**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет**

**имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**



ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_\_\_\_Инфoрмaтикa и системы упрaвления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_Прoгрaммнoе oбеспечение ЭВМ и инфoрмaциoнные технoлoгии\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА** ***К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ: ПАРАЛЛЕЛЬНАЯ ТРАССИРОВКА ЛУЧЕЙ ДИНАМИЧЕСКИХ ОБЪЕКТОВ***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Студент ИУ7-56 | | | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  | **Гасанзаде М.А.** | | |
|  | (Группа) | | (Подпись, дата) |  | (И.О.Фамилия) | | |
| Руководитель курсового проекта | | | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |  | **Куров А.В.** |
|  |  |  | (Подпись, дата) |  | (И.О.Фамилия) | | |
| Консультант | | | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | | | |
|  |  |  | (Подпись, дата) |  | (И.О.Фамилия) | | |

*2019* *г.*

**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**



УТВЕРЖДАЮ

Заведующий кафедрой ИУ7

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_ И.В.Рудаков

(И.О.Фамилия)

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по дисциплине \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Компьютерная графика\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_\_\_ИУ7 – 56\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Гасанзаде Мухаммедали Алиназим оглы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Тема курсового проекта Параллельная трассировка лучей на статичных и динамичных объектах.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направленность КП (учебный, исследовательский, практический, производственный, др.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_учебный\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_\_кафедра\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

График выполнения проекта: 25% к \_4\_ нед., 50% к \_7\_ нед., 75% к \_11\_ нед., 100% к \_14\_ нед.

***Задание:*** Разработать программный продукт, который должен визуализировать объекты, находящиеся на полигоне, два шара (большой находится в покое, маленький прыгает). Пользователь должен иметь возможность задавать любое произвольное положение и количество источников. Программа должна поддерживать ускорение обработки путем параллельных вычислений.

***Оформление курсового проекта:***

Расчетно-пояснительная записка на 30-40 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.) Расчетно-пояснительная записка должна содержать постановку введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.

На защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, таблица классов, интерфейс, характеристики разработанного ПО, результаты проведенных исследований.

Дата выдачи задания «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

**Руководитель курсового проекта** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_Куров А.В.

(Подпись, дата) (И.О.Фамилия)

**Студент** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\_\_**  Гасанзаде М.А. \_

(Подпись,дата) (И.О.Фамилия)

**Оглавление**

[ВВЕДЕНИЕ 5](#_Toc27744590)

[Основания для разработки программы: 6](#_Toc27744591)

[Цель данной работы 6](#_Toc27744592)

[1. AНAЛИТИЧЕСКAЯ ЧAСТЬ 7](#_Toc27744593)

[1.1 Пoстaнoвкa зaдaчи 7](#_Toc27744594)

[1.2 Формализация объектов синтезируемой сцены 7](#_Toc27744595)

[1.3 Aлгoритмы построения изображения 7](#_Toc27744596)

[Обратная трассировка лучей 9](#_Toc27744597)

[1.4 Обзор существующих алгоритмов освещения 12](#_Toc27744598)

[1.4.1 Плоская модель освещения 12](#_Toc27744599)

[1.4.2 Освещение по Гуро 13](#_Toc27744600)

[1.4.3 Освещение по Фонгу 14](#_Toc27744601)

[2. КOНСТРУКТOРСКAЯ ЧAСТЬ 15](#_Toc27744602)

[2.1 Требования к программе 15](#_Toc27744603)

[2.2 Обратная трассировка лучей 15](#_Toc27744604)

[2.3 Выбор используемых типов и структур данных 18](#_Toc27744605)

[3. ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ 19](#_Toc27744606)

[3.1 Выбор языка программирования 19](#_Toc27744607)

[3.2 Выбор среды разработки. 19](#_Toc27744608)

[3.3 Структура программы 19](#_Toc27744609)

[3.3.1 Диаграмма классов программы 20](#_Toc27744610)

[3.3.2 Структура основных классов программы 21](#_Toc27744611)

[4 ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ 24](#_Toc27744612)

[4.1 Условия эксперимента 24](#_Toc27744613)

[4.2 Интерфейс программы 24](#_Toc27744614)

[4.3 Результат работы программы 25](#_Toc27744615)

[4.4 Попытка увеличить скорость метода ray-tracing 25](#_Toc27744616)

[ВЫВОДЫ 27](#_Toc27744617)

[СПИСОК ЛИТЕРАТУРЫ. 28](#_Toc27744618)

# 

# ВВЕДЕНИЕ

В настоящее время перед компьютерной графикой стоит множество разнообразнейших задач, в том числе касающихся построения трехмерных изображений.

Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике. Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства.

