Московский государственный технический университет имени Н.Э.Баумана



*Факультет «Информатика и системы управления»*

*Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»*

**Курсовая работа по курсу: «Машинная графика»**

**Ускорение обратной трассировки лучей с помощью алгоритма bvh-дерева.**

*Студент: Могильников А.Б.*

*Группа: ИУ7-52*

*Научный руководитель: Романова Т.Н*

Москва 2012 г.

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Основание для разработки:

Задание на курсовую работу по курсу«Машинная графика»

***Тема курсовой работы:***

Ускорение обратной трассировки лучей с помощью алгоритма bvh - дерева

***Техническое задание:***

“Разработать программу, отрисовывающую сцену, состоящую из четырех стен, пола и потолка, образующих комнату. В данной комнате находятся стол, зеркало, на столе стоят куб, пирамида, сфера. Положение объектов сцены может быть редактировано. На сцене может находиться от одного до 3 точечных источников света. Интенсивность и цвет источников нельзя изменить, но можно изменить их положение. В пределах сцены можно задавать положение точки наблюдения и вектор-направление обзора. Интерфейс должен обеспечить ввод координат точки наблюдения, положения и свойства объектов сцены”

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ 1

1. ВВЕДЕНИЕ 5

2. КОНСТРУКТОРСКАЯ ЧАСТЬ 9

2.1. Выбор и обоснование используемых алгоритмов 9

2.2. Описание алгоритма 7

2.3. Математические предпосылки алгоритма 9

2.4. Модель освещения 9

2.5. Деревья ограничивающих объемов 10

2.5.1. Иерархии ограничивающих объемов 10

2.5.2. Функция стоимости прослеживания 12

2.6. Устранение ступенчатости 12

2.7. Текстуры. UV-развертка. 13

2.8. Оптимизация построения мягких теней 14

2.9. Математические абстракции 15

3. ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ 17

3.1. Выбор и обоснование языка и среды программирования 17

3.2. Структура классов программы 18

3.2.1. Источники света 18

3.2.2. Узел дерева BVH 19

3.2.3. Камера 19

3.2.4. Материалы 20

3.2.5. Текстуры 20

3.2.6. Полигоны 20

3.2.7. Загрузка моделей 21

3.2.8. Классы интерфейса 22

3.3. Пользовательский интерфейс 23

3.3.1. Главное меню 23

3.3.2. Основное диалоговое окно 23

3.3.3. Настройки источников света 23

3.3.4. Настройки камеры 23

3.3.5. Настройки качества 25

3.4. Формат исходных данных 25

4. ЭКСПЕРИМЕНТАЛЬНО-ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ 27

4.1. Описание экспериментов 27

4.2. Результаты экспериментов 28

4.3. Интерпретация результатов экспериментов 28

5. ЗАКЛЮЧЕНИЕ 30

6. СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 31

# **ВВЕДЕНИЕ**

В настоящее время вопросы, связанные с отображением на экране дисплея разнообразных изображений, как никогда актуальны. Графика используется практически во всех областях деятельности человека, так или иначе связанных с использованием компьютера. Графическое представление информации может быть как всего лишь более удобным средством представления информации и организации взаимодействия пользователя с вычислительной машиной, так и неотъемлемой частью вычислительного комплекса, например, при моделировании сложных процессов, конструкторской деятельности, подготовке мультимедийных презентаций, кинематографии и трехмерных компьютерных играх.

До недавнего времени основным критерием выбора способа отображения трехмерных объектом являлась скорость вычислений, в силу того, что мощности компьютеров не хватало для полноценной реализации существующих алгоритмов. Примером таких «быстрых» алгоритмов являются так называемые scanline-алгоритмы, использующие построчный просчет видимого изображения; эти методы давали вполне приемлемую скорость и до сих пор используются в ряде случаев, когда на первом месте стоит быстродействие. Однако scanline-алгоритмы имеют один существенный недостаток, который в ряде случаев сводит на нет их преимущества: с их помощью невозможно построить высоко реалистичные изображения с отображением таких оптических эффектов, как отражение, преломление, блики света от линз и т.д. Однако сейчас вычислительные мощности систем возрастают с всей большей скоростью, поэтому теперь актуально отдавать предпочтение высокой реалистичности изображения, чем скорости построения. Для решения этой задачи отлично подходят рекурсивные алгоритмы изображения.

Задание данной курсовой работы - построение реалистичных изображений с реализацией мягких теней.

Хотя в настоящее время все большее внимание уделяется именно качеству получаемого изображения, однако скорость работы приложения машинной графики все равно остается важным параметром. Именно поэтому основной целью данной работы является создание оптимизированного приложения, которое способно создавать высоко реалистичные изображения с невысокими вычислительными и временными затратами.

Данная программа предназначена для построения реалистических изображений трехмерных объектов. Результатом работы программы должно быть изображение высокого качества, включающее в себя поддержку отражения лучей, учет неточечных источников света, реализация таких оптических эффектов, как блики на поверхности, сглаживание. Для оптимизации программы реализованы алгоритмы, уменьшающие время построения. Среди этих алгоритмов присутствуют как уже существующие, так и собственные разработки.

