Elektrotehnički fakultet Univerziteta u Beogradu
Katedra za računarsku tehniku i informatiku

Razvoj grafičkog sistema praćenja zraka za interaktivne aplikacije

diplomski rad

Mentor

dr Igor Tartalja

Student

Mirko Mesner
01/137

Beograd, 2010.

Apstrakt

Do sada je računarska grafika u interaktivnim sistemima bila pretežno zasnovana na rasterizaciji poligona. Ovo proističe iz ograničenosti dosadašnjih računarskih sistema da se nose sa velikim brojem operacija koje moraju da se izvrše u jedinici vremena da bi se realizovao interaktivni sistem *praćenja zraka* (*eng. ray tracing*). Povećanjem performansi došlo se do trenutka kada je postalo moguće implementirati interaktivni sistem praćenja zraka na široko dostupnim računarskim sistemima.

Rezultat ovog rada je softverski sistem za sintezu digitalne slike primenom tehnike praćenja zraka. Rad poseduje pretežno implementacione, ali i istraživačke rezultate. U izradi ovog sistema ispitane su i primenjene neke od postojećih tehnika praćenja zraka, a prikazane su i njihove mogućnosti. Za razliku od preovlađujuće primene tehnike praćenja zraka za crtanje neinteraktivnog sadržaja, u ovom sistemu je omogućena interakcija korisnika i sadržaja, odnosno pozicija posmatrača i pozicija objekata u sceni dinamički su promenljivi. Pri tom su modifikovani postojeći algoritmi, kako bi se umanjio negativan uticaj interaktivnog rada na performanse sistema praćenja zraka. Te modifikacije se zasnivaju na novim idejama koje su poznate iz javno dostupnih informacija o postojećim srodnim sistemima. Sistem je projektovan tako da se značajan broj aktivnosti koje se odvijaju u svakoj iteraciji postupka sinteze slike može jednostavno zameniti kompatibilnim aktivnostima, što će pojednostaviti dalje eksperimente i razvoj sistema. Fleksibilnost koja se na taj način postiže treba da omogući i raznovrsniju primenu razvijenog sistema. U skladu sa tim, u demonstracionim aplikacijama koje su posebno razvijene demonstrirane su mogućnosti i performanse željenog sistema za različite tipove scena.

1. Uvod

Praćenje zraka (eng. *ray-tracing*) predstavlja tehniku za sintezu digitalne slike koja se ne zasniva na rasterizaciji poligona, već računa boju i osvetljenje svakog piksela slike prateći zrake izvora svetla, ali u obrnutom smeru: ne od izvora svetla ka ravni projekcije slike, već od ravni projekcije (odnosno oka virtuelne kamere) ka izvoru svetla. Zrak se ne prati od izvora svetla ka posmatraču iz praktičnih razloga smanjenja količine potrebnih resursa. Ako bi zraci bili praćeni od izvora svetla, veliki deo tih zraka ne bi imao nikakvog uticaja na osvetljenje bilo kog piksela generisane slike. To bi dovelo do velikog rasipanja resursa, jer svaki zrak koji se "šalje" u scenu troši određene resurse (memorijske i procesorske). Putanja svakog zraka svetla podleže optičkim zakonima na koje utiču gustina sredine, parametri materijala kroz koji zrak prolazi ili se odbija itd., pa složenost i preciznost kojom se oni modeliraju značajno utiču na performanse, te je smanjenje njihovog broja važno.

Tehnika rasterizacije ne omogućava fizički korektan model osvetljenja scene, ali, uprkos tome, ova tehnika uspeva da sintetizuje kvalitetne i relativno realistične slike. Međutim, to ima svoju cenu u vidu velikog manuelnog rada dizajnera aplikacije: na primer, senke u aplikaciji nisu direktna posledica primenjenih algoritama u okviru rasterizacionih tehnika, već se senke moraju dodatno implementirati. S druge strane, postoji tehnika praćenja zraka, koja se u velikoj meri koristi za sintezu slika kada je prihvatljivo da vreme sinteze bude značajno veće od 1/24 sekunde. Ova tehnika omogućava sintezu slike superiornog kvaliteta u odnosu na tradicionalnu rasterizaciju, i to bez velike pomoći dizajnera, zato što razni vizuelni efekti, poput refleksija, senki i drugih, predstavljaju prirodne posledice primenjenih algoritama u tehnici *praćenja zraka*.

Razvijen je sistem praćenja zraka koji omogućava razvoj interaktivnih aplikacija sa animiranom fotorealističnom slikom na PC računarima sa savremenom i već široko rasprostranjenom grafičkom podrškom. Pri tom su primenjeni već dobro poznati algoritami i tehnike praćenja zraka, ali su modifikovani, implementirani i ispitani i napredni algoritmi i tehnike za ubrzanje sistema praćenja zraka. Kada se pominje algoritam, misli se na konkretan korak u procesu praćenja zraka, dok se skup svih algoritama naziva tehnikom praćenja zraka. Modifikacije algoritma, kao što je modifikacija algoritma za obilazak Kd stabla, su urađene radi implementacije efikasnijeg sistema praćenja zraka za interaktivne aplikacije. Kao ilustracija mogućnosti ravijenog sistema, implementirana su tri demonstraciona programa, koji pokazuju mogućnosti sistema.

Sistem *praćenja zraka* je realizovan korišćenjem razvojnog okruženja *Microsoft Visual Studio 2010* [1]. Sistem je platformski nezavisan, ali je preveden i testiran samo pod *Windows* operativnim sistemom. Prilikom implementacije softvera korišćeni su postojeći algoritmi, biblioteke i alati. Za prikazivanje slike na ekranu, dobijene sistemom praćenja zraka, koristi se *OpenGL* [2] API. Prilikom stvaranja slike, pomoću sistema za praćenje zraka, ne koriste se mogućnosti *OpenGL* API. Rad sa programskim nitima je realizovan pomoću multiplatformske *BOOST* [3] biblioteke. Geometrijski modeli koje sistem koristi

realizovani su pomoću *3ds Max 2010* [4] programa i učitavaju se u sistem u vidu OBJ fajla. Animacije modela realizovane su pomoću *MilkShape3D* [5] programa i učitavaju se iz MS3D fajla koje ovaj program stvara.

U nastavku će biti objašnjene komponente razvijenog softverskog sistema koji se opisuje u ovom radu i osnove tehnike praćenja zraka, uz osvrt na poznate načine za modeliranje optičkih fenomena, neophodnih za postizanje realističnosti prikaza. U odeljku *2* biće dat detaljan opis problema i značaj rešavanja tog problema. Kratka istorija tehnike i rešenja problema praćenja zraka biće opisana u poglavlju *3*. Poglavlje *4* se bavi osnovnim tehnikama i algoritmima koji su implementirani u razvijenom sistemu. Tehnike optimizacije su podeljene u dva tipa, na softverske i hardverske. Softverska optimizacija je objašnjena u odeljku *4.2*, a hardverska u odeljku *4.3*. Nove ideje i tehnike za rad sistema sa interaktivnim sadržajem uvode se u poglavlju *5*. U poglavlju *6* dat je programski model sistema praćenja zraka. Opisane su klase i relacije između njih i priloženi su UML klasni dijagrami. Detalji implementacije sistema dati su u poglavlju *7*. U poglavlju *8* dat je zaključak uz osvrt na moguće pravce daljeg razvoja opisanog sistema.

2. Problem

Primena tehnika praćenja zraka danas je dominantan način za generisanje fotorealističnih slika. Međutim, do sada je ova tehnika uglavnom bila ograničena na neinteraktivne aplikacije. Još uvek postoje mišljenja [6] da ova tehnika nikada neće preuzeti primat od rasterizacionih tehnika u interaktivnim aplikacijama uprkos povećavanju brzine računarskih sistema. Osnovni problem koji je u ovom radu rešavan jeste implementacija poznatog algoritma za praćenje zraka uz njegovu optimizaciju za potrebe primene u interaktivnim grafičkim aplikacijama.

Glavni nedostatak tehnike praćenja zraka jeste značajno veća zahtevnost za računskom snagom računarskog sistema u odnosu na zahteve tehnika za rasterizaciju. Taj nedostatak posebno dolazi do izražaja kada scenu sa animiranim objektima treba prikazivati interaktivno vođenom kamerom. Zato problem rešavan u ovom radu predstavlja posebno atraktivan izazov.

Problem tehnike praćenja zraka je veliki zaostatak u performansama u odnosu na performanse sistema razvijenih pomoću rasterizacijskih tehnika. Te performanse još više opadaju ako se u sistem uključi podrška za interaktivi sadržaj. Rešenje problema se sastoji iz implementacije već postojećih algoritama i tehnika, prilagođavanja sistema interaktivnom sadržaju i ubrzanju sistema praćenja zraka. Prilagođavanje sistema praćenja zraka interaktivnom sadržaju je specifičan problem. Kako bi se ubrzao jedan od osnovnih algoritama tehnike praćenja zraka, određivanje preseka primitive i zraka, primitive se smeštaju u strukture koje ubrzavaju algoritam testiranja preseka primitive i zraka. Ako se u sistemu podrži interaktivni sadržaj ubrzavajuće strukture se moraju iznova generisati za svaku sliku. Problem nastaje kada potrebno vreme za generisanje ubrzavajuće strukture postane preveliko, usled broja primitiva u sceni, i dovodi do gubitka performansi.

3. Istorija tehnike praćenja zraka

U ovom poglavlju je prikazan pregled postojećih rešenja iz oblasti tehnike praćenja zraka. Rešenja su podeljena u kategorije softverskih i hardverskih rešenja.

3.1. Softverska rešenja

Kratak istorijski pregled razvoja algoritama i tehnika praćenja zraka je dat u ovom odeljku.

3.1.1. Emitovanje zraka (eng. *Ray-Casting*)

Prvi algoritam koji je postavio osnove *praćenja zraka* jeste *Ray Castng*, koji je 1968. godine razvio *Arthur Appel*[7]. Iza ovog algoritma leži ideja inverzna procesu koji se odvija u prirodi. Za svaki piksel ekrana šalje se zrak sa pozicije kamere u scenu, određuje se najbliži objekat koji se nalazi na putanji zraka i uzima se boja tačke proboja. U ovom sistemu nisu podržane senke koje prave neprozirni objekti. Praćenje zraka prestaje posle pronalaženja prvog preseka. Velika prednost ovog algoritma, u odnosu na algoritme zasnovane na rasterizaciji, jeste to što on podržava prikazivanje neplanarnih površina čvrstih tela.

3.1.2. Whittedovo praćenje zraka

Prirodna nadgradnja na prethodno opisani algoritam jeste to da se nastavi sa praćenjem zraka nakon prvog sudara sa nekim objektom. Do tog zaključka i algoritma došao je *Turner Whitted* [8] 1979. godine. Njegov algoritam nastavlja praćenje tako što posle sudara objekta i zraka mogu nastati tri nova tipa zraka: odbijeni zrak (eng. *reflected ray*), prelomljeni zrak (eng. *refracted ray*) i zrak senke (eng. *shadow ray*). Odbijeni zrak nastavlja putanju kao zrak odbijen od površine savršenog ogledala do seledećeg objekta (koji se vidi u refleksiji na površi prvog objekta). Prelomljeni zrak postoji kod transparentnih objekata. Ovaj zrak nastavlja putanju kroz transparentni objekat, s tim da zrak može više puta ulaziti i izlaziti iz objekta. Konačno, zrak senke služi da se odredi da li se iz date tačke preseka objekta i zraka vidi izvor svetla, čime se određuje doprinos svetla pikselu iz koga je zrak potekao. Dakle, ako se između tačke preseka zraka i objekta, sa jedne strane, i svetla, sa druge, nalazi neki objekat, onda se tačka preseka nalazi u senci koju pravi taj objekat.

3.1.3. Cookovo distribuirano praćenje zraka

Iako je *Whittedovo praćenje zraka* dozvolilo računanje refleksija, refrakcija i senki, bilo je ograničeno na perfektne refleksije i refrakcije i tačkaste izvore svetla. Te restrikcije kasnije je uklonio *Cook* [10], koji je proširio tehniku *praćenja zraka* tako da ona uključuje realističnije efekte, kao što su difuzione refleksije, meke senke, kvadratni izvori svetla, dubina polja i zamućivanje usled kretanja (eng. *motion blur*).