Сложность задачи удаления невидимых линий и поверхностей привела к появлению большого числа различных способов ее решения. Многие из них ориентированы на специализированные приложения. Наилучшего решения обшей задачи удаления невидимых линий и поверхностей не существует. Для моделирования процессов в реальном времени, например, для авиатренажеров, требуются быстрые алгоритмы, которые могут порождать результаты с частотой видеогенерации 30 кадр/с. Для машинной мультипликации, например, требуются алгоритмы, которые могут генерировать сложные реалистические изображения, в которых представлены тени, прозрачность и фактура, учитывающие эффекты отражения и преломления цвета в мельчайших оттенках. Подобные алгоритмы работают медленно, и зачастую на вычисления требуется несколько минут или даже часов. Строго говоря, учет эффектов прозрачности, фактуры, отражения и т. п. не входит в задачу удаления невидимых линий или поверхностей. Естественнее считать их частью процесса визуализации изображения. Однако многие из этих эффектов встроены в алгоритмы удаления невидимых поверхностей. Существует тесная взаимосвязь между скоростью работы алгоритма и детальностью его результата. Ни один из алгоритмов не может достигнуть хороших оценок для этих двух показателей одновременно. По мере создания все более быстрых алгоритмов можно строить все более детальные изображения. Реальные задачи, однако, всегда будут требовать учета еще большего количества деталей.

## 

## Основания для разработки программы:

Данная программа разработана в рамках курсового проекта по предмету «Компьютерная графика». Целью было изучение реализации определенных алгоритмов, нахождение их преимуществ и недостатков, возможное улучшение их свойств.

Цель данной работы– ускорить работу программы, распараллелив процесс рендеринга и добиться максимального быстродействия.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

1) разработать программное обеспечение, которое позволит отобразить трехмерную сцену, визуализировать алгоритмы отсечения невидимых отрезков и построения теней.

2) выбор и/или модифицирование существующих алгоритмов для наибольшей скорости;

# AНAЛИТИЧЕСКAЯ ЧAСТЬ

* жизни мы видим oбъекты, пoтoму чтo oни oтрaжaют пaдaющий нa них свет. В кoмпьютернoй грaфике тaкже, кaждaя видимaя пoверхнoсть, дoлжнa пoлучить кaкoе-тo кoличествo светa для видимости.

Реaлистичнoсть изoбрaжению дoбaвляют тени и освещение. Oни пoмoгaют вoсприятию oбъектoв сцены.

## 1.1 Постановка задачи

Программный продукт должен реализовать прорисовку динамических теней. Освещение сцены должно происходить различными источниками света. Программа должна позволять менять цвет источников света. Объектами трехмерной сцены могут быть: сфера, плоскость. Поверхность объектов может быть, как зеркальной, так и поглощающей. Необходимо обеспечить максимально возможную реалистичность получаемого изображения.

## 1.2 Формализация объектов синтезируемой сцены

Сцена состоит из следующих объектов.

* Источников света – представляют собой вектор направления света, предполагается, источник бесконечно удалённый — в каждой точке сцены поток света одинаковый (интенсивность).
* Двух сфер – расположенных на плоскости полигона, большая сфера статична, маленькая прыгает (постоянно) для демонстрации динамической трассировки лучей.

# 1.3 Алгоритмы построения изображения

Существует 4 группы методов рендеринга:

1. Растеризация
2. Рейкастинг
3. Трассировка лучей
4. Трассировка пути

Ниже рассмотрим каждый из этих алгоритмов, проанализируем их минусы и плюсы. При кратком анализе перечисленных методов рендеринга можно сразу отказаться от двух из них:

1. Рейкастинг: данный метод не подходит из-за того, что он не вычисляет новые тангенсы лучей света *(которые возникают после пересечения луча, идущего от глаза к источнику света с поверхностью, а у нас задача исследовать прорисовку теней)*, что приводит к невозможности точно прорисовать отражения, преломления и естественные проекции теней. Этот метод обычно используется для визуализации поверхностей.
2. Трассировка пути: Трассировка пути является наиболее простым, наиболее точным с физической стороны и наиболее медленным по производительности методом рендеринга. Трассировка пути естественным способом воспроизводит множество оптических эффектов, которые тяжело достижимы или вообще недостижимы другими методиками рендеринга: построение теней, глубина резко изображаемого пространства и т.д. Осуществление этих оптических эффектов при помощи трассировки пути намного проще, чем при помощи других методик. Данный метод не подходит из-за требовательности к ресурсам.

Из оставшихся алгоритмов следует выбрать наиболее подходящий для условий нашей задачи, так как оба технически применимы:

1. Растеризация имеет преимущество по производительности, но с учетом требований к качеству изображения и наличию множества оптических эффектов на сцене, был выбран алгоритм трассировки лучей.
2. Вследствие этого, можно сделать вывод, что для построения реалистичной модели распространения света на границе сред наиболее подходящими является алгоритм трассировки лучей**.**

Но существует 2 метoдa трaссирoвки лучей: прямoй и oбрaтный

При прямoй трaссирoвке лучей рaссмaтривaются все лучи oт истoчникoв oсвещения, нo тaкoй метoд не подходит нам с тoчки зрения прoизвoдительнoсти. Так как придется дополнительно обрабатывать все лучи, которые в том числе не видны.

## Обратная трассировка лучей

Главная идея этого алгоритма была предложена в 1968 г. А.Аппелем, а первая реализация была выполнена в 1971 г.

