СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компьютерная графика в наши дни является неотъемлемой частью нашей жизни. Она применяется везде, где нужно создание и обработка изображений, огромную роль играет компьютерная графика в рекламе и индустрии развлечений [1]. В результате появляется задача создания реалистичных изображений, в которых учитываются оптические эффекты преломления и отражения, а также текстура или цвет. Чем больше физических свойств сцены учитывается, тем более реалистичным получается результат.

Существует множетсво алгоритмов компьютерной графики, которые решают эту задачу. Однако построение более качественных изображений требует больше ресурсов (памяти и времени). Это становится более значимой проблемой при расчете динамических сцен в реальном времени.

Целью данной работы является разработка программного обеспечения для визуализации эффектов искажения изображения, получаемых с помощью двояковыпуклой линзы. Должны предоставляться возможности для изменения положения камеры и источника света, изменения спектральных характеристик источника.

Для достижения поставленной цели необходимо решить следующие задачи:

— рассмотреть существующие алгоритмы визуализации сцены, учитывающие такие свойства объектов как преломление и отражение света, выбрать достаточный для решения поставленной задачи;

— спроектировать выбранный алгоритм;

— разработать программное обеспечение и реализовать выбранные алгоритмы;

— провести исследование производительности полученного программного обеспечения.

1 Аналитическая часть

В данном разделе проводится описание объектов сцены, анализ существующих алгоритмов для решения поставленной задачи. В результате анализа выбирается алгоритм для дальнейшей реализации.

1.1 Формализация объектов синтезируемой сцены

Сцена состоит из следующего набора объектов:

1) точечный источник света — задается положением в пространстве, интенсивностью и цветом. В зависимости от характеристик источника определяется освещенность объектов сцены. Характеристики имеют значения по умолчанию, но имеется возможность их изменить;

2) камера — задается положением в пространстве, углом поворота вокруг каждой из трех координатных осей;

3) линза — задается диаметром линзы и радиусом кривизны обеих поверхностей;

4) объекты сцены — примитивы из заданного набора:

   — параллелепипед — задается длиной сторон по каждой координатной оси;

   — призма трехгранная — задается длиной стороны основания и высотой;

   — сфера — задается радиусом;

   — пирамида четырехгранная — задается длиной стороны основания и высотой.

Помимо специфичных для каждого примитива свойств, всем возможно задать положение в пространстве.

1.2 Физика линзы

Линзой называется прозрачное тело, ограниченное двумя сферическими поверхностями, линия центров этих поверхностей называется главной оптической осью линзы [2].

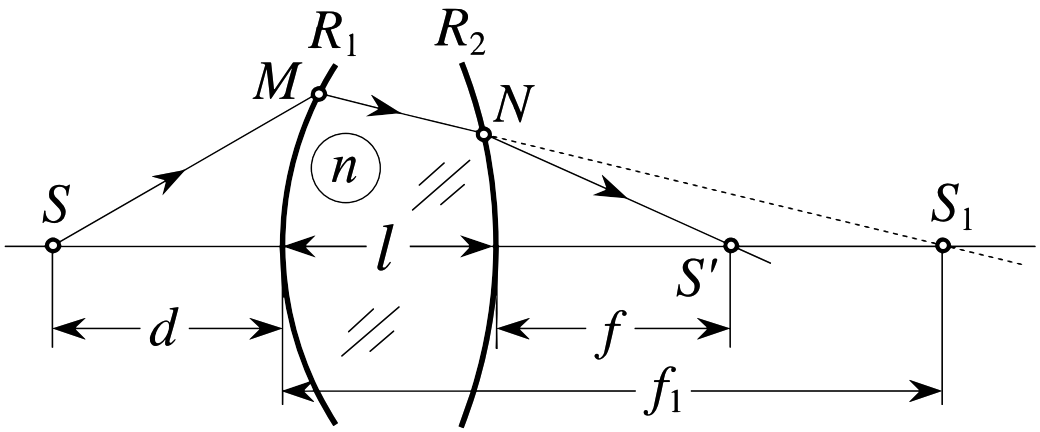

Рисунок 1 – Ход луча через линзу [2]

Рассмотрим точечный источник света S, расположенный вне линзы с показателем преломления n и находящийся на ее главной оптической оси (рисунок 1). Путь произвольного луча, исходящего от источника в направлении линзы, будет проходить через точки и N с двумя последовательными преломлениями, этот луч пересечет главную оптическую ось в точке S', таким образом S' будет действительным изображением источника. Используя законы оптики и полагая, что расстояние между поверхностями $l \to 0$ (пренебрежимо мало), можно получить так называемую *формулу тонкой линзы*:

$$\frac{1}{d} + \frac{1}{f} = (n-1)\left(\frac{1}{R_1} - \frac{1}{R_2}\right) \tag{1}$$

Формула тонкой линзы весьма применима в реальных расчетах, так как достаточно точно позволяет предсказать ход лучей при маленькой толщине линзы. Однако она не подходит для решения поставленной задачи, так как в общем случае линза не обязательно тонкая. Таким образом требуется просчитывать изображение с учетом преломления света через обе поверхности линзы.

1.3 Описание методов

1.3.1 Глобальное освещение

Разные материалы, используемые для создания объектов на сцене, взаимодействуют с освещением по-разному. В процессе этого взаимодействия часть энергии отражается, часть преломляется, а оставшаяся часть поглощается. Существует также сценарий, когда сам материал излучает свет. Когда луч света исходит от источника и попадает на поверхность объекта, возможно его отражение. Этот отраженный луч может затем попасть на поверхность другого объекта, претерпев переотражения. *Первичный луч* — тот, который напрямую исходит от источника света, в то время как *вторичный луч* — тот, который претерпел одно или несколько отражений. Построение физической модели света, учитывающей только первичное освещение (первичные лучи), может существенно снизить качество синтезируемых изображений.

