**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: Програма за фотореалистична графика с трасиране на лъчи

Дипломант: Научен ръководител:

*Дарина Атанасова Александър Соклев*

СОФИЯ

2 0 2 1

**ТЕХНОЛОГИЧНО УЧИЛИЩЕ**

**ЕЛЕКТРОННИ СИСТЕМИ**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

Дата на заданието: 15.11.2020 г. Утвърждавам:..............................

Дата на предаване: 15.02.2021 г. /проф. д-р инж. Т. Василева/

**ЗАДАНИЕ**

**за дипломна работа**

на ученика Дарина Светославова Атанасова от 12в клас

1. Тема: Програма за фотореалистична графика с трасиране на лъчи
2. Изисквания:

- Поддръжка на зареждане на сцена

- Поддръжка на pinhole камера

- Поддръжка на различни видове геометрия - реализацията на триъгълните мрежи ще е с Embree

- Поддръжка на различни видове лампи

- Поддръжка на видове материали

- Поддръжка на anti-aliasing

- Поддръжка на глобално засветяване

1. Съдържание 3.1 Обзор

3.2 Същинска част

3.3 Приложение

Дипломант :...........................................

Ръководител:..........................................

/Александър Соклев/

Директор:................................................

/ доц. д-р инж. Ст. Стефанов

**Мнение на научния ръководител**

Рейтрейсингът е изключително обширна тема. Човек може да напише рейтрейсър толкова прост, че да се събере на визитна картичка, или да посвети живота си на тази тематика и времето няма да му стигне да проучи всичко. Дарина се захвана с тази тежка задача с голяма увереност и интерес. За човек, който няма силен опит в аналитична геометрия, линейна алгебра и 3D визуализация са нужни много усилия, за да вникне в тази сложна материя и да изгради своя първи рейтрейсър. Дарина се справи успешно с това, като се запозна с трудовете на Питър Шърли, един от най-уважаваните изследователи на нашето време, занимаващи се с рейтрейсинг. В допълнение, Дарина проучи и успешно интегрира в проекта си Embree - сложна и популярна библиотека за ускорение на пресичанията на виртуални лъчи със 3D сцена.

По моя преценка постигнатите резултати са забележителни и напълно покриват изискванията на заложената задача. На база на това предлагам дипломантът да бъде допуснат до защита и работата му да бъде оценена високо.

# **Увод**

Намирането на начин за създаване на фотореалистични изображения е било основна цел на компютърната графика в продължение на много години. Под „реалистични“ се имат в предвид такива изображения, които възможно най-много наподобяват реалните снимки. За тази цел те трябва да включат сенки, отражения на светлината и други особености, съществуващи в истинския свят.

Една от първите идеи за алгоритъм, създаващ фотореалистични изображения, дошла от физиката. Когато създавали лещите, физиците проследявали пътя на лъчите от даден източник на светлина до пречупване им след като минат през лещата. Този процес бил наречен трасиране на лъчи. Много разработчици в сферата на компютърната графика смятали, че чрез него лесно ще могат да се създават фотореалистични изображения. [1]

С годините компютрите се усъвършенствали, а заедно с тях и алгоритъма с трасиране на лъчи се подобрявал. Вече можело да се поддържат много оптични ефекти, които да придават на изображенията все по-голяма реалистичност. В днешно време трасирането на лъчи е една от най-популярните и ефективни техники за създаване на реалистични сцени. Използва се в най-различни сфери – от киното до архитектурата.

Целта на текущата дипломна работа е да се създаде програма за фотореалистична графика с трасиране на лъчи, която да може да бъде доразвивана по-нататък. Програмата ще поддържа следните неща:

* Зареждане на сцена
* Pinhole камера
* Различни видове геометрия
* Лампи
* Различни материали
* Anti-aliasing
* Глобално засветяване

# **ПЪРВА ГЛАВА**

# **Въведение в компютърната графика. Обзор на сходни програми с трасиране на лъчи.**

## **Основни термини и понятия**

### **Компютърна графика**

Компютърната графика се занимава със създаването на изображения посредством компютър. Крайният резултат може да е двуизмерна или триизмерна картина, изобразяваща различни неща.

Използва се във филмовата индустрия, игрите, архитектурата, проектирането на обекти и др. Също така компютърната графика има големи ползи в някои не толкова артистични насоки. Например научните визуализации подпомагат по-лесното разбиране от хората чрез компютърни модели и графики. Медицинските изображения са друг добър пример, показващ как компютърната графика улеснява показването на данни. [2]

#### **Пиксел и резолюция**

Пикселът е най-малкият елемент, който изгражда дадено компютърно изображение. Той представлява миниатюрен квадрат с променлив цвят и яркост. Поради малкият размер на отделните пиксели, човешкото око не ги възприема като точки, а като цялостно изображение.

Изображенията се състоят от множество пиксели, подредени в правоъгълна решетка с определена разделителна способност. Това представлява резолюцията – броят на пикселите в едно компютърно изображение. Тя може да бъде представена като произведението на две числа – широчината по височината на картината. Следователно, колкото е по-голяма резолюцията, толкова е по-високо качеството на изображението. [2]

#### **Растерна графика**

Растерните изображения са съставени от множество пиксели. Те са толкова малки, че не се виждат с просто око, но заедно съставят цифровото изображение.

Качеството на растерните изображения зависи от разделителната способност и размерите им. Те не могат да бъдат мащабирани извън зададените параметри без да се влоши качеството. Причината е, че когато размерът се увеличава, пикселите не могат да се възпроизведат повече и затова те просто се уголемяват и заприличват на квадратчета - т.нар. пикселизиране. [2]

#### **Векторна графика**

Съществува алтернативен метод за решаване на проблема с пиксализирането в компютърната графика. Вместо изображението да се създава от пиксели, във векторната графика формите се задават чрез аналитични формули, които са безкрайни като точност. По този начин, когато изображението се мащабира, всичко се преизчислява и качеството не се губи. [2]

### **Моделиране**

Моделирането е процесът на изработка на математическа мрежа, представляваща триизмерен обект, с помощта на специализиран софтуер. Продуктът се нарича 3D модел е се формира от точки, наречени върхове, които определят структурата и формират полигони. Най-често моделите имат формата на многостени – фигура, чиито свойства улесняват следващия процес на рендериране. [3]

### **Рендериране**

Рендерирането представлява генериране на цифрово изображение (визуализация) от модел в компютърна графика. То има за задача да пресъздаде тримерен математически обект върху плоска двумерна повърхност (екрана) посредством различни алгоритми. Те се използват за постигане на перспектива в изображенията, за оцветяване, прозрачност, определяне на видимите обекти, пресмятане на сенките и т.н. [3]

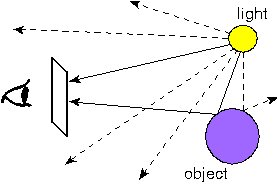
*Фиг. 1.1. Разлика между моделиране и рендериране*

### **Трасиране на лъчи**

В реалния свят светлината е съставена от фотони с различна енергия. Тези фотони се придвижват под формата на потоци в прави линии, наречени лъчи. Когато даден фотон срещне окото ни, енергията му се прехвърля в рецепторите на ретината ни. След това тя възприема дадения цвят в зависимост от големината на енергията на фотона. [1]

#### **Трасиране на лъчи в права посока**

Техниката с трасиране на лъчи в права посока много добре имитира начинът, по който светлината се движи в реалния свят – тръгва от даден източник на светлина и се разпространява в безброй много посоки. Точно по тази причина, ако гледаме сцената през камера, по-голямата част от лъчите няма да достигнат окото ни. Ще можем да видим единствено тези, които се отразят по такъв начин, че да се ударят в екрана.



*Фиг. 1.2. Трасиране на лъчи*

#### **Трасиране на лъчи в обратна посока**

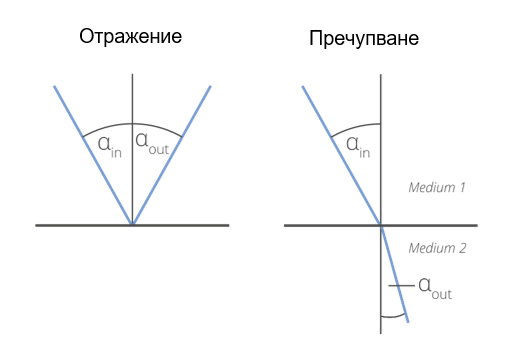
Трасирането на лъчи в права посока е прекалено неефективно, за да се имплементира на компютър, тъй като прекалено малък процент от лъчите достигат до екрана. За щастие може да се проследи обратния път на лъча – от камерата до източника на светлина. По този начин се знае в какви предмети се е отразил преди да стигне до нас и може да се сметне какъв е цвета му. С използването на този вид трасиране ще ни е необходимо много по-малко време за смятане на сцената от *фиг. 1.2.*,тъй като ще трябва да се сметнат само два лъча.

### **Видове лъчи**

За да се сметне цвета на даден пиксел, трябва да се вземат в предвид всички лъчи, които по някакъв начин му влияят. Съответно в случая тези лъчи могат да се разделят на няколко вида: светлинни, сенчести, отразени и пречупени лъчи. Математически всички те са просто лъчи, но при изчисленията е много по-удобно да се разглеждат поотделно. [1]

#### **Отразени и пречупени лъчи**

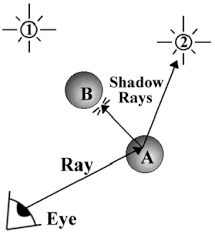
Светлината променя посоката си в зависимост от повърхностите, с които се сблъска. Когато даден лъч пресече точка, намираща се на непрозрачна повърхност, това създава нов отразен от нея лъч. Ако повърхността е прозрачна (напр. стъкло), лъчът преминава през нея и се пречупва.



*Фиг. 1.3. Отражение и пречупване*

#### **Светлинни и сенчести лъчи**

Когато лъч се пресече с точка от повърхността на предмет, е необходимо да се разбере дали тази точка попада в сянка или не. За целта от нея се пускат т.нар. сенчести лъчи, последователно към всички източници на светлина. Ако даден предмет, непропускащ светлина, е между тях, тогава съответната точка се намира в сянка спрямо този източник на светлина. В случай, че нищо не пресича пътя на лъча, светлината достига директно до точката и лъча се превръща в светлинен лъч. Ако вземем за пример *фиг. 1.4.*, там обект А е в сянка спрямо светлинен източник 1, но не е в сянка спрямо светлинен източник 2.



