcija kao dva efekta koji najviše doprinose realističnom izgledu vodene površi. Pored ovih biće opisan i efekat kaustike koji doprinosi realističnom izgledu celokupne scene, doduše ne u tolikoj meri kao prethodna dva.

### 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 se moglo videti u poglavlju 2.2.1, za dobijanje vektora reflektovanog i refraktovanog zraka potreban je vektor normale u dатој тачки koji se računa uporedo sa simulacijom 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 date tačke vodene površi. Međutim, takvih zraka ima previše da bi se sproveo fizički tačan proračun i dobio realan rezultat. Ovde će biti izložene dve tehnike pomoću kojih se jednostavno može odrediti boja vodene površi u dатој тачки.

Prva tehnika se zove praćenje zraka (engl. *ray tracing*[WikiF]). Koristi se 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 rekursivno ponavlja onoliko puta koliko je to potrebno. Naravno, kao što se može i zaključiti iz opisa, ovaj metod je previše zahtevan za crtanje u realnom vremenu. 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 14.



Slika 14. 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 se može 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 se na osnovu mapa visina vode i terena može odrediti pozicija 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 se pronađu tačke preseka novih zraka sa ostatkom sredine, odnosno terenom, boje datih tačaka interakcije se kombinuju da bi se dobila boja vodene površi, naravno poštujući Fresnel-ovu jednačinu. Pseudokod ovog algoritma dat je na slici 15.

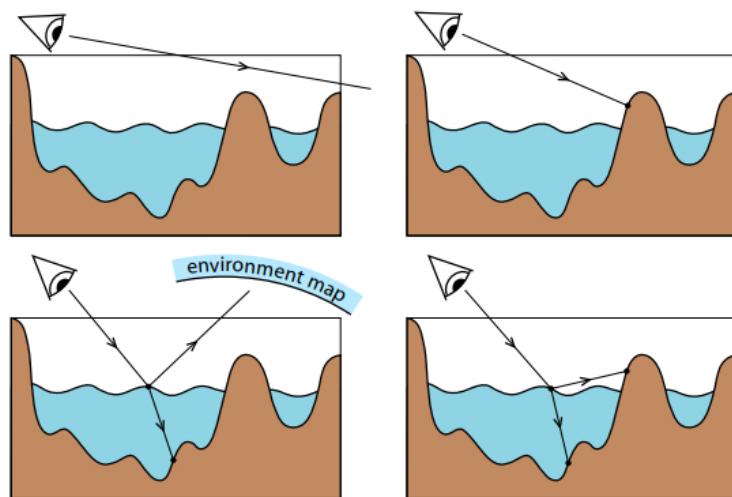
```

 $v \leftarrow \text{viewpoint}$ 
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  $\leftarrow (1 - F) \times C_t + F \times C_r$ 

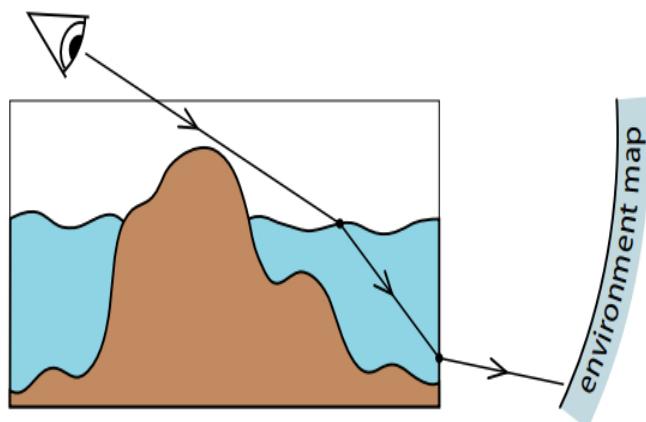
```

*Slika 15. Pseudo kod algoritma praćenja zraka[GalChi2006]*

Naravno, postoje situacije kada dati zrak izlazi van scene. To se može desiti i prilikom refleksije (kada odbijeni zrak ode u "nebo") i prilikom refrakcije kada izađe izvan granica scene. Oba slučaja se mogu rešiti uvođenjem statički definisane teksture koja predstavlja "okolinu" koja nije deo scene u kojoj se nalazi vodena površ. Na osnovu putanje zraka može se odrediti na kojem mestu se uzorkuje data tekstura. Ovakva pristup pokriva slučajeve date na slikama 16 i 17.



Slika 16. Različite putanje zraka [GalChi2006]



Slika 17. Zrak izlazi van granica nakon refrakcije [GalChi2006]

Nasuprot ovom algoritmu koji, iako računarski zahtevan, daje odlične rezultate, postoji i jednostavniji koji je korišćen u [Tur2014] i [Dha2016]. Algoritam radi tako što pre svakog crtanja vodene površi generiše dve teksture, refleksionu i refrakcionu, crtanjem određenog dela scene u njih. U refleksionu tekstuру se crta deo scene koji može da se vidi nakon refleksije. Ovaj deo scene čine svi objekti koji se nalaze iznad vodene površi. Međutim, kao što se može videti u prirodi, odraz objekta nakon refleksije na vodenoj površi je invertovan. Stoga, pre samog formiranja refleksione tekstuure, potrebno je primeniti transformaciju koja će scenu invertovati kao odraz u ogledalu, gde je ogledalo ravan vodene površi. U refrakcionu tekstuру se crta deo scene koji može da se vidi nakon refrakcije. Ovaj deo scene čine svi objekti koji se nalaze ispod vodene površi. Nakon što se refleksiona i refrakciona tekstura generišu, može se preći na crtanje vodene površi. Prilikom crtanja koristiće se tehnika uzorkovanja tekstuure koja se naziva projektujuće teksturiranje, tj. refleksiona i refrakciona tekstuura biće projektovane na vodenu površ kao što projektor prikazuje sliku na platnu. Sam "projektor" uglavnom ima istu poziciju kao i kamera. Mešanje boja dobijenih iz refleksione i refakcione tekstuure se vrši na osnovu Fresnel-ove jednačine.

### 3.2.2. Fresnel-ov efekat

Kao što je navedeno u odeljku 2.2.2, Fresnel-ova jednačina daje informaciju o tome u kojoj meri se zrak svetla reflektuje, odnosno refraktuje, kada udari u vodenu površ. Drugo objašnjenje ove jednačine bi bilo da ona određuje udeo boja tačaka terena u koje je zrak svetlosti udario nakon refleksije, odnosno refrakcije, u konačnoj boji tačke vodene površi. Da bi se dobila konačana boja, zraci se emituju iz kamere, pa će upadni ugao zraka zavisiti od vektora pogleda. Stoga je moguće izvesti sledeću zavisnost između Fresnel-ove jednačine i ugla koji vektor pogleda zaklapa sa vektorom normale: što je ugao između vektora pogleda i vektora normale vodene površi u tački interakcije manji to je dominantnija boja dobijena od refruktovanog 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. Korišćenjem ove zavisnosti i jednačine iz odeljka 2.2.2 može se doći do sledeće formule za refleksioni koeficijent, odnosno udeo boje dobijene od reflektovanog zraka, u slučaju nepolarizovane svetlosti:

$$R = \frac{\frac{n_1}{n_2} + \cos^2(\theta_i) - 1 - \cos(\theta_i)}{2 \cdot (\frac{n_1}{n_2} + \cos^2(\theta_i) - 1 + \cos(\theta_i))} \left( 1 + \frac{\cos(\theta_i) \left( \frac{n_1}{n_2} + \cos^2(\theta_i) - 1 + \cos(\theta_i) - 1 \right)^2}{\cos(\theta_i) \left( \frac{n_1}{n_2} + \cos^2(\theta_i) - 1 - \cos(\theta_i) + 1 \right)} \right)^2$$

U gorenavedenoj jednačini  $R$  je refleksioni koeficijent,  $\theta_i$  je ugao između vektora pogleda i vektora normale vodene površi u datoj tački, a  $n_1$  i  $n_2$  su indeksi refrakcije dva medijuma, u ovom slučaju vazduha i vode. Jednačina zahteva značajnu upotrebu računarsih resursa zato što se refleksioni koeficijent mora izračunati za svaki piksel vodene površi. Međutim, ovaj problem se može prevazići na dva načina. Prvi podrazumeva korišćenje aproksimacija gorenavedene jednačine koje zazimaju manje račuanarskih resursa, a drugi da se u nekom od inicijalizacionih koraka izračunaju refleksioni koeficijenti za odeđen broj uglova i smeste u neku strukturu podataka kojoj će se pristupati kasnije prilikom crtanja.

