Санкт–Петербургский государственный университет

БУХАЛОВ Максим Владимирович

Выпускная квалификационная работа

Методы рендеринга сцены на основе алгоритма трассировки лучей

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2016 «Прикладная математика, фундаментальная информатика и программирование»
Профиль «Информационные системы»

Научный руководитель:
доцент, кафедра технологии программирования,
к.т.н. Блеканов Иван Станиславович

Рецензент:
доцент, кафедра компьютерных технологий
и систем, к.ф. - м.н. Коровкин Максим Васильевич

Санкт-Петербург
2020 г.

Содержание

Введение

Актуальность

С развитием киноиндустрии, игровой индустрии и дизайна все больше растет потребность в реалистичной компьютерной графике. Долгое время вычислительные способности компьютеров позволяли использовать только метод растеризации треугольников [18, 15], что сказывалось на реалистичности получаемого результата, так как этот метод - очень грубая аппроксимация того, как свет на самом деле устроен. Однако компьютерная мощь растет с каждым годом, что позволяет использовать другие методы [16, 17], которые раньше использовать не представлялось возможным, ввиду вычислительной трудозатратности. Один из таких методов - метод трассировки лучей.

Метод трассировки лучей представляет собой метод генерации изображения путем отслеживания, или трассировки, пути света и моделирования эффектов его столкновения с объектами [20]. Он известен своей гибкостью, как правило, более достижимом реализмом и масштабируемостью, однако, как правило, достаточно трудозатратен [19].

В данной работе поставлена задача реализовать метод трассировки лучей, не акцентируя внимание на оптимизации данного метода, хотя некоторые вопросы оптимизации все же будут затронуты.

Цель работы

Целью данной работы является изучение и реализация метода трассировки лучей.

Задачи работы

Для достижения этой цели в рамках работы были поставлены следующие задачи:

- Определить и реализовать набор примитивных объектов, доступных для конструирования сцены.

- Определить и реализовать набор свойств поверхности примитивных объектов таких как:
  - Рассеивание лучей и затухание света
  - Текстуры
  - Свойство излучать свет
- Получить изображение, сгенерированное данным методом

Практическая значимость

Одна из самых крупных потребителей реалистичной графики - индустрия развлечений, а именно игровая индустрия и киноиндустрия.

Реалистичность спецэффектов в кино помогают людям создавать такие кадры, какие раньше не представлялось возможным. Также графика в разы удешевила съемочный процесс, что сделало кино более доступным для производства.

В игровой индустрии графика все ближе приближается к реалистичной, что позволяет делать их более правдоподобными, стирая барьер между игроком и игрой.

Если отойти от индустрии развлечений, то реалистичная графика пользуется спросом у дизайна. От дизайна интерьера квартиры, до дизайнов крупных продуктов и целых зданий. Возможность посмотреть будущий интерьер еще не отремонтированной квартиры, посмотреть еще на не выпущенную марку автомобиля или устроить экскурсию по какому-либо еще недостроенному объекту позволяет многим людям экономить время и деньги.

Глава 1. Обзор литературы

1.1 Обзор существующий методов

Метод бросания лучей (ray casting) - один из первых методов данного типа. Впервые данный термин был использован в 1982 году Скоттом Роттом [1]. Данный метод испускает только первичные лучи, что делает его нерекурсивным. Впервые в массовой индустрии данный алгоритм был применен Джоном Кармаком в игре Wolfenstein 3D [2]. Данный метод был достаточно прорывным для своего времени.

Метод трассировки лучей (ray tracing) [3] же является рекурсивным алгоритмом, что делает его более затратным в вычислительном плане алгоритмом. Однако, благодаря тому, что данный метод испускает не только первичные лучи, но и вторичные, делает данный метод заметно более реалистичным и позволяет реализовать многие эффекты, которые недоступны методу бросания лучей, таких как отражения, преломления, мягкие тени и другие. Однако в данном методе присутствуют некоторые допущения, что делает данный метод менее реалистичным, чем его усовершенствованная версия.

Более усовершенствованный метод называется методом трассировки пути (path tracing) [5]. Данный метод основывается на так называемом уравнении рендеринга, предложенном в статье Джеймса КаджWF-SP800Nии в 1982 году [4], и его решении методом Монте-Карло. Данный алгоритм является методом без допущений (unbiased method) [6], что делает его самым реалистичным методом из трех.

1.2 Обзор программных инструментов для работы с реалистичной компьютерной графикой

На данный момент одни из самых популярных инструментов для работы с реалистичной компьютеной графикой:

1. *Octane*[21]. Пользуется большим спросом в киноиндустрии для создания графики. Достаточно хорошо оптимизирован и имеет удобный интерфейс взаимодействия, однако является проприетарным, что очень

ограничивает его использование непроффесионалам.

2. *Mitsuba*[22]. Данный инструмент больше ориентировани на исллледователей. Имеет открытый исходный код, хорошо оптимизирован. Не совсем подходит для профессионального производства реалистичной графики. Инструмент достаточно сложен для освоения.

3. *Cycles*[23]. Данный инструмент разрабатывается *Blender Foundation*[24]. Данный инструмент имеет открытый исходный код. Также он встроен в *Blender* - инструмент для 3D моделирования и анимации, также инструмент с открытым исходным кодом, что делает его очень доступным для освоения. Однако данный инструмент достаточно молодой, поэтому он имеет не такой широкий функционал и еще не так хорошо оптимизирован.

