

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

Odjel za Matematiku
Sveučilišni preddiplomski studij matematike i
računarstva

ZAVRŠNI PRAKTIČNI PROJEKT - DOKUMENTACIJA

Globalna Iluminacija

Autor:

Antonio Janjić

Mentor:

doc. dr. sc. Domagoj
Ševerdija

Osijek, 2022.

Sadržaj

1	Opis projekta	1
2	Početak rada	1
2.1	Ovisnosti	1
2.2	Preuzimanje	1
2.3	Instaliranje	2
2.3.1	Windows	2
2.3.2	Linux	2
2.4	Korištenje programom	2
3	Implementacija	3
3.1	Opis rada	3
3.2	Namespaces	4
3.3	Klase	4
3.4	Funkcije	11
4	Za developere	12
4.1	Stil koda	12
4.2	Konvencije nazivlja	12
4.3	Organizacija koda	13
5	Galerija	14
6	Budući planovi	19
7	Licence	20
	Literatura	21

1 Opis projekta

Ovaj projekt je namijenjen kao implementacija raytracer-a sa globalnom iluminacijom, prvenstveno kao što je opisano u knjizi "*Advanced Global Illumination*" [1].

Projekt je u potpunosti implementiran u C++ jeziku.

2 Početak rada

2.1 Ovisnosti

Ovaj projekt ovisi o sljedećem softveru, kodovima i bibliotekama. Njihova instalacija prije kompiliranja programa je nužna za rad.

- *premake5* za automatsku izradu konfiguracija.
- *OpenMP* je potreban za paralelno izvršavanje u release buildu programa.
- *GLM* se koristi za vektore, matrice i operacije nad istim.
- Pomoćne klase i funkcije su uzete [iz ove biblioteke](#)
- *stbimage*-jevi headeri za učitavanje i spremanje slika

2.2 Preuzimanje

Klonirajte repozitorij pomoću

```
git clone --recursive https://github.com/Lame-een/globalillum
```

Ako je repozitorij bio kloniran ne rekurzivno ranije, koristite

```
git submodule update --init
```

kako bi se klonirali potrebni podmoduli.

2.3 Instaliranje

2.3.1 Windows

Pokrenite *premake5* pored *premake5.lua* datoteke sa *vs2022* argumentom kako bi kreirali potrebne Visual Studio datoteke za development i za kompiliranje.

2.3.2 Linux

Pokrenite *premake5* pored *premake5.lua* datoteke sa *gmake* argumentom kako bi kreirali Makefile, koji se može koristiti za kompiliranje programa.

Zadana konfiguracija bi trebala producirati release build. Konfiguracija se može odabrati postavljanjem make-ovog *config* argumenta na *config=release* ili *config=debug*.

2.4 Korištenje programom

Prema zadanim postavkama program će se izgraditi u
./bin/x64/%configuration% direktoriju.

Program se može pokrenuti u komandnoj liniji korištenjem sljedećih argumenta:

`GIrt.exe [OPTIONS] output_image_name`

Postavke programa se mogu zadati dodavanjem *OPTIONS* zastava.

Popis svih zastava:

- Postake raytracer-a:

<code>-help</code>	Prikaže zaslon s pomoći.
<code>-s, -samples</code>	Postavi broj uzoraka po pikselu. (1024)
<code>-depth</code>	Postavi maksimalnu dubinu rekurzije. (25)
<code>-w, -width</code>	Postavi širinu slike. (640)
<code>-h, -height</code>	Postavi visinu slike. (640)
<code>-fov</code>	Postavi Field Of View kamere (u stupnjevima). (60)
<code>-bg, -background</code>	Postavi pozadinu scene u hex RGB. ("#3a3a3a")
<code>-bgimg, -background-image</code>	Postavi pozadinu scene na day ili night . (<i>none</i>)
<code>-gamma</code>	Postavi korekciju game za sliku. (2.2)
- Postavke za primjere:

<code>-demo</code>	Odaberi jedan od postojećih primjera scene.
<code>-demo-list</code>	Izlistaj sve postojeće primjere scena.

(Napomena: Sve zastave i postavke su osjetljive na veliko i malo slovo.)

Izlazna slika će se nalaziti u *./out* direktoriju. Direktoriji će biti stvoren po-kraj same executable datoteke.

Primjer korištenja:

`GIrt.exe -help` – Prikaže sve moguće CLI argumente.

`GIrt.exe -w 250 -h 250 -demo 0 demo_0_render` – Postavi širinu i visinu slike na 250 piksela, započne renderanje primjera broj 0 i spremi sliku kao "demo_0_render.png"

3 Implementacija

3.1 Opis rada

Program je raytracer koji koristi Monte Carlo integraciju za aproksimiranje globalne iluminacije. Kao u standardnom raytraceru zrake svijetla se šalju iz točke koja simulira senzor kamere ili retinu oka. Zrake se šalju kroz svaki piksel i uz to za svaki piksel se pošalje zadani broj uzoraka zraka koje su sve blago perturbirane unutar piksela.

Svaka zraka se provjeri na presjek sa objektom ili pozadinom scene. Pri pre-sjeku sa objektom informacije o hitcu su spremljene u *HitInfo* klasi i materijal objekta će vratiti podatke o raspršenju zraka pomoću *ScatterInfo* klase.

Sa podatcima u *ScatterInfo* klasi možemo nastaviti rekurzivno slati zrake iz točke sjecišta oviseći o tome kakva je interakcija između zrake i materijala.

Odašiljane zrake nose informacije o boji/luminaciji natrag pomoću rekurzije, sve zrake po pikselu u prosjeku daju boju tog piksela slike.