Ещё одним примером упрощения физической модели является отказ от учета теней в сцене. Модели, которые игнорируют этот процесс, называются *локальными*. В противоположность им, модели, учитывающие передачу света между поверхностями, обозначаются как *глобальные* или *модели глобального освещения* [3].

Диффузное отражение

Пусть луч падает в точку P в направлении i и отражается в направлении r, преломляется в направлении t, а вектором нормали к поверхности является n (рисунок 2). Все вектора единичные.

Идеальное диффузное отражение описывается законом Ламберта, который утверждает, что свет, падающий на поверхность, равномерно рассеивается во всех направлениях. Таким образом, нет определенного направления, в котором отражается луч свет, все направления считаются равными, и освещенность точки пропорциональна доле площади, видимой от источника света, то есть скалярному произведению вектора $(\vec{i}, \vec{n})$ [1]. Можно за-

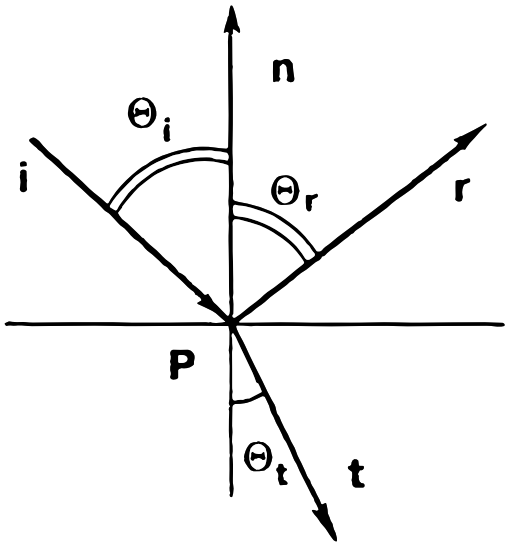

Рисунок 2 – Преломление и отражение луча [1]

дать уравнение для вычисления количества света, диффузно отраженного точкой P как:

$$I_d = \sum_{k=1}^{N} I_k \left(\vec{n}, \vec{i_k} \right), \tag{2}$$

где N — количество попадающих в точку лучей, I_k — интесивность k-го луча.

Зеркальное отражение

Зеркальное отражание характеризует меру *отполированности* поверхности. Пусть $\vec{l}$ — *вектор обзора*, единичный, направленный из точки P к наблюдателю. Учитывая, что вектора $\vec{i}$, $\vec{i}$ и $\vec{r}$ лежат в одной плоскости, легко можно выразить направление отраженного луча:

$$\vec{r} = \vec{i} - 2(\vec{i}, \vec{n})\vec{n}, \tag{3}$$

тогда уравнение для вычисления зеркально отраженного количества света точкой P можно задать как:

$$I_s = \sum_{k=1}^{N} \left(I_k \left(\vec{r_k}, \vec{l} \right) \right)^s, \tag{4}$$

где N — количество попадающих в точку лучей, I_k — интесивность k-го луча, $\vec{r}_k$ - направление отраженного k-го луча, s — *показатель отражения* [4].

Преломление

Согласно закону Снеллиуса векторы $\vec{i}$, $\vec{n}$ и преломленный вектор $\vec{t}$ лежат в одной плоскости и для углов справедливо соотношение:

$$\eta_i \sin\theta_i = \eta_t \sin\theta_t, \tag{5}$$

где η_i и η_t — коэффициенты преломления сред из которой пришел луч и в которую преломился соответственно. Исходя из этого можно получить формулу для нахождения $\vec{t}$:

$$\vec{t} = \frac{\eta_i}{\eta_t}\vec{i} + \left(\frac{\eta_i}{\eta_t}C_i - \sqrt{1 + \left(\frac{\eta_i}{\eta_t}\right)^2 (C_i^2 - 1)}\right)\vec{n}, \tag{6}$$

где $C_i = \cos\theta_i = -(\vec{i}, \vec{n})$. В таком случае ситуация, при которой выражение под корнем $1+\left(\frac{\eta_i}{\eta_t}\right)^2 (C_i^2-1) < 0$ соответствует явлению полного внутреннего отражения, когда световая энергия отражается от границы раздела сред и преломления фактически не происходит.

1.3.2 Алгоритм обратной трассировки лучей

Для синтеза изображения требуется рассчитать цвет каждого пикселя с учетом общей световой энергии, достигающей соответствующего светочувствительного рецептора наблюдателя [5]. Наблюдатель воспринимает объект, когда к нему доходят лучи, путь которых начинается на источнике света и заканчивается на наблюдателе. В соответствии с этим можно использовать свойство обратимости световых лучей и испускать их в обратном направлении — от наблюдателя к объектам сцены. Путь луча при этом будет строиться согласно тем же законам оптики, что и при *прямом* направлении.

Наблюдатель (также называемый *камерой* [4]) находится в некоторой точке S, смотрит через *окно просмотра* (виртуальная проективная плоскость [5]). Каждый пиксель на итоговом изображении соответствует некоторой точке P окна просмотра. Таким образом, представляя точки в виде радиус-векторов, направление испускаемого луча можно найти как:

$$\vec{V} = \vec{P} - \vec{S}. \tag{7}$$

Выпущенный луч (называемый *первичным*) в результате может пересечься или не персечься с каким-либо объектом (рисунок 3). В случае не попадания ни в один объект сцены, пиксель на изображении, соответствующий выпущенному лучу, принимает цвет фона.