Глава 2. Теоретические аспекты работы с графическими объектами

2.1 Примитивные объекты

Для определения примитивного объекта мы должны определить метод пересечения данного объекта с лучем. Имея луч и объект мы хотим уметь получать следующую информацию:

- Пересекает ли данный луч объект или нет
- Точку пересечения
- Нормаль в точке пересечения

Данной информации будет достаточно, чтобы повторно запустить луч из точки пересечения, тем самым имитируя поведение лучей света [12]. Луч будем определять с помощью двух векторов: $R(t) = O + t \cdot D$, где O - вектор начало луча, D - вектор направления луча, t - скаляр, который считаем неотрицательным.

2.1.1 Сфера

Сфера задается уравнением $S(x) = (x - C, x - C) = r^2$, где C - цетр сферы, r - радиус, x - точка, принадлежащая сфере. Пересечение с лучем считается следующим образом:

$S(R(t)) = (R(t) - C, R(t) - C) = r^2;$
$(O + t \cdot D - C, O + t \cdot D - C) = r^2;$
$(D, D)t^2 + 2(D, CO)t + (CO, CO) - r^2 = 0$, где $CO = O - C;$
$t^*_1 = -(D, OC) - \sqrt{(D, CO)^2 - (CO, CO) + r^2};$
$t^*_2 = -(D, OC) + \sqrt{(D, CO)^2 - (CO, CO) + r^2},$
учитывая, что D нормализован.

Если $discr = (D, OC)^2 - (OC, OC) + r^2 \geq 0$ и хотя бы один из t_1 или t_2 неотрицателен, значит, пересечение есть, выбираем наименьший из неотрицательных t, обозначим его за t^*, тогда точка пересечения равна $R(t^*) = O + t^* \cdot D$ Нормаль же тогда равна $N = R(t^*) - C = O + t^* \cdot D - C$.

Рис. 1: Рендер сферы.

2.1.2 Плоскость

Плоскость задается уравнением $P(x) = (p - p_0, N) = 0$, где p_0 - точка, принадлежащая прямой, N - нормаль плоскости. Тогда пересечение луча

$R(t) = O + t \cdot D$ находится следующим образом:

$$P(R(t)) = (R(t) - p_0, N) = 0;$$
$$(O + t \cdot D - p_0, N) = 0;$$
$$t = \frac{(p_0 - O, N)}{(D, N)}.$$

Если $(D, N) \neq 0$ и $t^* \geq 0$, то считаем, что пересечение есть. Нормаль в точке пересечения равна нормали плоскости.

Рис. 2: Рендер плоскости.

2.1.3 Треугольник

Треугольник определятся тремя точками A, B, C. Искать пересечение с лучем $R(t) = O + t \cdot D$ будем методом Моллера — Трумбора [3]. Данный метод примечателен тем, что он не требует предварительного вычисленяе уравнения плоскости, содержащей треугольник. Вычисление пересечения выглядит следующим образом. Рассмотрим барицентрические координаты точки внутри треугольника $P = w \cdot A + u \cdot B + v \cdot C$. Учитывая, что $u + v + w = 1$, выразим $w = 1 - u - v$. Подставляя в выражение точки P получаем:

$$P = (1 - u - v) \cdot A + u \cdot B + v \cdot C;$$
$$P = A + u \cdot AB + v \cdot AC, \text{ где } AB = B - A, AC = C - A;$$

Подставляя $R(t) = O + t \cdot D$ в уравнение выше получаем:

$$O + t \cdot D = A + u \cdot AB + v \cdot AC;$$
$$AO = -t \cdot D + u \cdot AB + v \cdot AC, \text{ где AO = O - A;}$$
$$\begin{bmatrix} -D & AB & AC \end{bmatrix} \cdot \begin{bmatrix} t \\ u \\ v \end{bmatrix} = AO;$$

Используя метод Крамера, мы получаем:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{\begin{vmatrix} -D & AB & AC \end{vmatrix}} \begin{bmatrix} \begin{vmatrix} AO & AB & AC \end{vmatrix} \\ \begin{vmatrix} -D & AO & AC \end{vmatrix} \\ \begin{vmatrix} -D & AB & AO \end{vmatrix} \end{bmatrix};$$

При этом, если $\begin{vmatrix} -D & AB & AC \end{vmatrix} = 0$, то это означает, что пересечения нет. Учитывая, что $\begin{vmatrix} V_1 & V_2 & V_3 \end{vmatrix} = V_1 \cdot (V_2 \times V_3)$ и $AB \times AC = N$ - нормаль треугольника, получаем:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(D,N)} \begin{bmatrix} (AO, N) \\ -D \cdot (AO \times AC) \\ -D \cdot (AB \times AO) \end{bmatrix};$$

Обозначим найденное значение t за t^*, тогда треугольник и луч имеет пересечение в $R(t^*)$, если $t^* \geq 0, u + v <= 1$ и $u \geq 0, v \geq 0$. Нормаль в точке пересечения равна $N = AB \times AC$

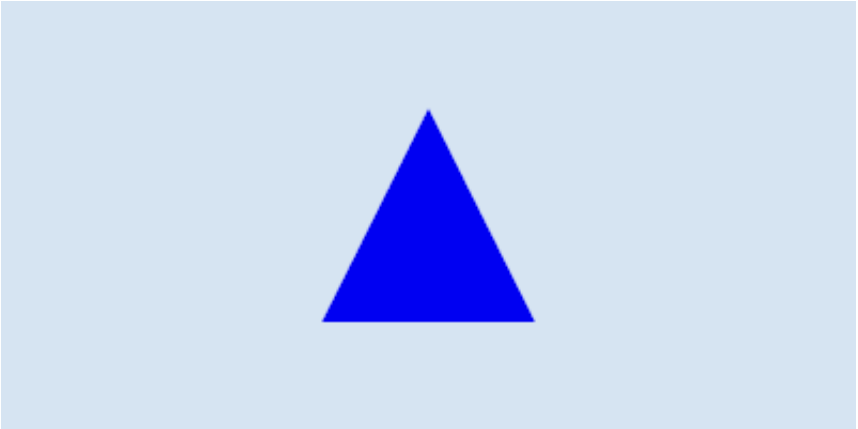

Рис. 3: Рендер треугольника.