Dodatno je implementirana i aproksimacija Fresnelovog efekta za prozirne materijale.

U nastavku ću napraviti kratak opis struktura, klase i funkcija koje su implementirane. Za potpune informacije pogledajte sam kod na [repozitoriju](#).

3.2 Namespaces

Colors

Colors namespace enkapsulira definirane konstante izraze za različite boje.

Omogućuje jednostavno korištenje osnovnih boja bez da se svaki puta moraju manualno unositi hex kodovi boja.

Primjer gdje se koristi namespace:

```

1 //primjer iz ./src/demos/cornellBoxWindow.h
2 ...
3 Diffuse matUpDown(StringToRGB("#808080"));
4 Diffuse matLeft(Colors::crimson);
5 ...

```

Vidimo kako za razliku od linije 3, u liniji 4 možemo iskoristiti samo naziv boje unutar namespacea *Colors*.

Settings

Ovaj namespace enkapsulira globalne varijable koje služe kao reprezentacija postavki i funkcije za rad nad postavkama.

Ova klasa se u budućnosti može pretvoriti u Singleton object, no za vrijeme izrade projekta bilo je jednostavnije koristiti namespace.

3.3 Klase

Object

Object klasa je bazna klasa za sve objekte koji se mogu nalaziti u sceni.

Deklarirana je na sljedeći način:

```

1 class Object
2 {
3     public:
4         Object(bool cull = true, bool samplingTarget = false);
5         Object(const Material* mat, bool cull = true,
6                 bool samplingTarget = false);
7
8         virtual bool Hit(const Ray& ray, double tMin,
9                           double tMax, HitInfo& hitInfo) const;
10
11        virtual bool BoundingBox(AABB& outputBox) const;
12
13        virtual Vec3 Random(Vec3 point) const;
14        virtual double PdfValue(const Vec3& origin, const Vec3& dir) const;

```

```

15     bool IsSamplingTarget() const;
16
17     const Material* GetMaterial() const;
18
19 protected:
20     const Material* m_Mat = nullptr;
21
22     bool m_Cull;
23     bool m_SamplingTarget;
24 };

```

Konstruktor vidljiv u liniji 5 prima pokazivač na materijal objekta, te dvije boolean vrijednosti: `cull` zadaje hoće li se vršiti *backface culling* nad objektu, `samplingTarget` zadaje hoće li se objekt dodatno uzimati u obzir pri uzorkovanju zraka.

```
virtual bool Hit(const Ray& ray, double tMin,
                  double tMax, HitInfo& hitInfo) const;
```

je funkcija koju je nužno implementirati u deriviranim objektima. Poziva se pri ispitivanju siječe li zraka `ray` sam objekt i vraća `true` ako siječe.

```
virtual bool BoundingBox(AABB& outputBox) const
```

se koristi za stvaranje kvadra poravnatog s osima koji omeđuje sam objekt.

```
virtual Vec3 Random(Vec3 point) const
```

se koristi za generiranje nasumičnog vektora koji pokazuje prema objektu.

```
virtual double PdfValue(const Vec3& origin, const Vec3& dir) const
```

se koristi za generiranje funkcije gustoće vjerojatnosti za objekt pri uzorkovanju.

Klase **Sphere**, **Triangle**, **TriangleMesh** su derivirane klase klase **Object**. Dok klasa **Quad** je derivirana klasa klase **TriangleMesh** i implementira trodimenzionalni četverokut. Za više detalja o pojedinim klasama pogledajte repozitorij.

Material

Material je bazna klasa pomoću koje se mogu stvoriti materijali objekata.

```

1  class Material
2  {
3      public:
4          Material(bool isEmissive = false,
5                  bool isTransmissive = false);
6          virtual RGB Emit(const Ray& incidentRay,
7                             const HitInfo& hitInfo) const;
8
9          virtual bool Scatter(const Ray& incidentRay,
10                           const HitInfo& hitInfo,
11                           ScatterInfo& scatterInfo) const;
12          virtual double ScatterPDF(const Ray& incidentRay,
13                                     const HitInfo& hitInfo,
14                                     const Ray& scatterRay) const;
15
16         static const Material* Default();
17
18         bool IsEmissive() const;
19         bool IsTransmissive() const;
20
21     private:
22         bool m_Emissive;
23         bool m_Transmissive;
24
25         static Material s_DefaultMat;
26     };

```

Konstruktor ove klase prima dva argumenta. *isEmissive* deklarira može li ovaj materijal emitirati svjetlost, dok *isTransmissive* deklarira hoće li ovaj objekt propuštati zrake.

```
virtual RGB Emit(const Ray& incidentRay,
                  const HitInfo& hitInfo) const
```

je funkcija koja se implementira u materijalima koji emitiraju svjetlost. Za sve ostale materijale može ostati ne implementirana i preuzeta iz bazne klase.

```
virtual bool Scatter(const Ray& incidentRay,
                     const HitInfo& hitInfo,
                     ScatterInfo& scatterInfo) const
```

jer virtualna funkcija koja se implementira u konkretnim materijalima i zadaje kako će se zraka *incidentRay* raspršiti pri hitcu. Rezultantna zraka se sprema u *scatterInfo*.

```
virtual double ScatterPDF(const Ray& incidentRay,
                          const HitInfo& hitInfo,
                          const Ray& scatterRay) const
```

vraća vrijednost funkcije gustoće vjerojatnosti za pojedini materijal.

Trenutno su implementirani sljedeći materijali: **Diffuse**, **Specular**, **Transmissive** i **Light**. Svaka pojedina od tih klasa derivira je od Material klase i implementira virtualne funkcije na prikladan način.

