

UNIVERSITATEA NAȚIONALĂ DE ȘTIINȚĂ ȘI TEHNOLOGIE
POLITEHNICA BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL DE CALCULATOARE



PROIECT DE DIPLOMĂ

Implementarea algoritmului Ray Tracing
folosind arhitectura DirectX Raytracing

Alex-Andrei Cioc

Coordonator științific:

Conf. Dr. Ing. Victor Asavei

BUCUREȘTI

2024

NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY
POLITEHNICA BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT



DIPLOMA PROJECT

Implementation of the Ray Tracing algorithm
using the DirectX Raytracing architecture

Alex-Andrei Cioc

Thesis advisor:

Conf. Dr. Ing. Victor Asavei

BUCHAREST

2024

CUPRINS

1	Introducere	1
1.1	Context	1
1.2	Problema	1
1.3	Obiective	3
1.4	Soluția propusă	4
1.5	Rezultatele obținute	5
1.6	Structura lucrării	5
2	Motivație de cercetare	7
3	Metode Existente	10
3.1	Rasterizare	10
3.2	Ray Tracing	11
3.3	Metode Monte Carlo	15
3.3.1	Integrare Monte Carlo	16
3.3.2	Eșantionare bazată pe importanță	17
3.3.3	Eșantionare Stratificată	18
3.3.4	Ecuatia transportului luminii	19
3.4	Radiosity	20
3.5	Path Tracing	22
3.5.1	Strategii de eșantionare a pixelilor	24
3.5.2	Strategii de alegere a funcției de distribuție a reflectantei	25
3.5.3	Strategii de eșantionare a direcțiilor de ieșire	32
3.5.4	Next Event Estimation	34
3.6	Accelerare Hardware	35

3.6.1	Vulkan	36
3.6.2	DirectX 12	36
4	Soluția Propusă	38
4.1	Tehnici de accelerare a convergenței	39
4.2	Suport pentru suprafete implicate și analitice	40
4.3	Sistemul de materiale	40
5	Detalii de implementare	45
5.1	Folosirea API-ului DXR	45
5.2	Meniul	46
5.3	Generator de numere aleatoare	48
5.4	Ray Generation shader	49
5.5	Closest Hit shader	51
5.5.1	Algoritmul Path Tracing	51
5.6	Miss shader	53
5.7	Evaluarea și eșantionarea BSDF-ului	53
6	Evaluare	54
6.1	Evaluarea performanțelor	54
6.1.1	Numărul de eșantioane per pixel	54
6.1.2	Adâncimea maximă a recursivității	54
6.1.3	Adâncimea minimă pentru ruleta rusească	55
6.1.4	Eșantionarea unei singure lumini	55
6.1.5	Strategia de eșantionare a direcției următoare	56
6.2	Evaluare calitativă	56
6.2.1	Comparatie cu Ray Tracing Whitted	56
6.2.2	Evaluarea strategiei de eșantionare a direcției următoare	59
6.2.3	Evaluarea modelului Disney principled BSDF	60
7	Concluzii	68

Anexe	72
Anexa A Figuri	73
Anexa B Extrase de cod	76

SINOPSIS

Algoritmul Ray Tracing este o tehnică de randare a imaginilor care simulează propagarea și comportamentul razelelor de lumină într-o scenă tridimensională. Acesta este adesea folosit în industria cinematografică pentru a obține imagini fotorealiste. Până de curând, natura computațională intensivă a acestui algoritm a limitat utilizarea sa în aplicații interactive, precum jocurile video. Totuși, cu avansul tehnologiei, utilizarea acestuia a devenit tot mai accesibilă și pentru aceste aplicații. Suport hardware pentru Ray Tracing în contextul consumatorilor a fost introdus de NVIDIA în 2018, prin intermediul arhitecturii Turing.¹ Tot în același an, Microsoft a anunțat lansarea DirectX Raytracing² (DXR), o extensie a API-ului DirectX 12 care permite programatorilor să folosească Ray Tracing în aplicațiile lor, utilizând hardware-ul compatibil.

Lucrarea de față își propune să studieze și să implementeze algoritmul Ray Tracing pe un sistem de calcul modern, folosind accelerarea hardware oferită de arhitectura DirectX Raytracing. Acest algoritm va fi folosit pentru randarea iluminării unor scene arbitrare, în timp real, oferind o reprezentare fotorealistă a acestora. În cadrul lucrării se va realiza o analiză a performanțelor implementării curente, atât a fidelității imaginilor generate, cât și a eficienței spațio-temporale a implementării. Se vor explora și posibilitățile de optimizare a algoritmului, precum și modul în care acestea pot fi folosite pentru a îmbunătăți performanțele sistemului.

ABSTRACT

The Ray Tracing algorithm is an image rendering technique that simulates the propagation and behavior of light rays in a three-dimensional scene. It is often used in the film industry to achieve photorealistic images. Until recently, the computationally intensive nature of this algorithm has limited its use in interactive applications, such as video games. However, with technological advances, its use has become increasingly accessible for these applications as well. Hardware support for Ray Tracing in the consumer context was introduced by NVIDIA in 2018, through the Turing¹ architecture. In the same year, Microsoft announced DirectX Raytracing² (DXR), an extension of the DirectX 12 API that allows programmers to use Ray Tracing in their applications, using compatible hardware.

This paper aims to study and implement the Ray Tracing algorithm on a modern computing system, using the hardware acceleration provided by the DirectX Raytracing architecture. This algorithm will be used for rendering the lighting of arbitrary scenes, in real-time, providing a photorealistic representation of them. The paper will perform an analysis of the current implementation's performance, both in terms of the fidelity of the generated images and the spatio-temporal efficiency of the implementation. It will also explore the possibilities for optimizing the algorithm, as well as how these can be used to improve the system's performance.

¹<https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf>. Accesat 10.06.2024.

²<https://devblogs.microsoft.com/directx/announcing-microsoft-directx-raytracing/>. Accesat 10.06.2024.

MULTUMIRI

Adresez mulțumiri coordonatorului meu de proiect, Conf. Dr. Ing. Victor Asavei, pentru îndrumarea și sprijinul acordat pe parcursul realizării acestei lucrări, dar și pentru inspirația și motivația oferită în cadrul cursurilor de Elemente de Grafică pe Calculator. De asemenea, mulțumesc familiei și prietenilor pentru susținere și încurajare.

1 INTRODUCERE

1.1 Context

Industria jocurilor video este una dintre cele mai mari și mai profitabile industrii de divertisment din lume. În 2021, piața jocurilor video era evaluată la aproximativ 202.64 miliarde de dolari și este estimat să se extindă la o rată anuală compusă de creștere de 10.2% în perioada 2022-2030.¹ Această industrie este alimentată de cererea pentru experiențe interactive și captivante, care să ofere o experiență de joc cât mai realistă și cât mai imersivă. Toate studio-urile de dezvoltare de jocuri video AAA (i.e., jocuri cu bugete mari și echipe de dezvoltare extinse) investesc resurse semnificative în dezvoltarea de tehnologii care să le permită să creeze jocuri cu grafică de înaltă calitate. Aceste tehnologii includ motoare grafice puternice, care să permită randarea unor scene complexe, cu iluminare realistă și efecte speciale impresionante. Multe studio-uri folosesc propriile motoare dezvoltate in-house (e.g., Frostbite de la EA, CryEngine de la Crytek, Anvil de la Ubisoft), dar există și motoare comerciale, precum Unreal Engine și Unity. Aceste motoare oferă un set de instrumente și funcționalități care permit dezvoltatorilor să creeze jocuri video de înaltă calitate, fără a fi nevoie să dezvolte de la zero toate componentele necesare. O componentă critică a acestor motoare este motorul grafic, care se ocupă de randarea scenei jocului, de la geometria obiectelor până la iluminare și efecte speciale. Astfel, programatorii, artiștii, animatorii și designerii de jocuri pot să se concentreze pe crearea conținutului jocului, fără a fi nevoie să se ocupe de detalii tehnice ale randării grafice.

1.2 Problema

Tehnica tradițională și cea mai răspândită de randare a imaginilor în jocurile video este rasterizarea. Această tehnică se bazează pe proiecția obiectelor 3D pe un plan bidimensional, folosind o serie de algoritmi și tehnici pentru a simula iluminarea și efectele speciale. Rasterizarea este o tehnică eficientă și rapidă, care permite randarea unui număr mare de obiecte în timp real, dar are și limitări. Una dintre cele mai mari limitări ale rasterizării este incapacitatea de a simula iluminarea globală, care este esențială pentru obținerea unor imagini fotorealiste. De asemenea, reflexiile și refractiile pot fi doar approximate, de exemplu prin tehnici de cubemapping sau screen-space reflections. Aceste tehnici sunt eficiente, dar nu oferă rezultate realistice, iar în multe cazuri pot fi observate artefacte vizuale care afectează calitatea imaginii.

În continuare se evidențiază aceste limitări (care nu sunt deloc exhaustive) ale rasterizării, prin

¹<https://www.grandviewresearch.com/industry-analysis/gaming-industry>. Accesat 10.06.2024.

comparație cu tehnica de Ray Tracing (așa numita *RTX* în jocurile sponsorizate de Nvidia). Comparând Figurile 1 și 2, se observă neajunsul reflexiilor în screen space. Atâtă timp cât obiectele reflectate se află în viewport, reflexiile sunt corecte și realiste. Însă, dacă obiectele ies din viewport, reflexiile se pierd, ceea ce duce la o imagine nerealistă. Un alt exemplu și mai elovent este ilustrat în Figura 4, unde imaginea randată cu Ray Tracing redă reflexii ale exploziei care nu este vizibilă decât parțial în cadru.



Figura 1: Screen space reflections în Minecraft²



Figura 2: Artefacte vizuale în screen space reflections.² Se poate observa cum reflexiile se pierd dacă obiectele reflectate ies din viewport

Am văzut cum tehnica de Ray Tracing poate oferi rezultate mult mai realiste decât rasterizarea, dar această tehnologie vine cu un cost. Algoritmul Ray Tracing este computațional intensiv,

²Shader folosit: <https://continuum.graphics/>. Accesat 10.06.2024.

³©Nvidia Corporation: <https://blogs.nvidia.com/blog/geforce-rtx-real-time-ray-tracing/>. Accesat 10.06.2024.

⁴©Nvidia Corporation: <https://www.youtube.com/watch?v=WoQr0k2IA9A>. Accesat 10.06.2024.

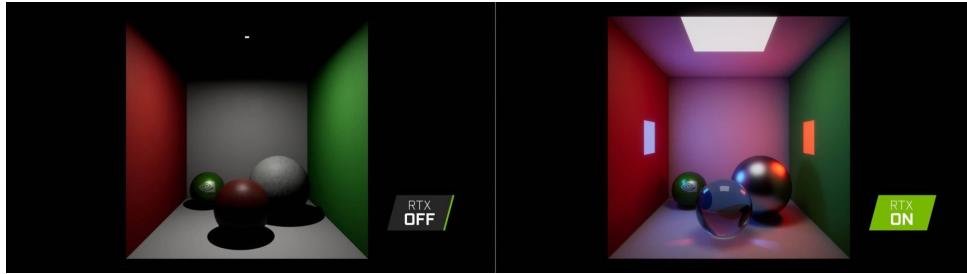


Figura 3: Iluminarea globală este absentă în imaginea randată cu rasterizare³

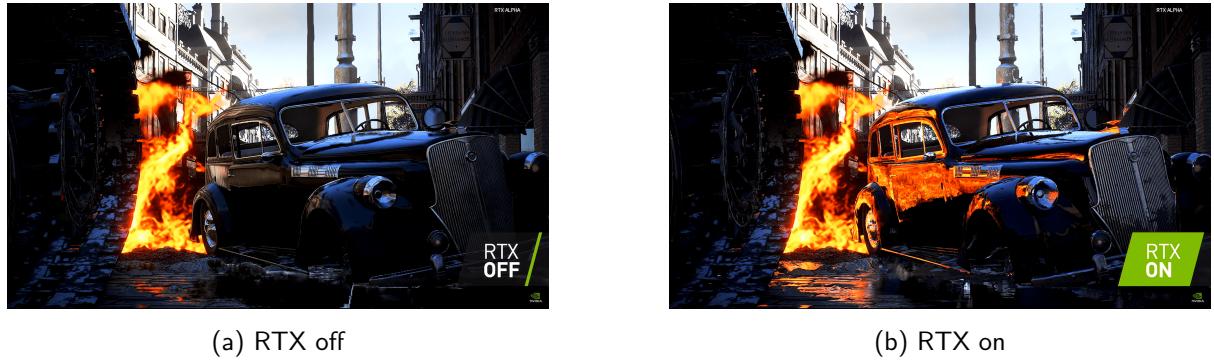


Figura 4: Comparație RTX on/off în Battlefield V⁴

deoarece necesită calcularea intersecțiilor mai multor raze de lumină pentru fiecare pixel cu obiectele din scenă și calcularea contribuției acestora la culoarea pixelului. Pentru a obține o imagine de calitate, este nevoie de un număr mare de raze de lumină și tehnici de denoising, ceea ce face ca algoritmul să fie greu de balansat între fidelitate și performanță.

1.3 Obiective

Scopul acestei lucrări este de a cerceta și implementa algoritmul Ray Tracing în context de timp real și a evalua performanțele acestuia, prin comparație cu implementări destinate producției cinematografice. Eforturile vor fi concentrate pe implementarea unui motor grafic simplu, cu câteva funcționalități de bază:

- Controle de cameră simple (mișcare, rotire)
- Randarea obiectelor definite ca mesh-uri triunghiulare
- Randarea obiectelor definite prin funcții implicate (e.g., sfere, planuri)
- Suport pentru materiale PBR (Physically Based Rendering)
- Meniu de configurare a mai multor parametri (e.g., pentru scenă, configurarea algoritmului etc.),

precum și a unor efecte de iluminare care să ofere o imagine fotorealistică a scenei:

- Iluminare globală
- Reflexii și refracții
- Umbre.

De asemenea, vom explora și implementa tehnici de optimizare a algoritmului propriu-zis, pe care le vom evalua în contextul de performanță și fidelitate obținute.

În final, dorim să obținem o implementare eficientă, care să întească un frametime de 33.3ms (30 de cadre pe secundă) la o rezoluție Full HD (1920x1080), pentru hardware entry-level cu suport pentru DirectX Raytracing (e.g., Nvidia GeForce RTX 2060).

1.4 Soluția propusă

Soluția propusă este un motor grafic simplu de utilizat. Din perspectiva utilizatorului, acesta are un meniu din care poate configura diversi parametri ai algoritmului de Ray Tracing, precum și ai scenei. Utilizatorul poate încărca scene predefinite și poate interacționa cu acestea folosind controalele de cameră.

La nivel de bază, motorul grafic conține două implementări din clasa algoritmilor de Ray Tracing. Prima este o versiune simplă a algoritmului original descris de Whitted (1979) [38], peste care se aplică modelul clasic de iluminare Phong (1975) [27]. Această implementare a fost adaptată din codul sursă suport oferit de Microsoft [25]. A doua implementare este scrisă de la zero și folosește algoritmul de tip Monte Carlo Path Tracing pentru a rezolva ecuația de iluminare globală propusă de Kajiya (1986) [17]. Fără a intra în prea multe detalii tehnice (vezi capitolul 3), această ecuație descrie cantitatea de lumină emisă dintr-un punct de pe o suprafață, de-a lungul unei direcții de vizualizare, dându-se o funcție de distribuție a luminii și un BRDF (Bidirectional Reflectance Distribution Function) pentru materialul de pe suprafață. Ecuația conține o integrală de suprafață (peste emisfera unitate), care integrează contribuțiile din toate direcțiile. Pentru eficiență, această integrală este eșantionată folosind tehnici de eșantionare bazate pe importanță, descrise în capitolul 3. Totuși, numărul de eșantioane per pixel trebuie să rămână în continuare limitat din cauza bugetului de calcul. Pentru a reduce în continuare variantă (manifestată prin zgromot în imaginea finală), se folosește o variantă progresivă a algoritmului, care acumulează contribuția din mai multe cadre. Mai multe detalii tehnice despre implementare sunt prezentate în capitolul 5.

Algoritmul de Path Tracing folosește un sistem de materiale diferit de cel folosit de algoritmul Whitted. Dacă acesta din urmă este limitat de modelul simplu de iluminare Phong, Path Tracing folosește un model PBR (Physically Based Rendering) inspirat de cel introdus de Burley în 2012 [7], pentru a fi folosit în producția filmelor marca Walt Disney Animation Studios. Varianta implementată în această lucrare este augmentată cu o componentă de transmisie, descrisă într-un curs organizat de Hill et al. la conferința SIGGRAPH din 2015 [14]. Acest

model este mult mai complex, având la bază un BSDF (Bidirectional Scattering Distribution Function - oferă și o componentă de transmisie a luminii prin materiale) care descrie cum un material interacționează cu lumina incidentă. Fiind totuși folosit în producția cinematografică, modelul se concentrează pe a avea o interfață cât mai intuitivă pentru artistul grafic, deviind puțin de la un model fizic strict. Bazele teoretice și implementarea acestui model PBR sunt descrise în secțiunile 3 și 5.

Pentru evaluare se va face o comparație între cei doi algoritmi de Ray Tracing. Se va mai observa și capabilitatea algoritmului de Path Tracing de a simula surse de lumină de tip area light, care sunt dificil de modelat în algoritmul Whitted. Scena de test prezintă materialele PBR, exclusive pentru algoritmul de Path Tracing. Aceasta conține mai multe obiecte definite prin suprafete implicate și obiecte definite ca mesh-uri triunghiulare. Se va evalua performanța de rulare și stabilitatea imaginii generate pentru diferite setări ale algoritmului de Path Tracing.

1.5 Rezultatele obținute

Rezultatele obținute sunt încurajatoare, deși mai este loc de multe îmbunătățiri. În primul rând, aplicația este ușor de folosit, este stabilă și destul de configurabilă. Algoritmul implementat are o fidelitate bună, iar sistemul de materiale este de calitate. În al doilea rând, imaginile generate sunt foarte plăcute și realistice, mai ales datorită sistemului de materiale de calitate.

Din păcate, obiectivul de performanță nu este atins decât pentru un număr foarte mic de eșantioane per pixel (1-4). Situația se îmbunătățește semnificativ dacă se activează varianta progresivă a algoritmului, însă, în absența unui denoiser cu informație de vectori de mișcare, imaginea finală este o simplă agregare a mai multor cadre, ceea ce duce la un efect de blur pronunțat pentru obiectele în mișcare. Implementarea unei astfel de soluții este primul lucru pe lista de îmbunătățiri viitoare.

Rezultatele și interpretarea acestora sunt prezentate în capitolul 6.

1.6 Structura lucrării

Vreau să încep prin a clarifica faptul că această lucrare nu își propune să introducă noi metode sau concepte în domeniul graficii pe calculator. Scopul acesteia este de a experimenta și de a înțelege mai bine tehnologiile existente, precum și de a pune în practică teoria care stă la baza acestora. Lucrarea este structurată într-o parte teoretică inițială, menită să familiarizeze cititorul printr-o introducere lină în concepțele care stau la baza algoritmului de Path Tracing, și o parte practică, care se concentrează pe utilizarea API-ului DirectX12 pentru a implementa algoritmul cu accelerare hardware. Concepțele tehnice din urmă nu vor fi prezentate într-o lumină precisă, ci mai degrabă într-un mod simplificat și intuitiv, din perspectiva unui tool

care este folosit pentru implementarea algoritmului în sine.

În capitolul 2 este descris contextul actual în care se plasează eforturile din domeniu, din perspectiva industriilor de gaming și de cinematografie, și se prezintă motivațiile principale pentru a continua avansurile în cercetare.

În capitolul 3 se analizează stadiul curent al cercetărilor în domeniul algoritmilor de Path Tracing, cu focus pe optimizări pentru timp real. Aici se prezintă teoria care stă la baza lucrării și se explică în detaliu fiecare aspect. De asemenea, se poziționează lucrarea de față în acest peisaj și se conturează aspectul didactic al acesteia.

În capitolul 4 se prezintă funcționalitățile prezente în aplicație, precum și punerea teoriei în practică. Tot aici se prezintă sistemul de materiale specific folosit și ecuațiile care îl definesc. De asemenea, se descrie arhitectura generală a motorului grafic, cu accent pe componentele modificate și adăugate față de schelet [25].

Capitolul 5 detaliază utilizarea API-urilor DirectX 12 și DXR pentru implementarea algoritmilor de Ray Tracing cu accelerare hardware. Accentul este pus pe transpunerea aspectelor teoretice în cod și pe deciziile de design luate pentru a obține o implementare eficientă și ușor de înțeles. Tot aici se notează și dificultățile întâmpinate și compromisurile făcute pentru a obține un echilibru între fidelitate și performanță.

Capitolul de evaluare 6 prezintă rezultatele obținute în urma testelor efectuate. Se analizează performanțele sistemului, fidelitatea imaginilor generate și se fac comparații între cei doi algoritmi de Ray Tracing. De asemenea, se analizează impactul pe care îl au diferitele optimizări asupra stabilității imaginilor generate.

Ultimul capitol (7) conține concluziile trase din rezultatele obținute și se analizează calitativ produsul final. De asemenea, se discută posibile direcții de dezvoltare pe termen scurt și termen lung și se oferă o perspectivă asupra importanței acestei lucrări în contextul cercetării în domeniul graficii pe calculator.

Extrase de cod și imagini detaliate sunt oferite în anexă.

2 MOTIVATIE DE CERCETARE

Proiectul de față cercetează tehnici de randare a imaginilor în timp real. Acst domeniu este de mare interes pentru industria jocurilor video, în care se investesc resurse semnificative pentru dezvoltarea de motoare grafice puternice.

Un studiu de caz recent¹ analizează costurile de dezvoltare ale unui joc video AAA. Două dintre jocurile cu cel mai mare buget sunt Grand Theft Auto V și Cyberpunk 2077, care investit peste 270, respectiv 300 milioane de dolari în dezvoltare și marketing. O parte importantă a acestor bugete a fost alocată pentru dezvoltarea motoarelor grafice proprietare.

RAGE (Rockstar Advanced Game Engine) este motorul grafic folosit de Rockstar Games pentru jocurile sale de tip open-world, precum GTA V. Acesta a trebuit să fie adaptat pentru portarea jocului pe consolele de nouă generație (PS4 și Xbox One) și pentru PC, suportând rezoluții de până la 4K pe PC.² RAGE a primit o iteratie semnificativă odată cu lansarea Red Dead Redemption 2 în 2018 (un alt joc cu un buget de dezvoltare mare, de peste 100 milioane de dolari¹), care a adus noi tehnici de randare precum suport PBR, nori volumetrici și iluminare globală precalculată.^{3,4}

Cyberpunk 2077 a început dezvoltarea folosind un nou motor REDEngine 3,⁵ creat special pentru a îmbina o lume de joc vastă și detaliată cu o poveste complexă, bazată pe decizile jucătorului. Totuși, acest motor nu era destul de flexibil pentru a suporta toate cerințele jocului⁶ (precum first person shooting și condus de mașini), așa că au început lucrul la o nouă versiune, REDEngine 4, folosind un grant de 7 milioane de dolari de la guvernul polonez.⁷ Nici acest proces nu a fost fără dificultăți (lansarea jocului a fost dezastruoasă și plină de bug-uri,⁸ iar studio-ul a decis după lansare să tranzitioneze către Unreal Engine 5⁷ pentru jocurile viitoare), dar a permis jocului să fie primul care să ofere iluminare realizată integral cu

¹<https://vnextglobal.com/category/blog/game-development-cost-an-in-depth-analysis>. Accesat 19.06.2024.

²<https://www.eurogamer.net/digitalfoundry-2015-grand-theft-auto-5-pc-face-off>. Accesat 19.06.2024.

³<https://www.eurogamer.net/digitalfoundry-2017-red-dead-redemption-2-trailer-tech-analysis>. Accesat 19.06.2024.

⁴<https://www.eurogamer.net/digitalfoundry-2018-red-dead-redemption-2-tech-analysis>. Accesat 19.06.2024.

⁵<https://www.engadget.com/2013-02-01-cd-projekt-red-introduces-redengine-3-latest-iteration-of-in-ho.html>. Accesat 19.06.2024.

⁶<https://www.superjumpmagazine.com/why-cd-projekt-reds-switch-to-unreal-engine-is-a-big-deal/>. Accesat 19.06.2024.

⁷https://www.wipo.int/edocs/mdocs/en/wipo_smes_ge_20/wipo_smes_ge_20_p3.pdf. Accesat 19.06.2024.

⁸<https://gamerant.com/cyberpunk-2077-review-bombing-negative-impact-bad-example/>. Accesat 19.06.2024.

Path Tracing.⁹ Acest lucru a fost posibil datorită colaborării cu Nvidia,¹⁰ care a oferit suport în implementarea tehnologiei RTX în REDengine 4.

Am văzut astfel ce eforturi depun companiile mari pentru a aduce fidelitate grafică în jocurile lor. Un studiu realizat de Tondello și Nacke în 2019[33] pe două esantioane de gameri a arătat că majoritatea jucătorilor sunt interesati de aspectele estetice ale jocurilor video, precum grafica și sunetul. În alt studiu realizat de Katja et al. în 2022[29], s-a analizat concepția literaturii asupra realismului în jocuri video. Deși s-a concluzionat că acest termen nu este bine definit de multe ori, cel mai adesea în literatură acesta se referă, printre altele, la fidelitatea grafică a jocului.

Așadar, unul dintre cele mai importante aspecte ale unui joc video, în relație cu experiența jucătorului, este fidelitatea grafică.¹¹ Aceasta este influențată de calitatea modelelor 3D, a texturilor, a animațiilor, a efectelor speciale, dar și de iluminare. Iluminarea este un aspect critic al fidelității grafice, deoarece aceasta influențează cum percepem obiectele din joc. O lume frumos modelată nu poate avea un impact vizual puternic dacă nu este pusă într-o "lumină bună". Această "lumină bună" izvorăște adânc din tehnologiile folosite de motorul grafic pentru a da valoare obiectelor în scenă. Deși există multe stiluri atractive de a prezenta o scenă (e.g., cel-shading, pixel art), realismul este unul dintre cele mai populare, deoarece oferă o experiență de joc mai imersivă. Clasa de algoritmi de Ray Tracing este una dintre cele mai bune tehnici de a obține realism în jocuri, însă aceasta vine cu un cost computațional ridicat. De aceea, eforturi de cercetare în domeniul său sunt necesare pentru a găsi soluții care să ofere un compromis între fidelitate și performanță.¹² Ca referință, NVIDIA oferă public multe studii de cercetare și articole științifice publicate de echipa lor de cercetare.¹³ Marea majoritate se concentrează pe optimizarea tehnicii de prezentare grafică (Path Tracing reprezentând o parte importantă a acestora) și oferă o privire de ansamblu asupra eforturilor de cercetare în domeniu.

Ca o ultimă observație, să ne imaginăm că fidelitatea cu care se realizează producția filmelor de animație ar putea fi adusă în jocurile video. Din cealaltă perspectivă, filmele video ar putea fi randate în timp real, fără consum enorm de energie. Aceste avantaje ar reduce costurile de producție și ar accelera procesul de dezvoltare a filmelor. Eforturile de cercetare în domeniu sunt necesare pentru a face aceste viziuni realitate.

La nivel personal, această lucrare reprezintă o oportunitate de a învăța și de a experimenta cu tehnologii avansate de randare a imaginilor. Scriu lucrarea de față în ideea în care dacă ar fi să o iau de la început și să învăț aceste concepte din nou, aş vrea să am la dispoziție un

⁹<https://www.tomshardware.com/news/cyberpunk-277-rt-overdrive-available-to-all>. Accesat 19.06.2024.

¹⁰<https://www.nvidia.com/en-us/geforce/news/cyberpunk-2077-nvidia-partnership-ray-tracing/>. Accesat 19.06.2024.

¹¹<https://goombastomp.com/why-good-graphics-matter-in-video-games-enhancing-the-visual-experience/>. Accesat 19.06.2024.

¹²<https://blogs.nvidia.com/blog/rtx-real-time-ray-tracing/>. Accesat 19.06.2024.

¹³<https://research.nvidia.com/labs/rtr/publication/>. Accesat 19.06.2024.

ghid simplu și intuitiv care să mă ajute să înțeleg teoria și să o pun în practică. În cercetarea efectuată de mine nu am găsit o introducere completă și accesibilă, mai ales în contextul API-urilor de ultimă generație precum DirectX 12. Așadar, această lucrare își propune să fie un astfel de ghid, care să ofere cititorului o introducere lină și încurajatoare.

3 METODE EXISTENTE

Literatura de specialitate din domeniul graficii pe calculator este vastă. Există multe metode de randare a imaginilor și se dă o luptă constantă între a balansa performanța cu fidelitatea. Direcția de cercetare cea mai proeminentă se axează în jurul metodelor de tip Monte Carlo, care reprezintă state-of-the-art în domeniu. Deși concepțele care vor fi prezentate nu sunt noi (metode de rezolvare a ecuației de randare există de aproape 40 ani), apar întotdeauna noi tehnici de optimizare și de îmbunătățire a performanțelor de convergență și de stabilitate a algoritmilor. Pentru o privire mai detaliată asupra avansurilor curente și asupra viitorului cercetării în domeniu, notițele de curs din 2019 ale lui Keller et al. [18] sunt o resursă excelentă. De asemenea, pentru o privire de ansamblu și de actualitate asupra tehnologiilor de accelerare în timp real folosite în industria jocurilor video (mai ales cele de la Nvidia - RTXDI, RTXGI, NRD, DLSS), recomand prezentarea de la GTC și GDC 2022 a lui Clarberg et al. [8], din partea Nvidia Corporation.

În continuare, vom prezenta fundamentele teoretice ale clasei de algoritmi Ray Tracing și vom analiza cele mai importante metode existente.

3.1 Rasterizare

Pentru a avea un punct de plecare, vom vorbi puțin și despre rasterizare. Aceasta este metoda de bază folosită în majoritatea jocurilor video din trecut și de astăzi. Ea este reprezentată ca un stagiu fix (neprogramabil) din pipeline-ul de randare al GPU-ului (vezi Figura 54), care transformă primitivele definite vectorial (triunghiuri, linii, puncte) în pixeli pe ecran. Acest proces este foarte eficient, deoarece folosește hardware specializat pentru a face calculele necesare.

Etapele principale ale rasterizatorului sunt¹:

1. *Clipping* - eliminarea primitiveelor care nu se află în câmpul vizual (view frustum)
2. *Perspective division* - împărțirea coordonatelor omogene pentru a obține coordonatele normalize (NDC)
3. *Transformarea viewport* - transformarea coordonatelor normalize în coordonate ecran
4. *Rasterizarea* - determinarea pixelilor acoperiți de primitivă.

După rasterizare urmează etapa programabilă de pixel shader, unde se calculează de obicei

¹<https://learn.microsoft.com/en-us/windows/uwp/graphics-concepts/rasterizer-stage--rs->. Accesat 19.06.2024.

efecte de iluminare, umbre, texturi etc. Un exemplu de imagine randată cu rasterizare se poate vedea în Figura 5.

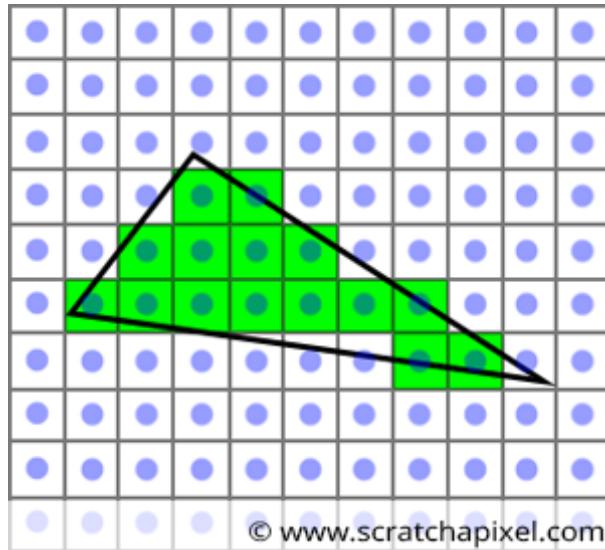


Figura 5: Exemplu de imagine randată folosind rasterizare²

Deși rasterizarea este foarte eficientă, ea are multe limitări. Efectele de iluminare pot fi doar approximate, iar umbrele și reflexiile/refracțiile sunt greu de realizat. Motoarele grafice folosesc metode de baking³ pentru a precalcula iluminarea statică în scenă la o calitate bună (folosind alte metode, e.g., radiosity (3.4)), dar iluminarea dinamică și umbrele dinamice sunt randate la calitate scăzută. Printre tehniciile folosite pentru a îmbunătăți aspectul vizual al jocurilor se numără ocluzia ambientală, shadow mapping, screen-space reflections, cubemap reflections, light probes, lightmaps, etc.

3.2 Ray Tracing

În sens general, Ray Tracing reprezintă o clasă de algoritmi și tehnici de randare a imaginilor care au la bază simularea transportului luminii într-o scenă. Spre deosebire de rasterizare, care proiectează obiectele 3D pe un plan 2D, Ray Tracing rezolvă problema vizibilității trasând raze din ochiul camerei în scenă și calculând intersecțiile cu geometria 3D. Vom prezenta în continuare diferite variații ale acestui algoritm.

