SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6440

POSTUPAK PRAĆENJA ZRAKE U OKVIRU NVIDIA RTX PLATFORME

Luka Radivoj

Zagreb, lipanj 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 13. ožujka 2019.

ZAVRŠNI ZADATAK br. 6440

Pristupnik: **Luka Radivoj (0036499188)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Postupak praćenja zrake u okviru Nvidia RTX platforme**

Opis zadatka:

Proučiti postupak praćenja zrake. Obratiti pažnju na mogućnosti izvođenja postupka u stvarnom vremenu a posebice na platformi Nvidia RTX uz pripadno programsko sučelje OptiX. Razraditi proučene materijale te ostvariti demonstraciju različitih postupaka i usporediti s ostalim postupcima izrade prikaza. Posebno obratiti pažnju na računalnu kompleksnost pojedinog postupka. Načiniti ocjenu ostvarenih rezultata.
Izraditi odgovarajući programski proizvod, koristiti programski jezik C++. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 14. lipnja 2019.

Mentor:
Prof. dr. sc. Željka Mihajlović

Predsjednik odbora za završni rad modula:
Doc. dr. sc. Marko Čupić

Dielovođa:
Izv. prof. dr. sc. Tomislav Hrkać

Zahvaljujem svojoj majci, Suzani Radivoj, i svome ocu, Tomislavu Radivoju, a posebno svojoj sestri, Lari Radivoj, na nevjerojatnoj podršci za vrijeme pisanja ovog rada.

Sadržaj

1. UVOD

Postupak praćenja zrake jedna je od temeljnih metoda iscrtavanja virtualnih scena. Idejno je postupak veoma jednostavan – za svaki slikovni element ekrana u virtualnu scenu se šalje zraka koja simulira put svijetlosti kroz scenu. Na temelju informacija koje navedena zraka skupi tokom svog puta kroz virtualnu scenu, određuje se boja pripadnog slikovnog elementa. Uz postupak praćenja zrake vežu se dva veoma poznata obilježja. Jedno je činjenica da se navedenim postupkom dobivaju izrazito realistične slike virtualnih scena u kojima su pravilno simulirani fizikalni fenomeni poput refleksije svijetlosti, refrakcije svijetlosti i sličnih. Drugo obilježje je činjenica da je postupak, čak i u svojim najprimitivnijim implementacijama izrazito zahtjevan za izvođenje. Upravo iz tog razloga je postupak ostao rezerviran za primjene koje ne zahtijevaju izvođenje postupka u stvarnom vremenu kroz veći dio razvoja računalne grafike. Zbog toga je svoju primjenu većinom nalazio u izradi animiranih filmova, izradi filmskih efekata i sličnom. No recentnim razvojem tehnologije postalo je moguće virtualna okruženja iscrtavati baš ovim postupkom u stvarnom vremenu. Kroz ovaj završni rad predstaviti će se postupak praćenja zrake iz raznih gledišta. Započet će se objašnjenjem ideje koja stoji iza postupka te kako se on provodi na konceptualnoj razini. Usporedit će se rezultati dobiveni navedenim postupkom s rezultatima drugih postupaka iscrtavanja virtualne scene. Nadalje će se razmotriti mogućnost izvođenja postupka u stvarnom vremenu putem tehnologiju RTX tvrtke Nvidia. Predstavit će se navedeno razvojno okruženje kao i arhitekture grafičkog procesora koje ubrzavaju postupak unutar navedenog okruženja. Usporedit će se rezultati izvođenja postupka sa i bez sklopovskih akceleracijskih struktura. Na samom kraju završnog rada bit će predstavljeno programsko sučelje (engl. *SDK*) OptiX koji služi kao jedno od mogućih sučelja prema RTX tehnologiji. Pomoću navedenog programskog sučelja implementirati će rudimentaran primjer provođenja navedenog postupka.

2. KONCEPT PRAĆENJA ZRAKE

U sklopu ovog poglavlja predstaviti će se elementarni algoritam kojim se izvodi postupak praćenja zrake. U svakom podpoglavlju objasnit će se jedan dio algoritma te će se na kraju predstaviti potpuni pseudokod po kojem se postupak izvodi.

2.1. Generiranje zrake

Početak izvođenja postupka praćenja zrake sastoji se od generiranja zrake koju će se pratiti te njezinog slanja u scenu. Zraka je zapravo pravac kroz dvije točke – točku očišta i točku koja predstavlja slikovni element čiju boju želimo izračunati [1]. Generiranje zrake je neophodan početak bilo koje varijante ovog postupka.

2.2. Pronalazak presjeka

Kada u scenu pustimo zraku za određeni slikovni element, neophodno je pronaći siječe li zraka u pitanju neki od objekata u sceni. Ako zraka ne siječe ni jedan od objekata, kao povratna vrijednost se vraća boja pozadine. U slučaju da postoji jedan ili više presjeka navedene zrake s objektima u sceni, bitno je da postupak pretrage vrati presjek s objektom koji je najbliži slikovnom elementu. Također je bitno da točka presjeka bude ona točka presjeka s objektom koja je također bliža slikovnom element. Ovako izrečen zahtjev je najlakše dočarati na primjeru. Ako zraka siječe sferu u dvije točke, važno je da postupak pronalaska presjeka kao rezultat vrati ono presjecište koje je bliže promatraču, odnosno ono koje predstavlja dio sfere koji je vidljiv iz očišta [1]. Važno je napomenuti da je upravo pronalazak presjeka zrake s objektima u sceni dio postupka praćenja zrake koji je među najzahtjevnijim što se tiče broja operacija potrebnih za njegovo izvođenje i računalne kompleksnosti.

2.3. Izračun lokalnog osvjetljenja

Kada je pronađen presjek zrake i nekog objekta, potrebno je u toj točki presjeka izračunati lokalno osvjetljenje. Izračun lokalnog osvjetljenja nema fiksan postupak. Moguće je koristiti neki od modela lokalnog osvjetljenja poput Phongovog modela lokalnog osvjetljenja [1].

2.4. Sjene

Prije nego što se krene s izračunom lokalnog osvjetljenja pomoću nekog od poznatih modela, bitno je provjeriti nalazi li se točka presjeka u sjeni, odnosno blokira li neki od objekata u sceni put svjetlosti od izvora do navedene točke presjeka. Ako postoji takav objekt, odnosno točka presjeka se nalazi u sjeni, tu činjenicu treba uzeti u obzir pri izračunu vrijednosti modela lokalnog osvjetljenja. Ako za primjer uzmemo Phongov model lokalnog osvjetljenja, činjenica da je točka presjeka u sjeni konkretno znači da će difuzna i spekularna komponenta biti jednake nuli. Važno je napomenuti da upravo o implementaciji metode koja određuje je li točka presjeka u sjeni ili nije ovisi uzima li određena implementacija postupka praćenja zrake u obzir prozirnost. Naime, ako metoda traži presjek s bilo kakvim objektom i pri tome ne uzima u obzir parametre materijala od kojih je objekt sačinjen, tada implementacija postupka u obzir ne uzima mogućnost da su neki od objekata u sceni prozirni. U tom slučaju implementacija postupka daje nekorektnu reprezentaciju objekata koji su sačinjeni od materijala koji su u nekoj mjeri prozirni. Ako u obzir uzmemo da svijetlost prolazi kroz prozirne objekte te da oni ne bacaju sjenu prilikom određivanja je li točka presjeka u sjeni ili nije, tada implementacija postupka dozvoljava uporabu prozirnih objekata te daje realnu reprezentaciju takvih objekata [1].

2.5. Rekurzija

Do sada promatrana zraka, zraka koja kreće iz očišta, prolazi točkom slikovnog elementa i završava u točki presjeka s najbližim objektom, naziva se primarna zraka (engl. *Primary Ray*). Korištenje isključivo primarnih zraka ne daje veoma dobre rezultate. Na njih se može gledati

kao na svojevrsni početak promatranog postupka. Jačina postupka praćenja zrake je u tome da se primarna zraka rekurzivno dijeli unatrag. To znači da iz točke presjeka primarne zrake s najbližim objektom kreiramo nove zrake koje dalje propuštamo u scenu. Tim rekurzivnim pozivima novih zraka u točki presjeka prirodno simuliramo dva fizikalna fenomena – refleksiju i refrakciju svijetlosti. Preciznije, iz točke presjeka u scenu puštamo dvije nove zrake, jednu koja predstavlja reflektiranu zraku i drugu koja predstavlja refraktiranu zraku svijetlosti. Za ove dvije novokreirane zrake provodimo ekvivalentan postupak kao i za primarnu zraku. Prilikom primjene ovog koncepta prirodno se postavljaju dva pitanja. Jedno od tih pitanja je kako za originalno presjecište kombiniramo doprinose lokalnog osvjetljenja, reflektirane i refraktirane zrake. Odgovor na ovo pitanje je varijabilan. Naime, kako se kombiniraju određeni doprinosi ovisi o vrsti materijala od kojeg je sačinjen objekt kojeg zraka siječe. Drugo pitanje koje se prirodno postavlja je pitanje kada zaustaviti rekurziju. Naime, zraka svijetlosti se može reflektirati i refkraktirati gotovo beskonačno mnogo puta unutar scene. No postupak praćenja zrake mora završiti u konačnom vremenu, odnosno u konačnom vremenu mora odrediti boju određenog slikovnog elementa. Određivanju granice za prekid rekurzije se može pristupiti na više načina. Jedan od njih je statički. To znači odrediti tvrdu granicu za dubinu rekurzije nakon koje se prekida rekurzivan proces za svaku primarnu zraku. Prednost ovog pristupa je lakoća implementacije i činjenica da je poznato kada će se rekurzija prekinuti za svaku primarnu zraku. Drugi pristup ovom problemu je ponešto dinamičniji. Rekurziju je moguće prekinuti kada doprinos određene zrake padne ispod neke ranije definirane granice. Prednost ovog rješenja je veća preciznost rezultata, dok je nedostatak činjenica da a priori ne znamo kada će rekurzija za određenu zraku završiti što do dovodi do varijacije u vremenu provođenja postupka za određenu zraku [1].

Koraci 2.1. do 2.5. ponavljaju se za svaki slikovni element. Tako opisan postupak se može predočiti sljedećim pseudo kodom:

Funkcija ***prati_zraku***(zraka, dubina):

Ako je dubina > maksimalna_dubina: vrati crnu boju

Pronađi najbliži presjek zrake i nekog od objekata u sceni

Ako je točka presjeka u sjeni: izračunaj samo ambijentalnu komponentu lokalnog osvjetljenja

Inače: izračunaj ambijentalnu, difuznu i spekularnu komponentu lokalnog osvjetljenja

reflektirana_zraka=***kreiraj_zraku***(presjek, vektor_refleksije)

prati_zraku(reflektirana_zraka, dubina+1)

refraktirana_zraka=***kreiraj_zraku***(presjek, vektor_refrakcije)

prati_zraku(refraktirana_zraka, dubina +1)

Vrati boju kao kombinaciju lokalnog osvjetljenja, doprinosa reflektirane i refraktirane zrake

Funkcija ***iscrtaj_sliku***():

Za svaki slikovni element ekrana:

zraka = ***kreiraj_zraku***(očište, slikovni element)

boja = ***prati_zraku***(zraka, 0)

Iscrtaj slikovni element dobivenom bojom

Važno je napomenuti da je ovdje opisani postupak samo konceptualni algoritam za izvođenje postupka praćenja zrake. Prilikom primjene postupka koristi se jedna od mnogih različitih inačica postupka. Nadalje, postupak u ovakvom obliku kao što je opisan u ovom poglavlju je veoma rudimentaran inačica postupka [2]. Usprkos tome, ta inačica i dalje predstavlja pomak u odnosu na klasičan postupak rasterizacije, no ne simulira u potpunosti vjerno sve fizikalne fenomene koji se vežu uz svijetlost. Prednosti ovako opisanog postupka u odnosu na rasterizaciju je da vjerno simulira fenomene refleksije i refrakcije svjetlosti. U sljedećem poglavlju će se detaljnije razmotriti prednosti postupka praćenja zrake u odnosu na rasterizaciju, no razmotrit će se i nedostaci klasične inačice postupka praćenja zrake te će se predstaviti neke od naprednijih varijanti postupka.

3. KARAKTERISTIKE POSTUPKA

U prošlom poglavlju je predstavljena osnovna inačica postupka praćenja zrake. No, takva rudimentarna verzija postupka nije u mogućnosti pružiti potpunu reprezentaciju fizikalne fenomenologije koja je vezana uz svijetlost i njezino širenje. Takva rudimentarna inačica postupka i dalje generira rezultate koji su fizikalno točniji od onih koje generira postupak rasterizacije. Dva najbitnija parametra po kojem se ocjenjuje neka metoda iscrtavanja u računalnoj grafici su brzina iscrtavanja i kvaliteta iscrtane slike. Generalno gledano, odnos ta dva parametra je obrnuto proporcionalan. Ako se želi postići veća kvaliteta iscrtavanja slike, mora se prihvatiti činjenica da će za iscrtavanje biti potrebno više vremena. S druge strane ako se želi postići velika brzina iscrtavanja, moraju se prihvatiti određeni kompromisi vezani uz kvalitetu iscrtane slike. Opisani odnos ova dva parametra vrijedi i kada se međusobno uspoređuju različite metode iscrtavanja virtualne scene, ali i odnos kvalitete i performansi unutar samih metoda. Kako bi se razumjela važnost mogućnosti izvođenja postupka praćenja zrake u stvarnom vremenu, potrebno je na trenutak se osvrnuti na metodu rasterizacije.

3.1. Metoda rasterizacije

Metoda rasterizacije jedna je od metoda iscrtavanja kojoj je glavna karakteristika brzina iscrtavanja [3]. Naime, metoda rasterizacije proizvodi veoma dobre rezultate, a da se pritom izvodi veoma brzo. Zbog toga se za iscrtavanje virtualnih scena u stvarnom vremenu do današnjeg dana gotovo isključivo koristila rasterizacija. Također, većina grafičkih procesora koja se koriste za iscrtavanje u stvarnom vremenu pogoduje metodi rasterizacije. Metodu je moguće primjenjivati i u primjenama koje se ne izvode u stvarnom vremenu, no u tim primjenama se u današnje vrijeme gotovo uvijek odbacuje u korist drugih metoda koje u takvim uvjetima daju superiornije rezultate. Jedna od najraširenijih primjena iscrtavanja virtualnih scena u stvarnom vremenu su računalne igre. Razvoj rasterizacije kao metode za iscrtavanje u stvarnom vremenu može se usko vezati uz razvoj trodimenzionalnih računalnih igara. Evolucija kvalitete

prikaza virtualnih okruženja u računalnim igrama korelirana je s razvojem i optimizacijom metode rasterizacije. No, kao što je prije napomenuto, problem iscrtavanja kao dijela računalne grafike se uvijek svodi na balansiranje između kvalitete iscrtane slike i brzine iscrtavanja. Stoga tom procesu balansiranja podliježe i metoda rasterizacije. Usprkos godinama razvoja metode, dio simulacije stvarnog svijeta koji se odnosi na simulaciju fenomenologije vezane uz svijetlost i osvjetljenje rasterizaciji predstavlja veliki problem. Rasterizacija kao metoda nije pogodna za realnu simulaciju osvjetljenja zbog toga što prirodno ne podržava tu fenomenologiju. Zbog toga je potrebno prilikom korištenja rasterizacije kao metode iscrtavanja za simulaciju osvjetljenja koristiti razne postupke razvijene tokom godina. Ti postupci se generalno mogu svrstati u dvije skupine:

- Statičke – ovoj skupini pripadaju postupci poput kreacije mapa svijetla (engl. *Light Maps*) i mapa sjena (engl. *Shadow Maps*) koji se provode prilikom kreiranja samog okruženja odnosno scene i služe za simulaciju generalnog osvjetljenja i sjena
- Dinamičke – ovoj skupini pripadaju postupci poput SRR (*Screen Space Refletions*) i SSAO (*Screen space ambient occlusion*) koji pripadaju učincima naknadne obrade te simuliraju određenu dinamičku fenomenologiju vezanu uz svjetlosne pojave, kao i mape sjena (engl. *Shadow Maps*) kada se generiraju u stvarnom vremenu za dinamičke objekte

Važna spoznaja je da su ovi postupci simulacija. Drugim riječima, ovi postupci generiraju rezultate koji su blizu stvarnom svijetu, no ne dolaze bez određenih artefakata i nedostataka. Primjeri nedostataka su neprirodne sjene u određenim situacijama kada se koristi postupak mapa sjena, činjenica da je postupkom SSR nemoguće prikazati refleksiju geometrije koja se ne nalazi na iscrtanoj slici i slično. Također je važno napomenuti da ovi postupci predstavljaju veliki trošak prilikom izrade računalnih igara zbog toga što zahtijevaju da podešavanje njihovih parametara obavlja dizajner koji procjenjuje vjernost dobivenih rezultata određenim postupkom.