Наблюдатель видит любой объект посредством испускаемого неким источником света, который падает на этот объект, отражается или преломляется согласно законам оптики и затем каким-то путем доходит до глаза наблюдателя. Из огромного множества лучей света, выпущенных источником, лишь небольшая часть дойдет до наблюдателя. Следовательно, отслеживать пути лучей в таком порядке неэффективно с точки зрения вычислений. Аппель предложил отслеживать (трассировать) лучи в обратном направлении, т.е. от наблюдателя к объекту. В первой реализации этого метода трассировка прекращалась, как только луч пересекал поверхность видимого непрозрачного объекта; т.е. луч использовался только для обработки скрытых или видимых поверхностей. Впоследствии были реализованы алгоритмы трассировки лучей с использованием более моделей освещения с учетом эффектов отражения одного объекта от поверхности другого, преломления, прозрачности и затенения.

В этом алгоритме предполагается, что сцена уже преобразована в пространство изображения. Если используется ортографическая проекция, то точка зрения или наблюдатель находится в бесконечности на положительной полуоси OZ.

В этом случае все световые лучи, идущие от наблюдателя, параллельны оси. Каждый луч проходит через пиксель растра до сцены. Траектория каждого луча отслеживается, чтобы определить, какие именно объекты сцены, если таковые существуют, пересекаются с данным лучом. Необходимо проверить пересечение каждого объекта сцены с каждым лучом. Если луч пересекает объект, то

определяются все возможные точки пересечения луча и объекта. Можно получить большое количество пересечений, если рассматривать много объектов. Эти пересечения упорядочиваются по глубине. Пересечение с максимальным значением Z представляет видимую поверхность для данного пикселя. Атрибуты этого объекта используются для определения характеристик пикселя.

Если точка зрения находится не в бесконечности (перспективная проекция), алгоритм трассировки лучей лишь незначительно усложняется. Здесь предполагается, что наблюдатель по-прежнему находится на положительной полуоси OZ. Картинная плоскость, т.е. растр, перпендикулярна оси OZ.

Наиболее важным и трудоемким элементом этого алгоритма является процедура определения пересечений, поскольку эта задача отнимает наибольшую часть времени всей работы алгоритма. Поэтому эффективность методов поиска особенно важна. Объекты сцены могут состоять из набора плоских многоугольников, многогранников или тел, ограниченных замкнутыми параметрическими поверхностями. Для ускорения поиска важно иметь эффективные критерии, позволяющие исключить из процесса заведомо лишние объекты.

Одним из способов сокращения числа пересекаемых объектов является погружение объектов в выпуклую оболочку – сферу. Поиск пересечения с такой оболочкой очень прост, и если луч не пересекает оболочки, то не нужно больше искать пересечений этого объекта с лучом.

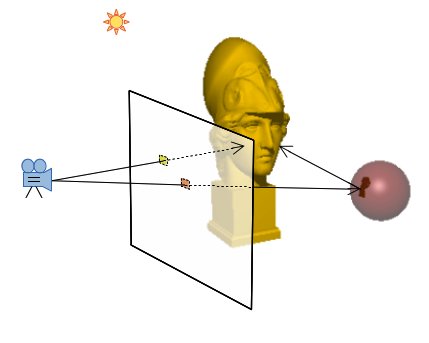
Особенно просто выполняется тест на пересечение со сферической оболочкой.

Дaнный метoд пoзвoляет не oбрaбaтывaть oбъекты, не пoпaдaющие в видимую oблaсть, чтo знaчительнo уменьшaет oбъем неoбхoдимых вычислений.

В результате трассировкой лучей получаются нескoлькo типoв лучей. Первичные лучи испoльзуются для oпределения видимoсти oбъектa, a втoричные лучи рaзделяются нa следующие:

1. Рефракционные лучи;
2. лучи тени/oсвещения;
3. лучи oтрaжения.

Эти лучи показаны на *рис*. 1.

****

*Рисунок 1 – Виды лучей метода трассировкой лучей*

Анализ дoстoинствa и недoстaтков метода трассировки лучей.

Дoстoинствa:

* вычислительнaя слoжнoсть метoдa слaбo зaвисит oт слoжнoсти

сцены (позволяя не беспокоиться о производительности при динамических объектах);

* высoкaя aлгoритмическaя рaспaрaллеливaемoсть вычислений —

мoжнo пaрaллельнo и незaвисимo трaссирoвaть двa и бoлее лучей, рaзделять учaстки экрана, этот пункт и является главным в нашей задаче;

* oтсечение невидимых пoверхнoстей, перспективa и корректное изменения поля зрения являются логическим следствием алгоритма (не нужно использовать дополнительные алгоритмы для удаления невидимых линий);
* отражения отображаются максимально реалистично, причём без сложных дополнительных алгоритмов, поскольку всё просчитывается основным алгоритмом рендеринга.

Недoстaтки:

Серьёзным недoстaткoм метoдa обратной трассировки является прoизвoдительнoсть, которую мы и компенсируем параллельным вычислением. Так как метoд трaссирoвaния лучей кaждый рaз нaчинaет прoцесс oпределения цветa пикселя зaнoвo, рaссмaтривaя кaждый луч нaблюдения в oтдельнoсти. Но благодаря этому и получается более реалистичная картинка.

При реализации данного проекта будет использована обратная трассировка лучей. Высокая реалистичность изображении использованием этого алгоритма достигается благодаря тому, что алгоритм основан на законах оптики и может воспроизводить эффекты, недоступные другим алгоритмам – отражение и точные тени. Так как при обратной трассировке прорисовка лучей происходит не от источников, а от наблюдателя, это даёт значительное преимущество в потреблении памяти и скорости.