Сцена, подлежащая построению, строится на основе полигонов (треугольников) и описывается в файлах особого формата, которые можно загружать во время работы программы. Пользователь имеет возможность добавлять или удалять источники света, меняя их расположение, а также задавать положение и ориентацию камеры и менять размер получаемого изображения.

Кол-во используемых в одной сцене объектов, а также минимальное и максимальное число источников света фиксированы.

# 2. КОНСТРУКТОРСКАЯ ЧАСТЬ

## 2. 1. Выбор и обоснование используемых алгоритмов

Т.к. лучше всего с созданием качественного изображения справляются рекурсивные алгоритмы, то выбор происходил из их числа.

Для получения реалистичных изображений в настоящее время существует следующие алгоритмы:

- Прямая трассировка лучей

- Обратная трассировка лучей

- Двунаправленная трассировка лучей

- Трассировка путей

- Распределенная трассировка лучей

- Метод Фотонных карт

- Алгоритм излучательности

Ниже кратко описаны основные из них:

Сам принцип построения изображения с помощью трассировки лучей разработан довольно давно и был представлен в виде метода прямой трассировки лучей.

Но, как и в момент его создания, в настоящее время алгоритм прямой трассировки применяется очень мало в силу исключительно больших вычислительных затрат и существенной избыточности вычислений. Кроме того, очень трудно предсказать время вычисления, которое в значительной степени случайно. Однако алгоритм все же используется, поскольку качество и реалистичность получаемого изображения очень высоко.

Фотонные карты – один из самых универсальных и реалистичных алгоритмов, решающих задачу вычисления интеграла освещенности в самом общем случае. Фотонные карты сложны как в реализации, так и в вычислительном плане. Алгоритм состоит из трех частей: трассировка фотонов, построение фотонной карты и сбор освещенности

Алгоритм обратной трассировки лучей в той или иной модификации является основным средством получения высоко реалистичных изображений с большинством возможных оптических эффектов, наблюдаемых в реальной жизни. Хотя методы расчета, применяемые в ходе работы алгоритма, в значительной степени искусственны, то есть получены эмпирически и не имеют ничего общего с реальными законами оптики, тем не менее, результат оказывается очень близким к реальности.

И, наконец, существует еще алгоритм распределенной трассировки лучей.

Чтобы получить такие эффекты как мягкие тени и визуализировать материалы из реальной жизни, обладающие нетривиальными свойствами, необходимо использовать более сложную модель, чем обратная трассировка лучей.

В реальности источники света часто не точечные (имеют размер), а материалы не всегда отражают всю энергию в одном направлении по закону “угол падения равен углу отражения”. Такое свойство может быть связано с микрорельефом поверхности или особыми свойствами краски и.т.д. Именно наличие не точечных источников света дает возможность реализовать мягкие тени.

Основным алгоритмом был выбран алгоритм Обратной Трассировки Лучей (далее ОТЛ) с надстройкой в виде алгоритма Распределенной Трассировки Лучей для мягких теней(далее РТЛ). Это объясняется тем, что ОТЛ работает быстрее РПЛ, однако частичная доработка алгоритма ОТЛ до алгоритма РПД позволить реализовать мягкие тени, что в разы повысит реалистичность изображения. При этом сложность и время вычислений повысится, однако не так сильно, как если бы основным алгоритмом был выбран полноценный алгоритм Распределенной Трассировки Лучей.

Стоит выделить основные преимущества алгоритма:

- Высокая реалистичность получаемой при рендере картины, т.к. данный алгоритм наиболее сильно завязан на физических законах оптики (в отличие от ОПЛ, РТЛ учитывает реальные источники света взамен точечных).

- Высокая гибкость алгоритма, возможность работы с объектами любых форм.

- Невысокая потребность в памяти, потому что при построении изображения интенсивности всех пикселов вычисляются автономно, и память расходуется только на хранение текущего дерева луча.

Также необходимо выделить недостатки алгоритма РПЛ:

- Низкая скорость работы алгоритма.

- Сложность реализации алгоритма.

Блок-схема алгоритма представлена в приложении (блок-схемы №1 и №2).

2.2. Описание алгоритма

**Общие принципы.** Суть алгоритма «Обратной Трассировки Лучей с надстройкой в виде алгоритма Распределенной Трассировки Лучей для мягких теней» заключается в том, что из камеры через каждый пиксель видимой части проекционной плоскости проводится луч, который трассируется либо до пересечения с каким-либо объектом сцены, либо до тех пор, пока не покинет пределы сцены. Если луч покидает сцену, не пересекая ни один из объектов, то считается, что по этому лучу освещенность в точку проекционной плоскости не приходит, и ей в общем случае сопоставляется нулевой (черный) цвет. Если же луч пересек какой-либо объект сцены, то в точке пересечения считается освещенность с учетом свойств поверхности и направления нормали к ней, а также направления, по которому пришел в точку пересечения луч и направлений от точки пересечения до источников света. Затем при необходимости строится отраженный луч, который также трассируются, и после этого освещенность, пришедшая по его направлению, добавляется к освещенности текущей точки. Вторичные лучи также могут пересекаться с объектами сцены и порождать новые отраженные лучи, и так далее. Глубина трассировки определяется конкретным значением.