U slučaju prvog pristupa, postoji veliki broj aproksimacija koje se mogu koristiti. Jedna takva aproksimacija je navedena u [Fle2007] i ona računa refleksioni koeficijent na sledeći način:

$$R = \frac{1}{(1 + \cos(\theta_i))^8}$$

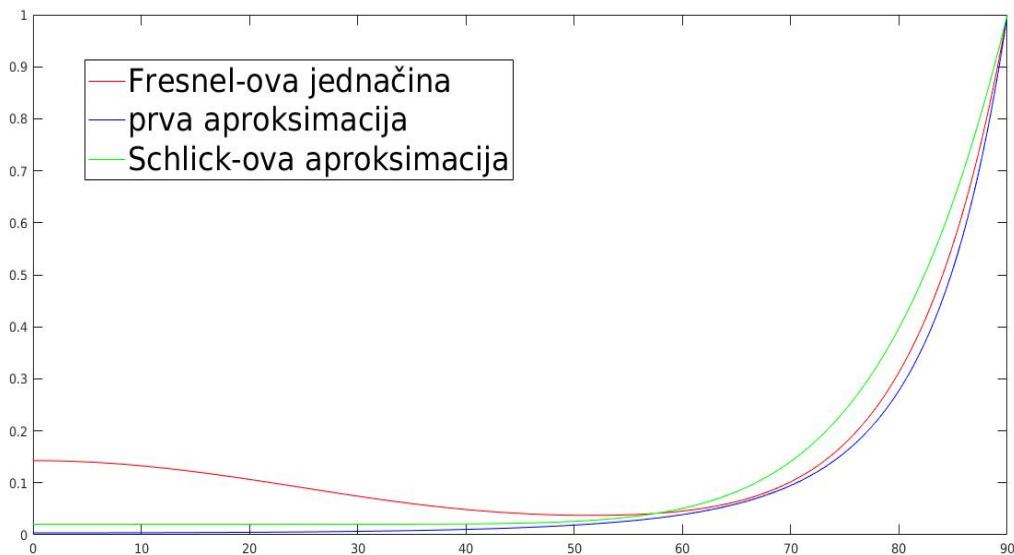
U gorenavedenoj jednačini  $\theta_i$  predstavlja ugao između vektora pogleda i vektora normale, a  $R$  je refleksioni koeficijent. Druga često koršćena aproksimacija je Schlick-ova aproksimacija[WikiG]. Data je sledećom formulom:

$$R = R_0 + (1 - R_0)(1 - \cos(\theta_i))^5$$

$$R_0 = \left( \frac{n_1 - n_2}{n_1 + n_2} \right)^2$$

U prethodnoj jednačini  $R$  je refleksioni koeficijent,  $\theta_i$  je ugao između vektora normale i vektora pogleda,  $n_1$  i  $n_2$  su indeksi refrakcije dva medijuma, u ovom slučaju vode i vazduha, a  $R_0$  je refleksioni koeficijent za upadne zrake paralelne normali. Odnos

navedenih aproksimacija i Fresnel-ove jednačine u zavisnosti od upadnog ugla dat je na slici 18.



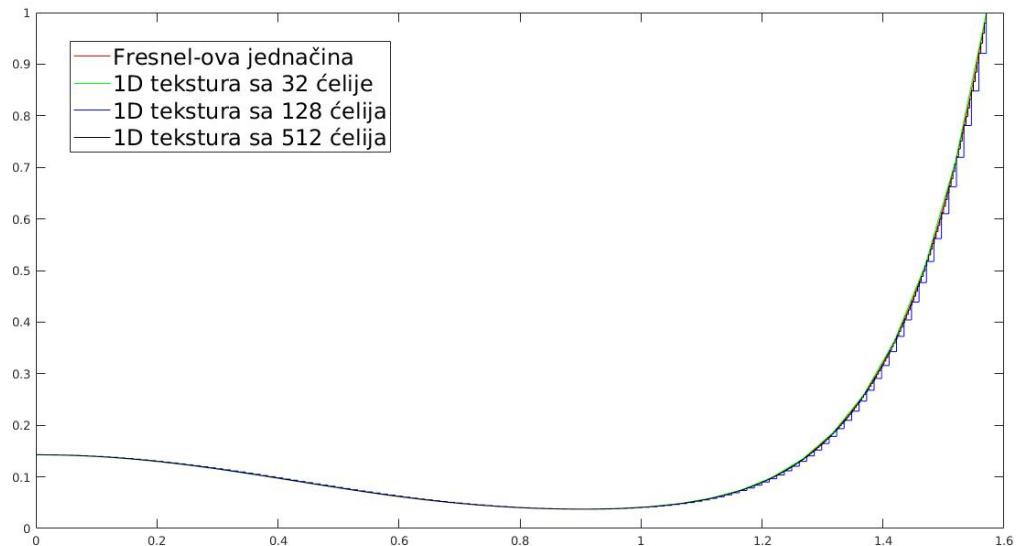
*Slika 18. Odnos Fresnel-ove jednačine i njenih aproksimacija*

Kao što se može videti sa slike 18, obe aproksimacije daju zadovoljavajuće rezultate za uglove veće od  $40^\circ$ , dok za manje uglove postoje odstupanja.

Drugi pristup podrazumeva korišćenje unapred izračunatih refleksionih koeficijenata. Za smeštanje ovih vrednosti može se koristiti heš mapa koja će se popuniti u nekom od inicijalizacionih koraka. Ključ ove heš map pomoći kojeg se pristupa podacima je ugao između vektora pogleda i vektora normale, a vrednost je refleksioni koeficijent dobijen pomoću 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 se uzorkuje tekstura predstavlja ugao između vektora normale i vektora pogleda, a pročitana vrednost predstavlja refleksioni koeficijent. Kasnije će se, prilikom crtanja vodene površi, pristupati ovoj teksturi, odnosno heš mapi, i pomoću pročitane vrednosti odrediti odnos boja.

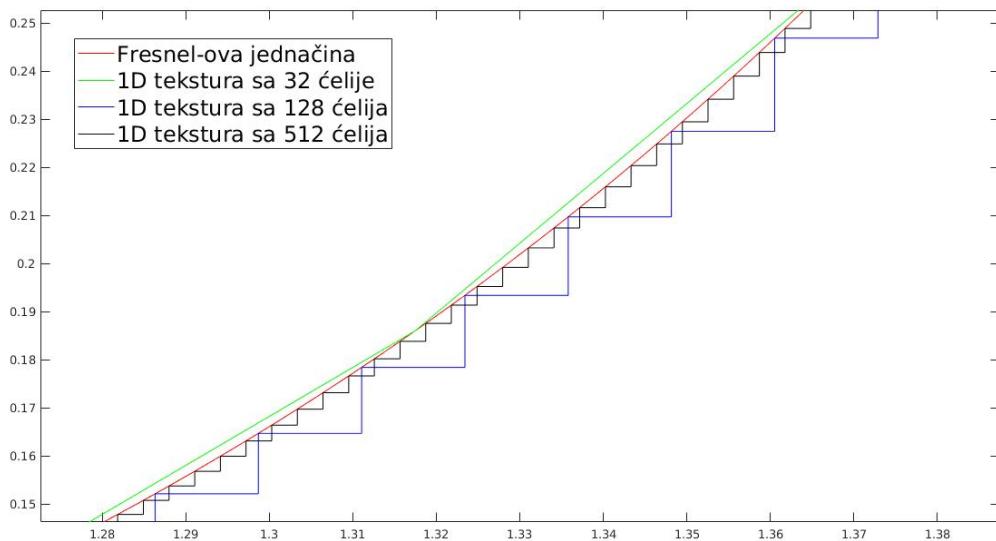
Međutim, kod ovog pristupa treba voditi računa o dve stvari. Prva je režim uzorkovanja testure, odnosno način na koji se u zavisnosti od koordinate sa kojima se pristupa teksturi računa vrednost. Potrebno je koristiti linearan režim uzorkovanja kod kojeg se na osnovu koordinate sa kojom se pristupa (koja ne mora biti ceo broj) nalaze dve ćelije teksture koje su najbliže datoj koordinati i kao rezultat čitanja se vraća vrednost dobijena linearnom interpolacijom vrednosti ove dve ćelije. Druga stvar o kojoj treba voditi računa je veličina ćelija teksture, i to ne samo njene dimenzije već i širina njenih ćelija. Tekstura veće dimenzije može da čuva veći broj refleksionih koeficijenata. Ukoliko su i ćelije teksture šire, ovi koeficijenti će se čuvati sa većom preciznošću. Međutim veća tekstura zahteva više memorijskog prostora na grafičkom procesoru što može dovesti do lošijih performansi program.