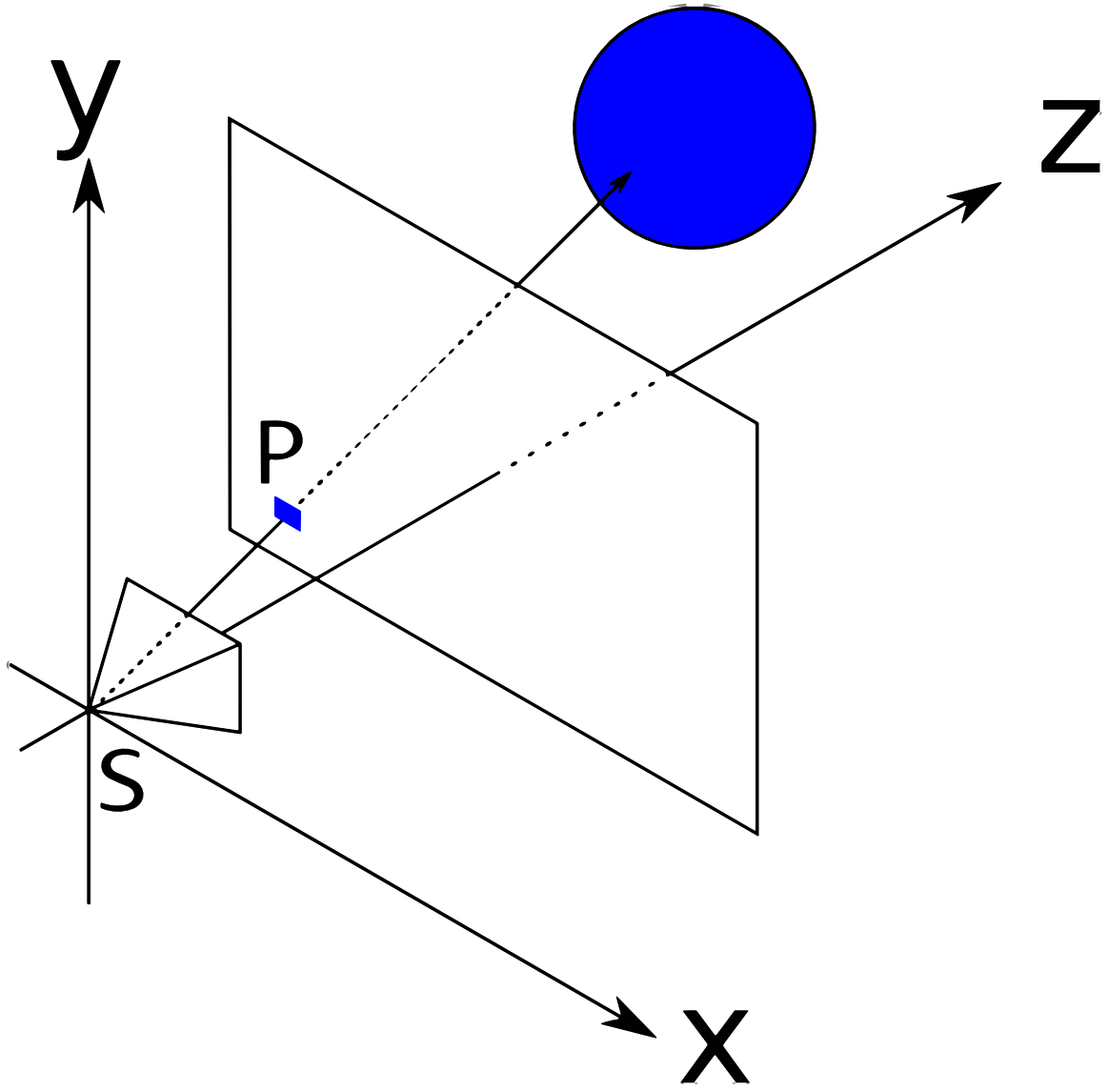

Рисунок 3 – Алгоритм обратной трассировки лучей [4]

При пересечении лучем объекта могут быть порождены *вторичные*

лучи в результате преломления или отражения, а цвет соответствующего пикселя может быть рассчитан по следующей формуле [1, 6, 7]:

$$C = k_a I_a C_p + k_d C_p \sum_j I_j(\vec{n}, \vec{i}_j) + k_s \sum_j I_j(\vec{l}, \vec{r}_j)^s + k_t I_t, \tag{8}$$

где I_a — интенсивность *фонового* освещения, I_j — интенсивность j-го источника света, I_t — освещенность, переносимая преломленным лучом, C_p — цвет в точке пересечения луча и объекта, k_a — коэффициент фонового освещения, k_d — коэффициент диффузного освещения, k_s — коэффициент зеркального освещения, k_t — вклад преломленного луча, $\vec{n}$ — вектор внешней нормали к поверхности в точке пересечения луча и объекта, $\vec{i}_j$ — единичный вектор направления из точки пересечения на j-й источник света, $\vec{r}_j$ — единичный вектор направления отраженного луча, $\vec{l}$ — единичный *вектор обзора*, направленный из точки пересечения к наблюдателю, s — показатель отражения.

