министерство НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
Российской федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Тюменский государственный университет»

институт математики и компьютерных наук

Кафедра программного обеспечения

**ОТЧЕТ**

по учебной практике

(практике по получению первичных профессиональных умений и навыков)

Тема практики: командная реализация проекта

«Разработка приложения для построения 3D моделей с помощью трассировки лучей».

Выполнили: студенты 2 курса

22МОиАИС184 группы

Скочко Андрей Евгеньевич  
Ибраев Ерлан Иржанович  
Чипак Данил Ильич

Руководитель практики:

к.ф.-м.н., доцент

Ступников Андрей Анатольевич

Оглавление

[Введение 3](#_Toc41954263)

[Глава 1 4](#_Toc41954264)

[Глава 2 10](#_Toc41954265)

[Глава 3 17](#_Toc41954266)

[Глава 4 20](#_Toc41954267)

[Заключение 21](#_Toc41954268)

[Список литературы 22](#_Toc41954269)

[Приложение 23](#_Toc41954270)

# Введение

Мы поставили себе цель создать приложение с помощью, которого пользователь сможет создавать определенные изображения путем трассировки лучей с 3D сцены, на которой пользователь сам растравляет определенные пользовательские 3D модели и источники света.

В нашем приложении пользователь сможет создавать 3D сцену, загружать и позиционировать на ней свои модели, расставлять и настраивать источники света, а также внутри редактора создавать и расставлять примитивы (куб, сфера, пирамида и т.д.). Потом, когда пользователь построит себе сцену, он, выбрав определенный ракурс камеры, сможет с помощью технологии трассировки лучей отрендерить себе изображение.

# Глава 1

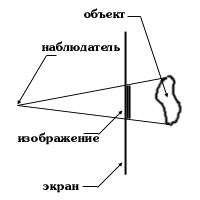
На стадии планирование проекта было решено разбить его на две части: рендер и редактор. Рендер было решено писать на чистом языке C++, сохранение кадра в файл будет реализовано с помощью сторонней библиотеки STB, также данная часть курсовой работы будет доминирующей. Редактор было решено создать на движке Unity с использованием языка C#.

Задачей рендера является, используя двумерный массив как полотно и набор объектов в пространстве, построить с помощью трассировки лучей изображение с учетом следующего: обработка таких материалов как стекло и зеркало, рендер теней, рендер бликов и рендер карты окружения.

Задачей редактора является обеспечить пользователя следующим функционалом: создание сцены, размещение на данной сцене своих пользовательских 3D моделей, создание с помощью функционала редактора объемных примитивов и последующее размещение их на сцене, размещение на сцене источников света с настраиваемой интенсивностью и цветом освещения. Также после того, как пользователь выберет определенный ракурс камеры, редактор должен в корректном формате передать все данные сцене рендеру и запустить его на отрисовку.

Обратная трассировка лучей (она же рэйкастинг, raycasting) - простой, хотя и довольно медленный, метод получения высокореалистичных изображений. Идея обратной трассировки лучей: для определения цвета пикселя экрана через него из камеры проводится луч, ищется его ближайшее пересечение со сценой и определяется освещенность точки пересечения. Эта освещенность складывается из отраженной и преломленной энергий, непосредственно полученных от источников света, а также отраженной и преломленной энергий, идущих от других объектов сцены. После определения освещенности этой точки учитывается ослабление света при прохождении через прозрачный материал и в результате получается цвет точки экрана.

Для определения освещенности, привносимой непосредственным освещением, из точки пересечения выпускаются лучи ко всем источникам света и определяется вклад всех источников, которые не заслонены другими объектами сцены. Для определения отраженной и преломленной освещенности из точки выпускаются отраженный и преломленный лучи и определяется привносимая ими освещенность. Непосредственное освещение от источника света считается по уравнению Фонга.



Ослабление света в прозрачной среде учитывается делением на коэффициент

k = exp(beta \* l), где

* k - коэффициент ослабления,
* beta - коэффициент прозрачности среды,
* l - длина пути внутри среды

Преломляется луч света по следующему закону: sin(b) = n1 \* sin(a) / n2, где

* a - угол между падающим лучом L и нормалью N
* b - угол между преломленным лучом L' и нормалью N
* n1 - коэффициент преломления среды, в которой проходит луч L
* n2 - коэффициент преломления среды, в которой проходит луч L'

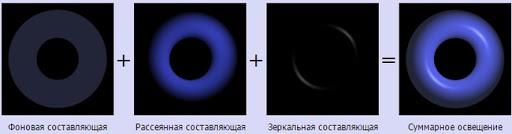
Алгоритм работы функции трассировки луча с началом o = (ox,oy,oz) и направлением d = (dx,dy,dz), возвращающей, кстати, освещенность по всем трем цветовым компонентам, будет выглядеть примерно так:

* ищем ближайшее пересечение луча со сценой (определяем точку p, где произошло пересечение, выясняем, с каким конкретно объектом оно произошло)
  + если не нашли, возвращаем 0
  + если пересечение есть, текущую освещенность полагаем равной фоновой освещенности (ambient)
* определяем вектор n, нормаль к объекту в точке пересечения
* для каждого (точечного) источника света
  + считаем вектор l, соединяющий источник и точку p (начало в p)
  + ищем пересечение луча с началом в точке p и направлением d со сценой
    - если нашли, прекращаем обработку этого источника, так как его не видно
  + считаем симметричный l относительно n вектор r
  + считаем по уравнению Фонга (оно использует n, l, r, d) освещенность в точке за счет этого источника света, добавляем ее к текущей
* считаем симметричный d относительно n вектор refl, то есть отраженный вектор
* трассируем отраженный луч (с началом p и направлением refl), добавляем полученную освещенность к текущей
* считаем направление преломленного луча (вектор refr) по d, n и коэффициентам преломления
* трассируем преломленный луч (с началом p и направлением refr), добавляем полученную освещенность к текущей
* покомпонентное умножаем текущую освещенность на цвет объекта

если для материала, заполняющего пройденную лучом от o до p трассу beta != 0, умножаем текущую освещенность на коэффициент ослабления

* возвращаем текущую освещенность

Модель отражения Фонга является довольно популярной и широкоиспользуемой в компьютерной графике. Пусть заданы точечный источник света, расположенный в некоторой точке, поверхность, которая будет освещаться и наблюдатель. Будем считать, что наблюдатель точечный. Каждая точка поверхности имеет свои координаты и в ней определена нормаль к поверхности. Её освещенность складывается из трех компонент: фоновое освещение (ambient), рассеянный свет (diffuse) и бликовая составляющая (specular). Свойства источника определяют мощность излучения для каждой из этих компонент, а свойства материала поверхности определяют её способность воспринимать каждый вид освещения.