*Фиг. 1.4. Светлинни и сенчести лъчи*

## **Текстови редактори. Интегрирани среди за програмиране.**

В днешно време се използват различни текстови редактори или интегрирани среди за програмиране, в зависимост от личните предпочитания на отделните програмисти.

Текстовите редактори се използват за създаване и редактиране на програмния код. Някои редактори поддържат функции, улесняващи програмирането: дописване на служебни думи, оцветяване на кода за по-лесно разграничаване на функции, променливи и др., автоматично форматиране и т.н.

Интегрираните среди за програмиране (IDE) обединяват отделните инструменти за разработване на приложение в едно. Най-често осигуряват текстов редактор, компилатор (транслатор или интерпретатор) и дебъгер. IDE-тата се използват все по-широко в практиката, тъй като улесняват работата и увеличават производителността. [5][6]

### **Microsoft Visual Studio**

Microsoft Visual Studio е интегрирана среда за разработка на софтуерни приложения. Предоставя интегрирана среда за писане на код, компилиране, изпълнение, дебъгване, изпълнение на тестове и др. Също така потребителят може да персонализира темата на редактора, големината на текста и клавишните комбинации.

Visual Studio е достъпен за Windows и macOS. Има безплатна community версия, която може да бъде изтеглена от всеки. Поддържа голяма част от най-популярните езици за програмиране и има възможност за добавяне на различни плъгини.

### **Eclipse**

Eclipse, също както и Visual Studio, е интегрирана среда за програмиране. Може да се използва за програмиране на С++, като се изтегли специално разширение за езика. Работи на различни операционни системи – Windows, Linux и macOS.

Има включен дебъгер, компилатор и може да автоматично да довършва думи от кода и да проверява синтаксиса за грешки. Също така Eclipse включва и drag-and-drop функционалност, дистанционно управление на проекти и готови шаблони.

### **Sublime Text**

Sublime Text е текстов редактор за различни платформи, написан на Python. Поддържа функционалности като едновременно редактиране на кода с повече от един курсор, автоматично довършване на думите, компилиране в редактора и др. Потребителите могат да добавят разширения, които най-често са с лиценз за свободен софтуер.

### **Notepad++**

Notepad++ е текстов редактор за Windows и се разпространява като свободен софтуер. Поддържа редактиране в раздели, което позволява работа с много отворени файлове в един прозорец. Също така предлага и ограничено довършване на думи, което означава, че не проверява за грешки при синтаксиса. Въпреки ограничените възможности, Notepad++ все още е един от най-използваните редактори в днешно време.

## **Съществуващи подобни програми**

### **Chaos Vantage**

Chaos Vаntage е най-новата разработка на Chaos Group, пусната на пазара през 2020г. Тя може да генерира огромни фотореалистични сцени в реално време. Въпреки, че потенциално може да се използва за визуални ефекти и анимации, продуктът е създаден най-вече за архитектурни визуализации. Освен, че потребителите могат да се движат из сцената посредством клавиши, могат и да правят прости редакции във Vantage, като местене, триене и копиране на обекти.

Chaos Vantage има пряка връзка с 3ds Max, което означава, че промените в Max ще се отразяват на тези във Vаntage в реално време. Все пак това не означава, че двете сцени ще са напълно идентични, тъй като Chaos Vаntage все още не поддържа всички особености на .vrscene формата. [6][7]

### **RenderMan**

RenderMan е създадено от Pixar и се използва за рендарирането на всички техни филми. Има много различни функционалности, които са съществено важни за обработката на комплексните филмови сцени. Pixar използват водещото в индустрията Pixar Marschner Hair за създаване на възможно най-реалистична козина и коса. RenderMan е първият софтуерен продукт, награден с Оскар. [6][8]

### **Autodesk Maya**

Maya е приложение за триизмерна компютърна графика, първоначално разработено от Alias Systems Corporation, а понастоящем - собственост на Autodesk. Използва се във филмите, телевизията, разработката на игри и архитектурата.

Потребителите имат виртуална сцена, в която могат да имплементират и редактират различни обекти. Проектите могат да бъдат запазвани в различни формати.

Ученици и учители могат да свалят пълната образователна версия безплатно. Версиите свободни за достъп имат лиценз само за некомерсиална употреба и идват с пълен 36-месечен лиценз. [6]

# **ВТОРА ГЛАВА**

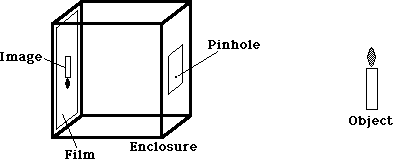
# **Методи и алгоритми за реализиране на програмата с трасиране на лъчи**

## **Функционални изисквания**

### **Pinhole камера**

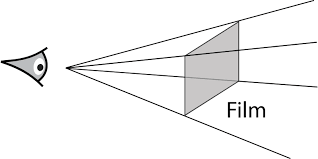
Има поддръжка на pinhole камера, която може да се настройва накъде да гледа и под какъв ъгъл да бъде завъртяна.

Pinhole камерата е една от най-простите и най-разпространени камери в компютърната графика (*фиг. 2.1.*). Базира се на камерата обскура. Тя представлява тъмна кутия, в която не прониква светлина. Кутията има малък отвор на една от стените и екран – матово стъкло или бяла хартия, на срещуположната страна. От отвора към „екрана“ се получава леко размито, увеличено и преобърнато изображение. Именно, на този принцип работели и някои от първите фотоапарати. [9]



*Фиг. 2.1. Модел на камера обскура*

Въпреки, че има и по-сложни камери, pinhole камерата се използва заради голямото си приложение и простотата си. За удобство в програмирането и графиката, класическата версия е леко променена - екранът на филма е изнесен пред отвора, който вече играе ролята на окото. На *фиг. 2.2.* може да се види прост модел на камерата в програмата с трасиране на лъчи.

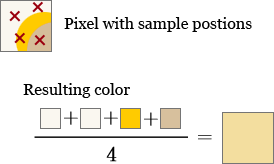


*Фиг. 2.2. Модел на pinhole камера*

### **Anti-aliasing**

Голям проблем при създаването на компютърно-генерирано изображение е това, че мониторът не може да показва продължителен аналогов сигнал. През очите ни или през камера картините се виждат точно, всяка линия и предмет е ясно изобразен. Когато се използва компютър за същите изображения, е невъзможно да се пресъздадат абсолютно същите им копия. Това е поради фактът, че компютрите имат ограничени пиксели, с които да представят аналоговия сигнал. Дори с използването на монитори с по-висока резолюция, ефектите от aliasing-a няма как да не се появят. Затова има измислени техники, наречени anti-aliasing, с които да се предотврати и намали проблема. [1]

В програмата е използван един от най-лесните начини за справяне с ефектите от aliasing-a – пускане на голям брой лъчи през един пиксел. Този метод се нарича supersampling. При него цветът на един пиксел се смята като средно аритметично от цветовете на всички лъчи, които са минали през него (*фиг. 2.3*).



*Фиг. 2.3. Supersampling*

### **Видове материали**

Програмата симулира това как светлината си взаимодейства с различните материали. Чрез проследяването на лъчите по начин, сходен на този в реалния свят, се придава фотореалистичност на изображението.

Различните материали (дифузни, стъкло, метал и др.) поглъщат, отразяват или пречупват светлината по различни начини. Когато лъч се удари в дадена повърхност, той променя посоката и цвета си в зависимост от нея.

Дифузните материали могат да погълнат светлината вместо да я отразят. Колкото са по-тъмни, толкова е по-голям шансът това да се случи. Има и други материали като стъклото и водата, които не могат да погълнат светлината. Те правят точно обратното - когато някой лъч се удари в тях, той се разделя на пречупен и отразен лъч.

### **Зареждане на сцена**

Зареждането на сцена представлява отварянето на 3D модел, който е в obj формат. Всеки такъв файл е мрежа от даден брой триъгълници, които после могат да бъдат пресечени.

### **Видове геометрия**

* + - 1. **Сфера и куб**

Поддържа се пресичането на лъчи със сфера и куб. Пресмята се по какъв начин ще се отрази лъчът след като се удари в съответния вид геометрия. Има функция за всички обекти, които могат да бъдат пресечени. Тя се пренаписва в зависимост от геометрията.

* + - 1. **Триъгълни мрежи**

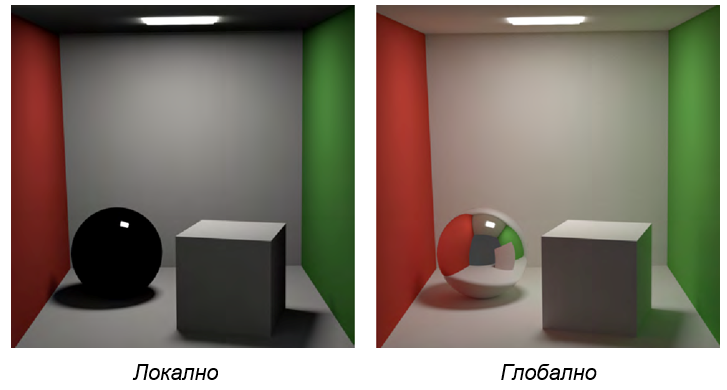
Чрез използването на триъгълни мрежи лъчите могат да бъдат пресечени с какъвто и да е обект. Те са съставени от много на брой триъгълници. Поради по-малкото операции, които трябва да се извършат, програмата става много по-бърза и ефективна. Използването на триъгълни мрежи позволява програмата да работи с 3D модели (obj файлове). [10]

### **Генериране на изображение**

След пускането на програмата, тя пресмята е генерира изображение, което се записва в ppm формат. Това е възможно най-базовият файлов формат за изображения.