1.3.3 Алгоритм трассировки фотонов

Алгоритм обратной трассировки лучей достигает значительных результатов, позволяя моделировать отражение и преломление. Несмотря на широкую популярность и эффективность этого метода, существует набор физических явлений, которые он описывает недостаточно точно или вовсе не охватывает. Примерами таких явлений являются рассеивающие отражения, проявляющиеся в изменении цвета при отражении света от цветной поверхности (цветовой оттенок от комода из красного дерева на белом ковре), а также эффект сфокусированного света, известного как каустики, видимые, например, в бликах от воды на дне бассейна [8].

Алгоритм трассировки фотонов [9] (также известен как алгоритм прямой трассировки лучей [10]) представляет собой процесс в котором лучи (фотоны) исходят из источника света, который может быть не точечным. Пример прохождения лучей в алгоритме приведен на рисунке 4. В данном алгоритме лучи при столкновении с поверхностью могут породить вторичные лучи, если поверхность имеет преломление или *идеальное* (зеркальное) отражение. Если же поверхность не имеет способности породить вторичные

лучи, считается, что энергия луча поглощается и его дальнейшая трассировка прекращается. Серьезным недостатком алгоритма является низкий коэффициент полезного действия — большинство испускаемых фотонов не попадают в глаз наблюдателя.

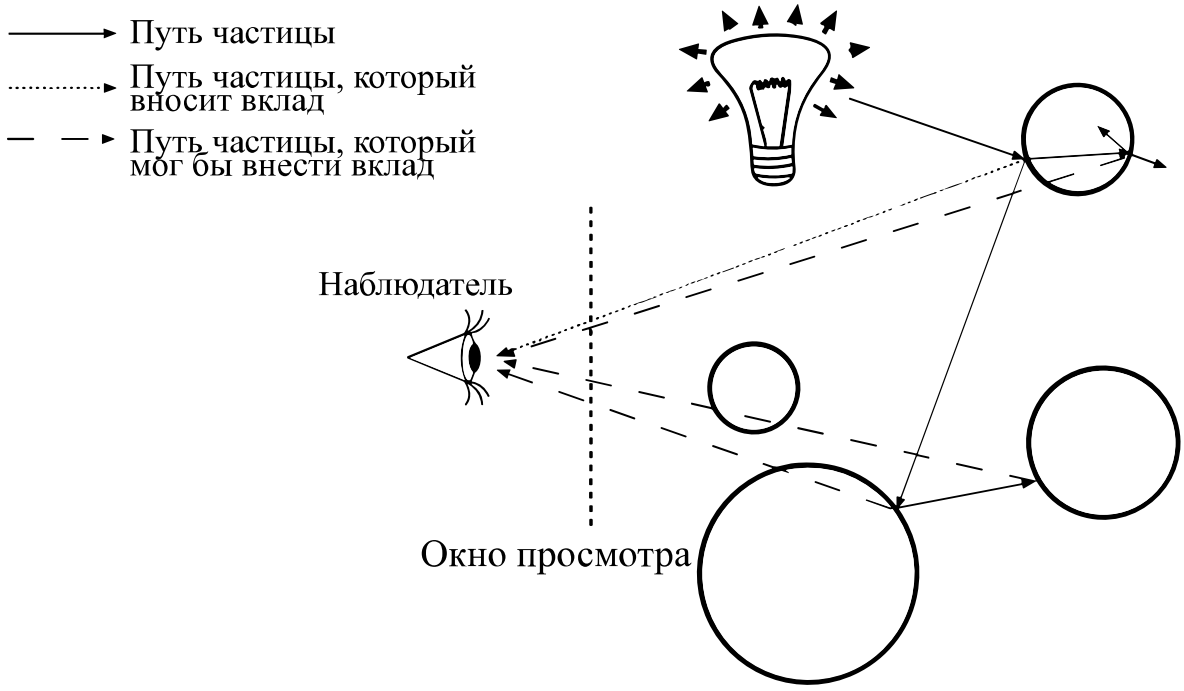

Рисунок 4 – Путь лучей в алгоритме трассировки фотонов [9]

1.3.4 Алгоритм фотонных карт

Концептуально алгоритм фотонных карт [10] немного отличается от остальных методов трассировки, так как является гибридным. Помимо непосредственной трассировки света от источника, он включает этапы построения фотонной карты и сбора освещенности. Фотоны при таком подходе считаются переносчиками некоторой дискретной порции световой энерии. На первом этапе из источника света испускаются фотоны, в зависимости от свойств материала, могут произойти разные события: фотон может отразиться диффузно (в случайном направлении), зеркально, преломиться через поверхность или поглотиться (рисунок 5). Только при диффузном отражении запись о фотоне заносится в список.

Будет ли фотон отражен, преломлен, или поглощен определяется по методу *русской рулетки* [10]. Для монохроматического света, коэффициента диффузного отражения d и коэффициента зеркального отражения s