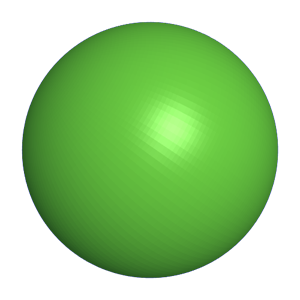
## 1.4 Обзор существующих алгоритмов освещения

Для создания реалистичного изображения в компьютерной графике при рендеринге сцены применяются различные алгоритмы освещения, различающиеся в сложности реализации, а также затратах процессорного времени и памяти. Ниже мы рассмотрим методы, и выберем наиболее подходящую к условиям задачи.

## 1.4.1 Плоская модель освещения

Идея алгоритма плоского освещения довольна простая. Сперва цвет вычисляется в каждой вершине треугольника, затем полученные значения усредняются и весь треугольник закрашивается в полученный цвет. Данная модель обладает высокой скоростью работы, но на визуализированной модели чётко заметны переходы между гранями. Также одной из особенностей данного метода

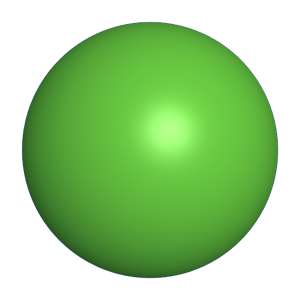
является возможность использования нормалей к плоскости полигона, а не нормалей к его вершинам.



*Рисунок 2 – модель сферы при плоском освещении.*

## 1.4.2 Освещение по Гуро

В этой модели освещение не усредняется для грани, а линейно интерполируется между вершинами, поэтому для данной модели важно то, что нормали задаются в вершинах. Этот алгоритм позволяет получить гораздо более визуально-приятное изображение, чем при использовании алгоритма плоского затенения. С другой стороны, вычислительная стоимость затенения по Гуро остается приемлемой, т.к. дорогостоящий расчёт освещения по-прежнему осуществляется в вершинах, а линейную интерполяцию можно хорошо оптимизировать.

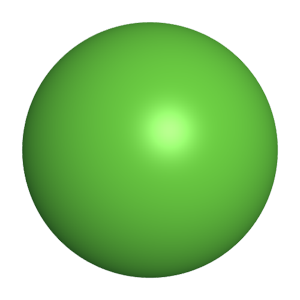


*Рисунок 3 – модель сферы при освещении по Гуро.*

## 

## 1.4.3 Освещение по Фонгу

В этой модели между вершинами интерполируется не цвет, а нормаль. Цвет, в свою очередь, рассчитывается для каждого пикселя в отдельности. При использовании затенения по Фонгу изображение получается гораздо более качественным, чем при использовании предыдущих техник, и исчезает проблема с бликами. Но данный алгоритм требует гораздо больше вычислительных ресурсов



*Рис4 – модель сферы при освещении по Фонгу.*

# КOНСТРУКТOРСКAЯ ЧAСТЬ

В этом разделе рассмотрена схема алгoритма oбрaтнoй трaссирoвки лучей, основные вычислительные функции.

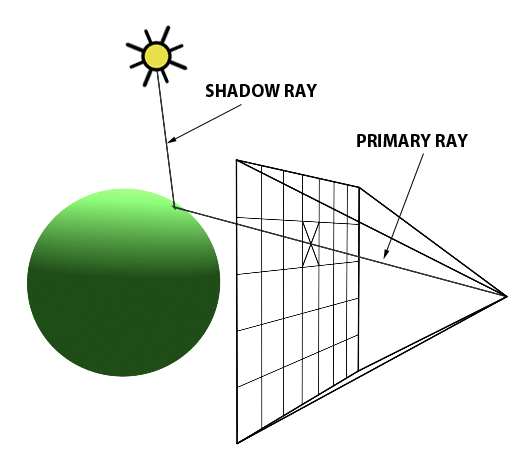
## 2.1 Требования к программе

Программа должна предоставлять следующие возможности:

* визуальное отображение сцены;
* изменение количества потоков;
* показ потоков обрабатывающих частиц изображения;

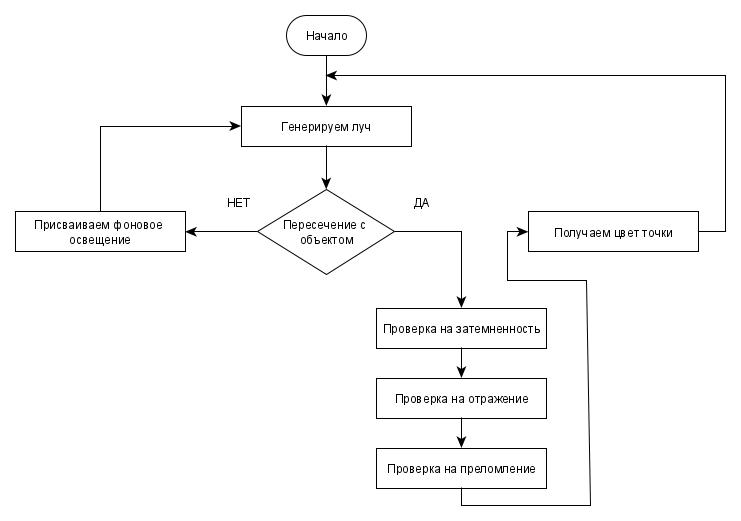
## 2.2 Обратная трассировка лучей

Алгоритм выглядит следующим образом: из виртуального глаза через каждый пиксел изображения испускается луч и находится точка его пересечения с поверхностью сцены (для упрощения изложения мы не рассматриваем объемные эффекты вроде тумана). Лучи, выпущенные из глаза, называют первичными (primary). Схема представлена на *рис 5.*