### **Глобално засветяване**

Под глобално засветяване се има в предвид идеята, че обектите не отразяват светлина единствено към камерата, а и към останалите обекти в сцената. Това се постига по подобен начин на anti-aliasing-a, като се изпратят много на брой лъчи през даден пиксел. На *фиг. 2.4.* е показана разликата между локално и глобално засветяване. [11]

*Фиг. 2.4. Разлика между локално и глобално засветяване*

### **Дифузно осветление**

Дифузното осветление разпръсква светлината във всички посоки. По този начин сенките стават много по-меки, а очертанията не са толкова резки, както при точковата лампа.

### **Рекурсивен алгоритъм**

Цветът на даден пиксел е комбинация от различни лъчи светлина (отразени, пречупени и др.). За да се пресметне техния цвят, трябва да се проследи обекта, от който са дошли съответните лъчи. Но в действителност, цветът, който се е отразил от този обект, отново е комбинация от предишните лъчи. Това наблюдение позволява алгоритъмът с трасиране на лъчи да е рекурсивен. Въпросът обаче е кога приключва програмата. Единият вариант е даден лъч да спре да се проследява, когато не се пресече с нищо в сцената. Тогава може да му се сложи предварително дефиниран цвят като на фона. Друг вариант е проследяването на лъча да спре, когато приносът му към финалния цвят стане прекалено малък. [1]

## **Избор на технологии и среда за разработка**

### **C++**

В една програма с трасиране на лъчи се извършват изключително много операции и пресмятания. Затова е важно езикът, който се използва да възможно най-бърз. С++ е един от най-бързите езици за програмиране, което го прави много подходящ за целта.

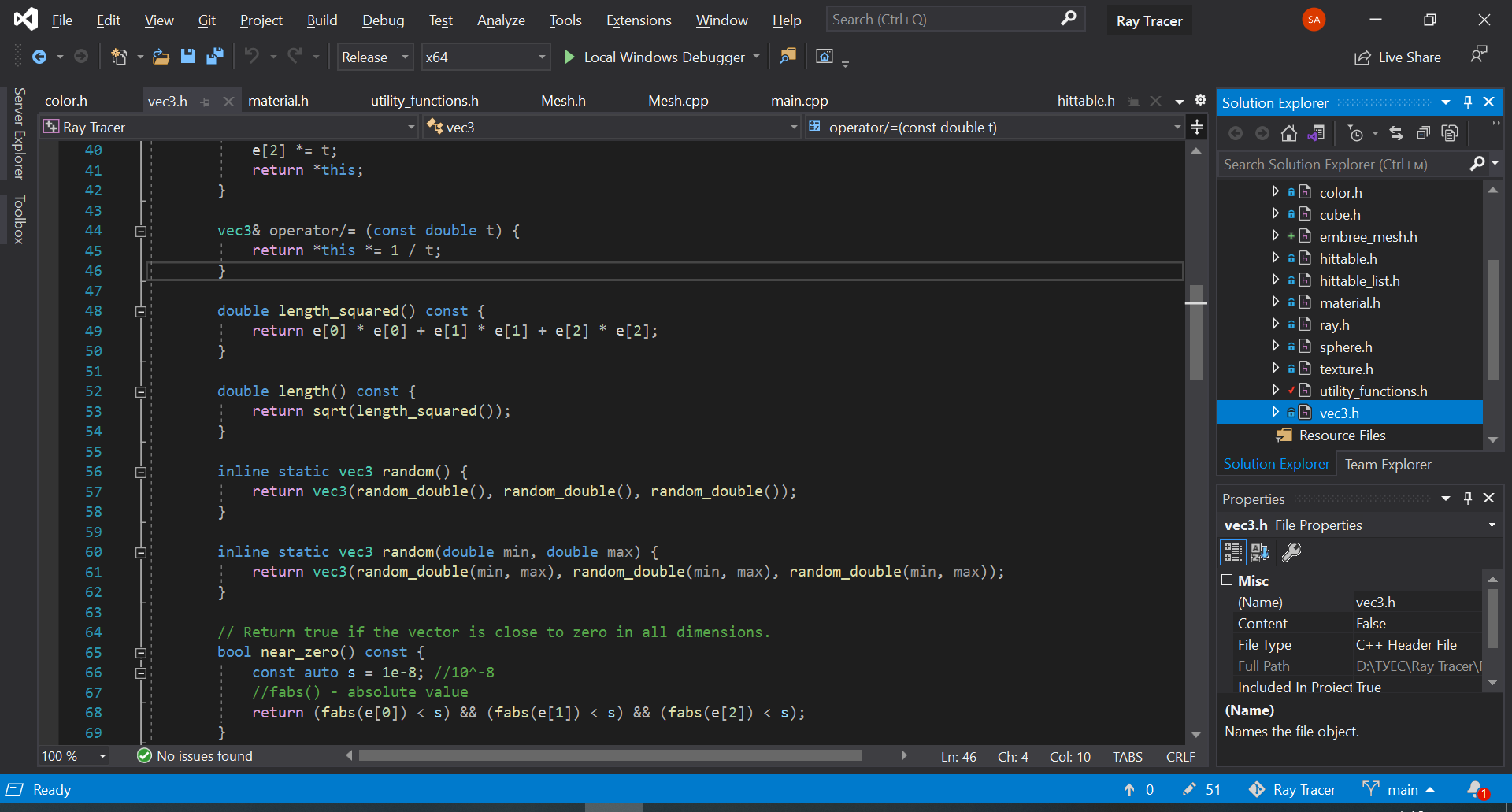
Също така, С++ е обектно-ориентиран език, което улеснява организацията на отделните класове.

### **Embree**

Embree е библиотека за трасиране на лъчи, изработена от Intel, която може да се интегрира в различни програми за създаване на фотореалистични изображения. Съставена е от алгоритми, които позволяват пресичането на лъчи с дадени геометрии. С помощта му скоростта на програмата се увеличава изключително много. Embree е една от най-подходящите библиотеки за програми с трасиране на лъчи, които работят на процесора, а не на видеокартата. [10][12]

### **Visual Studio**

Visual Studio има вграден компилатор и дебъгер, което много улеснява работата с него.



*Фиг. 2.5. Екран във Visual Studio*

# **ТРЕТА ГЛАВА**

# **Реализация на програмата за фотореалистична графика с трасиране на лъчи**

## **Векторен клас**

Векторният клас е най-базовият клас, тъй като след това всичко се основава на пресмятания посредством вектори. В него има функции за различни операции, които могат да се извършват с вектори.

Има един единствен атрибут, който представлява масив от 3 десетични дробни числа. Това са координатите по x, y и z в триизмерното пространство. [11]

### **Важни операции с вектори**

* + - 1. **Дължина на вектор**

Дължината на вектор в триизмерното пространство се намира по следната формула.

*Формула 3.1. Дължина на вектор*

* + - 1. **Скаларно произведение**

Скаларното произведение представлява произведението на дължинита на два вектора с косинуса на ъгъла между тях, като може да се пресметне и чрез координатите на векторите. [14]

*Формула 3.2. Скаларно произведение на два вектора*

* + - 1. **Векторно произведение**

Векторното произведение на два вектора и е трети вектор, който е перпендикулярен на равнината, в която лежат първите два вектора. Големината му се намира по следната формула:

*Формула 3.3. Големина на вектор получен при векторно произведение*

Координатите на вектора, получен при векторно произведение, могат да се пресметнат по следната формула:

*Формула 3.4. Координати на вектор получен при векторно произведение*

### **Точка**

Триизмерната точка се състои от три координата, също както и вектора. Затова класа за вектор, заедно с функциите му, се преизползва. Това позволява извършването на операции с точки без да се налага да има нов клас за тях.

## **Цвят**

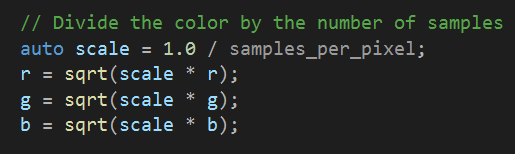
Цветовете също като точките са съставени от три числа, което отново позволява преизползването на класа за вектор. Всеки цвят може да бъде представен чрез прибавянето на трите основни цвята – червено, зелено и синьо, с различна наситеност. Така се използва операцията за сбор на вектори, която е еднаква и за цветовете. При дефинирането на даден цвят стойностите му трябва да са между 0 и 1.

### **Гама корекция**

Очите ни не възприемат светлината по начина, по който камерите. Когато двойно повече фотони достигнат до камерата, тя получава двоен сигнал, т.е. снимката изглежда двойно по-ярка (линейна връзка). Очите ни от друга страна не работят по този начин. Вместо това, те възприемат двойно повечето светлина като съвсем малко по-светло (нелинейна връзка). За сметка на това, очите ни са много по-чувствителни в тъмното. Има биологична причина за тази особеност - това позволява на зрението ни да работи с много по-голям диапазон от яркости. В противен случай спектъра от цветове и яркости, с които се сблъсквахме навън щеше да е прекалено натоварващ.

Гама корекцията е нещото, което приспособява нашето усещане за яркост и това на камерата. Например, когато някое изображение бъде запазено, то се “гама дешифрира”, за да може удвоената стойност във файла възможно най-близо да представлява това, което ние бихме възприели като двойно по-ярко. [15]

В програмата се използва гама 2, което най-просто казано означава цветът повдигнат на степен 1/гама. В случая това е цветън под корен квадратен. На *фиг. 3.1.* е показана гама корекцията, където samples\_per\_pixel е броят на лъчите, които се пускат през един пиксел. Това е необходимо, за да може да се пресметне цвета.



*Фиг. 3.1. Гама корекция*

## **Лъч**

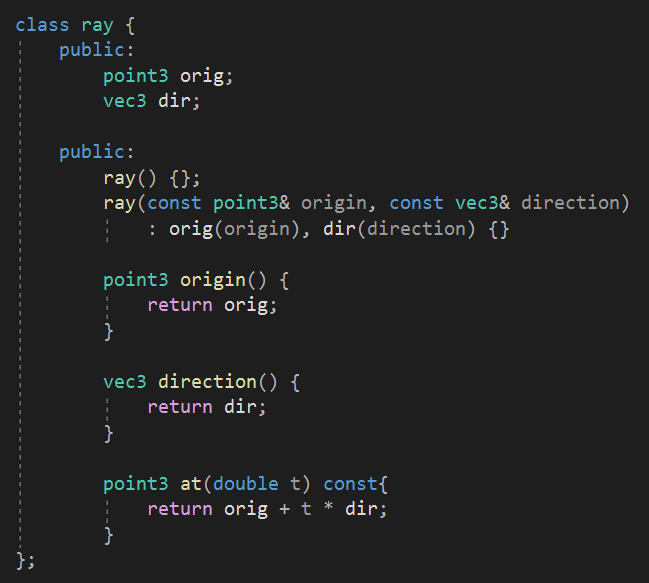
Лъчът има следните два атрибути:

* Orig – точката, от която започва лъчът;
* Dir – вектор, показващ посоката на лъча.