Освещенность в алгоритме трассировки лучей рассматривается как состоящая из двух частей: первичной освещенности, то есть энергии, получаемой непосредственно от источников света, и вторичной освещенности, то есть энергии, приходящей по отраженному лучу от других тел сцены. Важно заметить, что первичную освещенность можно определить сразу же после нахождения пересечения луча и объекта, а для вычисления вторичной освещенности нужно будет выполнить трассировку еще несколько лучей, то есть она становится известной спустя какое-то количество шагов рекурсии. Т.к. источники света не точечные, то для вычисления первичной освещенности текущей точки нужно до каждого источника света послать N лучей в случайные точки источника. Чем больше лучей будет послано, тем больше вероятность правильно вычислить освещенность точки.

2.3. Математические и физические предпосылки алгоритма.

## 2.4. Модель освещения.

Для расчета интенсивности каждого пиксела используются модели освещения Фонга и Ламберта.

// дописать

## 2.5. Деревья ограничивающих объемов. BVH. SAH

Для больших сцен поиск объекта, с которым пересекается луч методом перебора неприемлем из-за своих больших затрат. Так как в современной компьютерной графике ни одна сцена не обходится без большого числа объектов, возникает необходимость в методах с сублинейной сложностью (от общего количества объектов). Стандартным приемом, позволяющим заметно снизить сложность поиска нужного объекта в пространстве, являются различные типы так называемых пространственных индексов (*spatial index*).

Пространственный индекс - это некоторая структура данных (чаще всего иерархическая), строящаяся обычно на этапе подготовки сцены. [6]

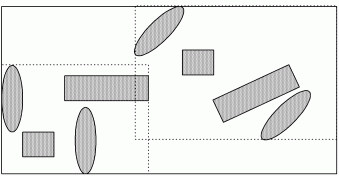
### *2.4.1.* Иерархии ограничивающих объемов (Bounding Volume Hierarchies)

Одним из вариантов иерархического пространственного индекса являются так называемые иерархии ограничивающих объемов.

Обычно BVH деревья используются для расчёта столкновений в физических экспериментах. Однако в последнее время BVH активно стараются задействовать в рейтрейсинге в связи с тем, что в анимированных сценах BVH можно быстрее перестраивать, и как правило можно перестраивать не все дерево.

При построении BVH вокруг исходной группы объектов описывается некоторое ограничивающее тело. После этого группа разбивается на несколько подгрупп, вокруг каждой из которых описывается свое ограничивающее тело. Далее каждая из них снова разбивается на подгруппы и так далее.

В результате получается дерево, где с каждым узлом связано некоторое ограничивающее тело и набор дочерних узлов.



Очевидно, что разновидностей BVH деревьев можно придумать сколько угодно. Достаточно взять некую произвольную фигуру и использовать ее в качестве ограничивающего объема. Исторически выделяют 5 типов BVH деревьев:

1. Sphere tree
2. AABB tree (Axis Aligned Bounding Box)
3. OBB tree (Oriented Bounding Box)
4. k-DOP (Discrete Oriented Polytope)
5. SSV (Swept Sphere Volume)

В данной работе была использована бинарная форма дерева AABB tree.

Алгоритм построения оптимального AABB довольно сложен, потому что вариантов возможных разбиений 2 в степени N. Перебрать их все не представляется возможным. BVH на каждом уровне может произвести разбиение на 2 совершенно произвольных подмножества примитивов. Всего таких подмножеств 2 в степени N.

Однако можно использовать упрощение, не сильно ухудшающее качество дерева, но позволяющее значительно упростить процедуру построения дерева. Фактически, с помощью этой процедуры можно свести построение дерева к сортировке**.** Поэтому на практике BVH строить во много раз проще:

Строится дерево  сверху вниз. На каждом уровне  BVH дерева сортируются все примитивы по максимуму ограничивающих боксов по осям X, Y и Z (на самом деле можно выбрать и другой признак). Таким образом получается 3 массива. Первый отсортирован по X-ам, второй по Y-кам и третий по Z. Затем для каждого массива  перебираются все разбиения на подмножества следующего вида:

**[0, 1..n]**

**[0..1, 2..n]**

**[0..3, 4..n]**

**...**

**[0..n-1, n]**

[7]

Для каждого из этих разбиений считается SAH. Запоминая каждый раз ограничивающие AABB в специальном массиве для получаемых подмножеств (**rightBounds**) мы можем не пересчитывать эти AABB полностью, что сводит поиск всего к 2 проходам по массиву. С конца к началу и с начала к концу.

Так как каждый массив заранее отсортирован по соответствущей оси, то выбранные таким образом разбиения позволят минимизировать самопересечения и дадут более-менее хороший SAH.

