МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
Национальный исследовательский нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики
Кафедра математического обеспечения и суперкомпьютерных технологий

Отчёт по практике

Тема :

Реализация трассировки лучей с использованием технологии OpenGL

Выполнил:
студент гр. 381706-1
Власов Андрей Сергеевич
Научный руководитель:
Профессор, доктор технических наук
Турлапов Вадим Евгеньевич

Нижний Новгород 2020

Содержание

1 Введение

Одним из методов визуализации в компьютерной графике является трассировка лучей, или рейтрейсинг (ray tracing). Он позволяет добиться синтеза реалистичных изображений, так как процедура трассировки лучей имитирует поведение фотона. Данный метод требует большой вычислительной мощности, поэтому ранее он применялся редко и в ограниченном объеме. Но сейчас с развитием технологий становится все популярнее и набирает обороты.

OpenGL (Open Graphics Library - открытая графическая библиотека, графическое API) - спецификация, определяющая независимый от языка программирования платформонезависимый программный интерфейс для написания приложений, использующих двухмерную и трёхмерную компьютерную графику.

Основным принципом работы OpenGL является получение наборов векторных графических примитивов в виде точек, линий и многоугольников с последующей математической обработкой полученных данных и построением растровой картинки на экране и/или в памяти.

Целью данной работы является
- рассмотрение алгоритма трассировки лучей
- разбор основных составляющих частей программной реализации
- описание основных положений из теории
- предъявление результатов проведенных экспериментов

2 Необходимый теоретический минимум

1. Нахождение пересечения луча и сферы [1]

Пусть у нас задана сфера, описанная уравнением ||p – c|| = r, а также задан луч, описанный уравнением p(t) = p_0 + lt. Существует довольно простой геометрический метод нахождения пересечения луча со сферой. Для начала найдем расстояние от начала луча до ближайшей к центру сферы точки на луче. Этой точке соответствует значение параметра

t_0 = (c – p_0, l).

Далее найдем квадрат расстояния от этой точки до центра сферы:

d^2 = (c – p_0, c – p_0) – (c – p_0, l)^2.

Если $r^2 - d^2 < 0$, то луч проходит мимо сферы и пересечения нет. В противном случае по теореме Пифагора найдем расстояние m(рис. 1) и по нему соответствующие точкам пересечения значения параметра t:

$m^2 = l^2$ - d^2

q = (r^2 - m^2)$^{1/2}$

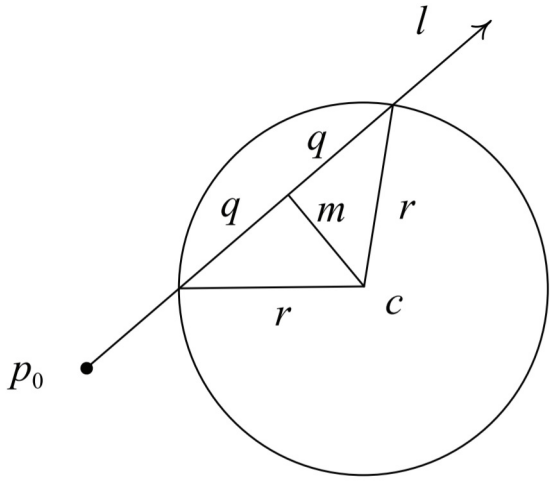

Рис. 1 - Нахождение ближайшей точки пересечения луча со сферой.

2. Модель Ламберта (идеальное диффузное освещение) [1]

Как легко можно заметить, большинство материалов, которые мы наблюдаем, отражает свет совсем иначе – падающий свет рассеивается по всей верхней полусфере (но не обязательно равномерно).

Построение аккуратной модели подобного рассеивания довольно сложно, поэтому мы начнем с описания ряда приближенных (упрощенных) моделей. Простейшей из подобных моделей является модель Ламберта.

Созданная более века назад диффузная модель считает, что падающий в точку P свет равномерно рассеивается во всех направлениях верхней полусферы (рис. 2).