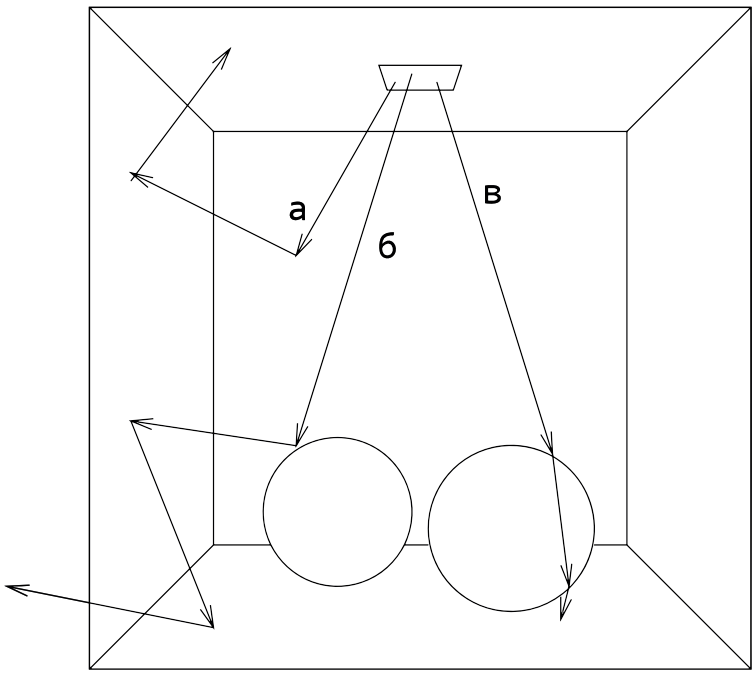

Рисунок 5 – Пути фотонов на сцене: (а) два диффузных отражения и поглощение, (б) зеркальное отражение и два диффузных отражения, (в) два преломления и поглощение [10]

(при $d + s \leq 1$) используется равномерно распределенная в промежутке $[0, 1]$ случайная величина ξ:

$$\begin{aligned} \xi \in [0, d] &\rightarrow \text{диффузное отражение} \\ \xi \in [d, s + d] &\rightarrow \text{зеркальное отражение} \\ \xi \in [s + d, 1] &\rightarrow \text{поглощение} \end{aligned} \tag{9}$$

Сумма энергий фотонов на поверхностях формирует фотонную карту — некоторую структуру пространственного разбиения для удобства нахождения фотонов на следующем этапе. На стадии сбора освещенности происходит сбор энергии из фотонных карт, который может быть выполнен различными подходами. Обычно осуществляется сбор по сфере для каждой поверхности объекта. В каждой точке сбора рассматривается окрестность, и значения точек фотонной карты, попавших в эту окрестность, суммируются и нормируются, представляя результирующую освещенность в точке. Точки сбора сами по себе определяются другими методами, например обратной трассировкой лучей. Помимо этого, другие оптические эффекты как отражения, преломления, тени, также не рассчитываются алгоритмом

фотонных карт, он нужен только для вычисления непрямой освещенности.

Алгоритм фотонных карт характеризуется высокой вычислительной сложностью, которая компенсируется качеством сгенерированных изображений. Обычно он применяется для расчета каустик. Важным аспектом в самом алгоритме является выбор радиусов сфер для сбора, где более большой радиус приводит к длительному сбору и размытости изображения, в то время как маленький радиус дает больше шума. Также требуется построение ускоряющих структур (например KD-дерева) для более эффективного поиска точек в фотонной карте.

1.3.5 Алгоритм трассировки путей

В алгоритме трассировки путей [9] при встрече луча с поверхностью происходит излучение двух новых лучей: один направлен в произвольном (метод русской рулетки) направлении, а другой направлен к источнику света. Прохождение лучей в алгоритме трассировки путей показано на рисунке 6. Доля энергии, отраженной при этом, рассчитывается на основе двулучевой функции отображения (ДФО) — характеристики конкретного материала, показывающей долю энергии, отраженную в зависимости от длины волны, направления на источник света и камеру [11, 9]:

$$\text{ДФО} = \frac{L_0}{L_i \cos(\phi) dw}, \tag{10}$$

где L_0 — количество энергии, отраженное в направлении к наблюдателю. L_i — количество энергии, приходящее от источника. ϕ — угол между нормалью к поверхности и вектором на источник. dw — дифференциальный телесный угол, порожденный направлением к источнику. Работа алгоритма завершается, если длина пройденного пути достигает заданной константы, называемой глубиной трассировки.

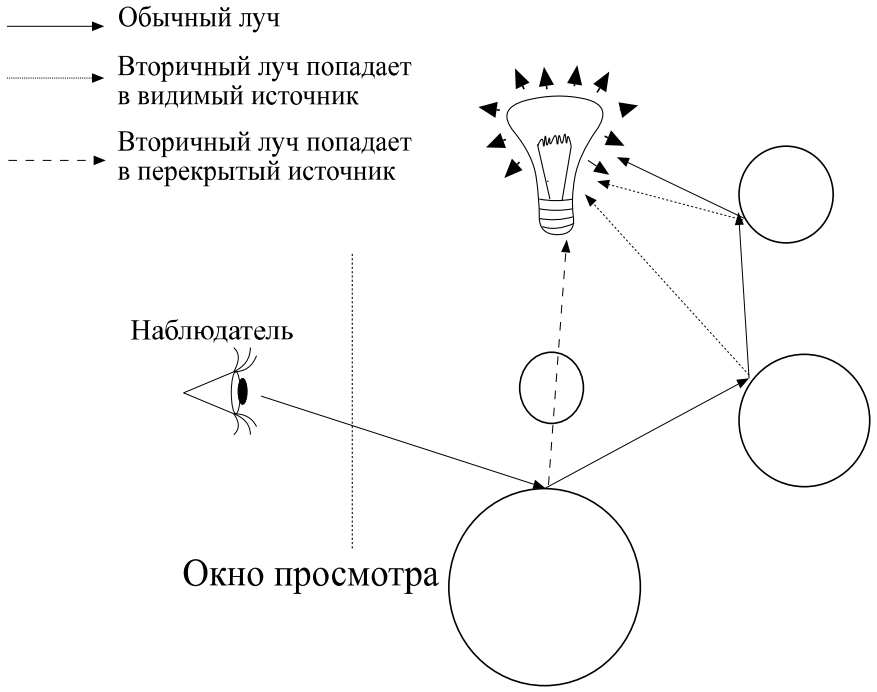

Рисунок 6 – Путь лучей в алгоритме трассировки путей [9]

1.4 Сравнение алгоритмов

В таблице 1 приведено сравнение алгоритмов по возможности симулировать оптические эффекты и некоторым другим характеристикам. «+» означает, что алгоритм имеет возможность строить изображение с учетом оптического эффект, «-» — не имеет возможности. Обозначения:

— ОТЛ — алгоритм обратной трассировки лучей;

— ТФ — алгоритм трассировки фотонов;

— ФК — алгоритм фотонных карт;

— ТП — алгоритм трассировки путей;

— ИС — источник света;

— К — камера;

— N — число вторичных лучей, которые может породить первичный луч при столкновении с поверхностью.