Da bi se ove restrikcije uklonile, Cookov sistem praćenja zraka dozvoljava korišćenje više zraka za određivanje osvetljenja jedne tačke objekta. Na primer, da bi se odredila osvetljenost neke tačke od strane svetla koje ima neku površinu, više zraka se distribuira po površini tog svetla. Sada neki delovi svetla mogu da osvetljavaju određenu tačku, čime se dobija delimično osvetljena tačka objekta i meke senke. Ista tehnika distribucije zraka može se primeniti i na refleksije i refrakcije.

3.1.4. Programibilno senčenje

Prethodni sistemi *praćenja zraka* imali su fiksirani model senčenja. Ubrzo je uočeno da se izgled i kvalitet slike može menjati promenom modela senčenja. Vremenom su razvijeni razni modeli senčenja kao što su *Blinn* [11], *Phong* [12], *Cook-Torrance* [13], *Torrance-Sparrow* [14], *Ward* [15], *Ashikmin* [16], *Lafortune* [17] itd. Ovi modeli senčenja su opisani pomoću jezika senčenja (*eng: shading language*), od kojih je najpoznatiji *RenderMan* [18]. Jezik senčenja *RenderMan* se može iskoristiti za opisivanje proizvoljnog modela senčenja, dok su gore navedeni modeli senčenja najpoznatiji. Takvi jezici su razdvojili proces senčenja od postupka praćenja zraka kroz scenu. Model senčenja se može dinamički menjati tokom rada sistema, jer sada nije fiksiran jedan model senčenja u sistemu. Do pojave jezika senčenja, model senčenja je bio direktno opisan u kodu sistema praćenja zraka, dok je sada opisan zasebnim programskim jezikom.

3.1.5. Moderni sistemi praćenja zraka

Praćenje zraka je postao dominantan način za generisanje slike i raznih specijalnih efekata u oblastima kao što su filmovi, reklame, muzički spotovi, industrijski dizajn, arhitektura itd. Na tržištu se nalazi veliki broj sistema *praćenja zraka* koji se razlikuju po svojim osobinama. Najpoznatiji su takvi sistemi *Blender* [19], *Brazil* [20], *Maxwell* [21], *Mental Ray* [22], *POV-Ray* [23], *V-Ray* [24]. Svaki od ovih sistema se integriše u neki od alata za modelovanje (npr. *3dsMax*) i generiše sliku na osnovu scene koja je modelovana u okviru tog programa. Na današnjim računarskim sistemima svakom od ovih sistema za praćenje zraka potrebno je u praksi od nekoliko desetina minuta do nekoliko sati za generisanje jedne fotorealistične slike. Svi ovi sistemi podržavaju tehniku globalne iluminacije, koja u velikoj meri doprinosi realističnosti generisanih slika.

Globalna iluminacija se zasniva na algoritmu koji prati zrake od izvora svetla ka oku i zove se *Path Tracing* [9]. Ovu tehniku je prvi predložio James Kajiya 1986. godine. Velika mana ove tehnike je ta što dolazi do ogromnog rasipanja resursa, jer će značajan broj zraka otići u prazan prostor i uopšte neće uticati na osvetljenje piksela generisane slike. Ovo prvenstveno važi za scene koje predstavljaju otvoreni prostor. Ipak, za postizanje što realističnijih slika potrebno je kombinovati *Ray Tracing* i *Path Tracing* u vidu tehnika *globalne iluminacije*. Globalna iluminacija predstavlja tehniku koja omogućava da se izračuna indirektna osvetljenost objekta. U realnom svetu objekti nisu osvetljeni samo direktno od strane izvora svetla nego i od svetla koje se odbija od drugih objekata.

3.1.6. Interaktivni sistemi praćenja zraka

Prvi interaktivni sistem *praćenja zraka REMRT/RT tools* razvio je *Mike Muus* 1986. godine za sistem *BRL-CAD*, koji je primenjivan u okviru industriskog modelovanja. Ovaj sistem je postizao brzinu od nekoliko frejmova u sekundi na tadašnjim računarskim sistemima koji su se sastojali od velikog broja umreženih računara. Od tada se ulažu veliki napori za razvoj interaktivnih sistema *praćenja zraka* koji će raditi na običnim kućnim računarima. Taj napor najviše je stimulisala želja da se takav sistem primeni u okviru video igrica.

OpenRT [25] je projekat čiji je cilj da se razvije sistem *praćenja zraka* koji ima API sličan OpenGL-u, kako bi postao alternativa sistemima zasnovanim na rasterizaciji za primenu u interaktivnim 3D aplikacijama.

Intel je 2008. godine izvršio demonstraciju modifikovane igrice *Enemy Territory: Quake Wars* [26], u kojoj je primenjen sistem praćenja zraka koji je razvio taj proizvođač. Na računarskom sistemu, koji se sastojao od 16 procesorskih jedinica koje su radille na frekvenciji od 2.93 GHz-a, igrica je u 720p HD rezoluciji (1280 × 720 piksela) radila brzinom od 14 do 29 frejmova u sekundi.

Nvidia radi na svom API-ju *OptiX* [27], koji će omogućiti razvoj sistema praćenja zraka koji će raditi na *Nvidinim* grafičkim karticama.

3.2. Hardverska rešenja

Do sada se pojavilo nekoliko komercijalnih i istraživačkih implementacija hardvera za praćenje zraka:

- Kompanija *ART VPS* [28] je proizvodila od 2002. do 2009. godine komercijalan hardver za neinteraktivno iscrtavanje tehnikom praćenja zraka. Hardver koji su razvili koristi višestruke procesore koji ubrzavaju testiranje preseka zraka i trouglova.
- Laboratorija na Sarlandskom univerzitetu je pod vođstvom *Philippa Slusalleka* 2002. godine proizvela prototip hardvera za praćenje zraka sa FPGA (*Field-Programmable Gate Array*) programibilnim integrisanim kolom nazvan SaarCOR [29] (*Saarbrücken's Coherence Optimized Ray Tracer*), a 2005. godine proizvela je čip RPU (*Ray Processing Unit*).
- Kompanija *Caustic Graphics* [30] je 2010. godine proizvela karticu sličnu grafičkim karticama koja preuzima na sebe izračunavanja u vezi sa praćenjem zraka. Kao što moderne grafičke kartice hardverski ubrzavaju algoritme rasterizacije, tako ova kartica hardverski ubrzava algoritme primenjene u tehnici praćenja zraka.

4. Specifikacija sistema

Prvi deo ovog poglavlja bavi se osnovnim tehnikama i algoritmima, dok se drugi deo odnosi na tehnike i algoritme optimizacije i ubrzanja praćenja zraka. Na kraju je objašnjen način na koji sistem podržava interaktivne aplikacije, što predstavlja istraživački rezultat ovog rada.

4.1. Osnovne faze

Kada se govori o *praćenju zraka*, pre svega se misli na skup većeg broja tehnika, od kojih svaka služi za rešavanje određenog potproblema u celokupnom postupku. Opšti oblik postupka praćenja zraka u velikoj meri se ne razlikuje između različitih implementacija sistema praćenja zraka. U tom smislu, delovi sistema razvijenog u okviru ovog rada ne razlikuju se značajno od postojećih, dobro utvrđenih tehnika i već implementiranih algoritama. Primenjeni algoritmi se dele na sledeće dve grupe, u zavisnosti od faze u kojoj se primenjuju:

- **Faza pripreme scene za iscrtavanje.** U ovoj fazi se vrši priprema za iscrtavanje geometrije scene. Takođe, u ovoj fazi se uzima u obzir dinamičnost izmene sadržaja scene.
- **Faza za generisanje slike.** U ovoj fazi se generiše slika. Za ovu fazu se može reći da je postala *de facto* standardizovana, jer je slična kod većine sistema *praćenja zraka* [6].

Iako je faza pripreme scene navedena kao prva, što deluje kao logičan sled stvari, ona će biti opisana u kasnijim poglavljima, jer je ona uvedena u kasnijem razvoju tehnike **praćenja zraka** kao vrsta softverske optimizacije.

4.1.1. Faza za generisanje slike

Implementacija osnovnog ciklusa se po sadržaju ne razlikuje od implementacija kod drugih sistema. Za svaki piksel na ekranu konstruiše se zrak koji se šalje u scenu, od posmatrača kroz dati piksel, nakon čega se traži najbliži presek tog zraka i primitiva u sceni, a na kraju se nalazi odgovarajuća boja na preseku. Ako presek postoji, boja se stavlja u *off-screen* bafer. Ovo predstavlja najjednostavniji algoritam bez daljeg slanja novih zraka u scenu (na primer, nakon ogledanja), i praćenje zraka se završava pri prvom preseku zraka i primitive. Zraci koji se ovde prate nazivaju se **primarni zraci**. Za razliku od ovog jednostavnog algoritma, u sistemu je implementirano dalje praćenje zraka, ukoliko dođe do preseka zraka sa objektima čiji materijali imaju osobine odbijanja i prelamanja svetlosti.

4.1.1.1. Svetlo

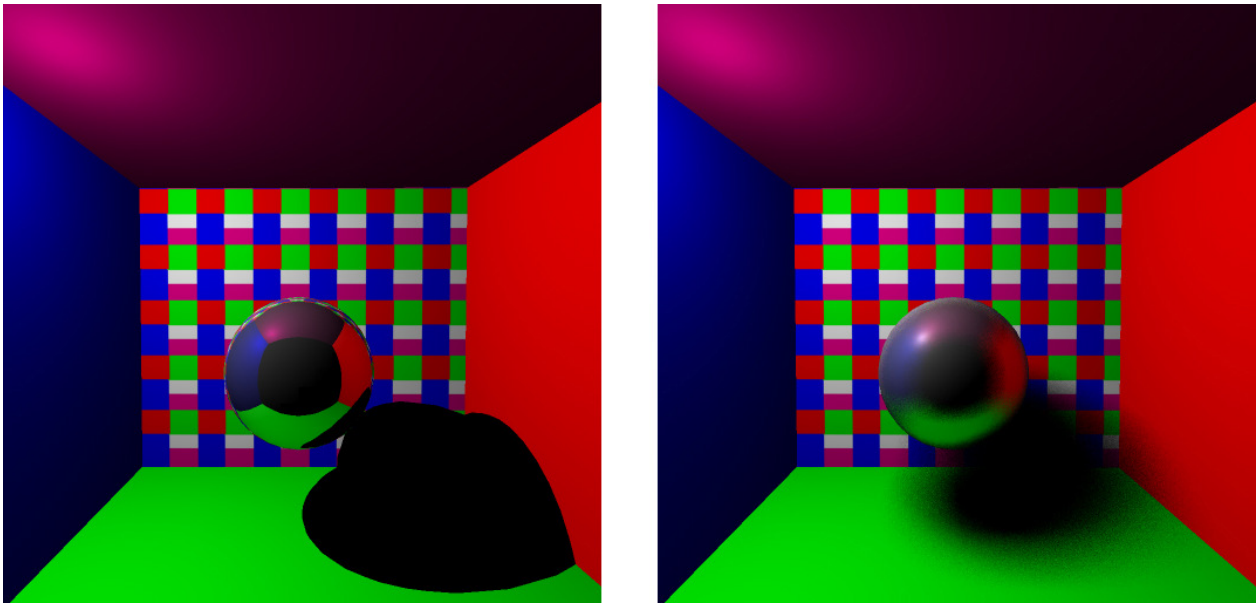
U sistemu su realizovana sledeća dva tipa svetla:

- **Tačkasti izvor svetla**, koji je određen svojom pozicijom u sceni i bojom. Rastojanje izvora svetla od objekata u sceni mora biti konačno (zraci svetla ne smeju biti paralelni). Ovaj tip svetla daje oštre senke.

- **Izvor svetla pravougaonog oblika**, koji je definisan pravougaonikom u 3D prostoru i bojom. Prilikom računanja osvetljenosti neke tačke u sceni od strane pravougaonog izvora svetla, svetlo će biti aproksimirano određenim brojem tačkastih izvora svetla. U cilju dobijanja što boljeg rezultata sa što manjim brojem tačkastih izvora svetla, uzorci koji se uzimaju sa površine pravougaonika izvora svetla neće biti uniformno raspoređeni, već će se upotrebiti generator slučajnih brojeva za njihovo odabiranje. Posledica ovakvog pristupa jeste pojava određenog stepena "šuma" u generisanoj slici. Međutim, u praksi je ustanovljeno da je tako dobijena slika (sa slučajnim šumom) prirodnija od slike dobijene primenom pravilnog rasporeda tačkastih izvora svetla, kada se pojavljuju jasno uočljivi pravilni artefakti. Sa povećanjem broja uzoraka uzetih sa površi pravougaonog izvora svetla, aproksimacija postaje vernija realnom modelu. Broj uzoraka je zadat u vidu parametra. Ovaj metod rešavanja problema pomoću uzimanja uzoraka spada u **Monte Carlo** [31] metode.