### *2.4.2.* Функция стоимости прослеживания (SAH)

Для формализации критериев построение правильных деревьев вводится функция стоимости прослеживания, которая будет отражать, насколько дорого по вычислительным ресурсам прослеживать данный узел случайным набором лучей. Вычислятся она по следующей формуле:

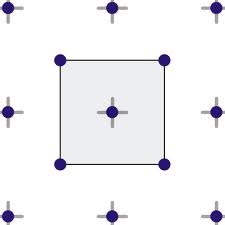
**SAH(x) = CostEmpty + SurfaceArea(Left)\*N(Left) + SurfaceArea(Right)\*N(Right)**

Где  CostEmpty -  стоимость прослеживания пустого узла (некоторая константа), SurfaceArea - площадь поверхности соответствующего узла, а N - число примитивов в нем. Нужно выбирать плоскость разбиениея так, чтобы минимизировать эту функцию. Аргумент функции x является одномерной координатой плоскости разбиения.

Хорошими кандидатами на минимум SAH могут служить границы примитивов.

## 2.5. Устранение ступенчатости (Antialiasing).

Для большей реалистичности изображения было решено использовать алгоритм устранения ступенчатости, который решает следующую проблему.

Сцена состоит из примитивов, которые являются непрерывными объектами, и лишь при растеризации превращаются в пикселы. Этот процесс называется *сэмплингом*, и ему присущ артефакт под названием *алиасинг*. При сэмплинге цвет пиксела определяется цветом полигона, которому принадлежит центр пиксела. Для решения проблемы алиасинга предназначен *антиалиасинг*.

*Суперсэмплинг* - наиболее популярный метод антиалиасинга. название пришло из области обработки сигналов и означает повышение частоты дискретизации для получения более качественных результатов). Каждый пиксел состоит из *субпикселов*, для каждого вычисляется цвет, а затем усреднением получается цвет пиксела. Конкретное представление пиксела называется *субпиксельной маской* (к примеру 2x2, 4x4). Чем больше субпикселов отводится на каждый пиксел, тем качественнее получается антиалиасинг. Маска 4x4 считается оптимальной с точки зрения качества, но при этом скорость заполнения снижается в 16 раз. С другой стороны, более простая маска обеспечивает более высокую скорость, но и менее качественное изображение.

Также стоит отметить, что в данной работе устранение ступенчатости благоприятно влияет и на построение мягких теней – при суперсэмплинге автоматически сглаживаются шумы, получающиеся при вычислении мягких теней.

## 2.8. Описание используемой структуры данных

В разработанной программе сцена состоит из объектов, аппроксимируемых полигонами.

Способ задачи объектов сцены с помощью полигональных примитивов была выбрана осознанно и аргументирована. Т.к. одним из основных условий качественно сделанной работы является реалистичность получаемого изображения, то очень важным является возможность задачи объектов абсолютно любых форм. При создании объектов сцены с помощью примитивов, заданных точно (на основе уравнений кривых и поверхностей), достичь высокой реалистичности очень трудно. Реализованные в данной программе алгоритм оптимизации построения мягких теней и дерево ограничивающих объемов позволяют достичь достаточной аппроксимации объектов сцены полигонами для создания реалистичного изображения, не уступающего объектам, рассчитанным точно (такие, как сфера, конус и т.п.).

Поэтому структура данной программы содержит только класс полигон, имеющий свои параметры и методы.

Полигон задается тремя вершинами (A, B, C) в пространстве, цветом (RGB модель) или соответствующей полигону текстурой. На основе вершин полигона вычисляется его нормаль.

**Хранение полигонов**

Свойства полигонов хранятся в классе, включающем в себя такие параметры, как коэффициенты фонового, диффузного и зеркального отражений. Полигоны содержатся в дереве ограничивающих объемов, а именно в узлах дерева, представленных соответствующим классом.

**Хранение источников света.**

В данной программе реализованы не точечные источники света. Формой источников света являются панели, реализованные с помощью двух полигонов.

Источник света задается положением (точка), двумя направляющими векторами (задающими положение световой панели) и цветом.

### 2.9. Математические абстракции

Математические абстракции в программе представлены тремя классами:

1. **Vector3f** – трехмерный вектор и операции над ним. Реализованы операции умножения на вектор, векторная сумма и разность, умножение и деление на число, нахождение скалярного и векторного произведения, нормализации, определение длины. Некоторые операции продублированы операторами для удобства записи, а также введены операторы действий, совмещенных с присваиванием.
2. **Ray** – трехмерный луч, задающийся двумя трехмерными векторами: точкой испускания и направлением.
3. **BvhNode** – вершина бинарного дерева, в качестве полей содержит указатели на двух своих потомков.

# 3. ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

## 3.1. Выбор и обоснование языка, ОС и среды программирования

Операционная система, под которую была бы написана программа курсовой работы, выбиралась между Linux и Windows. В итоге победила дружба.

Выбор языка и среды программирования осуществлялся по следующим параметрам:

* Язык среды программирования должен поддерживать большинство возможностей объектно-ориентированного программирования.
* Кроссплатформенной среды разработки
* Удобный интерфейс среды программирования

**C# сосет, плюсы рулят**

Основной выбор происходил между языками C# и C++.

Для разработки данной программы был выбран язык C# по следующим причинам:

* Этот язык предоставляет программисту широкие возможности реализации самых разнообразных алгоритмов, обладая высокой эффективностью и значительным набором стандартных классов и процедур.
* С# является одним из языков, в полной мере реализующих концепции объектно-ориентированного программирования, позволяющие использовать множественное наследование, абстрактные и параметризованные классы и т.д.. Следует отметить такое немаловажное свойство объектно-ориентированного подхода, как повторное использование кода.