Osnovni postupak praćenja zrake prirodno podržava fenomenologiju refleksije svjetlosti, refrakcije svijetlosti i tvrdih sjena upravo zbog toga kako je konceptualno zamišljen. Zbog toga eliminira potrebu za simulacijom navedene fenomenologije te potrebu za procjenom vjernosti rezultata koje te simulacije generiraju. Time predstavlja napredak u generiranju rezultata koji su vjerniji stvarnosti u odnosu na metodu rasterizacije. S druge strane ta prirodna podrška fenomenologije manifestira se kroz smanjenje performansi postupka u odnosu na simulacije kojima se koristi metoda rasterizacije. No, postoje fenomenologije koje osnovni postupak praćenja zrake ne podržava prirodno, već zahtjeva trikove kako bi ih se reprezentiralo. Najbitnije od tih pojava su:

- Ambijentalno zasjenjenje
- Globalno osvjetljenje
- Meke sjene

Ovi fenomeni su izrazito bitni za vjernu reprezentaciju stvarnog svijeta. Upravo zbog manjkavosti osnovnog postupka praćenja zrake, razvijeni su novi postupci koji se baziraju na postupku praćenja zrake. Jedna od tih varijanti postupka praćenja zrake je i postupak praćenja puta (engl. *Path tracing*).

3.2. Postupak praćenja puta (engl. *Path tracing*)

Postupak praćenja puta (engl. *Path tracing*) jedna je od inačica postupka praćenja zrake. Ideja postupka je da se za svaki slikovni element u scenu šalje više različitih zraka te se za svaku zraku, prate svi skokovi zrake kroz scenu sve dok zraka ne dođe do nekog izvora svijetlosti [3]. Konačna boja određenog slikovnog elementa je kombinacija rezultata koji se dobiju praćenjem svake od puštenih zraka za taj slikovni element. Postupak praćenja puta je kao postupak iscrtavanja iznimno računalno zahtjevan za izvođenje. To slijedi iz dvije opservacije. Jedna od njih je stohastičnost puta zrake kroz scenu. A priori nije moguće znati hoće li zraka doći do nekog od izvor i kada. Zbog uvjeta da postupak za svaku zraku mora završiti u konačnom vremenu, moguće je odrediti vremensku granicu nakon koje se prekida postupak. Druga opservacija, odnosno njezina

posljedica, nije na prvi pogled očita. Naime, spomenuto je da se za svaki slikovni element u scenu stohastički šalje veći broj zraka. Svaka zraka vraća informacije koje je prikupila na svom putu kroz scenu. Na kraju postupka kombiniraju se informacije koje su prikupile poslane zrake za određeni slikovni element. Intuitivno je da će kvaliteta rezultata biti veća što je veći broj zraka koji je poslan u scenu za određeni slikovni element. Time se dolazi do drugog razloga zbog kojeg je ovaj postupak veoma spor – za kvalitetne rezultate nužno je postupak provesti za veliki broj zraka po slikovnom elementu. Važno je napomenuti da postupak za jako veliki broj zraka, odnosno uzoraka po slikovnom elementu, generira rezultate koji je gotovo nemoguće razlikovati od fotografija. Upravo ova druga opservacija predstavlja najveći nedostatak ovog postupka. Za mali broj zraka, odnosno uzoraka po slikovnom elementu, postupak generira veoma zrnate slike. S druge strane, ovaj postupak u potpunosti prirodno podržava svu fenomenologiju vezanu uz svijetlost i njezino širenje u prostoru. Upravo zbog toga ova inačica postupka praćenja zrake sebe nalazi kao pretpostavljenom inačicom kada se danas govori o praćenju zrake.

4. TURING ARHITEKTURA GRAFIČKOG PROCESORA

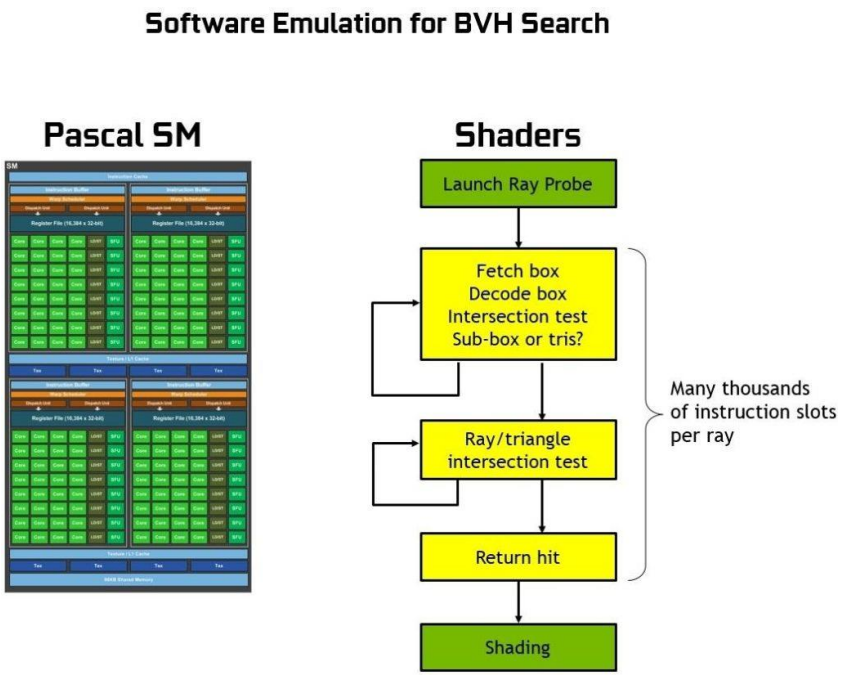
Prilikom rješavanja nekog problema uporabom računala i algoritama, u većini slučajeva prvotno rješenje nije optimalno. Optimalnost rješenja jedan je od bitnih parametara po kojem se ocjenjuje kvaliteta nekog programskog rješenja. Optimizacija jednog od rješenja nekog problema generalno se izvodi programski odnosno algoritamski. Drugim riječima, pokušavaju se pronaći algoritmi koji optimalnije obavljaju neki zadatak. Takve optimizacije su veoma bitne pošto obično za sobom povlače veću brzinu izvođenja nekog programskog rješenja. S druge strane, za dobivanje još boljih performansi računalnih sustava vrše se i sklopovske optimizacije. Sklopovske optimizacije obično se sastoje od promjena u arhitekturi sklopovlja. Promjene u arhitekturi su namijenjene ubrzanju izvođenja nekih algoritama ili operacija koje su česte u primjeni u kojoj se sklopovlje koristi. Sklopovske optimizacije su izrazito dobro vidljive u razvoju grafičkog sklopovlja. Grafičko sklopovlje je sklopovlje

koje je arhitekturom prilagođeno izvođenju operacija koje su česte u računalnoj grafici. U sklopu ovog poglavlja reći će se nešto o sklopovlju koje je primarno namijenjeno trodimenzionalnoj računalnoj grafici u stvarnom vremenu. Kao što je već spomenuto, glavna primjena trodimenzionalne računalne grafike u stvarnom vremenu su računalne igre. Razvoj trodimenzionalnih računalnih igara veže se uz početak 90-ih godina prošlog stoljeća. U tom vremenu sve su se potrebne operacije izvodile na glavnim procesorskim jedinicama, gledano iz perspektive računala opće namjene. Računala namijenjena za izvođenje računalnih igara oduvijek su u sebi sadržavala sklopovske jedinice koje su ubrzavale baratanje grafičkim objektima. Prvu veliku revoluciju u polju trodimenzionalne računalne grafike predstavlja pojava 3D akceleratorskih kartica. One su bile specijalizirano sklopovlje koje je svojom arhitekturom omogućavalo brže izvođenje operacija za koje procesor opće namjene jednostavno nije bio pogodan, a bile su neophodne za trodimenzionalnu računalnu grafiku. Drugu revoluciju u ovom području predstavlja pojava grafičkih kartica s programibilnim sjenčarima. Grafički procesori generalno predstavljaju procesore koji su sposobni provoditi jednostavne operacije nad velikom količinom podataka paralelno. Programibilni sjenčari omogućuju da te operacije nisu fiksne, nego daju mogućnost raznoraznih implementacija određenog dijela grafičkog protočnog sustava. Do sada su arhitekture grafičkih procesora pogodovale metodi rasterizacije. Metoda praćenja zrake se morala programski simulirati unutar sjenčara, odnosno nisu postojale sklopovske arhitekture koje bi bile namijenjene ubrzavanju nekih od kritičnih dijelova navedenog postupka. Nvidia je predstavljanjem svoje Turing arhitekture grafičkog procesora to promijenila. Naime, unutar arhitekture su implementirane sklopovske strukture koje ubrzavaju neke od kritičnih dijelova postupka praćenja zrake [4].

4.1. RT jezgre

Prilikom analize provođenja postupka praćenja zrake dolazi se do spoznaje da je jedan od najkritičnijih dijelova postupka pronalazak presjeka zrake i nekog objekta u sceni. Algoritamska optimizacija ovog dijela postupka je da se svi objekti pa tako i cijela scena organiziraju u

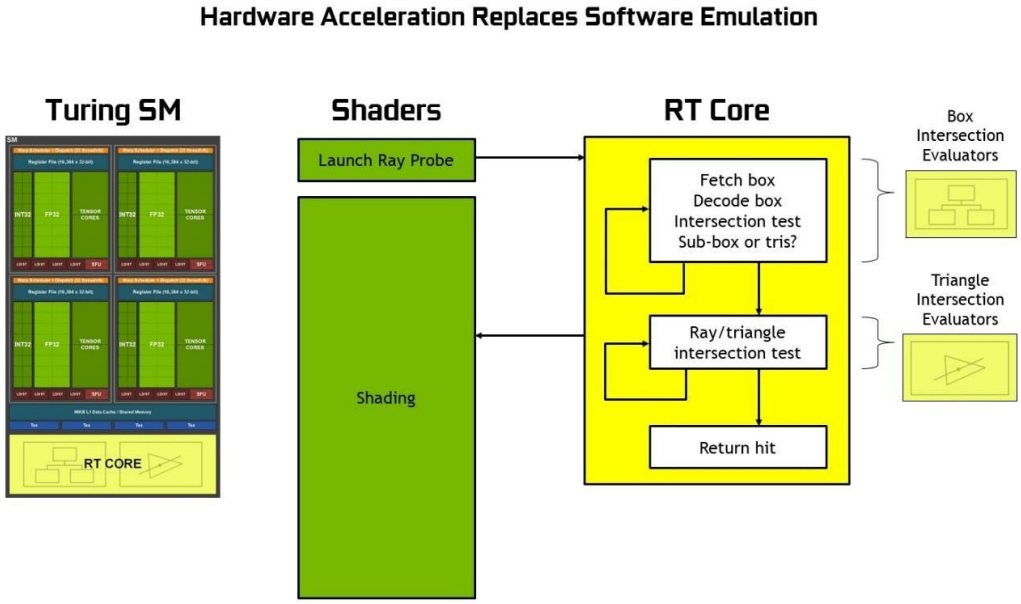
hijerarhijsku strukturu podataka. Najpoznatija od tih hijerarhijskih struktura je BVH (*Bounding Volume Hierarchy*) hijerarhijska struktura obuhvaćajućih volumena. BVH hijerarhijska struktura će detaljnije biti objašnjena kasnije u ovom završnom radu. No, ideja je da je scena podijeljena u progresivno manje dijelove koji su organizirani u stablo. Pronalazak presjeka se u tom slučaju svodi na prolazak kroz tako kreirano stablo kako bi se pronašla primitiva koju zraka siječe te od pronalaska presjeka zrake i primitive. Na prijašnjim arhitekturama grafičkih procesora kompanije Nvidia operacije potrebne za prolazak kroz stablo i pronalaska presjeka s primitivom su se morale izvoditi unutar sjenčara što je zahtijevalo da sjenčari troše na tisuće instrukcija [4]. Slika 4.1.1 prikazuje dijagram toka izvođenja postupka navedenom paradigmom.



Slika 4.1.1: *Pretraga BVH strukture bez sklopovske akceleracije*

Unutar Turing arhitekture postoje posebne sklopovske arhitekture poznate pod nazivom RT jezgre koje su specijalizirane upravo za prolazak kroz ovako kreirana stabla i pronalazak točke presjeka zrake i primitive. RT jezgre je moguće zamisliti kao crnu kutiju koja prima zraku i BVH hijerarhijsku strukturu te kao izlaz daje informaciju o presjeku primljene zrake s primljenom strukturom, odnosno vraća točku presjeka zrake i neke

primitive ako ona postoji. U sklopu Turing arhitekture sjenčari delegiraju posao pronalaska presjeka RT jezgrama. Time se sjenčari oslobađaju velike količine operacija koje su potrebne za pronalazak presjeka pa su za vrijeme dok RT jezgre obavljaju delegirani posao, sjenčari su mogućnosti obavljati neke druge operacije. Slika 4.1.2 prikazuje dijagram toka neposredno navedene paradigme izvođenja.



Slika 4.1.2: *Pretraga BVH strukture sa sklopovskom akceleracijom*

Važno je uočiti da RT jezgre ne razlikuju BVH strukture, odnosno one ne razlikuju nalazi li se u BVH strukturi cijeli scena ili samo neki određeni objekt. Time se dobiva na fleksibilnosti te se interpretacija rezultata ostavlja pozivatelju odnosno programeru koji programira sjenčar. Slika 4.1.3 prikazuje usporedbu brzine izvođenja postupka praćenja zrake na grafičkom procesoru GTX 1080 Ti koji ne posjeduje RT jezgre i RTX 2080 Ti koji posjeduje RT jezgre. Brzina izvođenja na grafičkom procesoru s RT jezgrama je približno 10 puta veća od one na grafičkom procesoru bez RT jezgara.

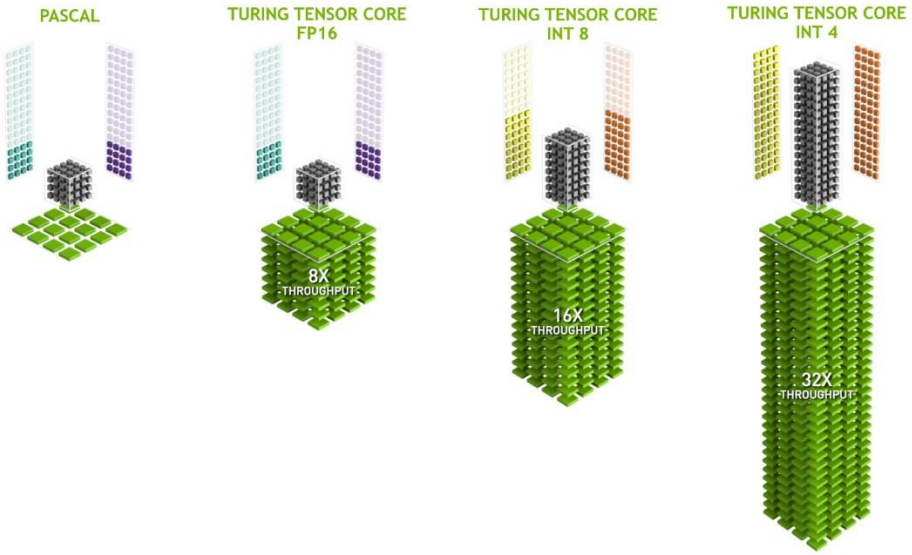
Slika 4.1.3: *Usporedba brzine izvođenja postupka sa i bez RT jezgara*

Kao što je ranije spomenuto, podrazumijevana inačica postupka praćenja zrake danas je neka od implementacija postupka praćenja puta. Ova inačica u potpunosti prirodno podržava svu fenomenologiju vezanu uz svijetlosti i njezino širenje u prostoru. No, glavni nedostatak takvog pristupa postupku praćenja zrake je činjenica da je za kvalitetne rezultate potrebno skupiti veliki broj uzoraka, odnosno za svaki slikovni element je potrebno u scenu poslati veliki broj zraka. Jedno od rješenja ovog problema je sljedeće – za svaki slikovni element se u scenu šalje manji broj zraka te se potom iskustveno uzorci nadopunjuju. Proces se svodi na provođenje slike kroz filter za smanjenje šuma (engl. *Denoising*). Računalno je mnogo manje zahtjevno kreirati sliku na temelju manjeg broja zraka, pa dobivenu sliku provesti kroz proces smanjenja šuma, nego kreirati sliku na temelju velikog broja zraka, odnosno uzoraka. Smanjenje šuma se unutar Turing arhitekture rješava uporabom dubokih neuronskih mreža (engl. *Deep Neural Networks – DNN*).