Ray Casting

Prima aplicație a algoritmului în grafica pe calculator a fost făcută de Appel în 1968 [5] - în acest context algoritmul era nerecursiv. Razele primare calculau intersecțiile cu obiectele din

²©<https://www.scratchapixel.com/>. Accesat 19.06.2024.

³https://www.flipcode.com/archives/Light_Mapping_Theory_and_Implementation.shtml. Accesat 20.06.2024.

scenă, iar razele secundare erau folosite pentru a calcula umbre. Această variantă nerecursivă a algoritmului este cunoscută sub numele de Ray Casting. Figura 6 prezintă vizualizarea algoritmului.

Datorită simplității sale, Ray Casting este un algoritm foarte eficient (fiind ușor paralelizabil), însă nu oferă în sine niciun efect de iluminare - este un algoritm de vizibilitate.

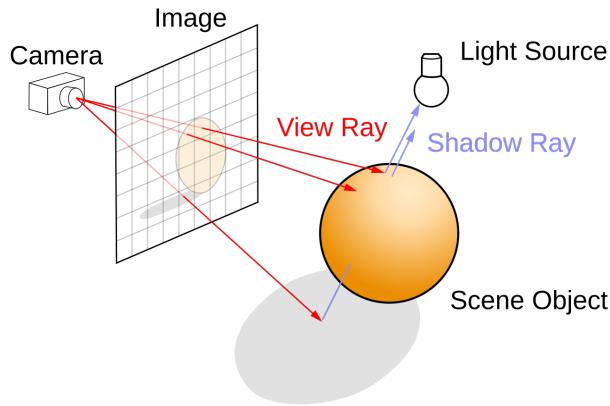


Figura 6: Algoritmul Ray Casting. Cu roșu sunt reprezentate razele primare, iar cu albastru razele secundare⁴

Ray Tracing Recursiv

Algoritmul inițial de Ray Casting nu putea calcula efecte precum reflexii și refracții. Pentru acestea, era nevoie de o variantă recursivă a algoritmului, care să calculeze traectoria razei de-a lungul mai multor puncte din scenă și să țină cont de contribuțiile tuturor suprafețelor intersectate. Această variantă a fost prezentată în practică pentru prima dată de Whitted (1979) [38]. Astfel, fiecare intersecție, se pot genera 3 noi raze: o rază de reflexie, o rază de refracție și o rază de umbrit. Adâncimea recursivității este, evident, limitată, căci complexitatea crește exponențial. Este de remarcat faptul că efectele de reflexie, refracție, umbrări, precum și alte efecte specifice (e.g., blur, depth of field) pot fi modelate foarte ușor folosind Ray Tracing recursiv, prin opoziție cu rasterizarea. O ilustrare a algoritmului este prezentată în Figura 7.

Turner Whitted a oferit într-un blog post⁵ pe site-ul de la Nvidia o mică retrospectivă a algoritmului său și a deciziilor pe care a trebuit să le facă la momentul respectiv - este o lectură scurtă dar interesantă.

⁴©<https://en.wikipedia.org/>. Accesat 20.06.2024.

⁵©<https://blogs.nvidia.com/blog/ray-tracing-global-illumination-turner-whitted/>. Accesat 20.06.2024.

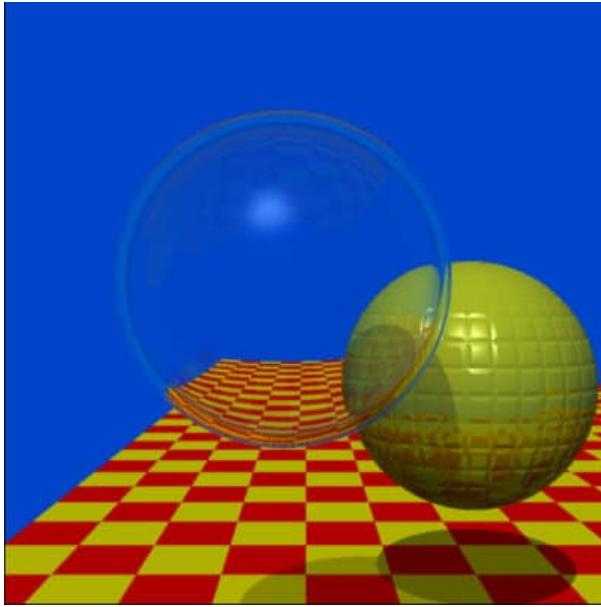


Figura 7: Algoritmul Ray Tracing recursiv, ilustrat de Turner Whitted⁵

Ray Marching

Un algoritm foarte drag mie, datorită eleganței matematice, este Ray Marching. Dacă celelalte metode de Ray Tracing calculează intersecții cu obiecte definite prin primitive geometrice (e.g., triunghiuri), această variantă ia în considerare doar suprafete definite prin câmpuri de distanță (SDF). Un câmp de distanță este o funcție asociată unei mulțimi care întoarce distanța ortogonală de la un punct din spațiu la frontieră mulțimii. Un SDF are valori pozitive pentru puncte aflate în afara mulțimii și valori negative pentru puncte aflate în interiorul mulțimii. Principala caracteristică a acestui algoritm care îl diferențiază de celelalte este faptul că nu calculează intersecții. În schimb, fiecare rază mărșăluiește treptat în spațiu, apropiindu-se din ce în ce mai mult de suprafetele definite, dar fără a le intersecta. La fiecare pas, algoritmul se folosește de aceste câmpuri de distanță pentru a calcula o rază minimă în jurul punctului curent, care reprezintă sferă de dimensiune maximă care nu intersectează cu certitudine nicio suprafață. Această rază este numită pas de marșăluire (marching step) și este folosită pentru a avansa în spațiu. Algoritmul se oprește când distanța minimă calculată este mai mică decât o anumită toleranță (indicând faptul că o intersecție este foarte aproape), sau când un număr maxim de pași prestatibil este atins (caz în care nu se înregistrează nicio intersecție). Datorită utilizării sferelor pentru mărșăluire, algoritmul mai este cunoscut și sub numele de Sphere Tracing. O ilustrație a funcționării acestuia se poate vedea în Figura 8.

O aplicație naturală a acestui algoritm este randarea suprafetelor implice. Acestea sunt suprafete în spațiul Euclidian definite prin ecuații de forma:

$$f(x, y, z) = 0. \quad (3.2.1)$$

Funcția f definește câmpul de distanță al suprafetei. De asemenea, se pot calcula foarte ușor și vectorii normali la suprafață (gradientul funcției f), ceea ce face posibilă randarea efectelor

de iluminare a suprafețelor. Exemple de randări ale unor suprafețe implicate se pot vedea în Figurile 9 și 10.

Deși acest algoritm a fost studiat în literatura de specialitate încă din 1996 de către Hart [12], el a devenit popular în cadrul demoscene-ului⁶ și a comunității de programatori de shadere. Inigo Quilez a fost printre primii care a adus algoritmul în atenția publicului larg, prin intermediul platformei Shadertoy⁷ și a blogului personal.⁸

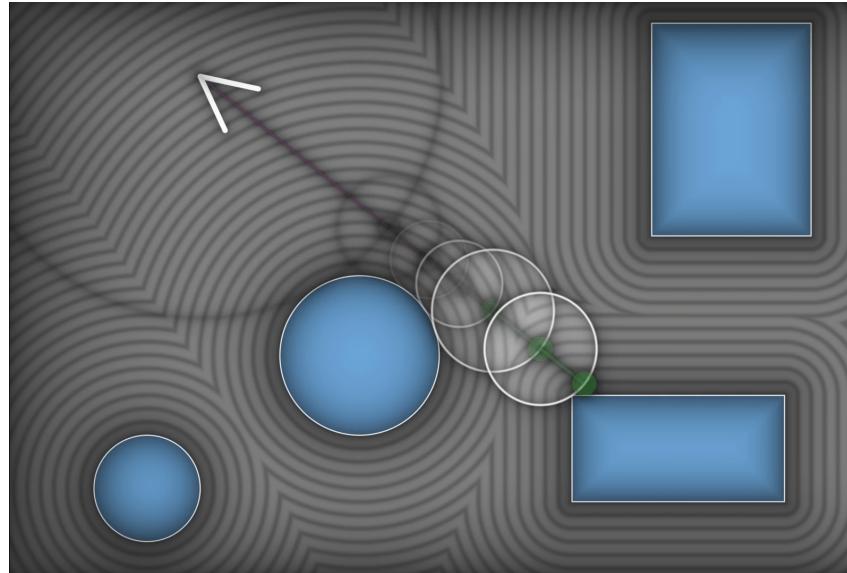


Figura 8: Algoritmul Sphere Tracing - reprezentare 2D. Se pot observa și câmpurile de distanță ale obiectelor⁹

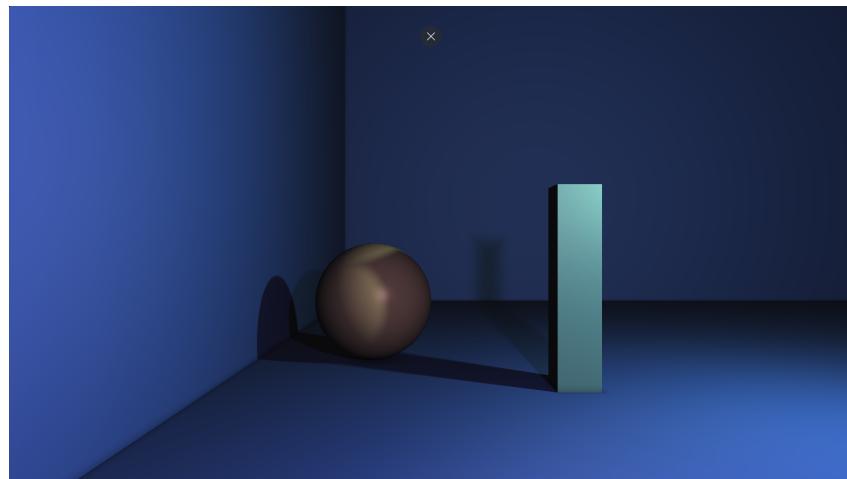


Figura 9: Shader scris în GLSL care demonstrează soft shadows bazate pe difracție¹⁰

⁶<https://en.wikipedia.org/wiki/Demoscene>. Accesat 20.06.2024.

⁷<https://www.shadertoy.com/>. Accesat 20.06.2024.

⁸<https://iquilezles.org/>. Accesat 20.06.2024.

⁹<https://www.youtube.com/@simondev758>. Accesat 20.06.2024.

¹⁰Shader scris de mine, disponibil la adresa: <https://www.shadertoy.com/view/tscSRS>. Accesat 20.06.2024.

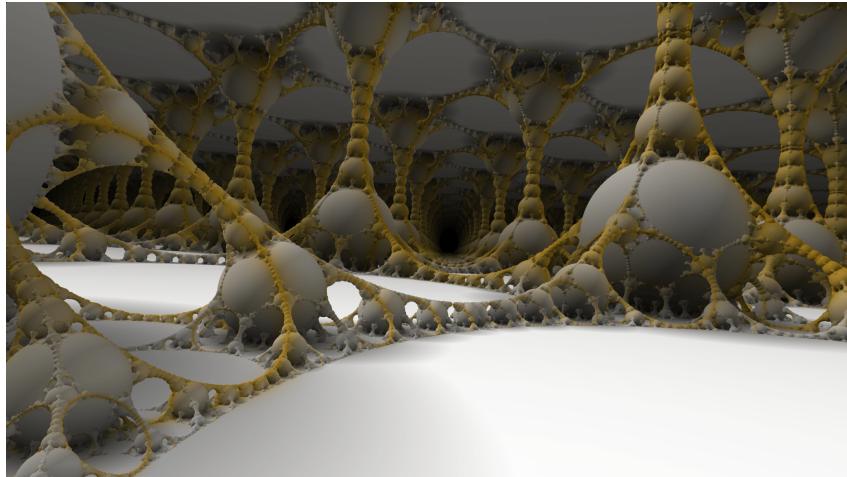


Figura 10: Randare 3D a unui fractal de tip Apollonian¹¹

3.3 Metode Monte Carlo

Algoritmii prezentăți mai sus au pus bazele teoretice și practice pentru metode mai avansate de randare, care se concentrează pe realism. Un dezavantaj major al metodelor prezentate până acum este faptul că acestea nu sunt fotorealistice - ele nu iau în considerare toate fenomenele interacțiunii luminii cu materialele. Deși Ray Tracing recursiv poate calcula reflexii și refracții perfecte, acesta nu poate simula din punct de vedere fizic efecte precum difuzia, dispersia, radianța, etc. Spre exemplu, un obiect care nu este perfect reflectant sau refractant va difuza lumina în toate direcțiile, nu doar în cea de reflexie sau refracție geometrică. De asemenea, algoritmul de Ray Tracing recursiv consideră că lumina este o sursă punctiformă, când de fapt aceasta are o distribuție finită în spațiu. Astfel, umbrele generate de acest algoritm sunt "dure" și nu iau în considerare umbrele penumbrale. Este adevărat că se pot approxima aceste efecte folosind diferite modele de iluminare care se aplică și în rasterizare (e.g., modelul Phong [27]), dar acestea sunt doar approximații și nu oferă realismul dorit.

Este nevoie aşadar de modele mai avansate ale transportului luminii, care să simuleze fenomenele fizice cu o acuratețe cât mai mare. Vom prezenta în continuare metode care au la bază soluționarea ecuației de transport a luminii [17] (folosind extensiv concepte din teoria probabilității), metode care oferă o fidelitate net superioară celor prezentate anterior. Pentru că scopul acestei lucrări nu este de a oferi o introducere în teoria probabilității, vom prezenta doar concepțele de bază necesare înțelegерii algoritmilor de tip Monte Carlo. Pentru o introducere mai detaliată în acest domeniu, cititorul poate consulta lucrări de specialitate precum [10] și [11].

Metodele de tip Monte Carlo sunt metode iterative care folosesc eșantionare aleatoare pentru a aproxima valoarea unei integrale sau pentru a simula comportamentul unui sistem complex determinist, modelat stocastic. Ele sunt utilizate pentru a găsi soluții aproximative, care să conveargă către soluția corectă, pentru probleme intractabile sau care nu pot fi rezolvate

¹¹©Inigo Quilez: <https://www.shadertoy.com/view/4ds3zn>. Accesat 20.06.2024.

analitic.

Un exemplu de pași pe care îi urmează un algoritm de tip Monte Carlo este:

1. Definirea unui domeniu de eșantionare
2. Generarea unui număr de eșantioane în domeniu, folosind o distribuție de probabilitate
3. Efectuarea calculelor (deterministe) pentru fiecare eșantion, care să aproximeze soluția
4. Agregarea rezultatelor.

3.3.1 Integrare Monte Carlo

Relevant în particular pentru această lucrare este conceptul de integrare Monte Carlo. Acesta se referă la aproximarea unei integrale definite folosind metode Monte Carlo de eșantionare.

Să luăm un exemplu de problemă. Dându-se o funcție

$$f : \mathbb{D} \rightarrow \mathbb{R}$$

și o variabilă aleatoare continuă X cu distribuția de probabilitate $p(x)$, vrem să calculăm valoarea medie (expected value):

$$\mathbb{E}_p(f(X)) = \int_{\mathbb{D}} f(x)p(x) dx. \quad (3.3.1)$$

Valoarea medie se poate approxima prin eșantionare, folosind formula:

$$\mathbb{E}_p(f(X)) \approx \frac{1}{N} \sum_{i=1}^N f(x_i), \quad (3.3.2)$$

aproximarea devenind mai bună cu creșterea numărului de eșantioane N .

Așadar, putem approxima integrala (3.3.1) prin:

$$\int_{\mathbb{D}} f(x)p(x) dx \approx \frac{1}{N} \sum_{i=1}^N f(x_i). \quad (3.3.3)$$

Partea dreaptă a ecuației (3.3.3) este estimatorul Monte Carlo. Teorema limitei centrale afirmă că, pentru un număr suficient de mare de eșantioane, distribuția de probabilitate a estimărilor se apropiă de o distribuție normală în jurul valorii medii estimate. Acest lucru înseamnă că estimatorul Monte Carlo este imparțial (unbiased) și are o varianță redusă. Dacă notăm cu f_p estimatorul, atunci au loc următoarele relații:

$$\begin{aligned} \mathbb{E}_p(f_p) &= \mathbb{E}_p(f(X)), \\ Var_p(f_p) &= \frac{1}{N} Var_p(f(X)). \end{aligned} \quad (3.3.4)$$

3.3.2 Eșantionare bazată pe importanță

În practică, distribuția de probabilitate $p(x)$ este adesea greu de eșantionat, sau nu produce varianța cea mai mică, ceea ce duce la o convergență lentă a estimării. Eșantionarea pe baza importanței (importance sampling) este o tehnică prin care se eșantionează variabila aleatoare X dintr-o altă distribuție de probabilitate, $q(x)$, cu scopul de a reduce varianța și a estima mai bine valoarea medie a funcției f .

Să introducem această distribuție în ecuația valorii medii (3.3.1):

$$\begin{aligned}\mathbb{E}_p(f(X)) &= \int_{\mathbb{D}} f(x)p(x) dx \\ &= \int_{\mathbb{D}} f(x) \frac{p(x)}{q(x)} q(x) dx \\ &= \mathbb{E}_q \left(f(X) \frac{p(X)}{q(X)} \right).\end{aligned}\tag{3.3.5}$$

Obținem astfel un nou estimator Monte Carlo, notat cu f_q , care își păstrează imparțialitatea (căci valoarea medie rămâne aceeași) și care folosește distribuția $q(x)$:

$$\mathbb{E}_q(f_q) = \mathbb{E}_p(f(X)) \approx \frac{1}{N} \sum_{i=1}^N f(x_i) \frac{p(x_i)}{q(x_i)}.\tag{3.3.6}$$

Varianța acestui estimator, este dată de:

$$Var_q(f_q) = \frac{1}{N} Var_q \left(f(X) \frac{p(X)}{q(X)} \right).\tag{3.3.7}$$

Scopul este de a alege distribuția $q(x)$ astfel încât varianța estimatorului să fie mai mică decât varianța inițială, i.e.:

$$\frac{1}{N} Var_q \left(f(X) \frac{p(X)}{q(X)} \right) < \frac{1}{N} Var_p(f(X)).\tag{3.3.8}$$

Pentru a minimiza varianța noului estimator, în mod ideal, vrem ca funcția $f(x) \frac{p(x)}{q(x)}$ să fie constantă, pentru orice x eșantionat, ceea ce ar duce la o varianță nulă. Acest lucru se întamplă dacă:

$$q(x) = c \cdot f(x)p(x),\tag{3.3.9}$$

unde c este o constantă de normalizare. Pentru a păstra imparțialitatea, c trebuie să fie egal cu $\frac{1}{\mathbb{E}_p(f(X))}$, însă această valoare este necunoscută (altfel nu am mai avea nevoie de estimator!). Așadar, nu este fezabilă această alegere pentru $q(x)$. Totuși, ecuația (3.3.9) ne arată că distribuția optimă este proporțională cu funcția $f(x)p(x)$. Acest produs măsoară fix importanța fiecărui eșantion în estimarea mediei, de unde și numele tehnicii.

În Figura 11 se poate observa efectul eşantionării bazate pe importanță. În acest caz, funcția f are o regiune mică de unde vine majoritatea contribuției către valoarea medie. Dacă distribuția de probabilitate p nu eşantionează bine această regiune (în acest caz, este o distribuție uniformă), varianța estimării va fi mare. Distribuția q este aleasă astfel încât să eşantioneze mai mult din zonele importante și se poate vedea pe subgraficul din dreapta cum raportul $\frac{p(x)}{q(x)}$ încearcă să echilibreze contribuția fiecărui eşantion, reducând varianța estimării.

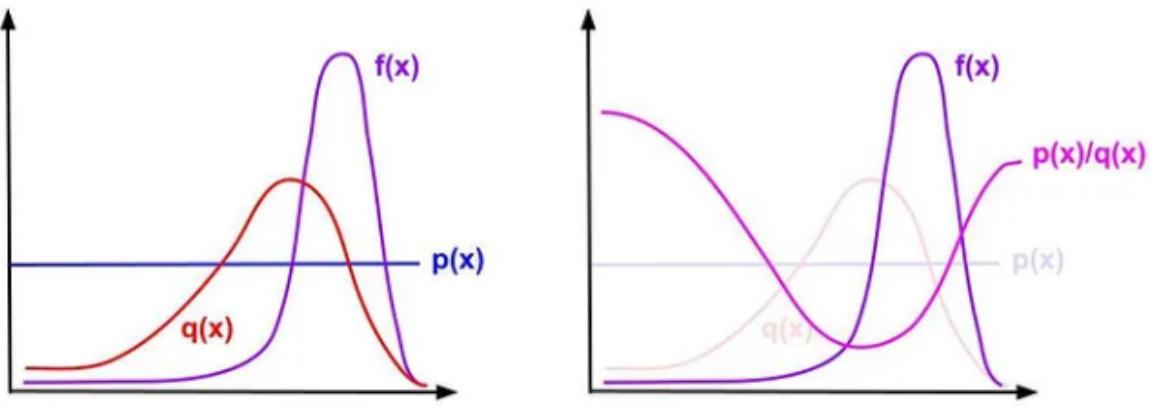


Figura 11: Efectul eşantionării bazate pe importanță¹²

3.3.3 Eşantionare Stratificată

O altă metodă de a reduce varianța estimării este eşantionarea stratificată (Stratified Sampling). În această metodă se partiționează domeniul de eşantionare \mathbb{D} în n subdomenii, numite straturi, iar evaluarea integralei se face pe fiecare strat îndeplinește:

$$\int_{\mathbb{D}} f(x)p(x) dx = \sum_{i=1}^n \int_{\mathbb{D}_i} f(x)p(x) dx. \quad (3.3.10)$$

În acest caz, varianța estimării devine suma varianțelor pe fiecare strat:

$$Var(f_s) = \sum_{i=1}^n Var(f_i). \quad (3.3.11)$$

Se poate demonstra că, în cazul în care toate straturile au aceeași măsură, i.e.:

$$\int_{\mathbb{D}_i} p(x) dx = \frac{1}{n} \int_{\mathbb{D}} p(x) dx, \quad \forall i \in \{1, 2, \dots, n\}, \quad (3.3.12)$$

atunci varianța estimării nu va fi mai mare decât fără eşantionare stratificată. Pentru mai multe informații se poate consulta cartea de specialitate a lui Kleijnen et al. [19].

¹²©<https://medium.com/>

Un exemplu de eșantionare stratificată este ilustrat sub tehnica de jittering în eșantionarea pixelilor, detaliată în capitolul 5.

3.3.4 Ecuăția transportului luminii

Introdusă simultan de Kajiya [17] și Immel et al. [15] în 1986, ecuația transportului luminii (sau ecuația de randare) este cea mai importantă ecuație din domeniul graficii pe calculator. Aceasta descrie modul în care lumina interacționează cu suprafetele și ajunge către observator. Una dintre formele acesteia este:

$$\begin{aligned} L_o(\mathbf{x}, \omega_o, \lambda, t) &= L_e(\mathbf{x}, \omega_o, \lambda, t) + L_r(\mathbf{x}, \omega_o, \lambda, t) \\ L_r(\mathbf{x}, \omega_o, \lambda, t) &= \int_{\Omega_+} f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t) (\omega_i \cdot \mathbf{n}) d\omega_i. \end{aligned} \quad (3.3.13)$$

Sunt destul de multe simboluri folosite în această ecuație, aşa că le vom explica pe rând. Folosind Figura 12 drept referință:

- \mathbf{x} este poziția punctului de intersecție cu suprafața
- ω_o este direcția de observare (direcția pe care vrem să măsurăm radianța)
- ω_i este opusul direcției de incidentă (direcționat de la punctul de intersecție către sursa de lumină)
- \mathbf{n} este normala la suprafață în punctul \mathbf{x}
- λ este lungimea de undă a luminii
- t este un moment particular în timp
- $L_o(\mathbf{x}, \omega_o, \lambda, t)$ este radianța spectrală de lungime de undă λ observată în punctul \mathbf{x} , pe direcția ω_o , la momentul t
- $L_e(\mathbf{x}, \omega_o, \lambda, t)$ este radianța spectrală de lungime de undă λ emisă de suprafață în punctul \mathbf{x} , pe direcția ω_o , la momentul t
- $L_r(\mathbf{x}, \omega_o, \lambda, t)$ este radianța spectrală de lungime de undă λ reflectată în punctul \mathbf{x} , pe direcția ω_o , la momentul t
- $L_i(\mathbf{x}, \omega_i, \lambda, t)$ este radianța spectrală de lungime de undă λ incidentă în punctul \mathbf{x} , pe direcția ω_i , la momentul t
- $f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t)$ este funcția de distribuție bidirecțională a reflectanței (BRDF), care reprezintă câtă lumină este reflectată în direcția ω_o din direcția ω_i
- Ω_+ este emisfera unitate superioară centrală în jurul normalei \mathbf{n} , care conține toate direcțiile posibile de incidentă ω_i pentru care $\omega_i \cdot \mathbf{n} > 0$.

Putem observa că această ecuație nu include o componentă de transmisie a luminii. Putem augmenta aditiv ecuația 3.3.13 cu o componentă de transmisie, definită astfel:

$$L_t(\mathbf{x}, \omega_o, \lambda, t) = \int_{\Omega_-} f_t(\mathbf{x}, \omega_t, \omega_o, \lambda, t) L_i(\mathbf{x}, \omega_t, \lambda, t) (\omega_t \cdot \mathbf{n}) d\omega_t, \quad (3.3.14)$$

cu diferență că Ω_- este emisfera unitate inferioară centrată în jurul normalei \mathbf{n} , care conține toate direcțiile posibile de incidentă internă ω_t pentru care $\omega_t \cdot \mathbf{n} < 0$. În acest caz, funcția $f_t(\mathbf{x}, \omega_t, \omega_o, \lambda, t)$ este funcția de distribuție bidirecțională a transmisiei (BTDF), care reprezintă câtă lumină este transmisă în direcția ω_o din direcția ω_t .

De obicei, componenta de reflectanță și cea de transmisie se combină într-o singură componentă, care are la bază o funcție de distribuție bidirecțională a împrăștierii (BSDF). Mai multe detalii despre această clasă de funcții, într-un context de materiale bazate pe fizică, se pot găsi în secțiunea 3.5.2.

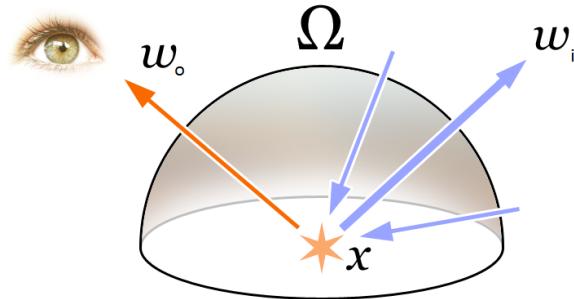


Figura 12: Ilustrație a componentelor din ecuația de randare⁴

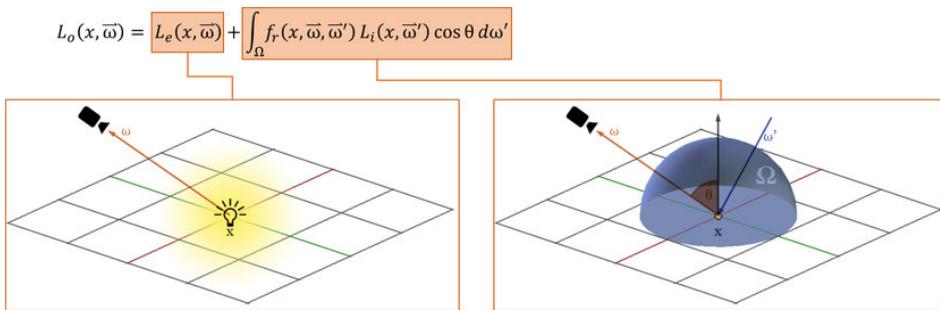


Figura 13: Formă alternativă a ecuației transportului luminii. Se pot vedea componente de emisie și împărăștiere¹³

Găsirea soluției la ecuația transportului luminii (i.e., determinarea radianței L_o) este provocarea primară în algoritmii de randare realistică. Vom enumera în secțiunea următoare câteva dintre metodele de rezolvare a acestei ecuații, însă ne vom concentra asupra algoritmului de Path Tracing.

3.4 Radiosity

Radiosity este o tehnică de iluminare globală bazată pe metoda elementului finit. Aceasta presupune împărțirea scenei în elemente mici, numite petice, și calcularea energiei luminoase reflectată de fiecare dintre acestea. Algoritmul în sine consideră numai interacțiunile de tip

¹³©<https://www.researchgate.net/>. Accesat 20.06.2024.

difuz, ceea ce îl face să fie un algoritm independent de direcția de vizualizare. De aceea, acesta poate fi folosit în special pentru a precalcula iluminarea statică a scenei (spre exemplu, la compilarea unei hărți create în editorul Hammer al companiei Valve se aplică acest algoritm sub forma utilitarului VRAD¹⁴).

Algoritmul funcționează prin calcularea vizibilității între petice și asocierea unor factori de vizualizare pentru fiecare pereche. Acest factor descrie cât de bine se văd două petice între ele.

Într-o variantă brută a algoritmului, factorii sunt folosiți drept coeficienți pentru a rezolva un sistem de ecuații liniare (unde ecuațiile sunt variante simplificate ale ecuației de transport al luminii). Soluția acestui sistem oferă radiozitatea fiecărui petic (i.e., luminozitatea). O ilustrație a rezultatului algoritmului într-o scenă de tip Cornell Box se poate vedea în Figura 14.

O variantă optimizată, denumită "shooting radiosity", folosește un proces iterativ în care la fiecare pas se emite lumină dintr-un petic și se calculează radianța reflectată de celelalte petice. Acest proces se repetă până când se atinge o stare stabilă, aşa cum se poate vedea în Figura 15.

Avantajele algoritmului sunt faptul că este relativ simplu de implementat, nu necesită matematică avansată și deci este un instrument didactic bun. De asemenea, caracterul său independent de direcție îl face potrivit pentru precalcularea iluminării statice. Ca dezavantaje, este un algoritm lent și trebuie făcut un compromis între timpul de randare și calitatea rezultatului (e.g., rezoluția peticelor, numărul de pași). Nu este potrivit pentru iluminarea speculară sau transmisie, fiind limitat la interacțiuni de tip difuz (deși poate fi extins la medii non-difuze [15]). Un alt aspect care îl limitează este nevoia de a precalcula funcția de vizibilitate între petice. Din experiență proprie, lucrând la hărți complexe în editorul Hammer,¹⁵ era necesar să ajut manual algoritmul de radiosity prin partionarea artificială a scenei în zone de vizibilitate,¹⁶ pentru a reduce numărul de perechi de petice care trebuiau luate în considerare. Fără această intervenție, doar calculul vizibilității¹⁷ putea să dureze și 24 de ore.

¹⁴<https://developer.valvesoftware.com/wiki/VRAD>. Accesat 20.06.2024.

¹⁵https://developer.valvesoftware.com/wiki/Valve_Hammer_Editor. Accesat 20.06.2024.

¹⁶https://developer.valvesoftware.com/wiki/VIS_optimization. Accesat 20.06.2024.

¹⁷<https://developer.valvesoftware.com/wiki/VVIS>. Accesat 20.06.2024.

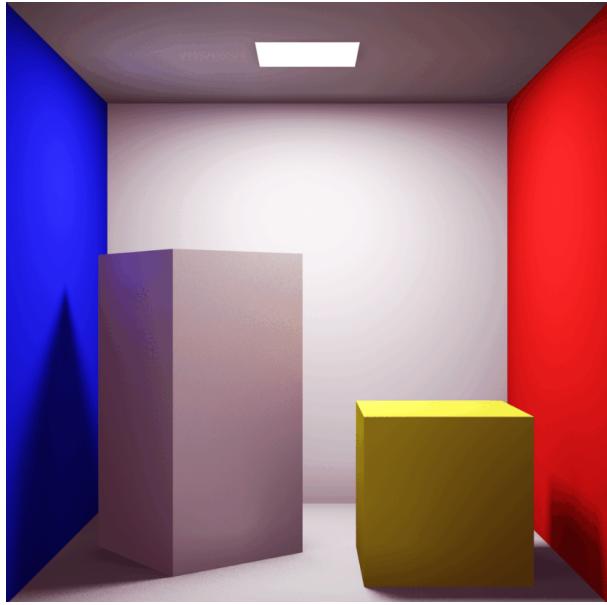


Figura 14: Rezultatul algoritmului de radiosity într-o scenă de tip Cornell Box⁴

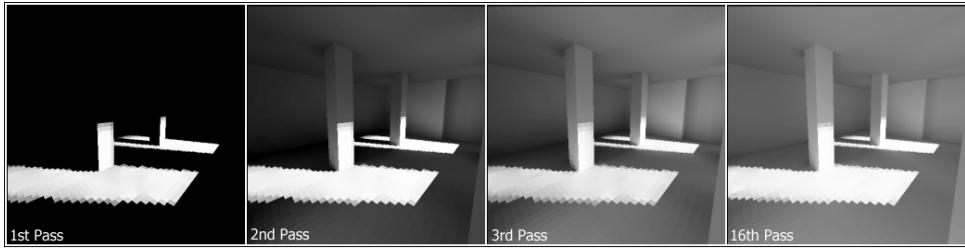


Figura 15: Ilustrație a algoritmului de shooting radiosity. Tot aici se poate observa și rezoluția peticelor⁴

3.5 Path Tracing

Algoritmul Radiosity face o presupunere simplificatoare, și anume că iluminarea venită de pe suprafete depinde doar de direcția de ieșire, și nu și de direcția incidentă. Vedem totuși că ecuația transportului luminii (3.3.13) include un termen de distribuție f care depinde de ambele direcții (de unde și numele de distribuție bidirectională). Algoritmul de Path Tracing modelează acest aspect, ajustând probabilistic contribuția fiecărui eșantion la radianță observată.

