SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1859

FIZIKALNO TEMELJENA IZRADA PRIKAZA

Dorijan Jančić

Zagreb, lipanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1859

FIZIKALNO TEMELJENA IZRADA PRIKAZA

Dorijan Jančić

Zagreb, lipanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. ožujka 2025.

ZAVRŠNI ZADATAK br. 1859

Pristupnik:	**Dorijan Jančić (0036550974)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Fizikalno temeljena izrada prikaza**

Opis zadatka:

U ostvarivanju prikaza u računalnoj grafici izuzetno je važna visoka kvaliteta modela izračuna osvjetljenja kako bi se postigao bilo foto-realizam bilo stilizirani vizualni učinak. Proučiti fizikalno temeljen model izračuna izrade prikaza (PBR) temeljen na funkciji dvosmjerne distribucije refleksije svjetlosti (BRDF). Posebnu pažnju posvetiti optimizacijskim postupcima u implementaciji i učinkovitom izvođenju algoritama proučenog fizikalnog modela. Načiniti testiranje i usporedbu na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 23. lipnja 2025.

Sadržaj

Uvod

Kvaliteta izrade vizualnog prikaza ovisi o načinu na koji se simulira interakcija svjetla i površine. Fizički temeljeno renderiranje (engl. *Physically Based Rendering*, PBR) predstavlja skup algoritama koji se temelje na teoriji koja pokušava što bolje modelirati svjetlost i površinu prema stvarnim fizičkim principima. Zbog toga što ova metoda pokušava što bolje modelirati svjetlost i površinu prema stvarnim fizičkim principima, generalno se čini realističnijom nego stariji standardi poput Blinn-Phong sjenčanja. Cilj ovog rada je implementirati skup algoritama koji se mogu koristiti za prikaz u stvarnom vremenu, te steći uvid u teorijsku osnovu modernih sustava za renderiranje, poput onih korištenih u Unreal Engineu, Unityu, Blenderu itd. Radi jednostavnosti, implementacija će se provesti u Unityu, no princip se može primijeniti na bilo koju drugu aplikaciju ili grafičko sučelje.

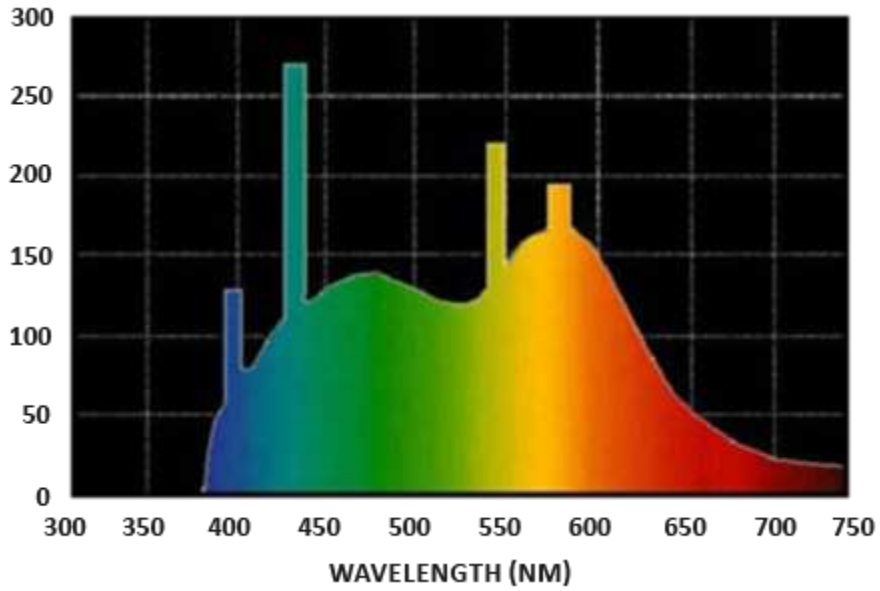
1. Teorijska osnova

1.1. Radiometrijske mjere

Radiometrija je grana optike koja se bavi mjerenjem svojstva elektromagnetskih valova. Pruža skup ideja i matematičkih alata koji opisuju širenje i refleksiju svjetlosti. Izvori svjetlosti emitiraju fotone, pri čemu svaki foton ima određenu valnu duljinu i nosi specifičnu količinu energije. Osnovne radiometrijske veličine zapravo su različiti načini mjerenja tih fotona.

1.1.1. Tok isijavanja

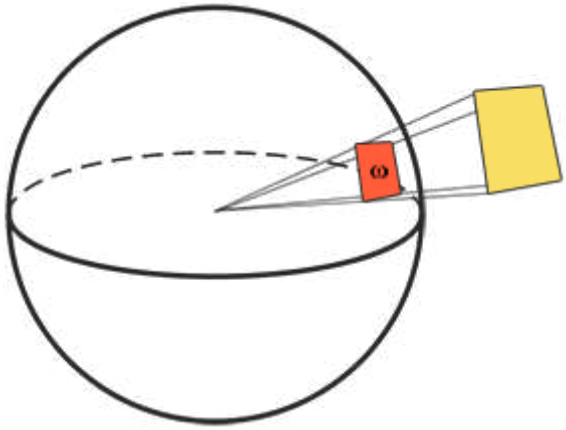
Tok isijavanja Φ (engl. *flux*) je ukupna količina energije koja prolazi kroz površinu ili dio prostora u jedinici vremena. Mjerna jedinica mu je vat (W). Svjetlost možemo promatrati kao kombinacija energije preko više različitih valnih duljina, pri čemu je svaka valna duljina povezana s određenom bojom. Emisija svjetlosti nekog izvora može se stoga gledati kao funkcija raspodjele energije po valnim duljinama. Valne duljine od 390nm do 700nm čine vidljivi spektar, tj. one valne duljine koje ljudsko oko može percipirati. Ukupni tok isijavanja odgovara površini ispod te spektralne funkcije (Slika 1.1). Korištenje takve funkcije bilo bi nepraktično u računalnoj grafici, stoga se svjetlost prikazuje kao RGB vektor. Ovakav način kodiranja uzrokuje gubitak informacija, ali je zanemariv kada je u pitanju vizualni dojam.



Slika 1.1 Različite duljine po valnoj duljini dnevnog svjetla [1]

1.1.2. Prostorni kut, obasjanost i sjajnost

Prostorni kut ω (engl. *solid angle*) predstavlja veličinu površine projiciranu na jediničnu kuglu. Mjerna jedinica je steradijan (sr).



Slika 1.2 Solidan kut kao direkcija sa volumenom [1]

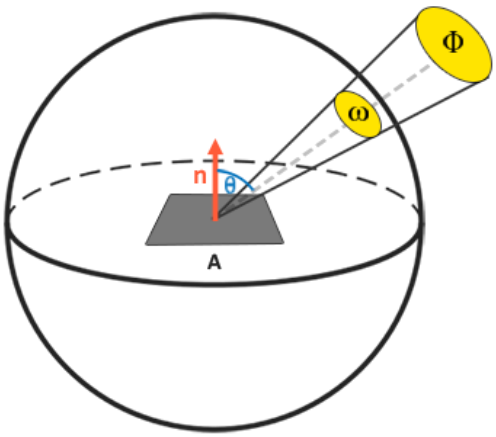
Obasjanost E (engl. *irradiance*) je količina svjetlosne energije koja dolazi na površinu. Označava snagu zračenja po jedinici površine i mjeri se u vatima po kvadratnom metru (Wm^{-2}).

$$E = \frac{d\Phi}{dA} \tag{1}$$

Obasjanost je skalarna veličina i ne uzima u obzir smjer iz kojeg svjetlost dolazi.

Sjajnost L (engl. *radiance*) je ukupna količina svjetlosne energije koja izlazi iz točke na površini u određenom smjeru. Mjeri se u vatima po kvadratnom metru po steradijanu ($Wm^{-2}sr^{-1}$). Radi se o vrlo preciznoj i lokalnoj mjeri koja uzima u obzir i smjer i kut pod kojim se zračenje širi.

$$L = \frac{d^2\Phi}{dAd\omega\cos\theta} \tag{2}$$



Slika 1.3 Prikaz gustoće toka isijavanja po jedinici površine i po jedinici prostornog kuta u određenom smjeru [1]

Sjajnost je temeljena veličina u računalnoj grafici, jer omogućuje simulaciju kako svjetlost putuje kroz scenu. Ako nas zanima sva moguća sjajnost koja dolazi na neku površinu, potrebno je izračunati sjajnost iz svih mogućih smjerova, odnosno obasjanost.

$$E = \int_{\Omega} L \cos\theta \, d\omega \tag{3}$$

Sjajnost je ključna fizikalna veličina u računalnom osvjetljenju jer omogućuje izračun svih ostalih radiometrijskih mjera. U sjenčarima upravo se sjajnost računa za svaki fragment, jer ona definira konačne boje piksela koje se prikazuju na ekranu.