AABB

Osno poravnati granični okvir (eng. Axis-Aligned Bounding box) je klasa koja opisuje kvadar koji omeđuje neki objekt. Koristi se za ubrzanje određivanja pogotka zrake sa objektom.

```
1 class AABB
2 {
3     public:
4         AABB();
5         AABB(const Vec3& min, const Vec3& max);
6
7         const Vec3& Min() const;
8         const Vec3& Max() const;
9
10        const void Set(const Vec3& min, const Vec3& max);
11
12        bool Hit(const Ray& r, double tMin, double tMax) const;
13        void Surround(const AABB& box);
14
15    private:
16        Vec3 m_Min = Vec3(0);
17        Vec3 m_Max = Vec3(0);
18 }
```

Konstruktor ove klase prima dvije točke u 3D prostoru koje predstavljaju "najmanju" i "najveću" točku kvadra.

```
bool Hit(const Ray& r, double tMin, double tMax) const
```

vraća prolazi li zraka r kroz okvir. Funkcija je implementirana po uzoru na Andrew Kensler-ovu implementaciju [2].

```
void Surround(const AABB& box)
```

je funkcija koja uspoređuje okvir koji poziva funkciju sa box okvirom. Funkcija određuje okvir koji opisuje ta dva te postavlja trenutni na taj veći.

BVHNode

Klasa *BVHNode* je derivirana klasa klase *Object*. Njezina uloga je kao građivni dio za hijerarhiju ograničavajućeg volumena (eng. Bounding Volume Hierarchy - BVH).

BVH je reprezentacija objekata u nekoj dimenziji pomoću stabla. Ovdje je korištena za brzi pronađak pogotka zrake sa objektima u sceni.

BVH je implementiran na način da se konstruira binarno stablo pri čemu su listovi objekti unutar scene. Pri grananju se nasumično odabire dimenzija po kojoj će se uspoređivati dva *AABB*-a i tako se određuje hoće li skup objekata biti u lijevom ili desnom djeletu stabla.

Za detalje vidite potpunu implementaciju u ". /src/objects/bvh.cpp"

HitInfo i ScatterInfo

Ove dvije strukture se koriste za prijenos podataka o presjeku zrake i objekta, te o raspršenju zrake pri hitcu od objekt.

```
1 struct HitInfo
2 {
3     public:
4         Vec3 point;
5         Vec3 normal;
6         const Object* object = nullptr;
7         double t;
8 }
```

point je točka u kojoj je došlo je presjeka.

normal je normala na točku presjeka.

object je pokazivač na objekt koji je pogodjen.

t je dužina zrake koja je pogodila objekt.

```
1 class ScatterInfo
2 {
3     public:
4         ~ScatterInfo();
5
6         Ray bounceRay;
7         bool isSpecular;
8         RGB color;
```

```

9
10     const PDF* Pdf() const;
11     void SetPdf(PDF* ptr);
12 private:
13     const PDF* m_pdf = nullptr;
14 };

```

bounceRay je zraka koja se stvorila pri raspršenju.

isSpecular specificira je li ova zraka prozirna ili spekularna.

color je boja zrake.

m_pdf sadrži funkciju gustoće vjerojatnosti zrake.

Camera i Viewport

Ove dvije klase implementiraju kameru i prozor za prikaz slike (eng. viewport) scene.

Camera je jednostavna klasa koja se konstruira pomoću sljedećeg konstruktora:

```
Camera(const Vec3& pos, const Vec3& dir, const Vec3& up)
```

pos je pozicija kamere, dir je smjer u kojem kamera gleda, up je vektor smjera koji izlazi iz gornje strane kamere.

Viewport je klasa koja služi za projekciju zraka na "površinu unutar kamere". Zapravo je struktura koja će spremati podatke o slici koja se želi kreirati (širina i visina te FOV) i uz kameru pomaže za operacije koje su potrebne za određivanje zraka koje se šalju u scenu.

Image

Image klasa sadrži sve nužne opracije nad ulaznim i izlaznim slikama. Implementirana je na sljedeći način:

```

1 class Image{
2 public:
3     Image(size_t width, size_t height);
4     Image(const std::string& path);
5     ~Image();
6
7     void SetPixel(int x, int y, const RGB& color);
8     RGB GetPixel(int x, int y);
9     RGB GetPixel(double u, double v);
10

```

```

11     const Vec2i& Dimensions();
12     bool IsLoaded() const;
13     bool HasData() const;
14
15     void WriteFile(const std::string& path);
16 private:
17     Vec2i m_Dimensions;
18     double m_Gamma;
19     bool m_Loaded;
20
21     unsigned char* m_Bitmap = nullptr;
22 };

```

Vidimo kako postoje dva konstruktora. Prvi

`Image(size_t width, size_t height)`

kreira praznu sliku širine `width` i visine `height`. Pikseli se mogu lagano bojati koristeći

`void SetPixel(int x, int y, const RGB& color)`

funkciju. Tako kreirana slika se zatim može spremiti kao datoteka na putanji `path` koristeći:

`void WriteFile(const std::string& path)`

Drugi konstruktor:

`Image(const std::string& path)`

učitava sliku sa putanje `path`.

Cubemap

Cubemap klasa enkapsulira 6 slika koje čine "environment map" scene. Koristi se vrlo lako. Konstruktor klase prihvata putanju do direktorija u kojoj se nalazi 6 slika koje čine cubemap. Sa tako konstruiranom klasom uz pomoć funkcije

`RGB Sample(const Vec3& dir) const`

koja prima vektor smjera može se uzorkovat boja koja se nalazi u "okolišu" u tom smjeru.