Функцията *at* показва позицията на лъча върху дадена линия в триизмерното пространство. В зависимост от променливата *t*, която се подава на функцията, тази позиция се променя. Във *формула 3.5.* А е началото на лъча, а b е посоката му.

*Формула 3.5. Лъч*

На *фиг. 3.2.* е показан класът за лъч.

****

*Фиг. 3.2. Клас за лъч*

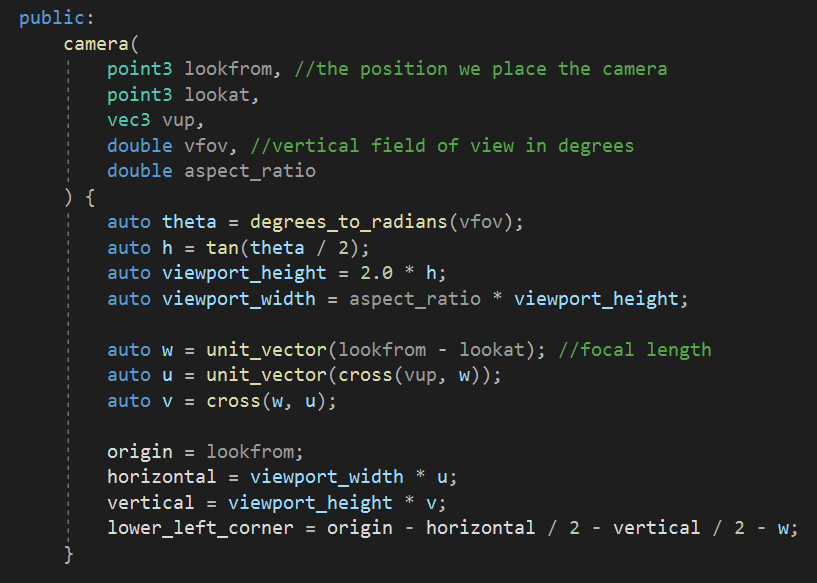
## **Pinhole камера**

За осъществяването на pinhole камерата има отделен клас camera, който съдържа следните атрибути:

* Origin – точката, в която е поставена камерата;
* Lower\_left\_corner – филмът, който е сложен пред камерата (*фиг. 2.2.*);
* Horizontal – широчина на камерата;
* Vertical – височина на камерата.

На *фиг.3.3.* е показан конструкторът на камерата. Аргументите, които му се подават са:

* Lookfrom – точката, в която е поставена камерата;
* Lookat – точката, към която гледа камерата;
* Vup – вектор, определящ ориентацията на камерата, който показва накъде е „нагоре“. Той е перпендикулярен на вектора, показващ „напред“ (той зависи от това накъде гледа камерата).
* Vfov – вертикално зрително поле; ъгълът, който показва каква част от сцената се вижда;
* Aspect\_ratio – отношението на широчината към височината.



*Фиг. 3.3. Конструктор на камерата*

## **Рендериране на изображението**

Рендерирането на изображението се случва чрез 3 вложени цикъла. Първият и вторият се въртят съответно по височината и широчината на изображението. Третият цикъл се върти докато не стигне samples\_per\_pixel. Това е число, зададено по-горе, което определя колко лъчи да се пуснат през даден пиксел. По този начин цвета на пикселите по очертанията на обектите се смесва от фона и самия обект и границите стават по-плавни. Така се постига anti-aliasing.

****

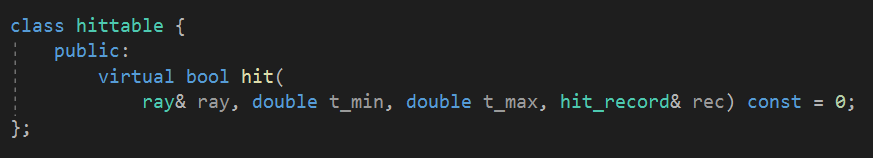
*Фиг. 3.4. Рендериране на изображение*

В този цикъл се вика функцията ray\_color, която пресмята цвета на всеки лъч. Този цвят се прибавя към pixel\_color, като по този начин се пресмятат цветовете на всеки един от пикселите. Тази информация се записва в ppm файл, който представлява самото изображение.

## **Пресичане с различни видове геометрия**

### **Абстрактен клас за геометрия**

Класът hittable е абстрактен клас за всички геометрии, които могат да се пресекат с дадени лъчи. Той има една единствена функция hit, която показва какво се е случило с лъча след пресичането му със съответната геометрия.

****

*Фиг. 3.5. Абстрактен клас hittable*

Един от аргументите, които приема функцията е от тип hit\_record. Това е структура, съдържаща най-важната информация относно точката на пресичане на лъча с геометрията. Структурата има следните 5 атрибута:

* p – точката на пресичане;
* Normal – нормалата;
* Mat\_ptr – указател за вида на материала;
* t – число, показващо разстоянието;
* front\_face – булева стойност за това накъде е насочена нормалата.

### **Пресичане със сфера**

Сферата има три атрибута:

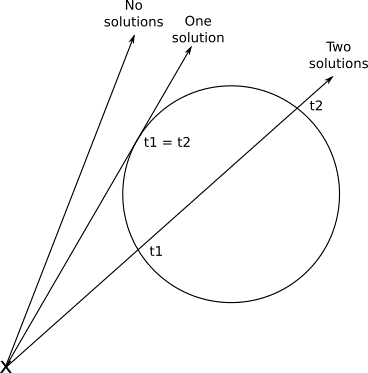
* Център
* Радиус
* Указател за вида на материала

Във *формула 3.6.* е показано общото уравнение за сфера, където е центърът на сферата, а R е радиусът ѝ. Съответно точките с координати (x, y, z) лежат върху повърхността на сферата.

*Формула 3.6. Общо уравнение за сфера*

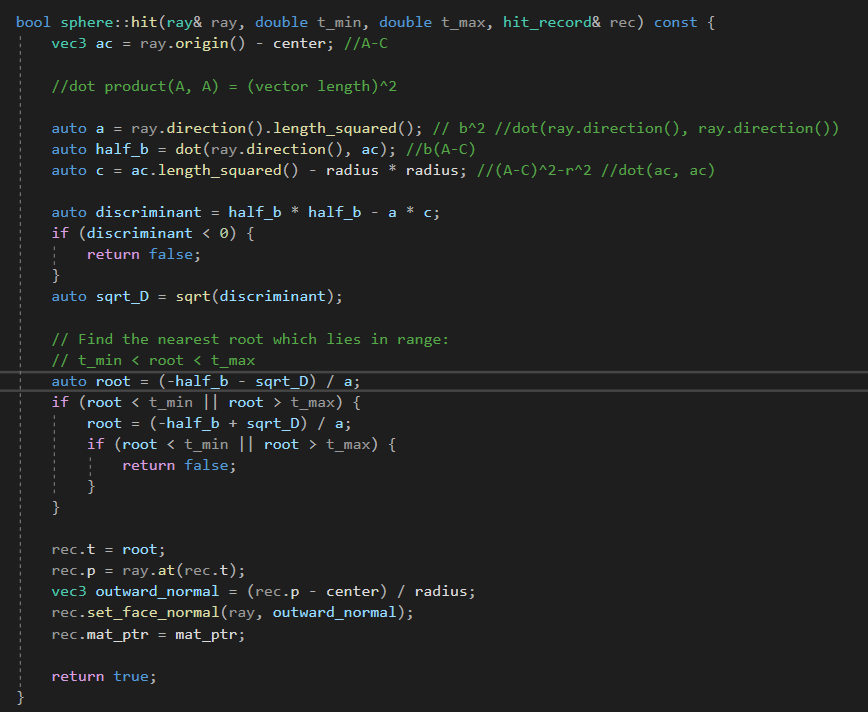
След като в уравнението се замести от формулата за лъч (*формула 3.5.*) и се опрости, се получава ново уравнение:

Единственото неизвестно в него е t и следователно се получава квадратно уравнение. В зависимост от броя на решенията му, се разбира къде се пресича лъчът със сферата. На *фиг. 3.6.* са описани трите възможни случая.



*Фиг. 3.6. Случаи на пресичане на лъч със сфера*

* Уравнението да няма решение – лъчът не се пресича със сферата;
* Уравнението да има 1 решение – точката на пресичане се намира по повърхността на сферата;
* Уравнението да има 2 решения – лъчът пресича сферата на две места, като се взима по-близкото

**

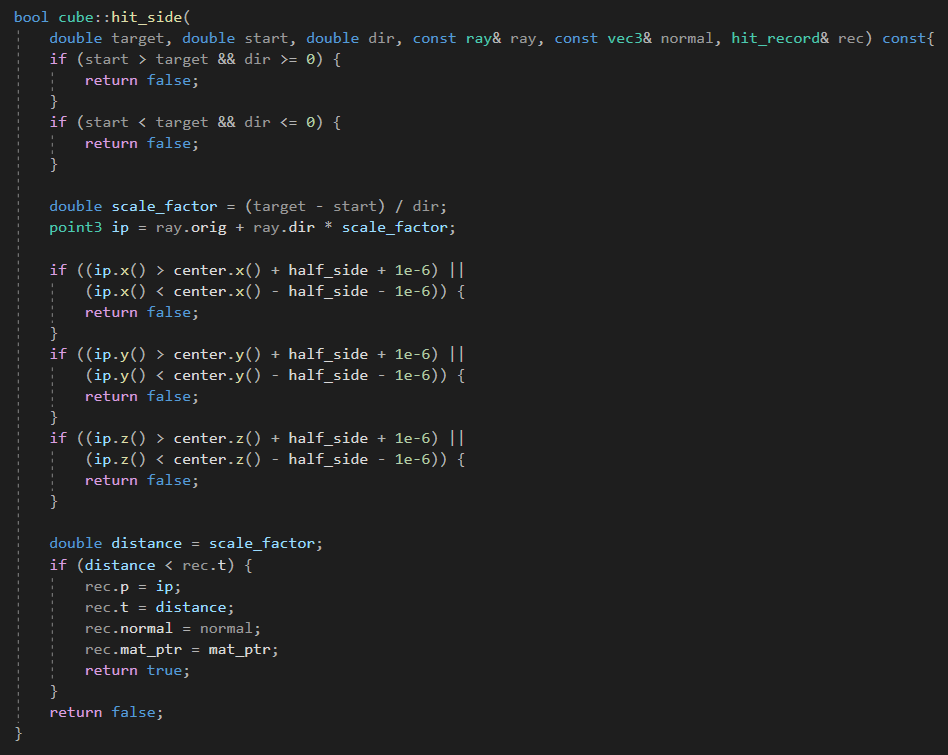
*Фиг. 3.7. Реализация на пресичането със сфера*