2.1.4 Четырехугольник

Четырехугольник задается четыремя точками A, B, C, D. Пересечение между лучем и четырехугольником будем находить, разбив четырехугольник на 2 треугольника с вершинами A, B, C и A, C, D. Если луч пересекает хотя бы один из треугольников, то пересечение имеет место быть. Нормаль можно найти с помощью любых трех вершин: $N = AB \times AC$, где $AB = B - A$, $AC = C - A$

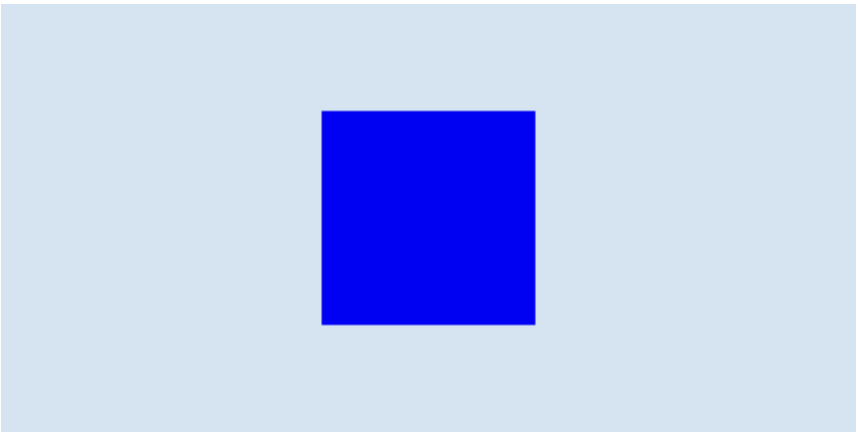

Рис. 4: Рендер четырехугольника.

2.2 Структура сцены

Самый наивный способ хранить объекты сцены - хранить их в массиве. Однако этот метод имеет множество недостатков, таких как:

- Время рендера сцены растет с количеством объектов, независимо от того, является ли данный оюъект видимым или нет.
- Каждый раз луч будет проверятся на пересечение со всеми объектами, даже если луч на самом деле не пересекает никакой из объектов.

Хранение объектов в иерархии ограничивающих объемов [9] решает данные проблемы. Это древовидная структура данных, в которой хранятся ограничивающие объемы, обычно это параллелепипеды, параллельные осям, так как проверять луч на пересечение с такими параллелепипедами достаточно просто. Данные параллелепипеды могут содержать в себе объекты или другие параллелепипеды.

Параллелепипед, параллельный осям, задается двумя точками B^{min}, B^{max} - вершина с минимальными координатами и максимальными соответственно. Проверка на пересечение с лучем $R(t) = O + t \cdot D$ выполняется следующим образом. Проверяется пересечение по каждой оси. Рассмотрим пример для осей x, y:

$$
\begin{aligned}
&O_x + t \cdot D_x = B_x^{min}; O_x + t \cdot D_x = B_x^{max};\\
&O_y + t \cdot D_y = B_y^{min}; O_y + t \cdot D_y = B_y^{max};\\
&t_{1x} = \frac{B_x^{min} - O_x}{D_x}; t_{2x} = \frac{B_x^{max} - O_x}{D_x};\\
&t_{1y} = \frac{B_y^{min} - O_y}{D_y}; t_{2y} = \frac{B_y^{max} - O_y}{D_y};\\
&t_x^{min} = min\{t_{1x}, t_{2x}\}; t_x^{max} = max\{t_{1x}, t_{2x}\};\\
&t_y^{min} = min\{t_{1y}, t_{2y}\}; t_y^{max} = max\{t_{1y}, t_{2y}\};
\end{aligned}
$$

Если пересекает, то $t_x^{min} <= t_y^{max}$ и $t_y^{min} <= t_x^{max}$, при условии, что $t_y^{max}, t_x^{max} \geq 0$. Добавим проверку по z и совместим их вместе, получаем:

$t = \frac{B^{min} - O}{D}$;
$T = \frac{B^{max} - O}{D}$, здесь отношение векторов - поэлементное деление;
$t^{max} = max\{t_x, t_y, t_z\}$;
$T^{min} = min\{t_x, t_y, t_z\}$;

Если $t^{max} > T^{min}$ или $T_{min} < 0$, то луч не пересекает параллелепипед.

Рассмотрим метод построения дерева. Метод выглядит следующим образом: пусть дана константа N^{max} - максимальное количество элементов в одном ограничивающем объеме (подбирается эмпирически) и массив объектов M^{obj} выполним:

1. Поместим каждый объект в свой ограничивающий объем.
2. Поместим данные объемы в общий, назовем его корнем.
3. Если количество объектов в текущем меньше N^{max}, прекращаем деление.
4. Иначе отсортируем объемы по расстоянию от начала координат до B^{max}, тем самым при разбиении на дочерние объемы мы уменьшим степень перекрытия объемов.
5. Первую половину объектов помещаем в один дочерний объем, вторую в другой.
6. Повторяем шаги 3, 4, 5 для дочерних объемов.