PDF

PDF je bazna klasa za implementiranje funkcije gustoće vjerojatnosti.

```

1 class PDF
2 {
3 public:
4     virtual double Value(const Vec3& dir) const = 0;

```

```
5     virtual Vec3 Generate() const = 0;
6 }
```

Deklarira dvije čiste virtualne funkcije koje se trebaju implementirati u deriviranim klasama. Trenutne implementirane bazne klase su: **CosinePDF**, **SpecularPDF**, **ObjectsPDF** **MixturePDF**.

ONB

ONB je pomoćna klasa za izgradnju ortonormirane baze. Konstruira se pomoću normale koja postaje vektor "gore" unutar baze.

Klasa je izuzetno korisna za generiranje vektora unutar *PDF* klase.

Implementacija je napravljena po uzoru na članak "*Building an Orthonormal Basis, Revisited*"[3].

Scene

Scene je jedna od najbitnijih klasa u cijelom programu. Sadrži sve objekte koji su stvoren i obavlja rad vezan za provjeru sjecišta zrake i objekta.

Napomena: bitno je pozvati funkciju

```
void ConstructBvh()
```

prije poziva Raytracer funkcije, inače neće se izvršiti izgradnja BVH stabla.

Unatoč tome što sama klasa nije previše komplikirana, ipak je velika te ostavljam detalje implementacije u header i source fileu na repozitoriju.

3.4 Funkcije

Raytracer

Funkcije `void RayTracer(const Viewport& vp, const Scene& scene)` i `RGB TraceRay(const Scene& scene, const Ray& ray, int depth = 0)` su dvije osnovne funkcije potrebne za funkcionalnost.

`RayTracer` funkcija obavlja prolazak kroz piksele slike te generiranje zraka koji izlaze iz kamere.

`TraceRay` je rekurzivna funkcija koja obavlja praćenje zrake kroz scenu i sve interakcije koje zraka može učiniti.

Matematičke funkcije

Implementirano je nekoliko pomoćnih matematičkih funkcija.

```
bool solveQuadratic(const double& a, const double& b, const double& c,
                     double& x0, double& x1)
```

je funkcija koja preko x_0 i x_1 vraća rješenja kvadratne jednadžbe.

Funkcije SchlicksApprox računaju aproksimaciju Fresnellovog koeficijenta refleksije.

Implementiran je i skup funkcija za nasumično generiranje točaka/smjerova:

`RandomCosineDir()` – generira smjer unutar hemisfere prema kosinusovoj distribuciji.

`RandomSpecularDir(double shininess)` – generira nasumični smjer prema spekularnoj distribuciji (implementacija po izvještaju Erica P. Lafourture [4]).

`PointInUnitSphere()` – generira nasumičnu točku unutar jedinične sfere.

`PointInUnitCircle()` – generira nasumičnu točku unutar jedinične kružnice.

`PointInCone(double radius, double sqrDist)` – generira nasumičnu točku unutar stošca kvadratne visine `sqrDist` i radijusa baze `radius`.

4 Za developere

4.1 Stil koda

Ovaj projekt koristi varijantu Microsoft-ovog stila. Za generalniji pogled u stil molim pogledati već postojeći codebase.

Napomena: implementacije klase unutar header fileova bi se trebale izbjegavati.

4.2 Konvencije nazivlja

Klase, funkcije, metode, enums, namespaces su uvijek u *PascalCase*. Dozvoljava se da pomoćne funkcije budu u *camelCase* u slučaju da su "privatne", tj. korištene od drugih funkcija i ne očekuje se da će se pozivati u više funkcija.

- Lokalne varijable i parametri funkcije su **uvijek** *camelCase*.
- Privatne varijable klasa se prefiksiraju sa "`m_`" i započinju sa velikim slovom.
- Statične varijable se prefiksiraju sa "`s_`" i započinju sa velikim slovom.

Primjer stila:

```
namespace ExampleNamespace{
    static int s_StaticInt = 0;

    static void StaticFunction();

    class ExampleClass
    {
        public:
            void MemberFunction(int var)
            {
                m_MemberVar = var;

                std::vector<int> localVector;
                localVector.push_back(var);
            }
        private:
            int m_MemberVariable = 0;
    };
}
```

4.3 Organizacija koda

Sve datoteke moraju biti *camelCase*. Moraju se nalaziti u *./GIrt/src* direktoriju i njihovi pripadajući header fileovi moraju biti pokraj njih u istom direktoriju.