4.1.1.2. Senke

Senke (*Slika 1*) u tehnici **praćenja zraka** nastaju kao prirodna posledica računanja osvetljenosti neke tačke. Prilikom računanja osvetljenosti preseka zraka i primitive u sceni, potrebno je utvrditi da li je ta tačka preseka osvetljena od strane izvora svetla, što



Slika 1. Tačkasti izvor svetla i površinski izvor svetla

se može utvrditi tako što se proveri da li se neka druga primitiva nalazi između tačke preseka i izvora svetla. Ako se neka primitiva nalazi između, onda je tačka preseka u senci. Transparentnost primitive, koja se nalazi između, ne uzima se u obzir. Uzimanje u obzir transparentnosti prilikom računanja osvetljenosti zahteva naprednije algoritme.

U realnom svetu objekti su osvetljeni i indirektno. Zrak svetla se može odbiti od nekog objekta i osvetliti drugi. Tehnika koja to omogućava, u okviru sistema **praćenja zraka**, naziva se **indirektna iluminacija** ili **globalna iluminacija**. Indirektna iluminacija nije podržana u implementiranom sistemu.

Postupak određivanja stepena osvetljenja neke tačke preseka zavisi od tipa izvora svetla za koji se realizuje. Kod tačkastog izvora svetla tačka preseka zraka i primitive može biti ili totalno u senci ili totalno osvetljena od strane izvora svetla. Funkcija koja određuje stepen osvetljenja tačke preseka, za slučaj tačkastog izvora svetla, vraća broj 0 ili 1, gde 0 znači da je tačka potpuno zaklonjena nekim objektom, a 1 da je potpuno osvetljena. U slučaju pravougaonog izvora svetla tačka preseka, osim slučaja potpune osvetljenosti ili potpune zaklonjenosti od strane drugog objekta, može biti i delimično osvetljena od strane pravougaonog izvora svetla. Tada funkcija koja određuje stepen osvetljenja tačke preseka vraća realan broj između vrednosti 0 i 1. Rezultat ovoga je da objekti osvetljeni tačkastim izvorom svetla daju oštre senke, dok objekti osvetljeni od strane pravougaonog izvora svetla daju meke senke.

4.1.1.3. Osobine materijala i optički modeli

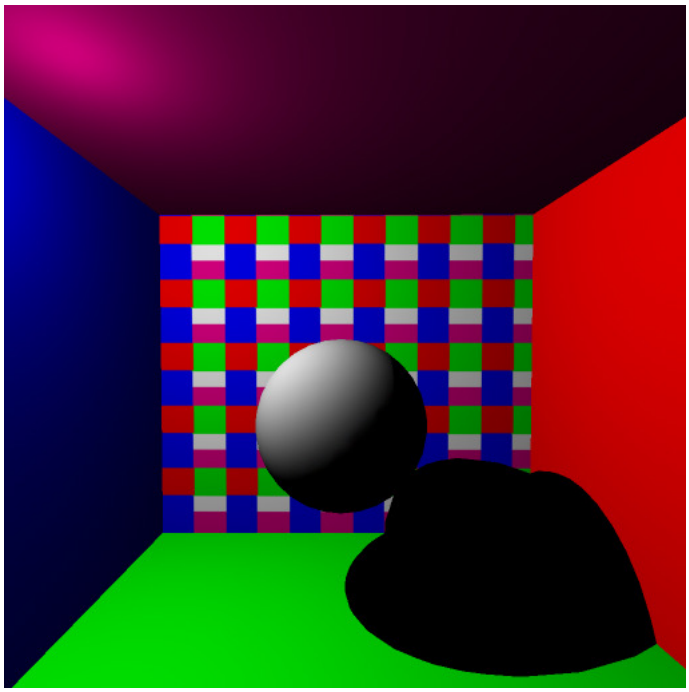
U ovom odeljku opisan je optički model koji je implementiran u sistemu i komponente iz kojih je taj optički model sastavljen.

4.1.1.3.1. Boja materijala, komponente difuzije i ogledanja

Kada se ustanovi presek zraka i primitiva u sceni, mora se odrediti boja onog dela površi primitive na kome dolazi do preseka. Prilikom određivanja boje primenjuje se određeni optički model. Od optičkih modela u razvijenom sistemu implementiran je **Phong** model senčenja. Implementacija i primena drugih modela (poput **Lambert, Gouraud, Blinn, Ward** itd) omogućena je specifikacijom apstraktne klase iz koje se svaki optički model može izvesti.

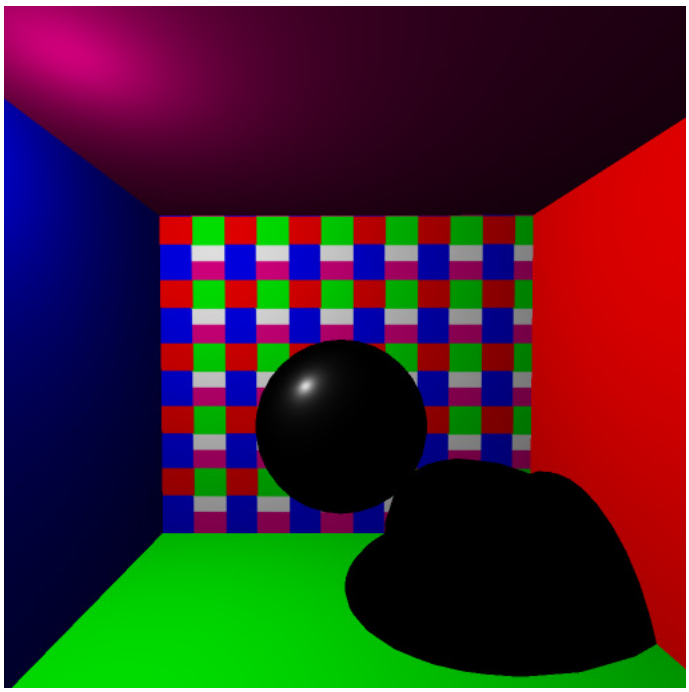
Phong senčenje je empirijski ustanovljen model koji nema direktnu fizičku interpretaciju. Sastoji se iz dve komponente koje doprinose boji: **difuziona** komponenta i komponenta **ogledanja** (eng. *specularity*). Obe komponente modeliraju dva načina na koje se svetlo reflektuje od površi poligona zadatog materijala.

Difuziona komponenta (*Slika 2*) predstavlja aproksimaciju ponašanja koje ispoljavaju svi materijali u manjoj ili većoj meri, a to je uniformno reflektovanje energije svetla, u svim pravcima.



Slika 2. Difuziona komponenta

Kod određenih materijala refleksija svetla je najintenzivnija u pravcu reflektovanog zraka svetla, odnosno na pravcu koji vodi od izvora svetla ka površi geometrijske primitive. Ovi materijali imaju visok stepen sjajnosti. Svetlo se od sjajnog materijala reflektuje tako da je ulazni ugao svetla (ugao pod kojim svetlo pada na površ u odnosu na normalu te površi) jednak izlaznom uglu. Ta komponenta Phong senčenja naziva se

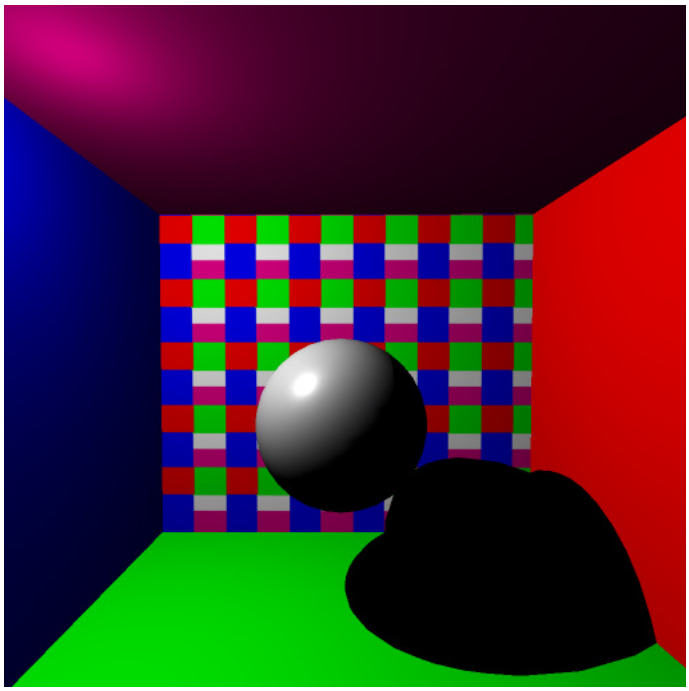


Slika 3. Komponenta ogledanja

komponentom ogledanja (*Slika 3*). Nivo komponente ogledanja koju vidi posmatrač zavisi od ugla pod kojim posmatrač posmatra datu površinu.

Deo površine objekta na kome je komponenta ogledanja vidljiva naziva se vrhunac (eng. *highlight*). Iako naziv komponente ogledanja sugeriše da bi se odgovarajući materijal mogao ponašati kao sekundarni izvor svetla (kao posledica ogledanja), to nije slučaj. Posebna komponenta reguliše refleksivnost materijala i opisana je u narednom odeljku.

Prilikom određivanja komponente ogledanja kao parametar se ne koristi boja materijala

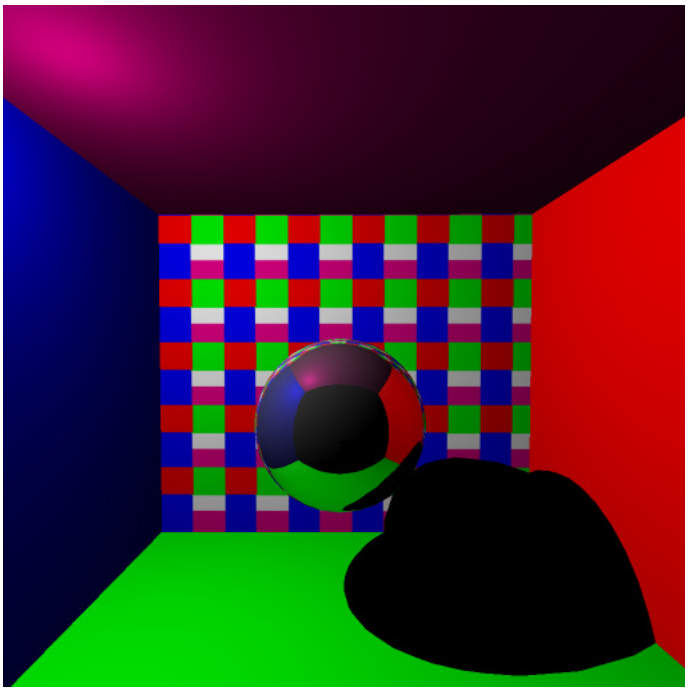


Slika 4. Zbir difuzione komponente i komponente ogledanja

nego boja svetla. Doprinos obe komponente (*Slika 4*) u sistemu može se kontrolisati pomoću koeficijenata difuzije i koeficijenta ogledanja. Koeficijenti komponenti mogu uzimati realnu vrednost u opsegu [0, 1].