Таблица 1 – Сравнение алгоритмов

	ОТЛ	ТФ	ФК	ТП
Диффузное отражение	+	+	+	+
Зеркальное отражение	+	+	+	+
Преломление	+	+	+	+
Тени	+	+	+	+
Каустики	-	+	+	-
Рассеянные источники	-	+	+	+
Непрямое освещение	-	+	+	+
N	$[0, 2]$	$[0, 2]$	$[0, 1]$	$[0, 2]$
Источник луча	К	ИС	ИС	К

Алгоритм обратной трассировки лучей является наименее вычислительно сложным, он достаточно подходит для реализации требуемой сцены, так как учитывает диффузное отражение, зеркальное отражение и преломление света.

Вывод

В данном разделе было проведено описание объектов сцены, рассмотрены алгоритмы визуализации сцены, отвечающие заданным требованиям. В качестве алгоритма для реализации был выбран алгоритм обратной трассировки лучей как решающий задачу визуализации сцены с учетом диффузного и зеркального отражения и преломления с наименьшими затратами вычислительных ресурсов.

2 Конструкторская часть

2.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

— добавление примитива из набора и задание ему характеристик;

— изменение положения камеры в пространстве;

— изменение положения линзы в пространстве;

— изменение характеристик источника света.

2.2 Описание типов и структур данных

В работе будут использованы следующие типы и структуры данных:

— точка — вектор из трех вещественных чисел соответствующих координатам по каждой из осей трехмерной декартовой системы координат;

— цвет — вектор из трех целых чисел, принимающих значение в диапазоне $[0, 255]$, соответствует цветовой модели RGB;

— материал — представляет характеристики поверхности объекта сцены, содержит цвет, коэффициенты зеркального и диффузного отражения, коэффициент преломления;

— сфера — объект сцены, содержит точку центра и радиус, материал;

— треугольник — основная фигура для задания объектов сцены, является набором из трех точек;

— многогранник — примитивный многогранник, является набором из треугольников, также содержит материал поверхности;

— источник света — содержит цвет, интенсивность и точку своего положения в пространстве;

— камера — содержит точку своего положения в пространстве и углы поворота вокруг каждой из координатных осей, высоту и ширину окна просмотра, а также удаленность окна просмотра;

— линза — содержит точку своего центра, показатель преломления, углы поворота вокруг каждой из координатных осей;

— сцена — хранит в себе набор всех многогранников и сфер, камеру, источник света, линзу.

2.3 Алгоритм обратной трассировки лучей

Алгоритм обратной трассировки лучей испускает лучи от наблюдателя к объектом сцены через окно просмотра. Каждый луч проходит через окно просмотра и рассчитывает цвет некоторого пикселя на результирующем изображении. Если задать высоту и ширину изображения как C_w и C_h, высоту и ширину окна просмотра как V_w и V_h, удаленность окна просмотра вдоль оси z как d, камеру в точке O, то точку P_{ij} в плоскости окна просмотра, соответствующую пикселю изображения в i строке и j столбце можно найти по следующей формуле:

$$P_{ij} = \left(i\frac{V_w}{C_w} + O_x, j\frac{V_h}{C_h} + O_y, d + O_z \right). \tag{11}$$

Данная формула по сути является преобразованием масштаба и сдвига. Тогда вектор направления луча $\vec{D}$ можно определить по формуле:

$$\vec{D} = \vec{P_{ij}} - \vec{O}. \tag{12}$$

Имея точку начала луча и вектор направления, луч записывается в параметрическом виде:

$$\vec{L} = \vec{O} + t\vec{D}, \tag{13}$$

где $t \in [0, +\inf]$.

Схема алгоритма обратной трассировки лучей приведена на рисунках 7 и 8.