O variantă naivă a algoritmului este prezentată în Pseudocodul 1. Dacă ar fi să sumarizăm pașii algoritmului, aceștia ar fi:

1. Pentru fiecare pixel din imagine:
 - (a) Generăm o rază de la observator către pixel
 - (b) Intersectăm raza cu scena
 - (c) Dacă nu există intersecție, pixelul primește culoarea de fundal și ne întoarcem la pasul 1

- (d) Altfel, calculăm contribuția de lumină pentru punctul intersectat (folosind BRDF)
- (e) Eșantionăm o nouă rază pornind din punctul intersectat, în funcție de BRDF
- (f) Calculăm contribuția de lumină pentru noua rază, recursiv

2. În final, mediem contribuțiile pentru a obține culoarea pixelului.

Algoritmul 1 Pseudocodul algoritmului de Path Tracing recursiv

```

1: function PATHTRACE(ray, depth)
2:   if depth > MAX_DEPTH then
3:     return backgroundColor                                ▷ Ne oprim dacă am atins adâncimea maximă
4:   end if
5:   intersection ← INTERSECTSCENE(ray)                ▷ Intersectăm raza cu scena
6:   if intersection = null then
7:     return backgroundColor                                ▷ Ne oprim dacă nu există intersecție
8:   end if
9:   m ← intersection.material
10:  p ← intersection.position
11:  n ← intersection.normal
12:  v ← -ray.direction
13:  bounceRay ← SAMPLERAY(m, p, n)                    ▷ Eșantionăm o nouă rază
14:  pdf ← PDF(m, n, bounceRay)                         ▷ Evaluăm probabilitatea de eșantionare
15:  reflectance ← EVALBRDF(m, n, v, bounceRay)        ▷ Evaluăm reflectanță
16:  color ← PATHTRACE(bounceRay, depth + 1)           ▷ Pasul recursiv
17:  return m.emission + reflectance · color/pdf      ▷ Evaluăm ecuația de rendering
18: end function
19: function RENDER(pixels, samples)
20:   for p in pixels do
21:     for i ← 1 to samples do
22:       ray ← GENERATECAMERARAY(p, i)                  ▷ Generăm raza de la observator
23:       pixel.color ← pixel.color + PATHTRACE(ray, 0)    ▷ Acumulăm contribuția
24:     end for
25:     pixel.color ← pixel.color / samples            ▷ Mediem contribuțiile
26:   end for
27: end function

```

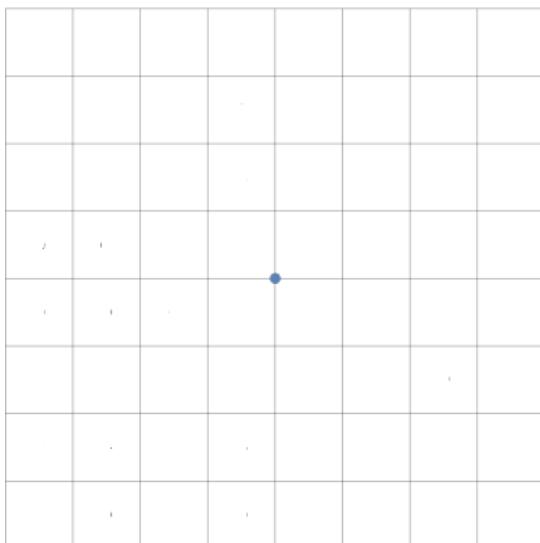
Algoritmul Path Tracing reprezintă transpunerea completă a ecuației de randare într-un algoritm de calcul numeric. Se poate cupla orice tip de BRDF la acest algoritm, ceea ce îl face extrem de versatil. Se pot obține simulări de interacțiuni difuze (precum cele obținute cu Radiosity), folosind un BRDF Lambertian [23], sau simulări de interacțiuni speculare, folosind un BRDF precum Cook-Torrance([9]). Se poate extinde de asemenea și la interacțiuni de transmisie, folosind un BTDF. Path Tracing este o metodă generală de rezolvare a ecuației de randare, iar componente sale pot fi alese în funcție de necesități. Identificăm astfel câteva dintre diferențele categorii de strategii de bază care pot fi customizate în Path Tracing.

3.5.1 Strategii de eșantionare a pixelilor

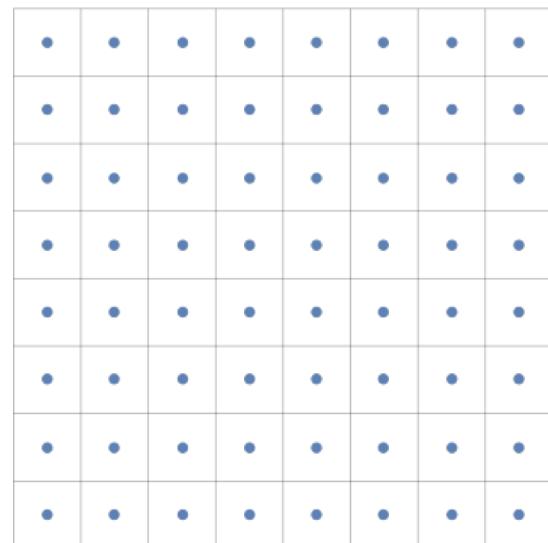
Pentru o convergență mai rapidă și o imagine mai stabilă, se rulează algoritmul de mai multe ori pe același pixel și se combină rezultatele. Strategia de alegere a eșantioanelor aferente unui pixel poate influența calitatea finală a imaginii.

Pentru vizualizare, să considerăm un sistem de 64 de eșantioane per pixel. Cele mai comune opțiuni sunt:

- Eșantionare uniformă (Figura 16): se alege același eșantion de fiecare dată
- Eșantionare stratificată uniformă (Figura 16): se împarte fiecare pixel în subpixeli și se alege câte un eșantion pentru fiecare subpixel
- Eșantionare aleatoare/jittering (Figura 17): fiecare eșantion este ales aleator
- Eșantionare stratificată + jittering (Figura 17): se împarte fiecare pixel în subpixeli și se alege un eșantion aleator pentru fiecare subpixel, fără a se suprapune zonele de eșantionare.



(a) Eșantionare uniformă

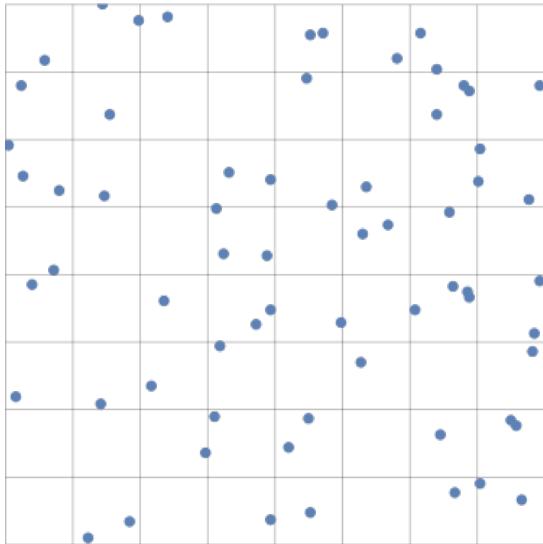


(b) Eșantionare stratificată

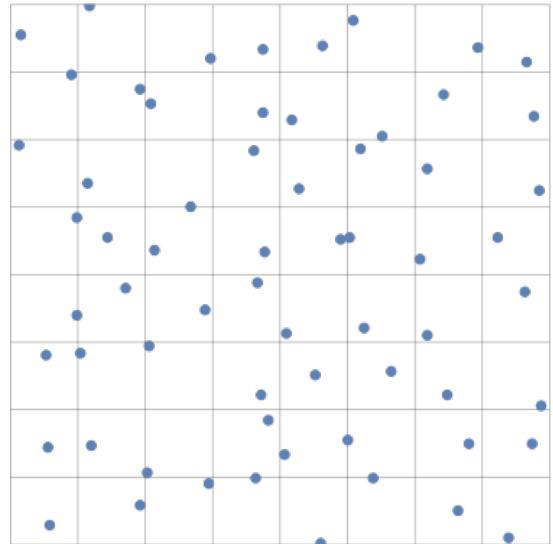
Figura 16: Eșantionare uniformă vs stratificată¹⁸

În mod evident, eșantionarea uniformă lasă multe de dorit. Aceasta produce artefacte de tip aliasing în jurul obiectelor (Figura 18). Eșantionarea stratificată (descrișă și în secțiunea 3.3.3) reușește să rezolve parțial această problemă, având date din mai multe zone ale pixelului. Eșantionarea aleatoare este altă soluție la problema aliasing-ului, însă introduce prea mult zgomot. Totuși, reușește să integreze mai bine semnalele de frecvență mare, cum ar fi cazul de randare a unei texturi sub unghi mare. Cea mai bună variantă este o combinație între cele două din urmă, care să ofere avantaje din ambele părți. Comparații calitative sunt reprezentate în Figurile 18 și 19.

¹⁸©<https://pbr-book.org/>. Accesat 20.06.2024.



(a) Eșantionare aleatoare



(b) Eșantionare stratificată + jittering

Figura 17: Eșantionare aleatoare vs stratificată + jittering¹⁸

3.5.2 Strategii de alegere a funcției de distribuție a reflectanței

Un model bun de material (mai ales PBR) face diferență între o randare fotorealistă și una care nu este. Chiar dacă algoritmul converge către soluția ecuației de transport, rezultatul poate fi unul nerealist dacă nu se folosește un model de materiale adecvat.

BRDF

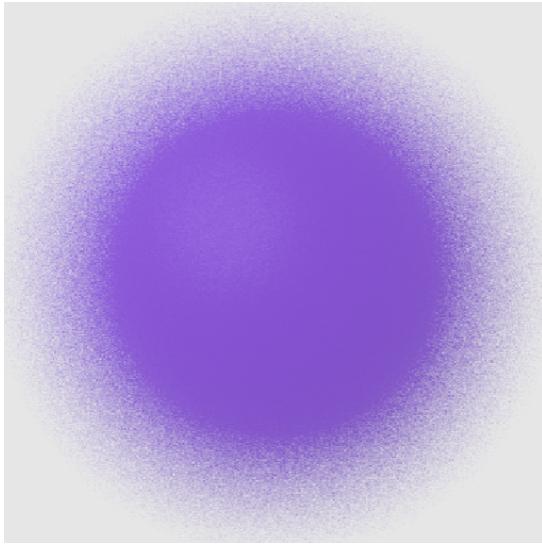
Definit prima oară de un model matematic general de către Nicodemus (1965) [26], BRDF-ul este o funcție care descrie câtă lumină este reflectată într-o anumită direcție. Am menționat-o de multe ori în cadrul acestei lucrări și am vorbit despre rolul ei în cadrul ecuației de randare (atât de importantă este), dar nu am explicitat cum se construiește. Să considerăm modelul simplificat în care nu se ține cont de lungimea de undă a luminii și nici de variabila de timp. În acest caz, tot ce ne mai rămâne de făcut pentru a defini BRDF sunt două concepte de radiometrie, pe care le vom defini informal aici:

Definiția 1. *Radianța L este măsura fluxului de lumină care trece printr-o suprafață unitară într-o direcție dată.*

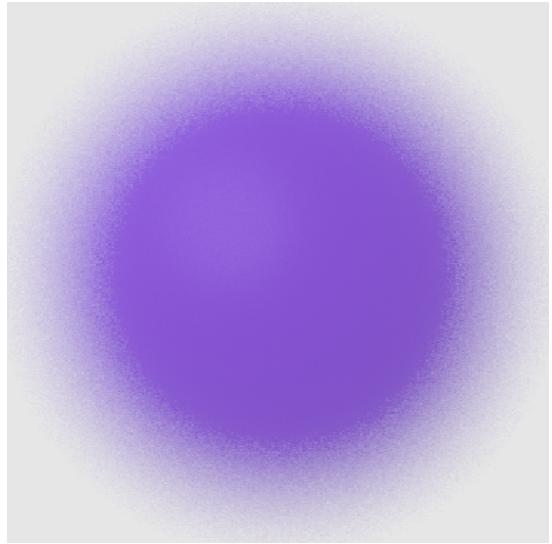
Definiția 2. *Iadianța E este măsura fluxului de lumină primit de o suprafață unitară (din toate direcțiile).*

Așadar, observăm că radianța depinde de direcția incidentă, în timp ce iradianța nu. O ilustrație a acestor concepte se poate vedea în Figura 20.

Putem acum să definim, informal, BRDF-ul:



(a) Eșantionare aleatoare



(b) Eșantionare stratificată

Figura 18: Comparație la 1 eșantion per pixel. Se pot observa artefacte de aliasing în varianta de jittering¹⁸

Definiția 3. *Functia de distribuție bidirecțională a reflectanței (BRDF) este raportul infinitesimal dintre radianța reflectată și iradianța incidentă, în funcție de direcția de incidentă și de cea de reflexie.*

Matematic, aceasta se scrie sub forma:

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{dL_r(\mathbf{x}, \omega_o)}{dE_i(\mathbf{x}, \omega_i)}, \quad (3.5.1)$$

Derivarea se face după direcția incidentă ω_i . Motivul pentru care valorile din raport sunt infinitezimale este faptul că ne interesează strict contribuția pe direcția incidentă - astfel, restrângem domeniul iradianței E_i la un unghi solid infinitezimal.

Plauzibilitate fizică

Scopul unui BRDF este de a modela împrăștierea luminii înapoi în mediul din care a venit. Contextul în care este folosit ignoră transmisia (refracția) și se concentrează strict pe reflectare. O funcție de distribuție care modelează această componentă se numește BTDF. Cel mai adesea în producție se folosește o generalizare care include ambele modele, numită BSDF, însă aceasta nu are o definiție precisă și de obicei se referă la combinarea a două modele separate de BRDF și BTDF. Totuși, nici aceste modele nu sunt suficiente pentru a descrie complet interacțiunile de lumină. De pildă, într-un fenomen de transmisie, lumina poate fi absorbită sau dispersată pe mai multe căi până când părăsește complet materialul. Astfel, o aparentă transmisie poate rezulta în ieșirea luminii în același mediu, dar din alt punct. Acest fenomen este cunoscut drept subsurface scattering (dispersie sub suprafață), iar o ilustrație



(a) Eșantionare stratificată



(b) Eșantionare aleatoare

Figura 19: Comparație la 1 eșantion per pixel. Varianta cu jittering minimizează artefactele dar adaugă zgomot¹⁸

se află în Figura 21. Totuși, pentru scopul lucrării nu vom modela acest fenomen, ci îl vom aproxima, aşa cum vom vedea în capitolul 5.

Pentru a fi considerat pentru uz PBR (physically based rendering), un BRDF este supus unor constrângeri care s-au standardizat în literatura de specialitate (Lafortune et. al. 1994 [21]):

1. **Pozitivitatea:** $f_r(\mathbf{x}, \omega_i, \omega_o) \geq 0$, pentru orice \mathbf{x} , ω_i și ω_o .
2. **Conservarea energiei:** $\int_{\Omega_+} f_r(\mathbf{x}, \omega_i, \omega_o)(\omega_i \cdot \mathbf{n}) d\omega_i \leq 1$, pentru orice \mathbf{x} , ω_i și ω_o .
3. **Reciprocitatea Helmholtz:** $f_r(\mathbf{x}, \omega_i, \omega_o) = f_r(\mathbf{x}, \omega_o, \omega_i)$, pentru orice \mathbf{x} , ω_i și ω_o .

Dacă modelul nu respectă cel puțin aceste constrângeri, algoritmul de Path Tracing poate să nu conveargă sau să producă rezultate nerealiste.

Există multe modele de distribuție în literatura de specialitate, cu grade diferite de complexitate și reprezentare a fenomenelor fizice. Printre primele modele apărute se numără modelul Lambertian (bazat pe modelul perfect difuz descris de Lambert în 1760 [23]), modelul Phong (1975) [27] împreună cu optimizarea Blinn-Phong (1977) [6], precum și modele speculare bazate pe microfațete ([34], [9]). Vom prezenta în continuare funcțiile de distribuție pentru aceste modele, fără a intra în prea multe detalii matematice de derivare. Pentru acestea, cititorul este îndrumat spre literatura de specialitate.

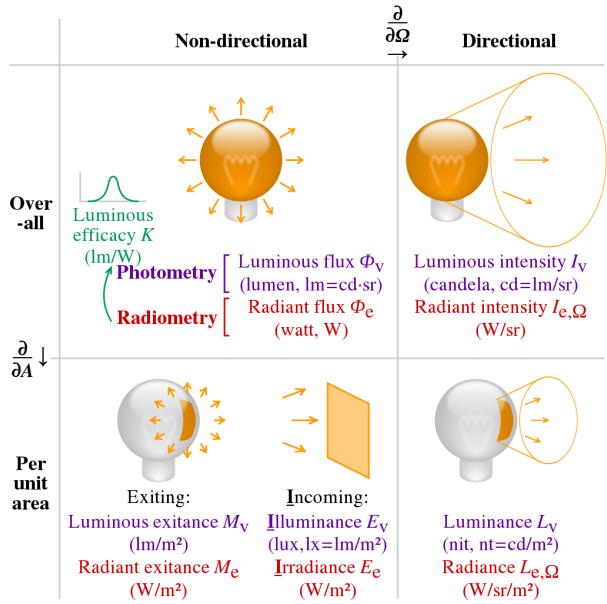


Figura 20: Ilustrație a diferitelor mărimi fizice din radiometrie⁴

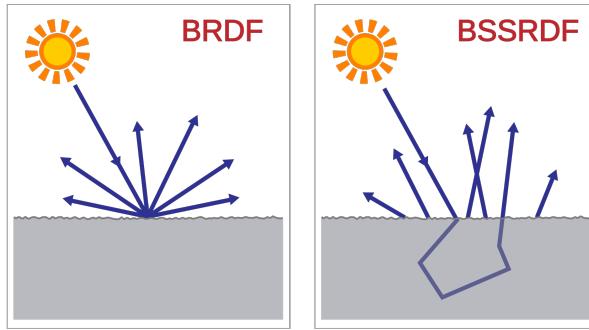


Figura 21: Ilustrație a fenomenului de subsurface scattering. Razele intră în obiect și ieș în alt punct⁴

Modelul Lambertian

Modelul Lambertian modelează reflectanța difuză a unui material. Dându-se o normală la suprafață \mathbf{n} și o direcție de incidentă ω_i , reflectanța Lambertiană se calculează ca produsul dintre albedo-ul materialului și cosinusul unghiului dintre normală și direcția de incidentă: $L = \rho \cdot (\omega_i \cdot \mathbf{n})$. Acest model trebuie totuși normalizat pentru a fi transformat într-un BRDF - o derivare folosind concepte de bază se găsește în [1]. BRDF-ul rezultat este:

$$f_{Lambert}(\mathbf{x}, \omega_i, \omega_o) = \frac{\rho}{\pi}, \quad (3.5.2)$$

unde ρ este culoarea materialului (albedo). Observăm că termenul de cosinus nu apare aici. Acesta este inclus implicit în ecuația de randare 3.3.13.

Definiția exactă pentru albedo seamănă destul de mult cu termenul integral din ecuația de transport a luminii:

$$\rho = \int_{\Omega_+} f_r(\mathbf{x}, \omega_i, \omega_o) (\omega_i \cdot \mathbf{n}) d\omega_i, \quad (3.5.3)$$

deoarece măsoară reflectanța unei suprafețe perfect difuze când este iluminată uniform de la toate direcțiile de lumină cu radianță unitară.

Modelul Phong

În adiție față de modelul Lambertian, modelul Phong adaugă un termen de reflexie speculară. Modelul inițial introdus de Phong în 1975 [27] nu este un BRDF propriu-zis (nu respectă constrângerile definite anterior), ci mai degrabă un model de iluminare. O adaptare a acestuia pentru a fi folosit ca BRDF a fost făcută de Lafortune și Willemens în 1994 [21].

Definiția BRDF-ului din publicație este:

$$f_{Phong}(\mathbf{x}, \omega_i, \omega_o) = k_d \frac{1}{\pi} + k_s \frac{n+2}{2\pi} (\omega_r \cdot \omega_o)^n, \quad (3.5.4)$$

unde k_d și k_s sunt coeficienții de reflexie difuză și speculară, definiți în intervalul $[0, 1]$, n este exponentul de reflexie speculară, iar ω_r este direcția de reflexie perfect speculară a luminii. Menționăm faptul că termenul de cos este restricționat de obicei să nu fie mai mic decât 0. Un exponent n mare va duce la o reflexie mai concentrată, în timp ce unul mic va duce la o reflexie mai difuză. Ideal, pentru a nu viola legea conservării a energiei, trebuie să nu reflectăm mai multă lumină decât am primit. Acest lucru se poate obține prin condiția $k_d + k_s \leq 1$.

Modelul Microfațetelor

Introdusă în anul 1982 de Cook și Torrance [9], teoria microfațetelor postulează ideea că majoritatea materialelor de diferite grade de luciu sunt compuse din microfațete (drepte sau modelate de curbe simple - vezi Figura 22). Acestea sunt orientate aleator și sunt descrise de două lucruri: distribuția orientării lor și profilul (i.e., cum sunt dispuse pe suprafață). Două suprafețe diferite pot avea aceeași distribuție dar profiluri diferite, ceea ce duce la aspecte diferite ale materialului (vezi Figura 24).

Asumptia de bază a modelului funcționează prin agregarea comportamentului luminii de la nivel microscopic (microfațete) la nivel macroscopic (suprafața modelată). Este același principiu pe care funcționează și monitoarele - fiecare pixel este compus din subpixeli care emit lumină de diferite culori (RGB), dar la distanță la care ne aflăm noi de monitor nu mai putem distinge acești subpixeli individuali, ci doar culoarea agregată a pixelului.

Un aspect foarte important care trebuie luat în considerare este profilul microfațetelor. Acestea pot fi reprezentate fizic de câmpuri de înălțimi, iar zonele dintre două vârfuri adiacente pot fi obstrucționate și private de lumină. Acest fenomen este mai departe categorizat în două:

- **Mascare (Masking):** microfațeta este obstrucționată din perspectiva observatorului
- **Umbrire (Shadowing):** microfațeta este obstrucționată din perspectiva sursei de lumină.

De asemenea, un alt fenomen extrem ce trebuie luat în considerare este cel al retroreflexiei. Acesta este manifestat drept o reflexie a luminii înapoi pe direcția incidentă (spre deosebire de o reflexie speculară), preponderent mai ales la unghiuri de incidentă mari. Totuși, există materiale care prezintă acest fenomen mai pronunțat, precum cele reflectorizante sau textile. În acest model teoretic, retroreflexia apare atunci când lumina este reflectată de mai multe ori între microfațete. O ilustrație intuitivă a tuturor acestor fenomene importante se poate vedea în Figura 23.

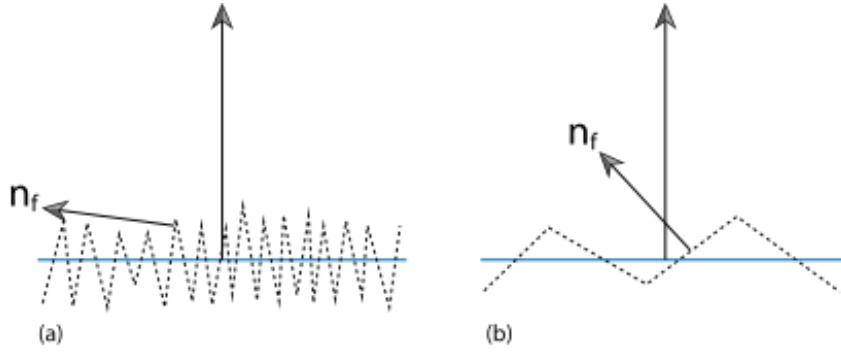


Figura 22: Profilul microfațetelor pentru materiale dure (stânga), respectiv licioase (dreapta)¹⁸

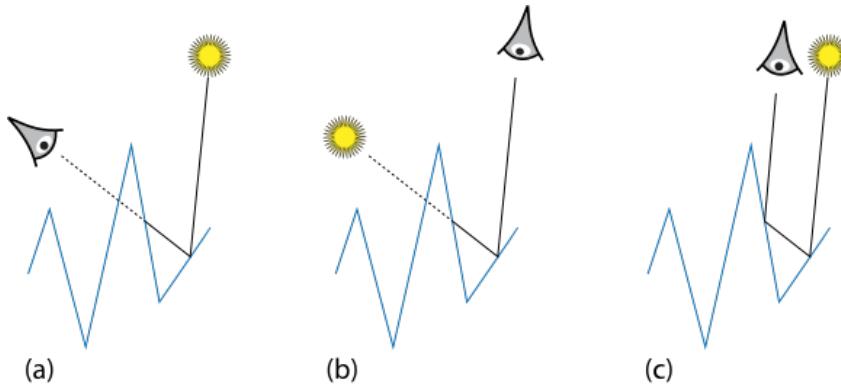


Figura 23: Efecte geometrice ale microfațetelor: mascare (a), umbră (b) și retroreflexie (c)¹⁸

O să concluzionăm prin a prezenta unul dintre primele și cele mai cunoscute BRDF-uri bazate pe teoria microfațetelor - descris de Torrance și Sparrow în 1967 [34]. În continuare, vom nota cu ω_i direcția de incidentă, cu ω_o direcția de ieșire, și cu ω_h normala microfațetei. Notăm funcția de distribuție a normalelor ca $D(\omega_h)$. De notat că această funcție este normalizată, adică:

$$\int_{\Omega} D(\omega_h) \cos \theta_h d\omega_h = 1. \quad (3.5.5)$$

Ecuția 3.5.5 este o condiție necesară pentru plauzibilitatea modelului, iar intuitiv reprezintă faptul că orice rază incidentă la suprafață de-a lungul macronormalei n va interseca microfațetele o singură dată.

Pentru a ține cont de efectele de mascare și umbră, ne trebuie o altă funcție de distribuție care să determine ce porțiune din microfațete sunt vizibile în funcție de direcția de incidentă

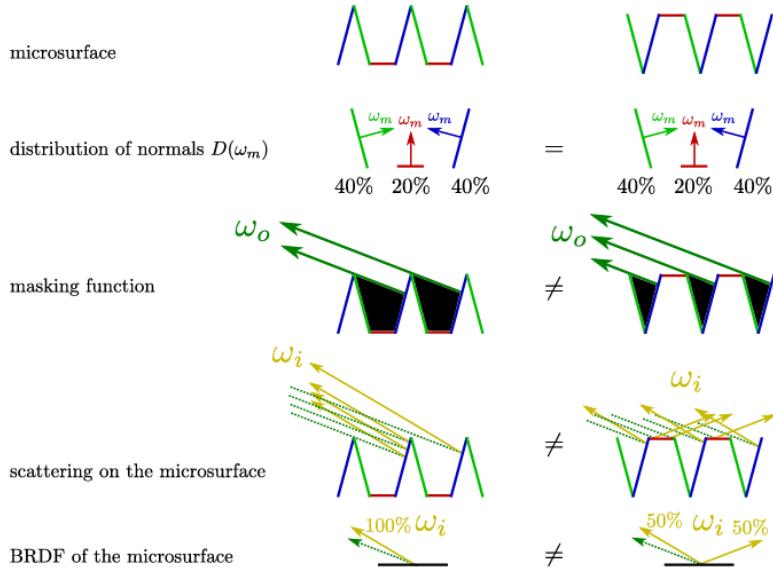


Figura 24: Diferite profiluri de microfațete pentru aceeași distribuție.
©Heinz [13]

(mascare) și de direcția de ieșire (umbră). Această distribuție poate fi modelată separat (pentru o singură direcție) de către funcția $G_1(\omega, \omega_h)$ a lui Smith, introdusă în 1967 [32]. Aceasta modelează proporția microfațetelor cu normală ω_h care sunt vizibile din direcția ω . O formă generală a acestei funcții este:

$$G_1(\omega, \omega_h) = \frac{1}{1 + \Lambda(\omega, \omega_h)}, \quad (3.5.6)$$

unde $\Lambda(\omega)$ este o funcție nespecificată. Totuși, ea trebuie aleasă pentru a satisface o constrângere de vizibilitate. Concret, aria dA a suprafeței văzută sub un unghi θ format între normală la suprafață n și direcția de observare ω trebuie să fie egală cu aria microfațetelor vizibile din aceeași direcție (vezi Figura 25):

$$\cos \theta = \int_{\Omega} G_1(\omega, \omega_h) \max(0, \cos \theta) D(\omega_h) d\omega_h, \quad (3.5.7)$$

unde adăugarea factorului $\max(0, \cos \theta)$ ne asigură că nu luăm în considerare microfațetele care sunt orientate în direcția opusă observatorului.

Pentru un model corect trebuie să ținem cont și de mascare și de umbră simultan. Cu alte cuvinte, trebuie să calculăm proporția microfațetelor vizibile din ambele direcții. Astfel, avem nevoie de o funcție $G_2(\omega_i, \omega_o, \omega_h)$ care să depindă de toate cele trei direcții. O tehnică comună (Walter et. al. [37]) este să se aproximeze această funcție prin produsul a doi termeni separabili Smith G_1 :

$$G_2(\omega_i, \omega_o, \omega_h) = G_1(\omega_i, \omega_h) G_1(\omega_o, \omega_h). \quad (3.5.8)$$

Deși nu este o soluție perfectă, căci nu ține cont de corelațiile dintre microfațete, această aproximare este suficient de bună pentru a fi folosită în practică.

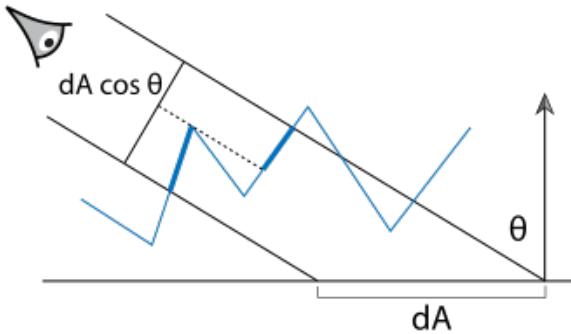


Figura 25: Ilustrație a vizibilității microfațetelor sub un unghi θ^{18}

În final, modelul Torrance-Sparrow are și o componentă speculară modelată de legea lui Fresnel, iar presupunerea modelului este că microfațetele sunt perfect speculare. BRDF-ul Torrance-Sparrow este definit ca:

$$f_{TS}(\mathbf{x}, \omega_i, \omega_o) = \frac{D(\omega_h)G_2(\omega_i, \omega_o, \omega_h)F(\omega_o)}{4(\omega_i \cdot \mathbf{n})(\omega_o \cdot \mathbf{n})}, \quad (3.5.9)$$

care poate fi simplificată sub presupunerea că vizibilitatea microfațetelor este independentă de orientarea acestora:

$$f_{TS}(\mathbf{x}, \omega_i, \omega_o) = \frac{D(\omega_h)G_2(\omega_i, \omega_o)F(\omega_o)}{4(\omega_i \cdot \mathbf{n})(\omega_o \cdot \mathbf{n})}. \quad (3.5.10)$$

Frumusețea acestui model generalizat este faptul că acceptă diferite distribuții de microfațete și funcții Fresnel, deci poate fi folosit atât pentru metale cât și pentru materiale dielectrice. În plus, modelul este destul de simplu pentru a putea fi folosit în practică, dar suficient de complex pentru a modela fenomene fizice sofisticate.

3.5.3 Strategii de eșantionare a direcțiilor de ieșire

Ce am prezentat mai devreme despre funcțiile de distribuție a reflectanței sunt expresiile acestora și modul în care se evaluatează. Concret, evaluarea BRDF-ului presupune cunoștința în prealabil a direcției de incidentă și a celei de ieșire. Într-un algoritm de Path Tracing, la fiecare pas al recursivității se cunoaște doar direcția incidentă (începând cu prima rază pornită de la observator), urmând să se determine, după intersecția cu o suprafață, direcția de ieșire (care va constitui direcția incidentă pentru următoarea rază s.a.m.d.). Această direcție de ieșire poate fi aleasă aleator sau conform unei strategii de eșantionare. În orice caz, fiind un algoritm Monte Carlo, se va ține cont de distribuția strategiei de eșantionare și se va pondera rezultatul contribuției reflectanței în consecință.

Așa cum am discutat în secțiunea 3.3.2, eșantionarea bazată pe importanță este crucială pentru a obține o convergență rapidă a algoritmului. Dacă BRDF-ul unui material modelează o suprafață speculară (poate chiar o oglindă perfectă), atunci distribuția direcțiilor de ieșire

va fi foarte îngustă, apropiindu-se de funcția delta Dirac:

$$\delta(x) = \begin{cases} \infty, & x = 0, \\ 0, & x \neq 0. \end{cases}, \quad (3.5.11)$$

$$\int_{-\infty}^{\infty} \delta(x) dx = 1.$$

Folosind o eșantionare uniformă (3.5.13), această distribuție va fi eșantionată neoptim, iar mareea majoritate a direcțiilor generate nu vor contribui la radianța observată, având probabilitate 0 (nefiind pe direcția de reflexie).