4.1.1.3.2. Refleksija

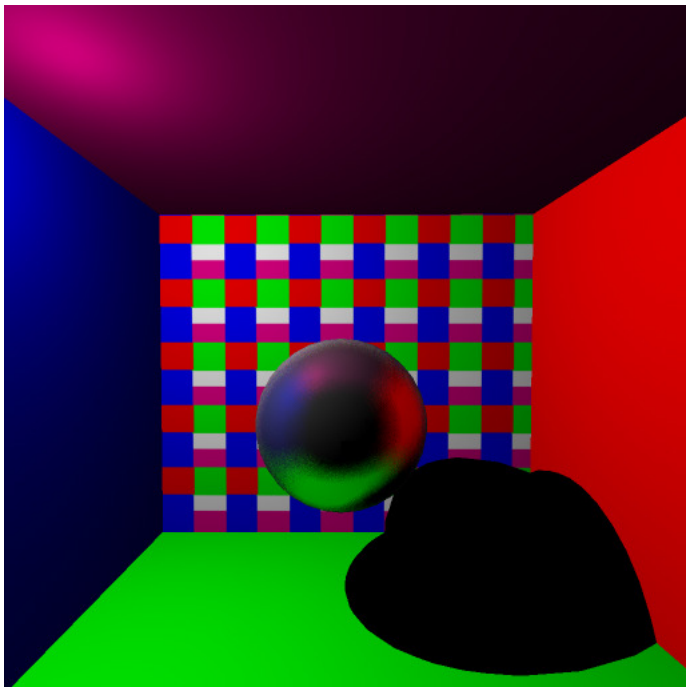
Reflektivnost materijala (*Slika 5*) modelirana je koeficijentom reflektivnosti. Ovaj koeficijent je realan broj u opsegu [0, 1]. Ako materijal, koji pripada objektu za koji se ustanovi presek sa primarnim zrakom, ima osobinu reflektivnosti (tačnije, koeficijent refleksije veći od 0), onda se dalje prati reflektovani zrak, od tačke preseka. Novi smer zraka se računa na osnovu **bidirekcione refleksivno-distribucione funkcije** (**BRDF**). S obzirom na to da je teoretski moguće da se ovakav zrak reflektuje beskonačno mnogo puta, omogućena je parametarska kontrola maksimalnog broja refleksija po zraku.

. 

Slika 5. Reflektivan materijal

4.1.1.3.3. Difuziona refleksija

Difuzione refleksije (*Slika 6*) predstavljaju nesavršene refleksije. Umesto da materijal reflektuje svetlo u tačno određenom smeru u zavisnosti od napadnog smera svetla, kod difuzionih refleksija dolazi do rasipanja odbijenog zraka u više smerova. Koeficijent difuzione refleksije materijala definiše prostorni ugao u okviru koga dolazi do rasipanja

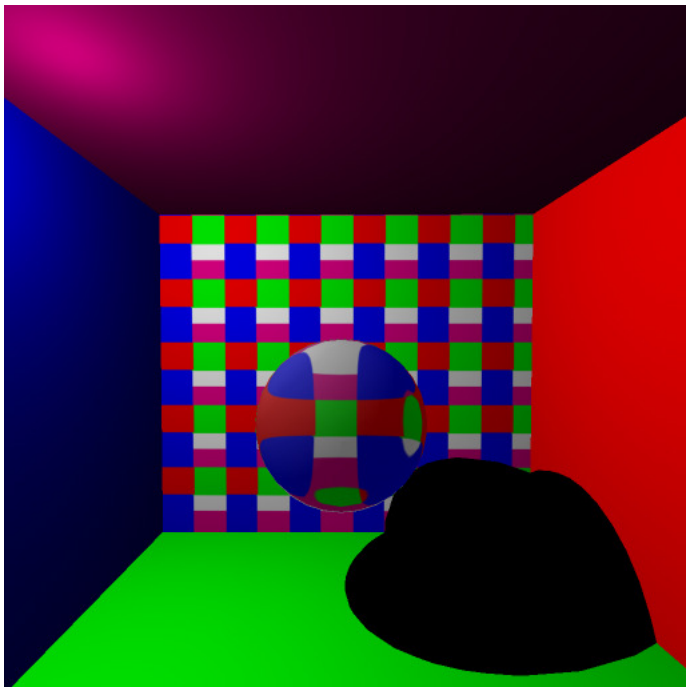


Slika 6. Difuziono reflektivni materijal

refleksije u odnosu na smer savršene refleksije koja se određuje pomoću **BRDF**. Može se reći da taj koeficijent definiše odstupanje odbijenih zraka u odnosu na smer savršene refleksije. U slučaju ove refleksije, umesto formiranja jednog reflektovanog zraka, formira se više zraka, čiji se smerovi generišu na osnovu smera savršene refleksije i koeficijenta refleksije. Broj zraka koji će biti formiran direktno utiče na kvalitet generisane slike, ali uz potencijalno veliki pad performansi. Zbog toga je omogućeno definisanje maksimalnog broja zraka koji će biti formirani na mestu nesavršene refleksije, čime će korisnik moći da pravi kompromis između brzine i kvaliteta prikaza.

4.1.1.3.4. Refrakcija

Refraktivnost (*Slika 7*) je opisana koeficijentom refraktivnosti i indeksom refrakcije. Koeficijent refraktivnosti predstavlja realan broj, u opsegu [0,1], koji određuje doprinos boji piksela boje prelomljenog zraka. Indeks refrakcije određuje ugao pod kojim će se zrak prelomiti i on je u opsegu [0,1]. Slično kao kod refleksije,



Slika 7. Refraktivni materijal

prilikom preseka zraka sa refraktivnim materijalom mora se odrediti novi smer prelomljenog zraka. To se računa na osnovu smera ulaznog zraka, njegovog ugla u odnosu na normalu onog dela površi gde dolazi do refrakcije i indeksa refrakcije materijala. Prilikom računanja doprinosa boji prelomljenog zraka, uzima se u obzir i dužina puta koju je prelomljeni zrak prešao. Doprinos osvetljenju eksponencijalno opada sa porastom pređenog puta. Svako sledeće slanje zraka, od zadnjeg preseka sa nekom primitivom, ima sve manje uticaja na osvetljenost, tako da se posle određenog broja slanja zraka, taj zrak prekida. Maksimalan broj slanja se zadaje kao parametar.

4.1.2. Teksture

U sistemu je omogućeno korišćenje tekstura umesto boje materijala. Tekstura je definisana kao interfejs, sa ciljem da se omogući upotreba ne samo tekstura koje se sastoje od bitmape nego i npr. proceduralno generisanih tekstura. U okviru ovog sistema realizovana je softverska implementacija, namenjena za centralni procesor, a ne za grafički orijentisan hardver koji je optimizovan za rad sa teksturama. Zbog toga veliki uticaj na performanse sistema ima broj korišćenih tekstura u sceni i način uzorkovanja teksture.

4.1.3. Primitive i složeni objekti

U sistemu su podržane jednostavne primitive i složeni objekti koji su sastavljeni od skupa primitiva.

4.1.3.1. Primitive

Osnovni geometrijski oblici su podržani kroz interfejs `Primitive`. Implementirane su dve osnovne primitive:

- **Sfera**, koja je realizovana kao idealna sfera (zadata centrom i poluprečnikom). Normala sfere i koordinate teksture u nekoj tački sfere izračunavaju se dinamički. Koordinate teksture *u* i *v* izračunavaju se tako što se one posmatraju redom kao geografska širina i dužina u odnosu na referentni meridijan i paralelu.
- **Trougao**, koji je modeliran svojim temenima. Temena sadrže poziciju, normalu i koordinate teksture.

4.1.3.2. Složeni objekti

Složeni objekti su sastavljeni od skupa trouglova i za njih je predviđena klasa `Object3D`. Trouglovi u okviru složenih objekata mogu deliti temena, a samim tim i njihove normale. Kada susedni trouglovi dele temena, složeni objekat dobija zaobljen izgled, zato što se kao normala temena ne koristi normala jednog od tih trouglova, već usrednjena normala svih trouglova u čijem sastavu se nalazi dato teme.

Učitavanje složenih objekata je omogućeno iz dve vrste fajla: OBJ i 3DS, mada se preferira učitavanje iz OBJ fajla, jer on sadrži više podataka o složenim objektima. OBJ fajl sadrži normale za svako teme poligona, za razliku od 3DS fajla, koji sadrži samo normalu poligona.

4.2. Optimizacija

U ovom odeljku će biti objašnjene najčešće primenjivane tehnike za ubrzavanje sistema za praćenje zraka koje su implementirane u sistemu predstavljenom u ovom radu. Optimizacija je podeljena u dve grupe, softversku i hardversku. Sistem je projektovan tako da jednostavno pruži podršku primeni drugih tehnika.

4.2.1. Softverske tehnike za ubrzavanje

Softverskoj optimizaciji i ubrzanju sistema pristupa se povećanjem efikasnosti na dva načina:

- **Ubrzavanje algoritma određivanja tačke preseka primitive i zraka.** Tradicionalni sistemi za praćenje zraka pokazali su da se najveći deo procesorskog vremena potrebnog za sintezu slike koristi za određivanje tačke preseka primitiva i zraka. Zato je mnogo napora bilo uloženo da bi se pronašli brži i efikasniji algoritmi za rešavanje tog problema. Svaki od algoritama ima različite osobine, kao što su broj potrebnih operacija nad celim brojevima, operacija nad realnim brojevima, broj uslova, potrošnja memorije, preciznost itd.
- **Prostorna i hijerarhijska podela scene.** Efikasan način za ubrzanje **praćenja zraka** jeste smanjenje broja testova preseka zraka i primitiva korišćenjem određenih struktura za prostornu podelu scene, koje omogućavaju brz pristup primitivama koje se nalaze u blizini zraka. To omogućava značajno smanjenje potrebnog broja testova, što je napredak u odnosu na naivnu implementaciju osnovnog algoritma, koja bi svaki zrak testirala sa svim primitivama u sceni. Postoji veći broj struktura podataka koje su našle primenu u ovoj optimizaciji. One se dele na dva osnovna tipa:
  - **Hijerarhija obuhvatajućih zapremina (eng. *bounding volume*):** organizuje primitive scene u hijerarhijski odnos koji pojednostavljuje i ubrzava pretragu.
  - **Prostorna podela (eng. *space partitioning*):** vrši hijerarhijsku podelu prostora.

4.2.1.1. Ubrzavanje algoritma preseka primitive i zraka

Nalaženje preseka trougla i zraka zasniva se na traženju tačke preseka **opisane baricentričnim koordinatama trougla**. Za određivanje preseka zraka i trougla koristiće se projekcija trougla na dve ravni koordinatnog sistema (npr. XZ i ZY), čime se problem određivanja preseka svodi na rešavanje jednačina sa dve nepoznate. Ubrzanje se zasniva i na prekalkulaciji (koja se obavlja samo jednom) i čuvanju potrebnih međurezultata za nalaženje preseka. Ovaj algoritam za određivanje preseka trougla i zraka osmislio je *Ingo Wald* u svom doktorskom radu[6].

4.2.1.2. Prostorna podela scene

U predstavljenom sistemu implementirana je struktura podataka pod nazivom *Kd-stablo*. Kd-stablo je binarno stablo u kojem svaki čvor koji nije list implicitno deli prostor na dva pod-prostora pomoću ravni koje su paralelne osnovnim ravnima koordinatnog sistema (XY, XZ ili YZ ravnima). Tačke koje se nalaze levo od ravni podele predstavljaju levo podstablo, a tačke desno od ravni podele predstavljaju desno podstablo. U slučaju da je izabrana ravan normalna na X osu, levi pod-prostor činiće tačke manje vrednosti koordinate X od pozicije ravni, dok će desni pod-prostor činiti tačke koje imaju veću vrednost X koordinate od vrednosti koordinate pozicije ravni. Analogno važi za ravni normalne na Y i Z ose. Da li će tačke koje leže tačno na ravni podele biti uključene u

levo ili desno pod-stablo jeste stvar implementacije i nije presudno za postupak. Ako se složenije primitive raspoređuju, one mogu istovremeno pripadati levom i desnom pod-stablu. Praksa je pokazala da ova struktura daje najbolje rezultate ako se prostor deli adaptivno, pomoću heuristične funkcije SAH (*Surface Area Heuristic*) [referenca]. Ova funkcija vrši procenu cene podele prostora na dva nova pod-prostora. Mesto za podelu prostora bira se kao ono koje ima najnižu cenu podele. SAH je izražena sledećom funkcijom:

```
površina= 2 * (širina * visina + visina * dubina + dubina * širina)
cena = cena obilaska + površina * broj primitiva * cena preseka
```

Cena obilaska predstavlja zahtevnost algoritma obilaska Kd stabla.a Cena preseka predstavlja zahtevnost algoritama testa preseka primitive i zraka . Ako je algoritam testa preseka primitive i zraka duplo sporiji od algoritma obilaska Kd-stabla onda će cena preseka biti duplo veća od cene obilaska. Ova dva parametra određuju se eksperimentalno. Problem sa ovom funkcijom i uopšte sa konstrukcijom Kd-stabla jeste posećivanje svih mogućih pozicija podele prostora, kojih ima tačno dva puta više nego primitiva.