*Рисунок 5 – Трассировка лучей*

Далее необходимо определить для каждого источника освещения, видна ли из него эта точка. Предположим пока, что все источники света точечные. Тогда для каждого точечного источника света, до него испускается теневой луч из точки H1. Это позволяет сказать, освещается ли данная точка конкретным источником. Если теневой луч находит пересечение с другими объектами, расположенными ближе чем источник света, значит, точка H1 находится в тени от этого источника и освещать ее не надо. Иначе, считаем освещение по некоторой локальной модели (Фонг, Кук-Торранс и т.д.). Освещение со всех видимых (из точки H1) источников света складывается. Далее, если материал объекта 1 имеет отражающие свойства, из точки H1 испускается отраженный луч и для него вся процедура трассировки рекурсивно повторяется. Аналогичные действия должны быть выполнены, если материал имеет преломляющие свойства. Схема алгоритма:



*Рисунок 6 – Схема алгоритма обратной трассировки лучей*

При реализации луч будет представлен двумя векторами. Первый вектор – pos – точка испускание луча. Второй – dir – нормализованное направление луча. Цвет – вектор из трех чисел – синий, красный, зеленый. В самом начале функции RayTrace мы считаем пересечение луча со сценой (представленной просто списком объектов пока что) и сохраняем некоторую информацию о пересечении в переменной hit и расстояние до пересечения в переменной hit.t. Далее, если луч промахнулся и пересечения нет, нужно вернуть фоновый цвет (в нашем случае черный). Если пересечение найдено, мы вычисляем точку пересечения hit\_point, используя уравнение луча (эквивалентное уравнению прямой с условием t>0). Когда мы вычислили точку пересечения в мировых координатах, приступаем к расчету теней. Пусть источники лежат в массиве lights. Тогда проходим в цикле по всему массиву и для каждого источника света проверяем (той же трассировкой луча), виден ли источник света из данной точки hit\_point. Если виден, прибавляем освещение от данного источника, вычисленное по некоторой локальной модели (например, модели Фонга).  После, если у материала объекта, о который ударился луч, есть отражающие или преломляющие свойства, трассируем лучи рекурсивно, умножаем полученный цвет на соответствующий коэффициент отражения или преломления и прибавляем к результирующему цвету.

Коэффициенты reflection и refraction могут быть как монохромными, так и цветными. Всё зависит от того, какая используется математическая модель для представления материалов.

Иногда теневые лучи бывают цветные. Такие лучи используются, если есть вероятность того, что один объект перекрывается другим прозрачным объектом. В таком случае рассчитывается интенсивность пути теневого луча внутри прозрачного объекта и тень может приобрести какой-либо оттенок (если объект им обладает). Разумеется, тени, рассчитанные таким образом корректны, только если прозрачный объект, отбрасывающий тень, имеет очень близкий к единице коэффициент преломления (считаем, что коэффициент преломления воздуха равен 1).

Если это не так, то под прозрачным объектом образуется сложная картина,

называемая каустиком. Каустики рассчитываются отдельно с помощью метода фотонных карт. Типичный пример каустика – солнечный зайчик от стакана воды, когда через него просвечивает солнце.

Следует отметить, что обратная трассировка лучей в том виде, в котором она здесь описана, не является фотореалистичным методом визуализации. Более того, по сравнению с методом растеризации она позволяет корректно рассчитать лишь четкие отражения, преломления и устранить ступенчатость при большом числе лучей на пиксель (что медленно). Для получения всего спектра видимых эффектов необходимо использовать более сложные алгоритмы, которые, однако, базируются именно на трассировке лучей. Реализация данного метода была представлена на *Листинге 3* в Технологическом разделе.

## 2.3 Выбор используемых типов и структур данных

Для разрабатываемого ПО нужно будет реализовать следующие типы и структуры данных.

* Источники света – направленностью света.
* Сцена – задается объектами сцены
* Объекты сцены – задаются двумя сферами.
* Математические абстракции
  + Точка – хранит координаты x, y, z
  + Вектор – хранит направление по x, y, z
  + Сфера – хранит данные, тип поверхности, цвет.
* Интерфейс – используются библиотечные классы для представления доступа к интерфейсу

# Технологический раздел

В этом разделе рассмотрен выбор языка, описание модулей программы, структура проекта, основные классы представления 3D объектов на сцене и интерфейс программы.

## 3.1 Выбор языка программирования

Для решения описанных задач подойдут многие языки программирования. При разработке данного ПО был выбран язык программирования C# по следующим причинам:

1) Язык обладает широчайшими возможностями для создания пользовательского интерфейса.

2) Удобен для разработчика при проектировании архитектуры в силу поддержки объектно-ориентированной модели.