4.2. Tensor jezgre

Tensor jezgre su sklopovske strukture koje su se prvi put pojavile u sklopu Volta arhitekture grafičkog procesora kompanije Nvidia, no njihovo pojavljivanje kao dio Turing arhitekture predstavlja prvo pojavljivanje na komercijalno dostupnoj arhitekturi grafičkog procesora. Tensor jezgre sklopovski akceleriraju operacije koje su neophodne za implementaciju neuronskih mreža. To su raznorazne operacije nad matricama. Ubrzanje

je ovisno o tome koja vrsta podataka, odnosno preciznost podataka koristi [4]. Slika 4.2.1. pokazuje razliku u ubrzanju provođenja operacija, odnosno u propusnosti Tensor jezgara, u ovisnosti od korištenoj preciznosti i vrsti podataka.



Slika 4.2.1: *Propusnost Tensor jezgara u ovisnosti o vrsti i preciznosti podataka*

5. PRIMJENA POSTUPKA U RAČUNALNIM IGRAMA

Kao što je već ranije spomenuto, jedna od glavnih grana primjene računalne grafike u stvarnom vremenu su računalne igre. Pri izradi računalnih igara prevladavajući postupak iscrtavanja do danas je bio postupak rasterizacije [3]. Već je objašnjeno da je rasterizacija izrazito brz postupak, no njime nije moguće prirodno prikazati određenu fenomenologiju vezanu uz svjetlost, već ju je potrebno simulirati raznoraznim aproksimativnim postupcima. No, bilo bi naivno u potpunosti odbaciti metodu rasterizacije iz dva jednostavna razloga. Jedan od razloga je činjenica da je iscrtavanje izrazito kompleksnih scena, poput onih koje se danas često pojavljuju u računalnim igrama, koristeći isključivo postupak praćenja zrake teško ostvariti u stvarnom vremenu bez obzira na sve algoritamske i sklopovske optimizacije koje su do današnjeg

dana ostvarene. Naime, kako vrijeme prolazi, računalne igre postaju sve kompleksnije. Objekti se sastoje od sve više poligona, kreiraju se scene u kojima se potencira dizajn otvorenog svijeta, pokušava se dočarati sve veći broj fenomena i to u obliku koji je što vjerniji stvarnom svijetu. Drugi razlog je povezan s činjenicom da iscrtavanje nije homogen proces. Naime, postupak iscrtavanja virtualne scene zamišljen je kao protočna struktura ekvivalentno kao i cijeli grafički protočni sustav. Iskustveno se pokazuje da u nekim stadijima iscrtavanja postupak praćenja zrake nema znatnih prednosti u odnosu na metodu rasterizacije što se tiče kvalitete rezultata. Štoviše, pokazuje se da je jednostavno bolje koristiti metodu rasterizacije zbog toga što daje jednako dobre rezultate dok pri tome koristi znatno manje računalnih resursa, što zbog same prirode metode, što zbog mnogobrojnih optimizacija koje su provedene nad postupkom kroz godine. Zbog toga tvrtka Nvidia u sklopu svoje RTX platforme preporuča korištenje takozvanog hibridnog iscrtavanja za iscrtavanje okruženja računalnih igara.

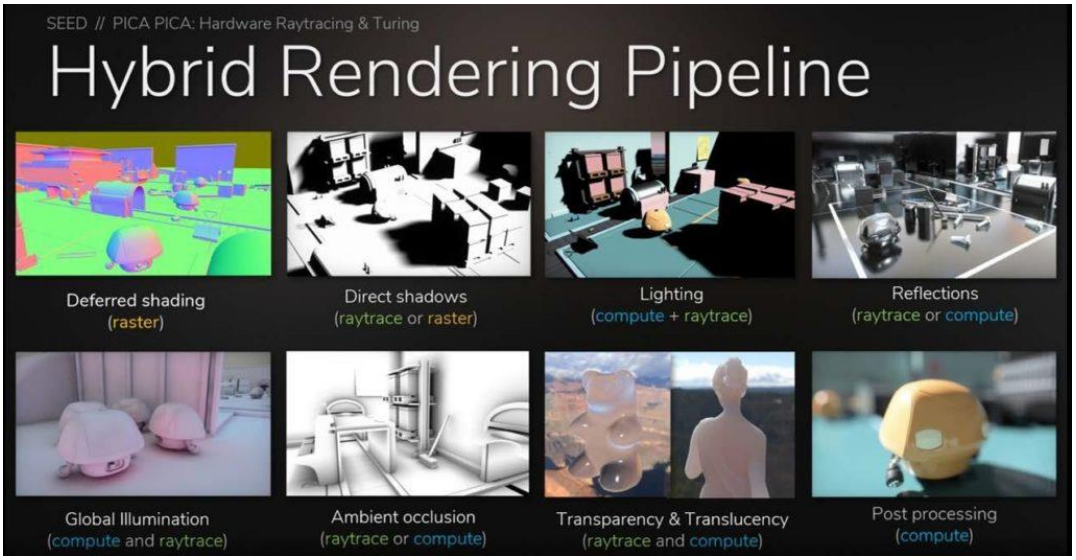
5.1. Hibridno iscrtavanje

Hibridno iscrtavanje je pristup iscrtavanju virtualnih scena koji dolazi prirodno iz razmatranja iznesenih ranije u poglavlju [4]. Koncept je jednostavan. Pošto se iscrtavanje sastoji od različitih faza, te pošto i rasterizacija i postupak praćenja zrake u nekim od tih faza daju superiornije rezultate, ili daju jednake dok je pritom jedan brži, iscrtavanju se pristupa tako da se dio faza iscrtavanja izvode postupkom rasterizacije, dok se drugi dio tih faza izvodi postupkom praćenja zrake. U fazama u kojima oba postupka proizvode slične rezultate sa sličnim vremenom izvođenja, odabir postupka je proizvoljan te eventualno ovisi o umjetničkim i dizajnerskim afinitetima. Ovakvim pristupom omogućava se povećanje kvalitete slike koju povlači primjena postupka praćenja zrake bez velikog gubitka performansi. Važno je napomenuti da su unutar Turing arhitekture grafičkog procesora protočne strukture za rasterizaciju i za postupak praćenja zrake odvojene te se mogu simultano koristiti što rezultira hibridnim pristupom iscrtavanju ili je moguće koristiti isključivo jednu, ovisno o željenoj kombinaciji performansi i kvalitete slike. Na slici 5.1.1

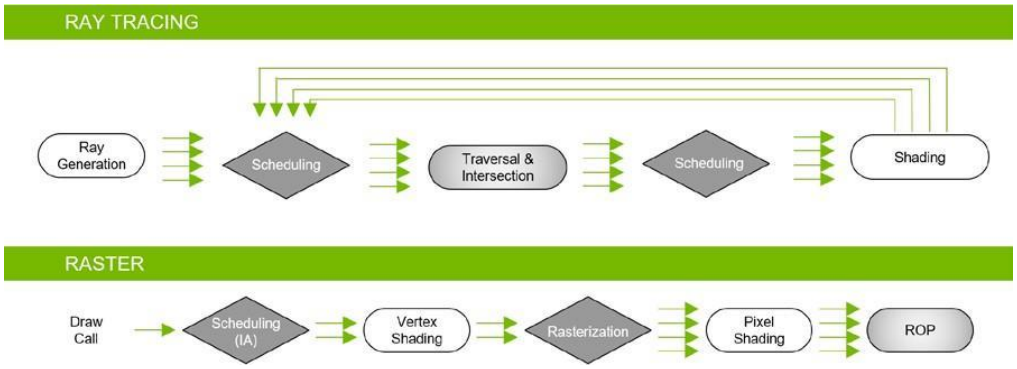
prikazane su navedene protočne strukture. Važno je napomenuti da hibridni pristup nije fiksan, odnosno programer ima potpunu slobodu odabrati koji će postupak koristiti za koju fazu iscrtavanja. Na slici 5.1.2. prikazane su faze procesa iscrtavanja virtualne scene te preporučena metoda iscrtavanja. Sa slike je vidljivo da je rasterizacija preporučena metoda za, na primjer, određivanje vidljivosti pomoću Z-spremnika. Time rasterizacija može zamijeniti korištenje primarnih zraka za određivanje vidljivosti. Praćenje zrake se koristi u fazama u kojima daje nedvojbeno superiornije rezultate u odnosu na rasterizaciju. To su faze kojima se postižu sljedeći efekti:

- refleksije, refrakcije i sjene
- ambijentalno zasjenjenje
- globalno osvjetljenje

Ove faze su ekvivalentne fazama puštanja sekundarnih zraka iz točke presjeka kada se za iscrtavanje koristi samo postupak praćenja zrake.



Slika 5.1.1: *Faze procesa iscrtavanja i preporučene metode*



Slika 5.1.2: *Protočne strukture rastera i praćenja zrake*

U nastavku će se napraviti detaljniji pregled efekata koje postupak praćenja zrake i njegova inačica praćenje puta prirodno podržavaju te koje su prednosti u odnosu na izvedbu tih istih efekata pomoću aproksimativnih postupaka pri iscrtavanju metodom rasterizacije.

5.2. Refleksije

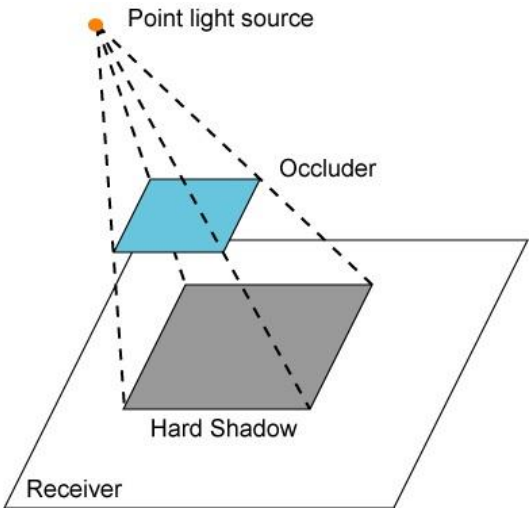
Kao što je već ranije spomenuto, rasterizacija kao metoda iscrtavanja prirodno ne podržava fenomen refleksije svjetlosti. Jedan od najpoznatijih i široko raširenih postupaka za simuliranje refleksija prilikom korištenja metode rasterizacije je SSR (*Screen Space Reflections*). Jedan od glavnih nedostataka ovog postupka je nemogućnost reflektiranja geometrije, odnosno objekata koji se ne nalaze na iscrtanoj slici. To uključuje i objekte koji se nalaze izvan projekcijskog volumena kamere, objekte koji su djelomično ili u potpunosti zakriveni drugim objektima i slično. Metoda praćenja zrake prirodno podržava fizikalno ispravne refleksije. Također rješava najveći problem SSRa zbog toga što se, za razliku od SSRa, ne oslanja na trenutno iscrtanu sliku scene nego u obzir uzima cijelu geometriju scene neovisno o tome koji dio scene se nalazi na trenutno iscrtanoj slici. Jedan od pristupa implementaciji postupka praćenja zrake za prikaz refleksija vidljiv je u računalnoj igri Battlefield 5. Razvojni tim je metodu praćenja zrake implementirao tako da za svaki slikovni element na kojem se nalazi neka reflektirajuća površina, pošalju u scenu jednu zraku kako bi pronašla objekt koji se reflektira u tom slikovnom elementu [3]. Za bolje rezultate moguće je dopustiti više skokova poslane zrake na

njezinom putu kroz virtualnu scenu. Slika 5.2.1 prikazuje scenu s reflektirajućom površinom na kojoj se reflektira okolna geometrija.

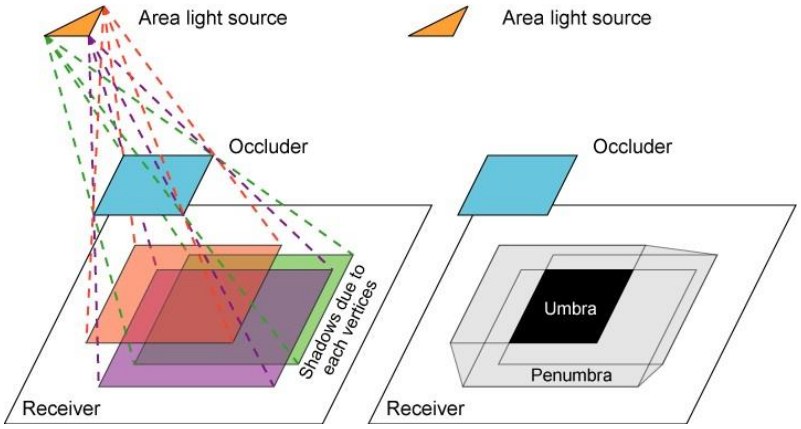
Slika 5.2.1: *Scena s reflektirajućom površinom*

5.3. Sjene

Vjerna reprezentacija sjena oduvijek je predstavljala problematično područje pri iscrtavanju virtualnih scena [3]. Rasterizacija kao metoda iscrtavanja je potpuno neadekvatno opremljena za ovaj zadatak. Rješenje koje je danas općeprihvaćeno prilikom korištenja metode rasterizacije za iscrtavanje je korištenje mapa sjena (engl. *Shadow maps)*. Takva metoda daje relativno tvrde sjene koje ne reagiraju dinamički na različite izvore svijetlosti. Korištenjem postupka praćenja zrake za reprezentaciju sjena dobivaju se mnogo vjerniji rezultati nego kada se koriste mape sjena i rasterizacija. No osnovni postupci praćenja zrake u svojoj suštini ne podržavaju prirodno sve vrste sjena. Generalno kada se govori o sjenama, u grubo se razlikuje dvije vrste – tvrde i meke sjene. Tvrde sjene se javljaju kada se tijelo nalazi na putu svijetlosti točkastog izvora. No u stvarnosti je rijetko slučaj da je izvor svijetla točkasti. U stvarnosti većina izvora svjetlosti ima neku površinu. Takvi izvori ne proizvode tvrde već, u manjoj ili većoj mjeri, meke sjene [5]. Na slici 5.3.1. prikazana je tvrda sjena, dok je na slici 5.3.2. prikazana meka sjena. Meka sjena se sastoji od dva dijela – umbre i preumbre. Umbra je oštar dio sjene, dok preumbra predstavlja meki dio sjene.



