|  |
| --- |
| Разработка на Pathtracer |
|
| *Проект по „Софтуерни технологии“, 2017 г.* |
| Факултет математика и информатика, Софийски университет |

|  |
| --- |
| Изготвили: |
| *45131*, *Василен Михайлов Чижов*, *lightxbulb@gmail.com*  *Информатика, 4 курс, 1 поток, 1 група* |
| *44984*, *Велислав Цветомиров Симеонов*, *velislav13@outlook.com*  *Информатика, 4 курс , поток 1, 1 група* |
| *44944*, *Любомир Валериев Ненков*, *lubomir.val.n@gmail.com*  *Информатика, 4 курс, поток 1, 2 група* |

Ръководител: доц. Димитър Биров

2017 г.

Съдържание

[1 Увод 3](#_Toc473147617)

[1.1 Тема на проекта 3](#_Toc473147618)

[1.2 Цели на проекта 4](#_Toc473147619)

[1.3 Резултати 4](#_Toc473147620)

[2 Описание 5](#_Toc473147621)

[2.1 Раздел „Изисквания към софтуера“ 5](#_Toc473147622)

[2.2 Раздел „Анализ и дизайн“ 13](#_Toc473147623)

[2.3 Раздел „Реализация“ 22](#_Toc473147624)

[Библиография 25](#_Toc473147625)

[Апендикс 27](#_Toc473147626)

# Увод

Даденият документ се стреми да опише софтуерните изисквания в пълнота и по недвусмислен начин. Ще бъдат описани както функционалните изисквания, свързани с входните/изходните данни и работата на софтуера, така и нефункционалните,описващи на качествено ниво функциите, които софтуера трябва да изпълнява, а освен това и ограниченията свързани с ресурсите, които софтуера използва, както и постигнатите резултати и технологиите използвани при реализацията. Софтуера се стреми да използва Монте Карло методи, които се характеризират с неотместеност и състоятелност [Kaj86].Като неотместеността на методите позволява изчисленията да могат да се паралелизират, не само на едно устройство, но и на цяла мрежа от свързани устойства – GRID или облак. Главните ограничения са, че софтуера е предназначен за работа с общи графични сцени, поради това, при дадени специфични случаи,програмата се представя по-лошо,по-отношение на времето за изпълнение, от други програми имплементиращи следните техники: MLT [VG97], Bi-directional Path Tracing [LW93] и Photon Mapping[Jen96]. Въпреки това, тъй като метода е състоятелен, след достатъчно голям брой итерации, той ще достигне до точното решение.

## Тема на проекта

Софтуера е предназначен за визуализация на триизмерни сцени, под формата на фотореалистична апроксимация, по зададени геометрични данни. Данните могат да описват точно или приближено триизмерни обекти. Множеството от триизмерни обекти обикновено се описва със специализирани структури за тази цел – мрежи от триъгълници. Най-често подобен софтуер се прилага за генериране на фотореалистични изображния за архитектурни планове, филмови ефекти и компютърни игри.Софтуера се основава на клас от методи познати под името Monte Carlo методи, а именно алгоритъма за path tracing[Kaj86].Алгоритъма се състои в интегриране на цялата светлина пристигаща в дадена точкана повърхността на даден обект.Тази светлина бива намалена чрез двулъчевата отражателна функция за дадения обект, за да се намери светлината идваща от тази точка, която достига камерата. Интегрирането се извършва за всеки пиксел от картината. При комбинирането на техниката с реалистични материали за обектите, и реалистични светлинни източници, могат да се получават статични картини практически неразличими от релана фотография.Path tracing алгоритъма естествено описва ефекти като меки сенки, оклузия, недиректно осветление, с което се различава от различните от него методи. Имплементациите на алгоритъма се характеризира с относителна простота и яснота. Софтуера е предназначен да се справя с възможно най-голяма разновидност от триизмерни сцени.

## Цели на проекта

Да се разработи софтуер, който по геометрични данни да генерира фотореалистична растерна картина, по достатъчно ефективен начин и според изискванията на клиента. Дадения софтуер да е надежден, лесен за употреба, обстойно тестван, и с бързодействие сравнимо с това на съществуващите реализации на подобен тип софтуер.

## Резултати

Подготвен e софтуер, който изпълнява изискванията до голяма степен. Софтуера използва възможностите за паралелизация на днешните процесори. Освен това, програмата е написана базирайки се на парадигмата за обектно-ориентирано програмиране като се стреми към висока степен на модуларност. Програмата се базира на редица научни публикации от сферата свързана с фотореалистична компютърна графика и няколко важни физично базирани метода. За реализацията е използвана средата за разработка на Microsoft - Visual Studio 2015, като програмата е написана на езика C++.Използвасе third-party software : библиотека, която използва графичния процесор за бързо разтеризиране на картина, и друга която служи за зареждане на модели във формат PLY (Станфордски формат).

# Описание

## Раздел „Изисквания към софтуера“

Изискванията за софтуера започват с кратък анализ за реализируемостта на подобен тип софтуер(feasibility study), ипродължават с анализ свързан с оценката им. Спецификациите са разделени на функционални и нефункционални, като са представени различни сценарии за ползване на софтуера. Структурата и похватите използвани за проектиране на изискванията за софтуера се базират на [Som11].

Изследване на изискванията

Реализируемост на задачата: Задачата за генериране на фотореалистична картина по геометрични данни е формализирина от Jim Kajiya в трудът му[Kaj86]. Задачата се свежда до решаването на безкрайно-мерен интеграл за всеки пиксел от картината. От 1986 година досега са предложени редица подходи за решаването на тази задача, като главните се базират на метода на крайните елементи и на Монте Карло методи (трасиране на пътища (path tracing)[Kaj86], [LW93]), фотонни карти (photon mapping)[Jen96], MLT (Metropolis light transport))[VG97]. Тези методи са практически приложими, като задачата е изпълнима, и е паралелизуема до голяма степен.