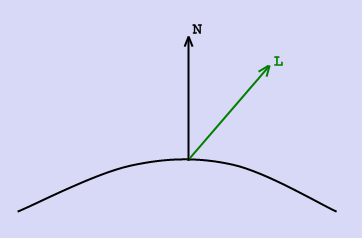
Для удобства все векторы, описанные ниже, берутся единичными. В этом случае косинус угла между ними совпадает со скалярным произведением.

1. Фоновое освещение — это постоянная в каждой точке величина надбавки к освещению. Вычисляется фоновая составляющая освещения как , где

* - фоновая составляющая освещенности в точке,
* - свойство материала воспринимать фоновое освещение,
* - мощность фонового освещения.

Из формулы выше видно, что фоновая составляющая освещенности не зависит от пространственных координат освещаемой точки и источника. Поэтому при моделировании освещения, в большинстве случае, не имеет смысла брать более одного фонового источника света. Часто просто задается некое глобальное фоновое освещение всей сцены.

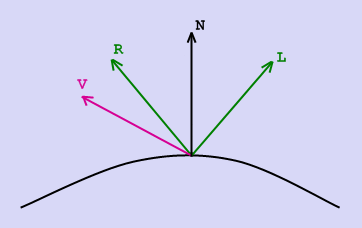
2. Рассеянный свет при попадании на поверхность рассеивается равномерно во все стороны. При расчете такого освещения учитывается только ориентация поверхности (нормаль) и направление на источник света. Рассеянная составляющая рассчитывается по закону косинусов (закон Ламберта):



, где

* - рассеянная составляющая освещенности в точке,
* - свойство материала воспринимать рассеянное освещение,
* - мощность рассеянного освещения,
* - направление из точки на источник,
* - вектор нормали в точке.

3. Зеркальный свет при попадании на поверхность подчиняется следующему закону: “Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части”. Т. о. отраженная составляющая освещенности в точке зависит от того, насколько близки направления на наблюдателя и отраженного луча. Это можно выразить следующей формулой:



В общем случае вектора не лежат в одной плоскости.

, где

* - зеркальная составляющая освещенности в точке,
* - коэффициент зеркального отражения,
* - мощность зеркального освещения,
* - направление отраженного луча,
* - направление на наблюдателя,
* α- коэффициент блеска, свойство материала.

Во всех вычислениях выше, для рассеянной и зеркальной компонент, если скалярное произведение в правой части меньше нуля, то соответствующая компонента освещенности полагается равной нулю.

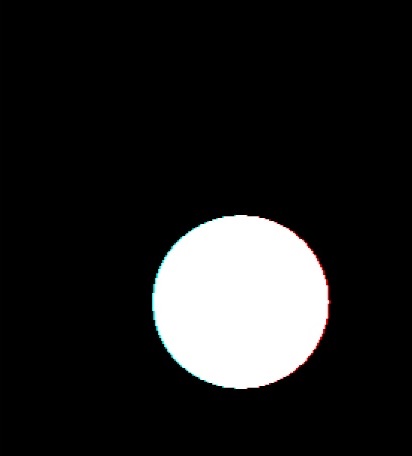
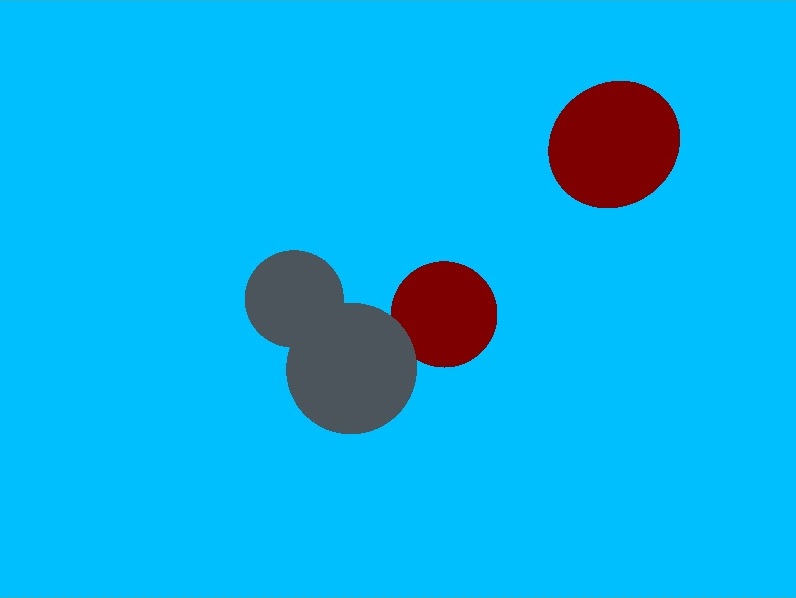
Для расчета отраженной компоненты требуется выполнить довольно громоздкие вычисления. Существует модель Блинна-Фонга, представляющая собой модель Фонга с упрощенным расчетом зеркального отражения. Вычислим в каждой точке вектор полупути

который показывает ориентацию площадки, на которой будет максимальное отражение. Тогда величину можно заменить величиной .