Slika 5.3.1: *Nastanak tvrde sjene*

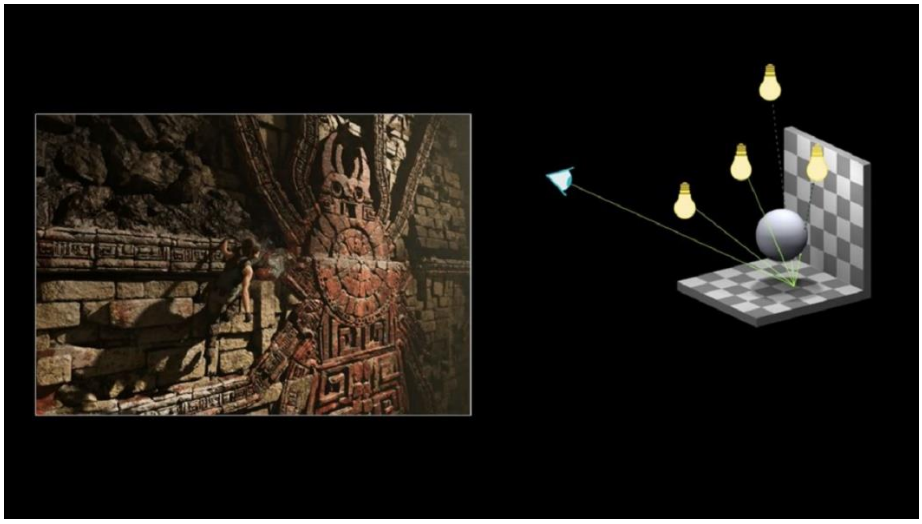


Slika 5.3.2: *Nastanak meke sjene*

Postupak praćenja zrake u osnovnim implementacijama kakva je opisana u drugom poglavlju, prirodno podržavaju isključivo tvrde, oštre sjene. To je zbog toga što tako opisan postupak izvor smatra točkastim. Upravo ovo predstavlja jedno od područja u kojoj osnovni postupak ne proizvodi zadovoljavajuće rezultate. Postupak praćenja puta, kao varijanta postupka praćenja zrake, prirodno podržava meke sjene zbog toga što uzorkuje mnogo različitih zraka, koje putuju stohastički kroz scenu, te tako ne tretira izvore kao točkaste ako to nisu. Već je spomenuto da postupak praćenja puta zahtjeva veliki broj uzoraka za kvalitetne rezultate. Ako se ne skupi dovoljno uzoraka, rezultat je zrnat. Isto vrijedi ako se navedeni postupak koristi za generiranje sjena. No, jedna od velikih prednosti RTX platforme je njezina sposobnost uklanjanja šuma iz rezultantnih slika. Tako se i

sjene generirane postupkom praćenja puta mogu generirati s malim brojem uzoraka te provesti kroz već spomenuti filter. Ovo predstavlja jedno od rješenja, odnosno proširenja osnovnog postupka. Naime, izvori se više ne tretiraju kao točkasti ako to nisu prilikom izračuna sjena. U obzir se također uzima i udaljenost izvora od predmeta, intenzitet izvora, vrsta izvora i slično. Upravo je ovo rješenje koje je implementirao razvojni tim računalne igre Shadow of the Tomb Raider (*Eidos Montreal*). Rezultat je prikazan na slici 5.3.3. Sjene na lijevoj strani su generirane pomoću mapa sjena, dok su sjene na desnoj strani slike generirane internom implementacijom postupka praćenja zrake unutar RTX platforme. Vidljivo je da su sjene na desnoj strani mnogo mekše i realističnije. Važno je napomenuti da implementacija izračuna sjena ovisi o razvojnom timu te RTX platforma dopušta mnoga različita rješenja koja mogu dati različiti omjer kvalitete i performansi. Slika 5.3.4 prikazuje koncept generiranja sjeni pomoću postupka praćenja zrake u sklopu RTX platforme.

Slika 5.3.3: *Implementacije sjena u računalnoj igre Shadow of the Tomb Raider*

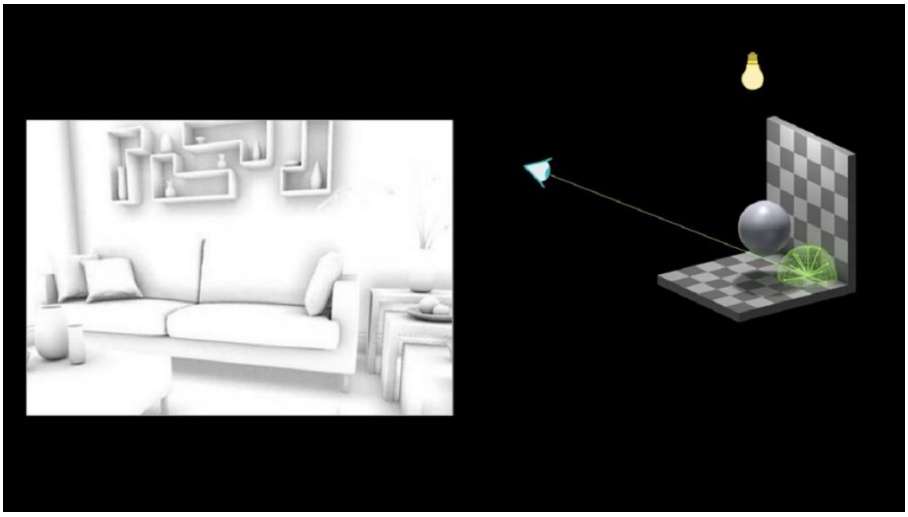


Slika 5.3.4: *Koncept generiranja sjena*

5.4. Ambijentalno zasjenjenje

Ambijentalno zasjenjenje (engl. *Ambient Occlusion*) je efekt u računalnoj grafici kojim se pokušava prikazati fenomenologija pojave tamnijih dijelova prostora kao rezultat sjena koje bacaju lokalni objekti [3]. Generalno rečeno će dijelovi prostora biti progresivno tamniji što je više objekata u blizini tog istog dijela prostora. Drugim riječima, želi se prikazati učinak ambijentalnog svijetla koje je gruba aproksimacija, odnosno simulacija, globalnog osvjetljenja u sceni [6]. Ambijentalno zasjenjenje je efekt kojeg rasterizacija ne podržava prirodno. Postupak kojim se simulira ambijentalno zasjenjenje prilikom korištenja rasterizacije je SSAO (*Screen Space Ambient Occlusion*). SSAO je efekt naknadne obrade koji koristi informacije već iscrtane slike. Također ga ne podržava ni osnovni postupak praćenja zrake. Lako je uočiti da osnovni postupak praćenja zrake ambijentalno zasjenjenje simulira unutar modela lokalnog osvjetljenja što nije poželjno zbog toga što je ambijentalno zasjenjenje globalna pojava odnosno ovisi o svoj geometriji koja se nalazi u sceni. Generiranje ambijentalnog zasjenjenja u sklopu RTX platforme se sastoji od toga da se iz neke točke stohaistički šalje velik broj zraka za koje se traži presjek s lokalnim objektima. Ovaj postupak također daje aproksimativne rezultate u odnosu na postupak praćenja puta koji ambijentalno zasjenjenje podržava prirodno. Generiranje ambijentalnog zasjenjenja pomoću postupka praćenja zrake je jeftinije u pogledu

računalne zahtjevnosti u usporedbi s drugim primjenama postupka pošto je potrebno testirati zraku do prvog presjeka te se za presjeke testiraju samo objekti, odnosno BVH strukture, koje se nalaze u neposrednoj blizini promatrane točke. Slika 5.4.1 prikazuje koncept generiranja ambijentalnog zasjenjenja i njegov efekt na primjeru scene u kojoj se ne primjenjuje ni jedan drugi efekt ni postupak osim ambijentalnog zasjenjenja. Vidljivo je da se dijelovi prostora u čijoj se blizini nalazi veći broj objekata prikazuju tamnijima u odnosu na one u čijoj blizini je manji broj objekata.

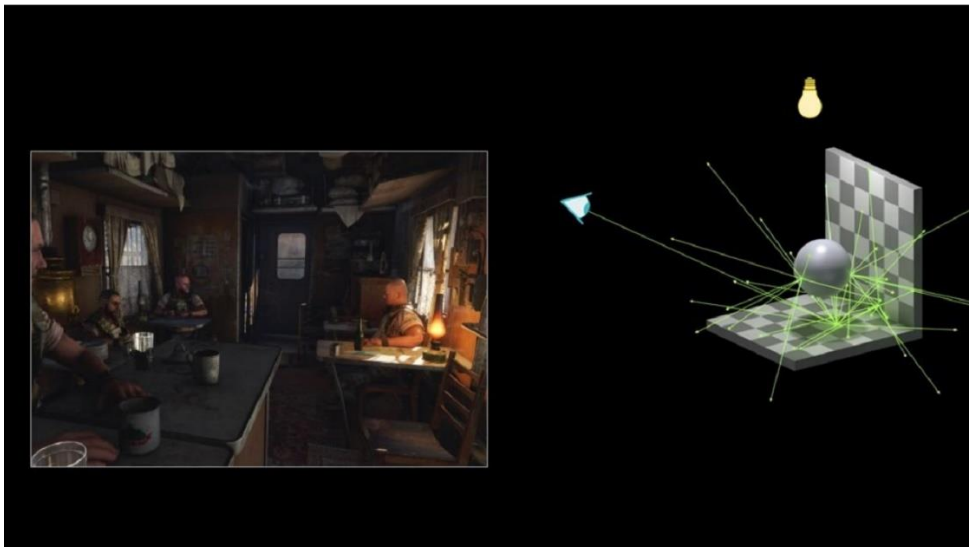


Slika 5.4.1: *Prikaz ambijentalnog zasjenjenja*

5.5. Globalno osvjetljenje

Detaljan opis globalnog osvjetljenja tema je koja daleko nadilazi granice ovo završnog rada. Bitno je samo napomenuti da se rješavanje problema simulacije globalnog modela osvjetljenja svodi na rješavanje jednadžbe iscrtavanja koja u potpunosti prirodno opisuje fizikalno ponašanje svjetlosti u bilo kojoj točki prostora [7]. Rješavanje ove jednadžbe u stvarnom vremenu je gotovo nemoguće te se zbog toga provode razne aproksimacije ovog modela. Jedna od tih aproksimacija je i već spomenuto ambijentalno zasjenjenje koje predstavlja jednu od mogućih aproksimacija globalnog osvjetljenja u sceni. Ambijentalno zasjenjenje efekte globalnog osvjetljenja aproksimira učincima ambijentalnog svijetla u sceni. Druga aproksimacija kojom se žele aproksimirati učinci globalnog osvjetljenja je GI (eng. *Global Illumination*). GI je u svojoj suštini numerička

aproksimacija rješenja jednadžbe iscrtavanja. Čak i kao aproksimativna metoda u sklopu metode rasterizacije, GI je veoma računalno kompleksan i zahtjevan postupak. Postupak praćenja zrake također su svojim implementacijama koje nisu naprednije varijante postupka praćenja puta prirodno ne podržava globalan model osvjetljenja. No aproksimativni rezultati dobiveni postupkom praćenja zrake su kvalitetniji i uvjerljiviji od rezultata dobivenih implementacijama GI u sklopu metode rasterizacije [3]. Simulacija efekata globalnog osvjetljenja u sklopu RTX platforme se svodi na implementaciju jednostavnije varijante postupka praćenja puta koja za svaki slikovni element u scenu šalje zraku čiji put prati. Slika 5.5.1 prikazuje koncept navedenog postupka unutar RTX platforme.



Slika 5.5.1: *Koncept globalnog osvjetljenja*

Rezultati ovog postupka su drastični, usprkos tome što su u svojoj suštini i dalje aproksimativni. Ako se simulacija globalnog osvjetljenja primjeni na scene koje prikazuju, na primjer, neku zatvorenu prostoriju u kojoj je izvor svjetlosti neki manji prozor, scena postaje mnogo svjetlija i bliža stvarnom svijetu. Slika 5.5.2 prikazuje jednu takvu scenu. Lijevi dio slike prikazuje scenu iscrtanu bez ovakve aproksimacije modela globalnog osvjetljenja, dok desni dio slike prikazuje scenu koja je iscrtana koristeći ovako opisanu aproksimaciju modela globalnog osvjetljenja. Važno je spomenuti performanse. Od svih navedenih postupaka unutar RTX platforme,

postupak simulacije globalnog osvjetljenja na ovaj način je računalno daleko najzahtjevniji zbog toga što se u suštini provodi cijeli postupak praćenja zrake u sceni.

Slika 5.5.2: *Scena bez i s uporabom globalnog osvjetljenja*

5.6. Kaustike

Kaustike su vizualni efekt koji se javlja kada svijetlost dolazi na zakrivljenu plohu te se reflektira od nje [3]. Najpoznatiji primjer kaustika su kaustike koje nastaju prilikom refleksije svijetla na površini vode. Prikaz kaustika je samo jedna od mogućih primjena postupka generiranja refleksija. No efekt je dovoljno zanimljiv da bi ga se izdvojilo. Kaustike se generiraju na ekvivalentan način kao i normalne refleksije, samo što se u ovom slučaju ne prikazuje smo geometrija na reflektivnoj površini nego su i linije nastale zbog zakrivljenosti reflektivne površine vidljive na geometriji. Slika 5.6.1 prikazuje istu scenu u kojoj su na desnoj strani kaustike projicirane na okolnu geometriju dok s lijeve strane nisu. Proces praćenja zrake se može koristiti i za prikaz površinskih kaustika, koje su vidljive na slici 5.6.1, ali i za prikaz volumnih kaustika. Volumne kaustike nastaju kada se svijetlost refraktira na zakrivljenoj površini te putuje kroz takav materijal na čijem se kraju ponovo reflektira. Takva fenomenologija je vidljiva na površinama koje predstavljaju dno nekog tijela vode.

Slika 5.6.1: *Virtualna scena bez i s kaustikama*

6. RTX PLATFORMA

RTX platforma je platforma koju je razvila tvrtka Nvidia. Služi kao nativna platforma za rad s postupkom praćenja zrake i njegovima varijantama na arhitekturama grafičkih procesora iste kompanije. RTX platformu podržavaju sve arhitekture grafičkih procesora tvrtke Nvidia koje dolaze poslije arhitekture Maxwell uključujući i nju. No prava moć RTX platforme dolazi do izražaja na arhitekturama Volta i Turing. Te dvije arhitekture, kao što je već spomenuto, sadrže sklopovske strukture koje ubrzavaju općekorištene algoritme u procesu praćenja zrake. Te arhitekture ubrzavaju dva ključna algoritma koji omogućuju izvođenje ovog procesa u stvarnom vremenu.