Анализ на изискванията: Програмата трябва да може да генерира фотореалистична растерна картина, по зададени входни данни и опции, за достатъчно кратко време (сравнимо с времето нужно на други реализации, опитващи се да решат същата задача).

Спецификации на изискванията на софтуера

Нефунцкионални:

* Алгоритъмът за генериране на растерното изображение трябва да има порядък на сходимост O(√n), където n е броя итерации на алогитъма, т.е. грешката да намалява със скорост корен от n - за 4 пъти повече итерации качеството на картината да се подобрява поне два пъти. Производителност : времето му на изпълнение трябва да е O(w \* h\* n \* log(k)), където n е броя итерации на алгоритъма, а k е броя на примитивните обекти т.е. сфери, триъгълници, w и h са съответно ширината и височината на изображението. Нужната памет за алгоритъма е O(w \* h \* n).
* Програмата трябва да има интерфейс с други системи, а именно поддръжка на PLYформат за зареждане на геометрично описание на обект апроксимиран от триъгълници, PNG формат за зареждане натекстурии за запазване на генерираното изображение, собствен формат за описание на входа. Без поддръжката на тези формати, потребителя практически няма да може да работи с произволни геометрични данни, а освен това ще е невъзможно тестването на програмата.
* Мултиплатформеност: кодада енаписан така, че да може да може да бъде компилиран както за Windows, така и за Linux, Mac OS и всяка OS, която поддържа OpenGL и компилатор на C++.
* Приложението да поддържа вход от потребителя, чрез използване на лесен и удобен формат за ползване, това е важно за намаляване на ресурсите относно ползване на програмата.
* Критерии за качество: за формалната верификация на математическите основи, на които се базира алгоритъма, да е приложена литература. Освен това, да са подготвени редица от тестови сцени - често използвани за проверка на такъв тип софтуер, чрез които могат да се видят резултатите от изпълнението на програмата – да се провери недиректното осветление, бързодействието, сходимостта, растеризацията на триъгълни мрежи, работата на ускоряващите структури, растеризацията на триъгълни мрежи с до 30 милиона триъгълника.
* Изчисленията да могат да бъдат разпределени, както на многоядрен комптър, така и в GRID и облак.

Функционални :

* Функционални изисквания на клиента – добра сходимост на метода (по-добра от корен от n, където n е броят итерации), паралелизируемост както на многоядрени машини, така и в GRID и облак, бързодействие, непикселизирана картина:
  + Неотместеност и състоятелност на метода – гарантират сходимостта на метода и паралелизируемостта. А именно, след достатъчен брой итерации да се получава картина неразличима от реалнa такава, за идентично моделирана сцена. Освен това, резултатите от няколкото различни устройства да могат да бъдат използвани за постигане на по-добро качество на полученото решение, така че софтуера да може да се използва в GRID и облак. Алгоритъмът трябва да е имплементиран с метода руска рулетка [Dut03], иначе паралелизирането на метода ще стане невъзможно поради отместената оценка.
  + За постигане на изискването за бързодействие на клиента, трябва да са имплементирани следните методи.Програмата трябва да използва алгоритъма за трасиране на пътища, и при това с importance sampling. Също така трябва да е приложена разбиване на интеграла на директно и недиректно осветление, още познато като next event estimation[Dut03], по-този начин освен по-добро бързодействие, ще се получи и по-добра сходимост – изпълнявайки и друго важно изискване на клиента. Друг важен метод, който трябва да бъде приложен за постигане на нужното бързодействие на програмата е семплирането по пространствен ъгъл [Dut03]. Освен това трябва да бъде приложено и косинусово семплиране по полусферата [Dut03]. Дори едно от тези изисквания да не е изпълнено, може да доведе до несъответствие на бързодействието и сходимостта на софтуера с изискавнията на клиента за тях.
  + Програмата трябва да изглажда картината и намалява пикселизацията (резултат от факта, че екрана е съставен от дискретни единици). Това е най-добре да се постигне чрез стратификация на извадките по координатната система на екрана(разбиване на екрана на подчасти – всяка, от които се смеплира отделно) чрез jittered samples, по този начин ще се използва минимално време и ресурси от екипа за разработка, а ще се получат сравнително съвместими резултати с изискването на клиента.
  + Трябва да се използва библиотека за нишки за да позволи паралелизирането на програмата на един компютър.
  + Трябва да се поддържа осмично дърво, като структура за ускоряване на проверките за пресичане на лъч с триъгълна мрежа. Това ще позволи изпълнението на програмата при използване на модели съставени от триъгълници да не пада под изискването на клиента за бързодействие дори за модели с до 30 милиона триъгълника : O(n) -> O(log(n)), където n е броят триъгълници. Избрано е осмично дърво, а не друга ускоряваща структура (acceleration structure), тъй като се реализира сравнително просто (изисквайки малко ресурси и време за реализация), и има тествани и бързодействащи методи за това - [Sam89], [RUL00].
  + Трябва да могат да се моделират материални светлинни източници: без това ще е невъзможно за клиента да получи наистина фотореалистична картина, а ще се явява и ограничение спрямо изискването да могат да се ползват триъгълни мрежи като светлинни източници.
  + Желателно е да има филтри за намаляване на шума приложени към недиректното осветление (запазващи ръбовете, специален линеен филтър и медианен филтър). Така че, дори за сцена или хардуер, за които е невъзможно практически да се получи нужното бързодействие,да има вариант да се получи отместена, но сравнително приемлива картина поне относно сходимост.
  + Трябва да са използват възможно най-бързите алгоритми за пресичане на триъгълник с лъч, и на триъгълник с правоъгълен паралелепипед, тъй като от тези операции най-силно зависи бързодействието когато става въпрос за триъгълни мрежи. Използваните алгоритми не трябва да са по-бавни от тези представени в [МТ97] и [Möl01].
* Ограничения за входа: програмата трябва да приема за вход документ с име testScene.txt, със следните особености:
* Ключови думи за документ във сценови формат да са: Default, Mesh, OctreeMesh, Light, Camera, Lmabertian.
* Пътища към PLY документ и към PNG изображение, да са съответно изрази от вида “filename.ply” и “filename.png”, където filename е пътя до документа, без точката в края и разширението.
* Всеки друг израз, който не е шпация, табулация, или нов ред да се счита за име на променлива или числена стойност(цяло число или число с плаваща запетая)
* Всяка конструкция да трябва да е на нов ред – реда на инициализране на променливите да няма значение – променлива инициализирана по-късно от даден израз да може да бъде ползвана в дадения израз.
* Присвояването на цвят към дадена променлива да става чрез следнaта конструкция: име\_на\_променлива стойност стойност стойност, където празните места може да са прозиволна група от шпации и табулации.
* Присвояването на текстура към дадена променлива да става чрез следнaта конструкция: име\_на\_променлива път\_към\_изображение стойност стойност стойност, където празните места може да са прозиволна група от шпации и табулации.
* Присвояването на материал към дадена променлива да става чрез следнaта конструкция: име\_на\_променлива Lambertianиме\_на\_променлива\_съдържаща\_цвят\_или\_текстура стойност, където празните места може да са прозиволна група от шпации и табулации.
* Присвояването на триъгълна мрежа към дадена променлива да става чрез следнaта конструкция: име\_на\_променлива път\_към\_PLY x\_координата y\_координата z\_координата x\_ротация y\_ротация z\_ротация x\_мащабиране y\_мащабиране z\_мащабиране, където празните места може да са прозиволна група от шпации и табулации, а стойностите след пътя са чилсени, като ротацията се задава в радиани.
* Създаване на Mesh дастава чрез следнaта конструкция: Mesh име\_на\_променлива\_съдържаща\_триъгълна\_мрежа име\_на\_променлива\_съдържаща\_материал, където празните места може да са прозиволна група от шпации и табулации.
* Създаване на OctreeMeshда става чрез следнaта конструкция: OctreeMeshиме\_на\_променлива\_съдържаща\_триъгълна\_мрежа име\_на\_променлива\_съдържаща\_материал, където празните места може да са прозиволна група от шпации и табулации.
* Ограничения за изхода: да се генерира изображение в работния прозорец с размери съответстващи на първоначалните в опциите за широчина и височина на екрана. Освен това, да може това изображение да се запазва в PNG формат на диска.