Na slici 19 dat je odnos između vrednosti dobijenih Fresnel-ovom jednačinom i unapred izračunatih vrednosti sačuvanih u teksturama različitih dimenzija.



Slika 19. Odnos vrednosti Fresnel-ove jednačine i vrednosti dobijenih sačuvanih u teksturama

Kao što se može videti razlike su neprimetne. Potrebno je povećati sliku 19 nekoliko puta da bi se data razlike uočile, što je i prikazano na slići 20.



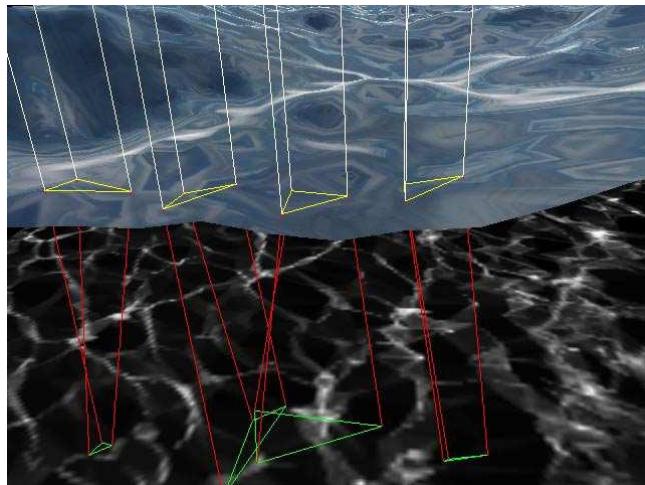
Slika 20. Uvećanje slike 19

Kao što se može videti, ova tehnika daje dobre rezultate uprkos povećanim zauzećem memorije. Potrebno je samo naći balans između željenog nivoa realnosti vodene površi i performansi programa.

### 3.2.3. Kaustika

Efekat kaustika je jedan od najinteresantnijih efekata koji se može prikazati prilikom crtanja vodene površi, međutim ujedno i jedan od najsloženijih. Kao što je navedeno u poglavlju 2.2.1, efekat kaustike predstavlja interferenciju refraktovanih zraka koji čine određene predele ispod vodene površine svetlijim. Efekat kaustiku je nemoguće implementirati bez primene tehnike praćenja zraka (engl. *ray tracing*) i to se najčešće primenjuje obrnuta tehnika praćenja zraka. Kod ove tehnike zraci ne polaze iz svetlosnog izvora već iz kamere.

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 mreže potrebno je sračunati upadni zrak svetla i odgovarajući refraktovani zrak. Refraktovani zrak će se u nekom trenutku sudariti sa zemljom ispod vodene površi pa je takođe potrebno naći tačku u kojoj se ovo dešava. Ovo rezultuje u projektovanju trouglava, koji čine mrežu vodene površi, na zemlju ispod nje i to je prikazano na slici 21.



Slika 21. Projektovani trouglovi[Fle2007]

Intenzitet refraktovanog zraka se može dobiti iz sledeće formule:

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

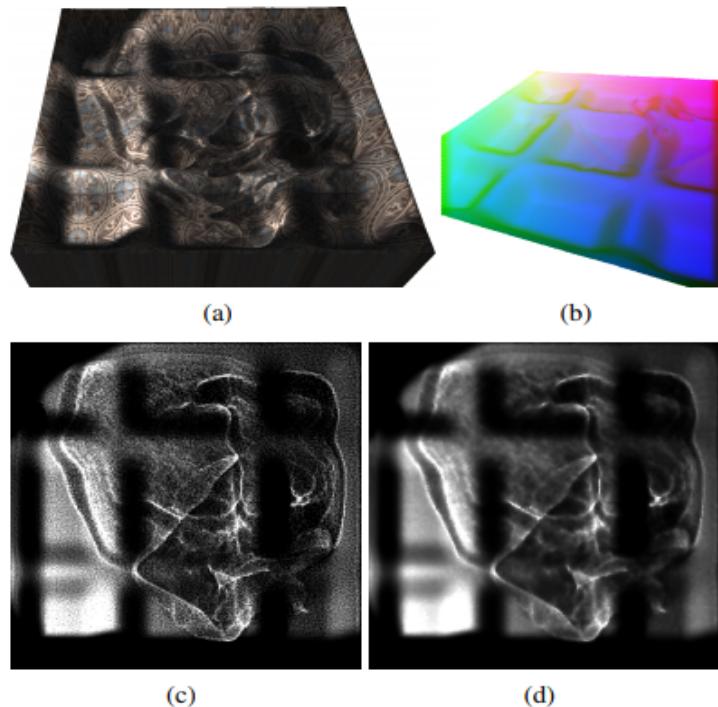
gde je  $N$  vektor normale vodene površi u odgovarajućoj tački,  $L$  je vektor upadnog zraka koji se prostire od čvora mreže trouglova do svetlosnog izvora, a  $a_c$  i  $a_s$  su površine projektovanog i površinskog trougla, respektivno. Sa obzirom na to da je zemlja paralelna sa vodenom površi i da je njena udaljenost od vodene površi poznata, veličine i pozicije ovih trouglovi se mogu lako naći. Nakon toga potrebno je projektovane trouglove smestiti u teksturu, na koju se moraju primeniti algoritmi protiv nazupčenja (engl. *antialiasing*) da bi se dobili vizuelno zadovoljavajući efekti. Prilikom crtanja zemlje ispod vodene površi ova tekstura se paralelno projektuje na nju sa visine vodene površine u pravcu izvora svetlosti.

Malo komplikovaniji algoritam naveden u [GalChi2006] koristi pristup sličan tehnici mapiranja fotona (engl. *photon mapping*), čiji detalji neće biti izloženi u ovom radu. Pristup

podrazumeva da se zemlja ispod vodene površi predstavlja pomoću mape visina. Efekat kaustike se realizuje kroz dva crtanja. Prilikom prvog crtanja vodena površ se crta u teksturu tako što se na nju gleda iz tačke izvora svetlosti. Svaka ćelija ove teksture na kojem se nalazi deo vodene površi čuva koordinate preseka zraka koji je udario u datu tačku vodene površi i zemlje ispod nje, tačnije refraktovanog zraka koji potiče od date tačke interakcije i zemlje ispod nje. Obzirom na činjenicu da je zemlja predstavljena pomoću mape visina, potrebno je čuvati samo  $(x, y)$  koordinate preseka. Ovo ostavlja slobodan prostor u ćeliji teksture koji se može iskoristiti za smeštanje doprinosa datog zraka celokupnom osvetljenju tog dela zemlje. Doprinos zraka zavisi od Fresnel-ove jednačine, proputovane putanje kroz vodu i upadnog ugla.

Naredni korak je formiranje teksture iluminacije (engl. *illumination texture*). Ona se formira tako što se prethodno dobijen tekstura obilazi ćeliju po ćeliju i iz svake se čitaju koordinate preseka refraktovanog zraka sa zemljom ispod vodene površi i njegov doprinos osvetljenju. Pročitani doprinos biće dodat vrednosti ćelije teksture iluminacije čije su koordinate jednake koordinatama preseka refraktovanog zraka i zemlje. Stoga može se zaključiti da vrednost svake ćelije teksture iluminacije predstavljati sumu doprinosa refraktovanih zraka koji su udarili u odgovarajuću tačku zemlje. Ova tekstura biće korišćena prilikom drugog crtanja kada će biti implementiran efekat kaustike.

Međutim, tekstura iluminacije je puna šuma, tako da je pre upotrebe teksture potrebno eliminisati ga. To se postiže primenom odgovarajućih filtera za uklanjanje šuma, ali nije pogodno koristiti bilo koji filter. Na primer, šum bi bio eliminisan primenom Gausovog filtera, ali bi takođe bili eliminasi oštri oblici koji su neophodni za prikazivanje što realističnijeg efekta kaustike. Stoga potrebno je koristite filtere koji eliminišu šum, ali takođe čuvaju oštrinu oblika smeštenih u tekstuру, kao što je na primer bilateralni filter. Celokupni postupak se može ilustrovati slikom 22.