2.3 Материалы

Материал поверхности определяет то, как луч отражается от поверхности в данной точке и то, как меняется цвет луча при отражении [12]. Обычно изменение цвета луча при отражении задается таким параметром как альбедо. Это может быть как и константный цвет, что означает, что поверхность отражает определенно заданый спектр цвета и поглощает все другие, либо это может быть текстура, которая задает такой параметр в конкретной точке поверхности. Подробнее о текстурах будет изложено далее. Способ отражения луча от поверхности, наоборот, сильно зависит от материала, и в основном является определяющим фактором.

Рассотрим три типа материалов:

1. Диффузный материал
2. Металический материал
3. Прозрачный материал

2.3.1 Диффузный материал

Диффузный материал имитирует шероховатые поверхности, в которых из-за неровностей лучи отражаются в разных направлениях. Направление отраженного луча выбирается случайно в единичной сфере с центром в N, где N - нормаль в данной точке. Отраженный луч $R'(t) = O' + t \cdot D'$ находится следующим образом :

$$
\begin{aligned}
&x = Rand(0, 2\pi);\\
&z = Rand(-1, 1);\\
&r = \sqrt{1 - r^2};\\
&D' = N + \{rcos(x), rsin(x), z\};\\
&O' = H;
\end{aligned}
$$

$Rand(x, y)$ - функция получения случайных чисел из равномерного распределения.

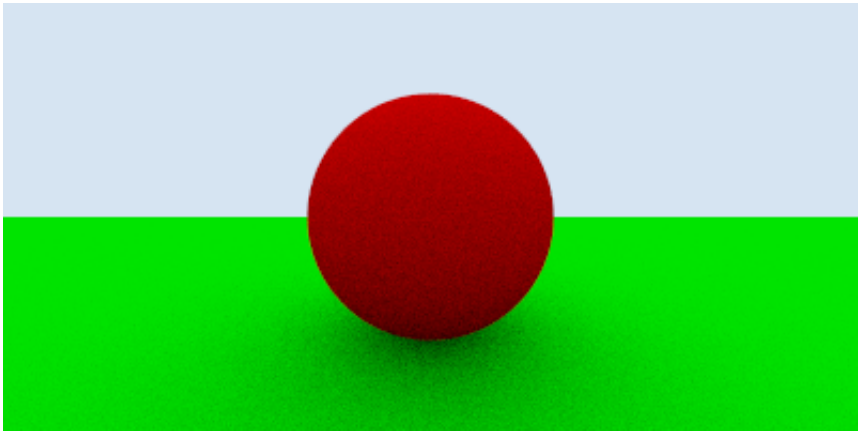

Рис. 5: Пример диффузного материала.

2.3.2 Металлический материал

Металлический материал имитирует гладкие поверхности, то есть отраженный луч находится в одной плоскости с падающем лучем и нормалью и угол между отраженным лучем и нормалью равен углу между падающим углом и нормалью. Отраженный луч $R'(t) = O' + t \cdot D'$ находится следующим образом :

$$D' = D - 2(D, N) \cdot N;$$
$$O' = H;$$

Мы можем добавить шероховатости материалу, добавив к D' случайный вектор из единичной сферы P_{rand} умноженный на некоторый коэффициент, определяющий степень шероховатости C_{rough}. Тогда гладкий материал будет иметь $C_{rough} = 0$. В общем случае D' будет иметь вид:

$$D' = (D - 2(D, N) \cdot N) + C_{rough} \cdot P_{rand};$$

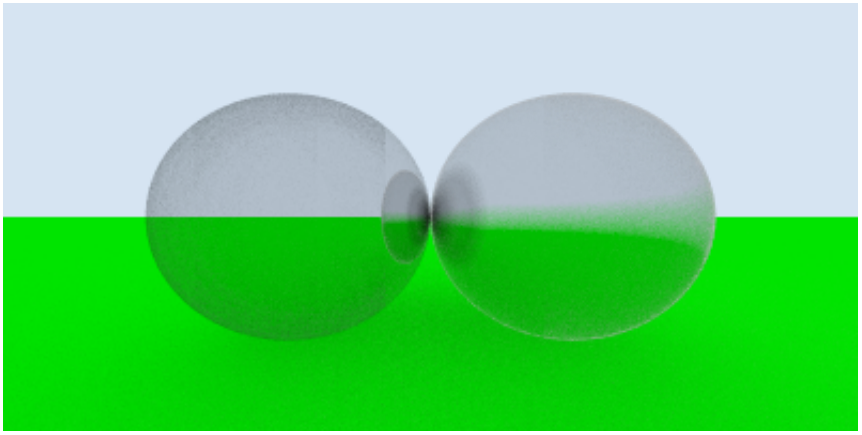

Рис. 6: Пример металического материала.