Algoritam za obilazak Kd-stabla je predložio *Vlastimil Havran*[32] u svom doktorskom radu. Implementacija u sistemu opisanom u ovom radu oslanja se na implementaciju Kd-stabla *Jacco Bikkera*[33], koja je modifikovana da bi podržavala interaktivni sadržaj.

4.2.2. Optimizacija za multiprocesorske sisteme

Obrada svakog piksela slike je nezavisna od obrade ostalih, pa je moguće razložiti sliku na disjunktne delove i vršiti njenu sintezu u različitim nitima programa. Ovo omogućava da se posao raspodeli na sve prisutne procesore u multiprocesorskom sistemu i na sva jezgra jednog procesora. Ova optimizacija ima smisla samo ako je u datom računaru dostupno više procesora ili procesor sa više jezgara. Da bi se postigao ovaj rezultat, korišćena je platformski nezavisna *BOOST*[3] biblioteka za jezik C++, koja omogućava formiranje niti. Nitima se prilikom njihovog kreiranja prosleđuju informacije o delovima slike na kojima će raditi i zajedničke scene koje će iscrtavati. Izbor broja niti u kojima će se vršiti sinteza slike biće konfigurabilan parametar.

5. Unapređene tehnike za interaktivni sadržaj

5.1. Priprema prostorne podele za iscrtavanje

Priprema prostorne podele za iscrtavanje predstavlja konstrukciju Kd-stabla za datu scenu. U sistemu je podržana dinamička promena sadržaja i pozicije objekata scene, zbog čega je potrebno rekonstruisati Kd-stablo scene svaki put kada dođe do promene u sceni usled toka animacije. Inkrementalno ažuriranje stabla nije praktično, zato što se posle relativno malog broja ažuriranja gubi korist od uvođenja stabla [34]. Kao glavni problem se ističe to da, kada se neka od primitiva pomera u sceni, mora doći do rekonstrukcije celog Kd-stabla, a cena tog postupka raste eksponencijalno sa porastom broja primitiva u sceni. To znači da će brzo, sa rastom broja primitiva u sceni, cena

rekonstrukcije strukture postati prevelika. Obim interaktivnog sadržaja scene može biti ograničen na mali broj poligona koji je prihvatljiv za rekonstrukciju celokupnog stabla, ali to ograničava složenost scene i samim tim kvalitet generisane slike.

Kao rešenje u ovom radu se koristi **pod-scena**, koja ima svoje Kd-stablo. Pod-scena predstavlja zasebnu scenu koja je obuhvaćena **obuhvatnom kutijom**. Obuhvatna kutija se učitava iz OBJ fajla zajedno sa pod-scenom koju obuhvata, tako da se ona modelira zajedno sa pod-scenom koju okružuje. Ovo rešenje problema je već bilo primenjivano u sličnim sistemima [6]. Implementacija se u ovom radu ne oslanja na već razvijane implementacije, već je realizovana nova implementacija služeći se samo idejom koja je već primenjivana. Umesto da se neki objekat direktno smešta u scenu i samim tim u Kd-stablo glavne scene, on se smešta u zasebnu pod-scenu koju obuhvata obuhvatna kutija. Obuhvatna kutija je takođe složen objekat tipa Object3D [refernca], sastavljen od trouglova, koji se smešta u glavnu scenu, a samim tim u glavno Kd-stablo. Nije neophodno da obuhvatna kutija bude oblika kvadra, već može uzeti bilo koji oblik, ali je termin "kutija" tradicionalno u upotrebi. Trouglovi koji sačinjavaju graničnu kutiju se ne crtaju i služe samo za definisanje geometrije kutije. Umesto informacija o materijalu, trouglovi obuhvatne kutije sadrže pokazivač na pod-scenu koju ta kutija obuhvata. Tako da se odmah, prilikom preseka zraka i obuhvatne kutije, može nastaviti obilazak zraka po pod-sceni. Glavna scena može sadržati i obuhvatne kutije i trouglove za iscrtavanje ili samo jedno od ta dva. Da bi uvođenje obuhvatne kutije imalo smisla, geometrija sadržana u pod-sceni u okviru obuhvatne kutije mora biti složenija od geometrije obuhvatne kutije. Što je složenija pod-scena, to će i korist od uvođenja obuhvatne kutije biti izraženija. Za pod-scenu se jednom pravi pod-Kd-stablo, prilikom inicijalizacije glavne scene.

Kada prilikom prolaska zraka po sceni dođe do preseka zraka i obuhvatne kutije onda se vrši obilazak Kd-stabla pod-scene. Tom prilikom nije neophodno transformisati primitive pod-scene u koordinatni sistem glavne scene, nego se vrši transformacija zraka u koordinatni sistem pod-scene, čime se dodatno povećava efikasnost sistema praćenja zraka.

5.2. Hijerarhijska animacija i prostorna podela

Mogu se izdvojiti dve vrste animacija: **nestrukturne** i **hijerarhijske**. Pod nestrukturnom animacijom podrazumeva se animacija u kojoj se primitive ili složeni objekti kreću jedni u odnosu na druge po pseudoslučajnim putanjama. Kao primer nestrukturne animacije može se navesti animacija eksplozije nekog objekta. Sa druge strane nalaze se hijerarhijske animacije u kojima se primitive ili složeni objekti kreću jedni u odnosu na druge uz poštovanje nekih pravila i ograničenja.

U razvijenom sistemu hijerarhijska animacija je implementirana u vidu skeletnog sistema. Skelet se sastoji od zglobova čije je kretanje definisano u odnosu na druge zglobove. Sistem zglobova je dat u vidu stabla, gde svaki zglob ima svog "roditelja" (izuzev zgloba u korenu stabla) i njegovo kretanje je dato u koordinatnom sistemu zgloba-roditelja. To implicira da su položaji zglobova definisani u okviru roditeljskih koordinatnih sistema. Položaj i animacija korenog zgloba dati su u odnosu na

koordinatni sistem scene. Da bi se dobio trenutni položaj nekog zgloba u okviru koordinatnog sistema scene, mora se utvrditi položaj svih zglobova od korenog zgloba do zgloba čiji se položaj traži.

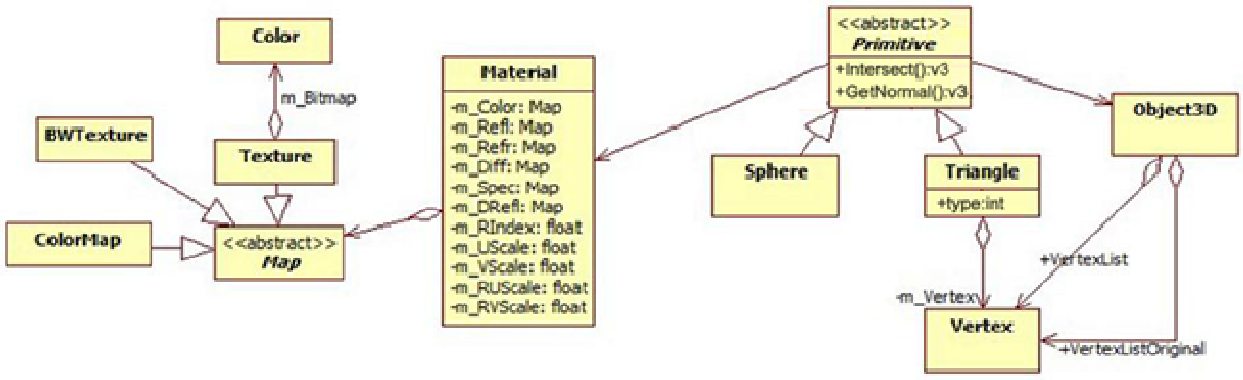
Svakom zglobu je pridružena obuhvatna kutija koja sadrži pod-scenu. Prilikom animacije zgloba nije potrebno transformisati primitive u koordinatnom sistemu pod-scene, već samo obuhvatne kutije u koordinatnom sistemu glavne scene. Ako se tokom generisanja slike, prlikom obilaska zraka po sceni, dođe do preseka zraka i obuhvatne kutije zgloba (nakon njegove transformacije), koordinate zraka se transformišu u koordinatni sistem obuhvatne kutije zgloba, kao što je objašnjeno u odeljku *5.1*.

6. Projekat softvera

UML model programskog sistema praćenja zraka napravljen je korišćenjem besplatnog alata *StarUML* [34].

6.1. Primitive i materijali

Iz apstraktne klase `Primitive` izvode se sve klase primitiva koje se iscrtavaju, a u slučaju realizovanog sistema izvode se dve klase: `Sphere` i `Triangle`. Najvažnije dve metode, koje sve primitive moraju da podrže, jesu metode `Intersect` i `GetNormal`. `Intersect` nalazi presek između zraka i primitive, a `GetNormal` normalu primitive na mestu preseka sa zrakom. U sistemu trougao se može iscrtavati kao deo scene ili biti deo obuhvatne kutije, u kom slučaju se on ne iscrtava. Da li će trougao biti iscrtavan ili će biti deo obuhvatne kutije, sistem određuje na osnovu polja `type` u okviru klase `Triangle`.



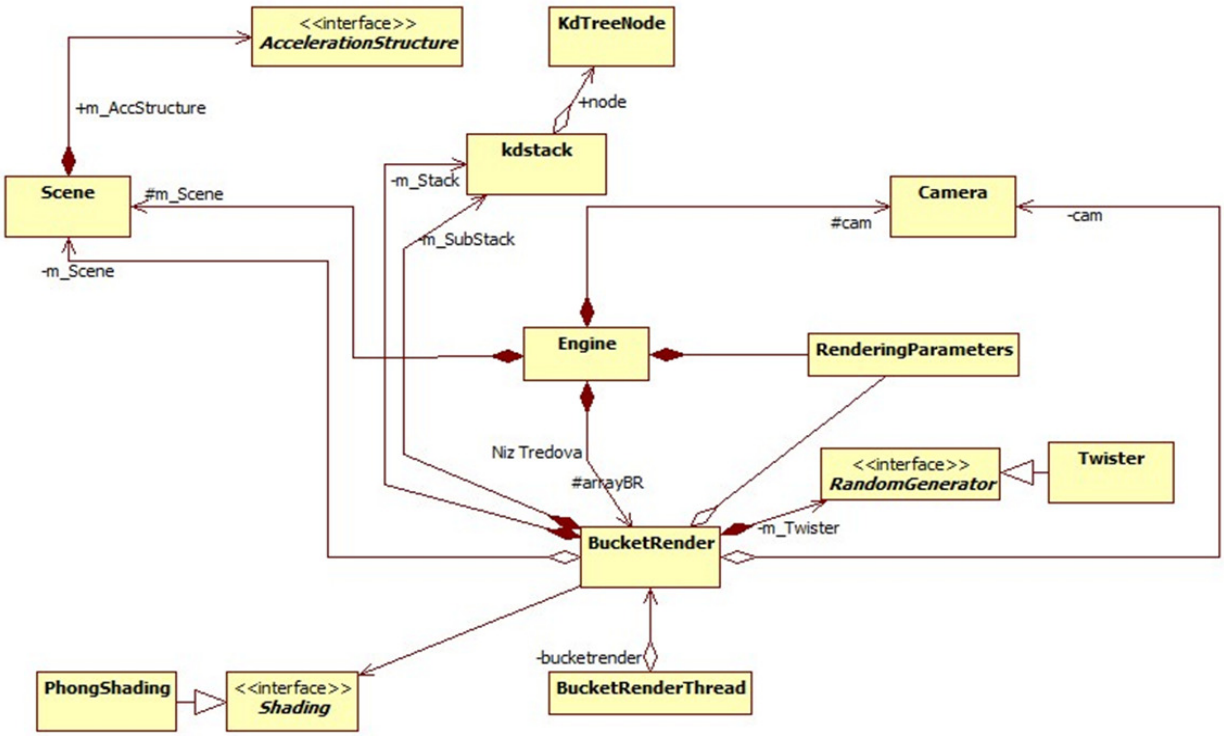
Slika 8. Pregled UML dijagrama Primitive i Materijala

Klasa `Material` sadrži sve parametre koje definišu osobine materijala. Implementirane su metode koje dohvataju svaki parametar pojedinačno, a parametri koji se prosleđuju tim metodama jesu koordinate tekstura (U,V i W). Svaki od tih parametara predstavljen je `Map` apstraktnom klasom. Da bi objekat, tipa materijal, bio validan, on mora sadržati sve parametre. Za svaki parametar klase `Material` mora biti kreiran objekat odgovarajuće klase (`ColorMap`, `BWTextureili Texture`). Svaka izvedena klasa iz

apstraktne klase `Map` mora podržati metodu `GetTexel`, koja vraća vrednost traženog parametra na osnovu koordinata teksture (U,V i W). U sistemu su podržane tri klase izvedene iz apstraktne klase `Map`. Klasa `Texture` predstavlja RGB bitmapu i njena metoda `GetTexel` vraća vrednost pomoću tehnike bilinearnog filtriranja. `BWTexture` predstavlja crno-belu bitmapu i njena metoda `GetTexel` vraća vrednost pomoću tehnike bilinearnog filtriranja. `ColorMap` predstavlja jedinstvenu float vrednost koja se uvek vraća, bez obzira na vrednost koordinata teksture (U,V i W).