1.2. Jednadžba renderiranja

Konstruiranje modela koji savršeno opisuje interakciju svjetla i površine u prirodi nije izvedivo zbog iznimne složenosti samog problema. Umjesto toga, u praksi se najčešće primjenjuju aproksimacije koje su računalno manje zahtjevne. Jedna od najpoznatijih takvih aproksimacija je jednadžba renderiranja. Jednadžba renderiranja je matematička formula koja na najvećoj razini apstrakcije opisuje prijenos svjetlosti unutar scene. Uveli su je David Immel i James Kajiya 1986. godine, a od tada ova jednadžba predstavlja temelj gotovo svih modernih algoritama za realističan prikaz slike. Opći oblik jednadžbe renderiranja glasi:

$$L_o(\boldsymbol{p}, \boldsymbol{\omega_o}) = L_e(\boldsymbol{p}, \boldsymbol{\omega_o}) + \int_{\Omega} f(\boldsymbol{p}, \boldsymbol{\omega_i}, \boldsymbol{\omega_o}) L_i(\boldsymbol{p}, \boldsymbol{\omega_i})(\boldsymbol{\omega_i} \cdot \boldsymbol{n}) \, \mathrm{d}\boldsymbol{\omega_i} \tag{4}$$

Rezultat jednadžbe je izlazni sjaj L_o iz neke točke p u smjeru ω_o. Prvi član sume predstavlja emitirani sjaj, količinu svjetlosti koju površina sama emitira. Drugi član opisuje reflektirani

sjaj, svjetlost koja dolazi iz drugih smjerova, reflektira se od površine i doprinosi ukupnom sjaju u promatranom smjeru ω_o. Reflektirani sjaj računa se na temelju ulaznog prostornog kuta ω_i, koji je beskonačno mali te se može smatrati vektorom. Na mikroskopskoj razini, površinu oko točke p možemo promatrati kao ravninu. Oko točke p konstruira se jedinična hemisfera Ω, imaginarna polukugla usmjerena prema normali površine n. Svaka ulazna zraka doprinosi rezultatu ovisno o svojoj sjajnosti L_i, kutu upada cosθ i funkciji dvosmjerne distribucije refleksije f. Skaliranje po kutu upada, poznato kao Lambertov zakon, izvedeno je preko skalarnog produkta jediničnih vektora $\omega_i \cdot n$. Funkcija f bit će objašnjena u zasebnom poglavlju. Integral u jednadžbi označava zbrajanje doprinosa svih mogućih ulaznih zraka nad hemisferom, odnosno ukupnu obasjanost točke p. Nije praktično računati cijeli integral za svaki piksel na ekranu, stoga će se koristiti aproksimacije.

2. Funkcija dvosmjerne distribucije refleksije

U jednadžbi renderiranja spomenuli smo funkciju f ispod integrala. Ta funkcija opisuje karakteristiku refleksije površine i zrake svjetlosti. Moderne aplikacije koriste model koji se naziva funkcija dvosmjerne distribucije raspršenosti (engl. *bidirectional scattering distribution function*, BSDF) koja se sastoji od dvije komponente:

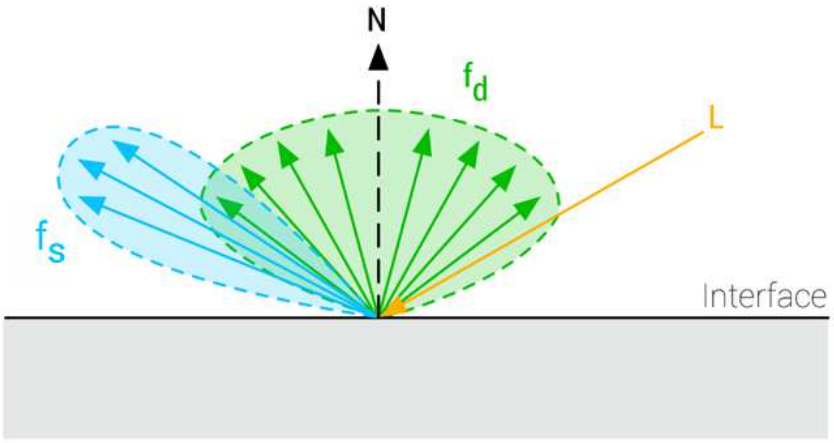
- funkcije dvosmjerne distribucije refleksije (engl. *bidirectional reflectance distribution function*, BRDF)
- funkcije dvosmjerne distribucije transmisije (engl. *bidirectional transmittance distribution function*, BTDF).

BRDF opisuje dio svjetlosti koji se reflektira prema promatraču. BTDF opisuje kako se zraka svjetlosti probija kroz površinu (transmitira) i izlazi prema promatraču. BTDF je računski složeniji i u ovom projektu neće biti implementiran. BRDF modelira refleksiju površine kao linearnu kombinaciju dvije komponente: difuzne i zrcalne.

$$f = f_d + f_s \tag{5}$$

Kada svjetlost dođe u kontakt s površinom, dio energije se apsorbira i pretvara u toplinu, dok se ostatak reflektira. Raspršeni dio reflektirane svjetlosti u svim smjerovima percipira se kao boja površine.

S druge strane, određeni dio svjetlosti neće biti raspršen, nego će se reflektirati u točno određenom smjeru, ovisno u kutu upada. Taj dio nazivamo zrcalna refleksija i ona se percipira kao (od)sjaj površine.



Slika 2.1 Prikaz zrcalne (plavo) i difuzne (zeleno) komponente na površini [2]

Modeli poput Blinn-Phonga također su BRDF funkcije, ali oni nisu fizikalno temeljeni. Da bi se BRDF smatrao fizikalno utemeljen, mora zadovoljiti dva ključna fizikalna svojstva: recipročnost i očuvanje energije.

Pravilo recipročnosti, poznato kao Helmholtzova recipročnost, je svojstvo simetrije s obzirom na smjer ulaza i izlaza svjetlosti. To znači da ako u funkciji zamijenimo ω_i i ω_o rezultat će biti isti, otuda riječ dvosmjerna u nazivu.

Očuvanje energije osigurava da količina svjetlosne energije koja izlazi iz površine ne smije biti veća od one koja dolazi, pod uvjetom da površina sama ne emitira svjetlost. Glatke površine imaju manje, ali jače zrcalne refleksije. Hrapavije površine imaju veće, ali slabije refleksije. Time se osigurava da ukupna energije ostane ista. Tip materijala također igra važnu ulogu. Dielektrici imaju i difuznu i zrcalnu komponentu refleksije, dok metali u potpunosti reflektiraju svjetlost kroz zrcalnu komponentu, ali bez difuznog doprinosa.

Postoji više modela koji fizikalno opisuju difuzne i zrcalne komponente. U ovom radu za difuznu komponentu koristiti će se Lambertov model, dok će se za zrcalnu refleksiju koristiti Cook-Torranceov model.