2.3.3 Прозрачный материал

В прозрачном материале лучи могут как и отражаться, так и преломляться. Преломление лучей описывается законом Снелла [10]. Выглядит он следующим образом: $n_1 \sin\phi_1 = n_2 \sin\phi_2$, где n_1 - показатель преломления среды, из которой свет падает на границу раздела, n_2 - показатель преломления среды, в которую свет попадает, пройдя границу раздела, ϕ_1 - угол падения света — угол между падающим на поверхность лучом и нормалью к поверхности, ϕ_2 - угол преломления света — угол между прошедшим через поверхность лучом и нормалью к поверхности. Преломленный луч $R_{refract}(t)$ находится следующим образом:

$$
\begin{aligned}
&k = 1.0 - (\tfrac{n_1}{n_2})^2(1.0 - (N, D)^2);\\
&D' = \tfrac{n_1}{n_2} \cdot D - (\tfrac{n_1}{n_2}(N, I) + \sqrt{k})) \cdot N;\\
&O' = H;
\end{aligned}
$$

В случае, когда $\frac{n_1}{n_2} \sin\phi_1 \geq 1$ или $k < 0$ имеет место только отражение, преломления не происходит. Для определения отражения или преломления будем использовать приближения Шлика для уравнения Френеля [11]. Аппроксимация выглядит следующим образом: $F = n + (1 - n)(1 - (N, -D))^5$, $n = \frac{n_1}{n_2}$ На практике это будет означать, что отражение или преломление луча будет выбираться случайным образом с поправкой на коэффициент, полученный из аппроксимации Шлика. Имея n_2, $n_1 = 1$ и $R(t) = O + t \cdot D$, получание нового

луча $R'(t)$ выглядит следующим образом:

Если $(n, D) < 0 \implies n = \frac{1}{n_2}$, иначе $n = n_2$;
$F = \frac{n_1}{n_2} + (1 - \frac{n_2}{n_1})(1 - (N, -D))^5$;
Если $Rand(0, 1) < F \implies R'(t) = R_{refract}(t)$, иначе $R'(t) = R_{reflect}(t)$;

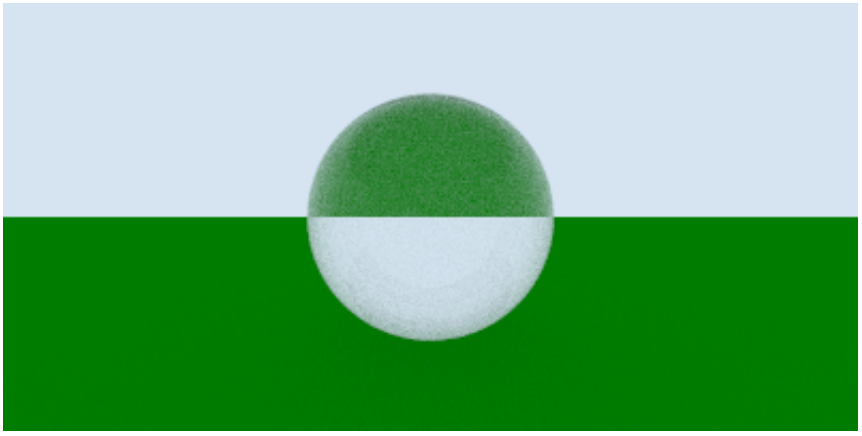

Рис. 7: Пример прозрачного материала.

2.4 Текстуры

Текстуры в компьютерной графике обычно означают функцию, которая определяет цвет точки поверхности [13]. Это может быть процедурная генерация, так и сэмплирование изображения. Текстуры - один из способов задания альбедо в материале, они позволяют задавать в каждой точке свой парамтр альбедо, что является очень удобным в некоторых ситуациях.Точка поверхности задается так называемыми текстурными координатами (u, v), где $0 \leq u, v \leq 1$. Для получения значения цвета из текстуры, нужно для каждой точки поверхности однозначно определить текстурные координаты. Рассмотрим способ задания текстурных координат для прямоугольника и сферы.

Определение текстурных координат для прямоугольника достаточно просто. Имея четыре вершины A, B, C, D и считая, что вершины указаны в порядке обхода против часовой стрелки начиная с вершины с наименьшими координатами, для точки P имеем:

$$u = \left| \frac{DP_x}{AB_x} \right|;$$
$$v = \left| \frac{DP_y}{AD_y} \right|;$$

Для получения текстурных координат сферы рассмотрим сферические координаты. Для каждой точки можно единственным образом задать широту и долготу (θ, ϕ), где ϕ - угол между осью x и проекцией отрезка, соединяющего начало координат с точкой P, на плоскость xy, θ - угол между радиус-вектором точки P и плоскостью xy. При этом $\phi = \arctan \frac{y}{x}$, $\theta = \arcsin z$, $-\pi \leq \phi \leq \pi$, $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$. В итоге текстурная координата точки P бедет равна:

$$u = 1 - \frac{\phi+\pi}{2\pi}; v = \frac{2\theta+\pi}{2\pi};$$

2.4.1 Константная текстура

Константная текстура - самая простая текстура. Она задает одинаковый цвет для любой точки поверхности. Для данной текстуры не нужны текстурные координаты, что делает ее очень универсальной, однако вариативность использования ее довольно скудная.

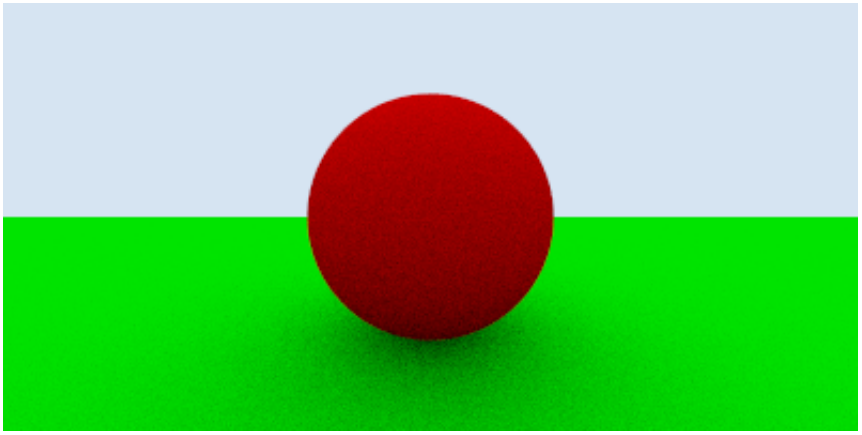

Рис. 8: Пример константной текстуры.