Slika 22. (a) - efekat kaustike, (b) - tekstura preseka refraktovanih zraka i zemlje i njihovih doprinosa, (c) - inicijalna tekstura iluminacije, (d) - tekstura iluminacije nakon primene filtera [GalChi2006]

## 4. Implementacija

U ovom poglavlju biće predstavljena jedna moguća implementacija vodene površi. Implementacija će biti podeljene po malo drugačijem principu nego prethodna poglavlja. Najpre će biti opisan način na koji su implementirane refleksije i refrakcije, zatim simulacija kretanje vodene površi, zatim Fresnel-ov efekat, zatim spekularno osvetljenje na vodenoj površi i na kraju implementacija efekta kaustike. Razlog za ovakav pristup je taj što je u ovoj implementaciji vodena površ samo jedan pravougaonik, a ne mreža trouglova. Nema nikakvih simulacija kretanja vodene površi 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 biblioteci 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 refleksiona i refrakciona tekstura tako što je u njih crtan određeni deo scene bez vodene površi. Najpre je generisana refleksiona tekstura, odnosno deo scene iznad vodene površi je crtan u datu teksturu. Međutim, pre samog crtanja, scena je invertovana, odnosno primenjena je transformacija "preslikavanja u ogledalu", gde je ogledalo  $XoZ$  ravan. Ovo se može postići transformacijom skaliranja sa faktorom  $-1$  u pravcu  $y$  ose, ali je ovde realizovano 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 refleksione tekture, generisana je refrakciona tekstura. Ovo je postignuto crtanjem dela scene ispod vodene površi u datu teksturu. Za razliku od refleksione tekture, prilikom generisanja refrakcione tekture scena nije transformisana ni na koji način.

Za dobijanje gore navedenih tekstura korišćen je koncept bafera slike (engl. *framebuffer[OpenGLWikiA]*). Pre samog prikazivanja scene na ekranu, ona najpre crta u bafer slike, pa se potom sadržaj ovog bafera prikazuje na ekranu. Sam bafer slike ne postoji fizički u memoriji, već predstavlja 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 se pored glavnog bafera slike, u kojem je smeštena scena koja će biti prikazana na ekranu, kreira 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, kao komponente ovih bafera mogu se koristiti teksture. Efekat ovoga je da će se scena crtati u neku teksturu, a ne u glavni bafer slike. Kasnije, u toku prikazivanja scene na ekranu, odnosno crtanja scene u glavni bafer slike, ove teksture se mogu koristiti na uobičajeni način.

Prilikom implementacije refleksije i refrakcije kreirana su dva nova bafera slike. Jedan se koristiti za generisanje refleksione teksture, a drugi za generisanje refrakcione teksture. Najpre je potrebno alocirati nove baferne slike i oni se, kao i svi ostali objekti u OpenGL biblioteci, alociraju pozivom *glGen* funkcije, konkretno funkcijom *glGenFrameBuffer*, koja vraća celobrojni identifikator novokreiranog bafera slike. Nakon toga je potrebno dodati baferu slike bafer boje i bafer dubine, odnosno teksture koje će se koristiti kao bafer boje i bafer dubine. Ovo se postiže pozivom funkcije *glFramebufferTexture2D* gde se kao parametri prosleđuju celobrojni identifikatori tekstura koje će se koristiti kao bafer dubine, odnosno bafer boje. Nakon ovoga ovi baferi se mogu koristiti za crtanje scene. Pre samog crtanja scene potrebno je specificirati koji se bafer slike koristi. Ovo se postiže pozivom funkcije *glBindFramebuffer* kojoj se kao parametar prosleđuje identifikator željenog bafera slike. Sve što se su bude crtalo nakon ovog poziva biće smešteno u odabrani bafer slike.

Međutim, neće se cela scena videti na vodenoj površini nakon refleksije, već samo deo iznad nje. Takođe, neće se cela scena videti na vodenoj površini nakon refrakcije, već samo deo ispod nje. Stoga je potrebno na neki način ukloniti deo scene ispod vodene površi prilikom generisanja refleksione teksture, odnosno deo scene iznad vodene površi prilikom generisanja refrakcione teksture. Srećom, OpenGL biblioteka pruža jednostavan način za uklanjanja delova scene u vidu odsecajućih ravnih (engl. *clipping plane*). Naime, moguće je specificirati ravan koja će ukloniti deo scene koji se nalazi ispod nje i ovaj deo scene neće biti prikazan na ekranu, odnosno neće biti smešten u bafer slike. U OpenGL biblioteci ovo je implementirano pomoću ugrađene nizovne promenljive u *vertex shader*-u. Ova nizovna promenljiva se zove *gl\_ClipDistance* i svaki element ove nizovne promenljive odgovara jednoj odsecajućoj ravni. Njena dužina, odnosno broj odsecajućih ravnih, zavisi od karakteristika grafičkog procesora. Koristi tako što se u *vertex shader*-u, prilikom obrade svake tačke, rastojanje trenutno obrađivane tačke od odsecajuće ravnih smešta u odgovarajući element nizovne promenljive. Ukoliko je to rastojanje pozitivno data tačka će biti prikazana na ekranu, u suprotnom biće eliminisana iz scene.

Pre samog korišćenja odsecajućih ravnih potrebno ih je omogućiti. Ovo se postiže pozivom funkcije *glEnable* kojoj se kao parametar prosleđuje identifikator odsecajuće ravnih koja će biti korišćena. U ovoj implementaciji 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  $A, B$  i  $C$  komponente vektora normale ravnih, a  $D$  rastojanje ravnih od koordinatnog početka. Ova jednačina će biti prosleđena kao uniformna promenljiva *vec4* tipa koja je u *vertex shader*-u nazvana *clippingPlane0*. Da bi se dobilo rastojanje tačke od odsecajuće ravnih potrebno je samo pomnožiti poziciju tačke u koordinatnom sistemu sveta sa četiri koeficijenta iz jednačine odsecajuće ravnih, što se postiže pozivom *dot* funkcije OpenGL biblioteke. Na kraju je potrebno izračunato rastojanje upisati u odgovarajući element nizovne promenljive *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 ravnii

Cela scena biće crtana tri puta. Prvi put će biti crtan deo scene iznad vodene površi u prvi novokreirani bafer slike. Drugi put će biti crtan deo scene ispod vodene površi 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 refleksiona tekstura i refrakciona tekstura. Refleksiona i refrakciona tekstura su date na slikama 23 i 24.