2.1. Difuzna komponenta

Difuzna komponenta koristi Lambertov BRDF, koji pretpostavlja da se reflektirana svjetlost raspršuje jednoliko u svim smjerovima, neovisno o kutu promatranja. Zbog te uniformnosti, količina reflektirane svjetlosti ostaje konstantna u svim smjerovima. Kako bi model zadovoljio načelo očuvanja energije, ova konstanta mora biti normalizirana dijeljenjem s π, budući da ukupna reflektirana energija ne smije premašiti dolaznu energiju.

$$f_{lambert} = \frac{c}{\pi} \tag{6}$$

Konstanta c je reflektirajuća boja materijala. Upravo difuznu refleksiju percipiramo kao boju površine, zbog čega se u izrazu koristi boja c. Implementacija Lambertovog BRDF-a:

```
1. half3 DiffuseLambert(half3 color)
2. {
3.     return color * rcp(PI);
4. }
```

2.2. Zrcalna komponenta

Zrcalna komponenta koristi Cook-Torranceov BRDF, koji koristi teoriju mikroploha. Teorija mikroploha pretpostavlja da je površina sastavljena od hrpu mikroskopskih zrcala zvanih mikroplohe. Ovisno o hrapavosti površine, poravnanje ovih sitnih malih zrcala može se dosta razlikovati. Što je površina grublja to će mikroplohe biti neskladno poravnate, dok će kog glatkih površina biti orijentirane u sličnom smjeru. Ovo rezultira da se zrake svjetlosti raspršuju u potpuno različitim smjerovima na grubim površinama, a na glatkim površinama zrake svjetlosti reflektiraju se u otprilike istom smjeru.

$$f_{cook-torrance} = \frac{DGF}{4(\boldsymbol{n} \cdot \boldsymbol{v})(\boldsymbol{n} \cdot \boldsymbol{l})} \tag{7}$$

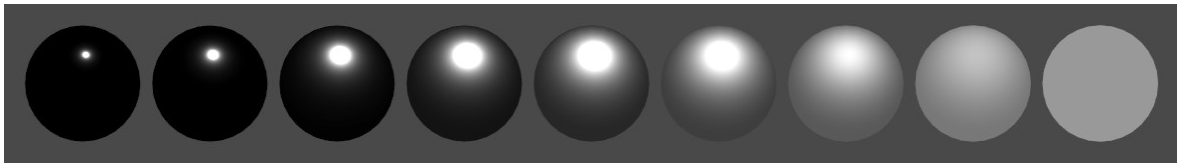
Cook-Torrance BRDF sastoji se od tri funkcije: funkcije distribucije normala, geometrijska funkcija i funkcije Fresnela.

2.2.1. Funkcija distribucije normala

Funkcija distribucije normala D (engl. *normal distribution function*, NDF) opisuje koliko mikroploha na površini je orijentirano u smjeru vektora h. Ova funkcija utječe na oblik i intenzitet zrcalne refleksije. Za izračun refleksije koristi se vektor h (engl. *halfway vector*) koji se nalazi na pola puta između vektora svjetlosti l i vektora smjera pogleda v.

$$\boldsymbol{h} = \frac{\boldsymbol{l} + \boldsymbol{v}}{\|\boldsymbol{l} + \boldsymbol{v}\|} \tag{8}$$

Samo miktroplohe koje su orijentirane u smjeru vektora h će reflektirati svjetlost. Kada je površina glađa broj mikroploha orijentiranih prema h je manji, što rezultira užom, ali jačom zrcalnom refleksijom. Obrnuto, kada je površina grublja, broj mikroploha orijentiranih prema h je veći što rezultira širu, ali slabiju zrcalnu refleksiju.



Slika 2.2 Prikaz NDF po različitim faktorima hrapavosti

Ovaj rad implementira Trowbridge-Reitz GGX distribuciju:

$$D_{GGX}(\alpha, \boldsymbol{n}, \boldsymbol{h}) = \frac{\alpha^2}{\pi((\boldsymbol{n} \cdot \boldsymbol{h})^2(\alpha^2 - 1) + 1)^2} \tag{9}$$

$$\alpha = roughness^2$$

Faktor hrapavosti (engl. *roughness*) skalarna je vrijednost koja je u rasponu [0, 1], a zadaje se kao ulazni parametar materijala. Implementacija NDF-a:

```
1. float DistributionGGX(float a2, float NoH)
2. {
3.     float d = (NoH * a2 - NoH) * NoH + 1;
4.     return a2 * rcp(PI * d * d);
5. }
```

2.2.2. Geometrijska funkcija

Geometrijska funkcija G (engl. *geometry function*) opisuje koji dio refleksije se gubi zbog toga što jedna mikroploha zasjenjuje drugu mikroploha. Funkcija aproksimira površinu gdje se njezine mikroplohe međusobno zasjenjuju. Nazivnik Cook-Torrance BRDF-a koristi se za normalizaciju geometrijske funkcije. Često modeli sadrže elemente koji se krate, zato se uvodi novi pojam, funkcija vidljivosti V (engl. *visibility function*). Ovaj pojam spaja geometrijsku funkciju i nazivnik BRDF-a u jednu funkciju.

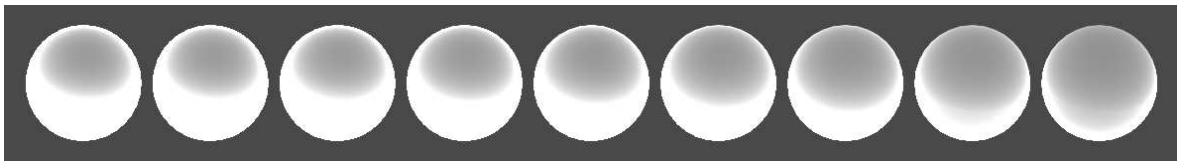
$$V(\alpha, \boldsymbol{n}, \boldsymbol{v}, \boldsymbol{l}) = \frac{G(\alpha, \boldsymbol{n}, \boldsymbol{v}, \boldsymbol{l})}{4(\boldsymbol{n} \cdot \boldsymbol{v})(\boldsymbol{n} \cdot \boldsymbol{l})} \tag{10}$$

Funkcija koju će ovaj rad koristiti je aproksimacija korelirane Smithove geometrijske funkcije.

$$V_{smith-correlated-approx}(\alpha, \boldsymbol{n}, \boldsymbol{v}, \boldsymbol{l}) = \frac{0.5}{(\boldsymbol{n} \cdot \boldsymbol{l})\big((\boldsymbol{n} \cdot \boldsymbol{v})(1 - \alpha) + \alpha\big) + (\boldsymbol{n} \cdot \boldsymbol{v})\big((\boldsymbol{n} \cdot \boldsymbol{l})(1 - a) + \alpha\big)} \tag{11}$$

Implementacija funkcije vidljivosti:

```
1. float VisibilitySmithGGXCorrelatedFast(float a2, float NoV, float NoL)
2. {
3.     float a = sqrt(a2);
4.     float smithV = NoL * (NoV * (1.0 - a) + a);
5.     float smithL = NoV * (NoL * (1.0 - a) + a);
6.     return 0.5 * rcp(smithV + smithL);
7. }
```



Slika 2.3 Prikaz funkcije vidljivosti po različitim hrapavosti

2.2.3. Funkcija Fresnela

Fresnel F opisuje promjenu količine reflektirane svjetlosti ovisno o kutu pod kojim svjetlost dolazi na površinu. Pri manjim kutovima refleksija je slabija, a pri većim kutovima refleksija postaje izraženija. Trenutak kada svjetlost pogodi površinu, na temelju kuta površine i smjera pogleda, Fresnel jednadžba nam govori koji postotak svjetlosti se reflektira. Ovaj rad implementira Schlickovu aproksimaciju:

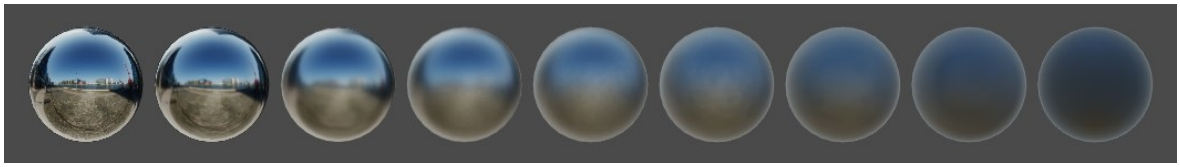
$$F_{schlick}(f_0, f_{90}, \boldsymbol{v}, \boldsymbol{h}) = f_0 + (f_{90} - f_0)(1 - \boldsymbol{v} \cdot \boldsymbol{h})^5 \tag{12}$$

Varijabla f_0 predstavlja reflektiranost pri okomitom upadu svjetlosti, kada je kut između vektora malen. Varijabla f_{90} predstavlja refleksivnost kada svjetlost upada gotovo paralelno s površinom. U praksi varijabla f_0 se postavlja na 0.04 za dielektrike, dok se za metale često definira ili računa na temelju boje površine. Varijabla f_{90} često se postavlja na 1.

```
1. float3 FresnelSchlick(float3 f0, float3 f90, float VoH)
2. {
3.     return f0 + (f90 - f0) * pow(1.0 - VoH, 5);
4. }
```

3. Osvjetljenje temeljeno na slici

U prirodi svjetlost se može odbiti od različitih objekata u okolišu prije nego dođe do oka promatrača. Cijelo okruženje oko objekta može se smatrati kao jedan veliki izvor svjetlosti. U praksi jedan od načina za obuhvatiti okoliš oko objekta je korištenjem kubnih tekstura. Kubna tekstura (engl. *cubemap*) je sastavljena od 6 različitih slika koje predstavljaju strane kocke. Okolinu možemo projicirati na jediničnu kocku i pohraniti u kubnu teksturu. Osvjetljenje temeljeno na slici (engl. *image based lighting*, IBL) je skup algoritama za osvjetljivanje objekata tretirajući sliku kao izvor svjetlosti. IBL postaje zanimljiv kada govorimo o PBR jer objekti izgledaju fizički preciznije kada uzmemo u obzir osvjetljenje okoliša, a teksture predstavljaju jednostavan zapis informacija.