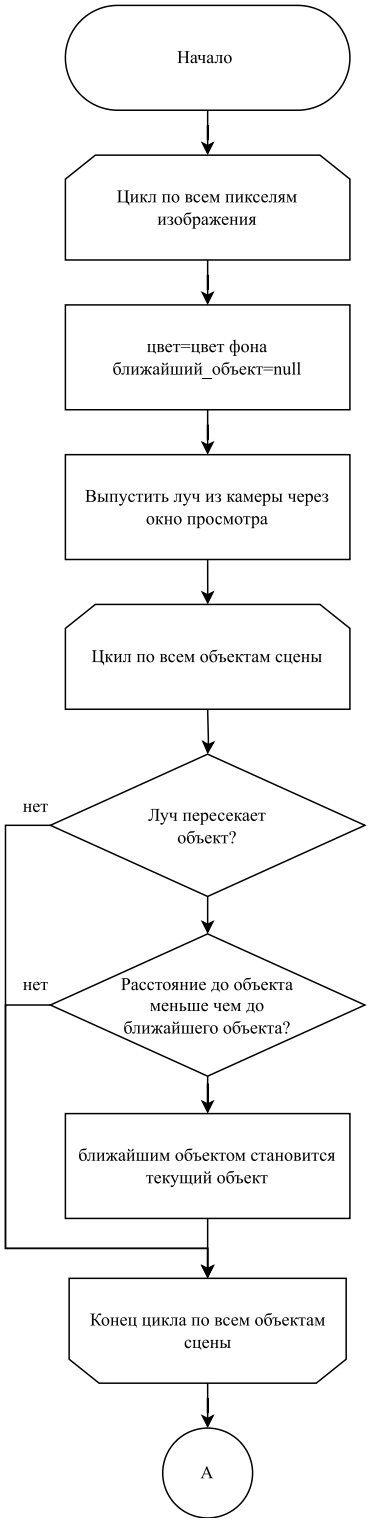

Рисунок 7 – Схема алгоритма обратной трассировки лучей

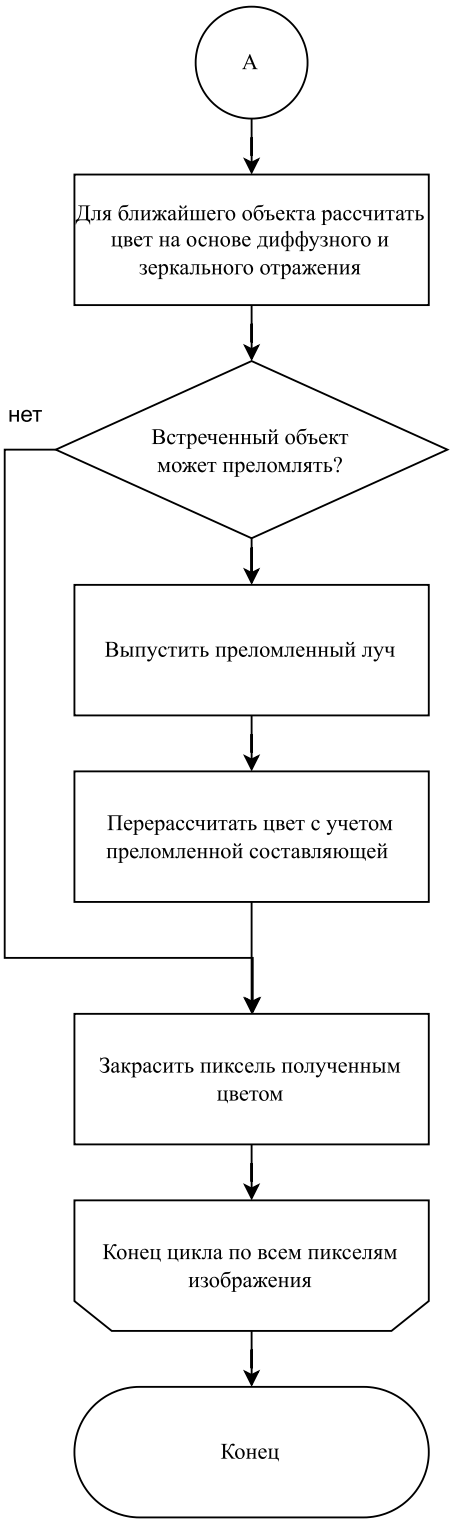

Рисунок 8 – Схема алгоритма обратной трассировки лучей (продолжение)

2.3.1 Пересечение сферы и луча

Сфера - множество точек, лежащих на некотором заданном расстоянии (*радиусе*) от точки центра сферы, соответственно считая точку C — центром, r — радиусом, а P — точкой на сфере, по свойствам скалярного произведения можно задать уравнение сферы как:

$$(\vec{P} - \vec{C}, \vec{P} - \vec{C}) = r^2. \tag{14}$$

Луч пересекается со сферой, если точка луча удовлетворяет соотношению сферы, в таком случае получается система из уравнений 13 и 14, в которой $\vec{L} = \vec{P}$:

$$\begin{cases} \vec{P} = \vec{O} + t\vec{D} \\ (\vec{P} - \vec{C}, \vec{P} - \vec{C}) = r^2 \end{cases}. \tag{15}$$

Подставив первое уравнение во второе в системе 15 и преобразовав, получим квадратное уравнение относительно параметра t:

$$t^2(\vec{D}, \vec{D}) + 2t(\vec{O} - \vec{C}, \vec{D}) + (\vec{O} - \vec{C}, \vec{O} - \vec{C}) - r^2 = 0. \tag{16}$$

При решении уравнения 16 можно получить от нуля до двух решений, что будет соответствовать ситуациям, когда луч не пересекает сферу, когда луч касается сферы, и когда луч проходит насквозь, пересекая сферу в двух точках. В последнем варианте следует выбирать ближайшую точку пересечения, которая соответствует меньшему значению параметра.