Slika 23. Refleksiona tekstura



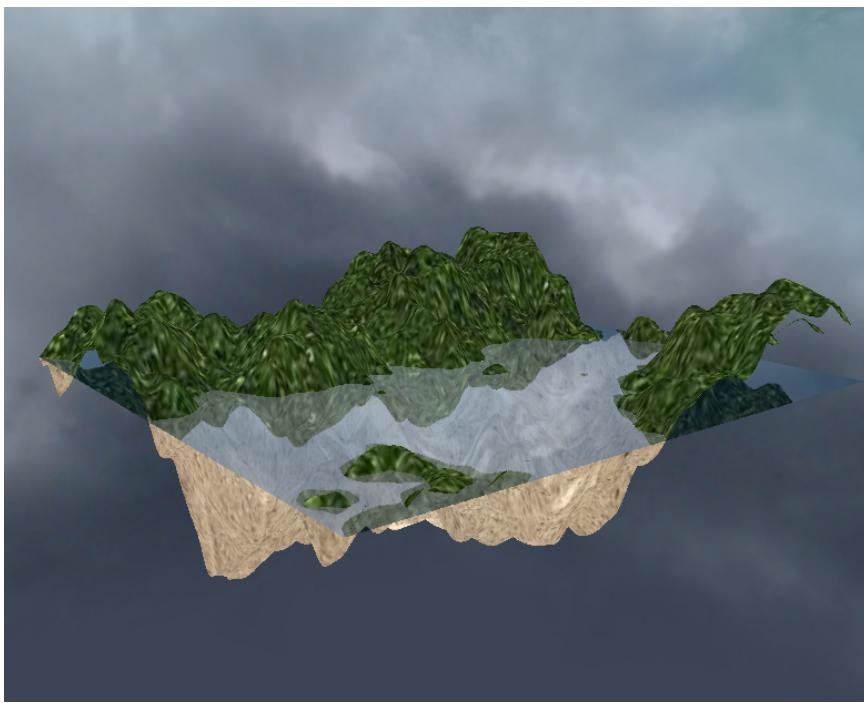
Slika 24. Refrakciona tekstura

Prilikom crtanja vodene površi biće korišćeno projektujuće teksturiranje (engl. *projection texturing*), odnosno teksture će biti uzorkovane na takav način da se postigne isti efekat kao kada bi se prikazivala pomoću projektor-a na platno. Pozicija projektor-a je ista kao pozicija kamere, a kao platno biće korišćenja vodena površ. Da bi se postigao ovaj efekat potrebno su koordinate tačaka vodene površi iz prostora odsecanja (engl. *clip space*), tačnije  $x$  i  $y$  koordinata iz prostora odsecanja. Ove koordinate su poznate u *vertex shader-a* nakon primene transformacija, stoga ih je potrebno proslediti u *fragment shader*. Nakon prosleđivanja, na koordinate se prvo primenjuje perspektivno deljenje sa  $w$  koordinatom. Ovako dobijene koordinate su u opsegu  $[-1,1]$ , tako da ih je pre upotrebe potrebno prebaciti u opseg  $[0,1]$ . Ovo se postiže njihovim deljenjem sa 2 i dodavanjem 0.5 na dobijenu vrednost. Poslednje o čemu treba voditi računa je to da je u prirodi prikaz nekog objekta na vodenoj površini prilikom refleksije invertovan, tako da  $y$  koordinata sa kojom se uzorkuje refleksionu teksturu 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. 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 25.



Slika 25. Efekat refleksije i refrakcije

## 4.2. Simulacija

Za simulaciju kretanja vodene površi nije primenjen nijedna tehnika navedena u poglavljiju 3.1, već se koristila metoda koja daje samo privid kretanje vodene površi. Za implementaciju simulacije korišćena je dUDV teksturu. Ova tekstura je RGBA teksturu kod koje se koristi samo crveni i zeleni kanal. Vrednosti iz ovih kanala korišćene su za ubacivanje distorzije, odnosno pomeraja, prilikom uzorkovanja refleksione i refrakcione teksture. Međutim, vrednosti koje se nalaze u ovim kanalima su pozitivne, a potreban je i negativan pomeraj. Tako da će se, pre konačnog korišćena, ove vrednosti prvo pomnožiti sa 2 i od dobijene vrednosti oduzeti 1. Na ovaj način vrednosti su prebačene iz opsega [0,1] u opseg [-1,1].

Naravno, da bi se postigao efekat kretanja, distorziju je potrebno menjati u toku vremena. U tu svrhu ubaćena je uniformna promenljiva tipa *float* nazvana *moveFactor* koja se menja prilikom crtanja svake slike i koja se koristi za računanje pomeraja prilikom uzorkovanje dUDV teksture. Distorzija smeštena u ovoj tekstuuri je velika, odnosno vrednosti u crvenom i zelenom kanalu su velike. Da bi se ovo rešilo ubaćena je dodatana uniformna promenljiva tipa *float* koja predstavlja faktor za skaliranje distorzije i koja efektivno 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;

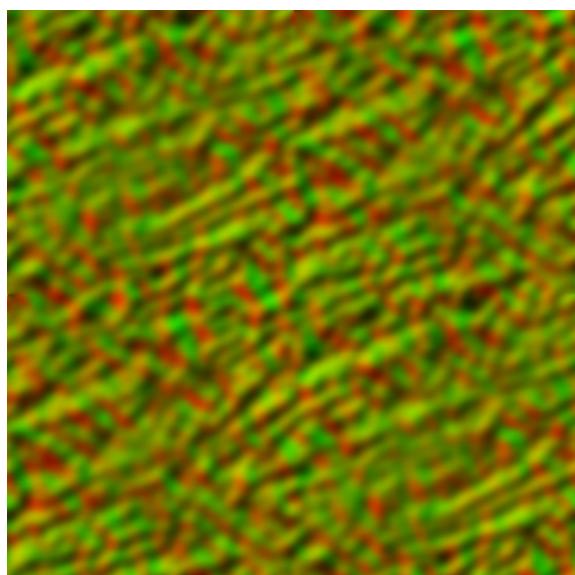
vec2 reflectionTextureCoordinates = vec2(
    refractionTextureCoordinates.x,
    1 - refractionTextureCoordinates.y
);
reflectionTextureCoordinates += totalDistortion;

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

```

*Listing 5. Kod za implementaciju prividnog kretanja vodene površi*

Na slici 26 prikazana je korišćena dUDV tekstura.



*Slika 26. dUDV tekstura*

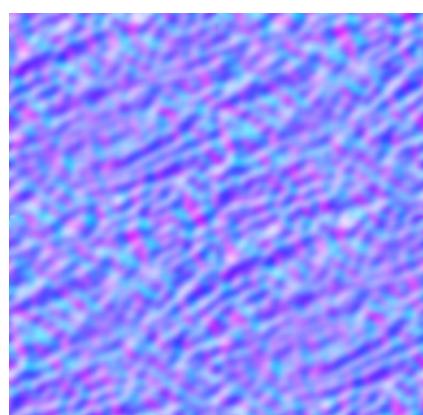
Na slici 27 prikazan je rezultat dobijen ovom tehnikom.



Slika 27. Konačan efekat nakon primenjivanja distorzije na reflektujuću i refraktujuću teksturu

### 4.3. Fresnel-ov efekat

Sledeća stvar koja je dodata implementaciji je Fresnel-ov efekat. Kao što je navedeno u poglavlju 3.2.2 Fresnel-ov efekat je komplikovan za egzaktnu računica zato se često koriste aproksimacije koje zavise samo od ugla između vektora pogleda i vektora normale. U ovoj implementaciji korišćena je slična aproksimacija. Ukoliko su oba vektora jedinični taj ugao je lako dobiti i računa se samo jednim pozivom *dot* funkcije koju pruža OpenGL biblioteka. Kao što je opisano u poglavlju 3.2.2, što je ugao između ova dva vektora manji to je dominantnija boja dobijena uzorkovanjem refleksione teksture, a što je ugao između ova dva vektora veći to je dominantnija boja dobijena uzorkovanjem refrakcione teksture. Naravno, za implementaciju ovoga potreban je vektor normale u dатој таčки водене површи. Za dobijanje vektora normale u određenoj таčki korišćena je mapa normala (engl. *normal map*). Ona je data na slici 28.



Slika 28. Mapa normala

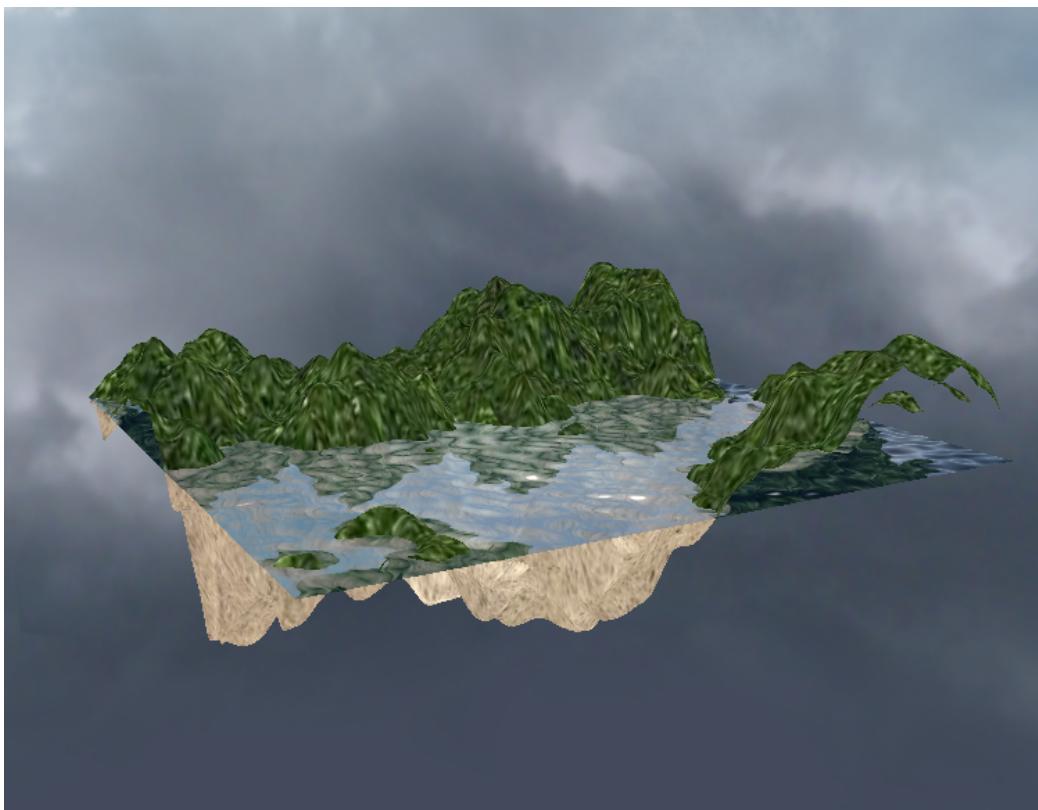
Mapa normala je RGBA tekstura, gde crveni, zeleni i plavi kanal svake ćelije predstavljaju  $x, z$  i  $y$  komponente vektora normale, respektivno. Kao što se može primetiti ova tekstura je plavičasta što i ima smisla jer su sve normale usmerene na gore pa je plavi kanal, koji predstavlja  $y$  komponentu vektora normale, dominantan. Međutim, sve tri komponente jedne ćelije su uvek pozitivne, a normala može imati i negativne komponente. Stoga je potrebno izvršiti konverziju pročitanih komponenti iz opsega  $[0, 1]$  u opseg  $[-1, 1]$ . To se postiže množenjem pročitanih vrednosti iz crvenog i zelenog kanala sa 2 i umanjivanjem dobijenog proizvoda za 1. Ovako dobijen vektor normale ne mora biti jednični vektori, stoga se mora normalizovati pre upotrebe.

Još jedna stvar o kojoj se mora voditi računa je uzorkovanje same mape normala. Mapa prikazana na slici 28 odgovara dUdV teksturi iz prethodnog odeljka, odnosno odgovara teksturi pomoću koje su simulirani talasi. Tako da se mapa normala mora uzorkovati sa istim koordinatama kao i dUdV teksturu, naravno uz odgovarajući pomeraj u toku vremena, da bi se dobio što realističniji efekat. Kod za uzorkovanje mape normala i normalizaciju vektora normale dat je na listingu 6.