В качестве среды разработки была выбрана Microsoft Visual Studio 2010:

* При разработке программы использовалась библиотека классов “.NET”, поставляемая вместе со средой разработки, которая позволяет существенно упростить разработку интерфейса программы по сравнению с использованием стандартных функций Windows API.
* Данная среда обладает удобным редактором кода и отладчиком, предоставляющими большое количество функций, необходимых для эффективного написания исходного кода и локализации и устранения ошибок в программе.
* Также данная среда обладает встроенным профилировщиком, дающим большие возможности для профилирования и оптимизации кода.

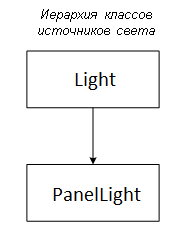
## 3.2. Структура классов программы

В силу того, что при написании программы использовалась технология объектно-ориентированного программирования, особое внимание при рассмотрении ее структуры должно быть уделено структуре ее классов, а именно основным классам:

* PanelLight
* BvhNode
* Camera
* Material
* Texture
* Triangle
* ObjMeshLoader
* Классы GUI

### 3.2.1. PanelLight

Источники света являются одними из основных понятий, участвующих в построении трехмерного изображения. В программе реализован точечный источник света, который имеет вид прямоугольной панели, однако для возможности добавления новых типов источников света и большей гибкости программы был выделен класс-предок Light, от которого наследуется класс точечного источника света PanelLight.

На рисунке изображена соответствующая диаграмма классов:

Члены:

\_position - позиция источника

\_width - ширина панели

\_height - высота панели

\_normal - нормаль к панели

\_vec\_1 - один из направляющих векторов, поределяющих положение световой панели в пространстве

\_vec\_2 - один из направляющих векторов, поределяющих положение световой панели в пространстве

\_vector\_dot - край панели (точка)

\_panel1 - один из полигнов, задающих панель

\_panel2 - один из полигнов, задающих панель

\_rgbColor – цвет источника

Функции члены:

Precalculate() – рассчет значений членов класса

Get\_random\_dot() – получает случайную точку на источнике

### BvhNode

Данный класс реализует узел дерева ограничивающих объемов. Он содержит как поля, отоносящие данный класс к математической абстракции (ссылки на правого и левого потомка узла), так и специальные поля, требующиеся для реализации алгоритма.

Члены:

\_LeftNode – ссылка на левый лист

\_RightNode – ссылка на правый лист

\_AABB - ограничивающий объем дерева

\_TriangleList – список полигонов, содержащихся в данном ограничивающем объеме

### 3.2.3. Camera

С помощью класса Camera реализован «виртуальный глаз», из которого в пространство испускаются лучи. Основой класса являются поля \_position и \_target, задающие позицию камеры в пространстве и направление ее обзора соответственно. Эти поля являются векторами и реализованы с помощью класса Vector3f. Также возможно изменение размеров получаемого изображения с помощью полей \_widthPx и \_heightPx, задающих размеры окна камеры.

Члены:

\_widthPx – ширина камеры

\_heightPx – высота камеры

\_position - позиция камеры

\_target – направление камеры

Функции члены:

Precalculate – рассчет значений членов класса

Direction – рассчет направления луча, пущенного из камеры через определенный пиксель

### 3.2.4. Material

Материал — это комбинация физических свойств модели: какой её основной цвет, как она поглощает, преломляет, пропускает, рассеивает или отражает свет, какова фактура поверхности и так далее. В данной программе реализованы следующие материалы:

\_METAL – имитация металлических материалов

\_WOOD – имитация деревянных материалов

\_MIRROR – имитация полностью отражающего зеркала

\_PLASTIC – имитация материалов из резины

Члены:

\_name – название материала

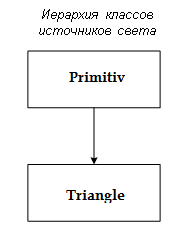
\_reflect - коэффициент отражения

\_diffuse - коэффициент диффузности

\_hotspot\_coef - коэффициент блика

\_hotspot\_pow - степень блика

### 3.2.6. Triangle

Данный класс включает в себя 3 вершины, задающих полигон в пространстве, нормаль к данному полигону, текстуру, реализованную с помощью класса Texture, задающую цвет полигону (для замены текстуры при ее отсутствии есть фоновый цвет). Методами класса являются вычислительные алгоритмы, такие как вычисление пересечения луча с полигоном и вычисление ограничивающего объема данного полигона. Также присутствует метод визуализации, вычисляющий цвет в данной точке. Для возможности добавления новых типов примитивов и большей гибкости программы был выделен класс-предок Primitiv, от которого наследуется класс полигона Triangle.

Члены:

\_a - вершина полигона

\_b - вершина полигона

\_c - вершина полигона

\_n - нормаль

have\_texture – флаг наличия текстуры

\_texture – текстура данного полигона

rgbColor – цвет данного полигона

\_bbox – ограничивающий объем данного полигона

Material – материал данного полигона

Функции члены:

NormalBy3Dot – вычисление нормали полигона по трем точкам

CalcAABBox – рассчитать ограничивающий объем полигона

### 3.2.7. ObjMeshLoader

Загрузка объектов сцены осуществляется с помощью класса ObjMeshLoader, содержащего помимо собственно процедур чтения файла и разбора его строк динамические списки всех объектов, которые могут присутствовать в сцене.