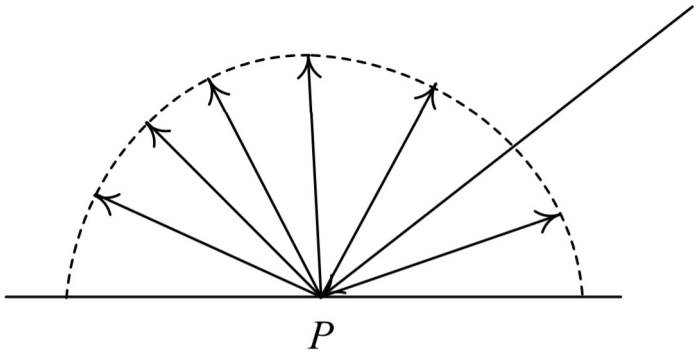

Рис. 2 - Идеальное диффузное рассеивание.

Тогда видимая в точке P освещенность вообще не будет зависеть от направления на наблюдателя/камеру. Единственное, от чего будет зависеть то, насколько яркой мы увидим точку P, – это падающая в эту точку световая мощность на единицу площади.

Давайте рассмотрим область S на плоскости, освещаемую падающим на нее лучом света. Через S_{perp} мы обозначим площадь поперечного сечения луча, освещающего S. Из курса школьной геометрии мы знаем, что эти площади связаны следующим соотношением:

$S_{perp} = \mathrm{S}\cdot\cos\theta$

Отсюда мы сразу получаем, что мощность на единицу площади будет прямо пропорциональна косинусу угла падения (рис. 3).

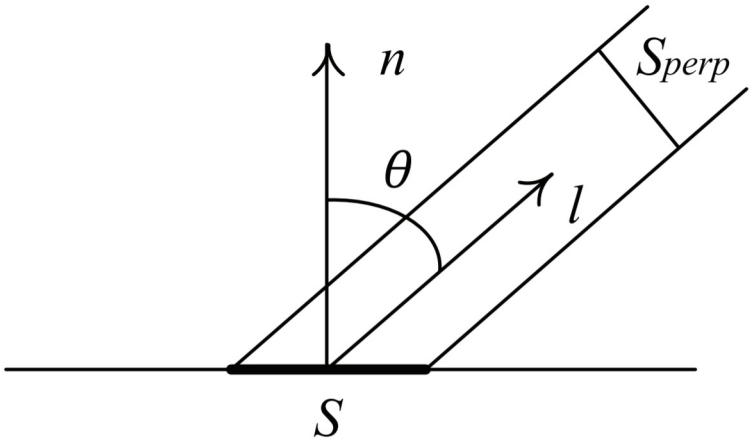

Рис. 3 - Удельная мощность падающего света.

Давайте через I_l обозначим RGB-вектор, задающий цвет и мощность источника света, через C – цвет поверхности в точке P (тоже как RGB-вектор) и через I видимую освещенность в этой точке. Тогда мы можем записать следующее уравнение для идеальной диффузной освещенности:

$\mathrm{I} = I_l\cdot\mathrm{C}\cdot\cos\theta$

Считая векторы l и n единичными, мы можем переписать это уравнение через скалярное произведение:

$\mathrm{I} = I_l\cdot\mathrm{C}\cdot\max(0,(\mathrm{n},\mathrm{l}))$.

В этой формуле используется max, для того чтобы не получалось отрицательных цветов, когда источник светит снизу (а такое может быть в реальных сценах). К сожалению, использование этой формулы приводит к тому, что неосвещенные участки оказываются полностью черными. Чтобы этого избежать, часто в эту формулу добавляют еще один член – фоновое (ambient) освещение: некоторый свет, падающий равномерно со всех сторон и ни от чего зависящий. Тогда итоговая освещенность может быть представлена в виде суммы двух членов с весами k_a и k_d соответственно:

$\mathrm{I} = k_a\cdot I_a\cdot\mathrm{C} + I_l\cdot\mathrm{C}\cdot\max(0,(\mathrm{n},\mathrm{l}))$.

Обратим внимание, что это приблизительная формула, не претендующая на физическую корректность.

3. Модель Фонга. [1]

Одним из наиболее заметных недостатков модели Ламберта является полное отсутствие бликов на гладких поверхностях. Есть различные способы учесть блики, большинство из них просто добавляет к ранее рассмотренной формуле еще один член, отвечающий за блики (со своим весом k_s).