2.4.2 Шахматная текстура

Шахматная текстура - текстура состоящая из двух других текстур, сочетающих их в шахматном порядке. Пусть цвет одной текстуры вычисленный в координатах (u, v) равен Col_{even}, цвет другой, вычесленный в тех же координатах - Col_{odd}. Тогда цвет шахматной текстуры $Col_{checker}$, вычисленный в (u, v), равен:

$sign = sgn(\sin ku \sin kv)$;
$Col_{checker} = Col_{even}$, если $sign = 1$, иначе $Col_{checker} = Col_{odd}$;

Где k - параметр, определяющий насколько мелкими будут клетки.

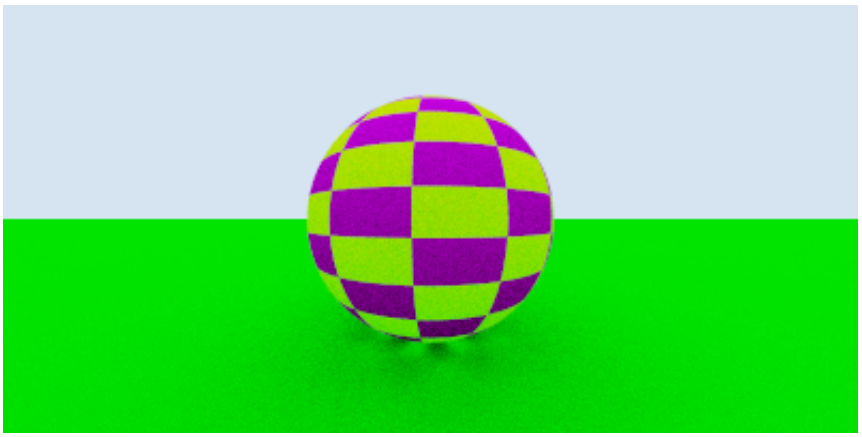

Рис. 9: Пример шахматной текстуры.

2.4.3 Текстура изображения

Для возможности использовать изображение в качестве текстуры, необходимо по текстурной координате определить координату пикселя. Пусть изобрадение имеет N_x на N_y пикселей. Тогда для пикселя с позицией (i, j), $0 \leq i \leq N_x$, $0 \leq j \leq N_y$ справедливо равенство:

$$u = \frac{i}{N_x - 1}; v = \frac{j}{N_y - 1};$$

Отсюда следует:

$$i = \lfloor u(N_x - 1) \rfloor; j = \lfloor (1 - v)(N_y - 1) \rfloor;$$

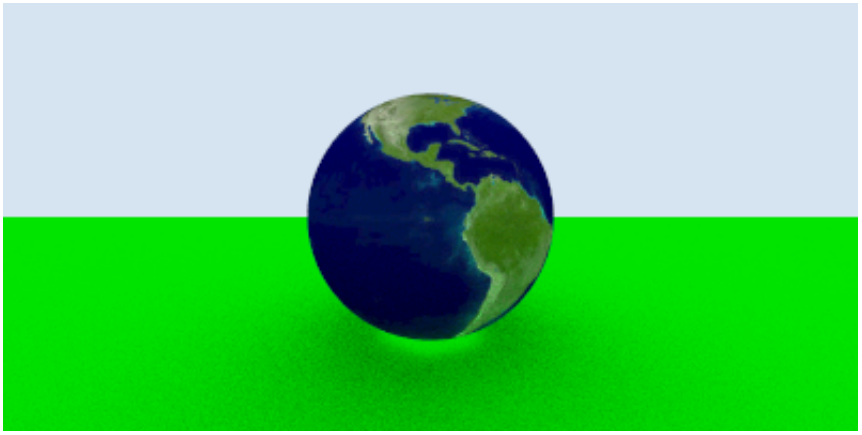

Рис. 10: Пример текстуры изображения.

2.5 Освещение

Освещение делится на две части - фоновое - освещение, которое уже присутствует в сцене, считается максимально рассеяным, то есть цвет постоянный для всей сцены, представляет из себя цвет пикселя, если луч, проходящий через него ничего не пересекает, и излучаемое - свет, излучаемый каким-либо объектом с материалом, способным излучать свет [13]. Описанные ранее типы материалов не излучали никакого света, рассмотрим другой тип материалов.

Рассмотрим материал диффузного света. Данный материал только излучает свет, не рассеивает лучи дальше. Данный материал очень простой в устройстве, все, что он имеет - это текстуру альбедо и интенсивность $I \geq 1$. Если луч пересекает объект с данным материалом, то луч окрашивается в цвет текстуры в данной точке, умноженный на интенсивность.

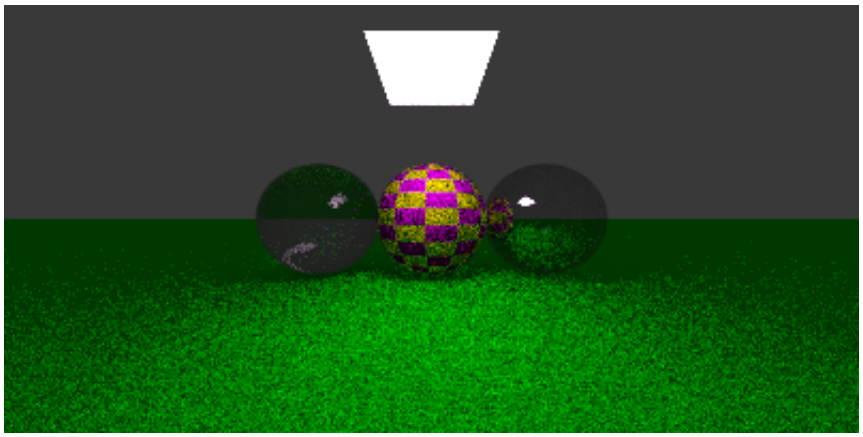

Рис. 11: Пример освещения.