### **Пресичане с куб**

Кубът има 3 атрибута:

* Център
* Големината на половина страна – използва се половината страна, защото пресмятанията стават по-бързи и лесни;
* Указател за вида на материала

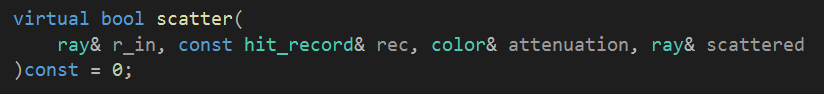
В класа за куб има функция hit\_side, която проверява дали лъчът се е пресякъл с една страна. Ако се е пресякъл, тогава структурата от тип hit\_record се попълва с данните за точката на пресичане. След това във функцията hit се извиква hit\_side за всяка една от шестте страни на куба. На *фиг. 3.8.* е показана реализацията на hit\_side.



*Фиг. 3.8. Реализация на hit\_side*

## **Материали**

Класът за материали е абстрактен клас, който се наследява от всички видове материали. Функцията scatter описва по какъв начин се отразява или пречупва светлината след пресичането с дадения материал.



*Фиг. 3.9. Абстрактна функция scatter*

Функцията приема следните аргументи:

* R\_in – лъчът, който ще се пречупи;
* Rec – структура от тип hit\_record, съдържаща най-важната информация за пресечната точка;
* Attenuation – цвета след пресичането на лъча;
* Scattered – отразеният лъч.

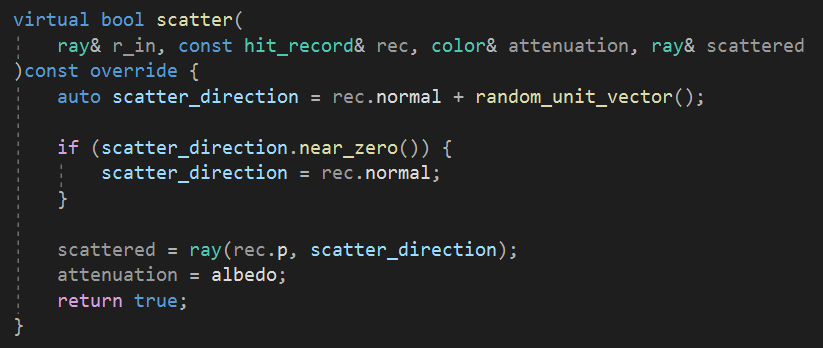
### **Дифузни материали**

Дифузните (матови) материали не излъчват светлина, а взимат цветовете от заобикалящата ги среда и ги променят спрямо собствения си цвят. Идеалното дифузно отражение са нарича Ламбертово отражение, което означава, че откъдето и да се погледна дадения обект, има еднаква яркост.

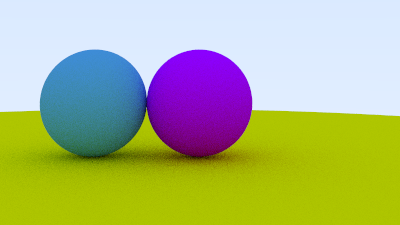
Лъчите, които се отразяват от такива повърхности, имат случайна посока. Това се случва като към нормалата в точката на пресичане се добави вектор със случайни стойности. След, което отразеният лъч scattered започва от точката на пресичане и е с посока равна на сбора на нормалата и случайния вектор.

Количеството погълната и отразена светлина зависи от цвета на повърхността – albedo. Колкото е по-тъмен той, толкова повече светлина ще бъде погълната.

На *фиг. 3.10.* е показана реализацията на пресичането с дифузни материали.



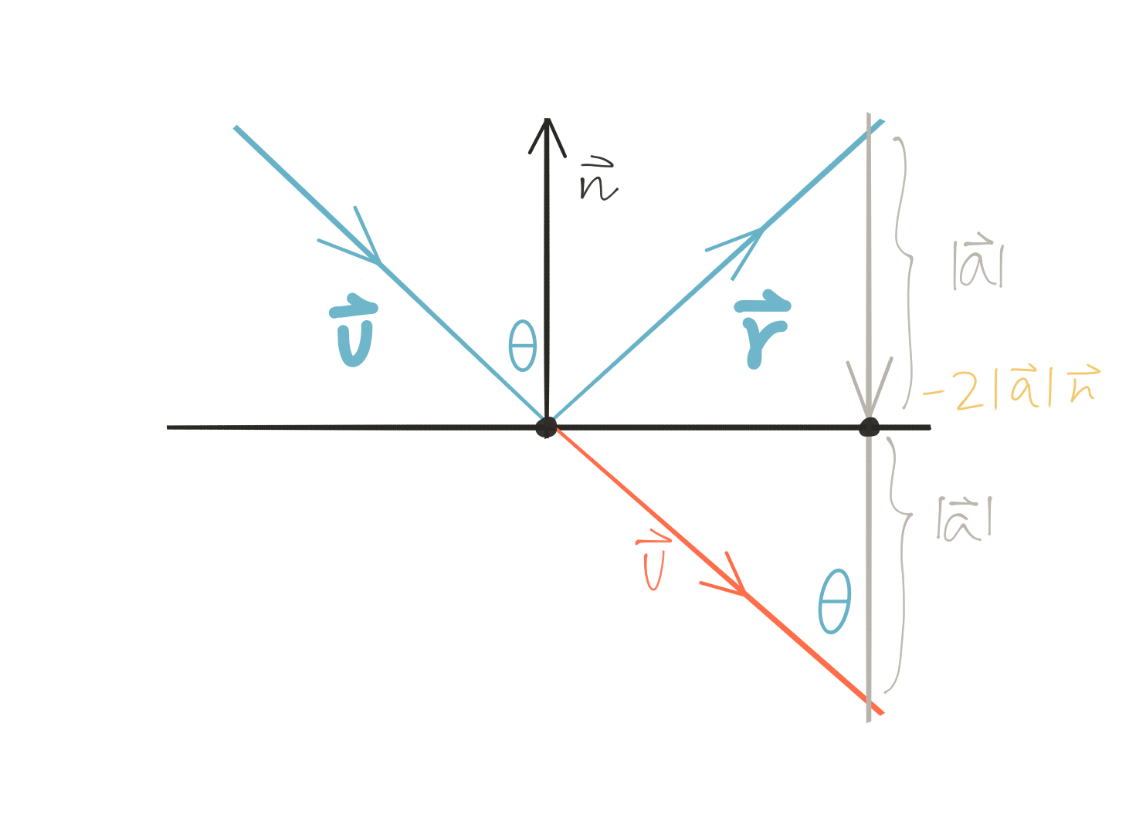
*Фиг. 3.10. Функцията scatter за дифузни материали*

**

*Фиг. 3.11. Дифузни сфери*

### **Метали**

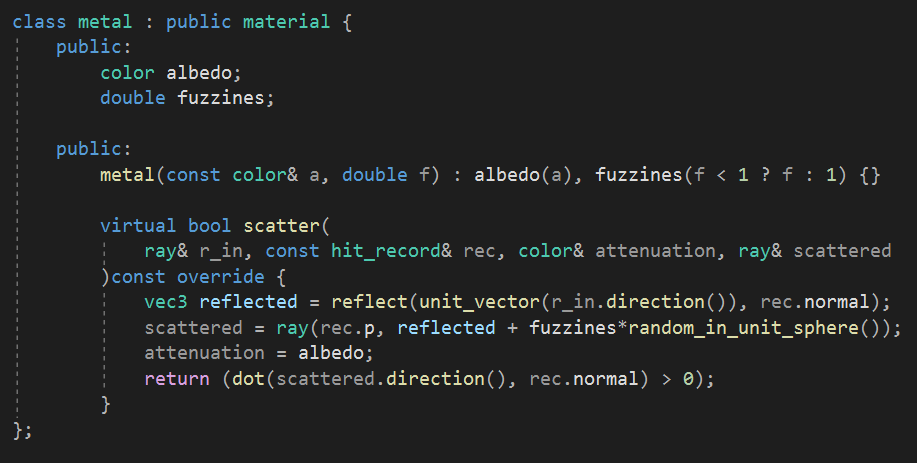
Металите, за разлика от дифузните материали, не отразяват светлината в случайна посока. На *фиг. 3.12.* може да се види как лъчите се отразяват от метална повърхност. [18]



*Фиг. 3.12. Отражение от метална повърхност*

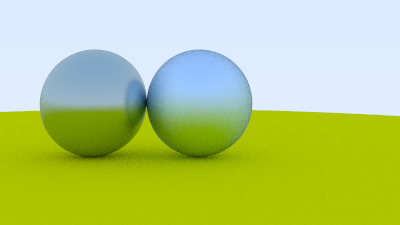
В случая е отразеният лъч, който се търси. Той може да бъде намерен по следната формула:

*Формула 3.7. Отразен лъч от метална повърхност*



*Фиг. 3.13. Клас за метали*

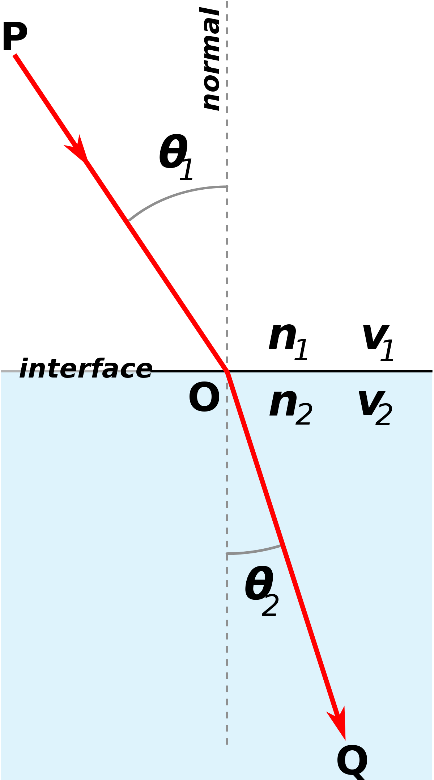
В класа за метали, освен цвета на повърхността – albedo, има и още един атрибут – fuzziness. Той е десетично число между 0 и 1, което определя колко ясна да бъде повърхността на метала. На *фиг. 3.14.* се вижда разликата между лявата сфера (fuzziness = 0.2), която е със сравнително ясна повърхност и дясната (fuzziness = 0.8), чиято повърхност е много по неясна.