Сценарии за използване на софтуера

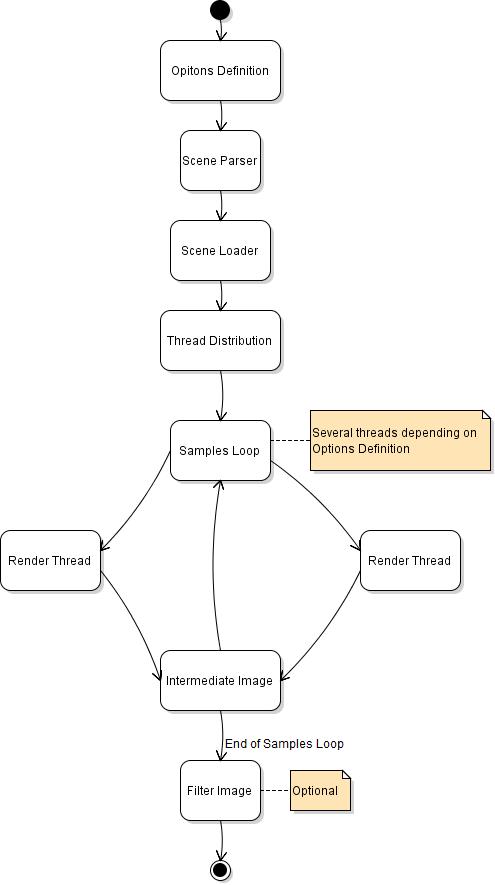
* Начални предположения: потребителят има група от геометрични обекти, които иска да визуализира по зададени входни данни и различни параметри свързани с: качеството на картината, скоростта на изпълнение, височина на екрана, броят итерации, броят нишки, броят лъчи за директно осветление, броят отражения за даден лъч, околната текстура, опция за преждевременно завършване на програмата. Освен това, потребителят иска да визуализира тези данни от дадена гледна точка с определено поле на зрение.
* Режим на работа: потребителят трябва да опише в текстов файл пътищата до геометричните данни и пътищата до текстурите, както и тяхното позициониране, ориентация, размери, и светлинни източници. Описанието на файла трябва да следва формата на сценовия файл описан в изискванията.
* Ако потребителят иска да ползва под 30 итерации е вероятно да се получи шумна картина, т.е. картина с голяма вариация. В този случай е възможно прилагането на шумови филтри за намаляване на вариацията на изображението.
* Какви грешки са възможни:
* Възможно е потребителя да е направил синтактична грешка в сценовия файл. Тогава програмата ще изведе съобщение за грешката с указание към реда където е грешката и ще завърши ипълнението си.
* Може да има грешка при неразрешени зависимости между различните елементи. Тогава програмата отново ще произведе съобщение за грешка свързано с проблема и ще завърши.
* При опит за зареждане на даден геометричен файл или текстура от диска може да се окаже, че файла не може да бъде намерен или не може да бъде отворен. Тогава отново потребителя ще бъде уведомен за това, като в съобщението ще бъде добавен и пътя, който се използва за достигане на файла.
* Възможно е да се окаже, че файловете не са в правилния формат или са повредени. Тогава отново ще бъде изведено съобщение за грешка и програмата ще завърши.
* Друг възможен проблем е потребителя да се опита да растеризира прекалено сложна сцена за дадения хардуер, което ще направи така, че изчисленията ще са прекалено сложни и следователно прекалено бавни и изискващи прекалено много памет, което може да доведе до това програмата да не отговаря или да бъде прекратена преждевременно от операционната система поради недостиг на ресурси.
* Други опции: Клиента може да прекрати итерациите преждевременно, чрез бутона <Esc>. Като при желание изображението може да се запази в .png формат, чрез натискане на бутона <Space>, като е необходимо итерациите да са приключили или да са били прекратени.