При этом α <> β и, в общем случае, соотношение между ними зависит от пространственной связи векторов . Вектор называется вектором полупути, т.к. если все три вектора лежат в одной плоскости, то угол между и составляет половину угла между

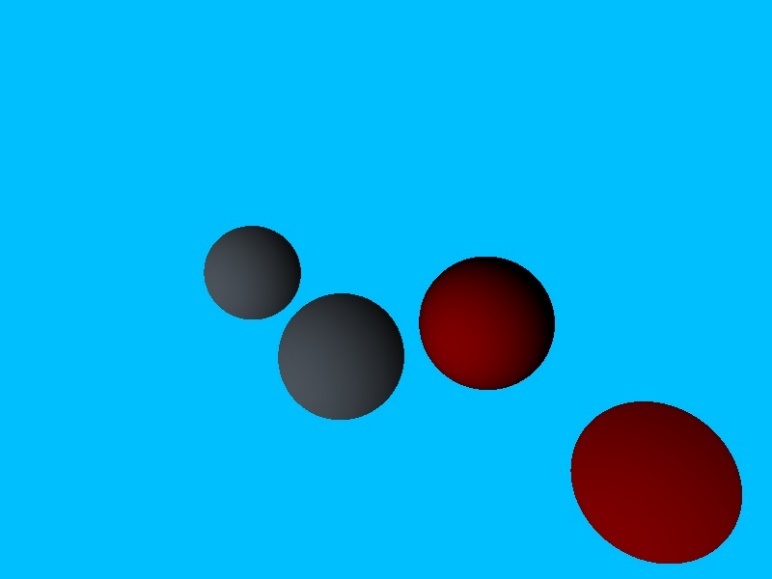
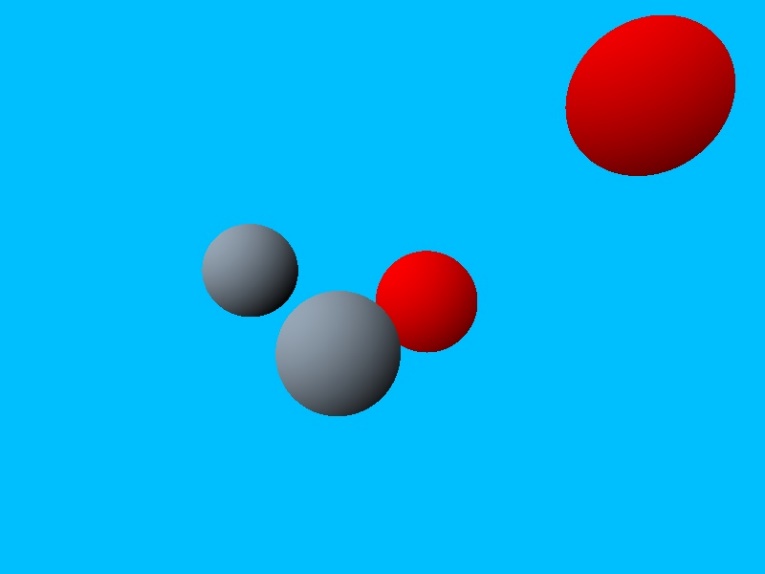
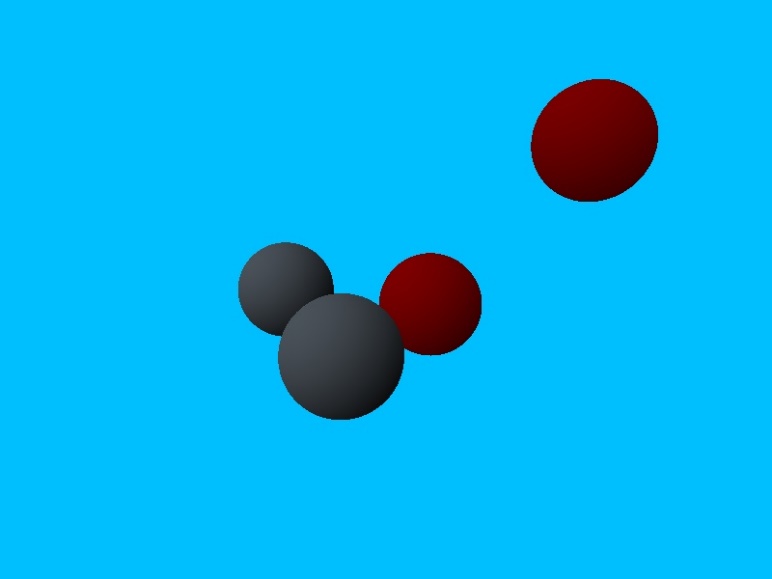
Модель отражения Блинна-Фонга никогда в точности не совпадает с моделью Фонга, однако можно подобрать соответствующие значения α и β, для которых распределения зеркальной составляющей по поверхности для обеих моделей будут очень близкими. Вместе с тем, в ряде случаев модель Блинна-Фонга требует значительно меньше вычислений, например в случае направленного бесконечно-удаленного источника.

# Глава 2

Первым делом было решено начать разработку рендера, т.к. это являлось приоритетным пунктом в достижении нашей цели. Создание рендера шло поэтапно. Сначала был создан двумерный массив, ячейками которого были трехмерные вектора, где координаты x, y, z обозначали красный, зеленый и синий цвет по цветовой модели RGB соответственно. Далее этот двумерный массив будем называть полотном. Потом необходимо было создать какой-либо примитив на 3D сцене и выбор пал на сферу. Поначалу для ее хранения в памяти достаточно знать координаты ее центра в формате трехмерного вектора, радиус и цвет по модели RGB в формате трехмерного вектора. Следующим шагом было написание самого трассировщика: поочередно через каждый пиксель в полотне от зафиксированной точки, которой являлась камера, пускались лучи. Если луч касался какого-либо объекта, то пиксель, через который проходил луч на данном полотне, окрашивался в цвет этого объекта. Если же луч проходил определенное расстояние, но так и не встречал какой-либо объект, то пиксель окрашивался в определенный цвет, выбранный в качестве фона. Таким образом мы смогли получить фоновое(ambient) освещение объекта, в нашем случае одной сферы (Рисунок 1). Далее мы решили поменять фоновый цвет, а также добавить объектам такое свойство, как материал. На данном этапе это свойство пока хранило только цвет объекта. Далее было создано уже несколько сфер, на сцене, но с разными материалами. В результате мы получили то, что изображено на Рисунке 2.