*Фиг. 3.14. Метални сфери*

### **Диелектрици**

Материали като вода и стъкло се наричат диелектрици. Когато даден лъч се пресече с тях, той се разделя на отразен и пречупен лъч. Пречупването на прага на две прозрачни среди се изразява чрез закона на Снелиус (*фиг. 3.15.*). [19]



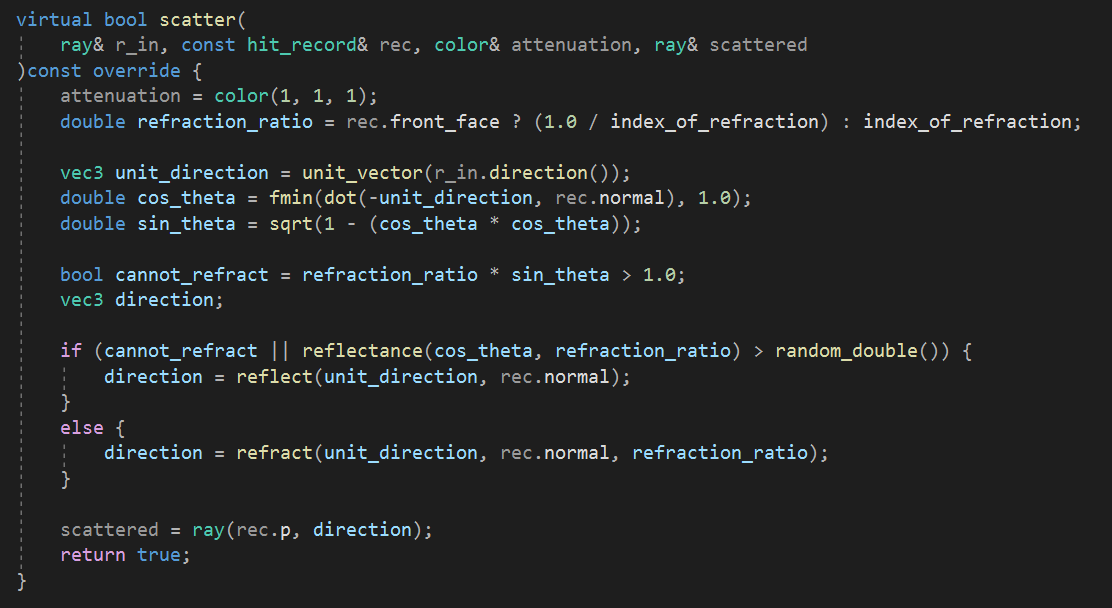
*Фиг. 3.15. Закон на Снелиус*

На *фигура 3.15.* е показан законът на Снелиус, където:

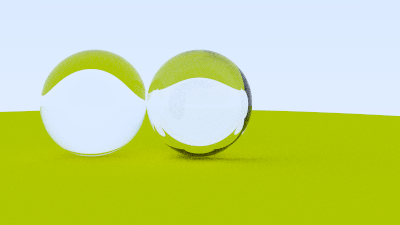
* – ъгъл на падане
* – ъгъл на пречупване
* *n* – показател на пречупване

*Формула 3.8. Закон на Снелиус*

На *фиг. 3.16.* е показана реализацията на функцията scatter за пресичане с диелектрици. Там се проверява дали лъчът може да бъде пречупен и, ако не може – ще бъде само отразен.



*Фиг. 3.16. Функцията scatter за диелектрици*

**

*Фиг. 3.17. Стъклени сфери*

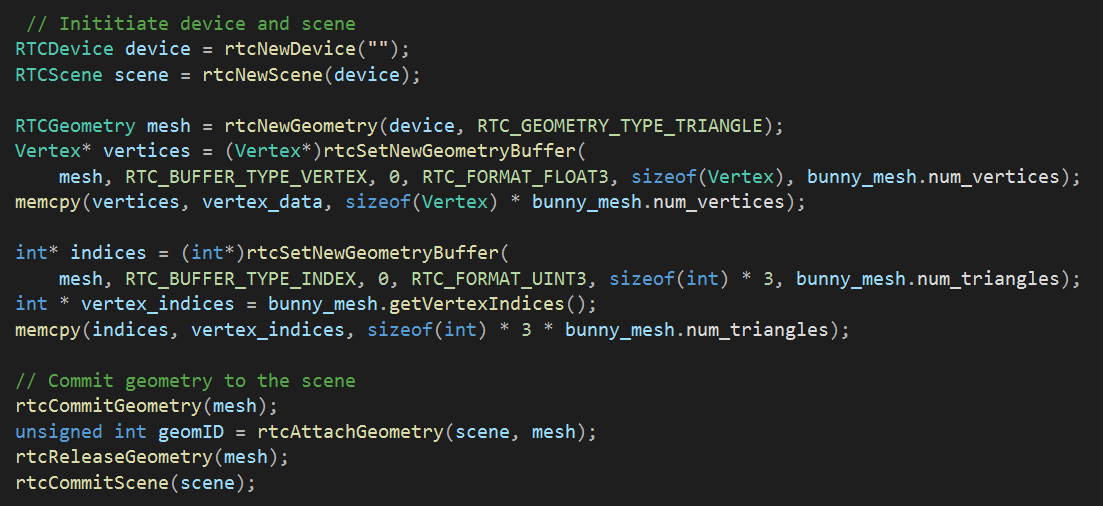
## **Пресичане с триъгълни мрежи**

### **Основни понятия**

Триъгълната мрежа (mesh) е съставена от два масива. Първият съдържа върховете на триъгълниците, като всеки връх има три координата – x, y, и z. Вторият масив съдържа поредици от по три числа, които представляват индексите на върховете от първия масив. Всяка такава поредица описва един триъгълник. [10]

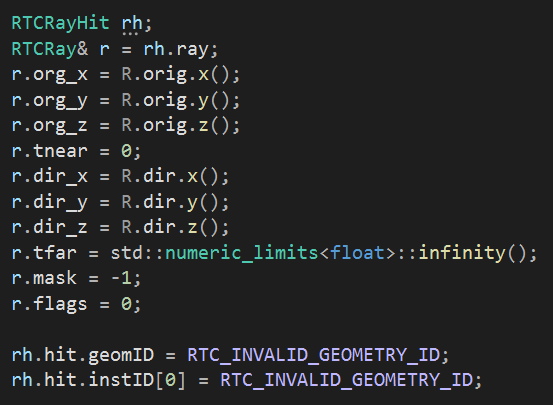
Преди да се осъществи пресичането с триъгълните мрежи се зарежда obj файл. Това се случва чрез използването на tiny\_obj\_loader.h. Файлът се зарежда под формата на триъгълна мрежа, която после ще може да бъде пресечена.

Пресичането с триъгълни мрежи се осъществява с помощта на Embree – библиотека на Intel за трасиране на лъчи. Първо се създава сцена, към която се добавя нова геометрия чрез функцията – rtcNewGeometry. Тази геометрия е триъгълна мрежа, на която двата масива за върхове и триъгълници се попълват от данните на obj файла. Накрая готовата геометрия се прилага към сцената.



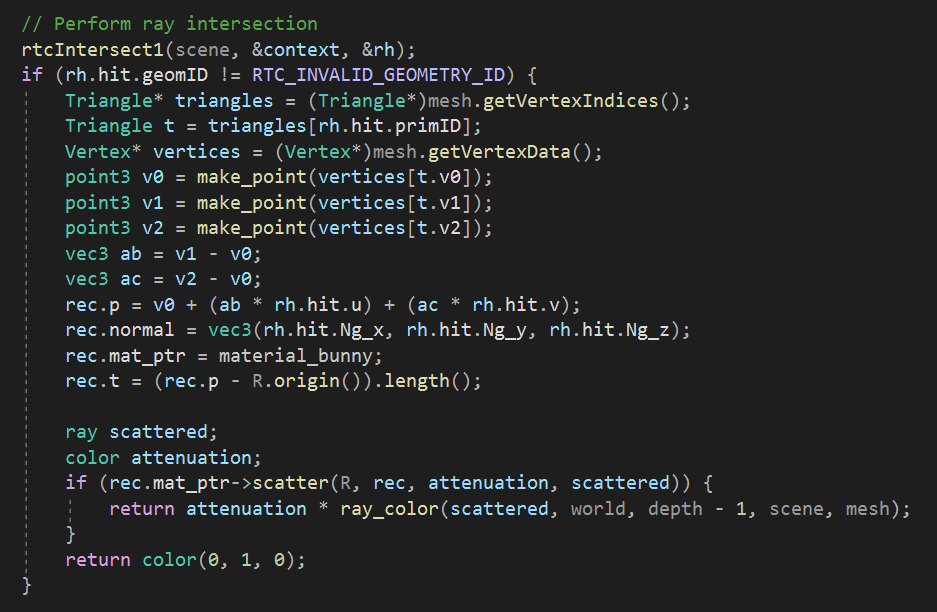
*Фиг. 3.18. Триъгълна мрежа в Embree*

Самото пресичане с триъгълната мрежа се случва във функцията ray\_color. Първо се попълва информацията за лъча RTCRay, който се съдържа в структурата RTCRayHit.