**Схемы включения, использования и наследования классов**

Object

BvhNode

Light

Triangle

PLight

Material

Texture

Условные обозначения между классами:

- включение, использование

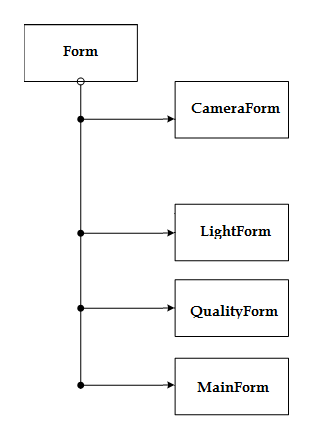
- наследование

(стрелка указывает на производный класс)

На этой схеме не указаны классы Vector3f, Ray, некоторые одиночные классы и соответствующие им связи, так как изображение всего этого значительно усложнило бы схему и ухудшило бы ее читаемость. Особого же интереса как сами эти классы, так и то, какие другие классы их используют, для общей схемы взаимодействия не представляют. Фактически, классы Vector3f и Ray так или иначе используются практически всеми другими классами.

### 3.2.8. Классы интерфейса (GUI)

Интерфейс данной программы строился путем создания классов диалоговых окон, меню и элементов управления. Общая диаграмма классов интерфейса изображена на рисунке:



Условные обозначения между классами:

- включение, использование

## 3.3. Пользовательский интерфейс

Программа имеет русскоязычный интерфейс, организованный в виде главного меню и диалоговых окон, позволяющих пользователю менять настройки сцены, изменять положение и кол-во источников света, положение и направление камеры, а также настройки качества получаемого изображения. Также интерфейс предоставляет пользователю полную информации о работе программы, а именно:

- Количество лучей, испускаемых в секунду

- Максималное кол-во полигонов в листе дерева ограничивающего объема

- Максимальная глубина вхождения в построенное дерев ограничивающих объемов

- Текущее состояние программы

**Программа может находится в четырех состояниях:**

- «Рендеринг» - процесс рендера изображения

- «Построение дерева» - процесс построения дерева ограничивающих объемов.

- «Загрузка объектов» - процесс загрузки объектов сцены

- «Ожидание» - процесс ожидания программы.

### 3.3.1. Главное меню

Структура главного меню:

* Настройки света – настройки источников света на сцене
* Настройки камеры – настройки положения и направления камеры на сцене
* Настройки качества – настройки качества получаемого изображения

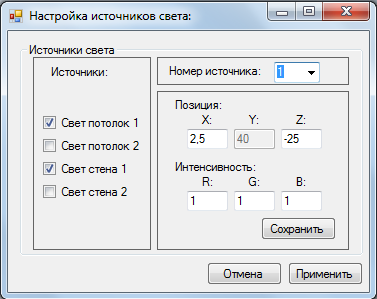
### 3.3.2. Основное диалоговое окно

Кроме Главного меню на основной форме программы есть такие элементы интерфейса, как кнопки и прочее, а именно:

* Рендер – запуск рендера изображения с выставленными параметрами.
* Стоп – остановка рендера изображения с выставленными параметрами.
* Выход – завершение работы программы.
* Разрешение – установка размеров получаемого изображения.

### 3.3.4. Настройки источников света

Рис. 1

Диалоговое окно редактора источников света позволяет редактировать список источников света сцены и изменять параметры отдельных источников. Внешний вид окна показан на рисунке.

Как видно из рисунка (рис 1), редактор источников света позволяет менять положение точечного источника света, но только в пределах плоскости стены (потолка), на которой(ом) он находится, его интенсивность и цвет, а также редактировать список существующих источников.

Изменения параметров определенного источника сохраняются с помощью кнопки «Сохранить», а чтобы изменения, связанные с количеством источников на сцене, вступили в силу, необходимо нажать кнопку «Применить».

### 3.3.5. Настройки камеры

В данном диалоговом окне, которое можно вызвать из редактора сцены, расположены параметры камеры. Окно изображено на рисунке:

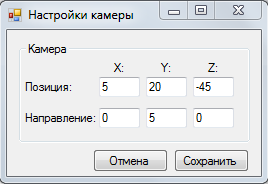


Рис. 2

Из рисунка (рис. 2) видно, что в диалоговом окне можно установить точку, в которой расположена камера, а также задать точку, в которую будет направлен луч наблюдения.

### 3.3.5. Настройки качества

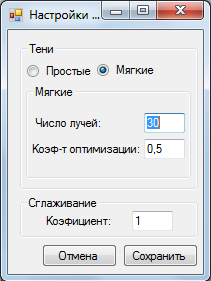
 Последним диалоговым окном, которое можно вызвать из редактора сцены, является окно настроек качества изображения, изображенное на рисунке (рис. 3) :

Рис. 3

В данном окне можно менять тип создаваемых теней (мягкие или обычные), число лучей на точку при отрисовке мягких теней и коэффициент оптимизации, влияющий на качество получаемого изображения при оптимизации РПЛ (Распределенной Трассировки Лучей).