Одной из самых первых моделей освещения, поддерживающих блики, была модель Фонга, задаваемая следующей формулой:

$\mathrm{I} = k_a\cdot I_a\cdot\mathrm{C} + I_l\cdot\mathrm{C}\cdot\max(0,(\mathrm{n},\mathrm{l})) + k_s\cdot I_l\cdot C_s\cdot\max(0,(\mathrm{r},\mathrm{l}))^p$

Здесь через C_s обозначен цвет блика, через r – вектор v, отраженный относительно вектора нормали n, и через p – некоторый коэффициент, задающий степень неровности поверхности (чем он больше, тем более гладкой является поверхность) (рис. 4).

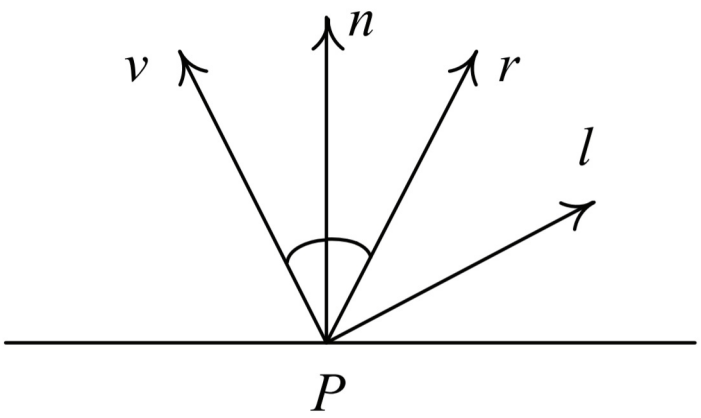

Рис. 4 - Векторы для модели Фонга.

Отраженный вектор r задается следующей формулой:

r = 2n·(n, l)–l.

Цвет блика C_s для диэлектриков является просто белым цветом, а для металлов он совпадает с цветом поверхности C.

3 Краткий обзор источников

1. Статья *Whitted T. An improved illumination model for shaded display.* [2].

   В статье Уиттед впервые изложил такой метод, как трассировка лучей. До статьи большинство программ уже имитировали внешний вид диффузных и зеркальных поверхностей. Знаменитая модель Фонга была уже известна, но моделирование сложных отражений и преломлений еще предстояло сделать. Уиттед как раз предложил использовать трассировку лучей для решения этой проблемы. Сам по себе алгоритм позволяет получить изображения лишь с базовым уровнем реалистичности и рассчитывает такие эффекты как тени, отражения и преломления.

2. Статья *Cook, R. L., Porter, T., and Carpenter, L. Distributed ray tracing.* [3].

   В статье был предложен алгоритм распределенной трассировка лучей. В реальности источники света часто не точечные, а материалы не всегда отражают всю энергию в одном направлении по закону “угол падения равен углу отражения”. Из-за того что источник света имеет размер, границы перехода свет-тень получаются нечеткими, потому что на этих границах затеняющим объектом закрыта только часть источника света. Чтобы вычислить, какая часть источника света закрыта, до него трассируют несколько теневых лучей, выбирая каждый раз случайную точку на источнике освещения для задания направления луча.

3. Книга *Бородков А.В. Программирование компьютерной графики. Современный OpenGL.* [1].

   В данной книге рассматриваются основы современной компьютерной графики. Подробно рассматривается ряд чисто математических понятий, различные способы задания ориентации объектов в пространстве, модели для представления цвета, простые эмпирические модели, например, Ламберта и Фонга, алгоритмы, а также физика освещения. Отдельные главы посвящены методу трассировки лучей и современному OpenGL.

4 Разбор источников

4.1 Статья *Whitted T. An improved illumination model for shaded display.*

Оптические и физические законы, управляющие явлениями отражения и преломления, хорошо известны. Направление отражения зависит только от ориентации поверхности и направления входящего света. Направление преломления можно рассчитать по закону Снеллиуса, оно зависит от ориентации поверхности, направления входящего света и показателя преломления материала.

Уиттед предложил использовать эти законы для вычисления направления отражения и преломления лучей, когда они пересекают отражающие или прозрачные поверхности, и проследить путь этих лучей, чтобы выяснить цвет объекта, который они будут пересекать.

Общая идея алгоритма состоит в том, чтобы проследить за ходом лучей от глаза наблюдателя через каждый пиксель на экране и вычислить цвет пикселя. Каждый луч проверяется на пересечение с объектом. После того, как определяется пересечение с ближайшим объектом, происходит поиск источника света до этой точки и анализ материала, из которого состоит объект. После анализа всех данных свойств определяется цвет пикселя. Идея довольно проста, однако вычислительная сложность данного алгоритма равна количеству пикселей на экране, умноженному на количество объектов.