*Фиг. 3.19. Попълване данните за лъча RTCRay*

След като бъдат попълнени, вече може да се осъществи пресичането с триъгълната мрежа. На *фиг. 3.20.* е показано как се случва това.

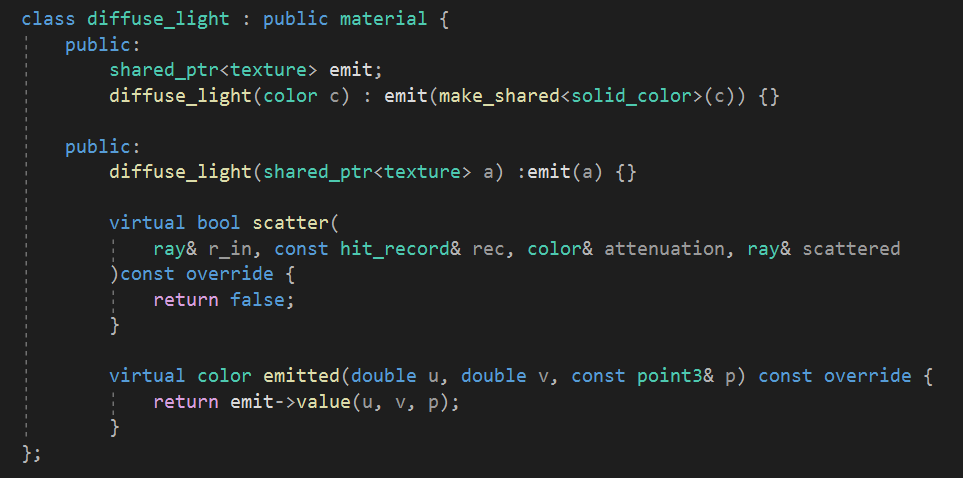


*Фиг. 3.20. Пресичане на триъгълна мрежа*

## **Дифузно осветление**

При дифузното осветление светлината идва от всички посоки, а не само от една както при точковото. По този начин цветът на лъчите се смесва с фона и цвета на обектите, с които се пресича. [20]

Реализацията на дифузното осветление е показана на *фиг. 3.21.*

**

*Фиг. 3.21. Дифузно осветление*

# **ЧЕТВЪРТА ГЛАВА**

# **Ръководства за потребителя**

## **Инсталация**

Програмата за фотореалистична графика с трасиране на лъчи е тествана и работи на Windows 10, като е компилирана, използвайки компилатора на Visual Studio 2019. Не е тествана на Mac и Ubuntu. Преди да бъде стартирана трябва да бъдат инсталирани няколко неща. Първото, от които е Embree.

Embree може да бъде изтеглено на следния линк: <https://www.embree.org/downloads.html>. Препоръчително е мястото на инсталация да бъде оставено по подразбиране.

След това трябва да се клонира проекта от GitHub. Това се случва чрез долната команда в GitBash. Ако потребителят няма инсталират Git, може да го направи на следния линк: <https://git-scm.com/download/win>.

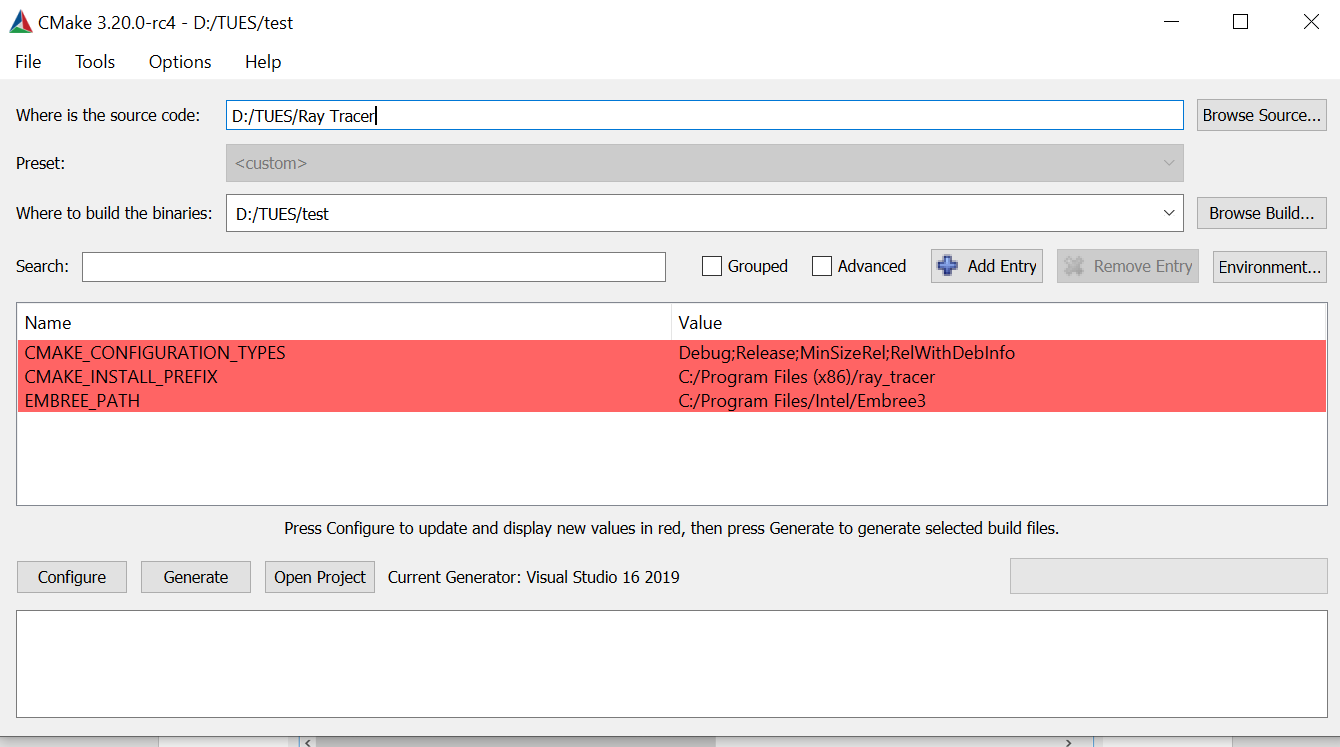
* git clone <https://github.com/DarinaAtanasova/RayTracer.git>

За генерирането на проекта заедно с всички пътеки е необходимо да се изтегли CMake: <https://cmake.org/download/>.

Когато се отвори, вътре трябва да се попълнят следните две полета:

* Where is the source code – пътеката до мястото, където е клонират проекта;
* Where to build the binaries – папка, в която да се създаде проекта (препоръчително е да е празна)

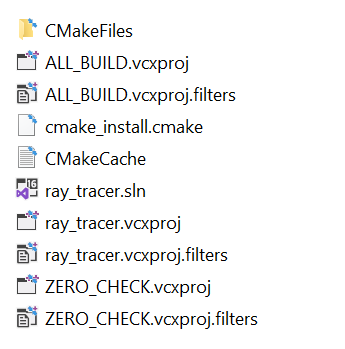
След като се попълнят се натиска бутона Configure и трябва да излезе прозореца от *фиг. 4.1.*



*Фиг. 4.1. CMake прозорец след натискането на Configure*

След това се натиска бутона Generate. В случай, че потребителят е инсталирал Embree на друго от посоченото по подразбиране място, ще трябва да отиде в полето срещу EMBREE\_PATH и да сложи пътеката до това място и чак след това да натисне бутона Generate.

Когато се отиде в папката, където е генериран проекта могат да се видят следните файлове:

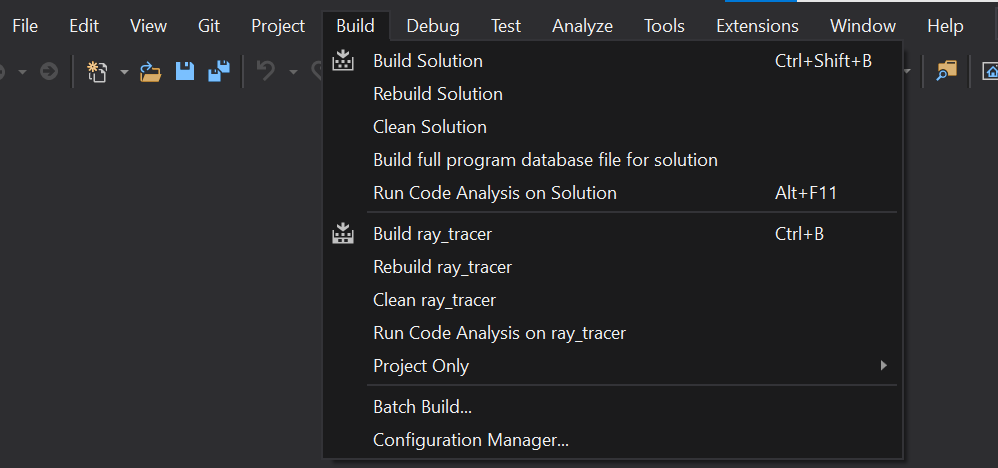


*Фиг. 4.2. Файлове в генерирания проект*

Най-лесно и удобно е проектът да се стартира чрез файла ray\_tracer.sln. За целта трябва да има свалено Visual Studio - <https://visualstudio.microsoft.com/downloads/>.

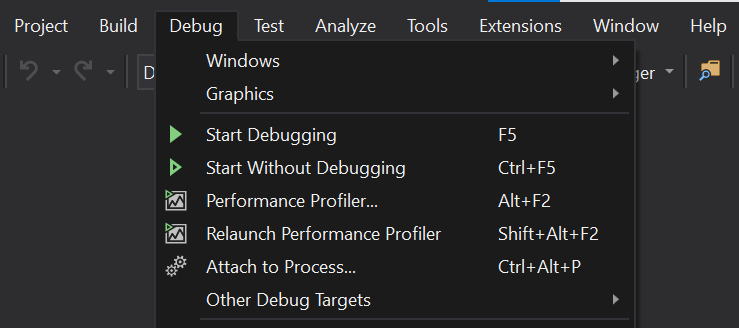
Преди да се построи проекта под горното меню вместо Debug се избира Release, за да може програмата да бъде по-бърза.