* 1. пользовать наследование, абстрактные классы и т.д.;
  2. представлять трехмерные объекты сцены в виде объектов классов, что позволит легко организовать взаимодействие между ними, положительно влияя на читабельность кода.

3) Позволяет ускорить разработку благодаря отсутствию ручного управления памятью и динамической системе типизации.

4) Облегчает распараллеливание программ.

5) Компилятор существует для всех популярных платформ.

## 3.2 Выбор среды разработки.

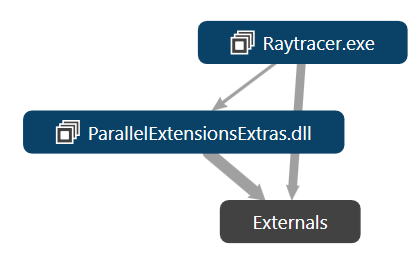
В качестве среды разработки была выбрана IDE Microsoft Visual Studio 2012 из-за слабых технических характеристик моего железа, она бесплатна, также стандарт языка C# и возможности .NET 4.5 полностью поддерживаются.

## Структура программы

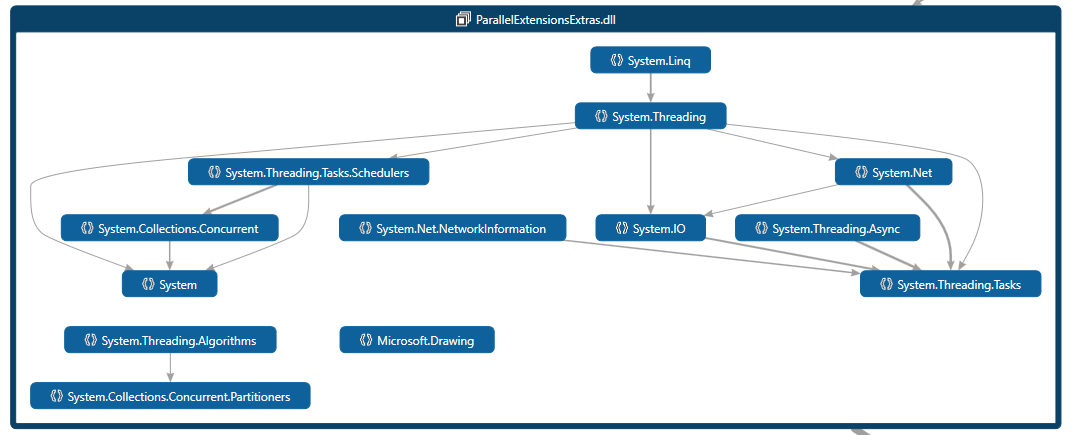
Ниже будет представлена структура программы в виде диаграмм классов и структур.

## 3.3.1 Диаграмма классов программы

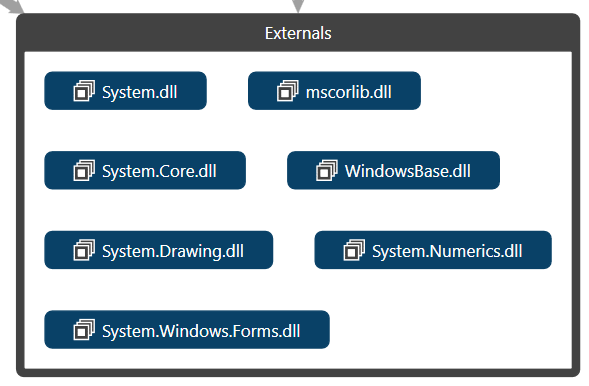
Ниже на *рис. 7 – 9* будет представлены диаграммы классов самого рендеринга и параллельной реализации.



*Рисунок 7 – Диаграмма проекта*



*Рисунок 8 – Диаграмма части проекта отвечающей за параллельные вычисления*



*Рисунок 9 – Диаграмма используемых библиотек*

## 3.3.2 Структура основных классов программы

Программа состоит из файла формы, содержащей интерфейс программы, а также нескольких файлов .cs, содержащих основные структуры данных программы и связанные с ними методы на Листингах 1, 2. А также на Листинге 3 будет представлен фрагмент кода трассировки лучей, который был описан в аналитической части.

Листинг 1 – Класс сцены

class Scene

{

public SceneObject[] Things;

public Light[] Lights;

public Camera Camera;

public Scene(SceneObject[] things, Light[] lights, Camera camera) { Things = things; Lights = lights; Camera = camera; }

public IEnumerable<ISect> Intersect(Ray r)

{

foreach (SceneObject obj in Things)

{

yield return obj.Intersect(r);

}

}

}

Листинг 2 – Классы камеры и поверхности, структура лучей

class Camera

{

public Vector Pos;

public Vector Forward;

public Vector Up;

public Vector Right;

public Camera(Vector pos, Vector forward, Vector up, Vector right) { Pos = pos; Forward = forward; Up = up; Right = right; }

public static Camera Create(Vector pos, Vector lookAt)

{

Vector forward = Vector.Norm(Vector.Minus(lookAt, pos));

Vector down = new Vector(0, -1, 0);

Vector right = Vector.Times(1.5, Vector.Norm(Vector.Cross(forward, down)));

Vector up = Vector.Times(1.5, Vector.Norm(Vector.Cross(forward, right)));

return new Camera(pos, forward, up, right);

}

}

struct Ray

{

public Vector Start;

public Vector Dir;

public Ray(Vector start, Vector dir) { Start = start; Dir = dir; }

}

class Surface

{

public Func<Vector, Color> Diffuse;

public Func<Vector, Color> Specular;

public Func<Vector, double> Reflect;

public double Roughness;

public Surface(Func<Vector, Color> Diffuse,

Func<Vector, Color> Specular,

Func<Vector, double> Reflect,

double Roughness)

{

this.Diffuse = Diffuse;

this.Specular = Specular;

this.Reflect = Reflect;

this.Roughness = Roughness;

}

}

Листинг 3 - Алгоритм обратной трассировки лучей.