6.2. Jezgro sistema za sintezu slike

Klasa `Engine` je zadužena za kreiranje svih niti i održavanje glavne scene za iscrtavanje. Ona sinhronizuje rekonstrukciju glavne scene sa nitima koje iscrtavaju scenu. Takođe, preko nje se kontrolišu parametri kvaliteta iscrtane slike. Parametri kvaliteta iscrtavanja sadržani su u klasi `RenderingParameters`. Metoda `Render` klase `BucketRender` zadužena je za iscravanje jednog segmenta rezultujuće slike. Niti koje paralelno računaju podatke za pojedinačne piksele izvršavaju ovu metodu. Metoda je kontrolisana sa dva semafora, zato što se ona mora zaustaviti u toku ažuriranja Kd-stabla scene. Jednim semaforom se signalizira niti, iz glavnog programskog toka, da može da nastavi sa iscrtavanjem, dok se drugim semaforom signalizira glavnom programskom toku da je nit završila crtanje tekućeg frejma. Na osnovu atributa objekta kamere, nit generiše zrake koji obilaze Kd-stablo glavne scene.



Slika 9.Pregled UML dijagrama jezgra sistema

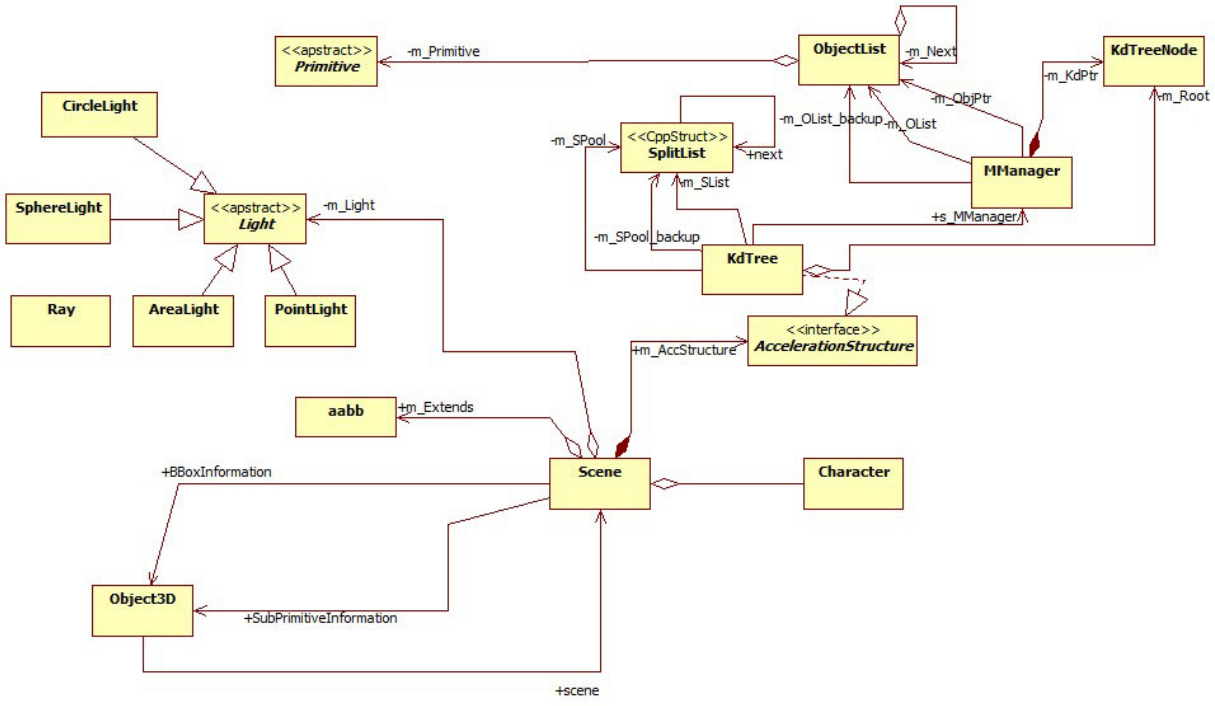
Atributi klase `Camera` jesu: pozicija, orijentacija, distanca do ravni projekcije i veličina prozora koji se nalazi u ravni projekcije. Obilazak Kd-stabla relizovan je iterativno

(pomoću steka). Zbog toga svaka nit koja se kreira pomoću `BucketRender` klase sadrži stek za obilazak Kd-stabla. Osim tog steka, svaka nit sadrži i stek za obilazak zraka po pod-sceni.

Za sve tehnike koje koriste *Monte Karlo* metod numeričke integracije potrebno je realizovati generator slučajnih brojeva, što je i urađeno pomoću apstraktne klase `RandomGenerator`, iz koje je izvedena klasa *Twister*, koja predstavlja *Mersenne*-ov [35] generator pseudo-slučajnih brojeva. `BucketRenderThread` klasa jeste omotač za `BucketRender`, koji omogućava rad sa nitima pomoću *BOOST* [3] biblioteke.

6.3. Sadržaj scene i ubrzavajuće strukture

Klasa `Scene` sadrži informacije o primitivama, animiranim modelima, svetlima i ubrzavajućoj strukturi koja je pridružena svakoj `Sceni`. `InitScene` metoda klase `Scene` zadužena je za formiranje glavne scene (tačnije, za učitavanje *OBJ* fajla). Pomoćna klasa aabb sadrži samo informacije o minimalnoj i maksimalnoj veličini koordinata trouglova koji mogu biti sadržani u sceni. U okviru *OBJ* fajla, objektima scene moraju biti data imena na unapred utvrđen način. Imenima objekata za iscrtavanje dat je sufiks OBJ3D, dok je njihovim obuhvatnim kutijama dat sufiks BBox. Deo imena svetla mora biti saglasan sa tipom svetla: *PointLight* u slučaju tačkastog izvora svetla, *AreaLight* u slučaju pravougaonog izvora svetla, *SphereLight* u slučaju sfernog izvora svetla i *CircleLight* u slučaju kružnog izvora svetla. Na primer, ako u sceni imamo tačkasti izvor svetla u OBJ fajlu biće zapisan objekat koji sadrži parametre tog svetla, a ime tog tačkastog svetla mora se završiti sa *PointLight* dok prvi deo imena svetla može biti proizvoljan.



Slika 10. Pregled UML dijagrama Scene, KDTree i Light

Object3D predstavlja skup trouglova, ali klasa Object3D ih ne sadrži diretkno, već se trouglovi svih Object3D objekata scene nalaze u jednom nizu klase Scene. Klasa

Object3D sadrži indeks prve i poslednje svoje primitive u tom nizu. Dve liste temena trouglova su sadržane u okviru klase Object3D. Jedna lista, VertexListOriginal, predstavlja netransformisana temena, dok druga lista, VertexList, predstavlja temena posle afinih transformacija. To je urađeno tako zato što animacije nisu date u vidu inkrementalnih transformacija, nego u vidu konačne (rezultujuće) transformacije. VertexList je lista temena koja se prikazuje. Object3D ima pokazivač na instancu klase Scene u kojoj je sadržan da bi moglo jednostavno da se pristupi trouglovima koji sačinjavaju Object3D. Kao što je ranije napomenuto, u zavisnosti od tipa trouglova od kojih je Object3D sačinjen, on može biti objekat za iscrtavanje ili obuhvatna kutija pod-scene.

AccelerationStructure predstavlja strukturu za ubrzanje nalaženja preseka primitiva i zraka u sceni. U sistemu Kd-stablo je implementirano kroz klasu KdTree. Klase izvedene iz interfejsa AccelerationStructure moraju implementirati sledeće metode:

- FindNearest nalaženje najbližeg preseka zraka i primitive u glavnoj sceni.
- FindNearestSub nalaženje najbližeg preseka zraka i primitive u pod-sceni.
- FindOcculder testiranje da li se bilo koja primitiva nalazi na putanji zraka.
- FindOcculderSub testiranje da li se bilo koja primitiva nalazi na putanji zraka u pod-sceni.
- Build generisanje strukture od primitiva sadržanih u sceni.

Podatak tipa Light definisan je svojom pozicijom i bojom. Klase izvedene iz apstraktne klase Light moraju obezbediti metodu GetNextPos koja vraća poziciju sledećeg uzorka svetla. Uzorak svetla predstavlja tačku na površi svetla u slučaju kompleksnih oblika svetla. Kompleksni oblici svetla se aprkosimiraju određenim brojem uzoraka. Uzorci se uzimaju na slučajan način, sa površine kompleknsog svetla, pomoću generatora slučajnih brojeva koja je data klasom Twister.

Klasa MManager je zadužena za kreiranje, održavanje i brisanje lista primitiva i listova Kd-stabla prilikom kreiranja Kd-stabla neke scene. ObjectList upravo predstavlja liste koje MManager održava. Elementi liste su u memoriji smešteni sekvencijalno, tj. smešteni su u dva niza sadržana u okviru Kd-stabla. Elementi liste su tako smešteni zbog kompaktnosti podataka u memoriji. Time su poboljšane performanse zahvaljujući keširanju, jer procesor pristupa sekvencijalno smeštenim podacima.

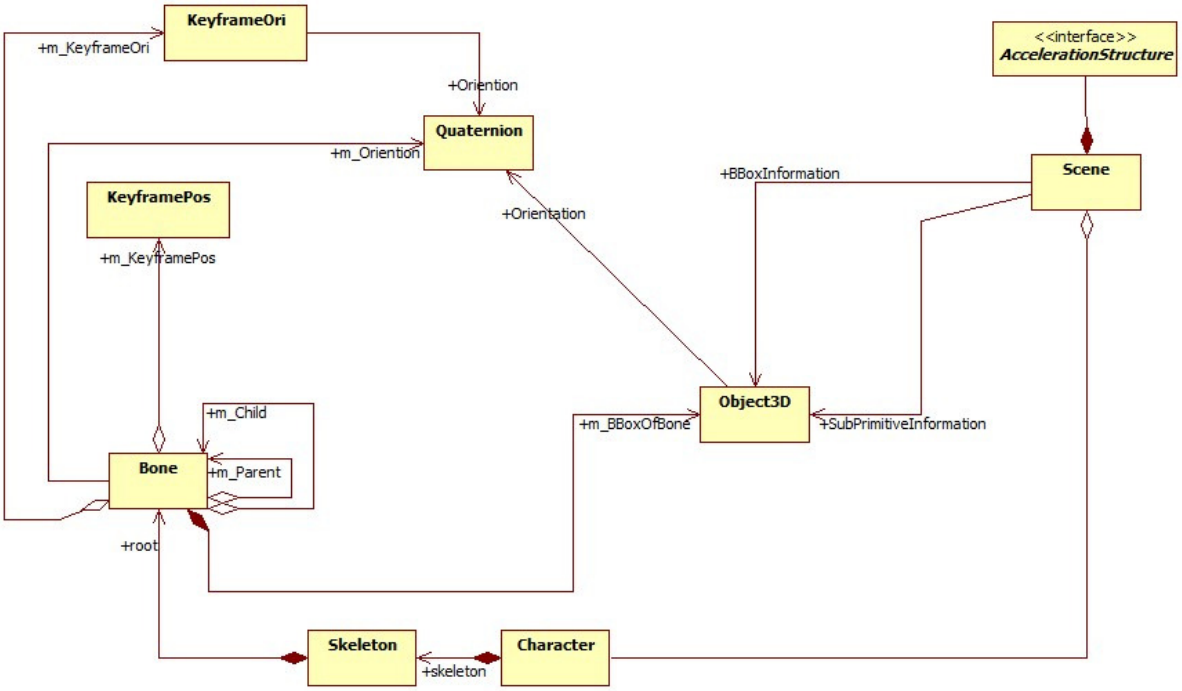
6.4. Deo sistema za rad sa animiranim sadržajem

Bone je definisana svojim relativnim položajem u odnosu na roditeljski Bone objekat. Taj položaj je određen pozicijom i orijentacijom u koordinatnom sistemu roditelja. Objekat Bone može imati samo jednog roditelja i može biti roditelj za više Bone objekata. Svakom Bone objektu je pridružena tačno jedna obuhvatna kutija. Obuhvatna kutija

sadrži informacije o pod-sceni koju okružuje. Svaki `Bone` kontroliše položaj i orijentaciju pod-scene transformišući položaj obuhvatne kutije koja mu je pridružena. Transformacijom položaja i orijentacije obuhvatne kutije u glavnoj sceni ujedno se transformiše položaj i orijentacija pod-scene u glavnoj sceni.