Pentru eșantionare, ne trebuie un generator de numere aleatoare. Fie ξ_1 și ξ_2 numere aleatoare din intervalul $[0, 1]$ (alese conform unei distribuții uniforme). Aceste numere vor fi folosite în strategiile descrise jos pentru a genera direcții de ieșire. De asemenea, facem precizarea că metodele prezentate mai jos presupun că ne aflăm în sistemul de coordonate tangent la suprafață $(\mathbf{t}, \mathbf{n}, \mathbf{b})$, cu normală \mathbf{n} orientată în sus, $\mathbf{n} = (0, 1, 0)$. Câteva metode presupun coordonatele sferice (θ, ϕ) , din care putem transforma în cele carteziene folosind formulele:

$$\begin{aligned} \mathbf{x} &= \sin \theta \cos \phi, \\ \mathbf{y} &= \cos \theta, \\ \mathbf{z} &= \sin \theta \sin \phi. \end{aligned} \quad (3.5.12)$$

Cea mai simplă strategie este eșantionarea emisferică uniformă. Aceasta presupune alegerea aleatoare a unei direcții de ieșire din emisfera superioară a punctului de intersecție. Considerând coordonatele sferice (θ, ϕ) , formulele de transformare, împreună cu probabilitatea eșantionului, sunt:

$$\begin{aligned} \theta &= \arccos \xi_1, \\ \phi &= 2\pi \xi_2, \\ pdf &= \frac{1}{2\pi}. \end{aligned} \quad (3.5.13)$$

Această strategie este foarte brută. Nu ține cont de distribuția BRDF-ului și poate să genereze multe direcții care nu contribuie la radianța observată (mai ales în cazul unui material specular - atunci rezultatele ar fi chiar greșite). O variantă mai bună (dar care tot nu ține cont de material) este eșantionarea ponderată de cosinusul unghiului de ieșire. Această strategie ponderează probabilitatea eșantionului cu $\cos \theta$ (unghiul făcut cu normala) și are la bază principiul cosinusului Lambertian. Formulele sunt:

$$\begin{aligned} \theta &= \arccos \sqrt{\xi_1}, \\ \phi &= 2\pi \xi_2, \\ pdf &= \frac{\cos \theta}{\pi}. \end{aligned} \quad (3.5.14)$$

Derivări ale ecuațiilor 3.5.13 și 3.5.14 se pot găsi în [3].

Am discutat în secțiunea 3.3.2 despre eșantionarea bazată pe importanță. În contextul de fată, aceasta înseamnă alegerea unei direcții de ieșire care să contribuie cât mai mult la radianță observată. Acest lucru este realizat prin modelarea unei distribuții de eșantionare care să aproximeze cât mai bine distribuția BRDF-ului.

În cazul unui material Lambertian, definit de ecuația 3.5.2, observăm că distribuția BRDF-ului are aceeași formă ca eșantionarea uniformă. Astfel, am fi tentați să concluzionăm că putem folosi ecuațiile 3.5.13 pentru a genera direcții de ieșire. Totuși, am omis factorul de cos care apare în ecuația de randare 3.3.13. Așadar, distribuția este mai degrabă una ponderată de cosinus, și e mult mai bine să folosim ecuațiile 3.5.14. De fapt, eșantionarea uniformă este inutilă în aproape toate cazurile, iar rezultatele ei dezamăgitoare se pot observa în Figura 39.

În cazul unui material Phong definit de BRDF-ul 3.5.4, o distribuție bună este greu de calculat. Autorii remarcă faptul că partea speculară nu poate fi integrată analitic și că eșantionarea acesteia se face printr-un proces Monte Carlo. Voi omisi detaliile matematice aici, pentru că nu prezintă interes; cititorul este îndrumat spre [21] pentru detalii.

Eșantionarea modelului Torrance-Sparrow este intuitivă. Deoarece factorul predominant este distribuția micronormalelor $D(\omega_h)$, eșantionarea se poate face fix pe această distribuție. În acest caz, se va reflecta raza de incidentă ω_i în jurul normalei eșantionate pentru a obține direcția de ieșire ω_o . Totuși, o metodă mult mai eficientă pentru unghiuri de incidentă mari o reprezintă eșantionarea normalelor vizibile. Reamintim că ecuația 3.5.7 trebuie să fie satisfăcută pentru a obține o distribuție corectă. Această ecuație descrie proporția microfațetelor vizibile sub un unghi θ față de normală la suprafață. Putem rearanja ecuația pentru a obține distribuția micronormalelor într-o direcție ω :

$$D_\omega(\omega_h) = \frac{G_1(\omega, \omega_h) \max(0, \cos \theta) D(\omega_h)}{\cos \theta_h}. \quad (3.5.15)$$

Un ultim aspect de luat în considerare în calculul probabilității de eșantionare este faptul că această distribuție reprezintă normalele în jurul micronormalei ω_h , în timp ce BRDF-ul ține cont de direcția de incidentă ω_i . Astfel, trebuie făcută o ajustare de schimbare de variabilă. Procesul este detaliat în [2]. Densitatea de probabilitate se modifică astfel cu un factor de $\frac{1}{4(\omega_o \cdot \omega_h)}$.

3.5.4 Next Event Estimation

Next Event Estimation (NEE) este o strategie de eșantionare bazată pe importanță multiplă (MIS) care presupune o combinație între eșantionarea BRDF-ului și a surselor de lumină. Aceasta reprezintă o îmbunătățire masivă a convergenței algoritmului de Path Tracing, deoarece multe dintre direcțiile generate de eșantionarea BRDF-ului în sine pot să nu ajungă să intersecteze o sursă de lumină și, implicit, nu vor contribui la radianță observată. În schimb, eșantionarea surselor de lumină la fiecare pas al recursivității ajută mult în acest sens. Astfel, eșantionarea luminii calculează iluminarea directă, în timp ce eșantionarea BRDF-ului calcu-

lează iluminarea indirectă. În final, aceste două contribuții sunt combinate probabilistic pentru a obține rezultatul final. Avantajele acestei metode se pot observa în Figura 26.

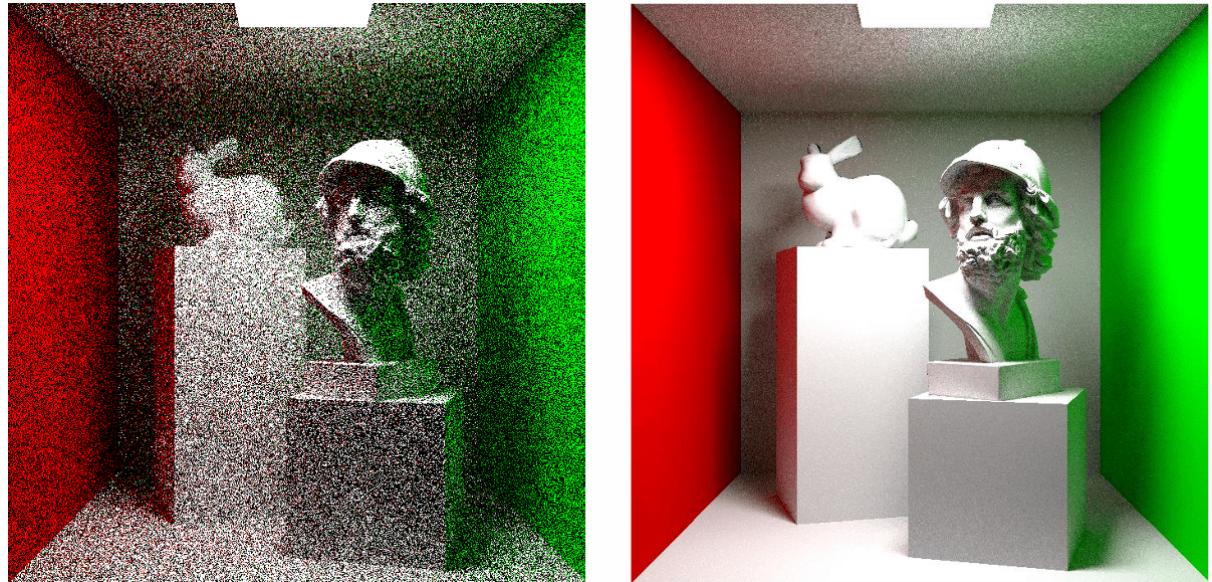


Figura 26: Comparatie intre esantionare a BRDF-ului (stanga) si NEE (dreapta)¹⁹

Idea de bază este de a combina cele două contribuții (directă și indirectă) cu o pondere în $[0, 1]$. Aceasta poate fi ori constantă (e.g., 0 sau 1) ori bazată pe o heuristică (precum cele studiate de Veach - balance sau power [36, 35]). Formula generală pentru a combina cele două contribuții este:

$$L(\mathbf{x}, \omega_i, \omega_o) = w \cdot f_r(\mathbf{x}, \omega_i, \omega_o) L_e(\mathbf{x}, \omega_i, \omega_o), \quad (3.5.16)$$

unde w este factorul de ponderare, f_r este BRDF-ul materialului și L_e este radianța emisă de sursa de lumină (normalizată cu probabilitatea de esantionare).

3.6 Accelerare Hardware

Până acum am discutat despre aspectele teoretice ale domeniului. În practică, algoritmii de tip Monte Carlo precum Path Tracing sunt foarte costisitori din punct de vedere computațional. Prin natura lor, aceștia sunt algoritmi paraleli și pot beneficia de accelerare hardware precum GPU-urile, care sunt fundamental concepute în jurul unei arhitecturi SIMD. În mod tradițional, totuși, algoritmii offline folosiți în motoarele grafice de producție (Blender Cycles,²⁰ Autodesk Arnold,²¹ Chaos V-Ray²²) sunt rulați pe CPU-uri, în parte deoarece este mult mai facil de

¹⁹©https://www.cg.tuwien.ac.at/sites/default/files/course/4411/attachments/08_next%20event%20estimation.pdf. Accesat 24.06.2024.

²⁰<https://www.cycles-renderer.org/>. Accesat 24.06.2024.

²¹<https://www.autodesk.co.uk/products/arnold/overview?term=1-YEAR&tab=subscription>. Accesat 24.06.2024.

²²<https://www.chaos.com/3d-rendering-software>. Accesat 24.06.2024.

dezvoltat și extins un program care rulează pe CPU. De pildă, un framework multi-vendor, OpenCL,²³ nu a primit prea mult suport în motoarele grafice de producție precum Cycles sau Arnold (dar tool-uri noi precum HIP²⁴ încep să primească suport²⁵). Povestea este puțin mai fericită în tabăra verde (Nvidia), care are suport mai răspândit pentru framework-ul CUDA. Totuși, într-un context de randare în timp real, CUDA este o opțiune ce aduce un overhead sesizabil.

În industria real-time (specific jocurilor video), accelerarea hardware este esențială. Totodată, se preferă să se folosească un API cât mai aproape de driver și cu o amprentă cât mai mică asupra sistemului. De aceea, se folosesc API-uri precum Vulkan sau DirectX 12 în defavoarea CUDA sau OpenCL. Acestea din urmă sunt mai degrabă folosite în aplicații de tip offline, unde timpul de randare nu este atât de critic. Totuși, nivelul jos de abstractizare al acestor API-uri face dezvoltarea mai dificilă și necesită un nivel mai înalt de expertiză - de aceea nu vedem în multe cazuri folosirea acestora la potențialul maxim.

3.6.1 Vulkan

Vulkan a fost dezvoltat cu scopul de a fi un succesor al OpenGL și de a utiliza eficient hardware-ul modern, oferind uneltele de optimizare necesare pentru a distribui sarcini pe mai multe fire de execuție.²⁶ A fost adoptat la scară largă în industria jocurilor video²⁷ și continuă să se dezvolte spre a fi un standard în industrie. De remarcat că acest API oferă suport pentru Ray Tracing accelerat hardware.²⁸

3.6.2 DirectX 12

DirectX 12 este cel mai popular API modern de grafică în industria jocurilor video,²⁹ dezvoltat de Microsoft pentru a fi folosit pe platformele Windows și Xbox.³⁰ Acesta oferă avantaje similare cu cele oferite de Vulkan, însă și acesta este un API de nivel scăzut, notoriu pentru dificultatea de a fi folosit la potențialul său maxim. Acesta oferă suport pentru Ray Tracing accelerat hardware prin intermediul extensiei DirectX Raytracing (DXR).³¹ Acesta a devenit standardul în industrie pentru randarea în timp real și este adoptat în majoritatea noilor lansări AAA de pe piață.³² Acest API, împreună cu extensia DXR, este folosit în acest proiect pentru

²³<https://www.khronos.org/api/opencl>. Accesat 24.06.2024.

²⁴<https://github.com/ROCm/HIP>. Accesat 24.06.2024.

²⁵<https://code.blender.org/2021/11/next-level-support-for-amd-gpus/>. Accesat 24.06.2024.

²⁶<https://www.vulkan.org/>. Accesat 24.06.2024.

²⁷https://www.pcgamingwiki.com/wiki/List_of_Vulkan_games. Accesat 24.06.2024.

²⁸https://docs.vulkan.org/guide/latest/extensions/ray_tracing.html. Accesat 24.06.2024.

²⁹https://www.pcgamingwiki.com/wiki/List_of_Direct3D_12_games. Accesat 24.06.2024.

³⁰<https://learn.microsoft.com/en-us/windows/win32/direct3d12/what-is-directx-12->. Accesat 24.06.2024.

³¹<https://devblogs.microsoft.com/directx/announcing-microsoft-directx-raytracing/>. Accesat 24.06.2024.

³²https://www.pcgamingwiki.com/wiki/List_of_games_that_support_ray_tracing. Accesat 24.06.2024.

a accelera randarea Path Tracing pe GPU (vezi capitolul 5).

4 SOLUȚIA PROPUȘĂ

Proiectul de față dezvoltă un motor de randare bazat pe Ray Tracing Whitted și Path Tracing, cu suport pentru accelerare hardware folosind API-ul DirectX 12. Acesta este scris în C++20, iar algoritmii de Ray Tracing sunt implementați în cadrul HLSL. Proiectul a fost dezvoltat pornind de la un exemplu oferit public de Microsoft [25], care oferă un schelet bun de DirectX 12 și este un punct de plecare ce oferă majoritatea boilerplate-ului necesar pentru a începe dezvoltarea. A fost modificat acest cod suport pentru a extinde funcționalitatea cu încă un set de shadere pentru Path Tracing și a suporta noi tipuri de suprafete (e.g., unghi solid - Figura 27). De asemenea, acesta a fost extins cu multe alte funcționalități, pe care le vom detalia în acest capitol.



Figura 27: Unghi solid reprezentat drept câmp de distanță¹

Pe scurt, am implementat, pe lângă codul din schelet:

- Un meniu de setări pentru a controla parametrii de randare și a manipula scenă
- Suport pentru reîncărcarea aplicației în timp real (la comandă sau automat la modificarea unor setări de bază)
- Un sistem de mișcare și rotire a camerei, folosind mouse-ul și tastele WASD

¹Formula câmpului a fost preluată de la adresa <https://www.shadertoy.com/view/Xds3zN>. Accesat 24.06.2024.

- Un sistem de materiale physically based, bazat pe modelul Disney principled BSDF [7, 14].
- Un algoritm de Path Tracing care folosește acest sistem de materiale. Particularități:
 - Algoritmul este accelerat hardware folosind API-ul DXR
 - Suport pentru reflexii, refracții, umbre, iluminare globală și multiple surse de lumină
 - Suport pentru configurarea parametrilor (a se vedea Figura 31)
 - Suport pentru eșantionare bazată pe importanță multiplă și Next Event Estimation
 - Metodă de ruletă rusească pentru terminarea prematură (unbiased) a recursivității
 - Suport pentru acumulare progresivă.

4.1 Tehnici de accelerare a convergenței

Algoritmul de bază prezentat în pseudocodul 1 a fost augmentat cu mai multe tehnici de accelerare a convergenței, majoritatea fiind descrise în secțiunea 3.5. Multe dintre acestea sunt configurabile din meniu, iar unele sunt activate implicit.

- **Next Event Estimation** - la fiecare pas se selectează o lumină (sau toate) din scenă și se eșantionează cu importanță multiplă contribuția directă și indirectă. În formula 3.5.16 s-a folosit pentru calculul ponderii w euristica puterii 2:

$$w = \frac{pdf_L^2}{pdf_f^2 + pdf_L^2}. \quad (4.1.1)$$

Această optimizare se efectuează implicit și nu este configurabilă.

- **Eșantionare a unei singure lumi** - pentru ca algoritmul să nu piardă performanță dramatic în cazul în care există multe surse de lumină, se poate alege aleator o singură sursă de lumină la fiecare pas pe care să se aplique NEE. Rezultatul final este ponderat apoi cu inversul probabilității de eșantionare a sursei de lumină (în acest caz, N - numărul de surse de lumină). Această optimizare poate fi dezactivată din meniu (este activată implicit).
- **Eșantionare bazată pe importanță** - alegerea unei noi direcții de ieșire pentru pasul recursiv se face folosind o eșantionare concepută pentru a aproxima distribuția de probabilitate a normalelor microfațetelor (vezi secțiunile 3.3.2, 3.5.2 și 4.3). Această optimizare este activată implicit și poate fi schimbată cu alte două tehnici de eșantionare: uniformă sferică, respectiv ponderată de cosinusul unghiului de ieșire.
- **Ruleta rusească** - o tehnică unbiased care termină prematur recursivitatea pentru a evita pierderea de timp în cazul în care contribuția pe o cale este foarte mică. Probabilitatea de eliminare este invers proporțională cu contribuția acumulată până la acel pas. Pentru a nu introduce bias din cauza energiei pierdute în terminarea prematură, contribuția acumulată se ponderează la fiecare pas cu probabilitatea de terminare. Această optimizare este activată implicit și poate fi configurată prin modificarea adâncimii de la care se aplică.

4.2 Suport pentru suprafete implice și analitice

Motorul grafic suportă, pe lângă suprafetele tradiționale definite prin primitive geometrice (e.g., triunghiuri), și suprafete definite implicit (vezi secțiunea 3.2) sau analitic. Acestea se încadrează în AABB-uri care se folosesc în structura de accelerare BVH oferită de DXR. Astfel, API-ul se ocupă de intersecția cu aceste încadrări, iar mai departe se utilizează rutine specifice pentru a calcula intersecția precisă cu suprafața definită. Pentru cele implice, se folosesc funcții de distanță cu semn și se aplică Ray Marching local (vezi secțiunea 3.2). Pentru suprafetele analitice, precum sferele și paralelipipedul, se folosesc formule analitice pentru a calcula intersecția. Rutinele de intersecție pentru suprafetele analitice sunt mult mai rapide decât cele de Ray Marching, dar nu oferă la fel de multă flexibilitate creativă.

Ofer credite lui Inigo Quilez [28] și Microsoft [25] pentru formulele care definesc aceste suprafete.

4.3 Sistemul de materiale

Sistemul de materiale este un BSDF compus ce combină mai mulți lobi definiți de BRDF-uri și BTDF-uri. Modelul particular este denumit Disney Principled BSDF și a fost introdus de Burley în 2012 [7] ca un BRDF și extins în 2015 [14] cu o serie de îmbunătățiri, precum un lob de transmisie. Acesta este modelul care a fost folosit în producția cinematografică, începând cu filmul Wreck-It Ralph.

Modelul este principiat în sensul că deviază în mod subtil de la modelele fizice pentru a oferi un control și o intuiție mai bună artistului. Astfel, acesta expune o interfață simplă, cu parametri normalizați (în $[0, 1]$) care să se coreleză cu proprietățile fizice ale materialelor, uneori alegând să nu aibă corespondență perfectă cu realitatea ci mai degrabă un impact artistic. O ilustrație a parametrilor disponibili în modelul din 2012 se poate vedea în Figura 53.

În lucrarea de față am ales să implementăm modelul din 2015, cu câteva modificări care să ușureze dezvoltarea. Câteva implementări de referință sunt [4, 31, 20]. Menționăm faptul că distribuțiile prezentate în continuare vor folosi calcule într-un sistem de coordonate local ($\mathbf{t}, \mathbf{b}, \mathbf{n}$), pe când în implementare se folosește un sistem de coordonate local ($\mathbf{t}, \mathbf{n}, \mathbf{b}$), unde \mathbf{t} este tangenta, \mathbf{n} este normala și \mathbf{b} este bitangenta la suprafață.

Lobul difuz

Reprezintă un model empiric ce încearcă să modeleze efectele observate de Burley în setul de date MERL [24]: la unghiuri de incidentă mari, materialele lucioase au reflectanță scăzută, în timp ce materialele dure au reflectanță crescută față de unghiul normal de incidentă. Aceste observații sunt prezise de reflexia Fresnel, motiv pentru care modelul propus ține cont de

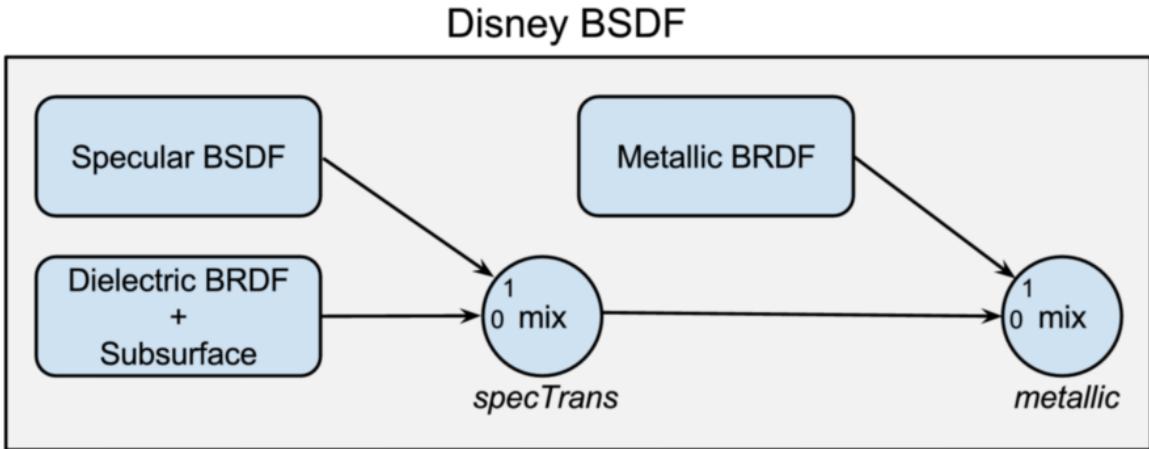


Figura 28: Componerea BSDF-ului Disney Principled [14]

aceasta:

$$f_d = \frac{albedo}{\pi} F_D(\omega_i) F_D(\omega_o), \quad (4.3.1)$$

unde $F_D(\omega, \omega_h)$ este o modificare a aproximăției lui Schlick [30] a reflexiei Fresnel pentru direcția de incidentă ω și micronormala ω_h :

$$\begin{aligned} F_D(\omega) &= 1 + (F_{D90} - 1)(1 - |\mathbf{n} \cdot \omega|)^5, \\ F_{D90} &= 0.5 + 2 \cdot roughness \cdot (\omega_o \cdot \omega_h)^2. \end{aligned} \quad (4.3.2)$$

Această modificare ignoră indicele de refracție al materialului și presupune că nu există pierderi de energie incidentă la difuzie, ceea ce permite BRDF-ului să depindă direct de o culoare *albedo* specificată. De asemenea, termenul F_{D90} este ales astfel încât contribuția de tip retroreflexie să fie modulată, în funcție de *roughness*, între 0.5 (*roughness* = 0) și 2.5 (*roughness* = 1). Această modificare este strict empirică și, conform lui Burley, are un feedback pozitiv din partea artiștilor.

Versiunea din 2015 [14] transformă f_d într-un BSDF cu componentă de dispersie sub suprafață, care este simulată prin Path Tracing Volumetric (introdus prima oară de Lafourcade și Willem [22]), dar acest algoritm este prea costisitor pentru randare în timp real. Așadar, rămânem la BRDF-ul 4.3.1 și îl complementăm cu o aproximare a acestui fenomen bazată pe legea Lommel-Seeliger,² aşa cum este descris în [4]:

$$f_{ss} = \frac{1.25 \cdot albedo}{\pi} \left(F_{SS}(\omega_i) F_{SS}(\omega_o) \left(\frac{1}{|\mathbf{n} \cdot \omega_i| + |\mathbf{n} \cdot \omega_o|} - 0.5 \right) + 0.5 \right), \quad (4.3.3)$$

unde F_{SS} este aceeași aproximare folosită pentru F_D (4.3.2), dar cu o remapare $F_{SS90} = roughness \cdot (\omega_o \cdot \omega_h)^2$.

Într-un final, cele două componente sunt interpolate liniar de parametrul *subsurface*:

$$f_d = (1 - subsurface) \cdot f_d + subsurface \cdot f_{ss}. \quad (4.3.4)$$

²[https://phys.libretexts.org/Bookshelves/Astronomy__Cosmology/Planetary_Photometry_\(Tatum_and_Fairbairn\)/03%3A_A_Brief_History_of_the_Lommel-Seeliger_Law](https://phys.libretexts.org/Bookshelves/Astronomy__Cosmology/Planetary_Photometry_(Tatum_and_Fairbairn)/03%3A_A_Brief_History_of_the_Lommel-Seeliger_Law). Accesat 24.06.2024.

Componenta de luciu (sheen)

Materialele textile precum catifeaua sau mătasea au un luciu pronunțat la unghiuri mari de incidentă, cauzat de retroreflexie. Pentru a compensa pentru acest efect, Burley a adăugat o componentă aditivă de luciu la BRDF-ul difuz (aceasta nu este evaluată ca un lob separat). La fel cum a procedat și la difuz, Burley a folosit un termen Fresnel modificat pentru a accentua retroreflexia:

$$\begin{aligned} f_{sheen} &= sheen \cdot C_{sheen}(1 - |\omega \cdot \omega_h|^5), \\ C_{sheen} &= (1 - sheenTint) + sheenTint \cdot C_{tint}, \\ C_{tint} &= albedo/luminance(albedo), \end{aligned} \quad (4.3.5)$$

unde $sheenTint$ este un parametru care controlează contribuția culorii materialului la luciu. Reflexia speculară a unui dielectric este totuși acromatică, însă această componentă modelează structura complexă a textilei și deci este lăsată la latitudinea artistului cât de colorată să fie.

Lobul de reflexie speculară

Reprezintă este un model BRDF de microfațete de tipul Torrance-Sparrow (vezi ecuația 3.5.10):

$$f_{sp} = \frac{D(\omega_h)G(\omega_i, \omega_o)F(\omega_o)}{4|\omega_i \cdot \mathbf{n}| |\omega_o \cdot \mathbf{n}|}, \quad (4.3.6)$$

Pentru distribuția microfațelor D , Burley folosește distribuția anizotropică GGX [37]:

$$D_{specular} = \frac{1}{\pi \alpha_x \alpha_y \left(\frac{h_x^2}{\alpha_x^2} + \frac{h_y^2}{\alpha_y^2} + h_z^2 \right)^2}, \quad (4.3.7)$$

unde $\mathbf{h} = (h.x, h.y, h.z)$ este normala microfațetei proiectată în planul de referință tangent, iar α_x și α_y sunt parametri de modelare a anizotropiei. Burley i-a definit cu o mapare în funcție de parametrii materialului *roughness* și *anisotropic*:

$$\begin{aligned} aspect &= \sqrt{1 - 0.9 \cdot anisotropic}, \\ \alpha_x &= \max(0.0001, roughness^2 / aspect), \\ \alpha_y &= \max(0.0001, roughness^2 \cdot aspect). \end{aligned} \quad (4.3.8)$$

Această mapare asigură un raport de aspect maxim de 10:1 și nu lasă rugozitatea să ajungă la 0 (în fel material este invizibil).

Pentru funcția de mascare și umbrire G , Burley folosește varianta separabilă cu funcția Smith G1 (vezi ecuația 3.5.6) derivată pentru distribuția GGX, care implică următoarea expresie a funcției Λ :

$$\Lambda(\omega, \omega_h) = \frac{-1 + \sqrt{1 + \frac{(w'.x \cdot \alpha_x)^2 + (w'.y \cdot \alpha_y)^2}{w'.z^2}}}{2}, \quad (4.3.9)$$

unde $w' = (w'.x, w'.y, w'.z)$ este proiecția lui ω pe planul tangent.

Pentru termenul specular F , trebuie să modelăm și din perspectiva materialelor dielectrice dar și a celor conductoare. Pentru partea conductoare o să folosim aproximarea lui Schlick, dar pentru parte dielectrică Burley a propus folosirea ecuației exacte a lui Fresnel, deoarece aproximarea introducea erori semnificative pentru indecși de refracție foarte mici. În contrast, ecuația lui Fresnel pentru conductoare este mai complexă și depinde de parametri neintuitivi, de aceea s-a decis să se rămână la aproximarea lui Schlick folosită în 2012.

Pentru completitudine, definim ecuația lui Fresnel pentru dielectrice:

$$F_{dielectric}(\theta_i, \theta_t, \eta) = 0.5 * \left(\left(\frac{\cos \theta_i - \eta \cos \theta_t}{\cos \theta_i + \eta \cos \theta_t} \right)^2 + \left(\frac{\cos \theta_t - \eta \cos \theta_i}{\cos \theta_t + \eta \cos \theta_i} \right)^2 \right), \quad (4.3.10)$$

unde η este indicele de refracție relativ al materialului, iar θ_i și θ_t sunt unghiurile de incidentă și transmisie, respectiv. Trebuie avută grijă la cazul de reflexie totală, când legea lui Snell nu are soluție reală pentru θ_t , i.e., $\sin \theta_t > 1$. În acest caz, se consideră $F_{dielectric} = 1$.

Aproximarea lui Schlick se definește astfel:

$$F_{Schlick}(\theta, n_1, n_2) = F_0 + (1 - F_0)(1 - \cos \theta)^5, \quad (4.3.11)$$

unde $F_0 = \left(\frac{n_1 - n_2}{n_1 + n_2} \right)^2$ este reflectanța la unghiul normal de incidentă. Remarcăm faptul că în această aproximare θ se măsoară relativ la micronormala ω_h .

Acum că avem definiții termenii necesari, putem defini funcția de mixaj a reflexiei speculare pentru dielectrice și conductoare:

$$F_{mix} = (1 - metallic) \cdot F_{dielectric} + metallic \cdot F_{Schlick}, \quad (4.3.12)$$

Reflectanța obținută este acromatică, ceea ce este reprezentativ pentru dielectrice, însă conductoarele au o reflexie speculară influențată de culoarea materialului. Totuși, factorul F_{mix} este acromatic, fiind scalar. Așadar, pentru a introduce culoarea în reflexie, Burley a propus un amestec similar celui folosit pentru componența de luciu 4.3.5:

$$specularColor = (1 - metallic) \cdot f_{sheen} \cdot F_0 + metallic \cdot albedo. \quad (4.3.13)$$

În concluzie, termenul Fresnel al reflexiei speculare este:

$$F_{specular} = F_{mix} + (1 - F_{mix}) \cdot specularColor, \quad (4.3.14)$$

unde termenul liber F_{mix} este de fapt o culoare cu toate componentele egale cu valoarea F_{mix} .

Lobul de transmisie speculară

Acesta este un lob modificat cu multiple corectii care să țină cont de schimbările care au loc la transmisie. Vom enunța formula lui Burley, cu observația că se omite un factor de $\frac{1}{\eta^2}$ care să

anuleze creșterea luminozității excesive când observatorul este plasat într-un mediu cu indice de refracție diferit de al aerului (e.g. în apă):

$$f_t = \frac{D(\omega_{ht})G(\omega_i, \omega_o)(1 - F(\omega_i))|\omega_i \cdot \omega_{ht}| |\omega_o \cdot \omega_{ht}| \cdot \eta^2}{((\omega_o \cdot \omega_{ht}) + \eta(\omega_i \cdot \omega_{ht}))|\omega_i \cdot \mathbf{n}| |\omega_o \cdot \mathbf{n}|}, \quad (4.3.15)$$

unde ω_{ht} este normala microfațetelor refractată:

$$\omega_{ht} = -\frac{\omega_i + \eta\omega_o}{\|\omega_i + \eta\omega_o\|}. \quad (4.3.16)$$

Lobul de clearcoat

Este tot un BRDF de tipul Torrance-Sparrow (4.3.6). Acesta modelează un strat subțire de material transparent și lucios, cu index de refracție 1.5 (reprezentativ poliuretanului), care acoperă materialul principal. Totuși, formularea acestui model nu este fizică, așa că funcțiile de distribuție și mascare/umbră sunt alese ad-hoc.

Pentru termenul de distribuție D , Burley a ales următoarea distribuție, care are coada mai largă decât cea folosită la lobul de reflexie speculară:

$$D_{clearcoat} = \frac{a^2 - 1}{\pi \log(a^2) \cdot (1 + (a^2 - 1)(\omega_h \cdot \mathbf{n})^2)}, \quad (4.3.17)$$

unde a este o măsură de rugozitate care remapează parametrul $clearcoatGloss$ invers proporțional:

$$a = (1 - clearcoatGloss) \cdot 0.1 + clearcoatGloss \cdot 0.001. \quad (4.3.18)$$

Lobul folosește același termen de mascare și umbră ca și lobul de reflexie speculară (vezi 3.5.6 și 4.3.9), însă termenii α_x și α_y sunt fixați la valoarea empirică 0.25.