**Рисунок 2. Несколько сфер.**

**Рисунок 1. Фоновая составляющая(ambient).**

 Следующим этапом было добавление диффузного освещения. Для этого необходимо было сначала создать источник освещения, в нашем случае точечный свет. Для этого мы создали отдельный объект, в котором хранились координаты данного точечного света и его интенсивность. Далее мы разместили пару источников света на сцене. Чтобы научить рендер обрабатывать объекты с учетом падающего на них света, нужно при каждом попадании луча на объект, когда мы начинаем рисовать картинку, вычислять сумму скалярных произведений нормального вектора в этой точке и всех векторов, описывающих направление света от каждого источника этого света. Если сумма скалярное произведение меньше нуля, то далее вместо данного скалярного произведения мы будем использовать нуль. Мы берем полученное значение и перемножаем его на интенсивность источника света и получаем коэффициент диффузного освещения. Этот коэффициент мы умножаем на цвет объекта. Таким образом мы получаем по-разному освещенный в разных областях объект, отображенный на полотне(Рисунок 3,4 и 5).

**Рисунок 5. Диффузное освещение(diffuse).**

**Рисунок 4. Диффузное освещение(diffuse).**

**Рисунок 3. Диффузное освещение(diffuse).**

Далее нужно было реализовать блестящие поверхности. Для этого мы обновим структуру материала: добавим альбедо - массив из двух элементов, который будет содержать в себе специальные коэффициенты, первый будет влиять на интенсивность диффузного освещения, второй будет влиять на интенсивность бликов на объектах, которые мы сейчас реализуем, так же добавим еще одно значение, которым будет являться зеркальный показатель, отвечающий за то, насколько гладким будет объект. Потом опять вернемся к тому месту, где мы высчитывали цвет пикселя при диффузном освещении, т.е. момент заполнения полотна пикселями с помощью трассирования лучей. Здесь немного изменим ход вычислений – теперь помимо умножения цвета материала на коэффициент диффузного освещения мы так же еще домножим на первый элемент массива альбедо. Но это еще не все, нам так же необходимо к полученному значению будет прибавить произведение единичного вектора на коэффициент зеркального освещения и второй элемент массива альбедо данного материала – это позволит нам получить эффект блика. Найти значение коэффициента зеркального освещения в определенной точке объекта можно следующим способом:

, где

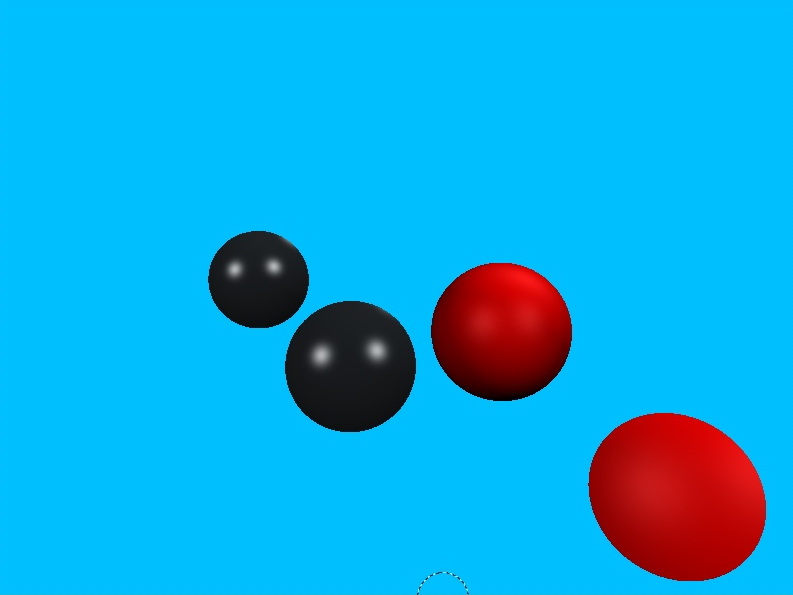
– единичный вектор нормали в данной точке

– единичный вектор, обратный вектору, описывающего направление света от n-ого источника света

k – количество источников света

E – зеркальный показатель материала

– интенсивность n-ого источника света

Таким образом мы получаем следующий результат, где уже видны блики от двух источников света только на черных сферах, т.к. у них материал более гладкий(Рисунок 6).

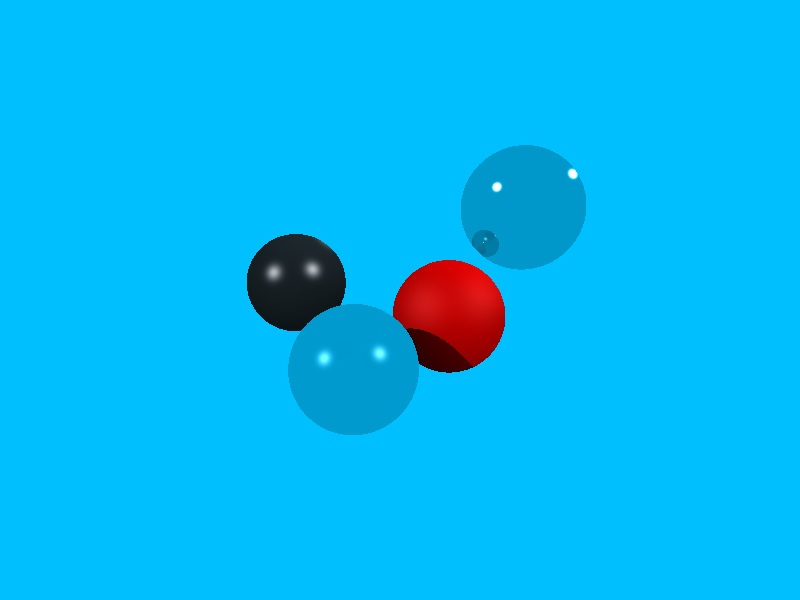
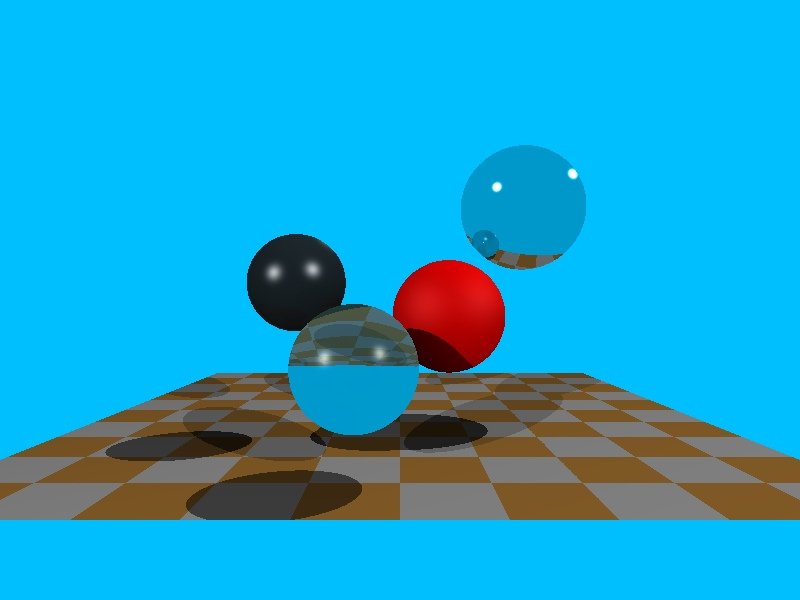
**Рисунок 6. Блики на объектах.**