```
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 29.



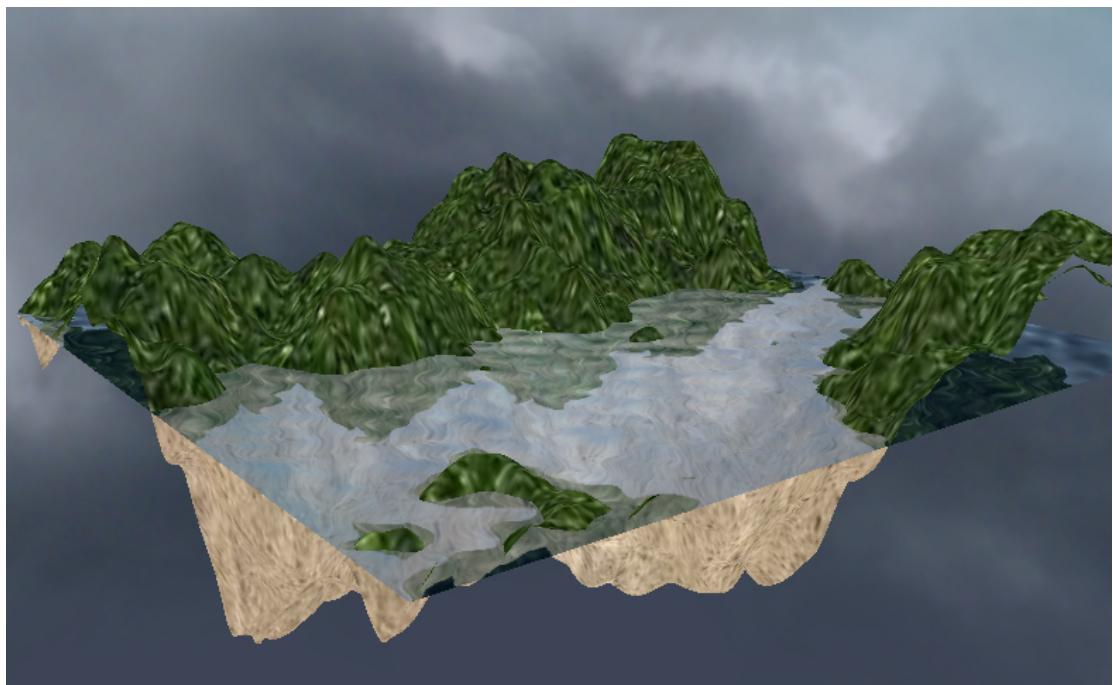
Slika 29. Fresnel-ov efekat

Kao što se može videti, normale, iako sve uperene na gore, su raštrkane i zbog toga nije dobijen realističan prikaz vodene površi. Ovo se može korigovati množenjem  $y$  komponente vektora normale sa određenim faktorom tipa *float* koji je nazvan *normalEqualizationFactor*. Nakon toga, prilikom normalizacije vektora normala, povećana  $y$  komponenta će doprineti tome da vektori normala ne budu toliko raštrkani, već većim delom usmereni na 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 30.



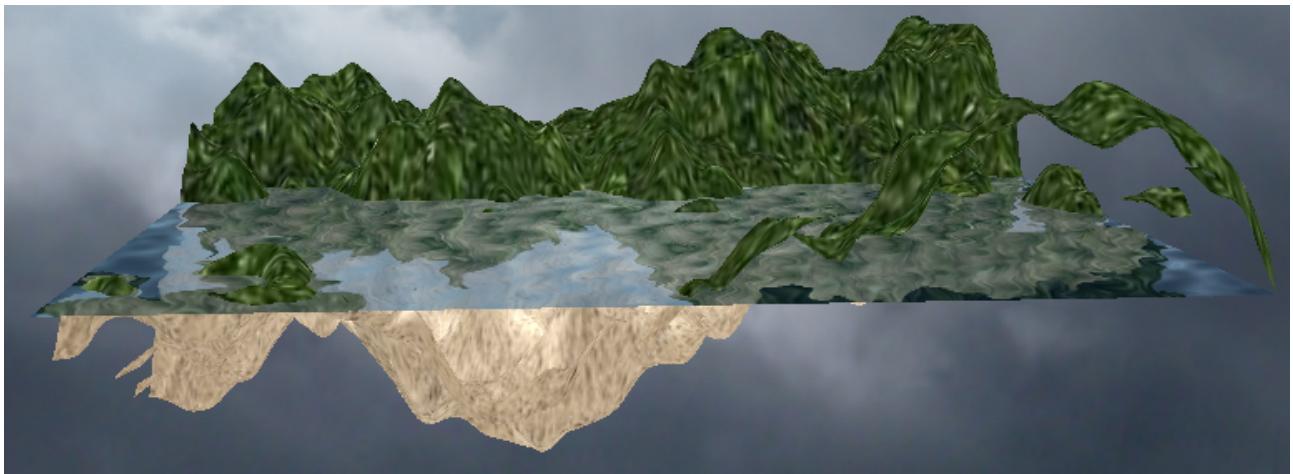
Slika 30. Popravljeni Fresnel-ov efekat

Pravi efekat Fresnel-ove jednačine se može videti promenom ugla gledanja. Ukoliko je ugao između vektor pogleda i vektora normale veći biće jači efekat refrakcije kao što se vidi na slici 31.



Slika 31. Refrakcija sa Fresnel-ovom jednačinom

Ukoliko je ugao između vektor pogleda i vektora normale veći biće jači efekat refleksije što se vidi na slici 32.