## 3.4. Формат исходных данных

Данные сцены задаются в файле особого формата, имеющем расширение \*.obj.

Каждая строка файла с исходными данными начинается с идентификатора:

* *#*
* *V*
* *Vn*
* *f*

Модификаторы могут повторяться

### Ниже представлено описание формата \*.obj и модификаторов, используемых в нем:

*# [string]*:

Комментарий, модификатор, служащий для внесения пояснений данных файла.

*v [float] [float] [float]*:

Позиция одной вершины в пространстве. Первая вершина из перечисленных в файле имеет индекс 1, последующие вершины нумеруются последовательно.

*vn [float] [float] [float]*:

Нормаль к полигону. Первая нормаль из перечисленных в файле имеет индекс 1, последующие вершины нумеруются последовательно.

*f [int] [int] [int]* или *f [int]/[int] [int] /[int] [int] /[int]* – Полигон. В первом случае задается индексами вершин, описанных выше в файле, во втором – индексами вершин и координатами на текстуре.

Разработанная в результате программа позволяет получать на экране растрового дисплея реалистическое изображение трехмерных сцен, составленных из ограниченного числа примитивов, благодаря алгоритму, разработанному на основе обратной трассировки лучей с надстройкой в виде распределенной трассировки лучей для вычисления мягких теней. Для вычисления освещения используется гибридая модель, состоящая из диффузного освещения и модели освещения Блинна-Фонга. Также реализовано сглаживание изображения и нанесение текстур. Примеры работы программы можно посмотреть в приложении (рис.4, рис.5, рис.6, рис.7).

# 4. ЭКСПЕРИМЕНТАЛЬНО-ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ

## 4.1. Описание экспериментов

Основным исследованием данной работы была реализация мягких теней и способов оптимизации Для чистоты проведения эксперимента при каждом опыте использовался один и тот же набор параметров:

* Положение и направбление камеры фиксировано
* Число источников света фиксировано (2)

Для того, чтобы исследование выглядело наглядно, были проведены следующие эксперименты:

* Рендер сцены с мягкими тенями без оптимизации с количеством лучей на точку от 10 до 100 с шагом увеличения 10.
* Рендер сцены с мягкими тенями с оптимизацией с количеством лучей на точку от 10 до 100 с шагом увеличения 10. Коэффициент оптимизации константен.
* Рендер сцены с мягкими тенями с оптимизацией с фиксированным количеством лучей на точку (50 лучей) и коэффициентом оптимизации от 0.05 до 1 с шагом 0.05.

По результатам данных экспериментов были составлены графики. (график №1, график №2).

В качестве объектов были взяты полигональные модели стола, двух стульев (одинаковые модели с разными координатами в пространстве), плоскостей, задающих стены, пол, потолок, зеркало и картину на сцене.

## 4.2. Результаты экспериментов

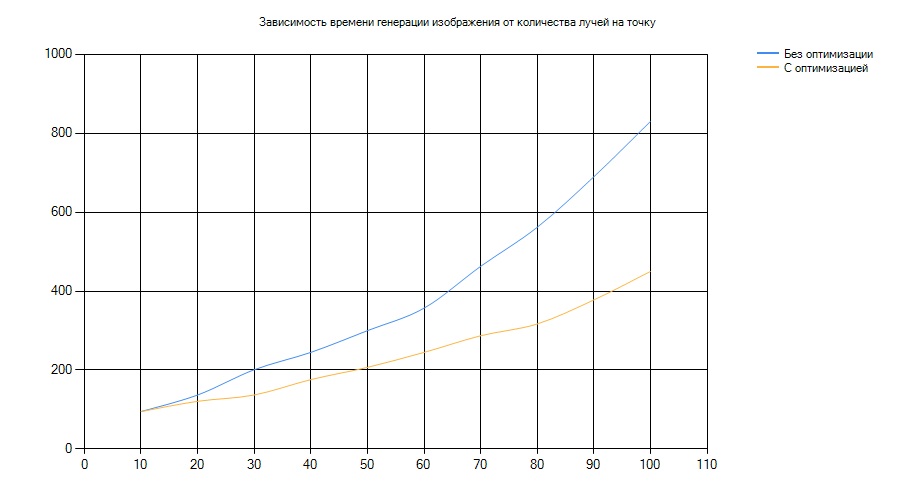
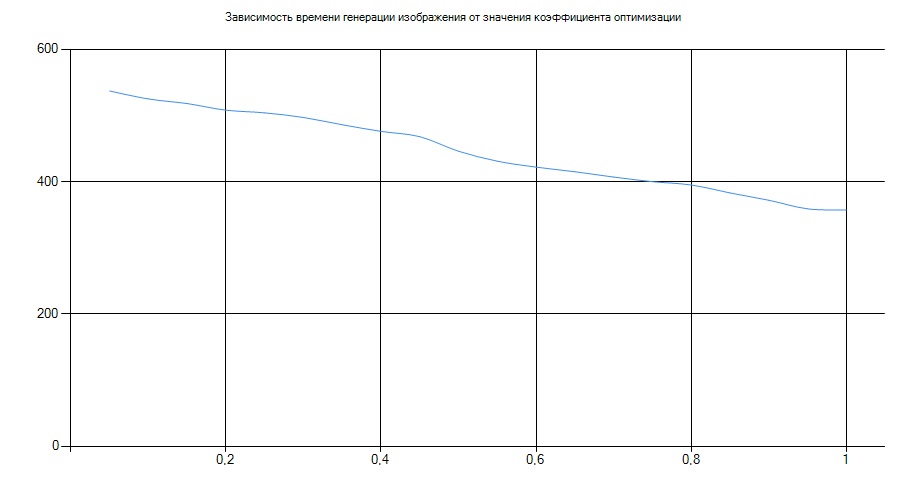