5 DETALII DE IMPLEMENTARE

5.1 Folosirea API-ului DXR

API-ul DXR este unul de nivel scăzut, care necesită o înțelegere profundă a arhitecturii GPU și a conceptelor de bază ale Ray Tracing-ului. Am ales să construiesc aplicația având acest API la bază din considerente de performanță și de suport hardware pentru accelerarea algoritmilor de Ray Tracing. Fără a intra în prea multe detalii, pașii efectuați pentru a pregăti pipeline-ul de Ray Tracing¹ sunt, în ordine:

1. Detectarea suportului pentru DXR și crearea obiectului de interfață
2. Construirea structurilor de accelerare BLAS și TLAS (care au la bază arbori BVH). Mentionăm faptul că structura BLAS (bottom) este construită din geometria de randat, conținând date despre vârfuri. Un BLAS poate conține mai multe obiecte. Spre deosebire, TLAS (top) este o structură de nivel superior care conține referințe la BLAS-uri și le asociază matrice de transformare. O ilustrație a acestei organizări se poate vedea în Figura 29.
3. Scrierea shaderelor necesare:
 - Ray Generation - apelat pentru fiecare pixel din imagine (aici se generează traiectoriile razelor care pornesc din cameră)
 - Hit - apelat atunci când are loc o intersecție cu geometria (aici se fac calculele de iluminare)
 - Miss - apelat atunci când nu are loc nicio intersecție (aici se oprește recursivitatea)
 - (optional) Intersection - apelat la traversarea structurilor de accelerare (aici se fac calculele custom de intersecție cu suprafete implicate sau analitice de exemplu)
4. Crearea unui tabel de legare de shading (SBT), care asociază diferite tipuri de geometrie cu diferitele tipuri de shader (aici se leagă spre exemplu shaderele de tip Hit cu geometria de tip triunghi).

Framework-ul oferit de Microsoft [25] oferă o structură minimală pentru setup-ul unui proiect folosind DXR, precum crearea resurselor Direct3D, gestionarea stărilor acestora și efectuarea pașilor descriși mai sus. Mare parte din acest cod nu a fost modificat, cu excepția adăugării legăturilor pentru shaderele de tip Ray Generation. De aceea, nu voi include cod legat de schelet în această secțiune, ci mă voi concentra pe implementarea efectivă a algoritmului de Path Tracing din cadrul HLSL.

¹<https://developer.nvidia.com/rtx/raytracing/dxr/dx12-raytracing-tutorial-part-2>. Accesat 24.06.2024.

²© <https://developer.nvidia.com/rtx/raytracing/dxr/dx12-raytracing-tutorial-part-1>. Accesat 24.06.2024.

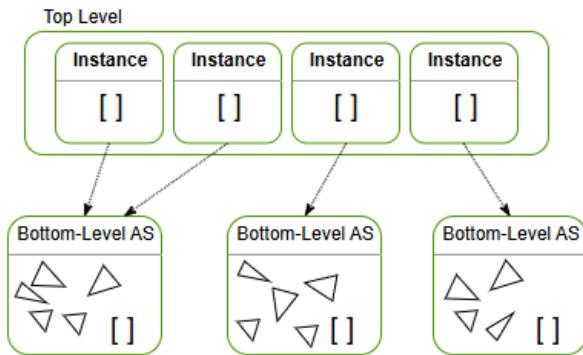


Figura 29: Organizarea structurilor de accelerare în DXR²

5.2 Meniuul

Meniul aplicației a fost creat folosind ImGui,³ o bibliotecă open-source ce oferă elementele de construcție pentru interfețe grafice imediate (care sunt integrate în bucla de randare a aplicației). Pentru integrarea cu DirectX 12, ImGui oferă un API minimal și ușor de folosit, însă am întâmpinat dificultăți. Conform documentației, trebuie folosite simultan integrarea cu Win32 și cu DirectX 12. Operațiile de initializare și de curățare trebuie să se facă simultan. Totuși, dintr-un motiv necunoscut, la recrearea resurselor de Direct3D, (împreună cu cele două contextele ale ImGui), aplicația dă crash. Soluția găsită de mine a fost să dau shutdown doar la contextul de Win32, iar după reinitializarea acestuia să repet aceeași secvență de restart și pentru contextul de DirectX 12. Soluția se află în listarea 5.1.

```
1 // Initialize imgui.
2 IMGUI_CHECKVERSION();
3 ImGui::CreateContext();
4 ImGuiIO& io = ImGui::GetIO();
5 io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;
6
7 ImGui_ImplWin32_Init(Win32Application::GetHwnd());
8 D3D12_CPU_DESCRIPTOR_HANDLE cpuHandle;
9 UINT descriptorHeapIndex = AllocateDescriptor(&cpuHandle);
10
11 ImGui_ImplDX12_Shutdown();
12 ImGui_ImplDX12_Init(m_deviceResources->GetD3DDevice(),
13     FrameCount,
14     DXGI_FORMAT_R8G8B8A8_UNORM,
15     m_descriptorHeap.Get(),
16     cpuHandle,
17     CD3DX12_GPU_DESCRIPTOR_HANDLE(m_descriptorHeap->GetGPUDescriptorHandleForHeapStart(),
18         descriptorHeapIndex, m_descriptorSize));
19
20 ImGui::StyleColorsDark();
```

Listarea 5.1: Reinitializare ImGu. Bazat pe documentatia oficială³

Meniul este modular. El poate fi redimensionat, mutat în cadrul ferestrei sau minimizat. Secțiunile sunt grupate astfel:

³<https://github.com/ocornut/imgui>. Accesat 24.06.2024.

- **Header:** aici sunt afisate informații despre performanța aplicației, precum FPS-ul, dar și timpul scurs și poziția camerei. Ilustrație în figura Figura 30.
- **Controls:** aici sunt afisate controalele pentru aplicație. Ilustrație în figura Figura 30.
- **Renderer:** aici sunt afisate setările de configurare ale algoritmului de randare, precum numărul de eșantioane, adâncimea maximă de recursivitate, etc. Aici se pot configura și tehnicele de accelerare a convergenței definite în secțiunea 4.1. Ilustrație în figura Figura 31.
- **Scene:** aici se pot configura luminile din scenă, precum culoarea de emisie, tipul (de suprafață sau direcționale), intensitatea, direcția sau poziția. Ilustrație în figura Figura 32.

Codul utilizat pentru compunerea acestui meniu se află în listarea B.1.

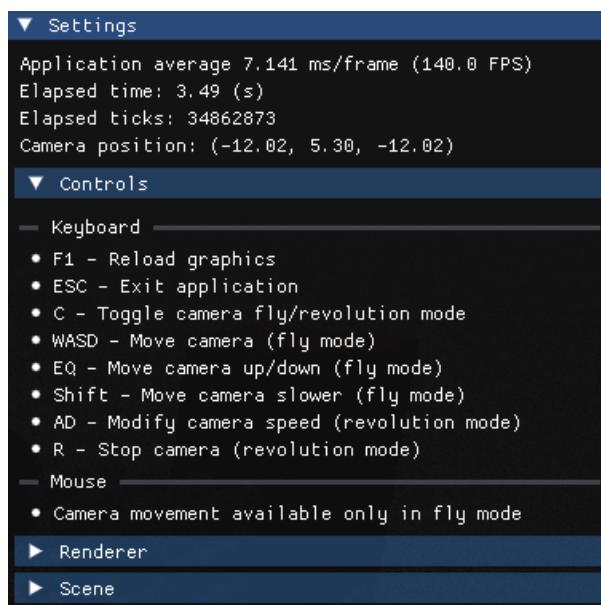


Figura 30: Meniul aplicației, secțiunea de statistici și controale

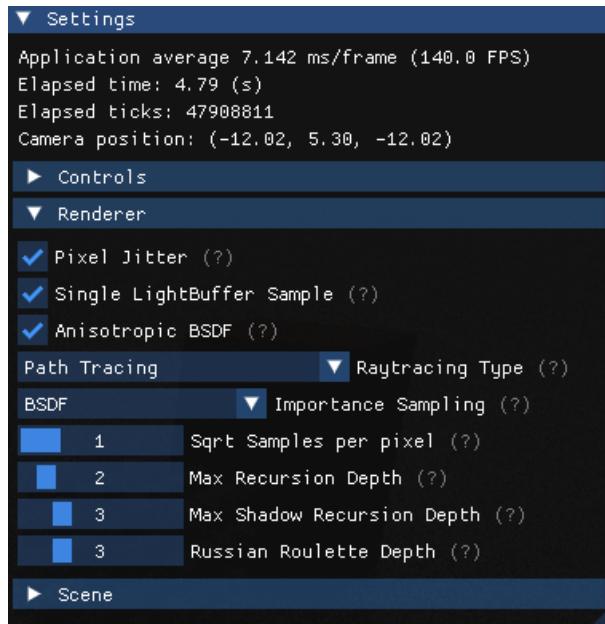


Figura 31: Meniul aplicației, secțiunea de configurare a algoritmului de randare

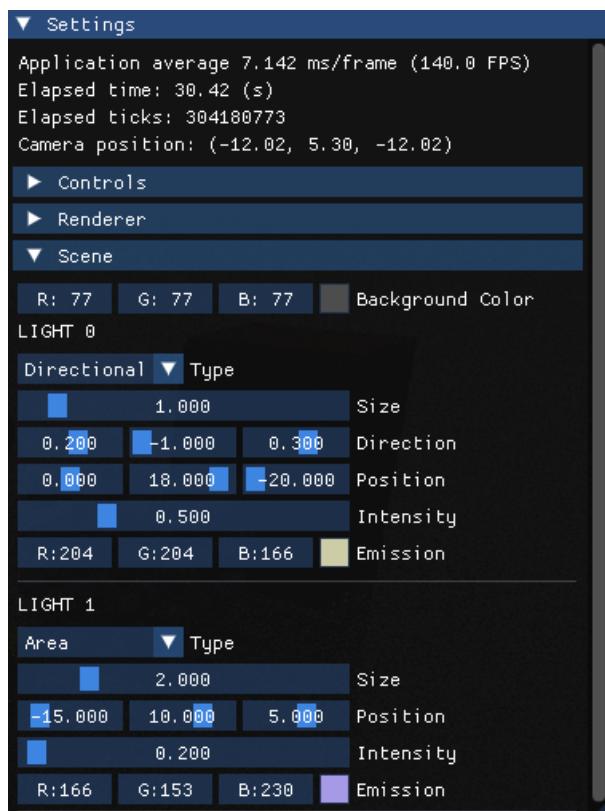


Figura 32: Meniul aplicației, secțiunea de configurare a luminilor din scenă

5.3 Generator de numere aleatoare

Natura algoritmului recursiv de tip Monte Carlo necesită un generator de numere aleatoare care să fie rapid și să ofere o distribuție bună atât în lățime (deoarece trebuie folosit de toti

pixelii simultan) cât și în adâncime (deoarece trebuie generate mai multe numere aleatoare per pixel). Am ales să folosesc un generator PCG descris în [16] care îndeplinește aceste cerințe (este și prng și hash). Codul generatorului se află în listarea 5.2.

```

1 // PRNG
2 // https://www.reedbeta.com/blog/hash-functions-for-gpu-rendering/
3 uint rand_pcg(inout uint rng_state)
4 {
5     uint state = rng_state;
6     rng_state = rng_state * 747796405u + 2891336453u;
7     uint word = ((state >> ((state >> 28u) + 4u)) ^ state) * 277803737u;
8     return (word >> 22u) ^ word;
9 }
10
11 float random(inout uint rng_state)
12 {
13     return float(rand_pcg(rng_state)) / float(0xFFFFFFFFu);
14 }
```

Listarea 5.2: Generatorul de numere aleatoare PCG⁴

5.4 Ray Generation shader

Razele sunt generate în scenă pornind din fiecare pixel al imaginii. În funcție de setări, se pot genera mai multe raze per pixel, pentru a obține o imagine mai clară. Strategia de eșantionare este la bază stratificată și poate fi configurată, putând fi activat sau dezactivat jittering-ul.

Pentru algoritmul normal de Path Tracing, se așteaptă să se calculeze contribuția pentru fiecare eșantion și la final se stochează în buffer media aritmetică a acestora. În schimb, pentru algoritmul progresiv, se calculează contribuția pentru un singur eșantion și se acumulează în buffer folosind un factor de interpolare care asignează o pondere dependentă de indexul eșantionului curent. Implementările pentru aceste rutine se află în shadere de tip Ray Generation și sunt ilustrate în listările 5.3 și 5.4.

```

1 [shader("raygeneration")] void RaygenShader_PathTracing()
2 {
3     // Initialize the random number generator.
4     uint rng_state = hash(DispatchRaysIndex().xy, g_sceneCB.elapsedTicks);
5
6     float3 finalColor = float3(0.0f, 0.0f, 0.0f);
7
8     uint numSamples = g_sceneCB.pathSqrtSamplesPerPixel * g_sceneCB.
9         pathSqrtSamplesPerPixel;
10
11    for (uint i = 0; i < g_sceneCB.pathSqrtSamplesPerPixel; i++)
12        for (uint j = 0; j < g_sceneCB.pathSqrtSamplesPerPixel; j++)
13    {
14        // Compute the ray offset inside the pixel (for stratified sampling), between 0
15        // and 1.
16        float2 offset = numSamples == 1 ? float2(0.5f, 0.5f) : float2((i + 0.5f) /
g_sceneCB.pathSqrtSamplesPerPixel, (j + 0.5f) / g_sceneCB.pathSqrtSamplesPerPixel);
17
18        // Apply jittering if enabled.
```

⁴<https://www.reedbeta.com/blog/hash-functions-for-gpu-rendering/>. Accesat 24.06.2024.

```

17     float2 jitter = select(g_sceneCB.applyJitter, float2(random(rng_state), random(
18         rng_state)), float2(0.5f, 0.5f)); // in [0, 1)
19     offset += (jitter - 0.5f) / g_sceneCB.pathSqrtSamplesPerPixel;
20
21     // Generate a ray for a camera pixel corresponding to an index from the dispatched
22     // 2D grid.
23     Ray ray = GenerateCameraRay(DispatchRaysIndex().xy, g_sceneCB.cameraPosition.xyz,
24         g_sceneCB.projectionToWorld, offset);
25
26     // Cast a ray into the scene and retrieve a shaded color.
27     UINT currentRecursionDepth = 0;
28     float4 color = TraceRadianceRay(ray, float4(1.0f, 1.0f, 1.0f, 1.0f), float4(0.0f,
29         0.0f, 0.0f, 0.0f), currentRecursionDepth);
30
31     // Accumulate the color.
32     finalColor += color.xyz;
33 }
34
35 // Average the accumulated color.
36 g_renderTarget[DispatchRaysIndex().xy] = float4(finalColor / numSamples, 1.0f);
37 }
```

Listarea 5.3: Generarea razelor în algoritmul Path Tracing

```

1 [shader("raygeneration")] void RaygenShader_PathTracingTemporal()
2 {
3     // Initialize the random number generator.
4     uint rng_state = hash(DispatchRaysIndex().xy, g_sceneCB.elapsedTicks);
5
6     uint numSamples = g_sceneCB.pathSqrtSamplesPerPixel * g_sceneCB.
7         pathSqrtSamplesPerPixel;
8
9     // Compute the ray offset inside the pixel (for stratified sampling), between 0 and 1.
10    float2 offset = numSamples == 1 ? float2(0.5f, 0.5f) : float2((g_sceneCB.
11        pathFrameCacheIndex - 1) % g_sceneCB.pathSqrtSamplesPerPixel + 0.5f) / g_sceneCB.
12        pathSqrtSamplesPerPixel, (floor((g_sceneCB.pathFrameCacheIndex - 1) / g_sceneCB.
13        pathSqrtSamplesPerPixel) + 0.5f) / g_sceneCB.pathSqrtSamplesPerPixel);
14
15    // Apply jittering if enabled.
16    float2 jitter = select(g_sceneCB.applyJitter, float2(random(rng_state), random(
17        rng_state)), float2(0.5f, 0.5f)); // in [0, 1)
18    offset += (jitter - 0.5f) / g_sceneCB.pathSqrtSamplesPerPixel;
19
20    // Generate a ray for a camera pixel corresponding to an index from the dispatched 2D
21    // grid.
22    Ray ray = GenerateCameraRay(DispatchRaysIndex().xy, g_sceneCB.cameraPosition.xyz,
23        g_sceneCB.projectionToWorld, offset);
24
25    // Cast a ray into the scene and retrieve a shaded color.
26    UINT currentRecursionDepth = 0;
27    float4 color = TraceRadianceRay(ray, float4(1.0f, 1.0f, 1.0f, 1.0f), float4(0.0f, 0.0f
28        , 0.0f, 0.0f), currentRecursionDepth);
29
30    // Accumulate the color.
31    const float lerpFactor = 1.0f * (g_sceneCB.pathFrameCacheIndex - 1) / g_sceneCB.
32        pathFrameCacheIndex;
33    g_renderTarget[DispatchRaysIndex().xy] = float4(lerp(color.xyz, g_renderTarget[
34        DispatchRaysIndex().xy].xyz, lerpFactor), 1.0f);
35 }
```

Listarea 5.4: Generarea razelor în algoritmul Path Tracing progresiv

5.5 Closest Hit shader

Sunt două shadere de acest tip - unul pentru primitive și unul pentru bounding box-urile suprafețelor implicate și analitice. Ambele apelează aceeași funcție helper care alege dintre algoritmul de Ray Tracing Whitted și Path Tracing pentru a calcula reflectanță în punctul de intersecție. Iluminarea pentru Ray Tracing era deja implementată în schelet, așa că nu o mai includem. Codul pentru Path Tracing este ilustrat în listarea 5.5.

```

1 void ClosestHitHelper(inout RayPayload rayPayload, in float3 normal, in float3
2   hitPosition)
3 {
4   float4 color = float4(0.0f, 0.0f, 0.0f, 1.0f);
5   if (g_sceneCB.raytracingType > 0) // path tracing
6   {
7     color.xyz = DoPathTracing(rayPayload, l_pbrCB, normal, hitPosition, RayTCurrent());
8   }
9   else // Whitted-style ray tracing
10  {
11    // Reflected component.
12    // color.xyz = ...
13  }
14
15  // Apply visibility falloff.
16  float t = RayTCurrent();
17  color = lerp(color, g_sceneCB.backgroundColor, 1.0 - exp(-0.000002 * t * t * t));
18
19  rayPayload.color = color;
}

```

Listarea 5.5: Closest Hit helper pentru Path Tracing

5.5.1 Algoritmul Path Tracing

Funcția DoPathTracing definește întreg algoritmul de Path Tracing. Aici se efectuează toate optimizările și tehniciile de accelerare a convergenței despre care am vorbit în capitolul 4. Listarea B.2 ilustrează implementarea algoritmului.

Observăm la liniile 10, 13 și 78 prezența unui termen de absorbție. Aceasta modelează absorbția volumetrică a luminii când parcurge un mediu și se aplică doar în cazul în care raza se află în interiorul unui obiect. Modelul este bazat pe legea Beer-Lambert:

$$T = e^{-\sigma_a d}, \quad (5.5.1)$$

unde T reprezintă coeficientul de transmisie, σ_a este coeficientul de absorbție, iar d este distanța parcursă în interiorul obiectului. Pentru parametrizare în cadrul materialului, am introdus 2 noi parametri: *atDistance* și *extinction*. Înversând ecuația 5.5.1, putem obține coeficientul de absorbție:

$$\sigma_a = -\frac{\log(extinction)}{atDistance}. \quad (5.5.2)$$

Functiile MIS și NextEventEstimation sunt folosite pentru eșantionarea multiplă bazată

pe importanță. Acestea sunt implementate conform teoriei prezentate în capitolul 3. Codul pentru aceste funcții se află în listarea 5.6.

```

1 // Samples a light source.
2 bool NextEventEstimation(inout uint rng_state, in LightBuffer light, in float3
3     hitPosition,
4     in uint recursionDepth, out LightSample lightSample)
5 {
6     float angle;
7     float2 eps;
8     float3 samplePos;
9     switch (light.type)
10    {
11        case LightType::Square:
12            eps = float2(random(rng_state), random(rng_state));
13            samplePos = light.position + float3(eps.x - 0.5f, 0.0f, eps.y - 0.5f) * light.size;
14            lightSample.L = samplePos - hitPosition;
15            lightSample.dist = length(lightSample.L);
16            lightSample.L /= lightSample.dist;
17            // bail out if the sample is on the wrong side of the light
18            angle = dot(lightSample.L, float3(0.0f, 1.0f, 0.0f));
19            if (angle <= 0.0f)
20                return false;
21            lightSample.emission = light.intensity * light.emission;
22            lightSample.pdf = (sq(lightSample.dist)) / (sq(light.size) * angle);
23            break;
24        case LightType::Directional:
25            lightSample.L = -normalize(light.direction);
26            lightSample.dist = INFINITY;
27            lightSample.emission = light.intensity * light.emission;
28            lightSample.pdf = 1.0f;
29            break;
30    }
31
32    // Check if the light is occluded.
33    Ray shadowRay = {hitPosition, lightSample.L};
34    bool shadowRayHit = recursionDepth < g_sceneCB.maxShadowRecursionDepth &&
35        TraceShadowRayAndReportIfHit(shadowRay, recursionDepth);
36    return !shadowRayHit;
37}
38
39 float3 MIS(inout uint rng_state, PBRPrimitiveConstantBuffer material, in float eta,
40     in float3 hitPosition, in LightBuffer light, in float3 N, in uint recursionDepth)
41 {
42     float3 reflectance = float3(0.0f, 0.0f, 0.0f);
43
44     // Sample the light source.
45     LightSample lightSample;
46     if (NextEventEstimation(rng_state, light, hitPosition, recursionDepth, lightSample))
47     {
48         // Evaluate the BSDF.
49         float bsdfPdf;
50         reflectance = EvaluateDisneyBSDF(material, g_sceneCB.anisotropicBSDF, eta, -
51             WorldRayDirection(), lightSample.L, N, bsdfPdf);
52
53         // Calculate the MIS weight.
54         float weight = 1.0f;
55         if (light.type != LightType::Directional)
56             weight = PowerHeuristic(1.0f, lightSample.pdf, 1.0f, bsdfPdf);
57
58         // Calculate the final color.
59         if (bsdfPdf > 0.0f)
60             reflectance *= weight * lightSample.emission / lightSample.pdf;
61         else
62     }
```

```

59     reflectance = float3(0.0f, 0.0f, 0.0f);
60 }
61
62 return reflectance;
63 }
```

Listarea 5.6: Funcțiile de eșantionare multiplă bazată pe importanță

5.6 Miss shader

În cazul în care o rază nu intersectează niciun obiect din scenă, se execută shader-ul de tip Miss. Acesta adaugă culoarea de fundal drept contribuție la cale (vezi listarea 5.7).

```

1 [shader("miss")] void MissShader(inout RayPayload rayPayload)
2 {
3     rayPayload.color = g_sceneCB.backgroundColor * rayPayload.throughput;
4 }
```

Listarea 5.7: Shader-ul de tip Miss

5.7 Evaluarea și eșantionarea BSDF-ului

În secțiunile 3.5.2 și 4.3 am prezentat bazele teoretice ale funcțiilor de distribuție a reflectanței și am prezentat modelul Disney Principled BSDF. Totuși, nu am vorbit despre eșantionarea acestora. Aceasta este un proces complex și de obicei nu se găsesc soluții analitice. Astfel, diferite implementări folosesc diferite metode de eșantionare și asignează ponderi diferite lobilor. Totuși, ideea de bază este aceeași: pentru evaluare, se evaluează toți lobii (care au sens - nu evaluăm transmisia dacă avem reflexie) și se ponderează în funcție de parametrii materialului, pentru a asigna un fel de probabilitate ad-hoc fiecărui lob. Pentru eșantionare (alegerea direcției razei următoare), se alege probabilistic (folosind aceeași ponderi ca la evaluare) un singur lob și se eșantionează conform unei distribuții bazate pe importanță ce aproximează cât mai bine distribuția microfațetelor din cadrul BRDF-ului.

În implementare am urmat modelul de eșantionare folosit în [20]. Codul se află în listarea B.3. Vreau să subliniez faptul că mi-a fost foarte greu să ajung la o implementare corectă a BSDF-ului, din cauza complexității matematice.

6 EVALUARE

6.1 Evaluarea performanțelor

În continuare prezentăm rezultatele obținute în urma testelor de performanță. Toate testele folosesc aceeași scenă și aceeași poziționare a camerei (ca în Figura 34), astfel încât să se observe impactul fiecărei setări asupra performanței. Vom varia pe rând fiecare setare, ținând restul constantă. Valorile implicate ale setărilor sunt:

- Numărul de eșantioane per pixel: 2
- Adâncimea maximă de recursivitate: 2
- Adâncimea minimă pentru ruleta rusească: 3
- O singură lumină eșantionată: adevărat
- Metodă de eșantionare a direcției următoare: BSDF.

Rezultatele sunt prezentate în tabelele următoare. Testele au fost efectuate la o rezoluție de 1920x1080 pixeli, pe o placă video Nvidia RTX 2060 Super. Menționăm faptul că FPS-urile sunt limitate la 140.

6.1.1 Numărul de eșantioane per pixel

Tabela 1: Performanță în funcție de numărul de eșantioane per pixel

Valoare Setată	FPS
1	75
4	16
8	6
16	1

Rezultatele din tabela 1 arată că numărul de eșantioane per pixel are un impact imens asupra performanței. Un număr modest de 4 eșantioane per pixel nu este totuși suficient pentru a crea o imagine stabilă în absența tehniciilor de denoising, aşadar algoritmul este limitat în acest aspect.

6.1.2 Adâncimea maximă a recursivității

Rezultatele din tabela 2 arată că adâncimea maximă a recursivității are și ea un impact semnificativ asupra performanței. Pentru a avea reflexii convingătoare, o adâncime de 3 este

Tabela 2: Performanța în funcție de numărul de adâncimea maximă a recursivității

Valoare Setată	FPS
1	140
2	77
3	40
4	36
5	33
6	32
7	31
8	31
9	30
10	29

suficientă, pe cînd în cazul refracțiilor, o adâncime de 5 este mai potrivită. Observăm totuși că impactul scade odată cu creșterea adâncimii, ceea ce este un efect al ruletei rusești.

6.1.3 Adâncimea minimă pentru ruleta rusească

Tabela 3: Performanța în funcție de adâncimea minimă la care se aplică ruleta rusească. Pentru acest test s-a setat adâncimea maximă de recursivitate la 10

Valoare Setată	FPS
1	35
2	34
3	34
4	32
5	32
6	31
7	31
8	30
9	30
10	29

Tabela 3 arată că ruleta rusească este o metodă destul de eficientă pentru a crește performanța de rulare, fără a sacrifica prea mult calitatea imaginii.

6.1.4 Eșantionarea unei singure lumini

Tabela 4 arată că eșantionarea unei singure lumini are un impact bun asupra performanței, fără a sacrifica deloc calitatea imaginii. Acest test s-a efectuat cu doar 2 lumini în scenă - scene mai complexe vor avea un impact imens asupra performanței dacă se eșantionează toate

Tabela 4: Performanța în eșantionarea unei singure lumini sau a amândurora

Valoare Setată	FPS
DA	87
NU	77

luminile deodată (spre deosebire de niciun impact în setarea cealaltă, însă asta ar putea aduce o reducere a calității imaginii în acest caz extrem).

6.1.5 Strategia de eșantionare a direcției următoare

Tabela 5: Performanța în funcție de strategia de eșantionare a direcției următoare

Valoare Setată	FPS
BSDF	87
Cosine-weighted Hemisphere	51
Uniform sphere	46

Tabela 5 arată că eșantionarea bazată pe BSDF este, surprinzător, cea mai eficientă dar și cea mai calitativă (vezi Figurile 37, 38 și 39). Motivul care stă în spatele scăderii performanței este faptul că celelalte două metode evaluatează complet BSDF-ul (toti lobii), în timp ce eșantionarea bazată pe BSDF alege un singur lob de evaluat. Eșantionarea uniformă a sferei este cea mai lentă, deoarece implică și o componentă de transmisie.

6.2 Evaluare calitativă

6.2.1 Comparație cu Ray Tracing Whitted

Algoritmul integrat deja în schelet este un Ray Tracer recursiv simplu, de tip Whitted, care folosește un model de iluminare Phong. Acesta nu suportă transmisie, iar reflexiile sunt perfect geometrice. O randare realizată cu acest algoritm se poate vedea în Figura 33. Torusul de tip inel și cele 3 sfere au un material ce simulează cromul și este reflectiv. Se pot observa și umbre geometrice, cauzate de două surse de lumină punctiforme. Aceeași scenă, dar randată folosind Path Tracing, se poate vedea în Figura 34. Condițiile de iluminare nu pot fi replicate perfect. Observăm imediat o diferență majoră - această randare ține cont de iluminarea globală. Scena este iluminată conform fundalului, iar contribuția reflectanței fiecărui obiect este vizibilă pe podea. În acest caz, lumina nu este destul de puternică să lupte cu fundalul, de aceea umbrele sunt destul de puțin pronunțate.

În randarea din Figura 35, culoarea de fundal a fost coborâtă. Se observă astfel contribuțile celor două surse de lumină - una de tip suprafață și una direcțională. Umbrele lăsate de

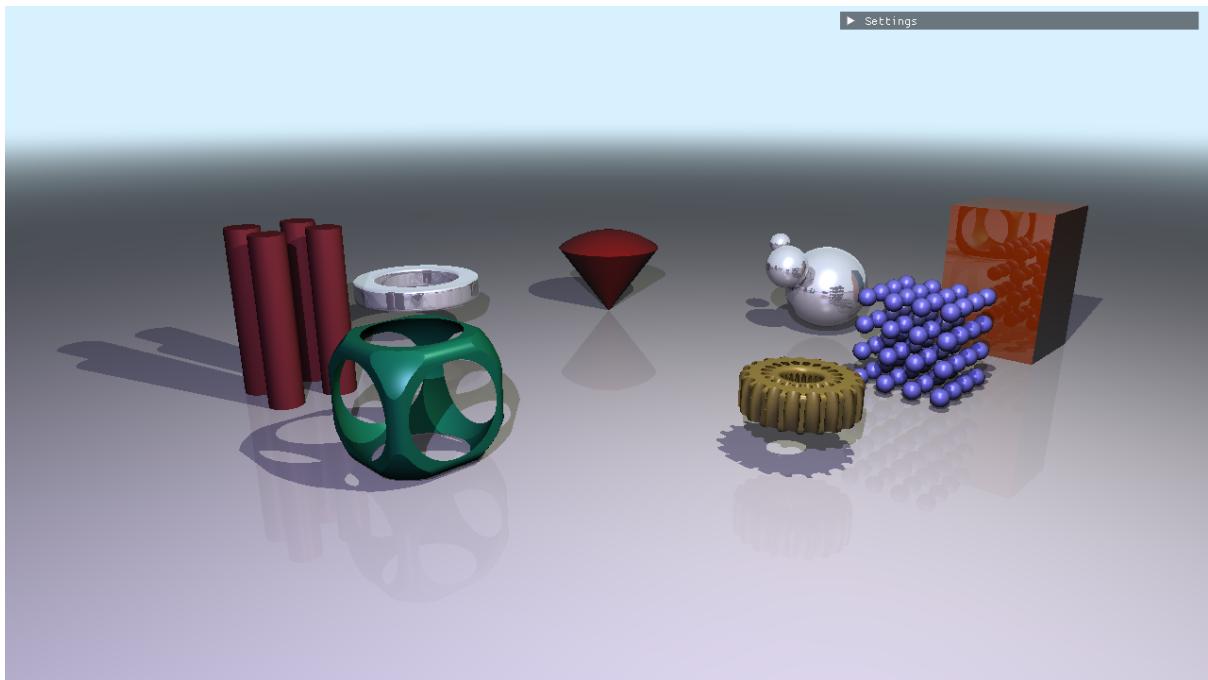


Figura 33: Randare folosind Ray Tracing Whitted

zonele obstructionate din perspectiva luminii de suprafață au un aspect natural, cu penumbre ce tranziționează lin între umbra completă și lumina directă.