6.1. Hijerarhija obuhvaćajućih volumena

Kao što je već par puta istaknuto tokom ovog završnog rada, proces pronalaska presjeka zrake i objekata u virtualnoj scenu jedan je od glavnih zadataka prilikom izvođenja postupka praćenja zrake. Već je ranije predstavljena sklopovska akceleracija ovog postupka. Ovdje će se detaljnije razmotriti algoritamska akceleracija odnosno struktura podataka koju kao ulaz primaju RT jezgre kako bi odredile presjek zrake i objekta. Ta struktura podataka je hijerarhija obuhvaćajućih volumena BVH

(*Bounding Volume Hierarchy*) [3]. BVH struktura je hijerarhijska struktura koja se sastoji od obuhvaćajućih volumena (engl. *Bounding Volumes*) [8]. Odabir obuhvaćajućeg volumena koji će se koristiti prilikom izrade ovakve strukture nije jedinstven. Jedan od najčešće korištenih obuhvaćajućih volumena je minimalna, osno-poravnata obuhvaćajuća kutija (engl. *AABB – Axis-aligned minimum bounding box*). AABB imaju razna poželjna svojstva poput lake izračunljivosti, malog zauzeća memorije te jednostavnosti implementacije pronalaska presjeka i brzine navedenog postupka. BVH strukture su izvrsna generalizacija zapisa objekata za razne operacije u kojima nam problem predstavlja raznolikost oblika koje objekti poprimaju. BVH strukture se dugi niz godina koriste za detekciju kolizije objekata. Svoju drugu veliku primjenu danas pronalaze u postupku praćenja zrake. Svaki objekt u sceni se reprezentira ovakvom strukturom. Postupak kreiranja BVH strukture za određenih objekt nije jedinstven, odnosno objekt se može u strukturu prevesti na različite načine. Konstrukcija BVH strukture se svodi na konstrukciju stabla zbog toga što je prirodno ovakvu hijerarhijsku strukturu predstaviti stablom. Korijen ovog stabla je obuhvaćajući volumen koji obuhvaća cijeli objekt, čvorovi su obuhvaćajući volumeni koji sadrže dio objekta, dok su listovi zapravo obuhvaćajući volumeni koji obuhvaćaju primitive od kojih je objekt sastavljen. Načini konstruiranja ovakve strukture, odnosno stabla, mogu se svrstati u sljedeće tri kategorije:

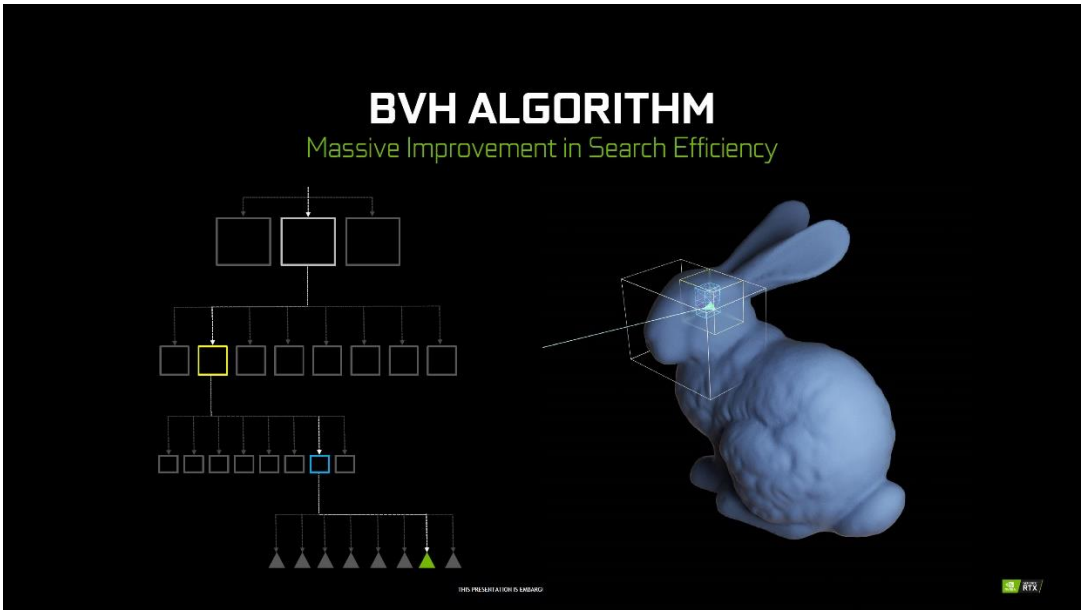
- Pristup konstrukcije od korijena prema listovima (engl. *Top-down methods*)
- Pristup konstrukciji od listova prema korijenu (engl. *Bottom-up methods*)
- Pristup konstrukciji umetanjem (engl. *Insretion methods*)

Prve dvije kategorije metoda spadaju u takozvane *off-line* metode konstrukcije zbog toga što svi elementi stabla moraju biti unaprijed poznati. Zadnja kategorija pripada takozvanim *online* metodama jer se elementi umeću u stablo jedan po jedan i ne moraju biti unaprijed poznati. Konstrukcija od korijena prema listovima provodi se tako da se objekt kontinuirano dijeli sve dok se unutar obuhvaćajućih volumena ne nalazi

samo po jedna primitiva. Konstrukcija od listova prema korijenu provodi se tako da se sve primitive smještaju u vlastite obuhvaćajuće volumene koji se onda kontinuirano spajaju u veće obuhvaćajuće volumene sve dok jedan volumen ne obuhvaća cijeli objekt. Konačna struktura stabla je za oba postupka konceptualno identična – svako od dvoje djece predstavlja određeni dio roditeljskog čvora. Kvaliteta konstruirane BVH strukture se može odrediti prema tome zadovoljava li određeni skup poželjnih svojstava. Neka od tih svojstava su:

- Čvorovi koji se nalaze u određenom podstablu moraju biti blizu jedni drugih – što se dublje u hijerarhijskoj strukturi nalazimo čvorovi trebaju međusobno biti sve bliži
- Svaki čvor mora biti minimalnog volumena
- Suma volumena svih čvorova treba biti minimalna
- Volumen u kojem se susjedni čvorovi preklapaju treba biti minimalan

Iskustveno se pokazuje da su binarna stabla najpogodnija vrsta stabala za reprezentaciju ovakve hijerarhijske strukture. Razlog tome je jednostavan – binarna stabla je lakše konstruirati te imaju mnoga druga poželjna svojstva. To povlači da svaki roditelj ima dvoje djece od kojih svako dijete predstavlja jednu polovicu roditelja. Podjela i spajanje se zbog toga također provode na dva dijela odnosno iz dva dijela. Glavna prednost ovako strukturiranih objekata je ta da se traženje presjeka zrake i objekta svodi na pretraživanje binarnog stabla. Također je bitno napomenuti da ako je potrebno promijeniti dio objekta, potrebno je samo zamijeniti određeno podstablo i nije potrebno izgraditi potpuno novo stablo. Nakon što se deformacije akumuliraju u tolikoj mjeri da ruše neka od zahtijevanih svojstava BVH strukture, tek tada se za objekt kreira nova struktura. Česta optimizacija je da se paralelno s izmjenama na originalnoj BVH strukturi, kreira i nova kojom se jednostavno zamjenjuje stara kada akumulirana pogreška postane prevelika. Slika 6.1.1 prikazuje objekt zeca koji je smješten u BVH strukturu. Jednakom bojom su naznačeni određeni čvor, na određenoj dubini u grafu stabla i na objektu zeca.



Slika 6.1.1: *Model zeca smješten u BVH strukturu*

Kao što je diskutirano u petom poglavlju ovog završnog rada, iscrtavanje scene koristeći isključivo postupak praćenja zrake, a pogotovo korištenjem neke od njegovih kompleksnijih varijanti, je veoma skupo i teško izvedivo u stvarnom vremenu. Jedan od pristupa smanjenju složenosti iscrtavanja je hibridno iscrtavanje koje je predstavljeno u petom poglavlju. RTX platforma dolazi s još dva pristupa smanjenju složenosti iscrtavanja – DLSS (*Deep Learning Super Sampling*) i filtar za otklanjanje šuma (engl. *Denoising*). Oba ova pristupa koriste duboke neuronske mreže te za operacije vezane uz njih koriste sklopovske akceleracijske strukture unutar Turing i Volta arhitektura – Tensor jezgre.

6.2. DLSS (*Deep Learinig Super Sampling*)

Jedan od važnih koraka prilikom iscrtavanja virtualnih scena odnosno kreiranja jedne sličice (engl. *frame*) je primjena učinaka naknadne obrade [4]. Učinci naknadne obrade su učinci koji se na sliku primjenjuju nakon što je ona već iscrtana. Ti se postupci oslanjaju na činjenicu da imaju potpunu informaciju od određenoj sličici. Već navedeni SRR i SSAO pripadaju u postupke kojima se nastoji poboljšati kvaliteta generirane slike, a koji se oslanjaju na potpunu informaciju o iscrtanoj slici. Zbog toga se ove postupke može promatrati kao zadnji stadij generiranja slike odnosno

iscrtavanja. U ove postupke pripadaju i razne metode kojima se pokušava riješiti problem aliasa. Alias je neželjena pojava nazubljenih rubova koja se pojavljuje zbog premale frekvencije uzorkovanja. Alias se također može javiti i zbog prevelike brzine transformacija određenog objekta. Problem nastaje kada je ta brzina veća od brzine iscrtavanja. Takav alias se manifestira kao pojava treperenja (engl. *flicker*). Jedna od metoda kojom se pokušava riješiti aliasa i poboljšati kvaliteta slike je TAA (*Temporal Anti Aliasing*). TAA je metoda koja se oslanja na činjenicu da je razlika između dvije susjedne sličice veoma mala. TAA koristi vektore pokreta (engl. *motion vectors*) kako bi analizirala susjedne sličice i pomoću tih informacija poboljšala kvalitetu generirane slike. TAA je metoda koja se implementira unutar sjenčara te kao takva je veoma zahtjevna te kada se koristi za generiranje slika, brzina generiranja osjetno opada. DLSS je novi pristup rješavanju problema poboljšanja slike i rješavanja aliasa. DLSS je zapravo duboka neuronska mreža (*DNN*) koja je trenirana tako da joj se skup za učenje sastojao od 64x SS (*Super Sampled*) slika. Za generiranje takvih slika se esencijalno svaki slikovni element iscrta 64 puta te se za krajnji rezultat uzima prosjek svih tih uzoraka. DLSS je DNN koji je treniran metodom propagacije pogreške unazad. To znači da se od mreže tražilo da generira sliku te se pomoću razlike između dobivenog rezultata i 64x SS slike mijenjaju težine na vezama neurona unutar mreže. DLSS se u sklopu RTX platforme može koristiti na dva načina. Jedan od njih je da se sličice generiraju na manjim rezolucijama te se DLSS-u prepusti da na temelju određenog broja takvih sličica, odnosno uzoraka, generira konačnu sliku željene rezolucije. Takvim pristupom DLSS daje slične rezultate kao i postupak TAA dok se generalno provodi 2 puta brže prema istraživanjima tvrtke Nvidia. Na slici 6.2.1 jasno je vidljivo da nema bitne razlike u rezultatima koje daju TAA i DLSS.

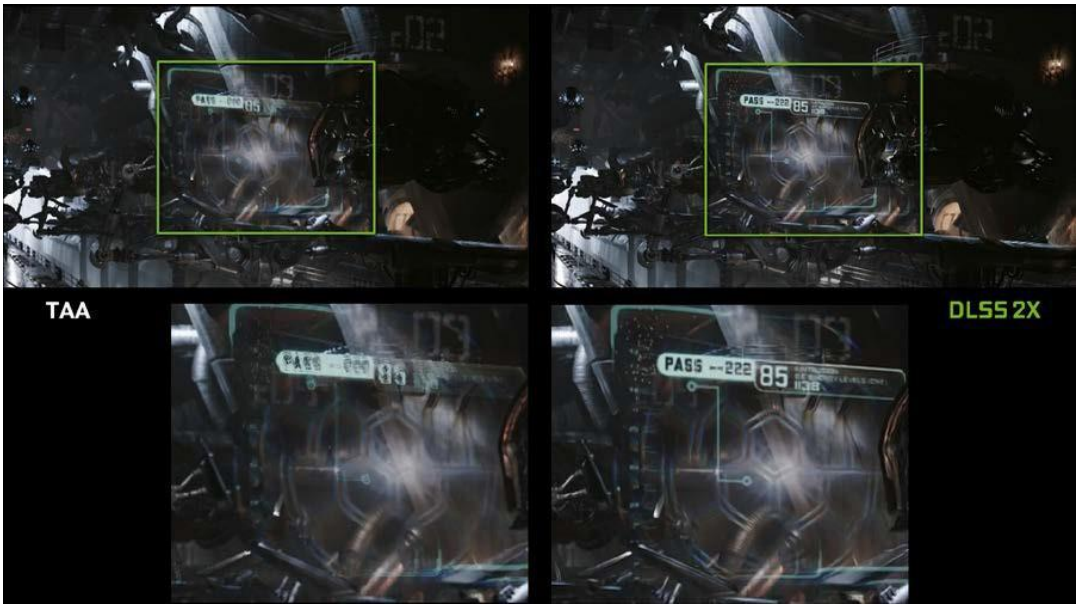
Slika 6.2.1: *Rezultati dobiveni TAAom i DLSSom*

Drugi način uporabe DLSS DNN mreže je da se iscrtavanje obavi na željenoj, a ne na smanjenoj rezoluciji i onda postupi ekvivalentno prvom postupku. Takav način uporabe daje rezultate koji nadilaze one dobivene postupkom TAA. Ti rezultati su na prvi pogled veoma bliski slikama dobivenim 64x SSgom. Slika 6.2.2 prikazuje rezultate dobivene 64x SSom i drugim načinom uporabe DLSSa za kojeg se koristi naziv 2x DLSS.

Slika 6.2.2: *Rezultati dobiveni 64X SSom i DLSSom 2X*

Još jedna od zanimljivih prednosti DLSSa nad TAAom je prikazana na slici 6.2.3. Na slici je prikazana scena u koju je smješten polu-proziran objekt ispred pozadine koja se mijenja na drugačiji način od same polu-prozirne površine. TAA slijepo prati vektore pokreta te zbog toga u tom dijelu slike

daje zamućen rezultat. DLSS je sposoban prepoznati da su promjene u sceni kompleksnije te zbog toga daje bolje rezultate u navedenom dijelu scene.



Slika 6.2.3: *TAA zamućuje dio slike na kojem se nalazi polu-prozirna površina*

6.3. Smanjenje šuma (engl. *Denoising*)

Osnovna varijanta postupka praćenja zrake prirodno ne podržava mnoge efekte koji su poželjni prilikom iscrtavanja virtualnih scena. Zbog toga se unutar RTX platforme definiraju razne varijante postupka koje ili veoma vjerno simuliraju efekte (već prikazani način kreiranja ambijentalnog zasjenjenja kao grube aproksimacije globalnog modela osvjetljenja) ili koje prirodno podržavaju takve efekte (postupak praćenja puta). Postupak praćenja puta je varijanta postupka praćenja zrake koja se danas nalazi u fokusu prilikom primjene postupka praćenja zrake u sklopu RTX platforme. No postupak praćenja puta nije jednostavan. Štoviše, postupak praćenja puta je jedna od računalno najintenzivnijih varijanti postupka praćenja zrake zbog svoje stohastičke prirode te velikog broja uzoraka, odnosno zraka, koji su potrebni kako bi postupak dao zadovoljavajuće rezultate. Kada se postupak provede na malom broju uzoraka, odnosno s malim brojem zraka, dobivaju se rezultati u kojima je, u manjoj ili većoj mjeri,

prisutan šum. Na prvi pogled ove opservacije navode na odbacivanje postupka praćenja puta zbog nemogućnosti njegove primjene u stvarnom vremenu na trenutnim grafičkim procesorima. No, ako se bolje sagledaju prošla poglavlja ovog završnog rada, lako je uočiti da se prilikom generiranja raznih efekata postupkom praćenja zrake unutar RTX platforme koriste koncepti postupka praćenja puta. U stvarnom vremenu nije moguće postupku praćenja puta dopustiti da prikupi dovoljno veliki broj uzoraka kako bi generirao zadovoljavajuće rezultate. Na prvi pogled se čini da je postupak zapeo u zatvorenoj petlji između nezadovoljavajućih rezultata i dugog vremena izvođenja. No u sklopu RTX platforme Nvidia nudi jedno moguće rješenje. Naime, unutar platforme RTX, Nvidia je kreirala filtar zasnovan na dubokim neuronskim mrežama (DNN) koji prima šumovitu sliku te ju nadopunjuje i kao rezultat daje sliku u kojoj je šum ili drastično smanjen ili u potpunosti eliminiran. Ovaj koncept je veoma sličan konceptu duboke neuronske mreže DLSS. Nvidia veliki naglasak daje umjetnoj inteligenciji kao pomoćnom konceptu prilikom iscrtavanja. [3] Konkretno rješenje navedenog problema je da se postupak praćenja puta provode s manjim brojem zraka, odnosno uzoraka, koje je moguće generirati dovoljno brzo da se postupak može izvoditi u stvarnom vremenu. Time se dobiva šumovita slika kao rezultat. Takvu šumovitu sliku se predaje filtru za smanjenje šuma, koji je zapravo duboka neuronska mreža, koji iskustveno popunjava sliku i time smanjuje šum koji postoji na slici. Na slici 6.3.1 prikazana je slika prije i poslije korištenja filtra za smanjenje šuma. Pitanje koje se prirodno postavlja je kolika je razlika u kvaliteti rezultata ovako provedenog postupka i rezultata dobivenih korištenjem postupka praćenja puta s velikim brojem uzoraka. Pokazuje se da su rezultati na prvi pogled gotovo identični, odnosno ne postoji značajna razlika između rezultata, dok su performanse drastično bolje kada se koristi postupak praćenja puta s malim brojem uzoraka i filtrom za smanjenje šuma [3].