Глава 3. Построение алгоритма

3.1 Описание алгоритма

Рассмотрим структуру алгоритма. Перед началом работы алгоритма должны быть инициализированы следующие параметры:

- H_{img}, W_{img} - высота и ширина итогового изображения в пикселях
- N_{spp} - количество лучей, пускаемых через каждый пиксель. Нужно для того, чтобы границы у итогового изображения были сглажены.
- N_{depth} - количество допустимых отражений луча. Необходимо для того, чтобы алгоритм остановился.
- $C_{ambient}$ - фоновый свет (цвет пикселя, при отсутствии пересечений луча и объектов)
- Cam - камера, у которой определены параметры, такие как:
  - O_{cam} - позиция
  - D_{cam} - направление
  - Up_{cam} - единичный вектор, который определяет направление вверх в пространстве камеры (для определения базиса пространства камеры)
  - FOV_{cam} - угол обзора
- $Scene$ - собранная сцена, у которой:
  - Указаны все объекты
  - У каждого объекта указан его материал
  - Построена иерархия ограничивающих объемов

Сам алгоритм состоит из двух функций - $Render$ и $CastRay$. $Render$ перебирает все пиксели и N_{spp} раз вызвает у пикселя функцию $CastRay$, которая пускает луч в заданном направлении и возвращает цвет. Функции выглядят

следующим образом:

Algorithm 1: CastRay

Input: $R(t), N_{depth}, Scene, C_{ambient}$

Output: C_{out}

$B_{res}, Material, P_{hit}, u, v, N = Intersect(R(t), Scene)$;

if $B_{res} = True$ **then**

  $C_{emit} = EmitMaterial(Material, u, v)$;

  $C_{attenuate}, R'(t) =$

  $ScatterMaterial(Material, R(t), u, v, P_{hit}, N)$;

  return $C_{emit} +$

  $+C_{attenuate} \cdot CastRay(R'(t), N_{depth} - 1, Scene, C_{ambient})$;

else

  return $C_{ambient}$;

end

Algorithm 2: Render

$i = 0; j = 0;$
while $i < H_{img}$ **do**
  while $j < W_{img}$ **do**
    $Col =$ **Black**;
    $n = 0;$
    while $n < N_{spp}$ **do**
      $u = \frac{j+Rand(0,1)}{W_{img}};$
      $v = \frac{i+Rand(0,1)}{H_{img}};$
      $R(t) = GetRayFromCam(u, v);$
      $Col = Col + ColorRay(ray, s_Depth);$
      $n = n + 1;$
    end
    $Col = \frac{Col}{N_{spp}};$
    $Img(i, j) = Col;$
    $j = j + 1;$
  end
  $i = i + 1;$
end
return $Img;$

3.2 Стек технологий

Для реализации данного метода был выбран язык C++ стандарта 2017 года, ввиду его производительности. Для реализации математических вычислений была выбрана библиотека glm [7], так как она отличается своей легковестностью и оптимизацией, так как данная библиотека поддерживает SIMD.

3.3 Тестирование и результаты работы

В качестве тестовой сцены будем использовать Cornell box [14]. Данная сцена достаточно часто применяется для тестирования методов рендеринга.