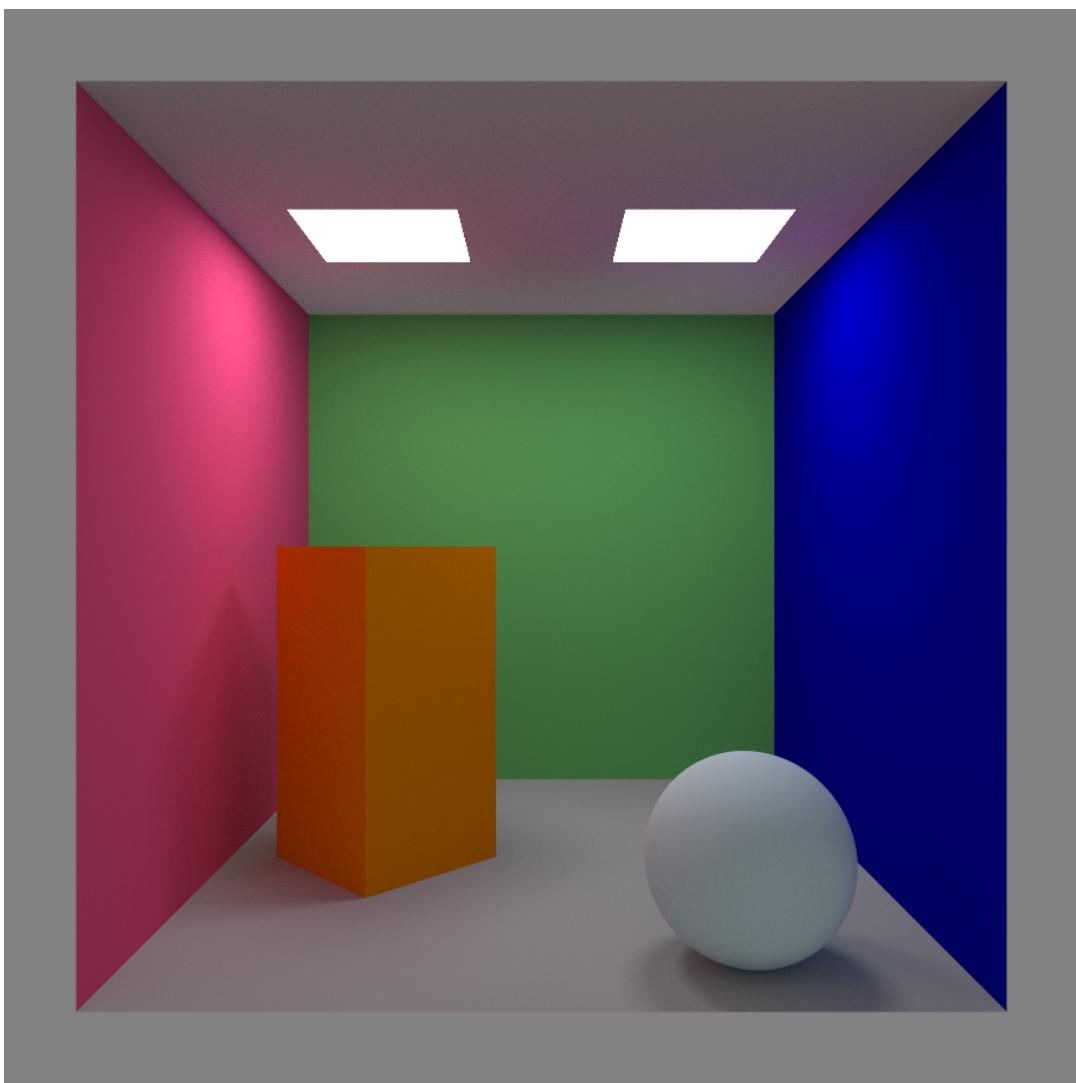
Pomoćne klase, funkcije i drugi objekti trebali bi se nalaziti u *./GIrt/src/util* direktoriju.

Vendor fileovi i druge datoteke o kojima projekt ovisi **moraju** biti u *./GIrt/vendor* direktoriju.