Slika 3.1 Prikaz metala po različitim faktorima hrapavosti

Računanje sjajnosti kod IBL ista je kao i kod jednadžbe renderiranja. Emitirani sjaj se može zanemariti jer se računa reflektirani sjaj. BRDF funkcija se dijeli na difuznu i zrcalnu komponentu, što omogućava rastavljanje integrala na dva podintegrala.

$$L_o(\boldsymbol{p}, \boldsymbol{\omega_o}) = \int_{\Omega} (f_d + f_s) L_i(\boldsymbol{p}, \boldsymbol{\omega_i})(\boldsymbol{\omega_i} \cdot \boldsymbol{n})\, d\boldsymbol{\omega_i}$$
$$= \int_{\Omega} f_d L_i(\boldsymbol{p}, \boldsymbol{\omega_i})(\boldsymbol{\omega_i} \cdot \boldsymbol{n})\, d\boldsymbol{\omega_i} + \int_{\Omega} f_s L_i(\boldsymbol{p}, \boldsymbol{\omega_i})(\boldsymbol{\omega_i} \cdot \boldsymbol{n})\, d\boldsymbol{\omega_i} \tag{13}$$

3.1. Metoda Monte Carlo

Računanje složenih integrala, poput onih u prethodnim poglavljima, nije moguće riješiti analitički, odnosno ne postoji jednostavna matematička formula koja bi dala točan rezultat. Umjesto toga, koristi se Monte Carlo metoda, koja predstavlja statistički pristup za aproksimaciju integrala pomoću nasumičnog uzorkovanja. Jedna od prednosti ove metode je to što koristi fiksni broj uzoraka, što znači da se algoritam uvijek izvršava u konstantnom

vremenu, bez obzira na složenost funkcije koju integriramo. Veći broj uzoraka vodi većoj točnosti rezultata, ali i većem vremenu izvođenja. Osnovna formula Monte Carlo integracije je:

$$\int f(x)\,dx \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{pdf(x_i)} \tag{14}$$

N označava broj uzoraka, a pdf je funkcija gustoće vjerojatnosti (engl. *probability density function*, PDF). Dijeljenje funkcije s PDF-om osigurava da doprinos svakog uzorka pravilno održava njegovu vjerojatnost pojave u uzorkovanju, čime algoritam čini nepristranim.

3.2. Difuzna komponenta

Budući da je difuzna komponenta konstanta, integral se može preoblikovati tako da se konstanta izvuče izvan integrala.

$$L_o(\boldsymbol{p}, \boldsymbol{\omega_o}) = \frac{c}{\pi}\int_{\Omega} L_i(\boldsymbol{p}, \boldsymbol{\omega_i})(\boldsymbol{\omega_i}\cdot\boldsymbol{n})\,d\boldsymbol{\omega_i} \tag{15}$$

Integral koji je ostao je formula za obasjanost, pa taj izraz možemo zamijeniti njenom oznakom.

$$L_o(\boldsymbol{p}, \boldsymbol{\omega_o}) = \frac{c}{\pi}E(\boldsymbol{p}) \tag{16}$$

Izračun obasjanosti provodi se konvolucijom kubne teksture, odnosno potrebno je konvoluirati svih 6 njezinih strana, što je računalno zahtjevna operacija. Dobra aproksimacija može se postići pomoću sfernih harmonika. Sferni harmonici omogućuju kompaktnu pohranu i efikasniji izračun.

3.3. Sferni harmonici

Sferni harmonici (engl. *spherical harmonics*, SH) su skup ortonormiranih funkcija definiranih na površini sfere, koje se koriste za aproksimaciju funkcija što ovise o smjeru. Mogu se zamišljati kao proširenje Fourierove transformacije s jedne dimenzije (kružnica) na dvije dimenzije (sfera). Kompleksni sferni harmonici dani su jednadžbom ispod:

$$Y_l^m(\theta, \phi) = K_l^m e^{im\theta} P_l^{|m|}\,cos\,\theta \tag{17}$$

Formula predstavlja kompleksni oblik sfernih harmonika, gdje je l red harmonike, a m njezin redoslijed unutar tog reda. Kut θ označava zenitni kut (kut od sjevernog pola), dok ϕ označava azimutni kut (rotaciju oko vertikalne osi). Izraz $P_l^{|m|}\ cos\ \theta$ označava pridružene Legendreove polinome koji određuju ovisnost funkcije o zenitnom kutu, dok izraz $e^{im\theta}$ određuje ovisnost o azimutnom kutu. Izraz K_l^m predstavlja normalizirajući faktor koji osigurava da su sferni harmonici ortonormirani. Na ovaj način, sferni harmonici omogućuju dekompoziciju bilo koje funkcije definirane na sferi u skup baznih funkcija, slično kao što Fourierova analiza dekomponira funkcije na pravcu.

3.4. Implementacija difuzne komponente

Algoritam koji se koristi za izračunavanje koeficijenata sfernih harmonika temelji se na pristupima korištenim u Unreal Engine-u te optimizacijama poznatim pod nazivom „Stupid Spherical Harmonics Tricks“ [4]. Implementacija se provodi korištenjem računskog sjenčara (engl. *compute shader*), koji omogućuje paralelno izvođenje dretvi. Svaka dretva neovisno uzrokuje sferu i doprinosi ukupnoj akumulaciji SH koeficijenata.

Prvi korak u algoritmu je generiranje smjera iz kojeg će se uzorkovati kubna tekstura. To se izvodi pomoću Monte Carlo uzorkovanja prema uniformnoj distribuciji po površini jedinične sfere. Konstrukcija vektora za uzorkovanje provodi se sljedećom funkcijom:

```
float3 UniformSampleSphere(float2 Xi)
{
    float phi = 2 * PI * Xi.x;
    float cosTheta = 1 - 2 * Xi.y;
    float sinTheta = sqrt(1 - cosTheta * cosTheta);

    float3 H;
    H.x = cos(phi) * sinTheta;
    H.y = sin(phi) * sinTheta;
    H.z = cosTheta;

    return H;
}
```

Ulaz u funkciju je slučajni 2D vektor Xi unutar jediničnog kvadrata $[0, 1]^2$, koji se koristi za konstrukciju sfernog koordinatnog sustava. PDF za uniformnu distribucije na jediničnoj sferi iznosi:

$$pdf_{sphere} = \frac{1}{4\pi} \tag{18}$$

Svaka dretva unutar grupe izračunava vlastiti smjer, učitava boju iz okolne teksture i računa doprinos za SH koeficijente:

```
#define THREAD_X 8
#define THREAD_Y 8
#define THREAD_GROUP (THREAD_X * THREAD_Y)

[numthreads(THREAD_X, THREAD_Y, 1)]
void ComputeIrradianceCS(uint3 id : SV_DispatchThreadId)
{
    const uint index = THREAD_X * id.y + id.x;
    const float2 uv = float2(id.xy + 0.5) / float2(THREAD_X, THREAD_Y);

    float3 H = UniformSampleSphere(uv);
    float pdf = rcp(4 * PI);
    float weight = rcp(pdf * THREAD_GROUP);

    float3 color = _SourceCubemap.SampleLevel(sampler_SourceCubemap, H, MipLevel).rgb;

    SphericalHarmonicsL2 basis = SHBasisFunction3(H);
    IrradianceSHShared[index] = MulSH3(basis, color * weight);

    // ...
}
```