Алгоритм трассировки лучей более подробно можно описать следующими действиями:

1. Для каждого пикселя трассируем первичный луч в направлении V к первой видимой поверхности.

2. Для каждого пересечения трассируем вторичные лучи:

-Лучи L в направлении источников света

-Отраженные лучи в направлении R

-Лучи преломления или проходящие лучи T

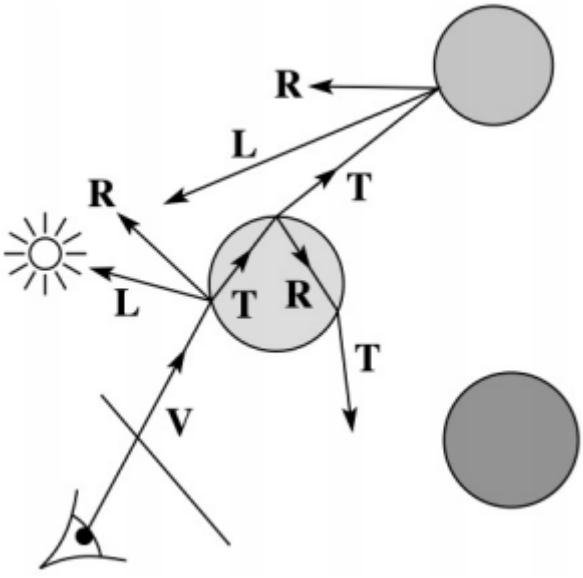

Рис. 5 - Общий вид алгоритма трассировки лучей

Для большей наглядности так же покажем алгоритм на рисунке пошагово:

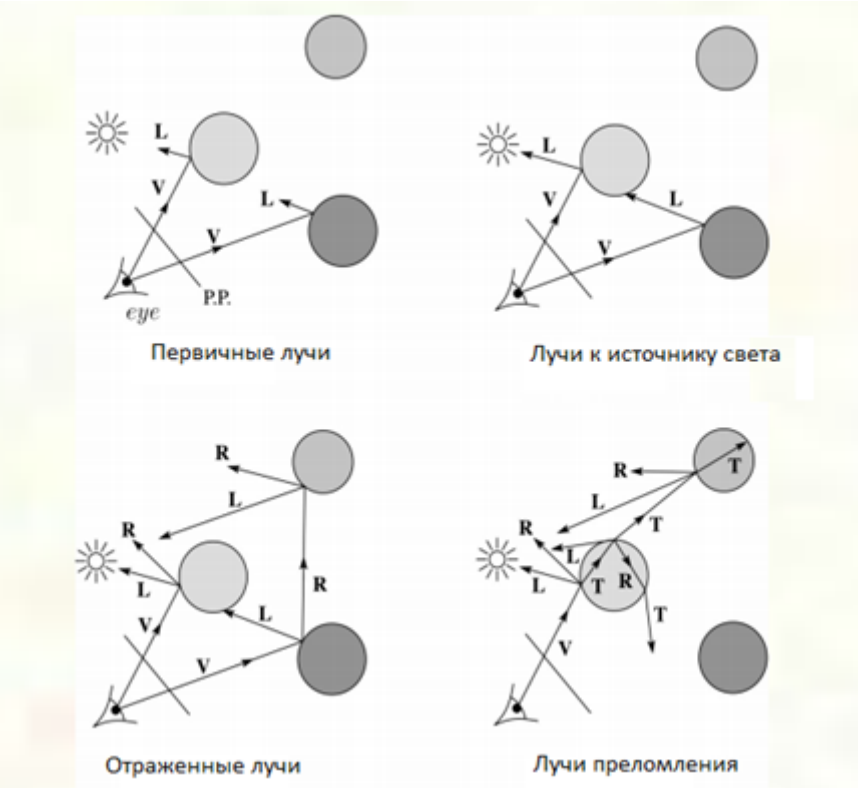

Рис. 6 - Пошаговая демонстрация алгоритма трассировки лучей.