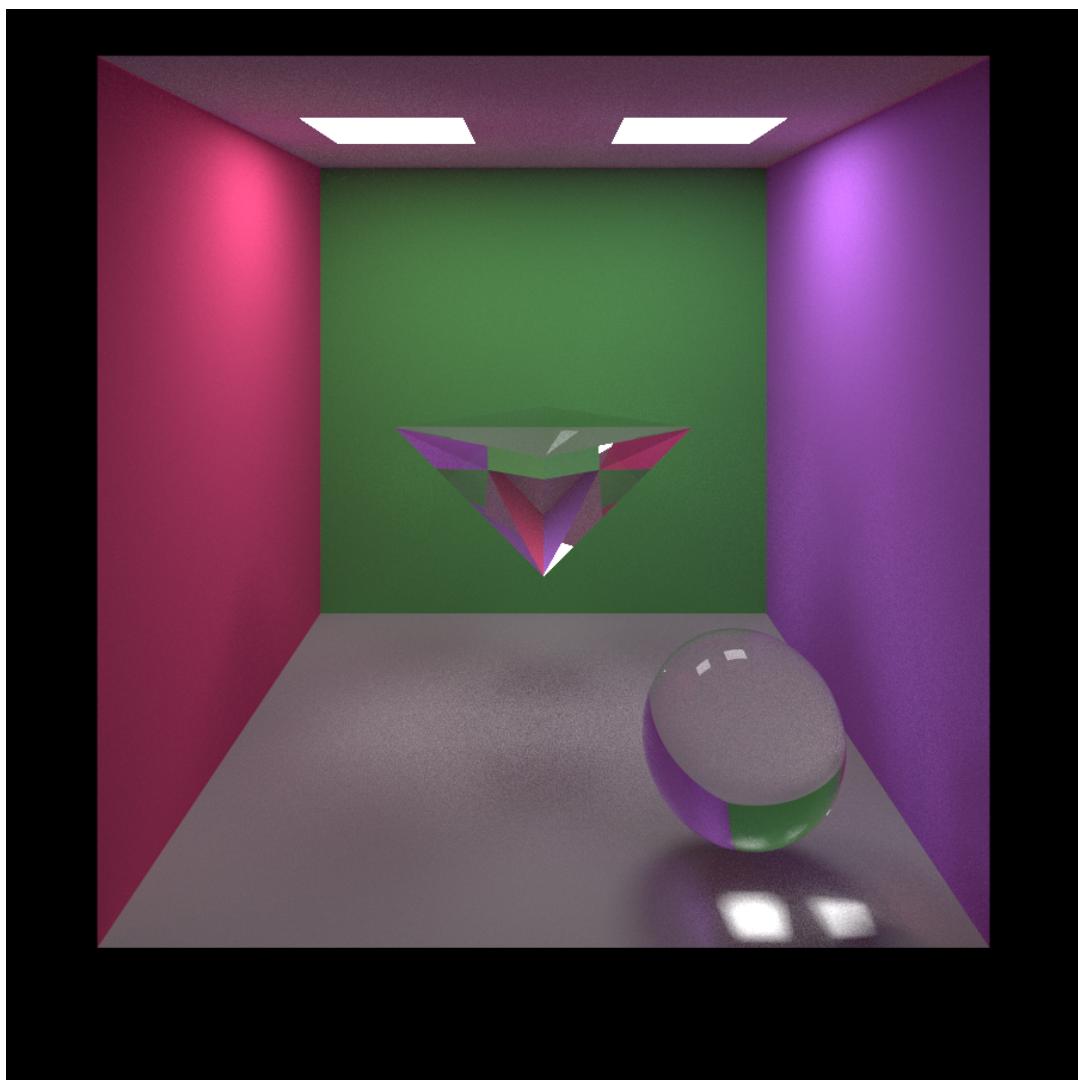
5 Galerija

Sve slike u ovom poglavlju su renderane sa 2000 uzoraka po pikselu, te su 900px široke i visoke.

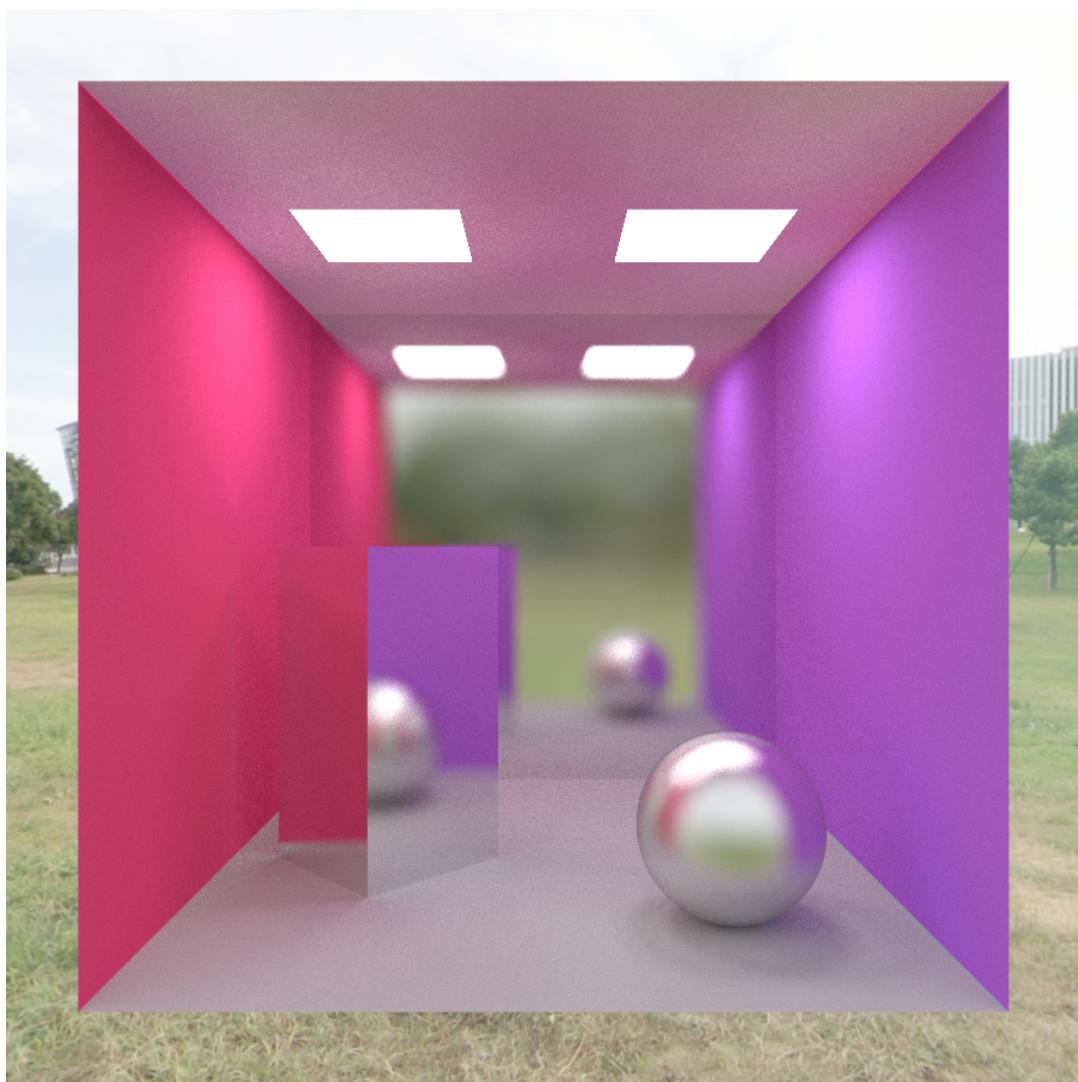
SLIKA 5.1: Primjer difuznih interakcija.