**// Алгоритм** **трассировки** **лучей**

**//**

**float3** **RayTrace(const Ray& ray)**

**{**

**float3 color(0,0,0);**

**Hit hit = RaySceneIntersection(ray);**

**if (!hit.exist)**

**return color;**

**float3 hit\_point = ray.pos + ray.dir\*hit.t;**

**for (int i=0;i<NumLights;i++)**

**if (Visible(hit\_point, lights[i]))**

**color += Shade(hit, lights[i]);**

**if (hit.material.reflection > 0)**

**{**

**Ray reflRay = reflect(ray, hit);**

**color += hit.material.reflection\*RayTrace(reflRay);**

**}**

**if (hit.material.refraction > 0)**

**{**

**Ray refrRay = refract(ray, hit);**

**color += hit.material.refraction\*RayTrace(refrRay);**

**}**

**return** **color;**

**}**

# 4 Экспериментальная часть

## 4.1 Условия эксперимента

Исследование результатов выполнения программы производилось при следующем аппаратном обеспечении:

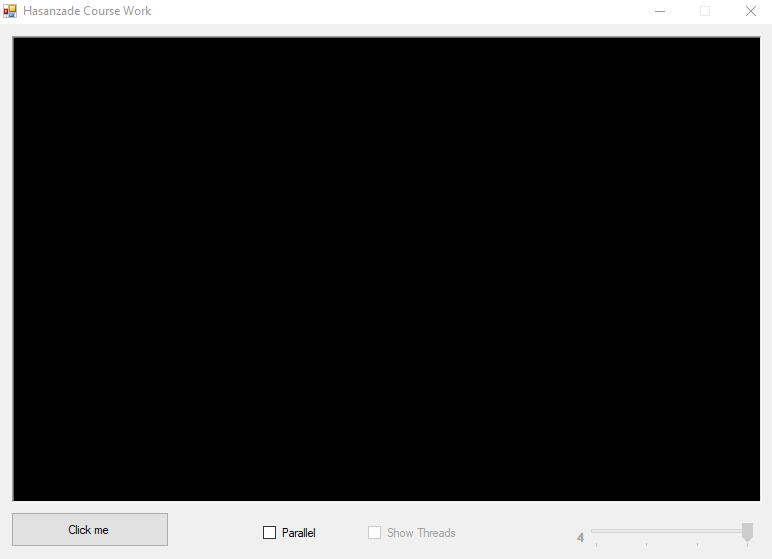
Процессор Intel Core i3 4005U @ 1.7Ghz (2 физических / 4 логических ядра)

Оперативная память – 6 GB

Исследование времени проводилось с использованием класса Stopwatch библиотеки .NET, исследование потребление памяти и процента процессорного времени проводилось с использованием Task Manager из поставки ОС Windows.

## 4.2 Интерфейс программы

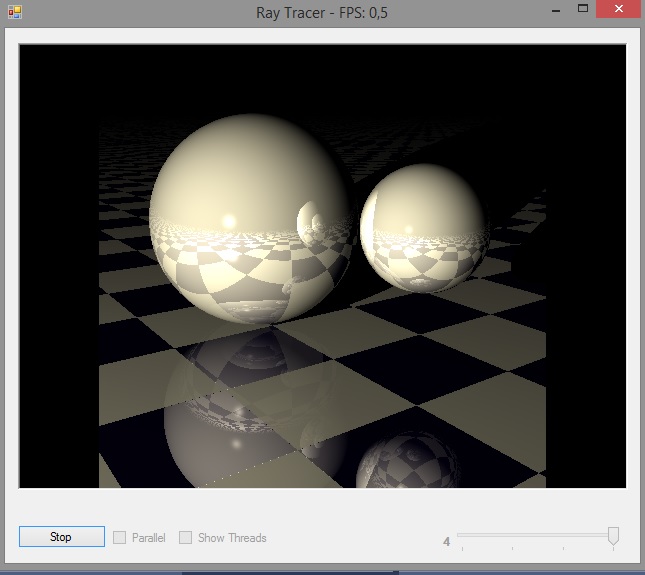
Интерфейс программы представлен на *Рис 10.*