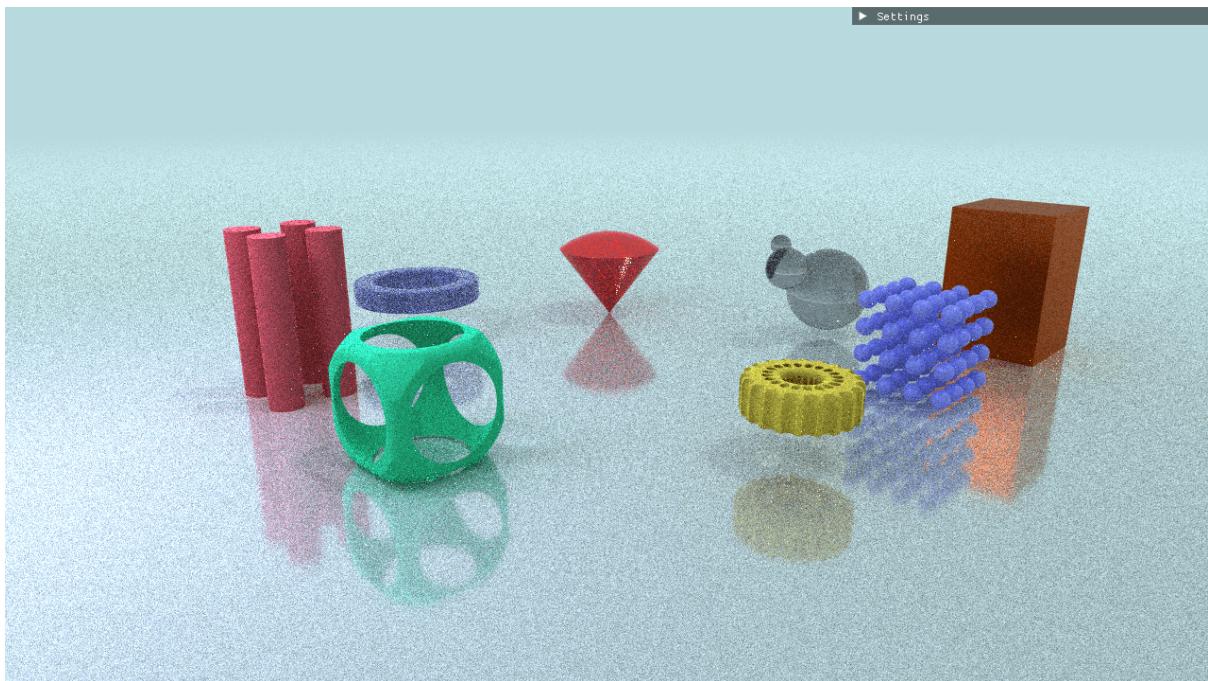


Figura 34: Randare folosind Path Tracing

Așa cum este descris în secțiunea 4.3, sistemul de materiale folosit este physically based și modelează efecte complexe precum reflexii și refracții. Un exemplu de material care dă dovadă de ambele comportamente se află în figura Figura 36. Acesta este un material dielectric, cu

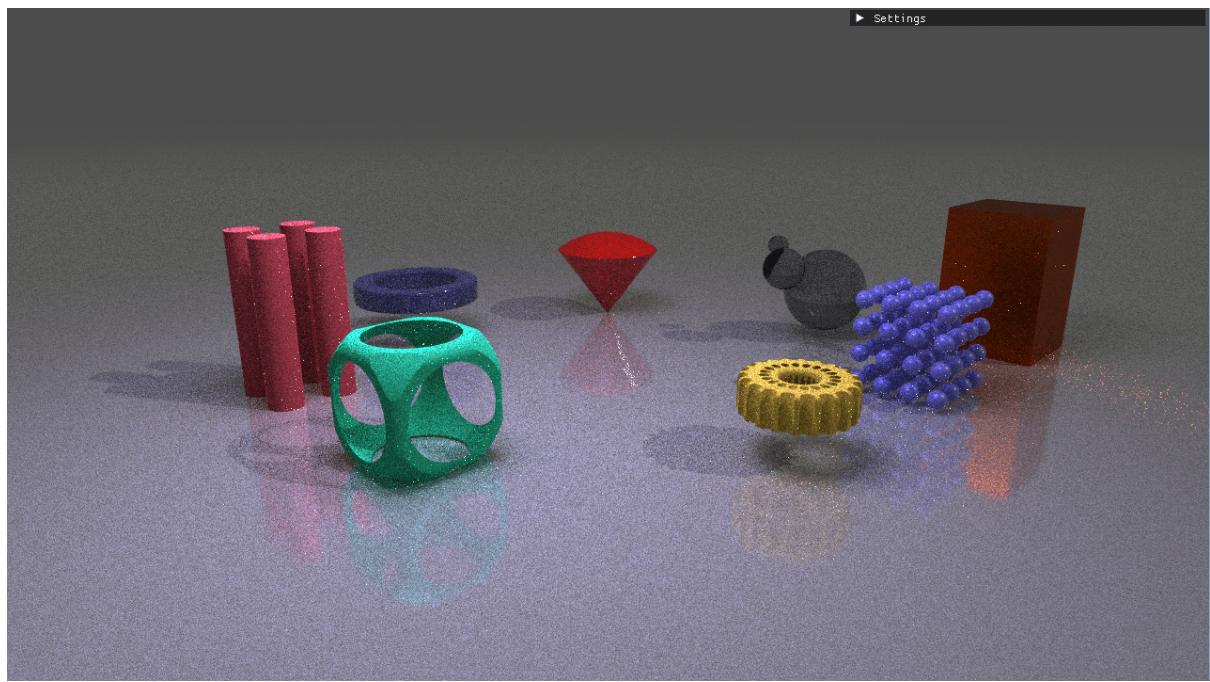


Figura 35: Randare folosind Path Tracing cu fundal mai întunecat

un indice de refracție de 1.5. Se observă refractii către obiectele din spate în zonele centrale ale sferelor și reflexii către mediul înconjurător către margini, unde unghiul de incidentă este mare. Acest comportament este consistent cu legile lui Fresnel.

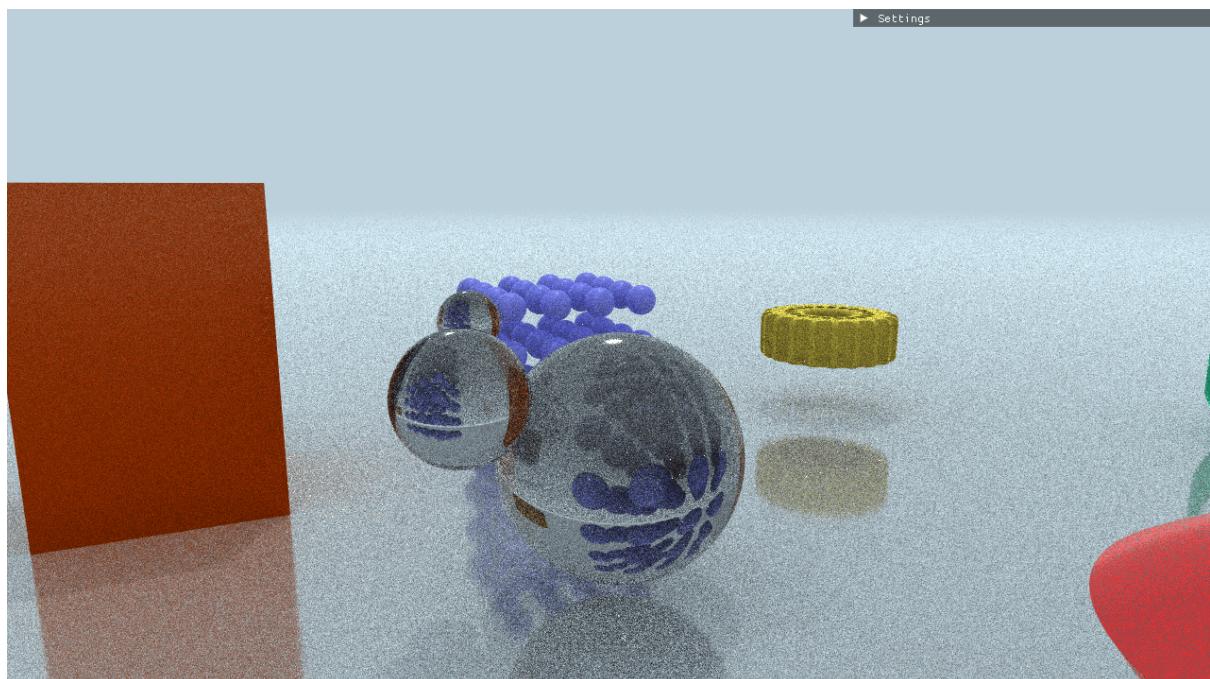


Figura 36: Randare folosind Path Tracing cu sticlă dielectrică

6.2.2 Evaluarea strategiei de eșantionare a direcției următoare

Așa cum era de așteptat, eșantionarea bazată pe importanța BSDF-ului converge cel mai repede. Celelalte strategii nu reușesc să conveargă nici la 256 eșantioane per pixel, în timp ce imaginea eșantionată cu BSDF este foarte stabilă. Ilustrații în Figurile 37, 38 și 39 arată această diferență.

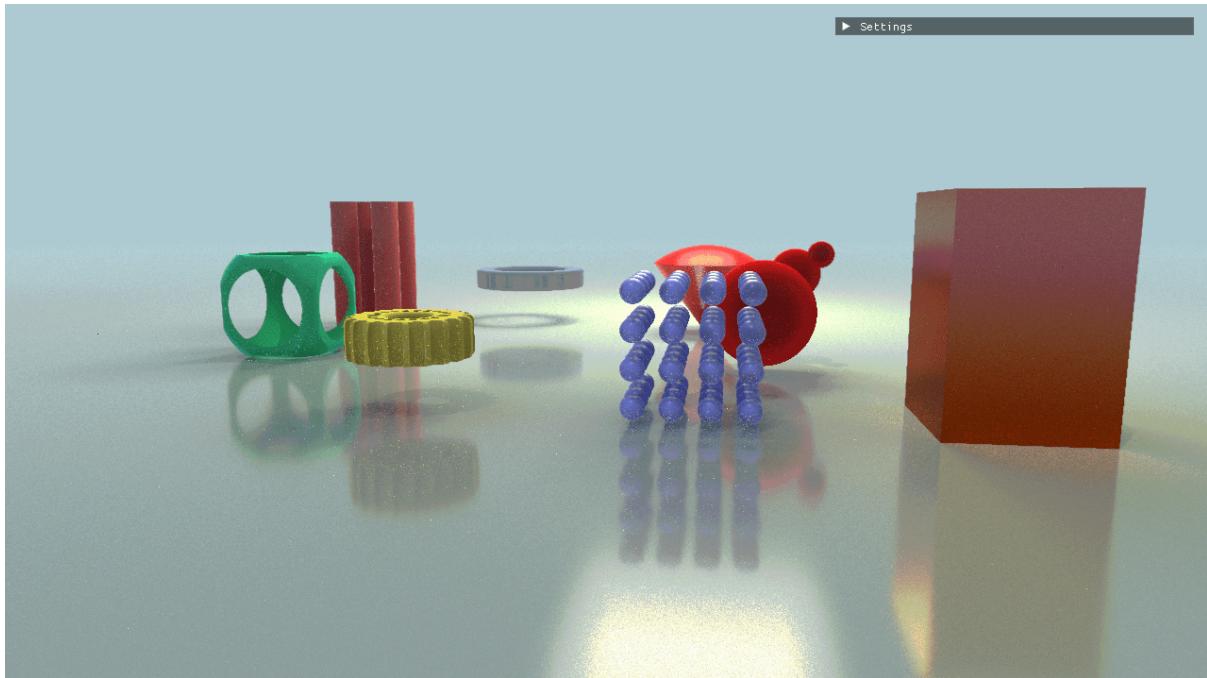


Figura 37: Eșantionare bazată pe BSDF

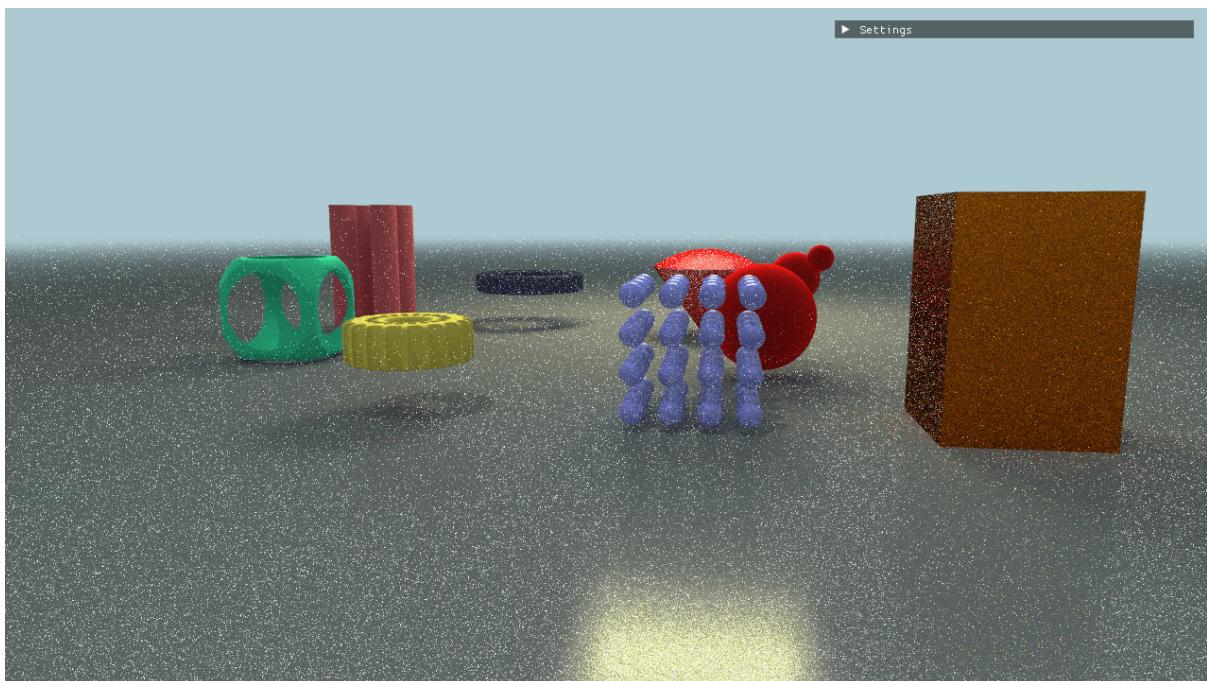


Figura 38: Eșantionare ponderată de cosinus a emisferei

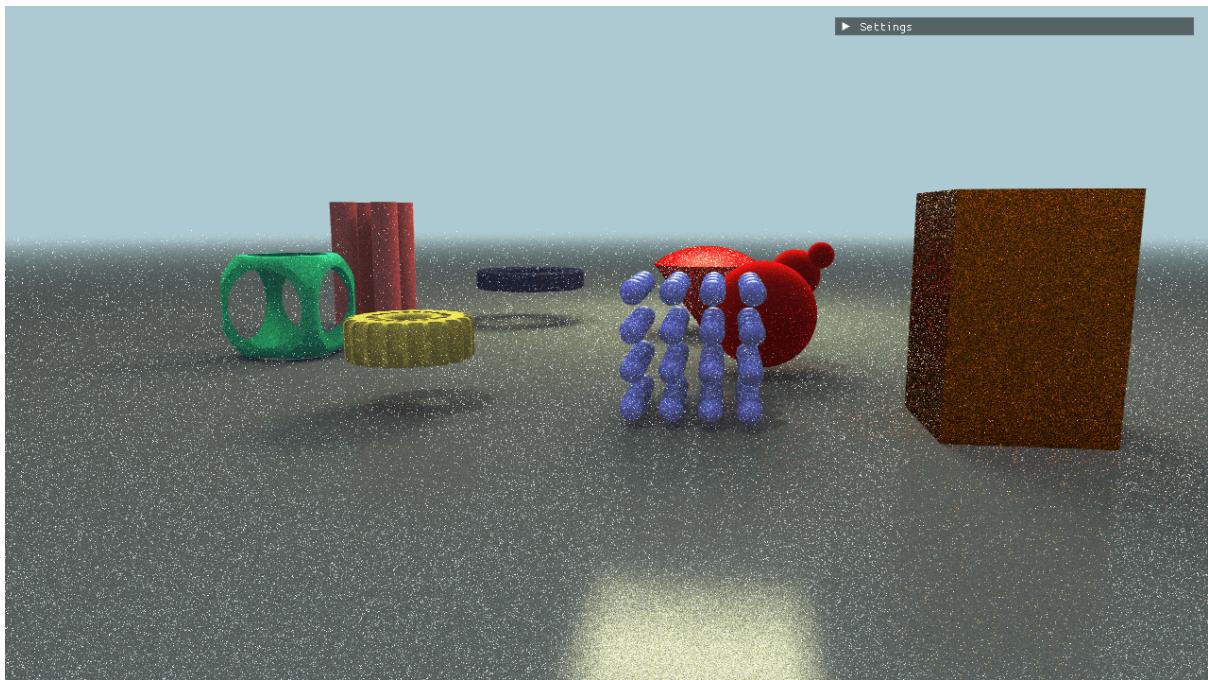


Figura 39: Eșantionare uniformă a sferei

6.2.3 Evaluarea modelului Disney principled BSDF

Am descris în secțiunea 4.3 modelul Disney principled BSDF, modelul de materiale folosit în această implementare. Acesta este un model complex dar foarte bun, folosit în producție la Disney pentru filmele lor animate încă din anul 2012 [7]. Vom supune aşadar acest model la teste pentru a ne convinge de calitatea și fidelitatea sa.

În continuare, vom evalua efectele parametrilor care definesc sistemul de materiale. Pentru această scenă s-au folosit setările la calitate maximă: 256 esantioane per pixel, algoritm progresiv și adâncime maximă de recursivitate 10. Am folosit pentru aceste randări varianta progresivă a algoritmului, pentru a obține performanțe comparabile cu un singur eșantion per pixel dar calitate mult mai bună. În următoarele randări, s-a variat câte un parametru în timp ce restul sunt la valorile implicate (i.e., 0 pentru majoritatea și 1.5 pentru indexul de refracție)

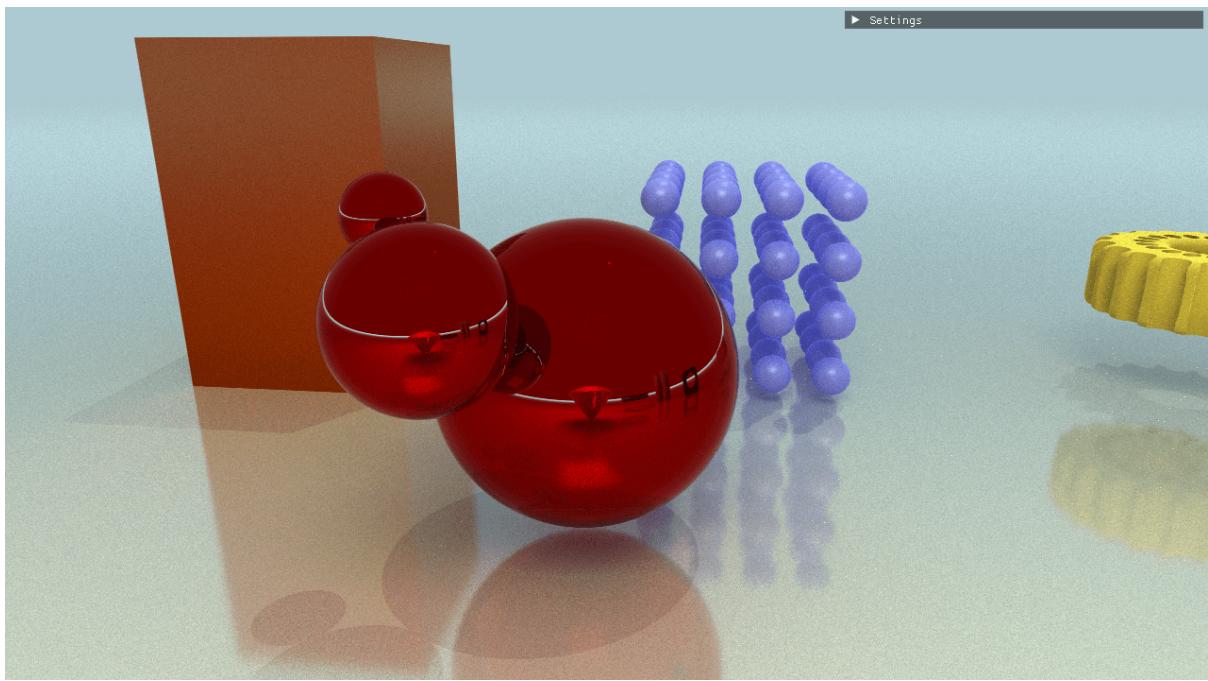


Figura 40: Material metallic cu $metallic = 1.0$. Se pot observa reflexiile speculare ce sunt colorate în funcție de culoarea materialului

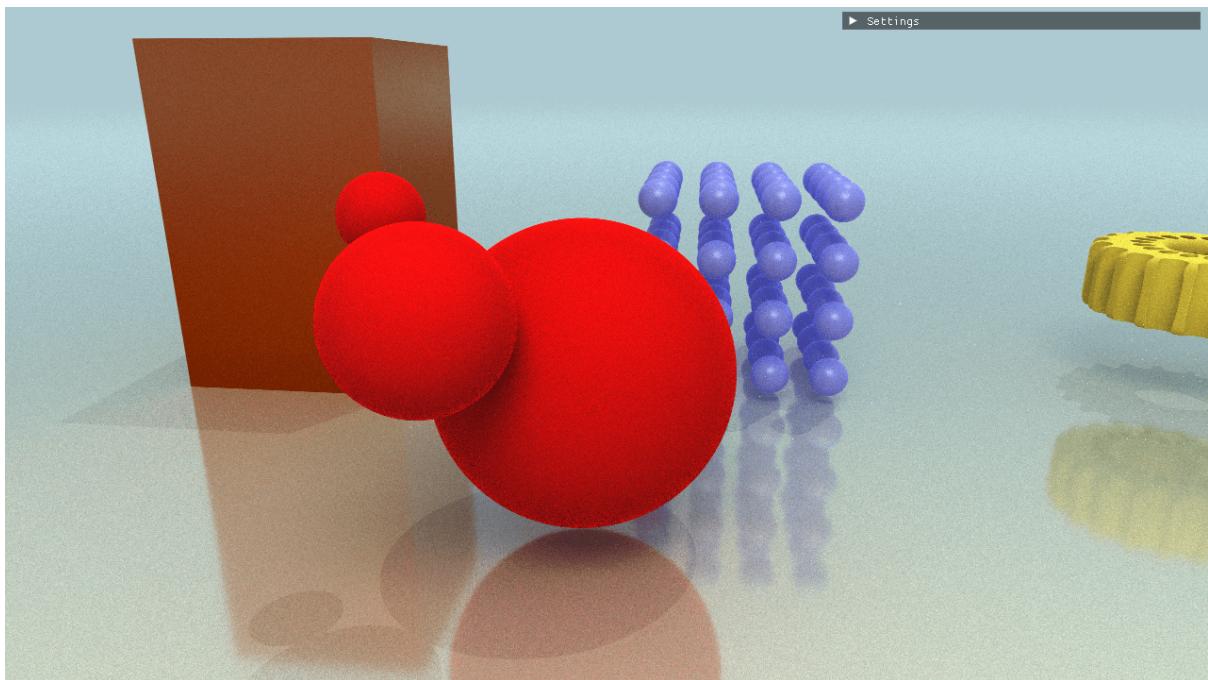


Figura 41: Material dielectric cu $roughness = 1.0$. Se observă modelul difuz

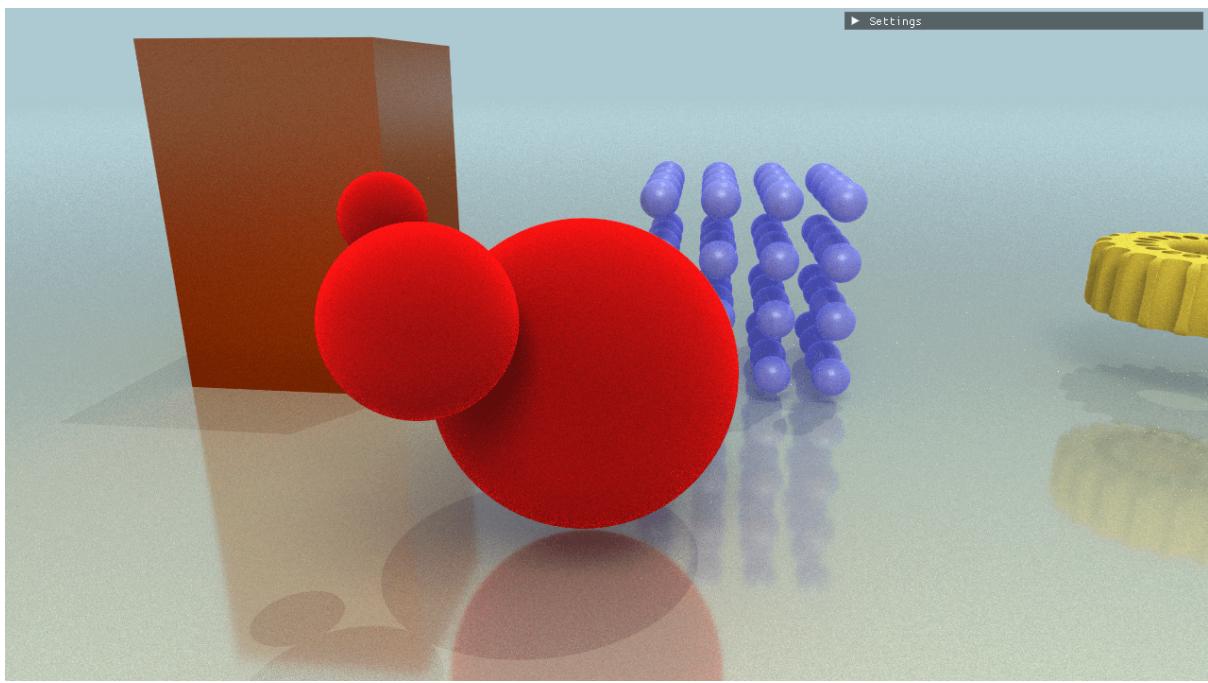


Figura 42: Material dielectric cu *subsurface* = 1.0 și *roughness* = 0.0. Se observă modelul difuz și efectul de subsurface scattering

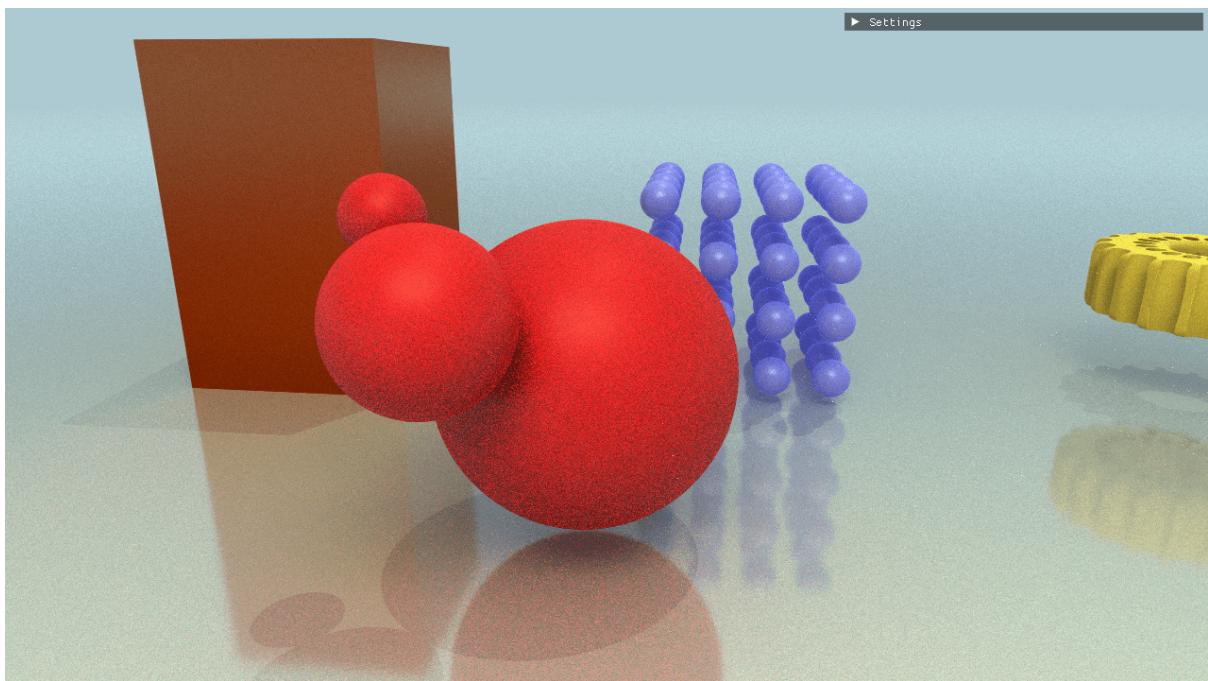


Figura 43: Material cu strat de clearcoat cu *clearcoat* = 1.0 și *roughness* = 0.5. Se observă un luciu adițional (deși minor) pe material

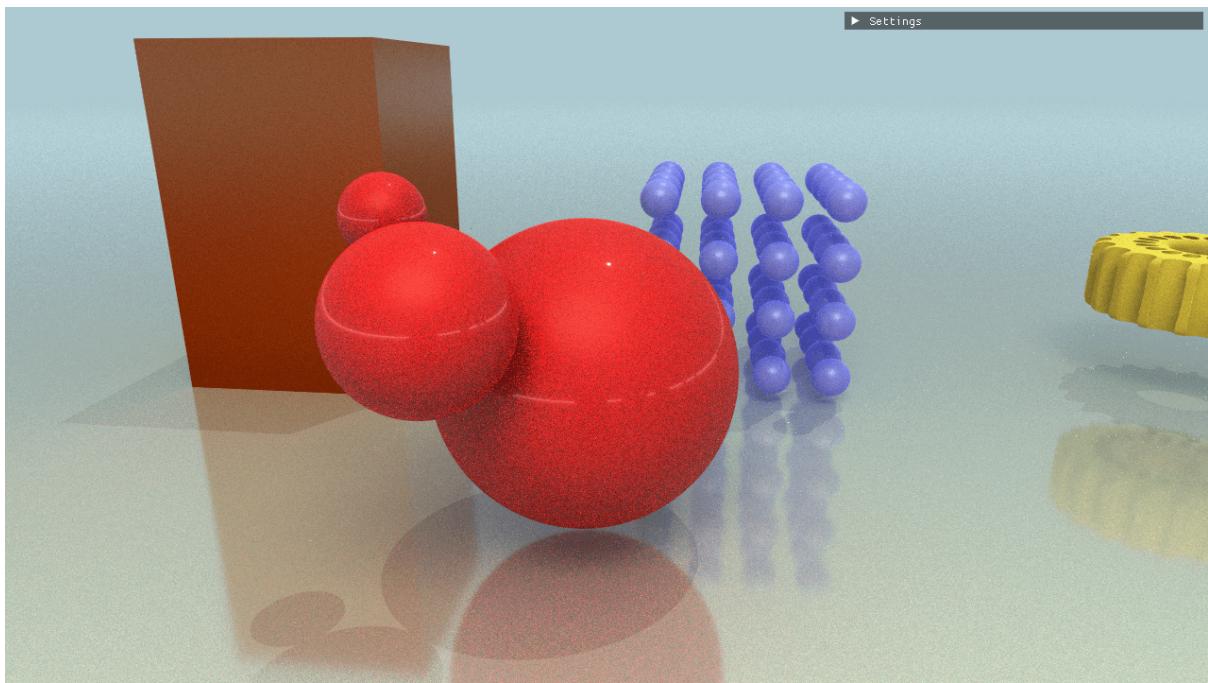


Figura 44: Material dielectric cu $clearcoat = 1.0$, $clearcoatGloss = 1.0$ și $roughness = 0.5$. Se observă efectul de luciu pronunțat

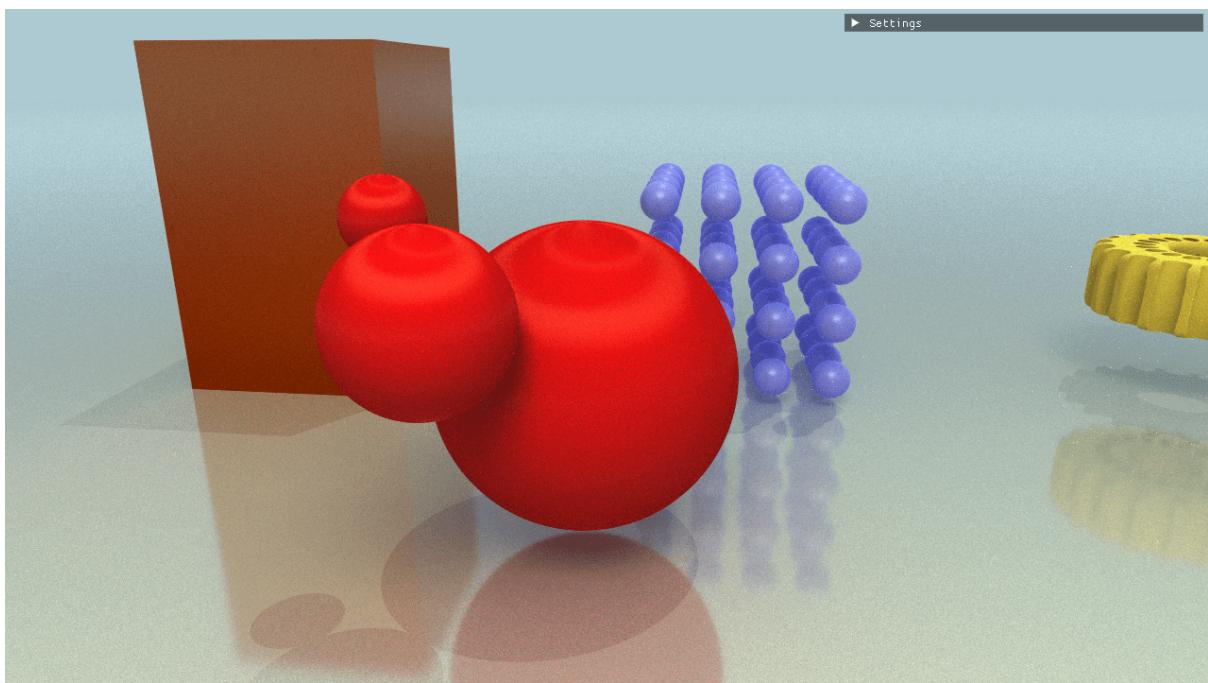


Figura 45: Material dielectric cu $anisotropy = 1.0$ și $roughness = 0.5$. Se observă anizotropia punctelor speculare