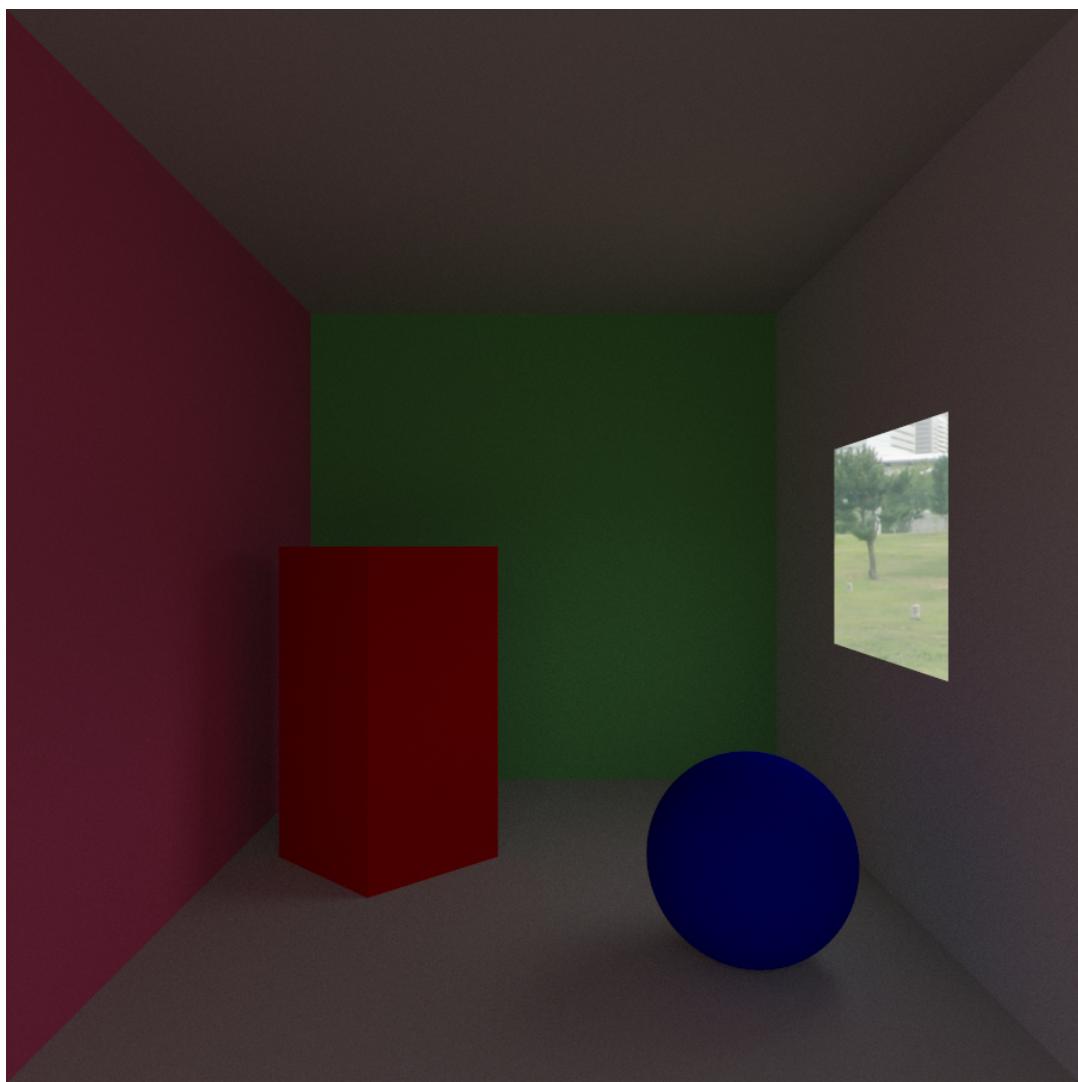
SLIKA 5.2: Primjer prozirnih interakcija.



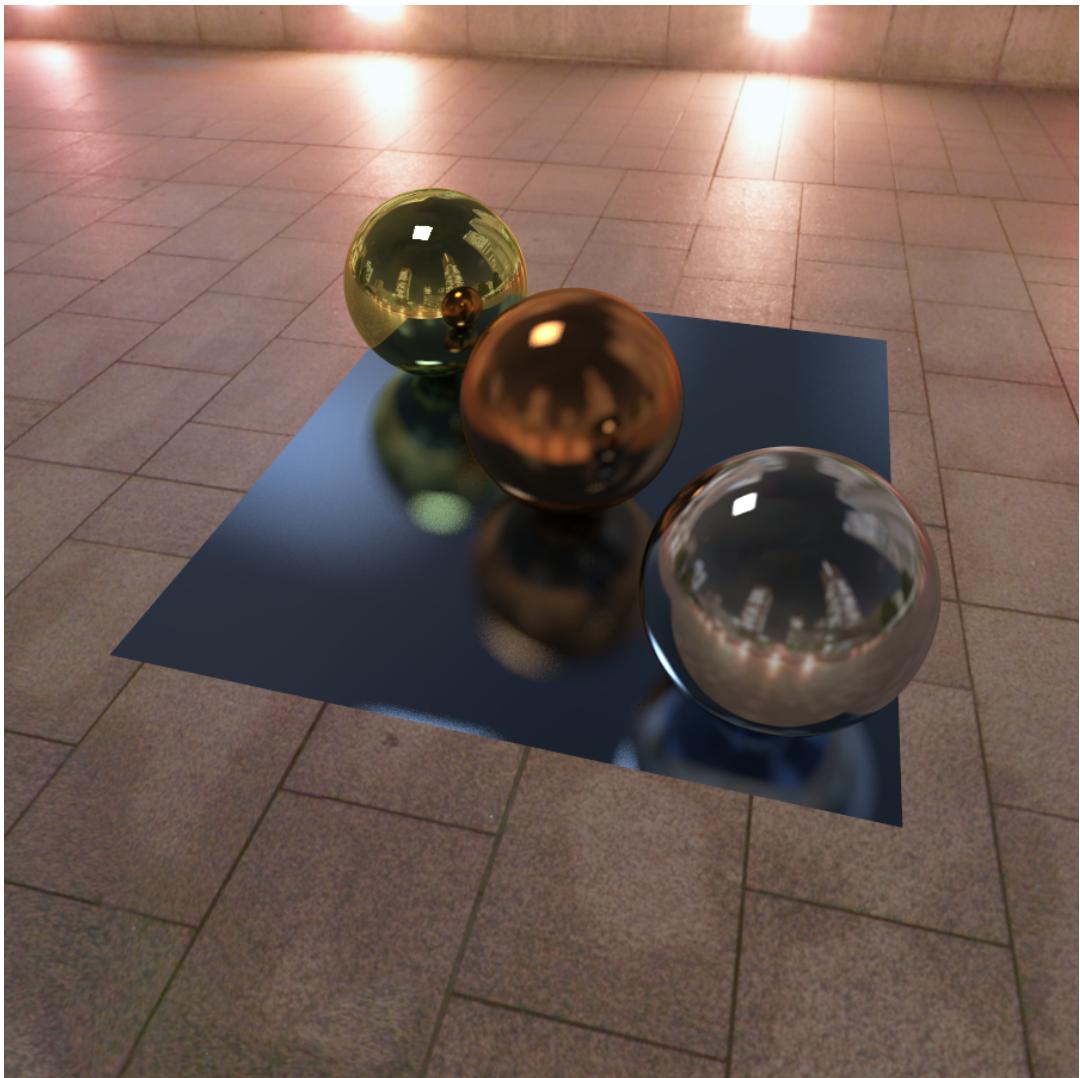
SLIKA 5.3: Primjer spekularnih interakcija sa hrapavostima materijala 0.01 i 0.5.