Klasa `Skeleton` je zadužena za kreiranje `Bone` hijerarhije. Ona sadrži metode koje ažuriraju celokupnu `Bone` hijerarhiju nekog animiranog modela.

Klasa `Character` je zadužena za celokupno održavanje i upravljanje animiranog modela: učitavanje animacija i kontrolu toka animacija. Iz ove klase bi trebalo dalje izvoditi razne animirane objekte: od ljudskog lika kojim se može upravljati do nekih objekata koji se pomeraju po prethodno definisanim putanjama.



Slika 11.pregled UML dijagrama Skeleton i Character

Pozicija u sistemu je data klasom `v3` koja predstavlja vektor u trodimenzionalnom prostoru. Za tu je klasu definisano mnoštvo metoda i operatora kojima se olakšava rad sa vektorima. Ova klasa nije prikazana na prethodnim dijagramima. Orijentacije i rotacije u sistemu su implementirane kvaternionima (eng. *quaternion*). Operacije nad kvaternionima su implementirane u klasi `Quaternion`.

7. Implementacija

Sistem *praćenja zraka* je implementiran na programskom jeziku *C++* korišćenjem razvojnog okruženja *Microsoft Visual Studio 2010*. Glavni razlog za upotrebu programskog jezika *C++* jeste veća brzina sistemskog koda u odnosu na kod dobijen drugim programskim jezicima. Razvojno okruženje *Microsoft Visual Studio 2010*

olakšava i ubrzava pisanje programskog koda. Ovo okruženje je posebno pogodno za efikasno pisanje objekno orijentisanog koda i njegovo debagovanje.

Da bi se omogućio rad sistema praćenja zraka i na drugim operativnim sistemima, biblioteka za rad sa nitima izabrana je na osnovu tog kriterijuma. Biblioteka *BOOST*, za rad sa nitima, jeste multiplatformska i njena rasprostranjenost je velika. Ova biblioteka ima odličnu podršku i visoko je optimizovana, tako da se njenim korišćenjem ne gubi na brzini koda.

Komunikacija sa operativnim sistemom, u smislu grafičkog okruženja, vrši se pomoću *Glut* i *OpenGL* API grafičkih biblioteka. Obe biblioteke su multiplatformske i nezavisne od platforme na kojoj se kod izvršava. U sistemu se ne koriste napredne funkcije *OpenGL*-a, već se on koristi samo za prikazivanje generisane slike u sistemu praćenja zraka. Glut biblioteka se koristi za kreiranje korisničkog prozora na kome se prikazuje generisana slika i daju još neke informacije vezane sa generisanu sliku, a koristi se i za komunikaciju sa korisnikom preko tastature i miša.

Modeli i scena koji su korišćeni u sistemu realizovani su pomoću *3ds Max 2010* [referenca] i *Rhinoceros 3D* [36] grafičkih programa. Svi modeli su eksportovani u OBJ format [referenca], koji je lak za korišćenje i učitavanje u sistem. Ovaj je format pogodan i zato što zadržava bitne informacije modela, kao što je normala u svakoj tački 3d modela, koje se u nekim drugim formatima gube. Izvorni kod za čitanje OBJ fajla dolazi uz SDK *Rhinoceros 3D* programa.

Animacije modela realizovane su pomoću *MilkShape3D* programa [referenca] i one se učitavaju iz MS3D fajla istog programa. Kod za učitavanje podataka iz MS3D fajla napisao je *Mete Ciragan* [referenca] i distribuira se uz aplikaciju *MilkShape3D*.

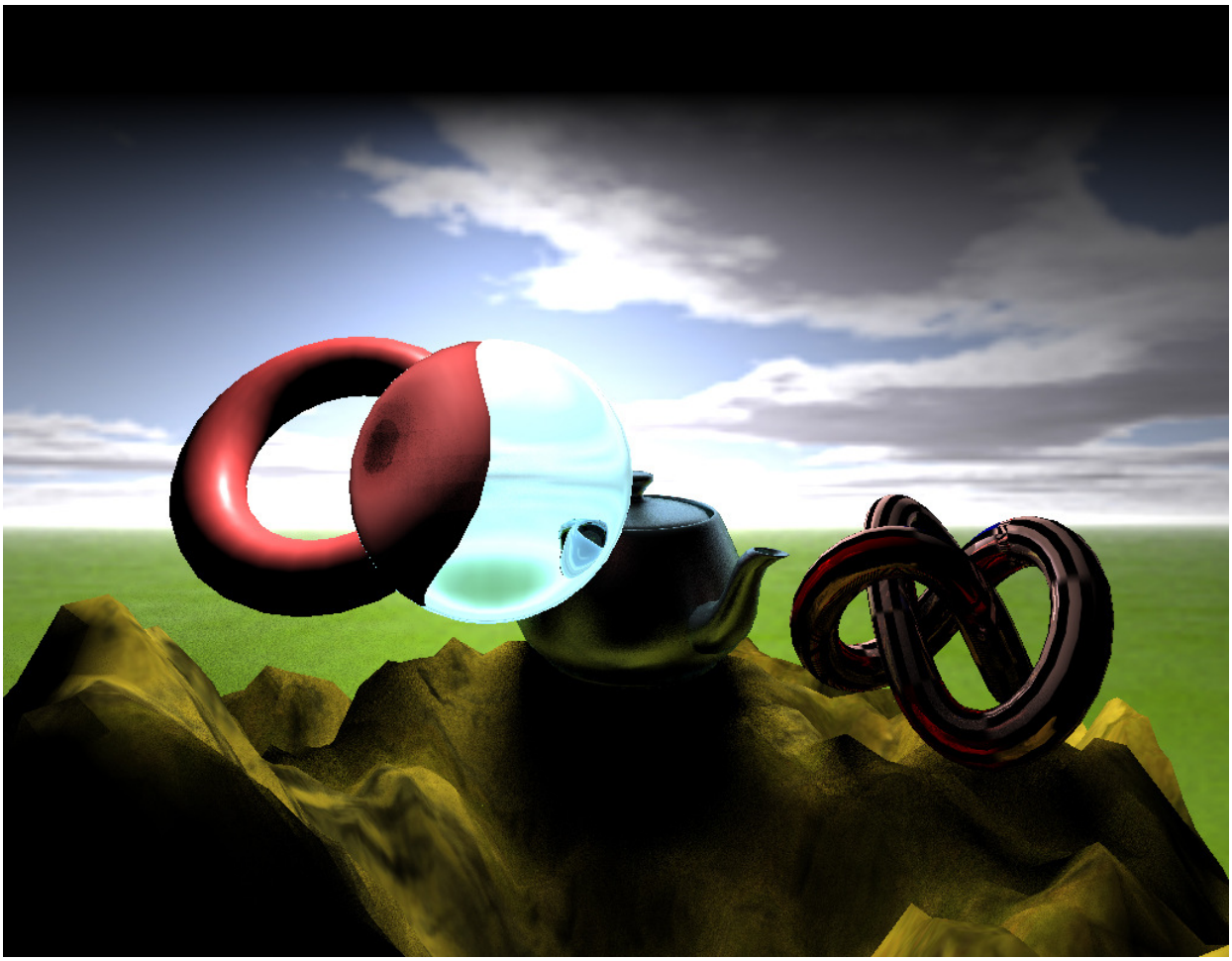
Izvorni kod programskog sistema praćenja zraka sadrži oko 12.000 linija i 79 klasa.

8. Demonstracioni programi

U sklopu demonstracije mogućnosti razvijenog sistema praćenja zraka razvijena su tri demonstraciona programa. Razvijeni programi su osmišljeni da bi demonstrirali i ispitali mogućnosti razvijenog sistema praćenja zraka bez interaktivnog sadržaja ili sa njim. Programi su gotovo identični osim dela koda koji upravlja interaktivnim sadržajem, ako ima interaktivnog sadržaja u demonstracionom programu.

8.1. Složene primitive iznad terena

Svrha ovog demonstracionog programa (*Slika 12*) jeste da prikaže optičke mogućnosti sistema praćenja zraka. Materijali objekta su tako konfigurisani da su u materijalima sadržane sve komponente boje (difuziona i ogledanja), koje omogućava sistem. Čajnik u sceni je od materijala koji demonstrira difuzione refleksije. Heliks u sceni je od materijala čiju reflektivnost kontroliše tekstura koja mu je pridružena kao reflektivna mapa. Lopta demonstrira mogućnosti materijala koji prelamaju svetlost.



Slika 12. Primitive iznad terena

U ovom demonstracionom programu objekti su statični, dok se interaktivno može uticati na položaj kamere. Scena je osvetljena jednim svetlom, tip svetla se može menjati u tačkasto ili površinsko, a na položaj svetla može se uticati preko komandi. Može se uticati i na kvalitet slike promenom parametara sistema: dubine paćenja zraka, kvaliteta difuzionih refleksija i kvaliteta mekih senki. Scena demonstracionog programa sastavljena je od 10K trouglova.

8.2. Simulacija vožnje

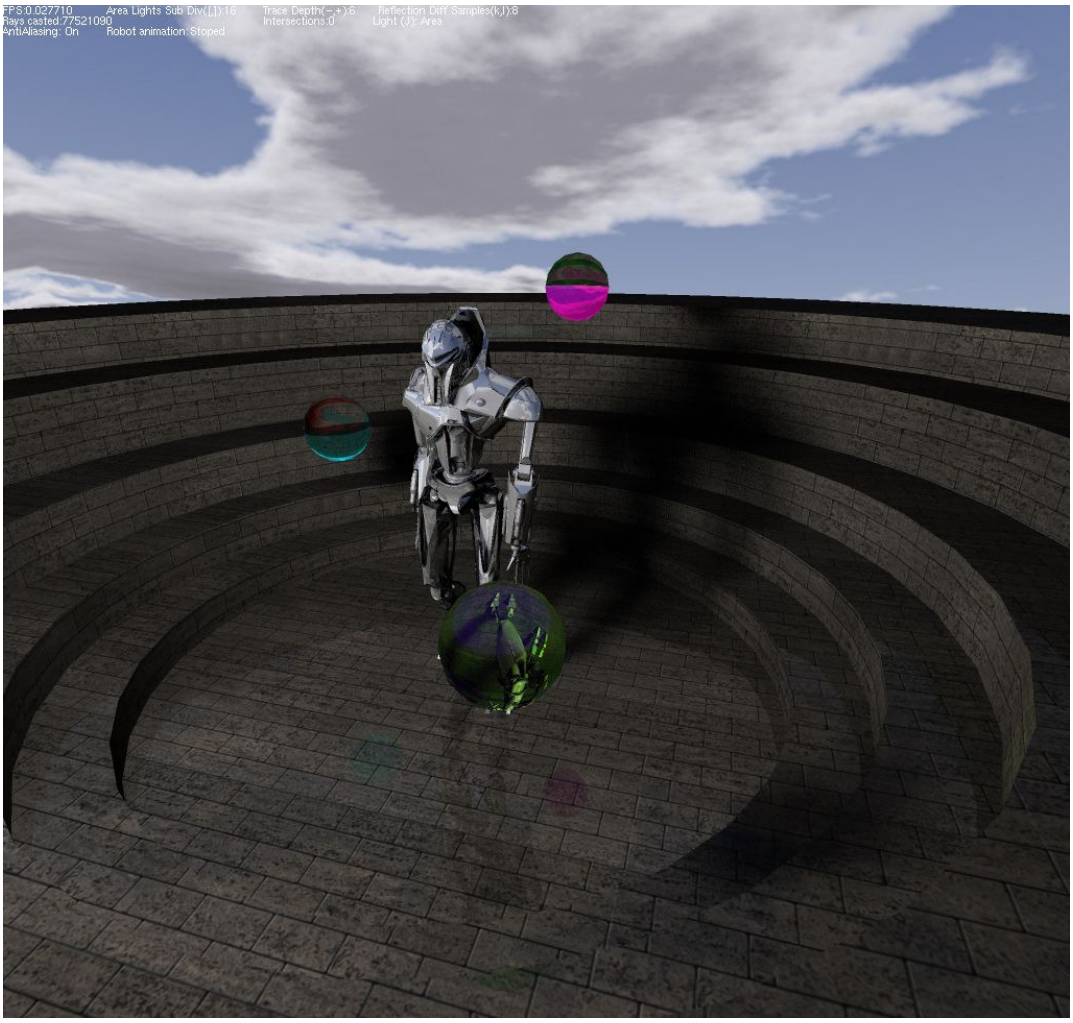
Svrha ovog demonstracionog programa (*Slika* 13*14*) jeste da prikaže mogućnosti interaktivnog sadržaja u sistemu praćenja zraka. Korisnik ove aplikacije, osim već navedenih mogućnosti iz prethodnog demonstracionog programa, može kontrolistati i automobil u sceni. Automobil u sceni je obojen materijalom koji sadrži difuzionu komponentu i komponentu ogledanja. Prozori automobila su refraktivni, dok je telo reflektivno. Točkovi automobila se okreću kako se automobil kreće, a prednji točkovi se

usmeravaju u zavisnosti od skretanja automobila. Scena demonstracionog programa sastavljena je od 20K trouglova.