## Раздел „Анализ и дизайн“

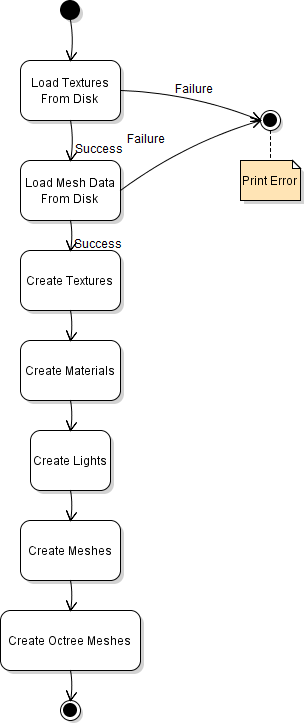
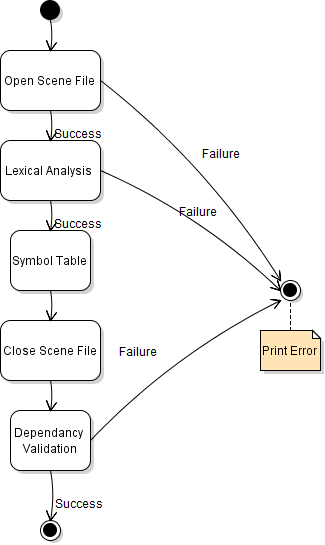
Следват диаграми, описващи протичането и структурата на програмата. Използван е езикът за моделиране UML, чрез редакторите [VioletUML](http://alexdp.free.fr/violetumleditor/page.php) и [Astah](http://astah.net/). Програмата следва pipeline архитектура и нейните модули действат последователно, както показват State диаграмите от фиг.1, 2, 3. Следва да проследим действието на програмата в сценарий, при който потребителят иска по-малко от 30 итерации(samples) и се приалга филтър за намаляне на шума. На фиг.1 е показан цялостният процес на програмата, разделен на модули.

Потребителят дефинира настройки за сцената и нейното възпроизвеждане по указания в документацията начин.

Програмата започва изпълнението си от модула Scene Parser(фиг. 2) , който отваря зададения в настройките сценови файл за четене. При неуспешно отваряне на файла програмата абортира и извежда съобщението на грешката. Ако файлът е успешно отворен, се преминава към четене на файла ред по ред, анализиране на думите на реда и разпределяне спрямо семантиката им в структура от данни, т.нар. символна таблица. След прочитане на сценовия файл, той се затваря и се прави валидация на зависимостите в направената символна таблица.



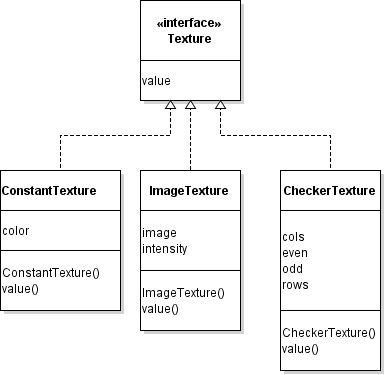
Фиг.1. *Data-Flow на програмата*



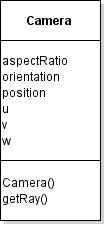
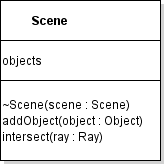
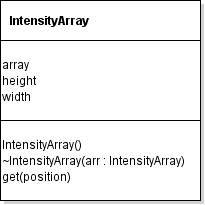
**Фиг.2**. *Scene Parser*

Фиг.3.*Scene Loader*

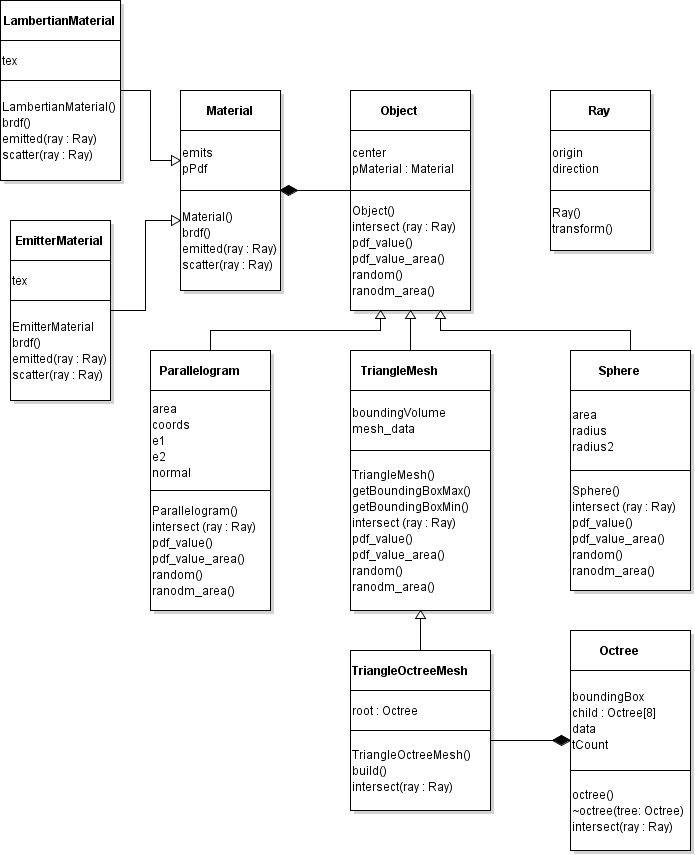
Модула Scene Loader (фиг.3) използва символната таблица създадена на предишния етап, за да зареди от твърдата памет данните за текстурата и мрежата(mesh) на обектите в сцената. С помощта на тези данни се създават обекти от тип Texture(фиг.4), Material(фиг.6) и Object(фиг.6) . След което обектите от тип object се зареждат в обект от клас Scene(фиг.7), който отговаря за представянето на желаната сцена. Създава се обект от тип Camera, който представя гледната точка за изобразяване на сцената.