SLIKA 5.4: Primjer globalne iluminacije pomoću vanjskog svijetla iz cubemap-a.



SLIKA 5.5: Primjer 3 metalne kugle - spekularne refleksije cu-
bemapa.



6 Budući planovi

Ovo je vrlo rudimentarna implementacija raytracer-a sa globalnom iluminacijom. Projekt služi više kao proučavanje teme i motivacija za budući development.

No, postoji nekoliko funkcionalnosti koje bi se idealno implementirale kako bi projekt postao koristan umjetnicima:

Učitavanje modela iz datoteka Trenutna implementacija zahtjeva ručno definiranje mreža trokuta i drugih objekata. Idealno bi bila mogućnost učitavanja .obj datoteka (i pripadnih materijala) pozivom funkcije.

Učitavanje scene iz datoteke Trenutna implementacija nema način uživog učitavanja scene iz datoteke, već zahtjeva ručno mijenjanje *sandbox.h* datoteke i ponovno kompiliranje.

Novi materijali Trenutno su implementirani difuzni, spekularni, prozirni i svijetleći materijal. Idealno bi bilo dodati dodatne materijale kao što su glossy i semi-prozirni materijal.

Podrška za teksture Teksture, HDRIs, normal maps, displacement maps, UV mapiranje, itd. nisu trenutno podržani.

Interaktivni prozor za scene Idealno bi bilo dodati mogućnost micanja objekata za vrijeme korištenja programa te renderanje tako kreirane scene.

Optimizacije Program trenutno se potpuno izvršava na CPU, idealno bi bilo iskoristiti tehnologije kao što su CUDA ili OpenCL da se postignu kraća vremena renderanja.

Potpuni API reference Trenutna dokumentacija nema potpunu API referencu. Sam izvorni kod ima određene dijelove dokumentirane pomoću Doxygen-a, no ne cijeli codebase.

Vlastita implementacija paralelizacije Trenutna implementacija koristi OpenMP za paralelizaciju izvršavanja. Idealno bi bilo implementirati vlastito paralelno izvršavanje radi veće kontrole.

Mnoge druge...

7 Licence

- GLM koristi "The Happy Bunny License", za detalje vidite [ovdje](#)
- stb_image koristi "MIT license", za detalje vidite [ovdje](#)
- Projekt uključuje 2 cubemap-a Emila "Humus" Perssona, koje su licencirane pod CC BY 3.0 licencom za dodatne informacije pogledajte njegovu [webstranicu](#).

Literatura

- [1] Philip Dutre, Philippe Bekaert i Kavita Bala. *Advanced Global Illumination, 2nd Edition*. ISBN: 9781498785624. CRC Press, 2016.
- [2] Andrew Kensler i Peter Shirley. *New simple ray-box test from Andrew Kensler*. 2016. URL: <http://psgraphics.blogspot.com/2016/02/new-simple-ray-box-test-from-andrew.html> (pogledano 15.9.2022.).
- [3] Tom Duff i dr. „Building an Orthonormal Basis, Revisited”. (2017.). jcgt.org. URL: <https://jcgt.org/published/0006/01/01/paper-lowres.pdf>.
- [4] Eric P. Lafortune i Yves D. Willems. „Using the Modified Phong brdf for Physically Based Rendering”. (1994.). Technical Report CW197.
- [5] Peter Shirley, Steve Marschner i Michael Ashikhmin. *Fundamentals of Computer Graphics, 2nd Edition*. ISBN-13: 978-1568812694. A K Peters, 2005.
- [6] Peter Shirley. *Ray Tracing in One Weekend*. 2020. URL: <https://raytracing.github.io/> (pogledano 15.9.2022.).