Slika 13. Simulacija vožnje

8.3. Hijerarhijska animacija robota

Svrha ovog demonstracionog programa (*Slika* 1415) jeste da prikaže hijerarhijske animacije u okviru sistema praćenja zraka. Korisnik ovog programa upravlja animacijama robota. Animacije se učitavaju iz *MilkShape 3D* [5] fajla. Robot je obojen reflektivnim materijalom, dok lopte sa od prozirnog materijala sa prelamanjem svetla kruže oko robota. Korisnik može preko komandi odabratii neku od animacija koju će robot ponavaljati ciklično; korisnik takođe može zadati komandu robotu kojom će on proći kroz sve moguće animacije učitane iz fajla. Scena demonstracionog programa sastavljena je od 40K trouglova.



Slika 14. Animacija robota

8.4. Merenje performansi sistema praćenja zraka

Demonstracioni programi su testirani na računaru sa Intel i7 920 procesorom i 6GB memorije. Procesor na sebi ima četiri jezgra koja rade na 2.66 GHz. Svako jezgro izvršava paralelno dve niti, tako da sistem prijavljuje kao da procesor ima 8 jezgara.

Performanse je teško uporediti sa već razvijenim sistemima praćenja zraka zato što su scene koje su korišćene u demonstracionim programima iscrtavane samo na ovom sistemu praćenja zraka. Da bi tačnije uporedili performanse razvijenog sistema, sa nekim sličnim sistemom praćenja zraka, bilo bi potrebno da oba sistema iscrtavaju istu scenu.

Performanse sistema su merene preko broja frejmova u sekundi (*eng. FPS frames per second*) koje demonstracioni program postiže. Merenja su izvršena za više rezolucija kada je u sceni bio prisutan jedan tačkasti izvor svetla. Izvršena su i merenja za jednu rezoluciju kada je u sceni bio aktiviran jedan površinski izvor svetla ili je bila aktivirana opcija Anti-Aliasinga u demonstracionom programu.

8.4.1. Rezultati merenja

Merenja su predstavljena u vidu tabela. Rezultati predstavljaju broj frejmova u sekundi (FPS) koje demonstracioni program postiže za razne vrednosti parametara sistema praćenja zraka. Vrednost kvadrata niti predstavlja veličinu (visina X širina u broju piksela), disjunktnog kvadrata dela slike koju nit iscrtava.

a) Složene primitive iznad terena

FPS		Rezolucija(px)							
		128X128	256X256	512X512	1024X1024	Površinskio svetlo	512X512	AntiAliasing	512X512
Veličina kvadrata niti	8X8	21.1	9.1						
	16X16	20.3	8.2	2.5			0.53		0.68
	32X32	17.2	6.4	2.4	0.69		0.52		0.63
	64X64	10.7	5.8	2.2	0.63		0.49		0.45
	128X128	7.1	3.5	1.6	0.60		0.31		0.43
	256X256		2.1	0.9	0.57		0.23		0.25
	512X512			0.6	0.25		0.10		0.14
	1024X1024				0.15				

b) Simulacija vožnje

FPS		Rezolucija(px)							
		128X128	256X256	512X512	1024X1024	Površinskio svetlo	512X512	AntiAliasing	512X512
Veličina kvadrata niti	8X8	8.2	4.1						
	16X16	7	4.0	3.0			0.82		0.95
	32X32	6.7	3.8	2.8	1.12		0.81		1.01
	64X64	6	3.5	2.7	1.06		0.71		0.98
	128X128	3.7	2.6	2.6	0.94		0.61		0.88
	256X256		1.4	2.4	0.84		0.43		0.71
	512X512			1.0	0.65		0.19		0.26
	1024X1024				0.25				

c) Hijerarhijska animacija robota

FPS		Rezolucija(px)							
		128X128	256X256	512X512	1024X1024	Površinskio svetlo	512X512	AntiAliasing	512X512
Veličina kvadrata niti	8X8	19.6	10.5						
	16X16	20.0	10.7	3.67			0.43		1.05
	32X32	19.2	9.8	3.90	1.10		0.44		1.06
	64X64	15.0	9.6	3.64	0.92		0.42		1.14
	128X128	10.0	7.6	3.75	1.05		0.39		0.76
	256X256		3.4	2.75	1.00		0.26		0.92
	512X512			0.96	0.80		0.09		0.25
	1024X1024				0.25				

9. Zaključak

U radu je predstavljen sistem praćenja zraka u kome su primenjeni javno dostupni algoritmi i tehnike praćenja zraka. Pored toga, ispitani su i istraživani algoritmi i tehnike za optimizaciju sistema praćenja zraka, predloženi u otvorenoj literaturi. Ti novi algoritmi i tehnike razvijani su sa ciljem podrške za interaktivne aplikacije koje koriste sistem za praćenje zraka, na široko dostupnim računarskim sistemima. Za razliku od većine sistema praćenja zraka razvijeni sistem podržava interaktivni sadržaj. Pri tome su implementirane tehnike koje smanjuju uticaj interaktivnog sadržaja na performanse sistema. Kvalitet slike je zadovoljavajući i može se porediti sa kvalitetom slike sličnih sistema. Performanse koje su postignute na oglednom računaru su zadovoljavajuće, ali ukazuju na postojanje prostora za dodatna unapređenja. Sistem je tako projektovan da se omogući njegovo dalje lako razvijanje. Nove tehnike i algoritme je lako ugraditi u projektovan sistem.

Dalji razvoj sistema praćenja zraka je omogućen zahvaljujući tome što je projektovan na način da se jednostavno mogu zameniti ili unaprediti postojeće komponente sistema. U sledećoj fazi razvoja sistema mogu se uvesti nove vrste prostorne podele scene, kako bi se istraživale mane i prednosti tih prostornih podela. Moguće je podržati u sistemu i druge geometrijske oblike, kao što su primitive definisane matematičkim formulama ili složeniji oblici sastavljeni od jednostavnijih primitiva.

Performanse koje su postignute na oglednom računaru mogu se oceniti zadovoljavajućim. Moguća je dodatna optimizacija koda za specifične procesore i njihove mogućnosti. Na primer, na Intelovim procesorima najnovije generacije poboljšanje performansi može se ostvariti optimizacijom koda za Intelove SSE instrukcije za rad sa vektorima.

Kao i svi grafički sistemi, sistem predstavljen u ovom radu koristi teksture prilikom sinteze slike. Kao jedan od pravaca daljeg razvoja, dodatna optimizacija se može postići upotrebom hardvera za uzorkovanje tekstura. Hardver koji bi se koristio mogu biti već postojeće grafičke kartice, koje u okviru svojih GPU procesora sadrže specijalizovane jedinice za rad sa teksturama ili se te specijalizovane jedinice za rad sa teksturama mogu ugraditi u hibridna GPU/CPU rešenja poput Intelovog Larrabee procesora [37].

Drugi smer ubrzanja sistema bi bila upotreba masivnih paralelnih procesora, koji su dostupni na modernim grafičkim adapterima. Jedan od mogućih izbora platforme jeste multiplatformska biblioteka OpenCL [38], koja pruža API za implementaciju paralelnih algoritama, bilo da se izvršavaju na centralnom ili grafičkom procesoru. Prenos sistema tako da se izvršava na grafičkiom adapteru nije jedonstavan, jer se paradigme programiranja CPU računara i GPU grafičkog adaptera značajno razlikuju.

10. Literatura

[1] Microsoft Visual Studio 2010
http://www.microsoft.com/visualstudio/

[2] OpenGL
http://www.opengl.org/

[3] BOOST
http://www.boost.org/

[4] 3ds Max 2010
http://images.autodesk.com/adsk/files/3dsmax2010_brochure_us.pdf

[5] MilkShape 3D
http://chumbalum.swissquake.ch/

[6] Ingo Wald,«Realtime Ray Tracing and Interactive Global Illumination», PhD thesis, Computer Graphics Group, Saarland University, 2004

[7] Arthur Appel, «Some Techniques for Shading Machine Renderings of Solids», 1968

[8] Turner Whitted, «An Improved Illumination Model for Shaded Display», 1979

[9] Kajiya, J T, «The rendering equation», Proceedings of the 13th annual conference on Computer graphics and interactive techniques, ACM, 1986

[10] R.L. Cook, T. Porter, and L.Carpenter , «Distributed Ray Tracing», 1984

[11] James F. Blinn,«Models of light reflection for computer synthesized pictures», 1977

[12] B. T. Phong,«Illumination for computer generated pictures»,Communications of ACM 18 (1975)

[13] R. Cook and K. Torrance,«A reflectance model for computer graphics». Computer Graphics (SIGGRAPH '81 Proceedings)

[14] K. E. Torrance and E. M. Sparrow,«Theory for off-specular reflection from roughenedsurfaces», Journal of the Optical Society of America,1967

[15] Ward, «Measuring and Modeling Anisotropic Relfection», Proc. ACM SIGGRAPH 1992

[16] Michael Ashikhmin, Peter Shirley,«An Anisotropic Phong BRDF Model», Journal of Graphics Tools, 2000

[17] E. Lafortune, S. Foo, K. Torrance, and D. Greenberg, «Non-linear approximation of reflectance functions», SIGGRAPH 97 Conference Proceedings, Annual Conference Series

[18] RenderMan
http://www.renderman.pixar.com/

[19] Blender
http://www.blender.org/

[20] Brazil
http://www.splutterfish.com/

[21] Maxwell
http://www.maxwellrender.com/

[22] Mental Ray
http://www.mentalimages.com/

[23] POV-Ray
http://www.povray.org/

[24] V-Ray
http://www.chaosgroup.com/

[25] OpenRT
http://www.openrt.de/

[26] Enemy Territory: Quake Wars,Intel
http://http://www.idfun.de/qwrt/

[27] OptiX
http://www.nvidia.com/

[28] ArtVps
http://http://www.artvps.com/

[29] SaarCOR
http://en.wikipedia.org/wiki/SaarCOR/

[30] Caustic Graphics
http://www.caustic.com/

[31] Monte_Carlomethods
http://en.wikipedia.org/wiki/Monte_Carlo_method

[32] Vlastimil Havran
http://www.cgg.cvut.cz/members/havran/phdthesis.html

[33] Jacco Bikker,Jacco Bikker,
http://www.devmaster.net/articles/raytracing_series

[34] StarUML
http://staruml.sourceforge.net/

[35] Mersenne twister
http://en.wikipedia.org/wiki/Mersenne_twister

[36] Rhinoceros 3D
http://www.rhino3d.com/

[37] Intel Larrabee
http://en.wikipedia.org/wiki/Larrabee_%28microarchitecture%29

[38] OpenCL
http://www.khronos.org/opencl/