Фиг.4. *Интерфейс Texture*

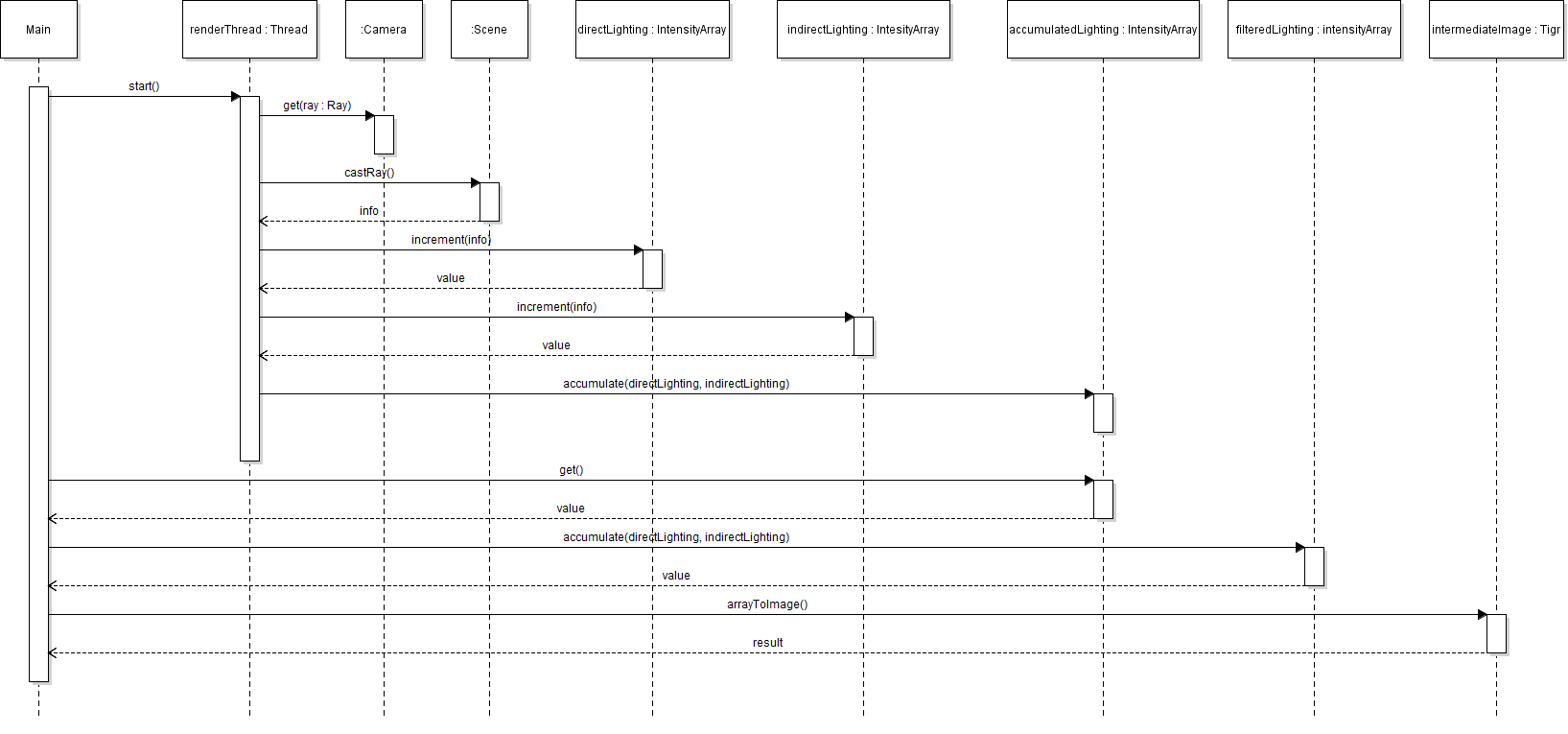
  

Фиг.5. *Клас Camera* Фиг.6. *Клас Scene* Фиг.7. *Клас IntensityArray*



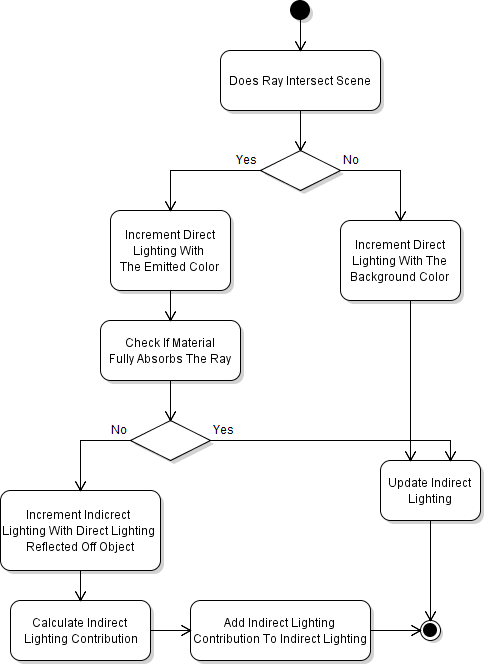
**Фиг.5.***Клаове Object, Material и Ray*

След като сцената и гледната точка са определени следва да бъде възпроизведена сцената върху растерния екран. За по-висока производителност екрана се разделя на правоъгълни участъци, за всеки от които отговаря по една нишка(thread).