Za svaki uzorak, izračunava se 9 vrijednosti koje predstavljaju SH drugog reda za dani smjer. To je implementirano u funkciji SHBasisFunction3, koja implementira standardni skup SH baza za $l = 0, 1, 2$:

```
SphericalHarmonicsL2 SHBasisFunction3(float3 direction)
{
    SphericalHarmonicsL2 result;
    result.a0[0] = 0.282095f;
    result.a0[1] = -0.488603f * direction.y;
    result.a0[2] =  0.488603f * direction.z;
    result.a0[3] = -0.488603f * direction.x;
    result.a1[0] =  1.092548f * direction.x * direction.y;
    result.a1[1] = -1.092548f * direction.y * direction.z;
    result.a1[2] =  0.315392f * (3 * direction.z * direction.z - 1.0f);
    result.a1[3] = -1.092548f * direction.x * direction.z;
    result.a2    =  0.546274f * (direction.x * direction.x - direction.y * direction.y);
    return result;
}
```

Regulirajući koeficijenti se množe s RGB komponentama, uzimajući u obzir težinu svakog uzorka. Svaka dretva upisuje svoj doprinos u zajednički spremnik koji je u dijeljenoj memoriji, odnosno sve dretve mogu upisivati ili čitati iz zajedničkog spremnika. Zatim se koristit paralelna redukcija kako bi se doprinosi svih dretvi u grupi zbrojili. Ovo se izvodi u više iteracija, gdje se u svakom koraku broj aktivnih dretvi prepolovljuje dok ne ostane jedna konačna dretva koja sadrži sumirani rezultat.

```
[unroll]
for (uint i = THREAD_GROUP * 0.5; i > 1; i  = i >> 1)
{
    GroupMemoryBarrierWithGroupSync();

    if (index < i)
    {
        IrradianceSHShared[index] =
        AddSH3(IrradianceSHShared[index], IrradianceSHShared[index + i]);
    }

}
GroupMemoryBarrierWithGroupSync();
```

```
14.
15. if (index < 1)
16. {
17.     SphericalHarmonicsL2RGB irradiance =
18.     AddSH3(IrradianceSHShared[index], IrradianceSHShared[index + 1]);
19.     // …
20. }
```

Nakon redukcije, posljednja dretva normalizira rezultate pomoću konstanti koje proizlaze iz integracije sfernih harmonika po cijeloj sferi. Te konstante se koriste kako bi se konačni SH koeficijenti mogli direktno koristiti za rekonstrukciju osvjetljenja u daljnjoj obradi. Koeficijenti se pohranjuju u izlazni spremnik, po 9 koeficijenata za svaku RGB komponentu:

```
 1. // Stupid Spherical Harmonics (SH) Tricks
 2. const float sqrtPI = sqrt(PI);
 3. const float C0 = 1.0 * rcp(2.0 * sqrtPI);
 4. const float C1 = sqrt(3.0) * rcp(3.0 * sqrtPI);
 5. const float C2 = sqrt(15.0) * rcp(8.0 * sqrtPI);
 6. const float C3 = sqrt(5.0) * rcp(16.0 * sqrtPI);
 7. const float C4 = 0.5 * C2;
 8.
 9. // cAr – primjer za crevnu komponentu
10. OutIrradianceMapSH[0].x = -C1 * irradiance.r.a0[3];
11. OutIrradianceMapSH[0].y = -C1 * irradiance.r.a0[1];
12. OutIrradianceMapSH[0].z =  C1 * irradiance.r.a0[2];
13. OutIrradianceMapSH[0].w =  C0 * irradiance.r.a0[0] - C3 * irradiance.r.a1[2];
14.
15. // cAg – isto kao za crvenu
16. // cAb – isto kao za crvenu
17.
18.
19. // cBr – primjer za crevnu komponentu
20. OutIrradianceMapSH[3].x =  C2 * irradiance.r.a1[0];
21. OutIrradianceMapSH[3].y = -C2 * irradiance.r.a1[1];
22. OutIrradianceMapSH[3].z = 3.0 * C3 * irradiance.r.a1[2];
23. OutIrradianceMapSH[3].w = -C2 * irradiance.r.a1[3];
24.
25. // cBg – isto kao za crvenu
26. // cBb – isto kao za crvenu
27.
28. // cC
29. OutIrradianceMapSH[6].x = C4 * irradiance.r.a2;
30. OutIrradianceMapSH[6].y = C4 * irradiance.g.a2;
31. OutIrradianceMapSH[6].z = C4 * irradiance.b.a2;
32. OutIrradianceMapSH[6].w = 1.0;
```

Algoritam koristi 64 uzorka, odnosno oko 10 uzorka po strani kubne teksture. Kako bi se izbjeglo uzorkovanje koje sadrži šum ili visoke frekvencije, bira se niža mip razina prema sljedećem izrazu:

$$mip = max\,(0, log_2(size) - log_2(16)) \tag{19}$$

Rekonstrukcija obasjanosti pomoću dobivenih SH koeficijenata i normale površine izvodi se sljedećom funkcijom:

```
1.  float3 Irradiance(float3 normal)
2. {
```

```
    float4 A = float4(normal, 1.0);
    float3 x1, x2, x3;

    x1.r = dot(_SkyIrradiance[0], A);
    x1.g = dot(_SkyIrradiance[1], A);
    x1.b = dot(_SkyIrradiance[2], A);

    float4 B = A.xyzz * A.yzzx;
    x2.r = dot(_SkyIrradiance[3], B);
    x2.g = dot(_SkyIrradiance[4], B);
    x2.b = dot(_SkyIrradiance[5], B);

    float vC = A.x * A.x - A.y * A.y;
    x3 = _SkyIrradiance[6].xyz * vC;

    return x1 + x2 + x3;
}
```

3.5. Zrcalna komponenta

Zrcalna komponenta osvjetljenja znatno je složenija jer uključuje integraciju BRDF-a unutar integrala.

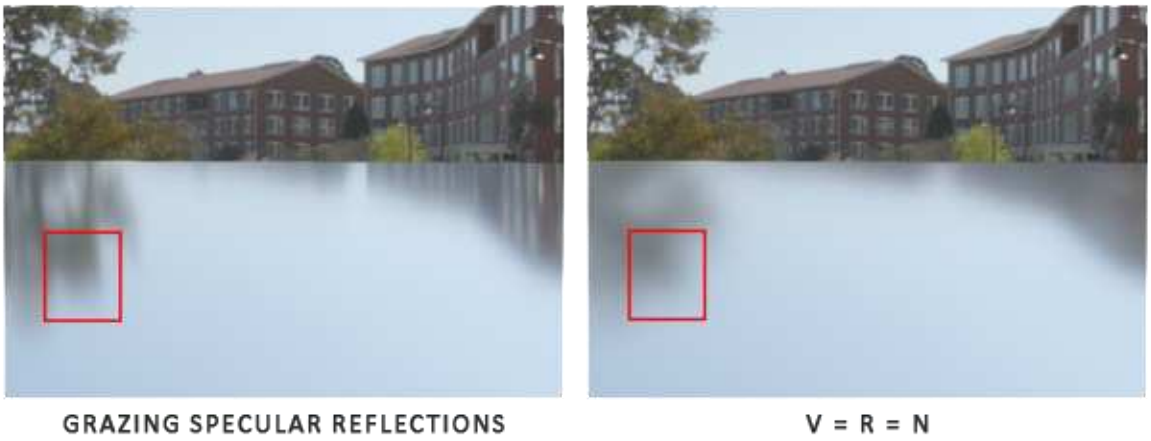
$$L_o(\boldsymbol{p}, \boldsymbol{\omega_o}) = \int_{\Omega} D(\alpha, \boldsymbol{n}, \boldsymbol{h}) V(\alpha, \boldsymbol{n}, \boldsymbol{\omega_o}, \boldsymbol{\omega_i}) F(f_0, f_{90}, \boldsymbol{\omega_o}, \boldsymbol{h}) L_i(\boldsymbol{p}, \boldsymbol{\omega_i}) (\boldsymbol{\omega_i} \cdot \boldsymbol{n}) \, d\boldsymbol{\omega_i} \quad (20)$$

Uzorkovanje ovog integrala Monte Carlo metodom i dalje predstavlja izazovan zadatak. Karis [3] predlaže aproksimaciju razdvojenih suma (engl. *split-sum approximation*), kojom se integral dijeli na dvije komponente:

$$L_o(\boldsymbol{p}, \boldsymbol{\omega_o}) \approx \int_{\Omega} L_i(\boldsymbol{p}, \boldsymbol{\omega_i}) \, d\boldsymbol{\omega_i} \cdot \int_{\Omega} f_s(\boldsymbol{\omega_i} \cdot \boldsymbol{n}) \, d\boldsymbol{\omega_i} \quad (21)$$