Рассмотрим рекурсию алгоритма. Каждый раз при пересечении луча генерируются два новых луча (отражение и луч преломления). Если эти лучи пересекают другой прозрачный объект, каждый луч будет генерировать еще два луча. Этот процесс может продолжаться вечно, пока вторичные лучи пересекают отражающие или прозрачные объекты. Самое простое решение заключается в том, чтобы ограничить максимальное количество разрешенных рекурсий (или максимальную глубину). Как только максимальная глубина достигнута, мы просто перестаем генерировать новые лучи.

4.2 Книга *Боресков А.В. Программирование компьютерной графики. Современный OpenGL.*

В книге рассматривается алгоритм обратной трассировки лучей. Этот метод фактически моделирует то, каким образом возникает изображение в глазу наблюдателя. Он позволяет строить изображения с точным учетом таких явлений, как преломление и отражение.

Таким образом, мы можем поступить так: выпустим из глаза наблюдателя (камеры) луч через каждый пиксель экрана и отследим, какая яркость (энергия) приходит вдоль этого луча. Для этого мы отследим распространение луча в направлении, противоположном направлению распространения световой энергии (отсюда и термин – обратная трассировка).

Мы отслеживаем луч из глаза наблюдателя до его ближайшего пересечения с каким-либо объектом сцены. Далее мы оцениваем световую энергию, падающую в точку пересечения, и, зная BRDF (двулучевая функция отражательной способности), определяем, какая ее доля уйдет по оттрассированному нами лучу. В общем случае, для того, чтобы определить световую освещенность, падающую в заданную точку, мы должны найти интеграл по всей верхней полусфере (или даже по всей сфере в случае прозрачного материала). Поскольку это тяжело реализуемо с практической точки зрения, то обычно выделяют несколько ключевых направлений. Вдоль этих направлений выпускаются лучи, определяющие приходящую в точку световую энергию.

Обычно накладывается ряд ограничений, часть из которых мы в дальнейшем снимем. Основным подобным ограничением является то, что мы считаем, что свет, непосредственно падающий от источника света (первичное освещение), вносит гораздо больший вклад, чем вторичное освещение. Также мы считаем, что все источники света являются точечными (или направленными).

В результате мы из точки выпускаем лучи ко всем источникам света и проверяем, не закрывает ли какой-то объект данный источник света. При этом мы считаем, что абсолютно не важно, прозрачен закрывающий объект или нет, – он все равно закрывает нужную нам точку от заданного источника света.

Подобная «дискриминация» прозрачных объектов связана с тем, что для прозрачных объектов все довольно сложно. В качестве примера представьте себе стеклянный шар, закрывающий от источника света точку. Этот шар на самом деле будет работать как линза, и в результате в каких-то «закрытых» им точках света будет очень мало, а в каких-то окажется, наоборот, очень много.

Поскольку в рамках обратной трассировки точно рассчитать это оказывается крайне сложно, то мы просто будем считать, что каждый объект всегда полностью закрывает источник света.

Также мы выпускаем отраженный луч и (если объект прозрачен) преломленный луч. Мы трассируем эти лучи, для того чтобы определить долю световой энергии, проходящей вдоль них.

Фактически мы получаем рекурсивный алгоритм, так как для расчета света, проходящего вдоль отраженного и преломленного лучей, нам опять необходимо выполнить трассировку лучей.

В силу такой рекурсивной природы алгоритма необходимо ввести некоторый критерий прекращения рекурсии. Обычно для этого одновременно используются сразу два таких критерия – по глубине рекурсии и по вкладу луча в готовое изображение.

Первый критерий вводит ограничение на максимальную глубину рекурсии и не выпускает отраженные и преломленные лучи, если максимальная глубина уже достигнута.

Второй критерий связан с тем, что вклад отраженного луча в итоговое значение цвета пиксела уменьшается с увеличением глубины рекурсии. Поэтому в ходе трассировки лучей отслеживается, какой максимальный вклад может внести луч, и если этот вклад меньше заданной погрешности, то луч дальше не трассируется. Можно показать, что на самом деле существуют такие конфигурации, когда каждый из этих критериев приводит к заметной ошибке. Однако на практике оба этих критерия отлично работают, и мы будем использовать их оба.