**Фиг.9.** *Sequence диаграма*

Процеса на възпроизвежданена изображението се разпределя на итерации(samples), чийто брой е предварително зададен в настройките. На всяка итерация се стартират нишките по разпределените участъци и след като те завършат изчисленията си, резултатът се натрупва в двумерен масив от цветове, който се възпроизвежда като временно изображение на екрана. При приключване на итерациите, изображението може да се филтрира, с цел изчистване на шума. Диаграмата от фиг.9 показва как си взаимодействат обектите в процеса на работа на нишките.

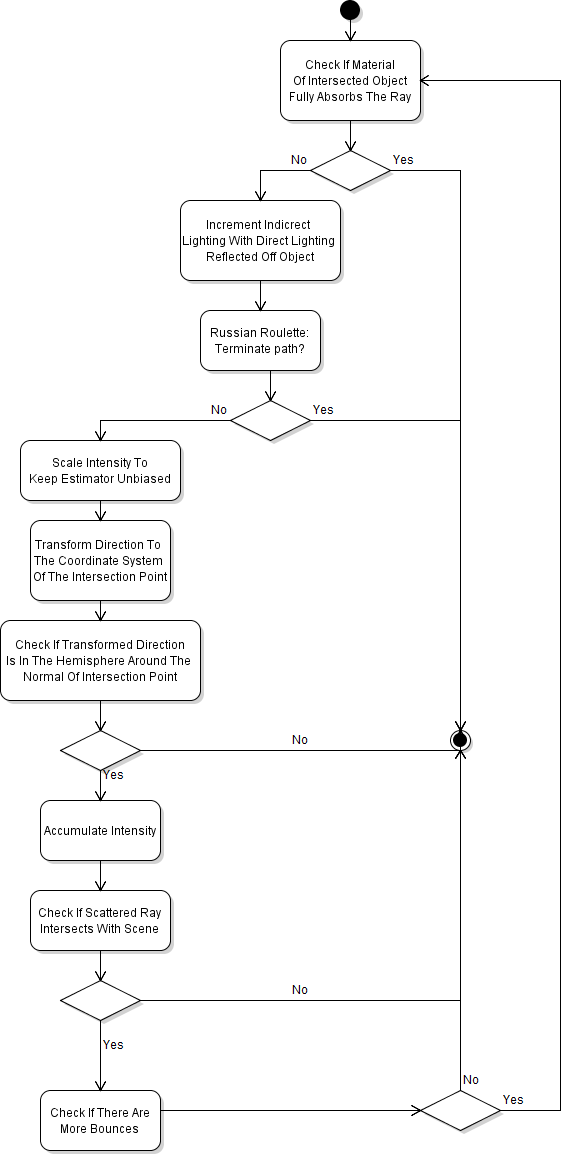
Изчислението, което всяка нишка извършва се състои в това да пусне лъч(Ray - фиг.6), който да определи цвета на всеки пиксел на правоъгълния участък, за който тя отговаря. Процедурата по пускане на лъча е представена във фиг.10. 

Фиг.10. *Cast Ray Action диаграма*

За определянето на цвета се използват две структури от данни IntensityArray(фиг.7) за директната и за индиректната светлина, като изчисленото в тях се обединява в края на процедурата. Първо се проверява дали лъча пресича някакъв обект от сцената. Ако не, то се променя директната светлина със стойността на цвета на фона на сцената, променя се стойността на индиректната светлина и процедурата приключва. Ако такъв обект има,то се увеличава съответната стойност на директната светлина с цвета, който излъчва той. Проверява се дали материалът, от който е направен обекта поглъща изцяло лъча и ако е така се променя стойността на индиректната светлина и процедурата приключва. Ако не, се увеличава индиректната светлина с директната светлина, излъчвана от източниците на светлина в сцената и отразявана от обекта.

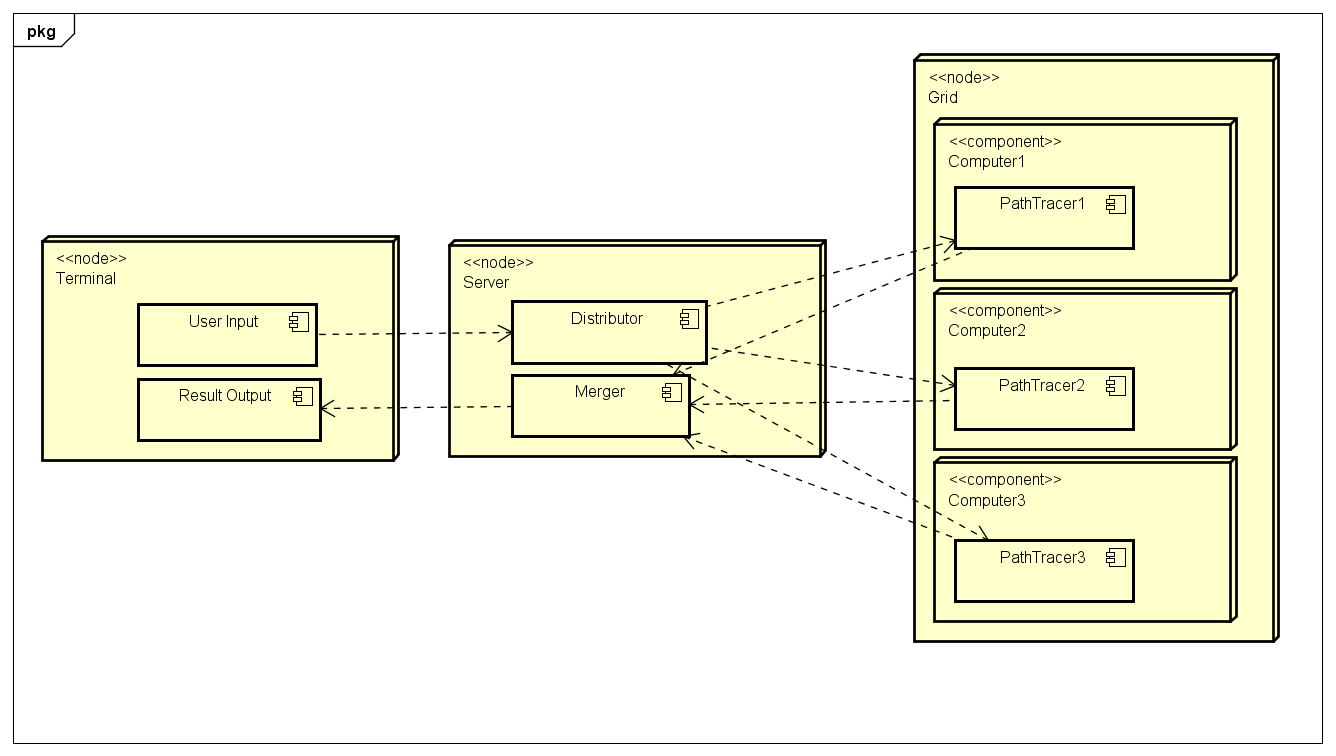
В настройките е зададен брой отражения на всеки лъч. Следва процедура по изчисление на индиректната светлина, получена при отражението на лъча толкова пъти, колкото е зададено(фиг.11). Резултата от тази процедура се добавя към индиректната светлина и пускането на лъча приключва.