Проектът се build-ва чрез натискането на Build -> Build Solution от менюто горе или Ctrl+Shift+B.



*Фиг. 4.3. Build-ване на проект*

След това от менюто се избира Debug -> Start Without Debugging или просто Ctrl+F5.



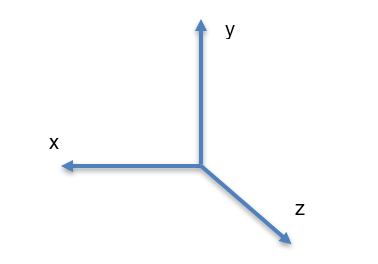
*Фиг. 4.4. Стартиране на проекта*

## **Използване на програмата**

Преди пускането на програмата има няколко неща, които могат да се променят, за да се получават различни сцени:

* Широчината и височината на изображението
* Позицията на обектите

Когато се прибавя обект (напр. сфера) към света, трябва да се зададат координатите на центъра му. Позицията на обекта се променя спрямо посоките в координата система от *фиг. 4.5.*



*Фиг. 4.5. Координатна система*

* Параметрите на камерата

Могат да се променят параметрите на камерата, за да се обхващат различни части от сцената под различен ъгъл.

* Параметрите на материалите

Основното, което може да се промени в материалите е цвета им. На този линк са показани комбинациите за всички възможни цветове, които могат да се изобразят - <https://www.tug.org/pracjourn/2007-4/walden/color.pdf>.

* 3D модела

За да се използва друг модел, който не присъства в папката 3D objects, трябва да се изтегли в obj формат и да се премести в папката с моделите. След това трябва да се промени името на файла във функцията loadMesh.

След пускането на програмата се отваря прозорец, в който се изписват още колко реда с пиксели трябва да се сметнат. Това зависи от зададените размери на изображението.

Когато програмата приключи, в папката на проекта се е генерирало изображение. То може да бъде отворено с всяка програма, поддържаща този формат. При липсата на такава програма може да се отвори онлайн с ppm визуализатор (напр. <http://www.cs.rhodes.edu/welshc/COMP141_F16/ppmReader.html>).

# **Заключение**

В настоящата дипломна работа успешно бе разработена програма за фотореалистична графика с трасиране на лъчи. Програмата покрива всички заложени функционални изисквания. Възможно е пресичането на лъчи с различни геометрии, включително и с триъгълни мрежи, използването на три основни вида материали, както и камера, която да променя ориентацията си. Генерира се изображение на базата на обектите, поставени в сцената, като се пресмятат цветовете на отделните пиксели с помощта на алгоритъм за трасиране на лъчи.

Програмата с трасиране на лъчи е разработена по такъв начин, че да позволява надграждане. Възможно е подобрението на някои от досега създадените компоненти, както и реализирането на много нови. Могат да се добавят различни източници на светлина, текстури, размазване на фокуса. Възможно е подобрението на бързината на програмата така, че да може да се смятат по-тежки сцени за по-кратко време.

# **Използвана литература**

[1] An introduction to Ray Tracing - <https://www.realtimerendering.com/raytracing/An-Introduction-to-Ray-Tracing-The-Morgan-Kaufmann-Series-in-Computer-Graphics-.pdf>

[2] Computer Graphics - <https://www.explainthatstuff.com/computer-graphics.html>

[3] Computer Generation of Photorealistic Images using Ray Tracing - <https://core.ac.uk/download/pdf/147605986.pdf>

[4] Text Editors and IDEs - <https://www.fullstackpython.com/text-editors-ides.html>

[5] Text Editors vs. IDEs: Which One Is Better for Programmers? - <https://www.makeuseof.com/tag/text-editors-vs-ides-one-better-programmers/>

[6] Best 3D Rendering Software - <https://all3dp.com/1/best-3d-rendering-software/>

[7] Chaos Vantage - <http://www.cgchannel.com/2020/12/chaos-group-releases-chaos-vantage/>

[8] RenderMan - <https://renderman.pixar.com/product>

[9] История на фотографията - <https://mpcva.org/%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D1%8F-%D0%BD%D0%B0-%D1%84%D0%BE%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F%D1%82%D0%B0/>

[10] Meshes and Scenes - <https://www.gamedev.net/blogs/entry/2261189-meshes-and-scenes/>

[11] Global Illumination - <https://cglearn.codelight.eu/pub/computer-graphics/global-illumination>

[12] Embree - <https://www.embree.org/index.html>

[13] Ray Tracing in One Weekend - <https://raytracing.github.io/books/RayTracingInOneWeekend.html>

[14] Сравнение на скаларно и векторно произведение - <https://bg.khanacademy.org/science/fizika-10-klas/xe85368f1153f10b4:za-dopalnitelna-podgotovka/xe85368f1153f10b4:magnitno-pole-dopalnitelno/v/dot-vs-cross-product>

[15] Gamma Correction - <https://www.cambridgeincolour.com/tutorials/gamma-correction.htm>

[16] C++ - <https://www.cplusplus.com/info/description/>

[17] Diffuse materials - <http://viclw17.github.io/2018/07/20/raytracing-diffuse-materials/>

[18] Reflecting materials - <http://viclw17.github.io/2018/07/30/raytracing-reflecting-materials/>

[19] Dielectric Materials - <http://viclw17.github.io/2018/08/05/raytracing-dielectric-materials/>

[20] Ray Tracing: The Next Week - <https://raytracing.github.io/books/RayTracingTheNextWeek.html>

# **Съдържание**

[**Увод** 4](#_Toc66823712)

[**ПЪРВА ГЛАВА** 6](#_Toc66823713)

[**Въведение в компютърната графика. Обзор на сходни програми с трасиране на лъчи.** 6](#_Toc66823714)

[**1.1.** **Основни термини и понятия** 6](#_Toc66823715)

[**1.1.1.** **Компютърна графика** 6](#_Toc66823716)

[**1.1.2.** **Моделиране** 7](#_Toc66823717)

[**1.1.3.** **Рендериране** 7](#_Toc66823718)

[**1.1.4.** **Трасиране на лъчи** 8](#_Toc66823719)

[**1.1.5.** **Видове лъчи** 9](#_Toc66823720)

[**1.2.** **Текстови редактори. Интегрирани среди за програмиране.** 11](#_Toc66823721)

[**1.2.1.** **Microsoft Visual Studio** 11](#_Toc66823722)

[**1.2.2.** **Eclipse** 12](#_Toc66823723)

[**1.2.3.** **Sublime Text** 12](#_Toc66823724)

[**1.2.4.** **Notepad++** 12](#_Toc66823725)

[**1.3.** **Съществуващи подобни програми** 13](#_Toc66823726)

[**1.3.1.** **Chaos Vantage** 13](#_Toc66823727)

[**1.3.2.** **RenderMan** 13](#_Toc66823728)

[**1.3.3.** **Autodesk Maya** 13](#_Toc66823729)

[**ВТОРА ГЛАВА** 15](#_Toc66823730)

[**Методи и алгоритми за реализиране на програмата с трасиране на лъчи** 15](#_Toc66823731)

[**2.1.** **Функционални изисквания** 15](#_Toc66823732)

[**2.1.1.** **Pinhole камера** 15](#_Toc66823733)

[**2.1.2.** **Anti-aliasing** 16](#_Toc66823734)

[**2.1.3.** **Видове материали** 17](#_Toc66823735)

[**2.1.4.** **Зареждане на сцена** 17](#_Toc66823736)

[**2.1.5.** **Видове геометрия** 18](#_Toc66823737)

[**2.1.6.** **Генериране на изображение** 18](#_Toc66823738)

[**2.1.7.** **Глобално засветяване** 18](#_Toc66823739)

[**2.1.8.** **Дифузно осветление** 19](#_Toc66823740)

[**2.1.9.** **Рекурсивен алгоритъм** 19](#_Toc66823741)

[**2.2.** **Избор на технологии и среда за разработка** 20](#_Toc66823742)

[**2.2.1.** **C++** 20](#_Toc66823743)

[**2.2.2.** **Embree** 20](#_Toc66823744)

[**2.2.3.** **Visual Studio** 20](#_Toc66823745)

[**ТРЕТА ГЛАВА** 22](#_Toc66823746)

[**Реализация на програмата за фотореалистична графика с трасиране на лъчи** 22](#_Toc66823747)

[**3.1.** **Векторен клас** 22](#_Toc66823748)

[**3.1.1.** **Важни операции с вектори** 22](#_Toc66823749)

[**3.1.2.** **Точка** 23](#_Toc66823750)

[**3.2.** **Цвят** 23](#_Toc66823751)

[**3.2.1.** **Гама корекция** 23](#_Toc66823752)

[**3.3.** **Лъч** 24](#_Toc66823753)

[**3.4.** **Pinhole камера** 25](#_Toc66823754)

[**3.5.** **Рендериране на изображението** 27](#_Toc66823755)

[**3.6.** **Пресичане с различни видове геометрия** 28](#_Toc66823756)

[**3.6.1.** **Абстрактен клас за геометрия** 28](#_Toc66823757)

[**3.6.2.** **Пресичане със сфера** 29](#_Toc66823758)

[**3.6.3.** **Пресичане с куб** 31](#_Toc66823759)

[**3.7.** **Материали** 32](#_Toc66823760)

[**3.7.1.** **Дифузни материали** 33](#_Toc66823761)

[**3.7.2.** **Метали** 34](#_Toc66823762)

[**3.7.3.** **Диелектрици** 36](#_Toc66823763)

[**3.8.** **Пресичане с триъгълни мрежи** 38](#_Toc66823764)

[**3.8.1.** **Основни понятия** 38](#_Toc66823765)

[**3.9.** **Дифузно осветление** 41](#_Toc66823766)

[**ЧЕТВЪРТА ГЛАВА** 42](#_Toc66823767)

[**Ръководства за потребителя** 42](#_Toc66823768)

[**4.1.** **Инсталация** 42](#_Toc66823769)

[**4.2.** **Използване на програмата** 45](#_Toc66823770)

[**Заключение** 47](#_Toc66823771)

[**Използвана литература** 48](#_Toc66823772)

[**Съдържание** 50](#_Toc66823773)