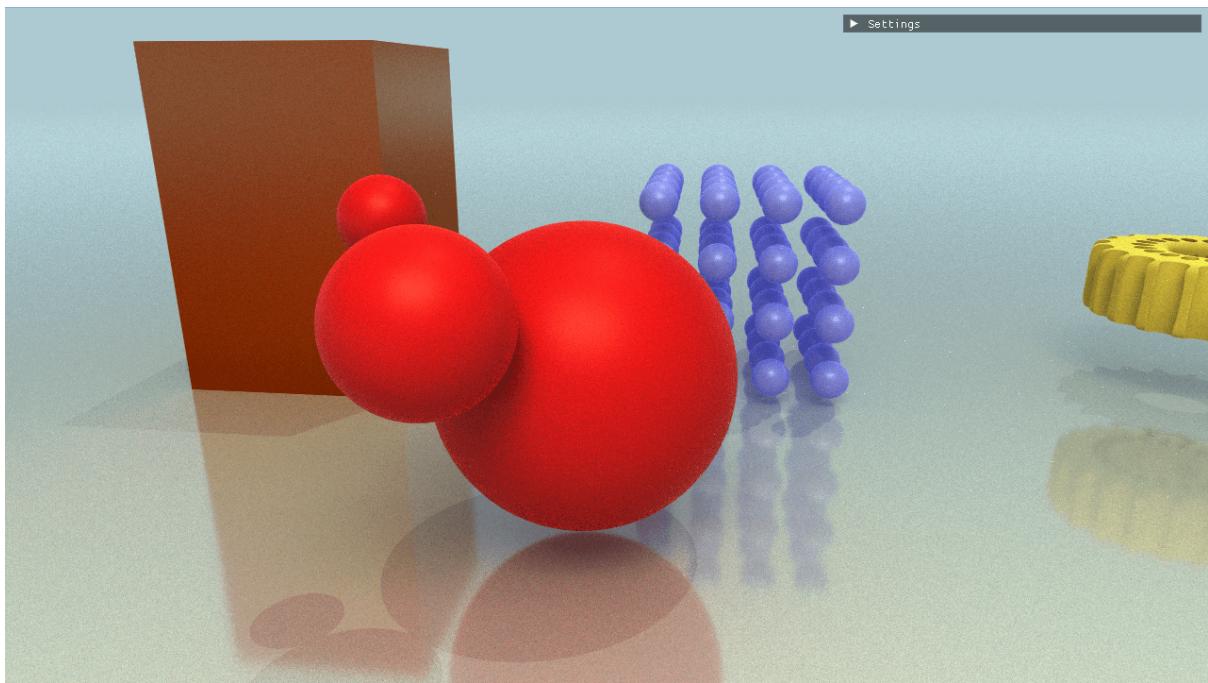


Figura 46: Material dielectric cu $sheen = 1.0$ și $roughness = 0.5$. Se observă retroreflexia la unghurile mari de incidentă

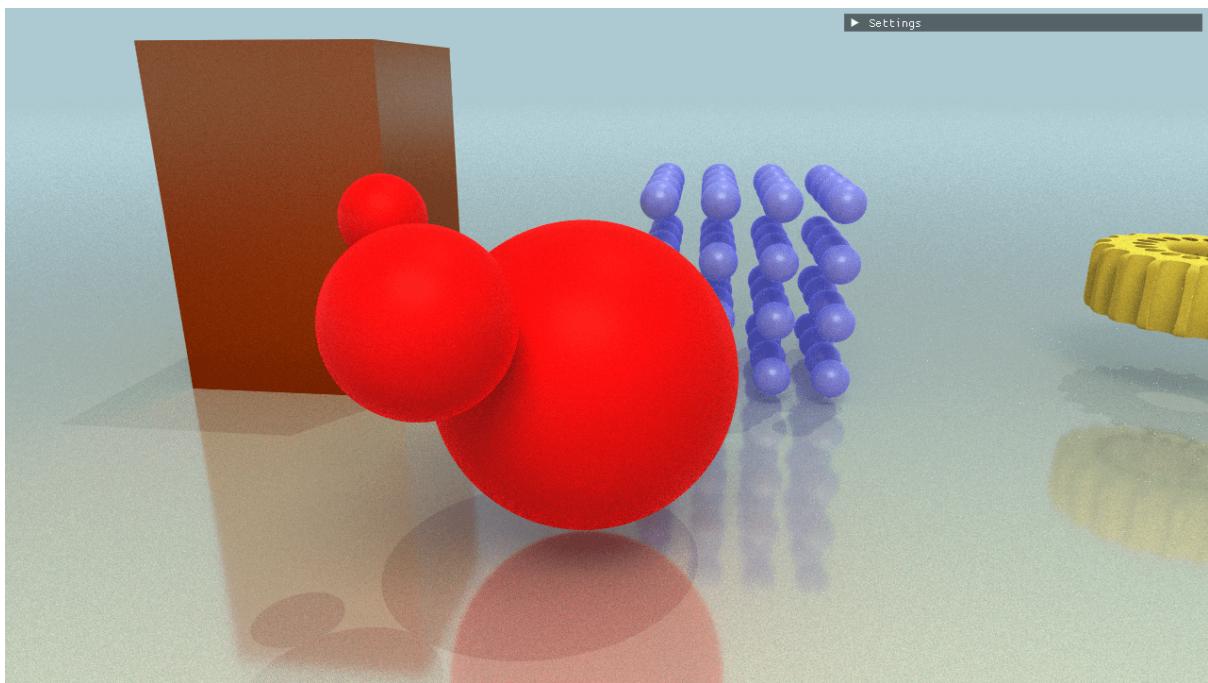


Figura 47: Material dielectric cu $sheen = 1.0$, $sheenTint = 1.0$ și $roughness = 0.5$. Se observă retroreflexia accentuată și colorată

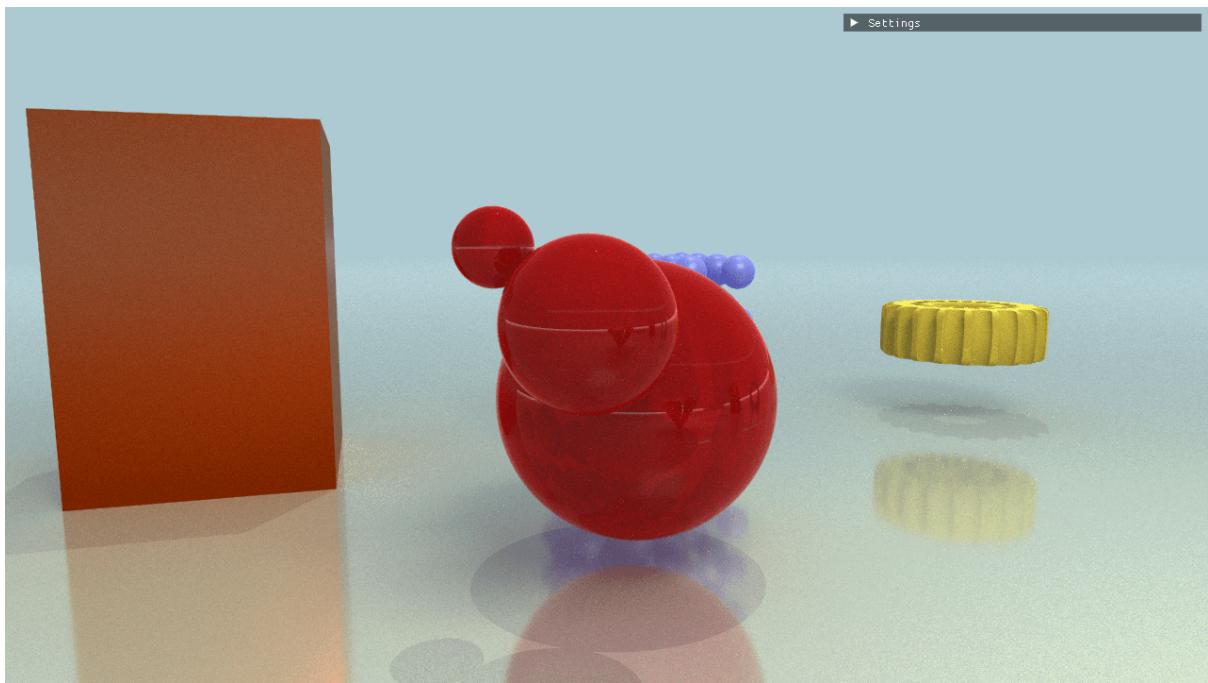


Figura 48: Material dielectric cu $\text{specularTransmission} = 0.5$ și $\text{roughness} = 0.0$. Se observă refractiile și reflexile dielectrice

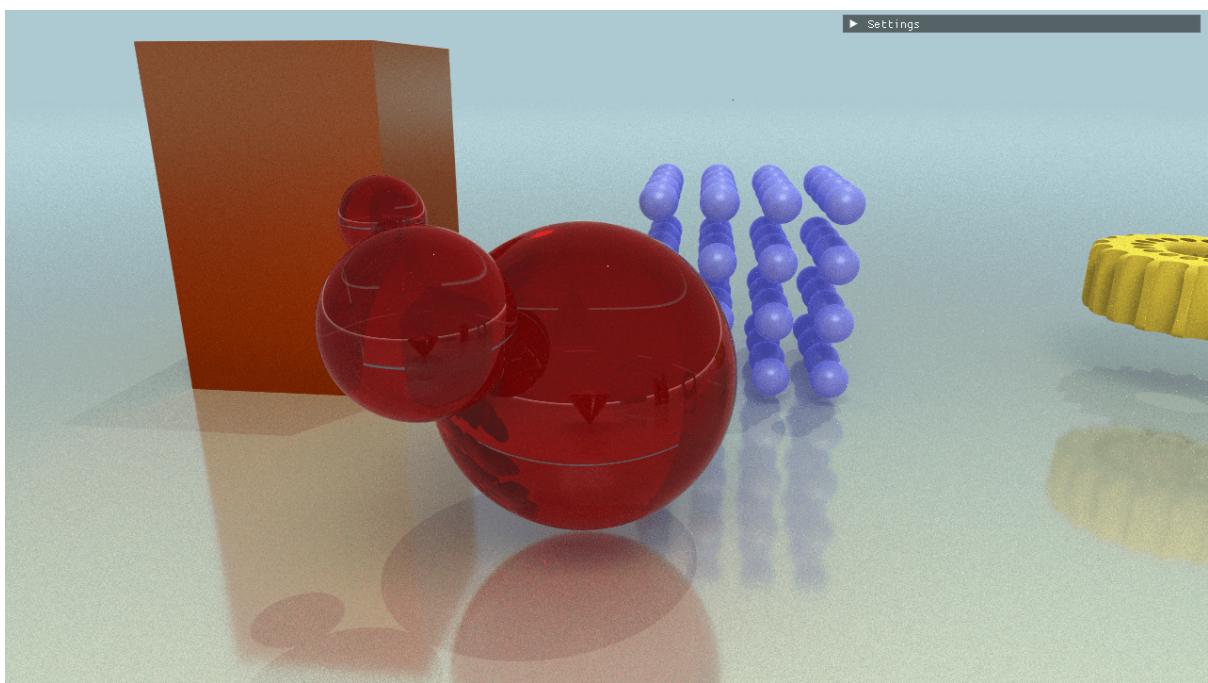


Figura 49: Material dielectric cu $\text{specularTransmission} = 1.0$ și $\text{roughness} = 0.0$. Se observă refractiile pronunțate și reflexile dielectrice

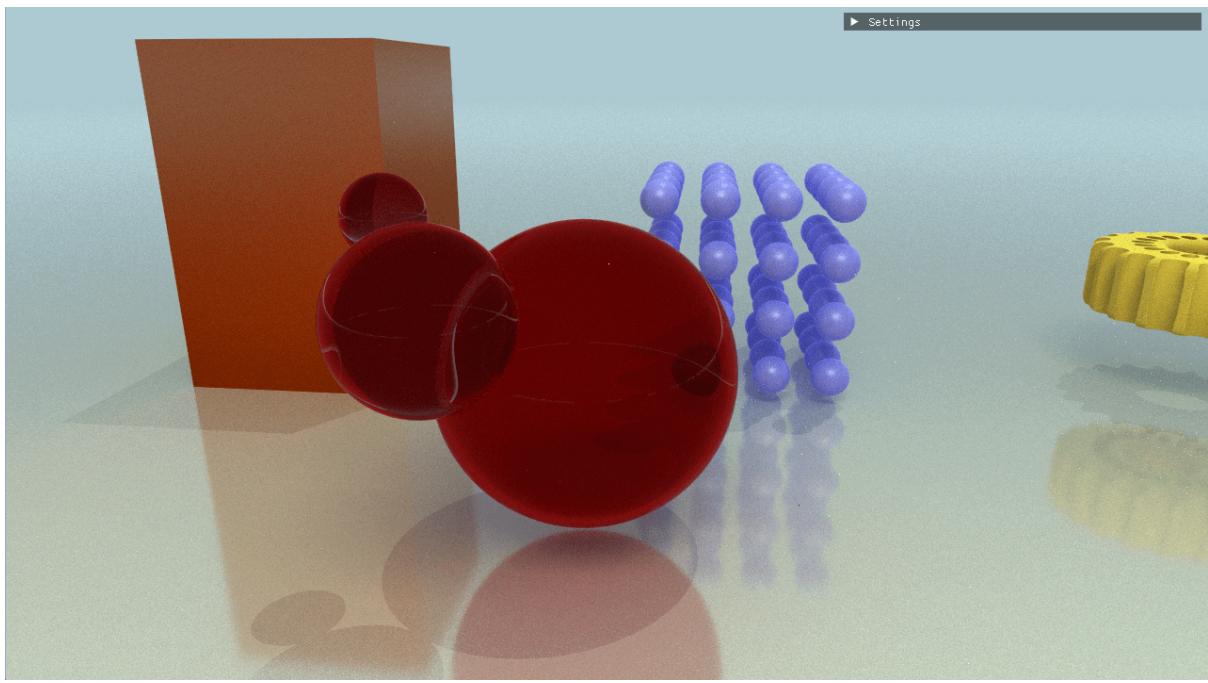


Figura 50: Material dielectric cu $specularTransmission = 1.0$ și $ior = 1.1$.
Se observă refracții la devieri mici

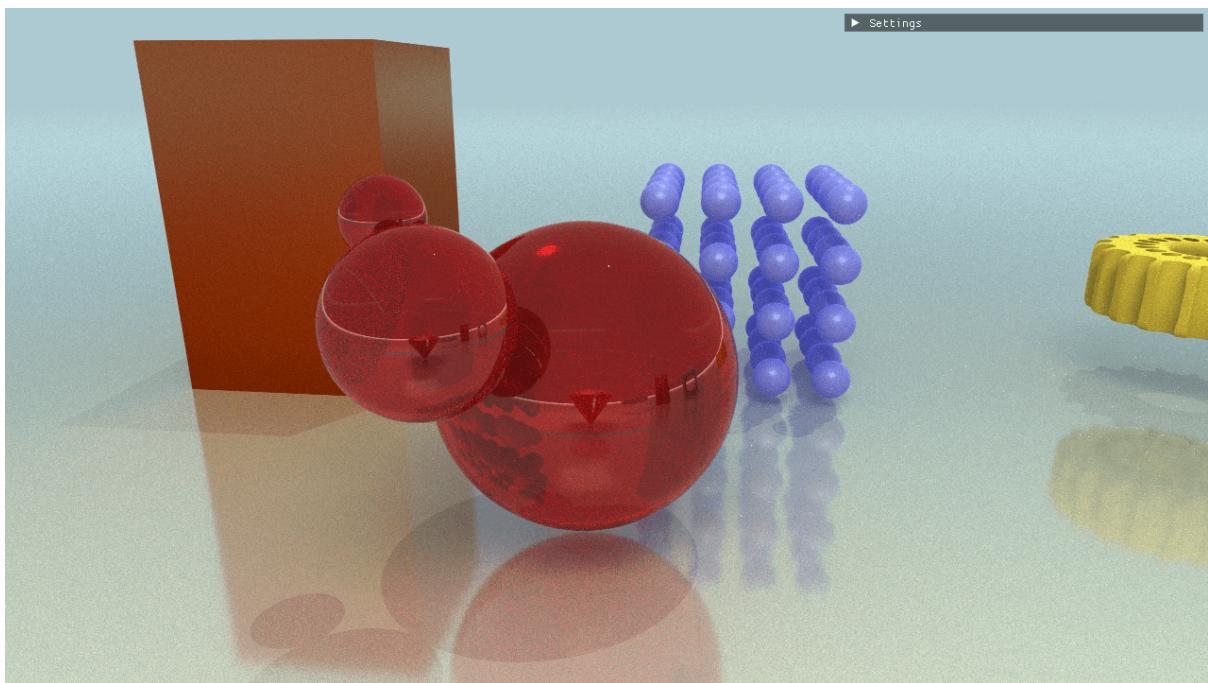


Figura 51: Material dielectric cu $specularTransmission = 1.0$ și $ior = 2.0$.
Se observă refracții la devieri mari

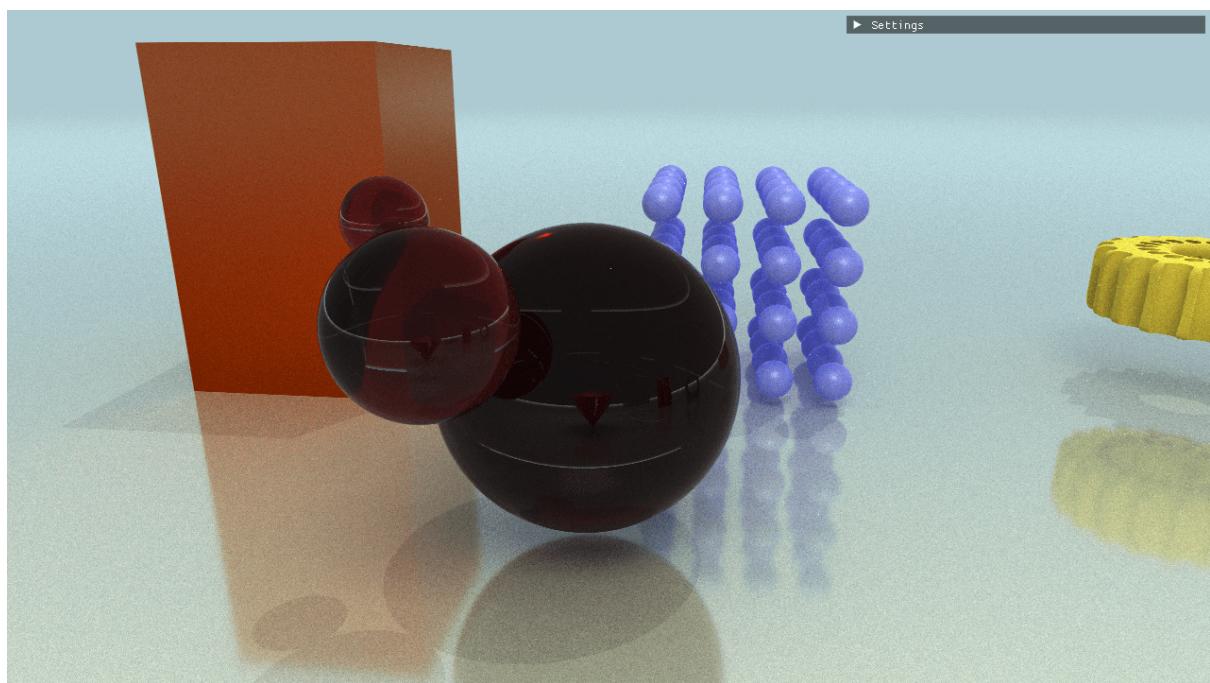


Figura 52: Material dielectric cu $extinction = (0.3, 0.0, 0.0)$ și $atDistance = 0.1$. Se observă cum refracția este afectată de absorbție

7 CONCLUZII

Procesul implementării unui Path Tracer a fost unul complex și plin de provocări. Desigur, varianta brută a algoritmului este simplă, însă pentru a obține o imagine de calitate care să conveargă repede și în timp real este nevoie de multe optimizări și tehnici avansate. Am observat în capitolele 3 și 5 cât de complexă este teoria probabilistică din spate, dar și modelele care aproximează fenomenele fizice ale interacțiunii luminii cu suprafețele. Rezultatele au fost totuși satisfăcătoare din punct de vedere calitativ, însă performanța nu este tocmai în timp real, din cauza sistemului complex de materiale. În practică, un Path Tracer real-time folosește un număr foarte mic de eșantioane per pixel dar compensează prin metode de eșantionare inteligente și tehnici de denoising și mai inteligente.

Ca direcție viitoare, aş dori să explorez algoritmii de denoising din literatura de specialitate și să implementez unul în proiectul meu. Am văzut cum impactul imens asupra performanței poate fi ameliorat printr-o implementare progresivă a algoritmului de Path Tracing, însă pentru a elimina blur-ul trebuie implementat un algoritm de denoising care să țină cont de istoricul eșantioanelor și de mișcarea camerei și a obiectelor din scenă.

În concluzie, consider că lucrarea de față și-a atins scopul de a explora teoria și a implementa într-un context real un Path Tracer. Doresc ca lucrarea de față să servească drept material didactic pentru cei aflați la început de drum și să reprezinte o privire de ansamblu, dar cu suficient de multe detalii pentru a înțelege cu adevărat algoritmul Path Tracing.

BIBLIOGRAFIE

- [1] Deriving lambertian brdf from first principles. <https://sakibsaikia.github.io/graphics/2019/09/10/Deriving-Lambertian-BRDF-From-First-Principles.html>. Accesat 24.06.2024.
- [2] Sampling reflection functions. https://pbr-book.org/3ed-2018/Light_Transport_I_Surface_Reflection/Sampling_Reflection_Functions#MicrofacetDistribution::Pdf. Accesat 24.06.2024.
- [3] Sampling the hemisphere. <https://ameye.dev/notes/sampling-the-hemisphere/>. Accesat 24.06.2024.
- [4] Ucsd cse 272 assignment 1: Disney principled bsdf. <https://cseweb.ucsd.edu/~tzli/cse272/wi2023/homework1.pdf>. Accesat 24.06.2024.
- [5] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), page 37–45, New York, NY, USA, 1968. Association for Computing Machinery.
- [6] James F. Blinn. Models of light reflection for computer synthesized pictures. *SIGGRAPH Comput. Graph.*, 11(2):192–198, jul 1977.
- [7] Brent Burley. Physically-based shading at disney. 2012.
- [8] Petrik Clarberg, Simon Kallweit, Craig Kolb, Paweł Kozłowski, Yong He, Lifan Wu, and Edward Liu. Research Advances Toward Real-Time Path Tracing. Game Developers Conference (GDC), March 2022.
- [9] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Trans. Graph.*, 1(1):7–24, jan 1982.
- [10] John H. Halton. A retrospective and prospective survey of the monte carlo method. *SIAM Rev.*, 12(1):1–63, jan 1970.
- [11] J. M. Hammersley and David Handscomb. Monte carlo methods. 1964.
- [12] John C Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.
- [13] Eric Heitz. Understanding the masking-shadowing function in microfacet-based brdfs. *Journal of Computer Graphics Techniques*, 3(2):32–91, 2014.

- [14] Stephen Hill, Stephen McAuley, Brent Burley, Danny Chan, Luca Fascione, Michał Iwanicki, Naty Hoffman, Wenzel Jakob, David Neubelt, Angelo Pesce, and Matt Pettineo. Physically based shading in theory and practice. In *ACM SIGGRAPH 2015 Courses*, SIGGRAPH '15, New York, NY, USA, 2015. Association for Computing Machinery.
- [15] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. *SIGGRAPH Comput. Graph.*, 20(4):133–142, aug 1986.
- [16] Mark Jarzynski and Marc Olano. Hash functions for gpu rendering. *Journal of Computer Graphics Techniques (JCGT)*, 9(3):20–38, October 2020.
- [17] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, aug 1986.
- [18] Alexander Keller, Timo Viitanen, Colin Barré-Brisebois, Christoph Schied, and Morgan McGuire. Are we done with ray tracing? In *ACM SIGGRAPH 2019 Courses*, SIGGRAPH '19, New York, NY, USA, 2019. Association for Computing Machinery.
- [19] Jack P. C. Kleijnen, Ad A. N. Ridder, and Reuven Y. Rubinstein. *Variance Reduction Techniques in Monte Carlo Methods*, pages 1598–1610. Springer US, Boston, MA, 2013.
- [20] knightcrawler25. Glsl-pathtracer. <https://github.com/knightcrawler25/GLSL-PathTracer>. Accesat 24.06.2024.
- [21] Eric P. Lafortune and Yves D. Willems. Using the modified phong reflectance model for physically based rendering. Report CW 197, Departement Computerwetenschappen, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium, November 1994.
- [22] Eric P. Lafortune and Yves D. Willems. Rendering participating media with bidirectional path tracing. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, page 91–100, Berlin, Heidelberg, 1996. Springer-Verlag.
- [23] J.H. Lambert. *Photometria sive de mensura et gradibus luminis, colorum et umbrae. sumptibus viduae E. Klett, typis C.P. Detleffsen*, 1760.
- [24] Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. A data-driven reflectance model. *ACM Transactions on Graphics*, 22(3):759–769, July 2003.
- [25] Microsoft. D3d12 raytracing procedural geometry sample. <https://github.com/microsoft/directx-graphics-samples/blob/master/Samples/Desktop/D3D12Raytracing/src/D3D12RaytracingProceduralGeometry/>. Accesat 24.06.2024.
- [26] Fred E. Nicodemus. Directional reflectance and emissivity of an opaque surface. *Appl. Opt.*, 4(7):767–775, Jul 1965.
- [27] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, jun 1975.

- [28] Inigo Quilez. Personal blog. <https://iquilezles.org/>. Accesat 24.06.2024.
- [29] Katja Rogers, Sukran Karaosmanoglu, Maximilian Altmeyer, Ally Suarez, and Lennart E. Nacke. Much realistic, such wow! a systematic literature review of realism in digital games. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.
- [30] Christophe Schlick. An inexpensive brdf model for physically-based rendering. *Computer Graphics Forum*, 13(3):233–246, 1994.
- [31] Joe Schutte. Rendering the moana island scene part 1: Implementing the disney bsdf. <https://schuttejoe.github.io/post/disneybsdf/>. Accesat 24.06.2024.
- [32] B. Smith. Geometrical shadowing of a random rough surface. *IEEE Transactions on Antennas and Propagation*, 15(5):668–671, 1967.
- [33] Gustavo F. Tondello and Lennart E. Nacke. Player characteristics and video game preferences. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '19, page 365–378, New York, NY, USA, 2019. Association for Computing Machinery.
- [34] K. E. Torrance and E. M. Sparrow. Theory for off-specular reflection from roughened surfaces*. *J. Opt. Soc. Am.*, 57(9):1105–1114, Sep 1967.
- [35] Eric Veach. *Robust monte carlo methods for light transport simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162.
- [36] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, page 419–428, New York, NY, USA, 1995. Association for Computing Machinery.
- [37] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07, page 195–206, Goslar, DEU, 2007. Eurographics Association.
- [38] Turner Whitted. An improved illumination model for shaded display. *SIGGRAPH Comput. Graph.*, 13(2):14, aug 1979.

ANEXE

A FIGURI

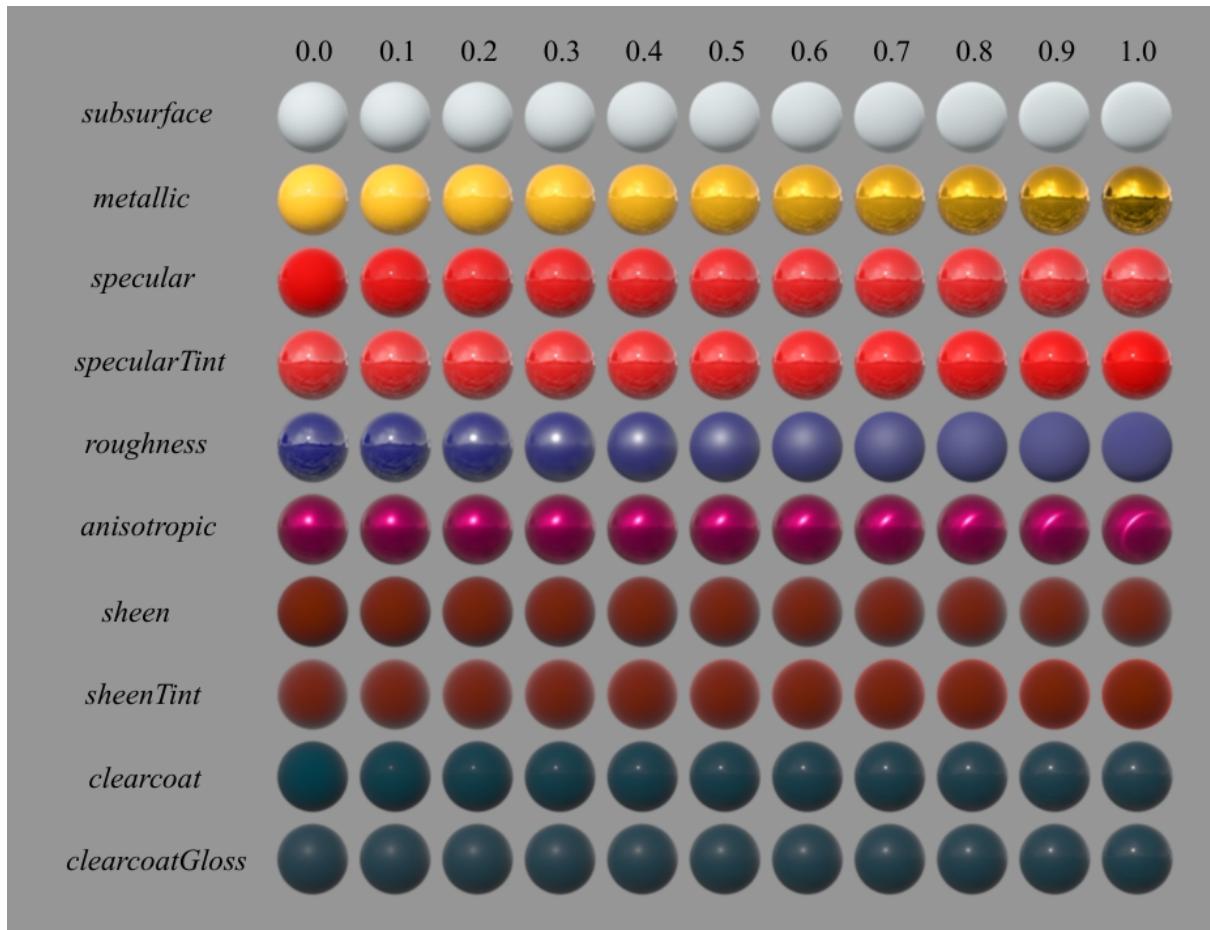


Figura 53: Parametrii modelului Disney Principled BRDF [7]

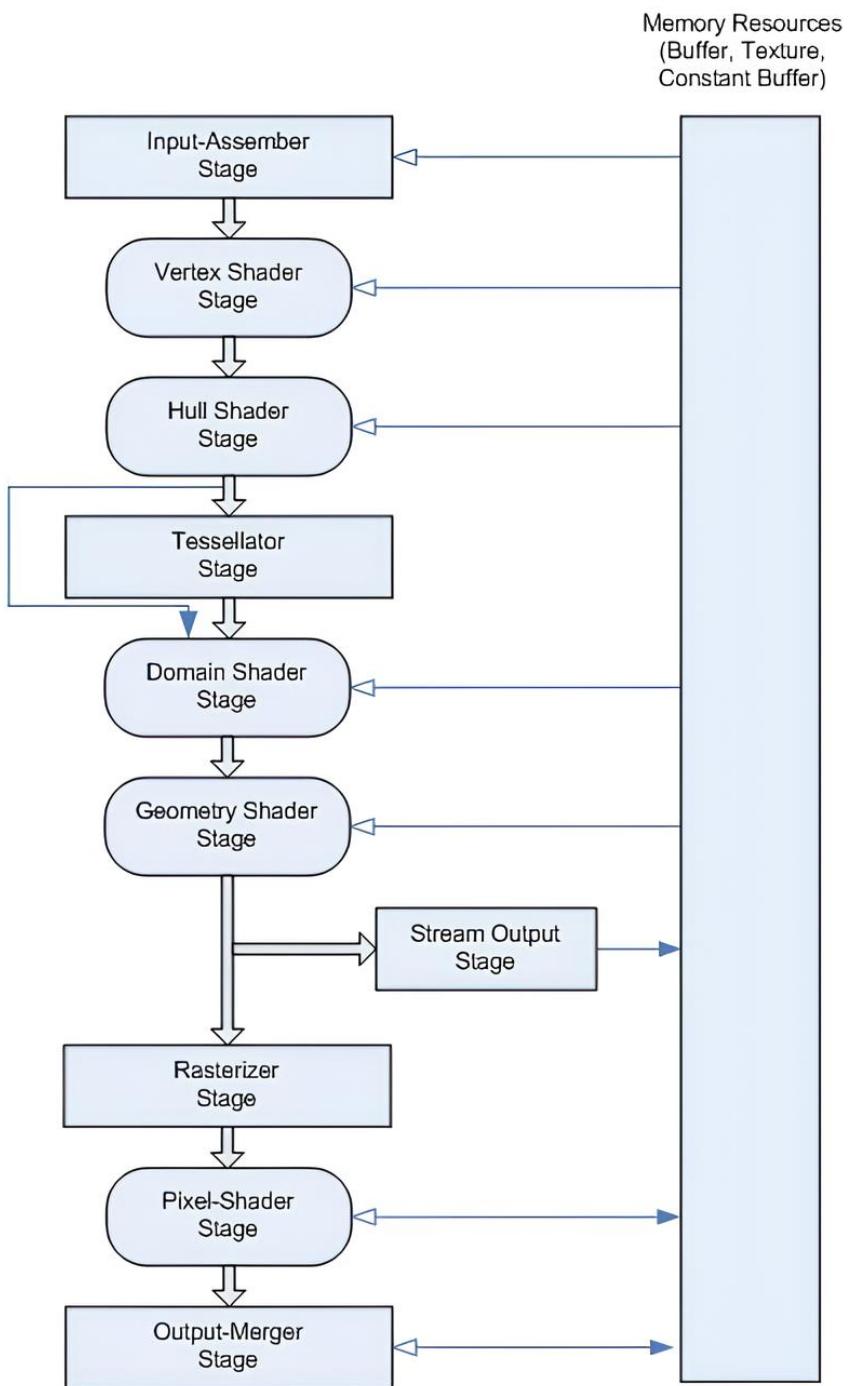


Figura 54: Pipeline-ul de randare Direct3D 11. ©Microsoft, <https://learn.microsoft.com/>. Accesat 19.06.2024.