Slika 32. Refleksija sa Fresnel-ovom jednačinom

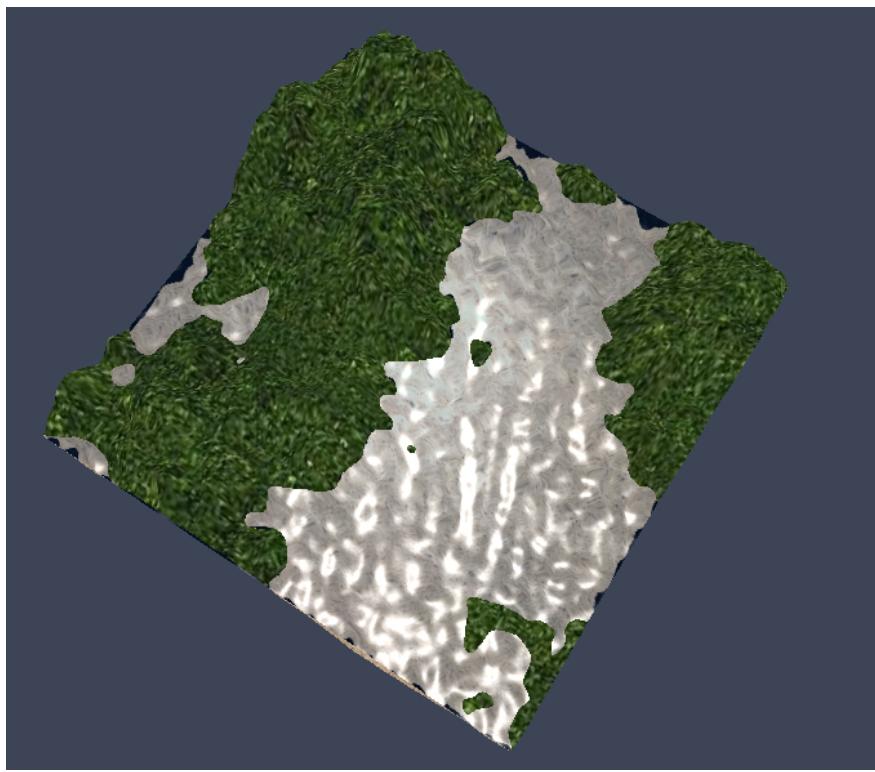
## 4.4. Spekularno osvetljenje

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 dатој таčки izračunat je vektor reflektovanog zraka. Sledeći korak je računanje ugla 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 se može dobiti pozivom *dot* funkcije koju pruža OpenGL biblioteka. 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 vrednosti se došlo eksperimentalnim putem. Kod pomoću kojeg 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 impletaciju spekularnog osvetljenja

Nakon ovog proračuna, *specularHighlights* vektor je jednostavno dodat boji koja je dobijena mešanjem vrednosti pročitanih iz refleksione i refrakcione teksture. Efekat ovog se može videti na slici 33.



Slika 33. Efekat spekularnog osvetljenja

## 4.5. Kaustika

Kao što je navedeno u odeljku 3.2.3, efekat kaustika je veoma teško implementirati i gotovo ga je nemoguće postići bez korišćenja tehnike praćenja zraka. U ovoj implementaciji 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 efekat kaustike utiču samo zraci refraktovani pod pravim uglom. Na ovaj način se sa svakog piksela zemlje ispod vodene površi može unazad pratiti refraktovani zrak do vodene površi i na osnovu njega odrediti originalni upadni zrak. Ovaj zrak ne potiče od svetlosnog izvora pa je nakon toga potrebno odrediti ugao između izračunatog vektora upadnog zraka i vektora zraka koji potiče od svetlosnog izvora. Što je ovaj ugao manji to je piksel zemlje svetlijiji.

Vektor upadnog zraka se dobija iz Fresnel-ove jednačine. Računicu je lako objasniti posmatranjem refrakcije na slici 1. Pre same računice potrebno je uvesti dve pretpostavke. Prva je da postoji još jedan vektor  $\vec{a}$  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. Druga je da vektor upadnog zraka  $\vec{v}$  ima suprotan smer od onog na slici 1. Korišćenjem vektora  $\vec{a}$  izvedene su jednačine za vektor upadnog zraka  $\vec{v}$  i vektor refraktovanog zraka  $\vec{v}_t$  i one glase:

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

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

U prethodnoj jednačini sa  $\vec{n}$  je ozančen vektor normale, sa  $\theta$  ugao između vektora upadnog zraka i vektora normale, a sa  $\theta_t$  ugao između vektora refraktovanog

zraka i inverznog vektora normale. Sledeći korak je određivanje jednačina vektora  $\vec{a}$  iz prethodne dve jednačine.

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

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

Izjednačavanjem dobijenih jednačina vektora  $\vec{a}$  moguće je odrediti jednačinu vektora upadnog zraka. Postupak je sledeći:

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

Dalje, korišćenjem Snell-ovog zakona i trigonometrijskih identiteta može se doći do sledeće formule za vektor upadnog zraka:

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

U prethodnoj jednačini sa  $n_1$  i  $n_2$  su označani indeksi refrakcije vazduha i vode, respektivno. Ukoliko su inverzni vektor normale i vektor refraktovanog zraka jedinični vektori, ugao  $\theta_t$  koji zaklapaju ova dva vektora se dobija jednostavnim pozivom *dot* funkcije koju pruža OpenGL biblioteka. Vektor upadnog zraka  $\vec{v}$  se dobija iz poslednje jednačine. Naravno, treba voditi računa da vektor upadnog zraka nije jedinični vektor pa ga je potrebno normalizovati. Poslednji korak je nalaženja ugla između vektora upadnog zraka i vektor koji se prostire od date tačke vodene površi do izvora svetla. Ovaj ugao se može iskoristiti na isti način kao kod spekularnog osvetljenja.

Za implementaciju efekta 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 efekta kaustike dat je na listingu 9.

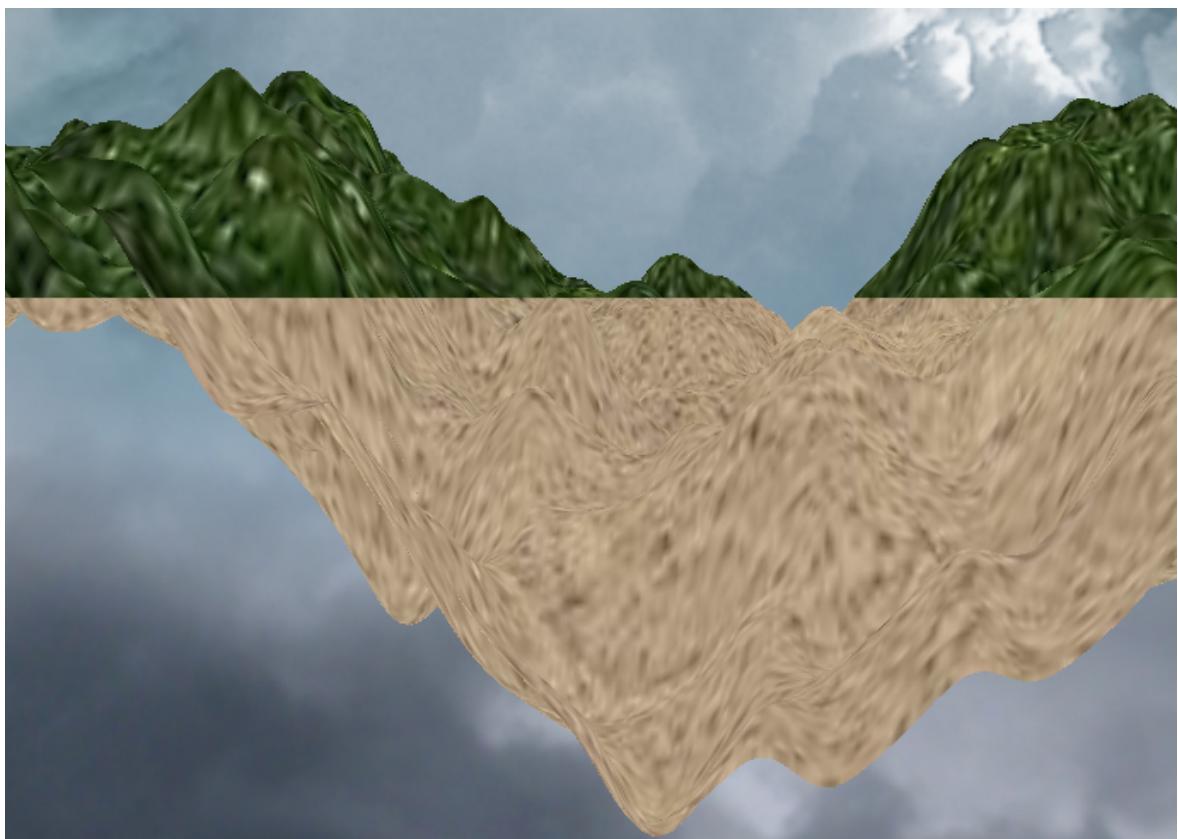
```

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
);

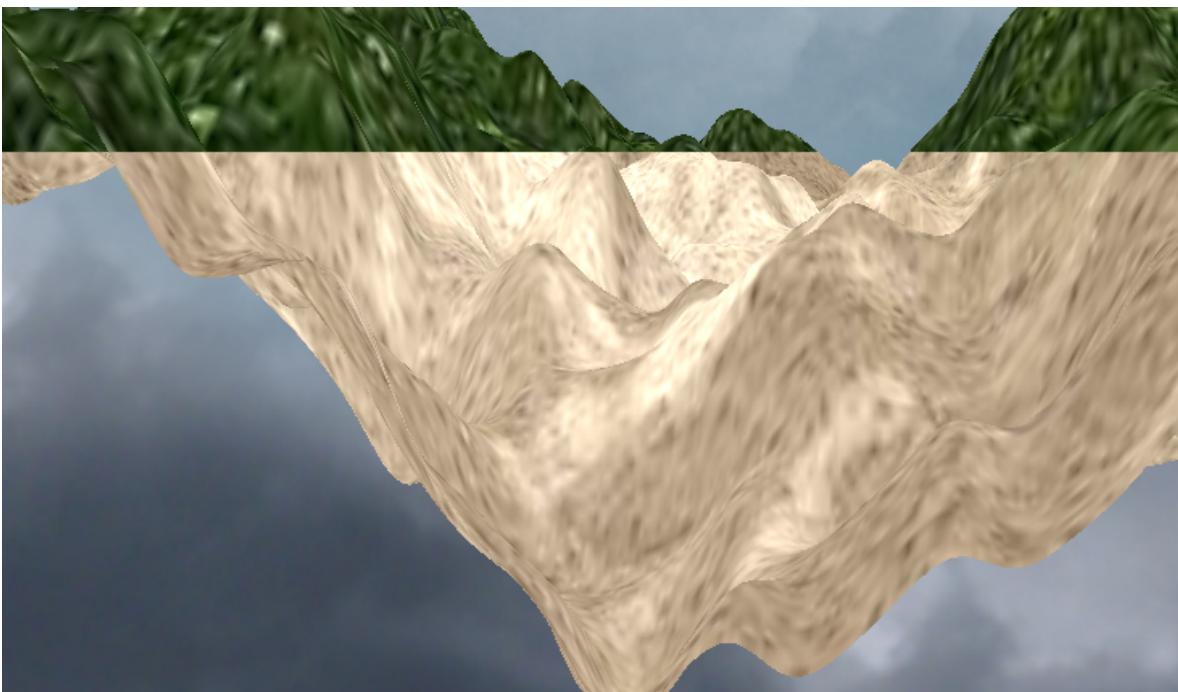
```