Следующий этап тени и отражения. В нашем случае определить тени на экране будет достаточно просто – нужно проверять для каждого источника света при попадании луча через полотно на какой-либо объект, не пересекает ли луч, идущий от источника света до точки пересечения нашего луча с объектом, по пути какие-либо объекты. Если пересекает, то мы не берем в учет при высчитывании коэффициентов диффузного и зеркального освещения данный источник освещения. Теперь отражения – их так же достаточно просто реализовать. Но сначала нам нужно расширить массив альбедо в структуре материала на одну ячейку – это будет коэффициент, влияющий на интенсивность зеркального отражения данного материала. Далее необходимо просчитать отраженный луч и повторно с помощью рекурсии отправить уже в сторону отраженного луча. Найти отраженный луч можно следующим путем:

, где

– единичный вектор нашего луча, который мы пропускаем через полотно

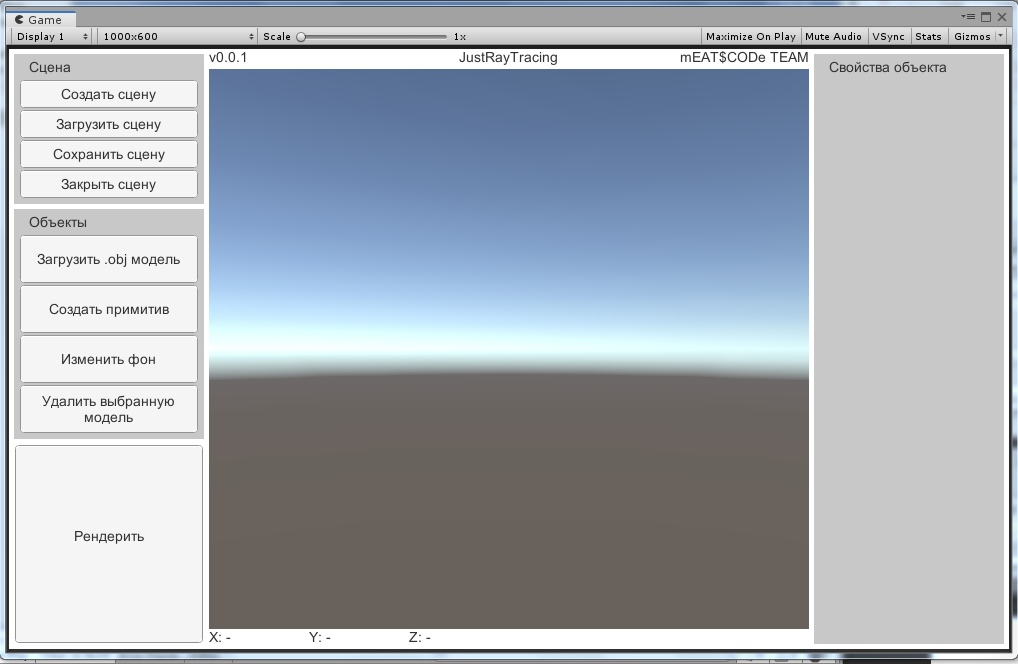
- единичный вектор нормали в данной точке

 Есть небольшой нюанс – необходимо ограничить количество отражений от зеркальной поверхности, иначе мы рискуем попасть в бесконечный цикл. Так же чем меньшее количество раз мы будем отражать луч, тем быстрее рендер будет работать. Теперь нужно окрасить сам пиксель. Отправимся обратно в место, где мы высчитывали цвет пикселя на полотне, но уже к данной формуле еще прибавим произведение полученного выше вектора на третий элемент массива альбедо материала данного объекта. В результате чего мы получим красивую картинку(Рисунок 7).

**Рисунок 7. Тени и отражения.**

Добавим немного мелочи на нашу сцену, чтобы увидеть наш рендер во всей красе(Рисунок 8).

**Рисунок 8. Готовый трассировщик лучей.**

Итак, мы имеем более-менее готовый рендер, теперь время переходить к редактору сцены, с помощью которого пользователь сам сможет себе рисовать нужные ему картинки с помощью нашего трассировщика лучей. Редактор мы реализуем при помощи игрового движка Unity. Первое что необходимо в нашем редакторе – непосредственно сам интерфейс (Рисунок 9). 