При написании трассировщика лучей будем использовать следующее уравнение освещенности:

$\mathrm{I} = I_a \cdot k_a \mathrm{C} + k_d \cdot \mathrm{C} \sum I_i \max(0, (\mathrm{n}, l_i)) + k_s \sum I_i \cdot f_r(\mathrm{n}, l_i, \mathrm{v}) + k_r \cdot I_r \cdot F_r(\theta_r) + k_t \cdot I_t (1 - F_r(\theta_r))$.

Через f_r обозначена BRDF (двулучевая функция отражательной способности) поверхности в точке попадания луча, через l_i и I_i – единичное направление на i-й источник света и его яркость/цвет, C – цвет поверхности в точке, n – нормаль к поверхности в точке. Через I_r и I_t обозначена световая энергия, проходящая вдоль отраженного и преломленного лучей. Коэффициенты k_a, k_d, k_s, k_r и k_t характеризуют поверхность в точке (и зависят от этой точки).

5 Результаты практики

В процессе изучения теоретической части была предпринята попытка реализовать обратную трассировку лучей.

В результате работы опирался на вышеразобранную книгу, в итоге был реализован трассировщик лучей на языке C++ с использованием OpenGL Mathematics (GLM).

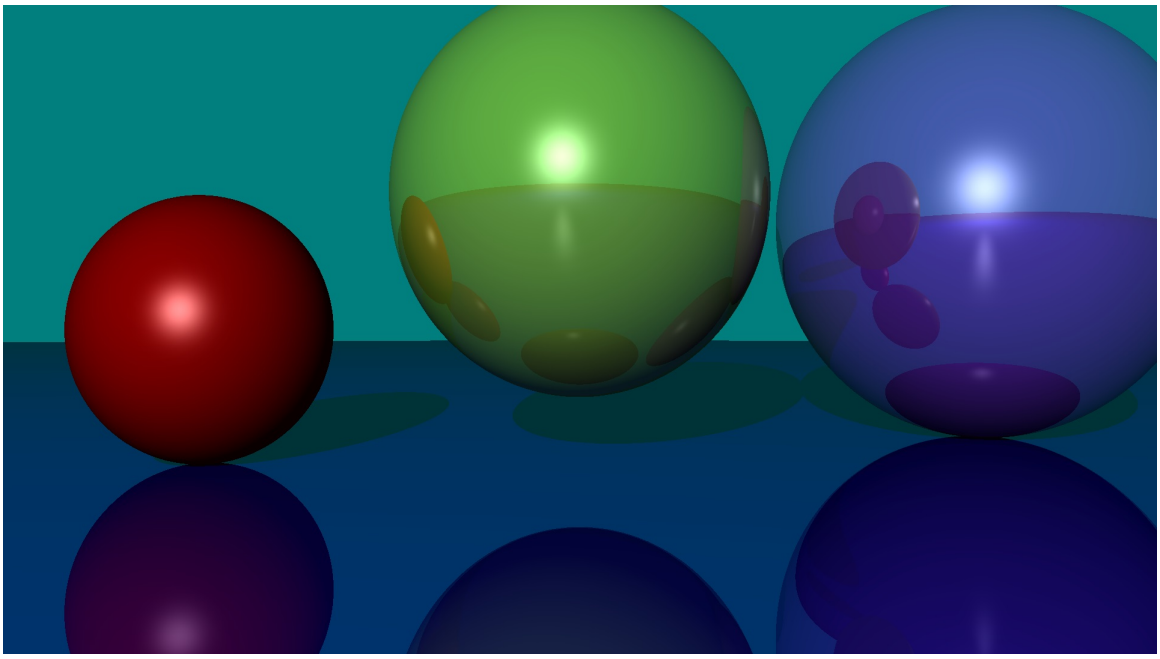

Рис. 7- Полученное изображение №1.