График №1

## 

## 4.3. Интерпретация результатов экспериментов

График №2

Из полученных графиков можно заметить:

* С ростом количества лучей на точку при вычислении мягких теней все больше заметен выигрыш в производительности оптимизированного алгоритма при небольшой потере в качестве в сравнении с неоптимизированным алгоритмом.
* Наибольшие вычислительные затраты идут на построение мягких теней, т.к. самым проблемным местом является вычисление мягких теней, что наглядно видно на графике зависимости скорости работы программы от количества лучей (график №1).
* С увеличением значение коэффициента оптимизации вычислительные затраты уменьшаются (график №2), однако и качество полученного изображения несколько снижается.

# 5. ЗАКЛЮЧЕНИЕ

Разработанная программа позволяет получать на экране растрового дисплея реалистическое изображение трехмерных сцен, составленных из ограниченного числа примитивов, благодаря алгоритму, разработанному на основе обратной трассировки лучей с надстройкой в виде распределенной трассировки лучей для вычисления мягких теней. Для вычисления освещения используется гибридая модель, состоящая из диффузного освещения и модели освещения Блинна-Фонга. Программа полностью удовлетворяет требованиям технического задания, обеспечивая:

* Возможность задания сцен, содержащих фиксированное количество трехмерных примитивов и источников света, а также задавать расположение и направление луча камеры.
* Реализация дерева ограничивающих объемов и оптимизация построения мягких теней.
* Поддержку наложения на трехмерные объекты текстуры.
* Улучшение качества изображения с использованием алгоритма суперсэмплинга.
* Возможность редактировать положение, количество (с фиксированным минимум и максимумом) и интенсивность источников света.
* Русскоязычный интерфейс пользователя.

По результатам работы программы были сделаны выводы:

* Данный алгоритм, разработанный на основе обратной трассировки лучей, обеспечивает высокую степень реалистичности изображения, а реализация дерева ограничивающих объемов и встроенные оптимизации позволяет повысить производительность вычислений. Но быстродействия данного алгоритма не хватает для реализации систем реального времени, генерирующих трехмерные сцены высокой реалистичности. Тем не менее, данный алгоритм можно использовать на параллельных вычислительных машинах, что позволит поднять производительность до уровня систем реального времени.
* Без дополнительной оптимизации увеличение числа объектов сцены ведет к значительному увеличению времени построения.
* Мягкие тени, а также алгоритмы текстурирования и устранения ступенчатости показали существенное увеличение реалистичности изображения.

# 6. СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

* + 1. Беклемишев Д.В. Курс аналитической геометрии и линейной алгебры. –М.: «Наука», 1971. 328 с

1. Порев В.Н. Компьютерная графика. – СПб.: БХВ-Петербург, 2002. 432 с.
2. Роджерс Д. Алгоритмические основы машинной графики. // Гл.5. Построение реалистичных изображений. 112 с.
3. Шикин Е. В., Боресков А.В. Компьютерная графика. – М.: Диалог-МИФИ, 1998. 287 с.
4. Шикин Е. В., Боресков А.В. Компьютерная графика. Полигональные модели – Москва, Диалог-МИФИ, 20
5. www.steps3d.narod.ru/tutorials/spatial-index-tutorial.html [интернет ресурс]
6. www.ray-tracing.com/articles184.html [интернет ресурс]

**Приложение**

**Схема алгоритма:**

**Схема подпрограммы Raytrace:**

Блок-схема №1

Raytrace

For

Расчет луча для текущего пикселя в МСК

For, по всем пикселям

Да

Вернуть текущий цвет

Отражение нулевое?

Raytrace

Вычислить отраженный луч

Да

Объект - источник?

Объект найден?

Да

Да

For, по всем источникам

For

Вычислить цвет

Вернуть цвет источника света

Вернуть текущий цвет

Поиск пересечения с объектом

\

Вернуть текущий цвет

Глубина > max

Блок-схема №2

**Пример работы программы:**

Минимальное качество рендеринга:

* 10 лучей на точку
* Коэффициент оптимизации = 0.5
* Без сглаживания



Рис. 1

Рис. 4

Рис 2.



Рис. 5

Высокое качество рендеринга:

* 70 лучей на точку
* Коэффициент оптимизации = 0.01
* Сглаживание с коэффициентом = 2



Рис. 7

Рис. 6