Slika 6.3.1: *Slika prije i nakon primjene filtra za smanjenje šuma*

7. PRIMJERI RAČUNALNIH IGARA

7.1. Quake 2

RTX platformu je moguće koristiti u širokom spektru primjena koje mogu a i ne moraju postavljati uvjet izvršavanja u stvarnom vremenu. Jedno od područja u kojima je primjena RTX platforme posebno interesantna su računalne igre. Razvojem računalnih igara generalno se stremi prikazu većih okruženja koja sve vjernije prikazuju stvarnost. Prikaz takvih okruženja ne samo da povećava osjećaj uronjenosti nego i otvara različite mogućnosti inkorporiranja raznih novih elemenata igrivosti. Primjena koncepata praćenja zrake u računalnim igrama je izrazito recentna. Ideja o primjeni je oduvijek bila primamljiva, no sklopovlje jednostavno nije bilo u mogućnosti izvoditi postupak u stvarnome vremenu. RTX platforma i Turing arhitektura grafičkih procesora predstavljaju prve korake u primjeni ovih koncepata u računalnim igrama. Neke od računalnih igara koje koriste RTX platformu i koncepte praćenja zrake spomenute su u poglavlju od efektima koji se nastoje prikazati ovim konceptima. Važno je uočiti da ni jedna od navedenih igara za iscrtavanje vlastitih scena ne koristi isključivo postupak praćenja zrake ili neku od njegovih varijanti. Naprotiv, postupak praćenja zrake se u tim igrama koristi se za generiranje određenih efekata

za koje raterizacijski postupci daju inferiornije rezultate. No jedan zanimljivi primjer igre koja se iscrtava u potpunosti postupkom praćenja zrake je Quake II. Quake II je veoma zanimljiv primjer iz nekoliko razloga. Originalni Quake je jedna od prvih potpuno trodimenzionalnih igara ikada napravljenih, te je i jedna od prvih takvih igara kojoj je dodana podrška za OpenGL prilikom njegovog izlaska zajedno s 3D akceleratorskim karticama. Pokretač koji je pogonio Quake II je evoluirana verzija pokretača originalnog Quakea te je sa sobom donio razne novosti poput osvjetljenja u boji. Pokretač koji pogoni Quake II je već dugi niz godina pod slobodnom licencom. Zbog te činjenice i velike popularnosti same igre, pokretač je doživio prijenos na mnogo različitih platformi tokom godina. Jedna od takvih konverzija je i konverzija koju trenutno provodi tvrtka Nvidia. Naime, Nvidia trenutno cijeli pokretač alterira tako da za iscrtavanje koristi varijantu postupka praćenja puta. Ova verzija pokretača je još uvijek u službenoj izradi. No još jedna inačica ove konverzije postoji. Naime, cijeli pokretač je već alteriran tako da iscrtava igru koristeći isključivo postupak praćenja puta pomoću programskog sučelja Vulkan. Autor ove konverzije je Christoph Schied. Analiza ove konverzije nam daje zanimljive uvide u performanse i kvalitetu rezultata dobivenih postupkom praćenja puta. Analizom performansi iscrtavanja igre na različitim grafičkim procesorima i na različitim rezolucijama, mogu se uočiti sljedeće dvije opservacije [9]:

- performanse igre prilikom izvođenja na grafičkom procesoru s RT jezgrama su približno 7 puta bolje od onih kada se igra izvodi na grafičkom procesoru bez RT jezgara
- performanse prilikom izvođenja igre na grafičkom procesoru s RT jezgrama jako variraju promjenom rezolucije iscrtavanja

Prednost u performansama arhitektura s RT jezgrama ne može se pripisati samo njima. Brzina izvođenja sjenčara također doprinosi dijelom boljim performansama. S druge strane, arhitekture koje u sebi imaju ukomponirane RT jezgre također u sebi sadrže i Tensor jezgre. Ova inačica pokretača za izvođenje postupka praćenja puta koristi već ranije spomenutu optimiziranju inačicu – slika se generira s manjim brojem

zraka, odnosno uzoraka, te se provodi kroz filtar za smanjenje šuma. Slika 7.1.1 prikazuje jednu iscrtanu sličicu na kojoj nisu iscrtani materijali niti je na nju primijenjen filtar za smanjenje šuma. Filtar za smanjenje šuma je duboka neuronska mreža koja koristi Tensor jezgre za ubrzavanje izvođenja vlastitih operacija. Time je moguće zaključiti da i Tensor jezgre doprinose boljim performansama.

Slika 7.1.1: *Iscrtana scena bez primjene filtra za smanjenje šuma*

Razlika u performansama na istom grafičkom procesoru pri različitim rezolucijama iscrtavanja je veoma varijabilna. Sljedeća mjerenja proveli su članovi grupe Digital Foundry:

- brzina iscrtavanja pri rezoluciji od 1920x1080 slikovnih elemenata je stabilnih 60 sličica po sekundi
- brzina iscrtavanja pri rezoluciji od 3840x2160 slikovnih elemenata je stabilnih 20 sličica po sekundi

Lako je primijetiti da performasne pri iscrtavanju pri standardu 4K padaju na trećinu onih pri iscrtavanju pri standardu FHD. Ovo je zanimljiva opservacija iz dva razloga. Standard 4K je standard kojem video igre u današnje vrijeme teže. Grafičkim procesorima standard rezolucije od 4K ne predstavlja nikakav problem pri iscrtavanju veoma kompleksnih scena metodom rasterizacije uključujući izrazito dobre performanse. Iz navedenih mjerenja je vidljivo da se Quake II iscrtava s istim rezolucijskim standardom no s razinom performansi koja graniči s prihvatljivom kada su

u pitanju računalne igre. No, takvi rezultati su u skladu s očekivanjima zbog kompleksnosti primijenjene metode iscrtavanja. S druge strane, činjenica da je Quake II računalna igra koja je stara 20 godina kombinirana s dobivenim mjernim rezultatima veoma zorno prikazuje koliko je zapravo postupak praćenja zrake računalno skup i kompleksan. Quake II je geometrijski višestruko manje složena igra od današnjih igara, ne sadrži kompleksne materijale i efekte, te su scene koje se iscrtavaju volumno mnogo manje od današnjeg standarda. Time se potvrđuje razmišljanje da je uporaba postupka praćenja zrake kao primarne i jedine metode prilikom iscrtavanja virtualnih scena još uvijek veoma ograničena.

7.2. Minecraft

Još jedan od recentnih primjera koji je zanimljiv zbog određenih svojih aspekata je primjena postupka praćenja puta na računalnu igru Minecraft. Riječ je zapravo o posebnoj verziji sjenčara za igru, koju je kreirao internetski korisnik Sonic Ether, u kojoj se implementira postupak praćenja puta. Zanimljivost ove primjene je u tome što veoma vjerno prikazuje koliko je zapravo postupak praćenja zrake odnosno puta računalno zahtjevan. Naime, ova inačica računalne igre postupak praćenja puta izvodi isključivo unutar sjenčara bez uporabe sklopovskih akceleratorskih struktura i RTX platforme. Članovi grupe Digital Foundry proveli su niz pokusa unutar ove verzije igre koji pokazuju moć postupka za generiranje realističnog modela osvjetljenja. S druge strane su također prikazali i rezultate izvođenja koji su sljedeći [10]:

- Na grafičkom procesoru GTX 2080Ti igra se iscrtava u relativno stabilnih 60 sličica po sekundi pri rezoluciji od 1920x1080 slikovnih elemenata
- Na grafičkom procesoru GTX 1070 igra se iscrtava u relativno stabilnih 30 sličica po sekundi pri rezoluciji od 1280x720 slikovnih elemenata

Rezultati prikazuju drastično nisku razinu performansi pogotovo kada se uzme u obzir jačina navedenih grafičkih procesora i grafička jednostavnost navedene računalne igre. Naime, Minecraft je računala igra u kojoj je cijeli

svijet sastavljen od kocaka na koje je primijenjena određena tekstura. Upravo ta činjenica je kreatoru ove modifikacije omogućila optimizaciju postupka kako bi se dosegla ova razina performansi što implicira da bi performanse bilo drastično manje da postupak nije primijenjen na geometrijski jednostavnu računalnu igru poput Minecrafta. No, ova modifikacija je također zanimljiva zbog činjenice da je u njoj lako demonstrirati određene efekte za čiju se generaciju stremi uporabi postupka praćenja zrake. Na slici 7.2.1 je vidljivo okruženje u kojem su prisutne refleksije svijetla unutar okruženja. Svijetlost koja ulazi u prostoriju reflektira se o crveni zid te tako dodaje crvenu boju inače rozim blokovima.

Slika 7.2.1: *Refleksija svijetlosti o crveni zid odražava se na ostatku objekata u sceni*

8. OPTIX

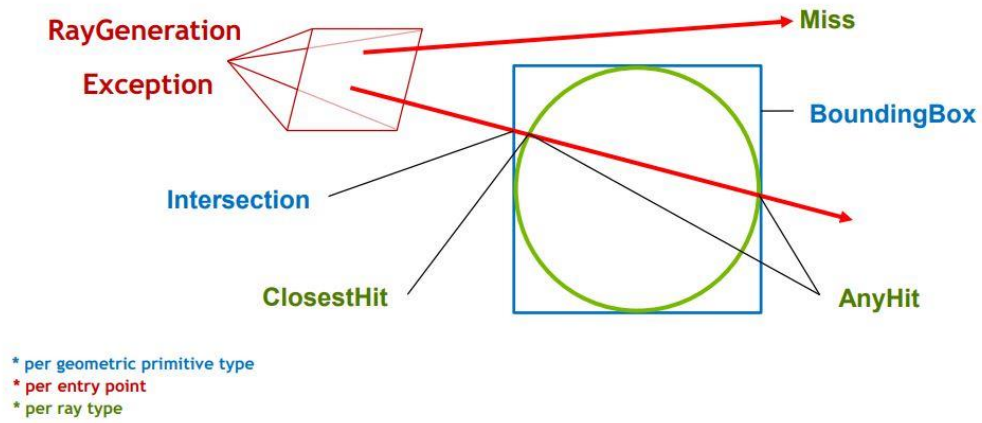
OptiX je SDK (*Software Development Kit*) tvrtke Nvidia za primjenu postupka praćenja zrake. Danas OptiX predstavlja jedno od mogućih sučelja putem kojih je moguće pristupiti RTX platformi. Neki od drugih sučelja su Vulkan i Microsoft DirectX. No, OptiX nije strogo vezan uz RTX platformu. Naime, OptiX je razvijen mnogo prije RTX platforme u vrijeme kada se postupak praćenja zrake i njegove varijante nisu mogle konzistentno izvoditi u stvarnom vremenu. Tako da je njegova uloga kao jedna od pristupnih točaka RTX platformi samo evolucija SDKa kao okruženja za rad s postupkom praćenja zrake. Generalno su za korištenje OptiXa potrebni [11]:

- CUDA Toolkit
- OptiX SDK
- C/C++ prevodilac za programe domaćina (eng. *host application*)
- Grafički procesor arhitekture Kepler ili novije

Aplikacije koje koriste OptiX konstruiraju se veoma slično CUDA aplikacijama. To znači da se aplikacija sastoji od programa domaćina i CUDA programa koji su namijenjeni izvođenju na grafičkom procesoru (eng. *device application*). Programi domaćina se općenito pišu pomoću programskog jezika C/C++. Prevođenje programa domaćina prepušta se nekom od pretpostavljenih prevodilaca za određenu platformu (za platformu Windows to je nativni prevodilac Microsoft Visual Studioa), dok se prevođenje CUDA programa prepušta nvcc prevodiocu. OptiX CUDA programe ne prima u njihovom standardnom obliku, nego ih prima u obliku .ptx datoteka koje nastaju posebnom opcijom prilikom prevođenja .cu datoteka prevodiocem nvcc.

8.1. Općenita struktura OptiX programa

Slika 8.1.1 prikazuje konceptualnu strukturu OptiX programa. OptiX u pozadini skriva relativno kompleksno izvođenje postupka praćenja zrake na grafičkom procesoru, dok prema korisniku pruža relativno jednostavnu strukturu za opisivanje algoritama za provođenje postupka praćenja zrake [11].



Slika 8.1.1: *Konceptualna struktura OptiX aplikacije*

Na slici 8.1.1 različitim bojama su obojani različiti sjenčari. Svim ovim sjenčarima je zajedničko to da se od korisnika očekuje implementacija tih sjenčara. Razlika između različito obojanih sjenčara na slici 8.1.1 je u tome što se za različito obojane sjenčare očekuje implementacija po određenoj komponenti OptiX programa. Implementacija crveno obojanih sjenčara se očekuje po točki ulaska. Crveno obojani sjenčari su:

- Sjenčar kreiranja zrake (engl. *RayGeneration shader*) – unutar ovog sjenčara se očekuje implementacija generiranja zraka pomoću nekog od modela kamere
- Sjenčar iznimke (engl. *Exception shader*) – ovaj sjenčar implementira akcije koje je potrebno napraviti u slučaju iznimke

Plavno obojani sjenčari se odnose na geometrijske primitive. Drugim riječima, za svaku geometrijsku primitivu je potrebno implementirati sljedeće sjenčare:

- Sjenčar obuhvaćajućeg volumena (engl. *BoundingBox shader*) – ovaj sjenčar implementira kreiranje obuhvaćajućeg volumena za određenu geometrijsku primitivu
- Sjenčar presjeka (engl. *Intersection shader*) – ovim sjenčarom se implementira pronalazak presjeka zrake i geometrijske primitive kojoj je sjenčar namijenjen

Zeleno obojani sjenčari se odnose striktno na postupak praćenja zrake. Svaki od ovih sjenčara je potrebno implementirati za svaku vrstu zrake:

- Sjenčar najbližeg presjeka (engl. *Closesthit shader*) – unutar ovog sjenčara se vrši proračun osvjetljenja u točki presjeka zrake i objekta, unutar njega se rekurzivno poziva postupak praćenja zrake za reflektirane i refraktirane zrake i slično
- Sjenčar općenitog presjeka (engl. *Anyhit shader*) – ovim sjenčarom se opisuju akcije koje se obavljaju kada se pronađe presjek zrake i objekta bez obzira na vrstu presjeka
- Sjenčar promašaja (engl. *Miss shader*) – ovaj sjenčar implementira što se događa kada ne postoji presjek zrake i bilo kojeg objekta u sceni

Iz navedenog je vidljivo da OptiX pruža veliku fleksibilnost prilikom definiranja načina provođenja postupka praćenja zrake i njegovih varijanti. Jedini dio postupka za koji OptiX ne dopušta korisničku implementaciju je samo praćenje zrake. Naime, OptiX se brine o tome kako točno provesti praćenje te o tome koju će se zraku u kojem trenutku pratiti i slično.

8.2. Struktura jednostavnog primjera

Za potrebe ovog završnog rada implementiran je jednostavan primjer koji zorno prikazuje učinke zbog kojih se danas teži primjeni postupka praćenja zrake pri iscrtavanju virtualnih scena. To su učinci refleksije i refrakcije svijetlosti te prirodnog generiranja sjena. Ovaj primjer je varijanta jednog od primjera koji dolazi uz OptiX. Implementacija je napravljena kao izmjena tog istog primjera. Scena implementacijskog primjera se sastoji od podloge i četiri kvadra, te pozadinske teksture i jednog izvora svijetlosti. Na tri kvadra je primijenjen jednostavan materijal simuliran Phongovim modelom osvjetljenja. Na zadnji kvadar je primijenjen materijal stakla, dok je na podlogu primijenjen jednostavan proceduralno generiran reflektivan materijal. [12] Svi izvorni kodovi priloženi u sklopu ovog poglavlja su isječci izvornog koda iz implementacije ovog primjera. U nastavku poglavlja biti će objašnjeni glavni gradivni elementi jedne OptiX aplikacije.