**Рисунок 9. Интерфейс редактора.**

В редакторе должен быть следующий, по нашему мнению, функционал:

* Возможность создавать, сохранять, загружать и выгружать сцену.
* Взаимодействовать со сценой, расставляя на ней все необходимые модели и т.д.
* Большая красивая кнопка, позволяющая отрендерить полученную сцену в нужном ракурсе с помощью написанного ранее трассировщика лучей.
* Справа находится динамическая область, в которой можно менять свойства выбранного объекта и т.д.
* Информационная панель внизу, отображающая координаты выбранного объекта в пространстве.
* Непосредственно сама рабочая область, через которую можно взаимодействовать с самой сценой напрямую.

Что бы в полной мере комфортно чувствовать себя во время построения своей сцены, мы реализовали следующее управление при зажатом по рабочей области ПКМ(правой кнопки мыши):

* W – двигать камеру вперед
* S – двигать камеру назад
* A – двигать камеру влево
* D – двигать камеру вправо
* Space – двигать камеру вверх
* Alt – двигать камеру вниз
* Если зажать Shift, то скорость движения камеры повысится, если отпустить, то понизится то стандартной

Далее в нашем редакторе появилась возможность создавать и размещать на сцене определенные примитивы, а так же загружать карту окружения (Рисунок 10).

**Рисунок 10. Создание примитива в редакторе сцены.**

Следующим шагом было создание возможности выбора с помощью ЛКМ(левой кнопки мыши) нужной модели. При этом вокруг модели появляются стрелки, позволяющие позиционировать объект в пространстве (Рисунок 11).

На момент написания данного отчета мы пока остановились на этом этапе разработки…

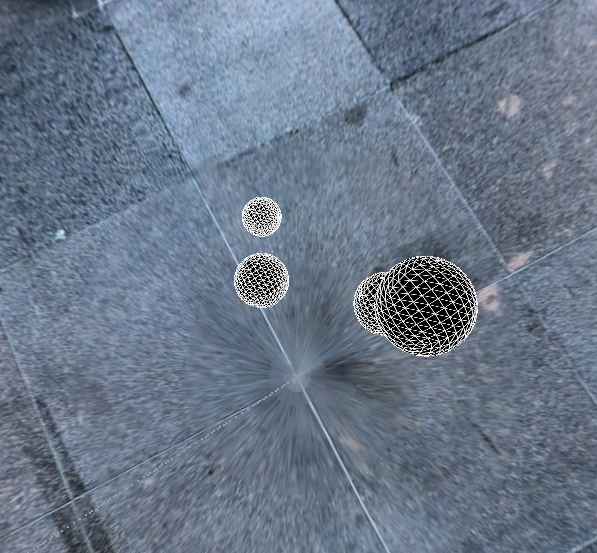
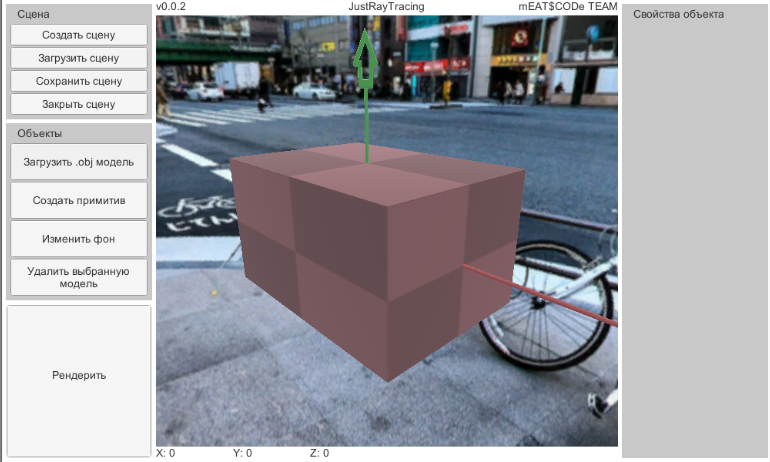
**Рисунок 11. Позиционирование объекта на сцене.**

# Глава 3

 В итоге мы уже имеем рендер, способный выдавать хорошую картинку. Смотрите Рисунок 12, на данное изображение разрешением 2000х2000 пикселей при рендере на довольно слабой машине (2-ядерный Pentium Dual-Core E6600 по 3.06GHz на ядро) ушло около часа.

**Рисунок 12. Впечатляющий результат работы нашего рендера.**

Касательно редактора мы имеем приложение, разработанное на Unity, которое уже способно создавать разные примитивы и позиционировать их на сцене(Рисунок 13), а так же весьма удобное управление камерой в рабочей области(Рисунок 14, 15 и 16).



**Рисунок 15. Свободный полет.**

**Рисунок 14. Свободный полет.**

**Рисунок 13. Параллелограм на сцене.**



**Рисунок 16. Свободный полет.**

# Глава 4

Скочко Андрей:

* был написан трассировщик лучей (см. Листинг программы 1 и 2 в Приложении);
* изучен С++ и переданы знания о языке команде;
* разбиение некоторых сложных задач на подзадачи.

Ибраев Ерлан:

* полный анализ проделываемой работы, сбор и откладывание заметок;
* формирование отчета на основе сделанных заметок;
* разработка части функционала для редактора сцен (см. Листинг программы 3 в Приложении);
* выявление багов в работе рендера.

Чипак Данила:

* плотное изучение теоретической части и передача опыта команде;
* предложение решений некоторых проблем на стадии разработки рендера;
* тестирование рендера и редактора сцен, выявление проблем в работе программ;
* написана часть функционала для редактора сцены(см. Листинг программы 4 в Приложении).