Prva komponenta zove se filtrirana okolišna tekstura (engl. *pre-filtered environment map*) i ona je odgovorna za zrcalne refleksije koje će se vidjeti na površini. Glađe površine rezultiraju izraženijim zrcalnim refleksijama, dok grublje površine proizvode mutnije odraze. Ovo sve opisuje parametar hrapavosti materijala koji je realan broj u rasponu [0, 1]. Uvelike možemo smanjiti broj uzoraka tako da odredimo konačan broj uzoraka za koje se integral računa zasebno. Svaki uzorak se tada sprema u jednu mip razinu unutar kubne teksture, čime se omogućuje jednostavno uzorkovanje teksture te automatska interpolacija između dva granična uzorka. Zbog memorijskog prostora i težine računanja, ovo je najzahtjevnija operacija ovog rada. Karis [3] dodatno uvodi pojednostavljenje pretpostavljajući da je kut između normale i smjera pogleda jednak nuli, čime se dodatno

ubrzava evaluacija, ali uz određenu pogrešku. Utjecaj ove aproksimacije vidljiv je na slici ispod:



Slika 3.2 Razlika između fizički korektnih (lijevo) i aproksimiranih (desno) refleksija [1]

Druga komponenta je ostatak početnog integrala, odnosno BRDF i kut upada. Može se primijetiti kako ovaj integral ne ovisi sjajnosti okoliša što znači da je ovaj integral potrebno izračunati jedanput. Integral možemo dalje pojednostaviti tako da izvučemo parametar f_0 izvan integrala.

$$F_{schlick}(f_0, f_{90}, \boldsymbol{v}, \boldsymbol{h}) = f_0 + (f_{90} - f_0)(1 - \boldsymbol{v} \cdot \boldsymbol{h})^5 \tag{22}$$

$$f_{s'} = D(\alpha, \boldsymbol{n}, \boldsymbol{h}) V(\alpha, \boldsymbol{n}, \boldsymbol{\omega_o}, \boldsymbol{\omega_i}) \tag{23}$$

$$f_0 \int_{\Omega} f_{s'}(1 - (1 - \boldsymbol{v} \cdot \boldsymbol{h})^5)(\boldsymbol{\omega_i} \cdot \boldsymbol{n})\, d\boldsymbol{\omega_i} + \int_{\Omega} f_{s'}(\boldsymbol{\omega_i} \cdot \boldsymbol{n})(1 - \boldsymbol{v} \cdot \boldsymbol{h})^5\, d\boldsymbol{\omega_i} \tag{24}$$

3.6. Implementacija filtrirane okolišne teksture

Ovaj algoritam je implementiran je pomoću računskog sjenčara. Počinje od najjednostavnijeg slučaja, kada je hrapavost površine jednaka nuli. Tada se površina ponaša kao ogledalo, što znači da se može jednostavno uzorkovati originalna tekstura. To je prikazano sljedećim isječkom:

```
1. [numthreads(THREAD_X, THREAD_Y, 1)]
2. void FilterEnvionemntCS(uint3 id : SV_DispatchThreadId)
3. {
4.     uint width, height, slices;
5.     _ReflectionMap.GetDimensions(width, height, slices);
6.
7.     float2 uv = (id.xy + 0.5) / float2(width, height);
8.     uv = uv * 2.0 - 1.0;
9.
```

```
10.     float3 N = normalize(GetCubemapVector(uv, id.z));
11.
12.     if (_MipIndex == 0)
13.     {
14.         _ReflectionMap[id] = _SourceCubemap.SampleLevel(sampler_SourceCubemap, N, 0).rgb;
15.         return;
16.     }
17.
18.     // …
19. }
```

Za svaku dretvu, komponente x i y predstavljaju koordinate na 2D teksturi, dok komponenta z označava lice (engl. face) koje se filtrira.

Složeniji slučaj je uzorkovanje teksture, koje se provodi pomoću Monte Carlo metoda. Neke Monte Carlo metode su pristrane, što znači da generirani uzorci nisu potpuno slučajni, već su usmjereni prema određenoj vrijednosti ili smjeru. Takve metode imaju bržu stopu konvergencije, što znači da se mogu približiti točnom rješenju brže nego nepristrani algoritmi, ali zbog svoje pristranosti vjerojatno nikad neće konvergirati na točno rješenje.

Kvazislučajni nizovi (engl. *low-discrepancy sequences*) generiraju uzorke koji su nasumični, ali bolje raspoređeni i ravnomjernije distribuirani nego kod klasičnih slučajnih nizova. Primjena Monte Carlo metoda nad kvazislučajnim naziva se Kvazi-Monte Carlo metoda. Ovakve metode imaju bržu konvergenciju, što ih čini pogodnima za računski zahtjevne aplikacije.

Težinsko uzorkovanje (engl. *importance sampling*) metoda je u kojoj se uzorci ne biraju ravnomjerno, već se daje veća vjerojatnost onim uzorcima koji više doprinose ukupnom rezultatu integrala.

Kombinacija kvazislučajnih nizova i težinskog uzorkovanja rezultira algoritmom s visokom brzinom konvergencije. Budući da takav algoritam brže dolazi do približnog rješenja, potreban je manji broj uzoraka za postizanje zadovoljavajuće aproksimacije.

U ovom radu implementirana je kvazislučajna Hammersley sekvenca, koja kao parametre prima indeks trenutnog uzorka i ukupan broj uzoraka:

```
1. float2 Hammersley(uint i, uint N)
2. {
3.     float x = frac((float)i / N);
4.     float y = float(reversebits(i)) * 2.3283064365386963e-10;
5.     return float2(x, y);
6. }
```

Također, implementirano je težinsko uzorkovanje prema GGX distribuciji. Funkcija generira vektor u tangentnom prostoru koji je pristran prema smjeru refleksije, ovisno o hrapavosti površine:

```
float3 ImportanceSampleGGX(float2 Xi, float a2)
{
    float phi = 2 * PI * Xi.x;
    float cosTheta = sqrt((1.0 - Xi.y) / (1.0 + (a2 - 1.0) * Xi.y));
    float sinTheta = sqrt(1.0 - cosTheta * cosTheta);

    float3 H;
    H.x = cos(phi) * sinTheta;
    H.y = sin(phi) * sinTheta;
    H.z = cosTheta;

    return H;
}
```

Algoritam koristi optimizacije opisane u [5], koje su izvedene iz originalnog algoritma iz Unreal Enginea [3]. Najznačajnija promjena je uzorkovanje po mip razinama pomoću omjera prostornog kuta koji zauzima uzorak i prostornog kuta koji zauzima pojedini teksel.

```
float roughness = float(_MipIndex) * rcp(_MipCount - 1);
float solidAngleByTexel = 4.0 * PI * rcp(6 * roughness * roughness);
float3x3 tangentToWorld = GetOrthonormalBasis(N);
uint nbSamples = 128;

float3 color = 0.0;
float weight = 0.0;

[loop]
for (uint i = 0; i < nbSamples; i++)
{
    float2 Xi = Hammersley(i, nbSamples);
    float3 H = ImportanceSampleGGX(Xi, pow(roughness, 4));
    float3 L = 2.0 * H.z * H - float3(0, 0, 1);

    float NoL = L.z;
    float NoH = H.z;
    if (NoL > 0.0)
    {
        L = mul(L, tangentToWorld);

        float pdf = DistributionGGX(pow(roughness, 4), NoH) * 0.25;
        float solidAngleBySample = 1.0 * rcp(nbSamples * pdf);
        float mip = 1.0 + log2(solidAngleBySample / solidAngleByTexel);

        color += _SourceCubemap.SampleLevel(sampler_SourceCubemap, L, mip).rgb * NoL;
        weight += NoL;
    }
}

_ReflectionMap[id] = color / weight;
```

3.7. Implementacija BRDF integracije

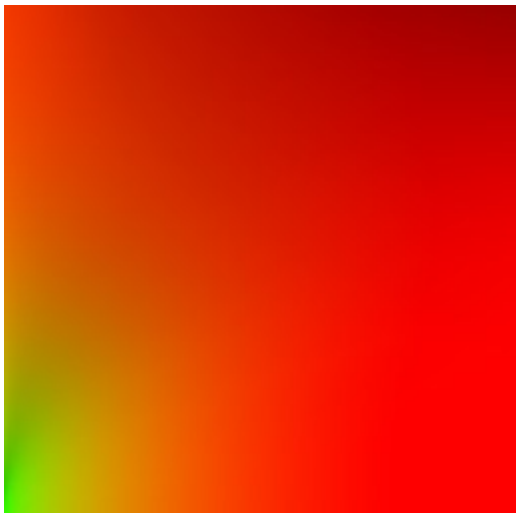
Kao što je prethodno spomenuto, ovu komponentu potrebno je integrirati samo jednom jer ne ovisi o kubnoj teksturi koju filtriramo. Međutim, ona ovisi o kutu između normale površine i smjera pogleda, kao i o hrapavosti površine. Zbog toga se rezultat integracije sprema u tablicu preslikavanja (engl. *look-up table*, LUT) koja je pohranjena kao 2D tekstura (Slika 3.3). U toj teksturi, x-koordinata označava kut između normale i vektora smjera pogleda, dok y-koordinata predstavlja hrapavost površine.