8.3. Kreiranje konteksta

Kreiranje konteksta jedan je od glavnih dijelova svakog programa domaćina neke aplikacije koja koristi OptiX [11]. Pojam kreiranja konteksta se odnosi na postavljanje nekih od osnovnih parametara za provođenje algoritama za praćenje zrake. Neki od tih parametara su broj različitih vrsta zraka, veličina stoga, maksimalna dubina, sjenčar koji služi za generiranje zraka, sjenčar promašaja i slično. Unutar konteksta se također definira međuspremnik u kojeg se prosljeđuje rezultat provođenja postupka praćenja zrake. Svi ovi parametri služe kao svojevrsno postavljanje pravila za provođenje postupka koje svi sjenčari moraju poštivati te kao definicija parametara koje ti isti sjenčari koriste pa izvođenje postupka. Sljedeći izvorni kod prikazuje dio funkcije za kreiranje

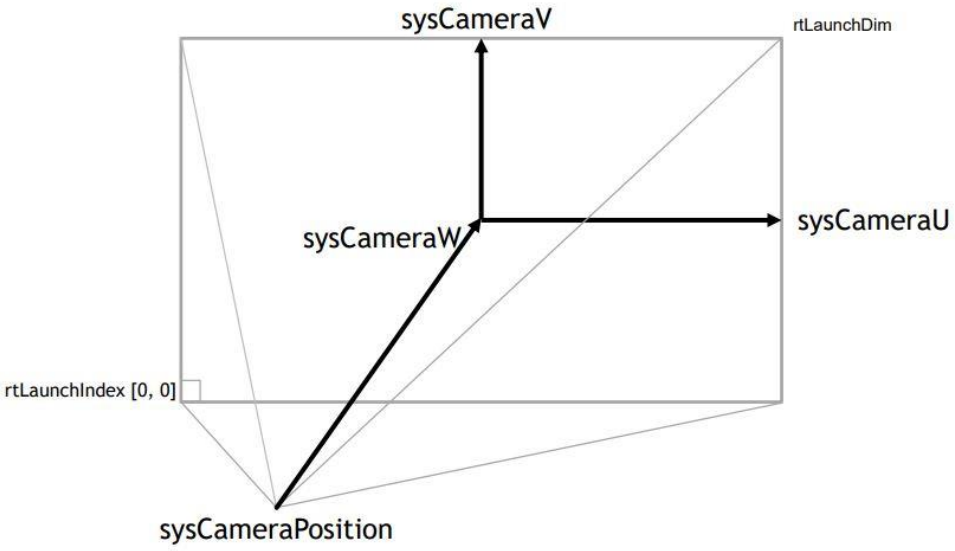
konteksta implementacijskog primjera koji se odnosi postavljanje sjenčara generiranja zraka i sjenčara promašaja:

```
// Ray generation program
    Program ray_gen_program = context->createProgramFromPTXString(
tutorial_ptx, "pinhole_camera" );
    context->setRayGenerationProgram( 0, ray_gen_program );

// Miss program
    context->setMissProgram( 0, context-
>createProgramFromPTXString( tutorial_ptx, "envmap_miss" ) );
    const float3 default_color = make_float3(1.0f, 1.0f, 1.0f);
    const std::string texpath = texture_path + "/" + std::string(
"CedarCity.hdr" );
    context["envmap"]->setTextureSampler( sutil::loadTexture(
context, texpath, default_color) );
    context["bg_color"]->setFloat(   make_float3(   0.34f,   0.55f,
0.85f ) );
```

8.4. Generiranje zrake i model kamere

Kao model kamere se u implementacijskom primjeru koristi jednostavna kamera s malim otvorom (engl. *pinhole camera*). Na slici 8.4.1 prikazan je model takve kamere. Vidljivo je da je kamera modelirana dimenzijama ekrana te vektorima U,V i W, od kojih vektori U i V služe za pomicanje po slikovnim elementima ekrana, dok vektor W služi kao vektor koji određuje smjer zrake kao smjer pravca kroz točku očišta i točku koja predstavlja slikovni element ekrana [11].



Slika 8.4.1: *Model jednostavne kamere*

Sljedeći izvorni kod prikazuje funkciju koja implementira navedeni model kamere te služi kao sjenčar za generiranje zraka:

```
rtDeclareVariable(float3,        eye, , );
rtDeclareVariable(float3,        U, , );
rtDeclareVariable(float3,        V, , );
rtDeclareVariable(float3,        W, , );

…
size_t2 screen = output_buffer.size();

  float2 d = make_float2(launch_index) / make_float2(screen) * 2.f
- 1.f;
  float3 ray_origin = eye;
  float3 ray_direction = normalize(d.x*U + d.y*V + W);
```

8.5. Parametri zrake

Svaka zraka koju se šalje u virtualnu scenu ima drugačije početne parametre te na svojem putu kroz scenu prikuplja različite informacije. Prilikom kreiranja zrake, konstruktoru zrake je potrebno predati informacije o početnoj točki zrake, njezinom smjeru, njezinoj vrsti i o graničnoj vrijednosti za tu zraku [11]. Jedan poziv ovog konstruktora vidljiv je u sljedećem izvornom kodu:

```
optix::Ray   ray(ray_origin,   ray_direction,   RADIANCE_RAY_TYPE,
scene_epsilon);
```

Jednom kada je zraka kreirana, potrebno je pustiti navedenu zraku da putuje scenom. Funkciji koja unutar OptiXa prati put zrake potrebno je predati informaciju o korijenskom čvoru hijerarhijske strukture kroz koju će se zraka pratiti, koju točno zraku treba pratiti te strukturu podataka u koju će se spremati informacije koje zraka prikuplja tokom svog puta. Poziv ove funkcije vidljiv je u izvornom kodu koji slijedi:

```
rtTrace(top_object, ray, prd);
```

Strukturu koja će pamtiti prikupljene informacije definira korisnik što implicira da korisnik definira koje informacije zraka prikuplja tokom svog puta. Uobičajeno je definirati ovakvu strukturu za svaku vrstu zrake. U implementacijskom primjeru koji prati ovaj završni rad, definirane su dvije

vrste zraka te su stoga definirane i dvije vrste navedenih struktura. Svaka vrsta zrake prikuplja različite vrste informacija, pa su stoga ove dvije strukture podataka različite. Strukture su vidljive u sljedećem izvornom kodu:

```
struct PerRayData_radiance
{
  float3 result;
  float  importance;
  int depth;
};

struct PerRayData_shadow
{
  float3 attenuation;
};
```

8.6. Komunikacija između sjenčara

Komunikacija između različitih sjenčara se vrši pomoću deklariranih, dijeljenih varijabli [11]. Primjer ovakvih dijeljenih varijabli je struktura podataka koja pamti informacije koje prikuplja određena zraka tokom svog puta kroz scenu. Primjer deklaracije ovakvih dijeljenih varijabli za navedenu strukturu podataka je prikazan u sljedećem izvornom kodu:

```
rtDeclareVariable(PerRayData_radiance, prd_radiance, rtPayload, );
rtDeclareVariable(PerRayData_shadow,   prd_shadow,   rtPayload, );
```

Još jedan od često korištenih primjera dijeljenih varijabli su informacije o zraci koju se trenutno prati. Naime, kada se pozove određeni sjenčar, na primjer sjenčar najbližeg presjeka, sjenčaru su potrebne informacije o zraci koja je izazvala njegovo pozivanje. Primjer deklariranja dijeljenih varijabli koje sadrže informacije o trenutnoj zraci prikazan je sljedećim izvornim kodom:

```
rtDeclareVariable(optix::Ray, ray,          rtCurrentRay, );
rtDeclareVariable(float,      t_hit,
rtIntersectionDistance, );
rtDeclareVariable(uint2,      launch_index, rtLaunchIndex, );
```

8.7. Geometrija

[11] Svaka virtualna scena je sastavljena od raznih objekata. Kao što je već komentirano, scena implementacijskog primjera se sastoji od 4 kvadra

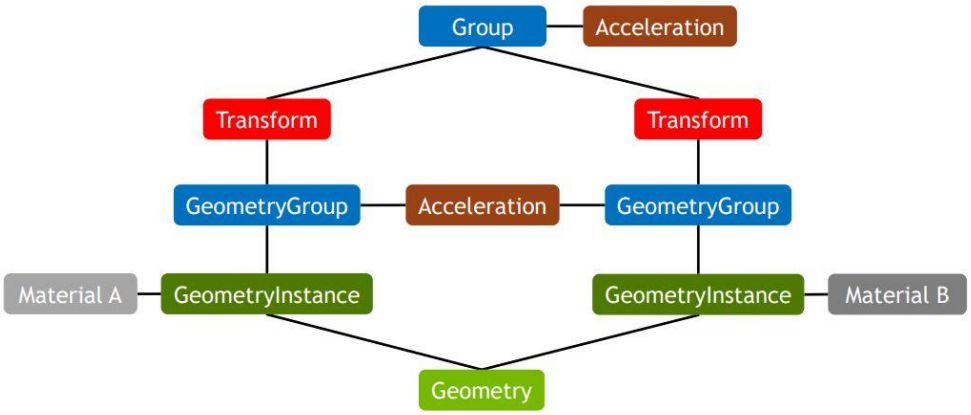
i jedne podloge koja ima oblik paralelograma. Sva četiri kvadra, kao i podloga, tretiraju se kao geometrijske primitive. OptiX za svaku geometrijsku očekuje implementaciju dva sjenčara. Prvi od tih sjenčara definira način pronalaska presjeka (engl. *intersection*) zrake i navedene primitive, dok drugi sjenčar služi za definiciju obuhvaćajućeg volumena navedene primitive. Oba sjenčara se opisuju unutar .cu datoteka odnosno namijenjene su izvođenju na grafičkom procesoru. U sklopu implementacijskog primjera navedeni sjenčari implementirani su za geometrijsku primitivu kvadra i geometrijsku primitivu paralelograma. Važno je napomenuti da se za sva četiri kvadra koriste isti sjenčari za pronalazak presjeka i kreiranja obuhvaćajućeg volumena. Razlog tome je što se ovi objekti geometrijski isti, odnosno njihove razlike se očituju u materijalima koji su primijenjeni na njih, a ne u samoj geometrijskoj strukturi. Također je bitno napomenuti da se navedeni sjenčari mogu implementirati u istoj .cu datoteci za neku geometrijsku primitivu kao u implementacijskog primjeru koji prati ovaj završni rad.

8.8. Organizacija geometrije

Instance geometrije se unutar OptiXa strukturiraju u scenu u obliku hijerarhijske strukture [11]. Slika 8.8.1 prikazuje općenitu hijerarhijsku strukturu scene u kojoj su vidljive različite vrste mogućih čvorova:

- Geometry – ovaj čvor predstavlja geometrijsku primitivu
- GeometryInstance – ovaj čvor predstavlja jednu instancu geometrijske primitive
- Material – čvor koji predstavlja materijal koji je moguće pridodati određenoj instanci geometrijske primitive
- GeometryGroup – čvor koji predstavlja nakupinu instanci geometrije
- Acceleration – čvor koji predstavlja vrstu akceleracijske strukture koja se primjenjuje na određenu grupu ili instancu geometrije
- Group – čvor koji predstavlja najopćenitiju nakupinu geometrijskih primitiva
- Transform – čvor koji predstavlja geometrijsku transformaciju koja se primjenjuje na neku nakupinu geometrijskih primitiva

Način generiranja ovakve hijerarhijske strukture, kao i izgled strukture za scenu implementacijskog primjera prikazan je izvornim kodom u nastavku. Iz izvornog koda je vidljivo da se scena sastoji od jedne nakupine geometrijskih instanci kojih u sceni ima 5. Nad skupinom se primjenjuje određena vrsta akceleracijske strukture te se u kontekst pohranjuje ova nakupina kao vršni čvor od kojeg se kreće u pretraživanje scene za obje vrste zraka u primjeru.



Slika 8.8.1: *Općenita struktura hijerarhijske strukture scene*

```
// Kreiranje instanci za komade geometrije
    std::vector<GeometryInstance> gis;
    gis.push_back( context->createGeometryInstance( box,
&glass_matl, &glass_matl+1 ) );
    gis.push_back( context->createGeometryInstance( parallelogram,
&floor_matl, &floor_matl+1 ) );
      gis.push_back(context->createGeometryInstance(box2,
&difuzni_materijal, &difuzni_materijal + 1));
      gis.push_back(context->createGeometryInstance(box3,
&difuzni_materijal2, &difuzni_materijal2 + 1));
      gis.push_back(context->createGeometryInstance(box4,
&difuzni_materijal3, &difuzni_materijal3 + 1));

    // Kreiranje grupe
    GeometryGroup geometrygroup = context->createGeometryGroup();
    geometrygroup->setChildCount( static_cast<unsigned
int>(gis.size()) );
    geometrygroup->setChild( 0, gis[0] );
    geometrygroup->setChild( 1, gis[1] );
      geometrygroup->setChild(2, gis[2]);
      geometrygroup->setChild(3, gis[3]);
      geometrygroup->setChild(4, gis[4]);
    geometrygroup->setAcceleration( context-
>createAcceleration("Trbvh") );

    context["top_object"]->set( geometrygroup );
    context["top_shadower"]->set( geometrygroup );
```

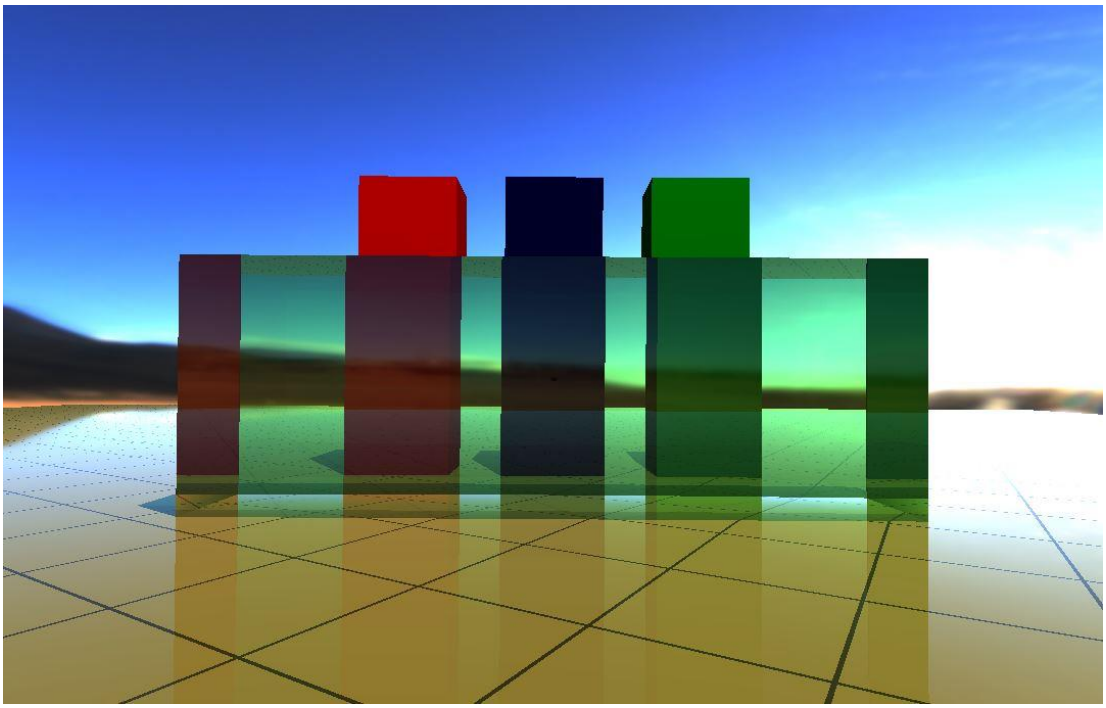
8.9. Materijali

Materijali koji su primijenjeni na geometrijske primitive se također definiraju unutar .cu datoteka. Materijali se implementiraju pomoću sjenčara [12]. Svratimo pozornost na trenutak na početak ovog poglavlja. Tamo je napomenuto da OptiX očekuje implementaciju određenih sjenčara. Mjesto implementacije sjenčara označenih crvenom i plavom bojom su već riješene, dok implementacije sjenčara označenih zelenom bojom nisu. Upravo se implementacijom tih sjenčara, odnosno kombinacijom različitih implementacija za svaku vrstu materijala i vrstu zrake generiraju materijali unutar OptiXa. Ovakav način generiranja materijala je prirodan zbog toga što se pomoću navedenih sjenčara definira ponašanje zrake svijetlosti kada dođe u dodir s objektom. To ponašanje je zapravo i glavno svojstvo nekog materijala iz stvarnog svijeta. Implementacije sjenčara koji definiraju određeni materijal su preuzete iz OptiX primjera.. Generalno govoreći, osnovno ponašanje materijala se definira pomoću sjenčara najbližeg presjeka. Unutar tog sjenčara se definira izračun boje na temelju nekog od modela osvjetljenja, rade se rekurzivni pozivi vezani uz refraktirane i reflektirane zrake, vrši se provjera je li točka presjeka u sjeni i slično. Sjenčari općenitog presjeka u implementacijskom primjeru definiraju ponašanje materijala kada s njim u dodir dođe zraka sjene. U sceni primjera mogu se uočiti tri različite vrste materijala:

- Jednostavni Phongov materijal – materijal koji nema prirodna refleksijska niti refrakcijska svojstva te se računanje boje provodi Phongovim modelom lokalnog osvjetljenja i potpuno zasjenjuje prostor iza sebe u odnosu na izvor svjetlosti
- Reflektivni materijal podloge – ovaj materijal je veoma sličan Phongovom materijalu, no za razliku od Phongovog materijala u obzir uzima refleksije odnosno unutar sjenčara najbližeg presjeka generira rekurzivne reflektirane zrake
- Materijal stakla – ovaj materijal u potpunosti uzima u obzir refleksiju i refrakciju svjetlosti, te implementira sjene koje proizvodi prozirno tijelo

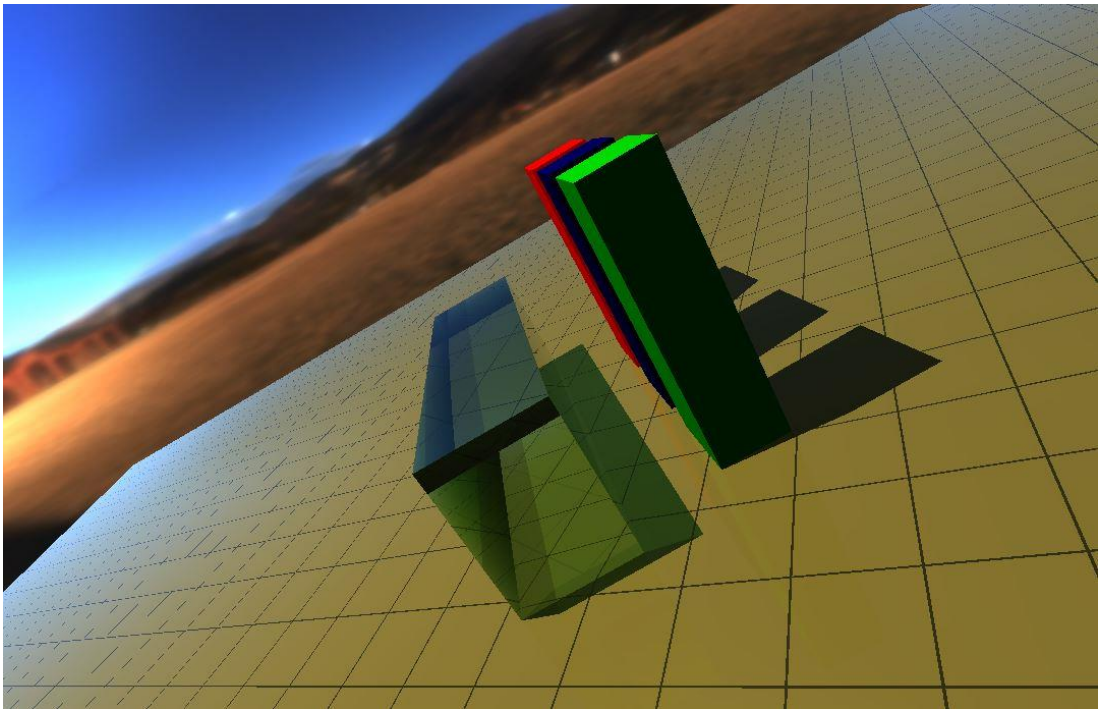
8.10. Dobiveni rezultati

Slika 8.10.1 prikazuje iscrtanu scenu primjera koji prati ovaj završni rad. Unutar iscrtane scene je moguće uočiti svu fenomenologiju koju se željelo prikazati ovim primjerom.



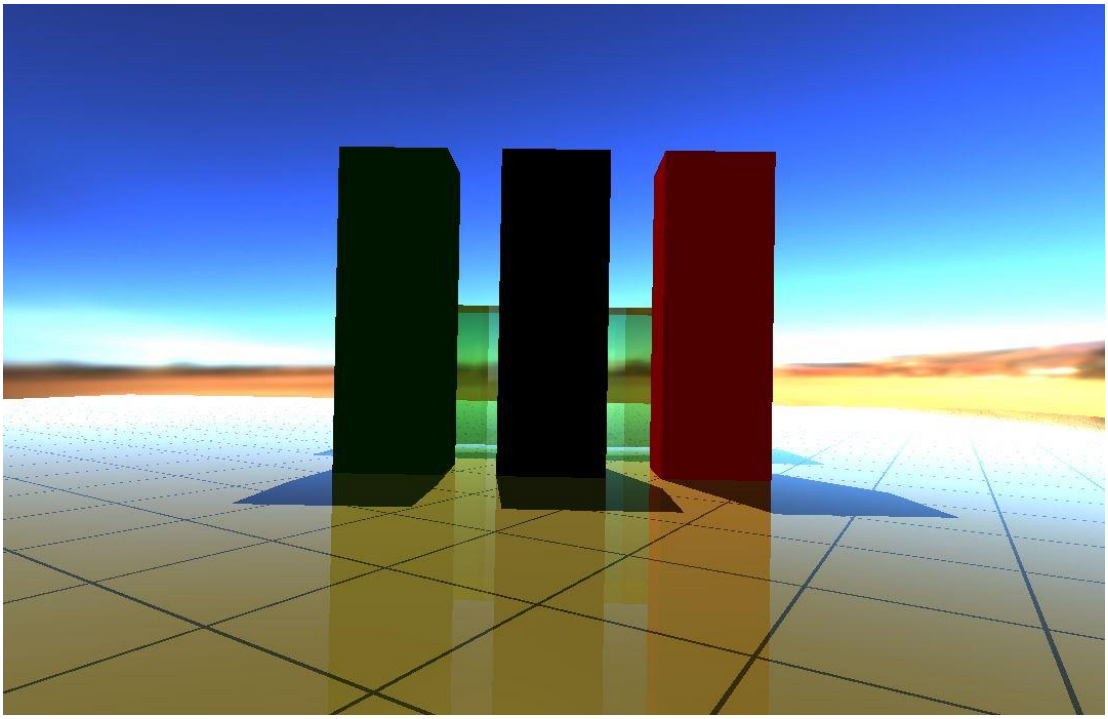
Slika 8.10.1: *Iscrtana scena*

Na slici 8.10.2. vidljiva je razlika u sjenama koje bacaju neprozirni i onoj koju baca prozirni objekt.



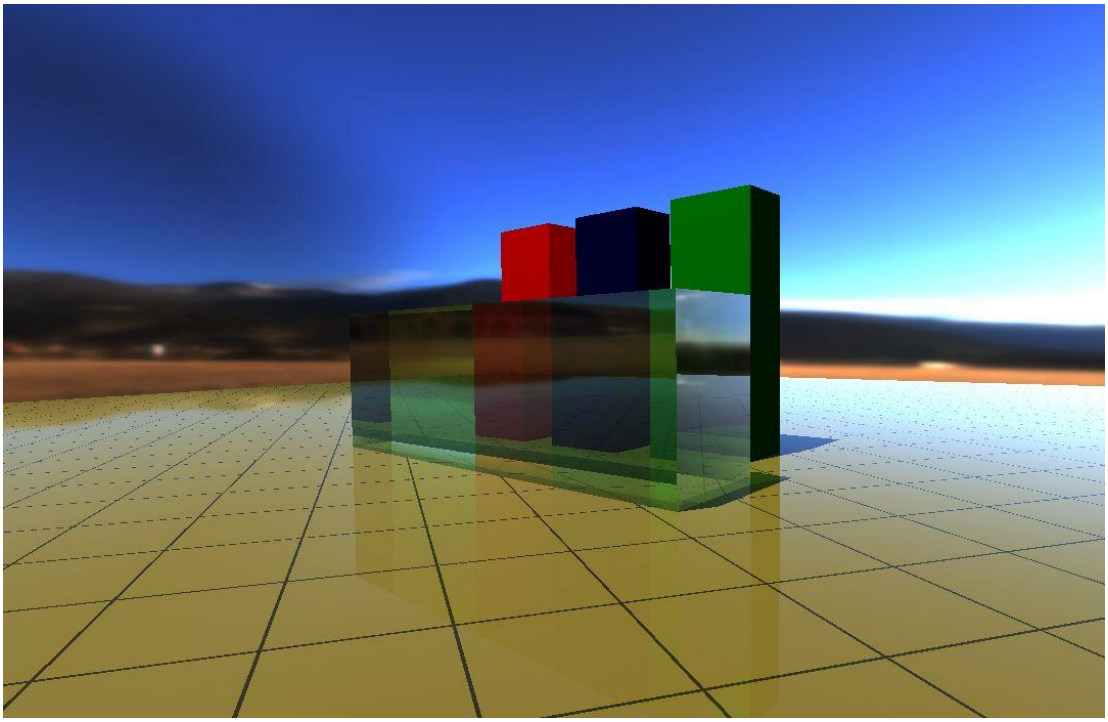
Slika 8.10.2: *Razlika u sjenama*

Slika 8.10.3. prikazuje fenomen refleksije. Na podlozi čiji je materijal reflektivan, vidljive su refleksije objekata koje se nalaze na podlozi.



Slika 8.10.3: *Fenomen refleksije*

Slika 8.10.4. prikazuje fenomen refrakcije. Zbog refrakcije svjetlosti unutar staklenog objekta, dobiva se dojam da su objekti iza njega prekinuti.



Slika 8.10.4: *Fenomen refrakcije*

9. ZAKLJUČAK

Postupak praćenja zrake i njegove varijante su postupci iscrtavanja virtualnih scena koji su već dugi niz godina standard pri iscrtavanju kada ono se ono ne mora izvoditi u stvarnom vremenu. Razvojem grafičkog sklopovlja postalo je moguće izvoditi postupak u stvarnom vremenu. Turing arhitektura grafičkog procesora i RTX programska platforma predstavljaju jedno od rješenja za izvođenja postupka u stvarnom vremenu. No postupak, a pogotovo njegove kompleksnije inačice kojima se teži te koje pretpostavlja RTX platforma, su i dalje veoma intenzivni u pogledu računalnih resursa koji su potrebni za njegovo izvođenje. Zbog toga se unutar RTX platforme nude različita rješenja i pristupi izvođenju postupka kako bi se dobili rezultati karakteristični za postupak, a u isto vrijeme dovoljno smanjila kompleksnost izvođenja kako bi bilo moguće primijeniti postupak u primjenama koje virtualne scene iscrtavaju u stvarnom vremenu. Jedno od tih rješenja je hibridno iscrtavanje koje postupak praćenja zrake koristi isključivo za generiranje učinaka u kojima rasterizacijske aproksimativne metode ne generiraju zadovoljavajuće rezultate. Drugi pristup je primjena dubokih neuronskih mreža na način da se postupkom praćenja puta generiraju slike u kojima postoji određena doza šuma te da se takve slike provedu kroz filtar koji je zapravo duboka neuronska mreža. Izvođenje postupka u stvarnom vremenu ne bi bilo moguće bez sklopovskih akceleracijskih struktura. U sklopu Turing arhitekture tvrtka Nvidia predstavila je dvije glavne takve strukture – RT jezgre za akceleraciju pronalaska presjeka zrake s objektima u sceni, te Tensor jezgre kojima se akceleriraju operacije koje koriste algoritmi dubokih neuronskih mreža. Jedna od primjena postupka u stvarnom vremenu su računalne igre. Današnje računalne igre su previše kompleksne da bi se za njihovo iscrtavanje koristio isključivo postupak praćenja zrake ili neka njegova varijanta, a da se pritom dobivaju zadovoljavajuće razine performansi. Primjena postupka praćenja zrake za iscrtavanje u stvarnom vremenu predstavlja revoluciju u području iscrtavanja virtualnih scena te je kao recentna metoda još uvijek podložan velikim promjenama.

Luka Radivoj

10. LITERATURA

[1] Igor S. Pandžić: Osnove virtualnih okruženja – Laboratorijske vježbe – Vježba 3 – Principi iscrtavanja: Praćenje zrake, s Interneta, https://www.fer.unizg.hr/_download/repository/OVO-V3-upute[2].pdf

[2] Wikipedia: Path tracing, s Interneta, https://en.wikipedia.org/wiki/Path_tracing, 03.06.2019.

[3] Jarred Walton, Alan Bradley: What is ray tracing, and how does it differ from game to game?, s Interneta, https://www.pcgamer.com/what-is-ray-tracing/, 03.05.2019.

[4] Emmett Kilgariff, Henry Moreton, Nick Stam and Brandon Bell: NVIDIA Turing Achitecture In-Depth, s Interneta, https://devblogs.nvidia.com/nvidia-turing-architecture-in-depth/, 14.09.2018.

[5] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, François Sillion: A survey of Real-time Soft Shadow Algorithms, s Interneta, http://maverick.inria.fr/Publications/2003/HLHS03a/index.php, 01.07.2005.

[6] Wikipedia: Ambient occlusion, s Interneta, https://en.wikipedia.org/wiki/Ambient_occlusion, 16.04.2019.

[7] Wikipedia: Global illumination, s Interneta, https://en.wikipedia.org/wiki/Global_illumination, 01.02.2019.

[8] Wikipedia: Bounding Volume hierarchy, s Interneta, https://en.wikipedia.org/wiki/Bounding_volume_hierarchy, 18.05.2019.

[9] Digital Foundry: Quake 2 Path Tracing/ Ray Tracing Analysis: Retro Meets Nvidia RTX!, s Interneta, https://www.youtube.com/watch?v=BRCAfdBMe2Y, 27.01.2019.

[10] Digital Foundry: Minecraft Ray Tracing Live Play: A Path Traced Showcase?, s Interneta, https://www.youtube.com/watch?v=5jD0mELZPD8&t=620s, 21.04.2019.

[11] Ankit Patel, Detlef Roettger: An introduction to Nvidia OptiX, s Interneta, http://on-demand.gputechconf.com/gtc/2018/presentation/s8518-an-introduction-to-optix.pdf, 26.03.2018

[12] Nvidia: Nvidia OptiX Quickstart Guide, s Interneta, https://raytracing-docs.nvidia.com/optix_6_0/tutorials_6_0/index.html#preface#1001, 06.03.2019.

11. SAŽETAK

U sklopu ovog završnog rada razrađena je primjena postupka praćenja zrake i njegovih inačica na iscrtavanje virtualnih scena. Posebna je pažnja dana izvođenju postupka u stvarnom vremenu u sklopu RTX programske platforme tvrtke Nvidia. Razrađene su sklopovske akceleracijske strukture unutar Turing arhitekture grafičkog procesora te algoritamske akceleracijske strukture unutar RTX platforme. Dani su primjeri implementacije određenih učinaka postupkom u raznim računalnim igrama. Izrađen je jednostavan primjer pomoću OptiX SDKa.

Ključne riječi: *praćenje zrake; RTX; Turing; računalne igre; praćenje puta; OptiX*

This paper explains how ray tracing and its variants are used to render computer generated images. It is explained, in further detail, how ray tracing is used to render scenes in real time with Nvidias RTX platform. It is also explained how the Turing graphics procesor architecture is used to accelerate ray tracing and also which algoritmic structures it accelerates. Examples of using ray tracing to generate certain effects in video games are also given. A simple example scene was created with OptiX SDK.

Key words: *ray tracing; RTX; Turing; video games; path tracing; OptiX*