# Заключение

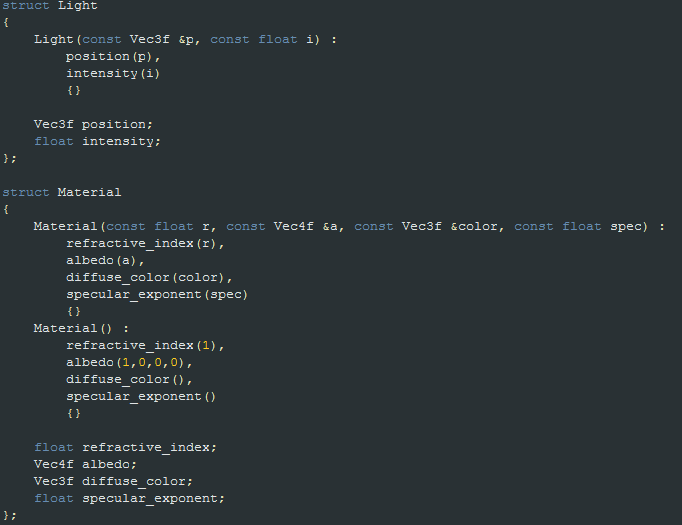
В ходе выполнения данной работы, был получен неоценимо большой опыт работы с 3D графикой, написания собственной системы рендера, изучения языка С++. Хоть данная работа и кажется нам очень сложной, мы все же смогли добиться немалых успехов по ходу разработки нашего приложения. У нас есть рендер, написанный только с использованием языка программирования C++, способный выдавать картинку весьма высокого качества. Так же был приобретен большой опыт работы с Unity в процессе создания редактора сцен. Ну и что более важно, мы смогли получить хороший опыт работы в команде во время разработки нашего ПО. Каждый из нас внес большой вклад в разработку, прикрывая определенные профессиональные слабости своего сокомандника, а также смог передавать свой определенный опыт всей команде.

# Список литературы

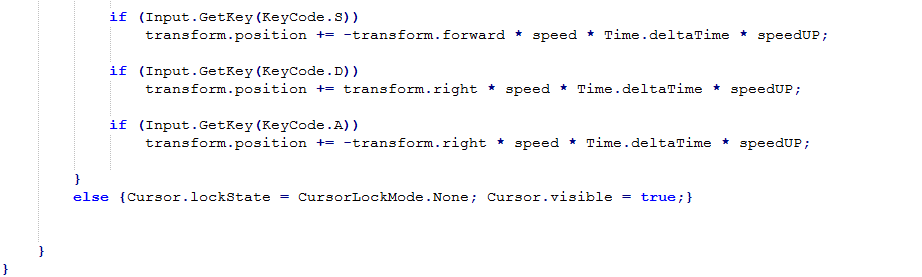
1. 256 строчек голого C++: пишем трассировщик лучей с нуля за несколько часов. [Электронный ресурс. URL:<https://habr.com/ru/post/436790/>. Дата обращения 16.03.2020]
2. Метод обратной трассировки лучей. [Электронный ресурс. URL:<http://shackmaster.narod.ru/raytrace.htm>. Дата обращения 16.03.2020]
3. Обратная трассировка лучей. [Электронный ресурс. URL:<http://students.uni-vologda.ac.ru/pages/pm01/kps/intro.htm>. Дата обращения 16.03.2020]
4. Особенности моделирования света: Аппроксимации Фонга (Phong Shading).[Электронный ресурс. URL:<https://www.ixbt.com/video/light-model-phong.html>. Дата обращения 21.03.2020]
5. Фонг затенения - Phong shading. [Электронный ресурс. URL:<https://ru.qwe.wiki/wiki/Phong_shading>. Дата обращения 21.03.2020]
6. Закон Ламберта. Модель отражения Фонга. Модель отражения Блинна-Фонга .[Электронный ресурс. URL:<https://compgraphics.info/3D/lighting/phong_reflection_model.php>. Дата обращения 04.04.2020]
7. Бьерн Страуструп. Язык программирования С++. [Книга. Издательство Бином 2015г. Дата обращения 07.03.2020]
8. Джозеф Хокинг. Unity в действии. [Книга. Издательство ПИТЕР 2016г. Дата обращения 15.04.2020]

# Приложение

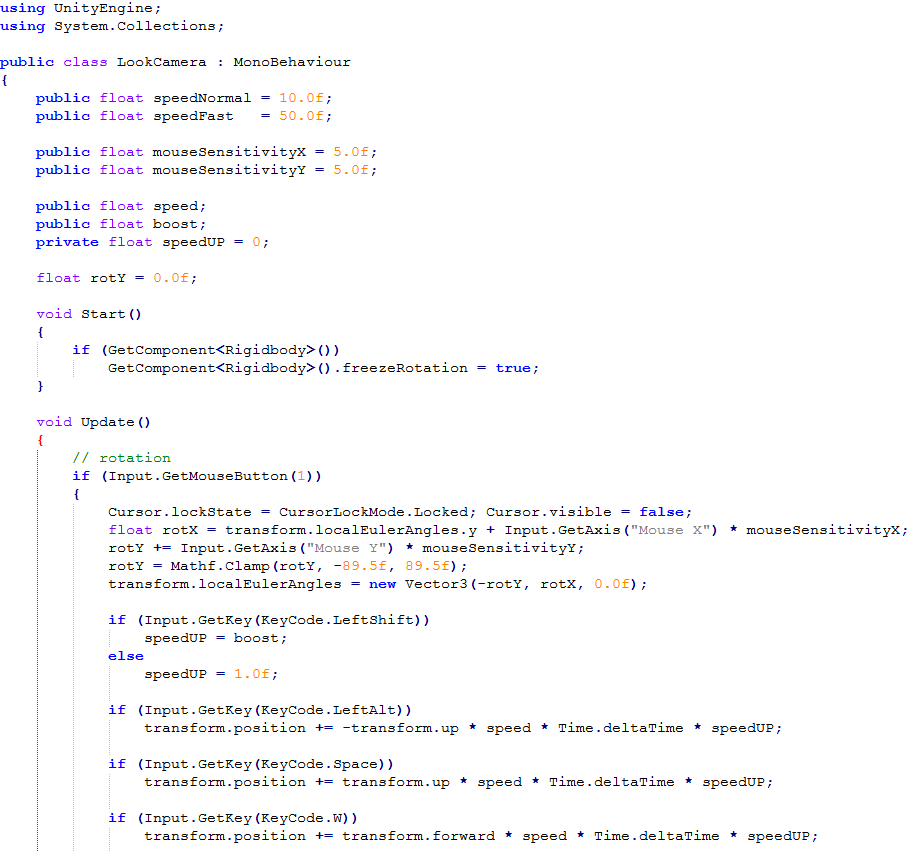
**Листинг программы 1. Метод вычисления цвета пикселя путем выпуска луча через полотно(С++).**