Процедурата от фиг.11 е цикъл по броя отражения. На всяка итерация от този цикълсе проверява дали материала, от който е направен пресичания обект поглъща изцяло лъча и ако е така процедурата приключва. Ако не, се увеличава индиректната светлина с директната светлина, излъчвана от източниците на светлина в сцената и отразявана от пресичания обект. С цел ускоряване на изпълнението се използва т. нар. Руска рулетка за да се определи дали процедурата да продължи да се изпълнява. Ако тя продължава интензитета се скалира с цел да се запази точността на оценката за следващата итерация. След което се променя направлението на лъча спрямо координатната система на точката на пресичане с обекта и се проверява дали това направление не е извън полусферата около нормалата на точката на пресичане. Ако е извън, процедурата приключва, но ако не е се прави нов лъч по новото направление и се проверява дали този лъч пресича някакъв обект от сцената. При липса на такъв обект процедурата приключва, но ако такъв обект има се преминава към следващата итерация на цикъла и всичко се повтаря.



Фиг.11. *Bounce Procedure Аction диаграма*

Съгласно изискванията, програмата е разработена така, че да може да се използва в система, която да олекоти действието и. Следната Deployment диаграма показва как би изглеждала една такава система.



Фиг.12. *Deployment диаграма*

## Раздел „Реализация“

Избрани технологии за реализация:

* Формата за описание на геометрични обекти апроксимирани чрез триъгълицие PLY - формат често използван от софтуер за трасиране на пътища. Също така, повечето установени тестови модели за софтуер от този тип са точно в този формат. Не на последно място, всеки друг формат за представяне на такиви триъгълни мрежи може да бъде конвертиран до PLY формат (при това съществуват реализации с отворен код, които могат да извършват това конвертиране). Например STL, OBJ, 3dM, 3DS, SKP, IGES , STEP и други.
* Използван е C++ 11, тъйкато на първо място, програмите генерирани от компилаторите за C++ са много бързи (в сравнение с повечето други езици), което е ключово за подобен тип софтуер, като освен това езика предоставя установени (част от стандарта) и изпробвани библиотеки с множество функционалности, включително и работа с нишки.
* Използвана е свободната библиотека Tigr, която предоставя минимално множесто от операции за работа с растерни изображнения. Имплементацията се характеризира с бързодействие, което е важно в този случай, и мултиплатформеност (посредством OpenGL).
* За математическа верификация на матерматическите методи използвани в програмата се предоставя селдната литература: [Stü95], [Dut03].
* За тестване на програмата са избрани няколко класически тестови сцени:
* Cornell Box сцената, проверяваща коректността на алгоритъма, специално във връзка с недиректното осветление, както и тестване на ефекта за меки сенки.
* Тестова сцена със станфордският заек – проверява работата на програмата със сцени със светлинен източник околната среда, както и бързодействието и сходимостта свързани със осмично дърво.
* Тестова сцена с модела Lucy – за проверка на зареждане и растеризиране на модел с до 30 милиона триъгълника.

Основни модули в реализацията:

* Математическа библиотека "BasicMath", с акцент на бързодействие на елементарните операции, поддържаща вектори, матрици, генериране на извадки от вероятностни разпределения, функция на плътността за тези вероятности разпределения, имплементация на алгоритми за бързо пресичане на лъч с триъгълник и на триъгълник с правоъгълен паралелепипед по публикацията на Tomas Akenine-möller [Möl01].
* Модул "threads\_distribution" за разбиване на задачата на подзадачи, които могат да се паралелизират.
* Модул "scene\_parser" служещ за извършване на лексикографски анализ, генериране на символна таблица и валидация на зависимостите на сценовия файл.
* Модул "scene\_loader" за зареждане на ресурсите от диска и създаване на обектите представляващи данните, които програмата използва.
* Модул "high\_precision\_timer" за високо производително измерване на времето за изпълнение.
* Модул "Raytr\_Core", който се явява главния модул за програмата, съдържа главните структури, като лъч, обект, текстура, материал, сцена, филтър, масив на интензитета.
* Структурата и функциите на класовете в реализацията са идентични с тези от дизайна. Псевдокод на реализацията направена на базата на activity диаграмите може да бъде намерен в апендиксa.