B EXTRASE DE COD

```
1 void DieKaR::ShowUI()
2 {
3     IM_ASSERT(ImGui::GetCurrentContext() != NULL && "No ImGui context.");
4
5     if (!ImGui::Begin("Settings"))
6         // Don't draw if the window is collapsed.
7         ImGui::End();
8         return;
9 }
10
11 ImGuiIO& io = ImGui::GetIO();
12
13 ImGui::Text("Application average %.3f ms/frame (%.1f FPS)", 1000.0f / io.Framerate, io
14 .Framerate);
15 ImGui::Text("Elapsed time: %.2f (s)", m_sceneCB->elapsedTime);
16 ImGui::Text("Elapsed ticks: %u", m_sceneCB->elapsedTicks);
17 ImGui::Text("Camera position: (%.2f, %.2f, %.2f)", XMVectorGetX(m_eye),
18 XMVectorGetY(m_eye), XMVectorGetZ(m_eye));
19 ImGui::Spacing();
20
21 if (ImGui::CollapsingHeader("Controls"))
22 {
23     ImGui::Spacing();
24
25     ImGui::SeparatorText("Keyboard");
26     ImGui::BulletText("F1 - Reload graphics");
27     ImGui::BulletText("ESC - Exit application");
28     ImGui::BulletText("C - Toggle camera fly/revolution mode");
29     ImGui::BulletText("WASD - Move camera (fly mode)");
30     ImGui::BulletText("EQ - Move camera up/down (fly mode)");
31     ImGui::BulletText("Shift - Move camera slower (fly mode)");
32     ImGui::BulletText("AD - Modify camera speed (revolution mode)");
33     ImGui::BulletText("R - Stop camera (revolution mode)");
34
35     ImGui::SeparatorText("Mouse");
36     ImGui::BulletText("Camera movement available only in fly mode");
37
38     ImGui::Spacing();
39 }
40
41 if (ImGui::CollapsingHeader("Renderer"))
42 {
43     ImGui::Spacing();
44
45     // Jitter
46     ImGui::SetNextItemWidth(ImGui::GetFontSize() * 8);
47     ImGui::Checkbox("Pixel Jitter", &m_applyJitter);
48     ImGui::SameLine(); HelpMarker("Enable pixel jittering for better sampling of the
49     scene");
50
51     // Only one light sample
52     ImGui::SetNextItemWidth(ImGui::GetFontSize() * 16);
53     ImGui::Checkbox("Single LightBuffer Sample", &m_onlyOneLightSample);
54     ImGui::SameLine(); HelpMarker("Whether light sampling should be done one at a time
55     or all at once");
```

```

53
54 // Anisotropic BSDF
55 ImGui::SetNextItemWidth(ImGui::GetFontSize() * 12);
56 ImGui::Checkbox("Anisotropic BSDF", &m_anisotropicBSDF);
57 ImGui::SameLine(); HelpMarker("Enable anisotropy model");
58
59 // Ray Tracing Type
60 const char* options[] = { "Whitted", "Path Tracing", "Progressive Path Tracing" };
61 RaytracingType::Enum prevRaytracingType = m_raytracingType;
62 ImGui::SetNextItemWidth(ImGui::GetFontSize() * 16);
63 if (ImGui::BeginCombo("Raytracing Type", options[m_raytracingType]))
64 {
65     bool selected = m_raytracingType == RaytracingType::Whitted;
66     if (ImGui::Selectable("Whitted", selected))
67         m_raytracingType = RaytracingType::Whitted;
68     if (selected)
69         ImGui::SetItemDefaultFocus();
70
71     selected = m_raytracingType == RaytracingType::PathTracing;
72     if (ImGui::Selectable("Path Tracing", selected))
73         m_raytracingType = RaytracingType::PathTracing;
74     if (selected)
75         ImGui::SetItemDefaultFocus();
76
77     selected = m_raytracingType == RaytracingType::PathTracingTemporal;
78     if (ImGui::Selectable("Progressive Path Tracing", selected))
79         m_raytracingType = RaytracingType::PathTracingTemporal;
80     if (selected)
81         ImGui::SetItemDefaultFocus();
82
83     ImGui::EndCombo();
84 }
85 ImGui::SameLine(); HelpMarker("Select the raytracing type to use");
86
87 // Trigger a graphics reload if the raytracing type has changed.
88 if (prevRaytracingType != m_raytracingType)
89     m_shouldReload = true;
90
91 // Importance Sampling type
92 ImGui::SetNextItemWidth(ImGui::GetFontSize() * 12);
93 const char* importanceSamplingOptions[] = { "Uniform", "Cosine", "BSDF" };
94 if (ImGui::BeginCombo("Importance Sampling", importanceSamplingOptions[
95     m_importanceSamplingType]))
96 {
97     bool selected = m_importanceSamplingType == ImportanceSamplingType::Uniform;
98     if (ImGui::Selectable("Uniform Sphere", selected))
99         m_importanceSamplingType = ImportanceSamplingType::Uniform;
100    if (selected)
101        ImGui::SetItemDefaultFocus();
102
103    selected = m_importanceSamplingType == ImportanceSamplingType::Cosine;
104    if (ImGui::Selectable("Cosine Hemisphere", selected))
105        m_importanceSamplingType = ImportanceSamplingType::Cosine;
106    if (selected)
107        ImGui::SetItemDefaultFocus();
108
109    selected = m_importanceSamplingType == ImportanceSamplingType::BSDF;
110    if (ImGui::Selectable("BSDF", selected))
111        m_importanceSamplingType = ImportanceSamplingType::BSDF;
112    if (selected)
113        ImGui::SetItemDefaultFocus();
114
115    ImGui::EndCombo();
116 }

```

```

116     ImGui::SameLine(); HelpMarker("Select the importance sampling type to use");
117
118     // Sqrt Samples per pixel
119     int maxSqrtSamplesPerPixel = m_raytracingType == RaytracingType::PathTracingTemporal
120         ? 16 : 4;
121     if (m_raytracingType != RaytracingType::PathTracingTemporal)
122         m_pathSqrtSamplesPerPixel = min(m_pathSqrtSamplesPerPixel, 4);
123     UINT oldPathSqrtSamplesPerPixel = m_pathSqrtSamplesPerPixel;
124     ImGui::SetNextItemWidth(ImGui::GetFontSize() * 8);
125     ImGui::SliderInt("Sqrt Samples per pixel", reinterpret_cast<int*>(&m_pathSqrtSamplesPerPixel), 1, maxSqrtSamplesPerPixel);
126     ImGui::SameLine(); HelpMarker("Sqrt of number of samples per pixel");
127     if (oldPathSqrtSamplesPerPixel != m_pathSqrtSamplesPerPixel)
128         ResetPathTracing();
129
130     // Max recursion
131     UINT oldMaxRecursionDepth = m_maxRecursionDepth;
132     ImGui::SetNextItemWidth(ImGui::GetFontSize() * 8);
133     ImGui::SliderInt("Max Recursion Depth", reinterpret_cast<int*>(&m_maxRecursionDepth),
134         1, 10);
135     ImGui::SameLine(); HelpMarker("Maximum recursion depth for path tracing");
136     // we need to reload graphics if this changed (because it was set in stone for
137     // optimization purposes)
138     if (oldMaxRecursionDepth != m_maxRecursionDepth)
139         m_shouldReload = true;
140
141     // Max shadow recursion depth
142     ImGui::SetNextItemWidth(ImGui::GetFontSize() * 8);
143     ImGui::SliderInt("Max Shadow Recursion Depth", reinterpret_cast<int*>(&m_maxShadowRecursionDepth), 1, 10);
144     ImGui::SameLine(); HelpMarker("Maximum recursion depth for shooting shadow rays");
145
146     // Russian Roulette
147     ImGui::SetNextItemWidth(ImGui::GetFontSize() * 8);
148     ImGui::SliderInt("Russian Roulette Depth", reinterpret_cast<int*>(&m_russianRouletteDepth), 1, 10);
149     ImGui::SameLine(); HelpMarker("Depth at which to start Russian Roulette");
150
151     ImGui::Spacing();
152 }
153
154     if (ImGui::CollapsingHeader("Scene"))
155     {
156         ImGui::Spacing();
157
158         // Background color
159         ImGui::SetNextItemWidth(ImGui::GetFontSize() * 16);
160         ImGui::ColorEdit3("Background Color", &m_backgroundColor.x);
161
162         // Lights
163         for (UINT i = 0; i < m_scenes[m_crtScene].GetLightCount(); ++i)
164         {
165             ImGui::PushID(i);
166             ImGui::Text("LIGHT %u", i);
167             ImGui::Spacing();
168
169             // Light type
170             const char* lightTypeOptions[] = { "Area", "Directional" };
171             ImGui::SetNextItemWidth(ImGui::GetFontSize() * 8);
172             if (ImGui::BeginCombo("Type", lightTypeOptions[m_scenes[m_crtScene].m_lights[i].type]))
173             {
174                 for (UINT j = 0; j < LightType::Count; ++j)
175                 {

```

```

173     bool selected = m_scenes[m_crtScene].m_lights[i].type == j;
174     if (ImGui::Selectable(lightTypeOptions[j], selected))
175         m_scenes[m_crtScene].m_lights[i].type = j;
176     if (selected)
177         ImGui::SetItemDefaultFocus();
178     }
179     ImGui::EndCombo();
180 }
181
182 ImGui::SetNextItemWidth(ImGui::GetFontSize() * 16);
183 ImGui::SliderFloat("Size", &m_scenes[m_crtScene].m_lights[i].size, 0.1f, 10.0f);
184
185 if (m_scenes[m_crtScene].m_lights[i].type == LightType::Directional)
186 {
187     ImGui::SetNextItemWidth(ImGui::GetFontSize() * 16);
188     ImGui::SliderFloat3("Direction", &m_scenes[m_crtScene].m_lights[i].direction.x,
189 -1.0f, 1.0f);
190 }
191
192 ImGui::SetNextItemWidth(ImGui::GetFontSize() * 16);
193 ImGui::SliderFloat3("Position", &m_scenes[m_crtScene].m_lights[i].position.x,
194 -20.0f, 20.0f);
195
196 float maxIntensity;
197 if (m_raytracingType == RaytracingType::Whitted || m_scenes[m_crtScene].m_lights[i].
198 .type == LightType::Directional)
199     maxIntensity = 2.0f;
200 else if (m_scenes[m_crtScene].m_lights[i].type == LightType::Square)
201     maxIntensity = 10.0f;
202 m_scenes[m_crtScene].m_lights[i].intensity = min(m_scenes[m_crtScene].m_lights[i].
203 .intensity, maxIntensity);
204
205 ImGui::SetNextItemWidth(ImGui::GetFontSize() * 16);
206 ImGui::SliderFloat("Intensity", &m_scenes[m_crtScene].m_lights[i].intensity, 0.0f,
207 maxIntensity);
208
209 ImGui::SetNextItemWidth(ImGui::GetFontSize() * 16);
210 ImGui::ColorEdit3("Emission", &m_scenes[m_crtScene].m_lights[i].emission.x);
211
212 if (i < m_scenes[m_crtScene].GetLightCount() - 1)
213     ImGui::Spacing();
214     ImGui::Separator();
215     ImGui::Spacing();
216     ImGui::PopID();
217 }
218 }

```

Listarea B.1: Compunerea meniului ImGui

```

1 float3 DoPathTracing(in RayPayload rayPayload, in PBRPrimitiveConstantBuffer material,
2                         in float3 N, in float3 hitPosition, in float hitDistance)
3 {
4     bool inside = dot(WorldRayDirection(), N) > 0.0f;
5     float3 normalSide = inside ? -N : N;
6
7     // Set IOR.
8     float eta = inside ? material.eta : 1.0f / material.eta;
9
10    // Reset absorption if going outside.

```

```

10 float3 absorption = inside ? rayPayload.absorption.xyz : float3(0.0f, 0.0f, 0.0f);
11
12 // Add absorption.
13 float3 throughput = rayPayload.throughput.xyz * exp(-absorption * hitDistance);
14
15 // Initialize the random number generator.
16 uint rng_state = hash(DispatchRaysIndex().xy, g_sceneCB.elapsedTicks + rayPayload.
17   recursionDepth * 1337 + hash(hitPosition));
18
19 //--- Multiple importance sampling.
20 float3 color = float3(0.0f, 0.0f, 0.0f); // Accumulated color
21
22 // 1. Sample direct lighting.
23
24 if (g_sceneCB.onlyOneLightSample)
25 {
26     // If one light at a time, choose randomly and adjust pdf
27     uint idx = random(rng_state) * g_sceneCB.numLights;
28     color += throughput * MIS(rng_state, material, eta, hitPosition, g_lights[idx],
29       normalSide, rayPayload.recursionDepth) * g_sceneCB.numLights;
30 }
31 else
32 {
33     // Sample all lights at once
34     for (uint i = 0; i < g_sceneCB.numLights; i++)
35         color += throughput * MIS(rng_state, material, eta, hitPosition, g_lights[i],
36           normalSide, rayPayload.recursionDepth);
37 }
38
39 // Apply Russian roulette.
40 float russianRoulettePdf = 1.0f;
41 if (rayPayload.recursionDepth >= g_sceneCB.russianRouletteDepth)
42 {
43     russianRoulettePdf = max(throughput.x, max(throughput.y, throughput.z));
44     if (random(rng_state) > russianRoulettePdf)
45         return color;
46 }
47
48 // 2. Sample BSDF.
49
50 float3 reflectance;
51 float3 L;
52 float samplePdf, bsdfPdf;
53
54 // Compute local space.
55 float3 T, B;
56 ComputeLocalSpace(N, T, B);
57
58 if (g_sceneCB.importanceSamplingType == 0)
59 {
60     L = UniformSampleSphere(random(rng_state), random(rng_state), samplePdf);
61     L = normalize(GetTangentToWorld(N, T, B, L));
62     reflectance = EvaluateDisneyBSDF(material, g_sceneCB.anisotropicBSDF, eta, -
63       WorldRayDirection(), L, normalSide, bsdfPdf);
64     bsdfPdf = samplePdf;
65 }
66 else if (g_sceneCB.importanceSamplingType == 1)
67 {
68     L = CosineSampleHemisphere(random(rng_state), random(rng_state), samplePdf);
69     if (random(rng_state) < 0.5f)
70         L = -L;
71     L = normalize(GetTangentToWorld(N, T, B, L));
72     reflectance = EvaluateDisneyBSDF(material, g_sceneCB.anisotropicBSDF, eta, -
73       WorldRayDirection(), L, normalSide, bsdfPdf);

```

```

69     bsdfPdf = samplePdf / 2.0f;
70 }
71 else
72 {
73     reflectance = SampleDisneyBSDF(rng_state, material, eta, g_sceneCB.anisotropicBSDF,
74     -WorldRayDirection(), normalSide, L, bsdfPdf);
75 }
76
77 // Update absorption.
78 if (dot(normalSide, L) < 0.0f)
79     absorption = -log(material.extinction) / material.atDistance;
80
81 // Update throughput and continue only if the pdf is non-zero.
82 if (bsdfPdf > 0.0f) {
83     throughput *= reflectance / bsdfPdf;
84
85     // Apply the Russian roulette pdf.
86     throughput /= russianRoulettePdf;
87
88     // Shoot the next ray and accumulate the color.
89     Ray newRay = {hitPosition, L};
90     color += TraceRadianceRay(newRay, float4(throughput, 1.0f), float4(absorption, 1.0f)
91     , rayPayload.recursionDepth, dot(N, L) < 0.0f).xyz;
92 }
93
94 return color;
95 }
```

Listarea B.2: Algoritmul Path Tracing

```

1 // Credits: https://media.disneyanimation.com/uploads/production/publication\_asset/48/
2 // asset/s2012_pbs_disney_brdf_notes_v3.pdf
3 //
4 // This computes the alpha x and y values for the anisotropic distribution of GTR2.
5 // (the variations of the roughness in the x and y directions in tangent space).
6 void ComputeAnisotropicAlphas(in float roughness, in float anisotropic, out float ax,
7     out float ay)
8 {
9     float aspect = sqrt(1.0f - 0.9f * anisotropic); // limits the aspect ratio to 10:1
10    float roughness2 = roughness * roughness;
11    ax = max(0.0001f, roughness2 / aspect);
12    ay = max(0.0001f, roughness2 * aspect);
13 }
14
15 // Mixes the diffuse and metallic specular components.
16 float DisneyFresnelMix(in float dotLH, in float eta, in float metallic)
17 {
18     float metallicFresnel = FresnelReflectanceSchlick(dotLH);
19     float dielectricFresnel = FresnelDielectric(dotLH, eta);
20     return lerp(dielectricFresnel, metallicFresnel, metallic);
21 }
22
23 // Computes both the sheen color and the diffuse specular color.
24 // It is used in the Fresnel term of the Metal BRDF.
25 void ComputeSpecularColor(in PBRPrimitiveConstantBuffer material, in float eta, out
26     float3 specularColor, out float3 sheenColor)
27 {
28     float luminance = GetLuminance(material.albedo.xyz);
29     float3 tint = luminance > 0.0f ? material.albedo.xyz / luminance : float3(1.0f, 1.0f,
30     1.0f);
31     float3 R0 = ROFromIOR(eta);
32     sheenColor = lerp(float3(1.0f, 1.0f, 1.0f), tint, material.sheenTint);
33     specularColor = lerp(R0 * sheenColor, material.albedo.xyz, material.metallic);
34 }
```

```

31 // Computes the intensity of the clearcoat BRDF lobe.
32 float3 EvaluateClearcoat(in PBRPrimitiveConstantBuffer material, in bool anisotropic, in
33     float3 V, in float3 L, in float3 H, out float pdf)
34 {
35     pdf = 0.0f;
36
37     // Reject if we are below the surface.
38     if (L.y <= 0.0f)
39         return float3(0.0f, 0.0f, 0.0f);
40
41     // Precompute dot products.
42     float dotVH = dot(V, H);
43
44     // Compute anisotropic roughness.
45     float a = lerp(0.1f, 0.001f, material.clearcoatGloss);
46
47     // Compute D term for the clearcoat.
48     float D = DGTR1(H.y, a);
49
50     // Compute F term with IOR = 1.5 and F0 = 0.04.
51     float F = lerp(0.04f, 1.0f, FresnelDielectric(dotVH, 0.6667f));
52
53     // Compute geometric shadowing term G as product of two G1 terms.
54     float G = anisotropic ? (SmithG1Anisotropic(L.x, L.z, L.y, 0.25f, 0.25f) *
55         SmithG1Anisotropic(V.x, V.z, V.y, 0.25f, 0.25f)) : (SmithG1(L.y, 0.25f) * SmithG1(V.
56         y, 0.25f));
57
58     // Compute pdf for sampling.
59     pdf = 0.25f * H.y * D / dotVH;
60 }
61
62 // Disney Diffuse BRDF, with sheen and approximated subsurface scattering.
63 float3 EvaluateDiffuse(in PBRPrimitiveConstantBuffer material, in float3 sheenColor,
64     in float3 V, in float3 L, in float3 H, out float pdf)
65 {
66     pdf = 0.0f;
67
68     // Reject if we are below the surface.
69     if (L.y <= 0.0f)
70         return float3(0.0f, 0.0f, 0.0f);
71
72     // Precompute dot products.
73     float dotHL = dot(H, L);
74
75     // Compute diffuse component.
76     float fd90 = FD90(material.roughness, dotHL);
77     float d = FD(fd90, V.y) * FD(fd90, L.y);
78
79     // Compute subsurface scattering approximation.
80     float fss90 = FSS90(material.roughness, dotHL);
81     float Fss = FD(fss90, V.y) * FD(fss90, L.y);
82     float ss = 1.25f * (Fss * (1.0f / (L.y + V.y) - 0.5f) + 0.5f);
83
84     // Compute the sheen component.
85     float3 sh = material.sheen * FresnelReflectanceSchlick(dotHL) * sheenColor;
86
87     // Compute the pdf.
88     pdf = L.y * INV_PI;
89
90     return (1.0f - material.metallic) * (1.0f - material.specularTransmission) * (INV_PI *

```

```

        material.albedo.xyz * lerp(d, ss, material.subsurface) + sh);
91 }

92

93 // Disney Specular BRDF (only reflection) - standard Cook-Torrance microfacet BRDF.
94 // This is modified to include a dielectric specular term that is missing in the diffuse
95 // model.
96 float3 EvaluateSpecularReflection(in PBRPrimitiveConstantBuffer material, in float eta,
97         in float3 specularColor, in float3 V, in float3 L, in float3 H,
98         in bool anisotropic, in float ax, in float ay, out float pdf)
99 {
100    pdf = 0.0f;
101
102    // Reject if we are below the surface.
103    if (L.y <= 0.0f)
104        return float3(0.0f, 0.0f, 0.0f);
105
106    // Precompute dot products.
107    float dotLH = dot(L, H);
108    float dotVH = dot(V, H);
109
110    // Compute distribution term D.
111    float D = anisotropic ? DGTR2Anisotropic(H.x, H.z, H.y, ax, ay) : DGTR2(H.y, material.
112        roughness);
113
114    // compute Fresnel chromatic component (to account for dielectric specular reflection)
115    float F = lerp(specularColor, float3(1.0f, 1.0f, 1.0f), DisneyFresnelMix(dotLH, eta,
116        material.metallic));
117
118    // compute geometric shadowing term G as product of two G1 terms
119    float Gv = anisotropic ? SmithG1Anisotropic(V.x, V.z, V.y, ax, ay) : SmithG1(V.y,
120        material.roughness);
121    float G = Gv * (anisotropic ? SmithG1Anisotropic(L.x, L.z, L.y, ax, ay) : SmithG1(L.y,
122        material.roughness));
123
124    // Compute pdf.
125    pdf = 0.25f * Gv * max(0.0f, dotVH) * D / (V.y * dotVH);
126
127    return 0.25f * D * F * G / (L.y * V.y);
128 }
129
130 // Disney Specular BSDF (only refraction).
131 float3 EvaluateSpecularTransmission(in PBRPrimitiveConstantBuffer material, in float eta
132         , in float3 V, in float3 L,
133         in float3 H, in bool anisotropic, in float ax, in float ay, out float
134         pdf)
135 {
136    pdf = 0.0f;
137
138    // Reject if we are above the surface.
139    if (L.y >= 0.0f)
140        return float3(0.0f, 0.0f, 0.0f);
141
142    // Precompute dot products.
143    float dotLH = dot(L, H);
144    float dotVH = dot(V, H);

```

```

145 // compute geometric shadowing term G as product of two G1 terms
146 float Gv = anisotropic ? SmithG1Anisotropic(V.x, V.z, V.y, ax, ay) : SmithG1(V.y,
147     material.roughness);
148 float G = Gv * (anisotropic ? SmithG1Anisotropic(L.x, L.z, L.y, ax, ay) : SmithG1(L.y,
149     material.roughness));
150
151 // Compute pdf.
152 pdf = Gv * max(0.0f, dotVH) * D * tmp * abs(dotLH) / V.y;
153
154
155 // Computes the lobes' probability distribution functions for the microfacet model.
156 // Ignores the sheen lobe because its influence is minimal.
157 void ComputePdfs(in PBRPrimitiveConstantBuffer material, in float3 specularColor, in
158     float fresnelMix,
159     out float pSpecularReflection, out float pDiffuse, out float pClearcoat, out
160     float pSpecularRefraction)
161 {
162     float wDiffuse = GetLuminance(material.albedo.xyz) * (1.0f - material.metallic) * (1.0
163         f - material.specularTransmission);
164     float wSpecularReflection = GetLuminance(lerp(specularColor, float3(1, 1, 1),
165         fresnelMix));
166     float wSpecularRefraction = GetLuminance(material.albedo.xyz) * (1.0f - fresnelMix) *
167         material.specularTransmission * (1.0f - material.metallic);
168     float wClearcoat = material.clearcoat * (1.0f - material.metallic);
169
170     float totalWeight = wDiffuse + wSpecularReflection + wClearcoat + wSpecularRefraction;
171 }
172
173 // Computes the Disney BSDF as an aggregate of all the components.
174 float3 EvaluateDisneyBSDF(in PBRPrimitiveConstantBuffer material, in bool anisotropic,
175     in float eta, in float3 V, in float3 L, in float3 N, out float pdf)
176 {
177     pdf = 0.0f;
178     float3 reflectance = float3(0.0f, 0.0f, 0.0f);
179
180     // Transform V and L into the tangent space.
181     float3 T, B;
182     ComputeLocalSpace(N, T, B);
183     V = GetWorldToTangent(N, T, B, V);
184     L = GetWorldToTangent(N, T, B, L);
185
186     // Compute the half vector (with correction for transmission).
187     float3 H;
188     if (L.y >= 0.0f)
189         H = normalize(L + V);
190     else
191         H = normalize(L + V * eta);
192
193     // Correct the half vector if it is below the surface.
194     if (H.y < 0.0f)
195         H = -H;
196
197     // Compute dot products.
198     float dotLH = dot(L, H);
199     float dotVH = dot(V, H);

```

```

200
201 // Compute anisotropic parameters.
202 float ax = 0.0f, ay = 0.0f;
203 if (anisotropic)
204     ComputeAnisotropicAlphas(material.roughness, material.anisotropic, ax, ay);
205
206 // Compute specular and sheen color.
207 float3 specularColor, sheenColor;
208 ComputeSpecularColor(material, eta, specularColor, sheenColor);
209
210 // Compute lobe pdfs.
211 float pDiffuse, pSpecularReflection, pClearcoat, pSpecularRefraction;
212 float fresnelMix = DisneyFresnelMix(dotVH, eta, material.metallic);
213 ComputePdfs(material, specularColor, fresnelMix, pSpecularReflection, pDiffuse,
214             pClearcoat, pSpecularRefraction);
215
216 // Evaluate the lobes.
217 float lobePdf;
218
219 // apply diffuse, sheen and approximate subsurface scattering
220 if (pDiffuse > 0.0f && L.y > 0.0f)
221 {
222     reflectance += EvaluateDiffuse(material, sheenColor, V, L, H, lobePdf);
223     pdf += pDiffuse * lobePdf;
224 }
225
226 // apply specular reflection (only if visible)
227 if (pSpecularReflection > 0.0f && L.y > 0.0f && V.y > 0.0f)
228 {
229     reflectance += EvaluateSpecularReflection(material, eta, specularColor, V, L, H,
230                                                 anisotropic, ax, ay, lobePdf);
231     pdf += pSpecularReflection * lobePdf;
232 }
233
234 // apply clearcoat (only if visible)
235 if (pClearcoat > 0.0f && L.y > 0.0f && V.y > 0.0f)
236 {
237     reflectance += EvaluateClearcoat(material, anisotropic, V, L, H, lobePdf);
238     pdf += pClearcoat * lobePdf;
239 }
240
241 // apply transmission
242 if (pSpecularRefraction > 0.0f && L.y < 0.0f)
243 {
244     reflectance += EvaluateSpecularTransmission(material, eta, V, L, H, anisotropic, ax,
245                                                 ay, lobePdf);
246     pdf += pSpecularRefraction * lobePdf;
247 }
248
249 return reflectance * abs(L.y); // factor in the cosine term
250 }
251
252 float3 SampleDisneyBSDF(inout uint rng_state, in PBRPrimitiveConstantBuffer material, in
253                           float eta,
254                           in bool anisotropic, in float3 V, in float3 N, out float3 L, out float pdf)
255 {
256     pdf = 0.0f;
257     float3 reflectance = float3(0.0f, 0.0f, 0.0f);
258
259     // Transform V into the tangent space.
260     float3 T, B;
261     ComputeLocalSpace(N, T, B);
262     V = GetWorldToTangent(N, T, B, V);

```

```

260 // Compute the specular and sheen color.
261 float3 specularColor, sheenColor;
262 ComputeSpecularColor(material, eta, specularColor, sheenColor);
263
264 // Compute the lobe weights.
265 float pDiffuse, pSpecularReflection, pClearcoat, pSpecularRefraction;
266 float fresnelMix = DisneyFresnelMix(V.y, eta, material.metallic);
267 ComputePdfs(material, specularColor, fresnelMix, pSpecularReflection, pDiffuse,
268             pClearcoat, pSpecularRefraction);
269
270 // Generate random numbers.
271 float eps0 = random(rng_state);
272 float eps1 = random(rng_state);
273 float choice = random(rng_state);
274
275 // Pick one of the lobes to sample.
276 float3 H;
277 float3 cdf;
278 cdf.x = pDiffuse;
279 cdf.y = cdf.x + pSpecularReflection;
280 cdf.z = cdf.y + pClearcoat;
281 if (choice < cdf.x)
282 {
283     // Sample the diffuse lobe with cosine-weighted distribution for outgoing direction.
284     L = CosineSampleHemisphere(eps0, eps1, pdf);
285     H = normalize(V + L);
286
287     reflectance = EvaluateDiffuse(material, sheenColor, V, L, H, pdf);
288     pdf *= pDiffuse;
289 }
290 else if (choice < cdf.y)
291 {
292     // Sample the specular reflection lobe with VNDF for the half vector.
293     float ax = 0.0f, ay = 0.0f;
294     if (anisotropic)
295     {
296         ComputeAnisotropicAlphas(material.roughness, material.anisotropic, ax, ay);
297         H = SampleVNDFAnisotropic(eps0, eps1, ax, ay, V);
298     }
299     else
300         H = SampleVNDF(eps0, eps1, material.roughness, V);
301
302     // Correct the half vector if it is below the surface.
303     if (H.y < 0.0f)
304         H = -H;
305
306     // Reflect the view vector.
307     L = normalize(reflect(-V, H));
308
309     reflectance = EvaluateSpecularReflection(material, eta, specularColor, V, L, H,
310                                              anisotropic, ax, ay, pdf);
311     pdf *= pSpecularReflection;
312 }
313 else if (choice < cdf.z)
314 {
315     // Compute anisotropic roughness.
316     float a = lerp(0.1f, 0.001f, material.clearcoatGloss);
317
318     // Sample the clearcoat lobe by sampling GTR1 distribution.
319     H = SampleDGTR1(eps0, eps1, a);
320
321     // Correct the half vector if it is below the surface.
322     if (H.y < 0.0f)

```

```

322     H = -H;
323
324     // Reflect the view vector.
325     L = normalize(reflect(-V, H));
326
327     reflectance = EvaluateClearcoat(material, anisotropic, V, L, H, pdf);
328     pdf *= pClearcoat;
329 }
330 else
331 {
332     // Sample the specular refraction lobe with VNDF for the half vector.
333     float ax = 0.0f, ay = 0.0f;
334     if (anisotropic)
335     {
336         ComputeAnisotropicAlphas(material.roughness, material.anisotropic, ax, ay);
337         H = SampleVNDFAnisotropic(eps0, eps1, ax, ay, V);
338     }
339     else
340         H = SampleVNDF(eps0, eps1, material.roughness, V);
341
342     // Correct the half vector if it is below the surface.
343     if (H.y < 0.0f)
344         H = -H;
345
346     // Compute the refracted direction.
347     L = refract(-V, H, eta);
348
349     // Check if the refraction is total internal.
350     if (dot(L, L) == 0.0f)
351         L = reflect(-V, H);
352
353     L = normalize(L);
354
355     reflectance = EvaluateSpecularTransmission(material, eta, V, L, H, anisotropic, ax,
356     ay, pdf);
357     pdf *= pSpecularRefraction;
358 }
359
360 // Transform the outgoing direction back to world space.
361 L = GetTangentToWorld(N, T, B, L);
362
363 return reflectance * abs(dot(N, L)); // factor in the cosine term
}

```

Listarea B.3: Evaluarea și eşantionarea BSDF-ului Disney