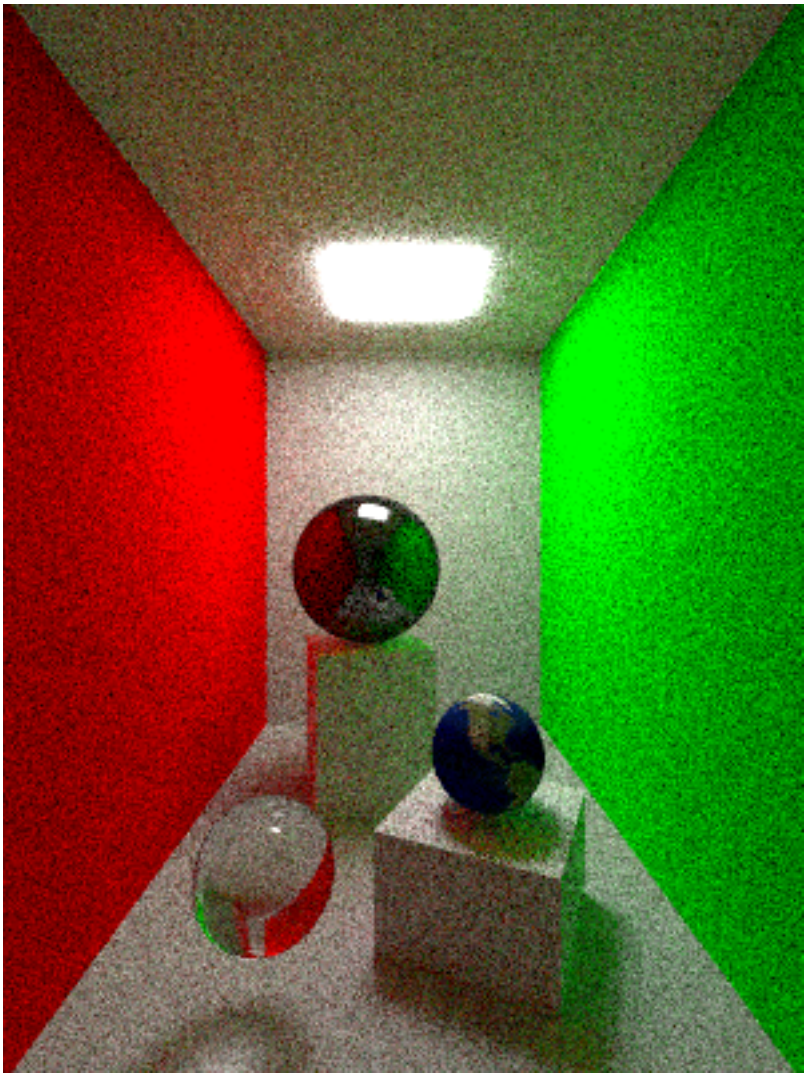

Рис. 12: Cornell box, сгенерированный данным методом

Заключение

Результаты работы

В ходе данной работы были получены следующие результаты:

- Определен и реализован набор примитивов, позволяющий собирать сцены для генерации изобрадений
- Определен и реализован набор свойств поверхности примитивов, таких как:
  - Материалы - отвечают за рассеивание лучей и затухание света
  - Текстуры
  - Специальный тип материала, способный излучать свет
- Получены изображения с помощью данного метода.
- Разработан и реализован алгоритм трассировки лучей. Исходный код доступен по ссылке: `https://github.com/Sphag/raytracer`

Перспективы развития

Есть три вектора, по которому можно развивать данный метод:

1. Оптимизация алгоритма. Использование как дополнительных потоков центрального процессора, так и использование граффического. Рассмотреть возможность применения аппаратных ускорителей, как, например, ядра видеокарт семейства Nvidia RTX [25].
2. Работа в сторону улучшения самого алгоритма. Данная реализация делает достаточное количество допущений о том, как свет на самом деле устроен. Уже существуют модификации, которые моделируют поведение света намного точнее, что делает их результат намного ближе к реализму.

3. Удобство использования. В данной реализации метода достаточно трудно настроить сцену пользователю, который не знаком с данной реализацией. Нет просмотра результата в окне в реальном времени, нет интерактивного редактора сцен. Удобство использования экономит много времени, что имеет большую роль в некоторых областях.

Список литературы

[1] Roth S. «Ray Casting for Modeling Solids». DOI: 10.1016/0146-664X(82)90169-1

[2] Premadi F. «Ray-Casting Tutorial For Game Development And Other Purposes». https://permadi.com/1996/05/ray-casting-tutorial-table-of-contents/

[3] Nikodym T. «Ray Tracing Algorithm For Interactive Applications». https://dip.felk.cvut.cz/browse/pdfcache/nikodtom_2010bach.pdf

[4] Kajiya J. T. «The rendering equation». DOI: 10.1.1.63.1402

[5] Lafortune E. «Mathematical Models and Monte Carlo Algorithms for Physically Based Rendering». http://www.graphics.cornell.edu/~eric/thesis/index.html

[6] Farnsworth M. «Biased vs Unbiased Rendering». http://renderspud.blogspot.com/2006/10/biased-vs-unbiased-rendering.html

[7] «OpenGL Mathematics (GLM)». https://glm.g-truc.net

[8] Moller T., Trumbore B. «Fast, Minimum Storage Ray/Triangle Intersecton». http://fileadmin.cs.lth.se/cs/Personal/Tomas_Akenine-Moller/code/raytri_tam.pdf

[9] Ericson C. «Real-Time Collision Detection». p. 238. ISBN 1-55860-732-3.

[10] «Snell's law». https://en.wikipedia.org/wiki/Snell%27s_law

[11] Schlick C. «An Inexpensive BRDF Model for Physically-based Rendering». http://www.cs.virginia.edu/~jdl/bib/appearance/analytic%20models/schlick94b.pdf

[12] Shirley P. «Ray Tracing in One Weekend». https://raytracing.github.io/books/RayTracingInOneWeekend.html

[13] Shirley P. «Ray Tracing the next week». https://raytracing.github.io/books/RayTracingTheNextWeek.html

[14] «Cornell box». https://en.wikipedia.org/wiki/Cornell_box

[15] Akeley K., Kirk D., Seiler L., Slusallek P., Grantham B.«When will ray-tracing replace rasterization?». DOI: 10.1145/1242073.1242120

[16] Lafortune E. P., Willems Y. D. «Bi-directional path tracing». http://graphics.cs.kuleuven.be/publications/BDPT/

[17] Hachisuka T., Ogaki S., Jensen H. W. «Progressive photon mapping». DOI: 10.1145/1457515.1409083

[18] Akenine-Möller T., Aila T. «Conservative and Tiled Rasterization Using a Modified Triangle Set-Up». DOI: 10.1080/2151237X.2005.10129198

[19] Georgiev I., Slusallek P. «Generic concepts for flexible and high performance ray tracing». DOI: 10.1109/RT.2008.4634631

[20] Glassner A. S. «An Introduction to Ray Tracing».

[21] Octane Renderer https://home.otoy.com/render/octane-render/

[22] Mitsuba Renderer https://www.mitsuba-renderer.org/

[23] Cycles Renderer https://www.cycles-renderer.org/

[24] Blender Foundation https://www.blender.org/foundation/

[25] Nvidia RTX https://www.nvidia.com/en-us/design-visualization/technologies/rtx/