*Listing 9. Kod za implementaciju efekta kaustike*

Konstanta **eta** predstavlja odnos indeksa refrakcije vode i indeksa refrakcije vazduha i ima vrednost 0.75. Na slikama 34 i 35 dati su izgledi površi bez i sa efektom kaustike, respektivno.



Slika 34. Površ bez efekta kaustike



Slika 35. Površ sa efektom kaustike

Kao što se može videti sa slike 35 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 tehnike praćenja zraka, a samim tim nema prevelikog zauzeća računarskih resursa.

## 5. Zaključak

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

Glavni problemi na koje se nailazi prilikom crtanja su simulacija kretanje vodene površi i implementacija optičkih efekata, od kojih refleksija i refrakcija najviše doprinose realističnom izgledu vodene površi. Pored ova dva efekta dodat je i efekat kaustike zato što i on 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. Opisi tehnika uključuju i komentare o kojima treba voditi računa prilikom primene date tehnike koji se tiču performansi i detalja implementacije. Na kraju je dat opis jedne moguće realizacije vodene površi, koja ilustruje prethodno predstavljene probleme i njihova rešenja.

Međutim, ovo nije kraj jer crtanje vodenih površina je jako zahtevan zadatak i svakim danom se pronalaze nove tehnike koje još više doprinosi realističnom izgledu vodenih površi. Sledeći korak u daljem istraživanju bi bio da se detaljnije istraže tehnika praćenja zraka i njenog 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 vodene površi 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%E2%80%93Stokes\\_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](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

[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

[WikiD] "Heightmap", <https://en.wikipedia.org/wiki/Heightmap>, 06.04.2018

[Joh2004] Claes Johanson, "Real-time water rendering, Introducing the projected grid concept ", Lund University, , 2004

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

[WikiE] "Curve fitting", [https://en.wikipedia.org/wiki/Curve\\_fitting](https://en.wikipedia.org/wiki/Curve_fitting), 04.03.2018

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

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

[WikiF] "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

[WikiG] "Schlick's approximation", [https://en.wikipedia.org/wiki/Schlick%27s\\_approximation](https://en.wikipedia.org/wiki/Schlick%27s_approximation),

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

# Slike

Slika 1. Refleksija (levo) i refrakcija (desno)[Fle2007].....	3
Slika 2. Efekat kaustike.....	4
Slika 3. Usmereni (levo) i kružni (desno) talasi[FerKir2004].....	7
Slika 4. Gerstner-ovi talasi.....	8
Slika 5. Mapa visina[WikiD].....	9
Slika 6. Teren dobijen pomoću mape visina[WikiD].....	9
Slika 7. Ograničenje tehnika koje koriste mape visina[Joh2004].....	10
Slika 8. Kubna kriva[Zel2004].....	12
Slika 9. 2D jednačina talasa[Zel2004].....	13
Slika 10. Perlin-ov šum.....	14
Slika 11. Suma 6 oktava Perlin-ovog šuma.....	15
Slika 12. 3D Perlin-ov šum.....	15
Slika 13. FFT mapa visina[Tes2001].....	17
Slika 14. Algoritam praćenja zraka sa jednim nivoom rekurzije[GalChi2006].....	18
Slika 15. Pseudo kod algoritma praćenja zraka[GalChi2006].....	19
Slika 16. Različite putanje zraka[GalChi2006].....	20
Slika 17. Zrak izlazi van granica nakon refrakcije[GalChi2006].....	20
Slika 18. Odnos Fresnel-ove jednačine i njenih aproksimacija.....	22
Slika 19. Odnos vrednosti Fresnel-ove jednačine i vrednosti dobijenih sačuvanih u teksturama.....	23
Slika 20. Uvećanje slike 19.....	23
Slika 21. Projektovani trouglovi[Fle2007].....	24
Slika 22. (a) - efekat kaustike, (b) - tekstura preseka refraktovanih zraka i zemlje i njihovih doprinosa, (c) - inicijalna tekstura iluminacije, (d) - tekstura iluminacije nakon primene filtera[GalChi2006].....	25
Slika 23. Refleksiona tekstura.....	28
Slika 24. Refrakciona tekstura.....	28
Slika 25. Efekat refleksije i refrakcije.....	29
Slika 26. dUDV tekstura.....	30
Slika 27. Konačan efekat nakon primenjivanja distorzije na reflektujuću i refraktujuću teksturu.....	31
Slika 28. Mapa normala.....	31
Slika 29. Fresnel-ov efekat.....	33
Slika 30. Popravljeni Fresnel-ov efekat.....	34
Slika 31. Refrakcija sa Fresnel-ovom jednačinom.....	34
Slika 32. Refleksija sa Fresnel-ovom jednačinom.....	35
Slika 33. Efekat spekularnog osvetljenja.....	36
Slika 34. Površ bez efekta kaustike.....	39
Slika 35. Površ sa efektom kaustike.....	39

# Listinzi

Listing 1. Kod za generisanje mape visina pomoću 2D jednačine talasa.....	13
Listing 2. Kod za računanje vektora normale.....	14
Listing 3. Kod za implementaciju odsecajuće ravni.....	27
Listing 4. Kod za implementaciju refleksije i refrakcije.....	28
Listing 5. Kod za implementaciju prividnog kretanja vodene površi.....	30
Listing 6. Kod za uzorkovanje i normalizaciju vektora normale.....	32
Listing 7. Kod za popravljanje normala.....	33
Listing 8. Kod za implementaciju spekularnog osvetljenja.....	35
Listing 9. Kod za implementaciju efekta kaustike.....	38

## 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
    );
}
```