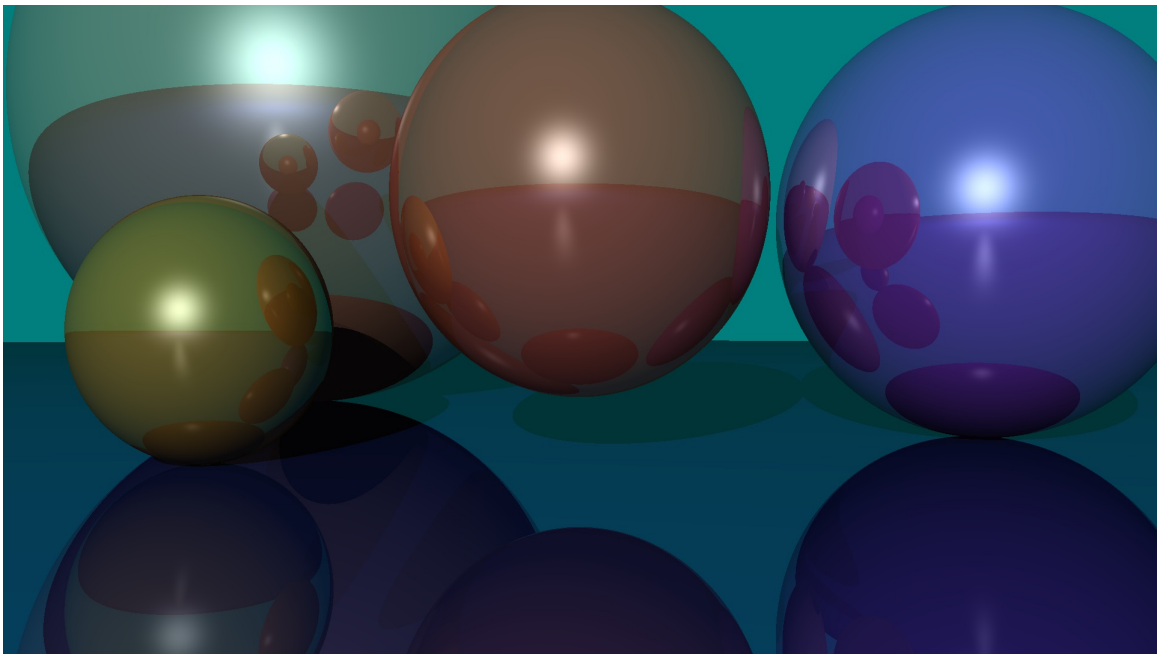

Рис. 8 - Полученное изображение №2.

6 Заключение

В ходе проделанной работы были достигнуты следующие цели:

1. Изучен алгоритм обратной трассировки лучей
2. Получены основные положения из теории
3. Применены полученные знания на практике

Трассировщик лучей, который мне удалось реализовать, позволяет строить изображения с базовым уровнем реалистичности и рассчитывать такие эффекты как тени, отражения и преломления. Но реализация является достаточно медленной, что не позволяет ее использовать в графике реального времени.

7 Список литературы

1. Боресков А.В. Программирование компьютерной графики. Современный OpenGL. – М.: ДМК Пресс, 2019. – 372 с.

2. Whitted T. An improved illumination model for shaded display. Communications of the ACM. – 1980. – June. – Vol. 23, no. 6. – Pp. 343–349.

3. Cook, R. L., Porter, T., and Carpenter, L. Distributed ray tracing. SIGGRAPH Comput. Graph. 18, 3 (Jul. 1984), 137-145.

4. Боресков А.В. Графика трехмерной компьютерной игры на основе OpenGL. - М.: ДИАЛОГ-МИФИ, 2004. - 384 с.

5. Эдвард Эйнджел. Интерактивная компьютерная графика. Вводный курс на базе OpenGL, 2 изд.- М.: Издательский дом "Вильяме 2001. - 592 с

6. Glassner, A. (ed) An Introduction to Ray Tracing. Academic Press New York, N.Y. 1989, pp. - 353.

7. Турлапов В.Е., Боголепов Д.К. Элементы глобального освещения сцены. Классическая трассировка лучей. [Электронный ресурс]:
URL: http://www.graph.unn.ru/rus/materials/CG/CG15_RayTracing.pdf

8. Ray tracing(graphics) [Электронный ресурс]: Википедия. Свободная энциклопедия.
URL: http://en.wikipedia.org/wiki/Ray_tracing_(graphics)

9. An Overview of the Ray-Tracing Rendering Technique. [Электронный ресурс]:
URL: https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-overview/light-transport-ray-tracing-whitted

10. Brian Caulfield. What's the Difference Between Ray Tracing and Rasterization?
URL: https://blogs.nvidia.com/blog/2018/03/19/whats-difference-between-ray-tracing-rasterization/

11. learnopengl. Урок 1.9 — Камера. Автор: 0xEEd. [Электронный ресурс]:
URL: https://habr.com/ru/post/327604/