Funkcija gustoće vjerojatnosti težinskog uzorkovanja za BRDF koji se koristi, definirana je formulom:

$$pdf_{GGX} = \frac{D(\alpha, \boldsymbol{n}, \boldsymbol{h})(\boldsymbol{n} \cdot \boldsymbol{h})}{4(\boldsymbol{v} \cdot \boldsymbol{h})} \tag{25}$$

Uvrštavanjem ovog PDF-a u integralnu jednadžbu BRDF funkcije moguće je skratiti izračun NDF-a. Integracija BRDF-a izvodi se na sličan način kao i računanje filtrirane okolišne teksture, koristeći računalni sjenčar.

```
[numthreads(THREAD_X, THREAD_Y, 1)]
void IntegrateBRDFCS(uint3 id : SV_DispatchThreadId)
{
    uint width, height;
    _BRDFLut.GetDimensions(width, height);

    float2 uv = (id.xy + 0.5) / float2(width, height);

    float NoV = uv.x;
    float roughness = uv.y;

    float3 V;
    V.x = sqrt(1.0 - NoV * NoV);
    V.y = 0.0;
    V.z = NoV;

    float A = 0.0;
    float B = 0.0;

    uint nbSamples = 128;

    [loop]
    for (uint i = 0; i < nbSamples; i++)
    {
        float2 Xi = Hammersley(i, nbSamples);
        float3 H = ImportanceSampleGGX(Xi, pow(roughness, 4));
        float3 L = 2 * dot(V, H) * H - V;

        float NoL = saturate(L.z);
        float NoH = saturate(H.z);
        float VoH = saturate(dot(V, H));

        if (NoL > 0.0)
        {
            float Vis = VisibilitySmithGGXCorrelatedFast(pow(roughness, 4), NoV, NoL);
            Vis = Vis * VoH * NoL * 4 * rcp(NoH);
            float F = pow(1.0 - VoH, 5.0);

            A += (1.0 - F) * Vis;
            B += F * Vis;
        }
    }

    _BRDFLut[id.xy] = float2(A, B) / nbSamples;
}
```



Slika 3.3 BRDF LUT

4. Programsko rješenje

4.1. Unity API

Programsko rješenje bit će implementirano u Unity verziji 6000.0.32f. Unity 6 donosi značajne promjene u sustavu renderiranja, što znači da implementirani kod neće biti kompatibilan s prethodnim verzijama Unityja. U sklopu rješenja koristit će se prilagođeni cjevovod renderiranja.

Za početak, potrebno je definirati klasu koja nasljeđuje RenderPipelineAsset. Ova klasa mora implementirati apstraktnu metodu CreatePipeline, koja stvara instance vlastitog cjevovoda renderiranja.

```
1. [CreateAssetMenu(menuName = "Rendering/Render Pipeline")]
2. public class FERRenderPipelineAsset : RenderPipelineAsset<FERRenderPipeline>
3. {
4.
5.     // ...
6.
7.     protected override RenderPipeline CreatePipeline()
8.     {
9.         return new FERRenderPipeline(this);
10.    }
11.
12.    // ...
13.
14. }
```

Sljedeći korak je implementacija klase koja nasljeđuje RenderPipeline. Ova klasa definira renderiranje putem metode Render.

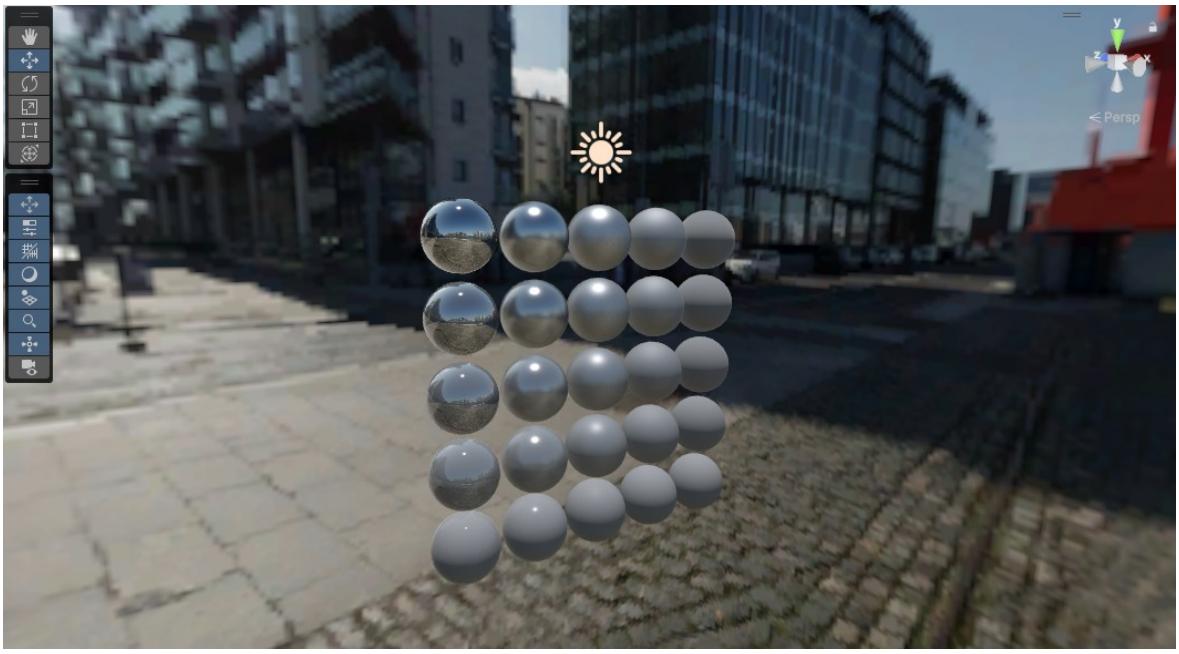
```
1. public class FERRenderPipeline : RenderPipeline
2. {
3.
4.    // ...
5.
6.    protected override void Render(ScriptableRenderContext context, List<Camera> cameras)
7.    {
8.        // logika za renderiranje svih kamera u sceni
9.    }
10.
11.   // ...
12.
13. }
```

Ukratko, RenderPipelineAsset je objekt koji se može kreirati unutar Unity editora i služi kao početna točka za definiranje prilagođenog cjevovoda. Ako je potrebno definirati ulazne parametre ili konfiguracije, to se radi unutar ove klase. Kada Unity koristi prilagođeni cjevovod on instancira objekt RenderPipeline pomoću metode CreatePipeline. Unutar ove instance se odvija kompletna logika renderiranja. Metoda Render prima dva parametra,

ScriptableRenderContext koji je apstraktni kontekst najviše razine za komunikaciju s grafičkom karticom putem API-ja za renderiranje i listu svih kamera koji zahtijevaju renderiranje u trenutnom okviru.