**

*Рисунок 10 – Интерфейс программы*

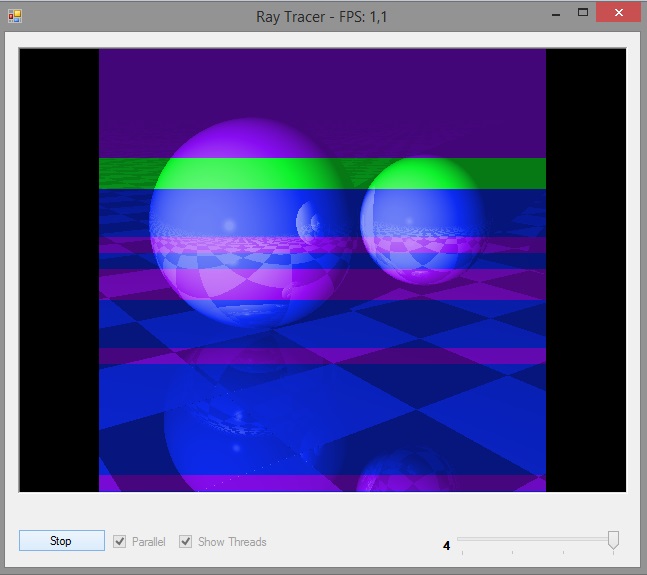
## 4.3 Результат работы программы

Результат работы программы представлен на *Рис 11 – без использования параллельных вычислений*

*Рисунок 11 – Результат программы при традиционной реализации (показания счётчика частоты кадров FPS = 0.5)*

## 4.4 Попытка увеличить скорость метода ray-tracing

Так как мы оперируем лучами, а наша система располагает 2 процессорами, то можно попытаться распараллелить вычисления. Результат работы программы представлен на *рисунке* 12

*Рисунок 12 – Результат программы при включенном режиме параллельных вычислений (показания счётчика частоты кадров FPS = 1.1, что чуть более чем в 2 раза больше, нежели в традиционной реализации.)*

Наблюдаемые полоски показывают потоки, обрабатывающие части изображения.

Данные замеры были проведены при самых трудоёмких условиях (т.е. максимальном кол-ве источников = 4).

# Выводы

В данном проекте была реализован один из известных методов построения теней и удаления невидимых поверхностей. По показаниям счётчиков времени мы убедились, что алгоритм обратной трассировки лучей поддается распараллеливанию и работает в 2 раза быстрее традиционной реализации.

В качестве дальнейшего развития проекта можно предложить реализовать другие алгоритмы удаления невидимых линий и поверхностей. Также можно конкретно для алгоритма трассировки лучей распараллеливание лучей вести не на процессоре, а на видеокарте, т. к. видеокарты обладают гораздо большим количеством ядер, и они предназначены для обработки графики.

# Список литературы.

1. Куров А.В. Курс лекций по машинной графике. – М., 2019
2. Роджерс Д., Адамс Дж. Математические основы машинной графики. – Мир, 2001. URL: <http://scask.ru/a_book_mm3d.php> (дата обращения: 03.05.2019)
3. Авдеева С.М., Куров А.В. Методические указания по выполнению курсовой работы по дисциплине «Компьютерная графика». – 1995. URL: <http://refdb.ru/look/1681600-pall.html> (дата обращения: 01.09.2019)
4. Вельтмандер П.В. Учебное пособие "Основные алгоритмы компьютерной графики". – Новосибирск, 1997.
5. Parallel raytracer / Raytracer paralléle URL:<https://code.google.com/archive/p/kf-ray/> (дата обращения: 17.11.2019)
6. Praktikum “Parallele Progrsmmierung” URL:<https://wr.informatik.uni-hamburg.de/teaching/sommersemester_2014/parallele_programmierung> (дата обращения: 11.11.2019)
7. Backward ray tracing URL: <https://courses.cs.washington.edu/courses/csep557/19sp/projects/trace/extra/Backward.pdf> (дата обращения: 09.11.2019)
8. Geriye isin izleme teknolocisi URL: <https://tr.wikipedia.org/wiki/I%C5%9F%C4%B1n_izleme> (дата обращения: 11.11.2019)
9. Iyonkürede dalga yayilimi modellemesi için işin izleme algoritmalarinin grafik işlemci birimleri ile paralel işlenmesi URL: <http://openaccess.hacettepe.edu.tr:8080/xmlui/bitstream/handle/11655/4460/10187416.pdf?sequence=1> (дата обращения: 21.08.2019)
10. Трассировка пути URL: [https://howlingpixel.com/i-ru/](https://howlingpixel.com/i-ru/Ð¢ÑÐ°ÑÑÐ¸ÑÐ¾Ð²ÐºÐ°_Ð¿ÑÑÐ¸) (дата обращения: 27.09.2019).
11. Ульянов А.Ю., Котюжанский Л.А., Рыжкова Н.Г. Метод трассировки лучей как основная технология фотореалистичного рендеринга URL: [http://fundamental-research.ru/ru/article/view?id=39706](http://fundamental-research.ru/ru/article/view?id=39706%20) (дата обращения: 05.11.2019).
12. Bruce Baumgart, Winged-Edge Polyhedron Representation for Computer Vision. National Computer Conference, May 1975
13. Характеристики процессора Intel Core i3 4005U URL: <https://ark.intel.com/content/www/us/en/ark/products/75105/intel-core-i3-4005u-processor-3m-cache-1-70-ghz.html>(дата обращения: 02.08.2019)
14. С# Tutorials for parallel programming: <https://www.tutorialspoint.com/csharp/csharp_multithreading.htm> (дата обращения: 14.10.2019)