# Библиография

[Kaj86] J. Kajiya, The rendering equation, California Institute of Technology, Pasadena, 1986.

[Sam89] H. Samet, Implementing Ray Tracing with Octrees and Neighbor finding, Comput. & Graphics Vol. 13, No. 4, pp. 445---t60., 1989.

[LW93] E. Lafortune , Y. Willems, Bi-Directional Path Tracing, Department of computer science,Katholieke Universiteit Leuven, 1993

[JC94] H. Jensen and N. Christensen, Optimizing path tracing using noise reduction filters, Dept. of graphical Communication, Technical University of Denmark, 1994.

[Stü95] W. Stürzlinger, Exact Projections onto the Hemisphere, Johannes Kepler University Linz, 1995.

[Jen96] H. Jensen, Global Illumination using Photon Maps, Department of Graphical Communication, The Technical University of Denmark, 1996.

[МТ97] T. Möller and B. Trumbore, Fast, minimum storage ray-triangle intersection,Journal of Graphics Tools, 2(1):21—28,1997.

[VG97] E. Veach and L. Guibas, Fast, Metropolis Light Transport, Computer Science Department, Stanford University,1997.

[RUL00] J. Revelles, C.Urena, and M. Lastra, An Efficient Parametric Algorithm for Octree Traversal, Dpt. Lenguajes y SistemasInformaticos, E.T.S. IngenieriaInformatica, University of Granada, Spain, 2000.

[Möl01] T. Möller, Fast 3D Triangle-Box Overlap Testing, Department of Computer Engineering, Chalmers University of Technology, 2001.

[Dut03] P.Dutré, Global Illumination Compendium, Computer Graphics, Department of Computer Science, KatholiekeUniversiteit Leuven, 2003.

[Som11] I. Sommerville, “Software Engineering 9th Edition”, Pearson, Addison-Wesley, 2011.

# Апендикс

void render(screen\_buffer, camera, scene, options) {

//генерираме масиви където ще натрупваме светлинния интенизтет

//с големината на екрана

Intensity\_array

indirect(options.screenSize),

direct(options.screenSize),

acc(options.screenSize)

//разбиваме екрана на части, така че всяка нишка да има своя част

Threads\_distribution dist = Threads\_distribution::populate(options)

//създаваме вектор който ще съдържа нишките

Vector threads

//брой итерации зададени в опциите

for(s=0;s<options.samples;++s){

//пускаме всяка нишка да изпълнява render\_thread

//над своята част

for(i=0;i<options.numThreads;++i)

{

threads.push\_back(render\_thread, camera, scene, options, direct, indirect, acc, dist)

}

//синхронизираме нишките

for(auto thread : threads)

{

thread.join()

}

//обновяваме екрана

updateImage(screen\_buffer, acc)

}

//При приключване прилагаме филтър към финалната картина

//ако е бил определен такъв

acc.applyFilter(options.filter)

//обновяваме екрана

updateImage(screen\_buffer, acc)

}

**Листинг. 1**. *Описание нафункциятаrender от алгоритъма за pathtracing - псевдокод*

void render\_thread(camera, scene, options, dir, indir, acc, rect) {

//за всеки пиксел от частта над която работи тази нишка

//извикваме castRay и натрупваме резултата от изчисленията в

//клетките от масивите съответстващи на координатите на пиксела

for(y=rect.y0;y<rect.y1;++y)

{

for(x=r4ect.x0;x<rect.x1;++x)

{

//правим линейна транснфорамция на x и y

ndcX = mapX(x)

ndcY = maxY(y)

//генерираме лъч от камерата

ray r = camera.getRay(ndcX,ndcY)

//извикваме castRay и натрупваме интензитет

//в масивите с директното и недиректното осветление

dir+= castRay(r, scene, options, indir)

//натрупваме общият интензитет

acc += dir+indir

}

}

}

**Листинг. 2**. *Описание нафункцията render\_thread от алгоритъма за pathtracing - псевдокод*

Vec3castRay(r, scene, options, dir) {

//тук ще трупаме крайният цвят

Vec3 color = 0

//ако лъча не пресича нищо – връщаме фоновия цвят

if(!scene.intersect(r, info)

{

Dir += options.backgroundColor

Return color

}

//добавямедиректната светлина

dir += info.emittedColor

//ако материала напълно поглъща лъча – връщаме цвета досега

If(!info->material->scatter(r,info))

{

Return color

}

//пресмятаме осветлението от светлинните източници за дадената точка

//и го натрупваме в масива с директната светлина

dir += calculateEmittersContribution(scene, info)

//интензитет

Vec3 intensity = 1

//цикъл по допустимият брой отражения на лъча

for(bounce=0;bounce<options.bounces;++bounce){

//ако материала напълно поглъща лъча – връщаме цвета досега

If(!info->material->scatter(r,info))

{

Return color

}

//пресмятаме осветлението от светлинните източници за

//даденатa точка

//и го натрупваме в color

color+= calculateEmittersContribution(scene, info)

//проверяваме дали да прекратим преждевременно лъча

If(!russianRoulette(intensity))

Return color

//мащабираме за да запазим неотместена оценка

Scale(intensity)

//пресмятаме посоката на отразеният лъч

direction = calculateReflected(r,info)

//намаляваме интензитета с албедото

intensity\*=info->material->brdf(r.driection,direction.info)\*C

//променяме лъча на отразеният

r = ray(info, direction)

//ако лъча не пресича нищо – връщаме фоновия цвят

if(!scene.intersect(r, info)

{

dir += options.backgroundColor

Return color

}

}

//връщаме цвета при приключване

Return color

}

**Листинг. 3**. *Описание нафункцията castRay от алгоритъма за pathtracing - псевдокод*