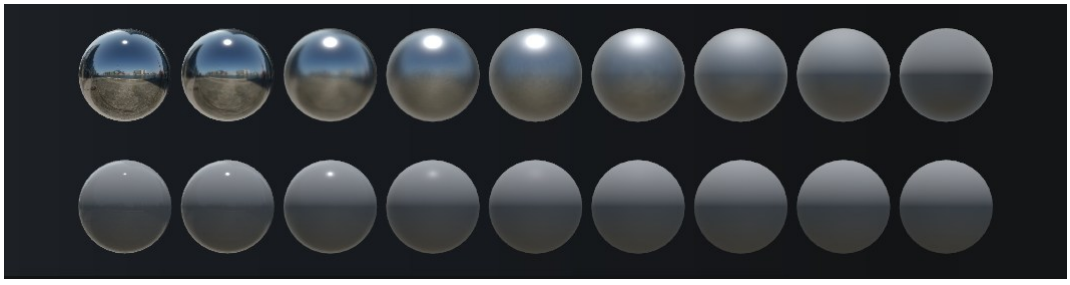
4.2. Rezultati

Na slici ispod (Slika 4.1) prikazan je raspored od 5x5 sfera koje demonstriraju izgled materijala pod različitim parametrima BRDF-a. Scena uključuje jedno direkcijsko svjetlo i svjetlo temeljeno na okolišnoj slici. X-os prikazuje promjenu hrapavosti, dok y-os prikazuje promjenu metalnosti. Parametri koji se mogu mijenjati uključuju boju površine, skalarne vrijednosti hrapavosti te metalnosti površine.



Slika 4.1 Programsko rješenje prikaza materijala

Direktna usporedba potpunog metala i dielektrika po razinama hrapavosti dana je slikom ispod (Slika 4.2).



Slika 4.2 Prikaz potpunog metala (gornji red) i potpunog dielektrika (donji red) po hrapavostima

Računanje zrcalne komponente u ovom projektu predstavlja najzahtjevniji proces, zbog čega se preporučuje njeno unaprijedno izvođenje. Za razliku od BRDF LUTa, zrcalna komponenta ima ključnu ulogu jer je željeni cilj hvatanje okoliša u stvarnom vremenu. U svrhu evaluacije performansi, izvedena su mjerenja vremena izvođenja za računanje zrcalne komponente, koristeći teksturu rezolucije 256×256.

Broj uzoraka	NVIDIA GeForce RTX 3060	NVIDIA GeForce GTX 1650
32	0.1 ms	0.2 ms
64	1.9 ms	5.9 ms
128	4.7 ms	13.0 ms
256	10.2 ms	29.4 ms
512	21.0 ms	60.3 ms

Tablica 4.1 Mjerenje performansi filtriranja okolišne mape

Kako bi se izmjerila efikasnost implementiranog sjenčara, provedena su mjerenja na scenama s različitim brojem trokuta. Za mjerenje su korištena dva sjenčara: jednostavan jednobojni sjenčar (koji dodjeljuje konstantnu boju fragmenta) i implementirani sjenčar razvijen u sklopu ovog projekta. Rezultati renderiranja prikazani su u tablici ispod.

Broj trokuta	NVIDIA GeForce RTX 3060		NVIDIA GeForce GTX 1650	
	Jednobojni sjenčar	Implementirani sjenčar	Jednobojni sjenčar	Implementirani sjenčar
~1 mil	1.42 ms	1.42 ms	2.38 ms	2.42 ms
~4 mil	1.42ms	1.43 ms	2.55 ms	2.80 ms
~8 mil	1.98 ms	2.33 ms	3.91 ms	4.60 ms
~16 mil	3.54 ms	4.22 ms	8.03 ms	9.04 ms

Tablica 4.2 Mjerenje performansi renderiranja scene s jednostavnim i implementiranim sjenčarom

Analizom izvedenih mjerenja može se zaključiti da se vrijeme računanja zrcalne komponente proporcionalno povećava s brojem uzoraka. Kvaliteta prikaza također raste s povećanjem broja uzoraka, pri čemu se najbolji vizualni rezultati postižu sa 128 i 256 uzoraka, dok je razlika između 256 i 512 uzoraka zanemariva. Izazov predstavljaju hrapavije površine na kojima značajno manji broj uzoraka prolazi if-test, što rezultira smanjenom kvalitetom prikaza u odnosu na glađe površine. Moguća optimizacija uključuje selektivno testiranje uzoraka i slanje samo onih koji prolaze if-test, kako je opisano u [5]. Iako implementirani sjenčar zahtijeva nešto veće vrijeme renderiranja u usporedbi s jednobojnim

sjenčarom, razlika nije značajna i nalazi se u prihvatljivim granicama. Za daljnja vizualna poboljšanja predlaže se dodavanje parametara poput emisije i ambijentne okluzije, podrške za teksture, te potpuna implementacija BSDF-a.

Programsko rješenje dostupno je kao Unity paket na GitHub repozitoriju: https://github.com/DorijanStyle/com.dorijanstyle.render-pipeline.git.

Zaključak

U ovom radu obrađena je fizikalno temeljena izrada prikaza u računalnoj grafici. Cilj rada je bio steći potrebno znanje i implementirati algoritme koji se u praksi koriste u brojnim aplikacijama za renderiranje. Sve kreće od jednadžbe renderiranja, koja na najvišoj razini apstrakcije definira vizualni rezultat interakcije svjetlosti s površinom objekta u sceni. Sljedeći korak je BRDF funkcija, koja definira na koji način se zraka svjetlosti reflektira od površine prema promatraču. Fizikalno temeljeni BRDF modeli temelje se na teoriji mikroploha, koja opisuje svojstva površine materijala.

Refleksija svjetlosti modelira se kao linearna kombinacija difuzne i zrcalnu komponente. U radu su opisani i implementirani Lambertov BRDF za difuznu, te Cook-Torranceov BRDF za zrcalnu komponentu. U stvarnosti se svjetlosne zrake odbijaju od više objekata u sceni, pri čemu svi oni doprinose ukupnoj izlaznoj sjajnosti. Jedna od metoda aproksimacije takvog osvjetljenja je korištenje slike okoline kao velikog izvora svjetlosti, pri čemu svjetlost dolazi iz svih smjerova, što zahtijeva računanje integrala.

Računanje integrala je računski zahtjevno, zato se oslanjamo na optimizacijske metode i aproksimacija koje dolaze na cijenu kvalitete prikaza i odstupanja od fizičke točnosti. Temeljena izrada prikaza je i dalje aktivno područje istraživanja u kojem se nastoji pronaći što bolje ravnoteže između točnosti i računalne učinkovitosti.

Literatura

[1] Joey de Vries, *PBR - Theory,* LearnOpenGL. Poveznica: https://learnopengl.com/PBR/Theory

[2] Romain Guy, Mathias Agopian, *Physically Based Rendering in Filament*. Poveznica: https://google.github.io/filament/Filament.md.html

[3] Brian Karis, *Real Shading in Unreal Engine 4*. Poveznica: https://cdn2.unrealengine.com/Resources/files/2013SiggraphPresentationsNotes-26915738.pdf

[4] Peter-Pike Sloan, *Stupid Spherical Harmonics (SH) Tricks*. Poveznica: https://www.ppsloan.org/publications/StupidSH36.pdf

[5] Padraic Hennessy, *Implementation Notes: Runtime Environment Map Filtering for Image Based Lighting*. Poveznica: https://placeholderart.wordpress.com/

Sažetak

Fizikalno temeljena izrada prikaza

Ovaj rad obrađuje fizikalno temeljenu izradu prikaza u računalnoj grafici s naglaskom na teorijsku osnovu i praktičnu implementaciju. Objašnjena je jednadžba renderiranja, funkcija dvosmjerne distribucije i refleksije i osvjetljenje temeljeno na slici. Objašnjen je i implementiran Lambertov i Cook-Torranceov model osvjetljenja. Zbog cilja razvoja aplikacije u stvarnom vremenu korištene su brojne optimizacije koje ubrzavaju renderiranje. Implementacija je provedena u Unityu. Fizikalno temeljeno renderiranje ostaje aktivno područje istraživanja s ciljem bolje ravnoteže između realizma i računalne učinkovitosti.

Ključne riječi: fizikalno temeljena izrada prikaza, računalna grafika, funkcija dvosmjerne distribucije refleksije, Lambert, Cook-Torrance, osvjetljenje temeljeno na slici, Unity, sjenčari

Summary

Physically based rendering

This paper explores the physically based rendering in computer graphics, with an emphasis on both theoretical foundations and practical implications. It explains the rendering equation, the bidirectional reflectance distribution function and image-based lighting. Both Lambertian and Cook-Torrance lighting models are described and implemented. Due to the goal of developing a real-time application, numerous optimizations were applied to accelerate rendering. The implementation was carried out in Unity. Physically based rendering remains an active research area aimed at achieving a better balance between realism and computational efficiency.

Keyword: physically based rendering, computer graphics, bidirectional reflectance distribution function, BRDF, Lambertian, Cook-Torrance, image-based lighting, IBL, Unity, shaders