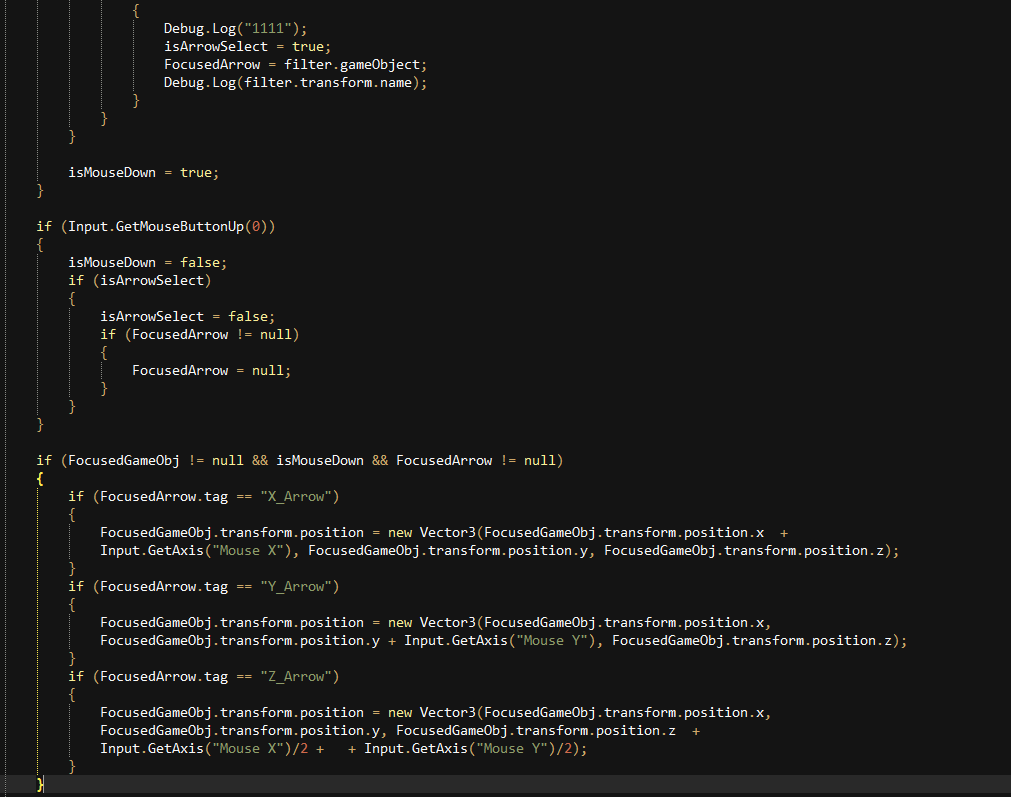
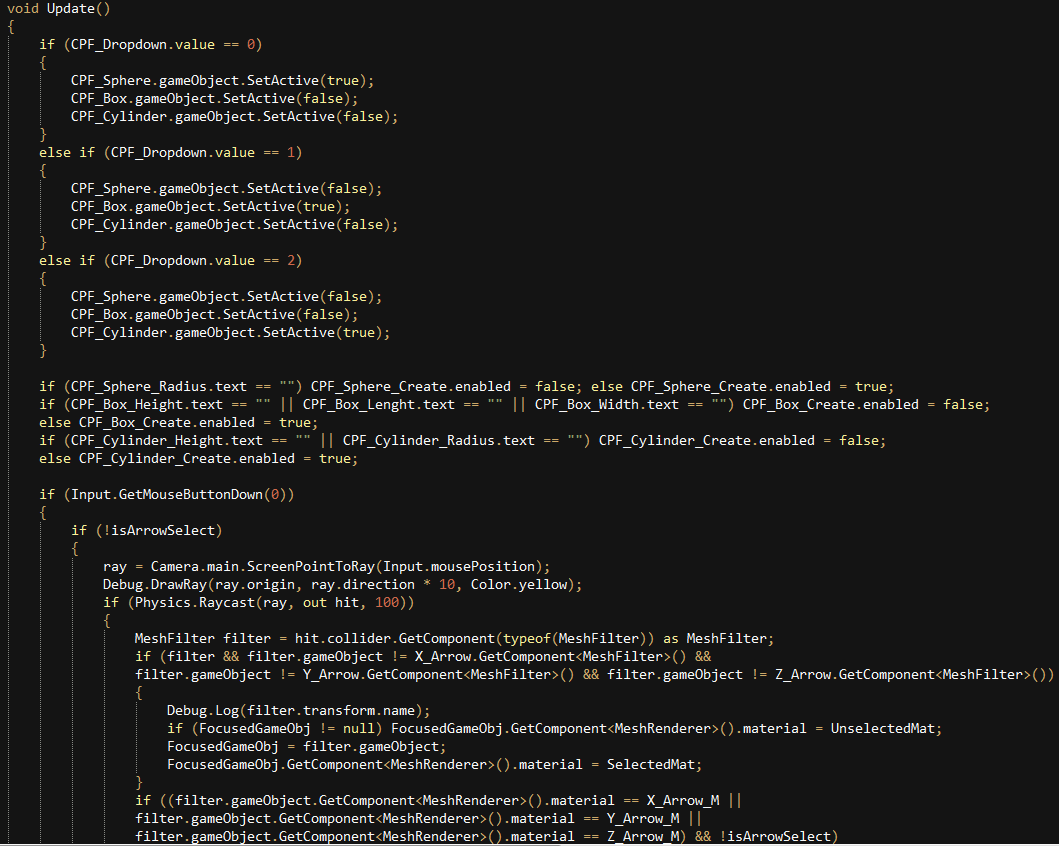


**Листинг программы 2. Структуры источника света и материала объекта(С++).**



**Листинг программы 3. Реализация управления камерой в редакторе сцен(Unity\C#).**





**Листинг программы 3. Метод Update для редактора сцен(Unity\C#).**