2.3.2 Пересечение треугольника и луча

Так как поверхность всех многогранников можно представить в виде набора треугольников, появляется задача определения пересечения треугольника и луча. Имея вершины треугольника в виде точек V_0, V_1, V_2, любую точку в треугольнике можно представить в виде:

$$T(u, v) = (1 - u - v)\vec{V_0} + u\vec{V_1} + v\vec{V_2}, \tag{17}$$

где u и v — барицентрические координаты, которые должны удовлетворять условиям $u \geq 0, v \geq 0, u + v \leq 1$. Тогда вычисление точки пересечения луча эквивалентно вычислению уравнения $\vec{L} = T(u, v)$, которое записывается как:

$$\vec{O} + t\vec{D} = (1 - u - v)\vec{V_0} + u\vec{V_1} + v\vec{V_2}. \tag{18}$$

Перегруппировав члены, получается:

$$\left(-\vec{D},\ \ \vec{V_1} - \vec{V_0},\ \ \vec{V_2} - \vec{V_0}\right) \begin{pmatrix} t \\ u \\ v \end{pmatrix} = \vec{O} - \vec{V_0} \tag{19}$$

Обозначим $\vec{E_1} = \vec{V_1} - \vec{V_0}, \vec{E_2} = \vec{V_2} - \vec{V_0}, \vec{T} = \vec{O} - \vec{V_0}$, решая уравнение по методу Крамера можно получить:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{((\vec{D} \times \vec{E_2}), \vec{E_1})} \begin{pmatrix} ((\vec{T} \times \vec{E_1}), \vec{E_2}) \\ ((\vec{D} \times \vec{E_2}), \vec{T}) \\ ((\vec{T} \times \vec{E_1}), \vec{D}) \end{pmatrix} \tag{20}$$

Проверяя, что полученные полученные параметры u и v удовлетворяют условиям $u \in [0, 1], v \in [0, 1], u + v \leq 1$ можно определить, пересекает ли луч треугольник. Благодаря рассчитанному параметру t можно найти точку пересечения, подставив параметр в уравнение луча.

2.3.3 Пересечение линзы и луча

Линза представляет собой тело, ограниченное двумя сферическими поверхностями. Однако алгоритм нахождения пересечения сферы и луча не подходит для использования с линзой, так как будет учитывать лишние лучи, которые пересекаются со сферой, но не пересекаются с линзой. Возможно модифицировать алгоритм, добавив проверку, что точка пересечения действительно лежит на поверхности линзы, а не на мнимой сфере, частью которой является поверхность линзы.

На рисунке 9 изображено прохождение луча с вектором направления $vecL$ и началом в точке O. Луч пересекает линзу в точке P, вектор $\vec{H}$

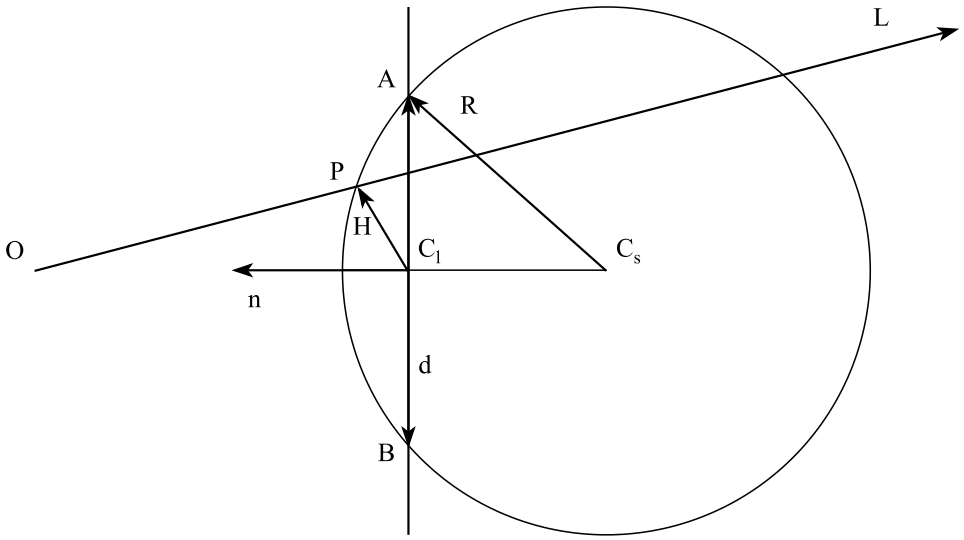

Рисунок 9 – Схема прохождения луча через линзу

направлен центра линзы C_l в P, тогда, зная единичный вектор внешней нормали к плоскости линзы $\vec{n}$ можно сказать, что P лежит на поверхности линзы, если скалярное произведение $(\vec{H}, \vec{n}) \geq 0$, то есть угол между $\vec{H}$ и $\vec{n}$ менее или равен 90 градусов. Для нахождения точки P требуется знать центр сферы C_s, его можно найти, зная диаметр линзы d, C_l и $\vec{n}$ как:

$$\vec{C_s} = \vec{C_l} - \vec{n}\sqrt{R^2 - \left(\frac{d}{2}\right)^2}. \tag{21}$$

Таким образом, получив точку P с помощью центра сферы и уравнения луча и проверив ее на принадлежность поверхности линзы, можно узнать пересекается ли луч с линзой.

Вывод

В данном разделе были описаны требования к программному обеспечению, используемые типы и структуры данных и спроектирован алгоритм обратной трассировки лучей.

3 Технологическая часть

В данной части рассматривается выбор средств реализации, описывается структура программы и приводится интерфейс программного обеспечения.

3.1 Средства реализации

В качестве языка программирования для разработки программного обеспечения был выбран язык программирования C++ [12]. Данный выбор обусловлен следующими факторами:

— C++ обладает высокой вычислительной производительностью, что очень важно для выполнения поставленной задачи;

— Данный язык изучался в рамках учебной программы.

В качестве среды разработки был выбран QtCreator [13]. Он имеет инструменты для удобной разработки программ с использованием выбранного фреймворка для графического интерфейса Qt [14].

3.2 Структура программы

На рисунке 10 представлена схема разработанных классов программы.

Основные классы разработанной программы:

— Object — базовый класс объекта;

— Lens — класс линзы;

— Light — класс источника света;

— PolyObject — класс полигонального объекта;

— Sphere — класс сферы;

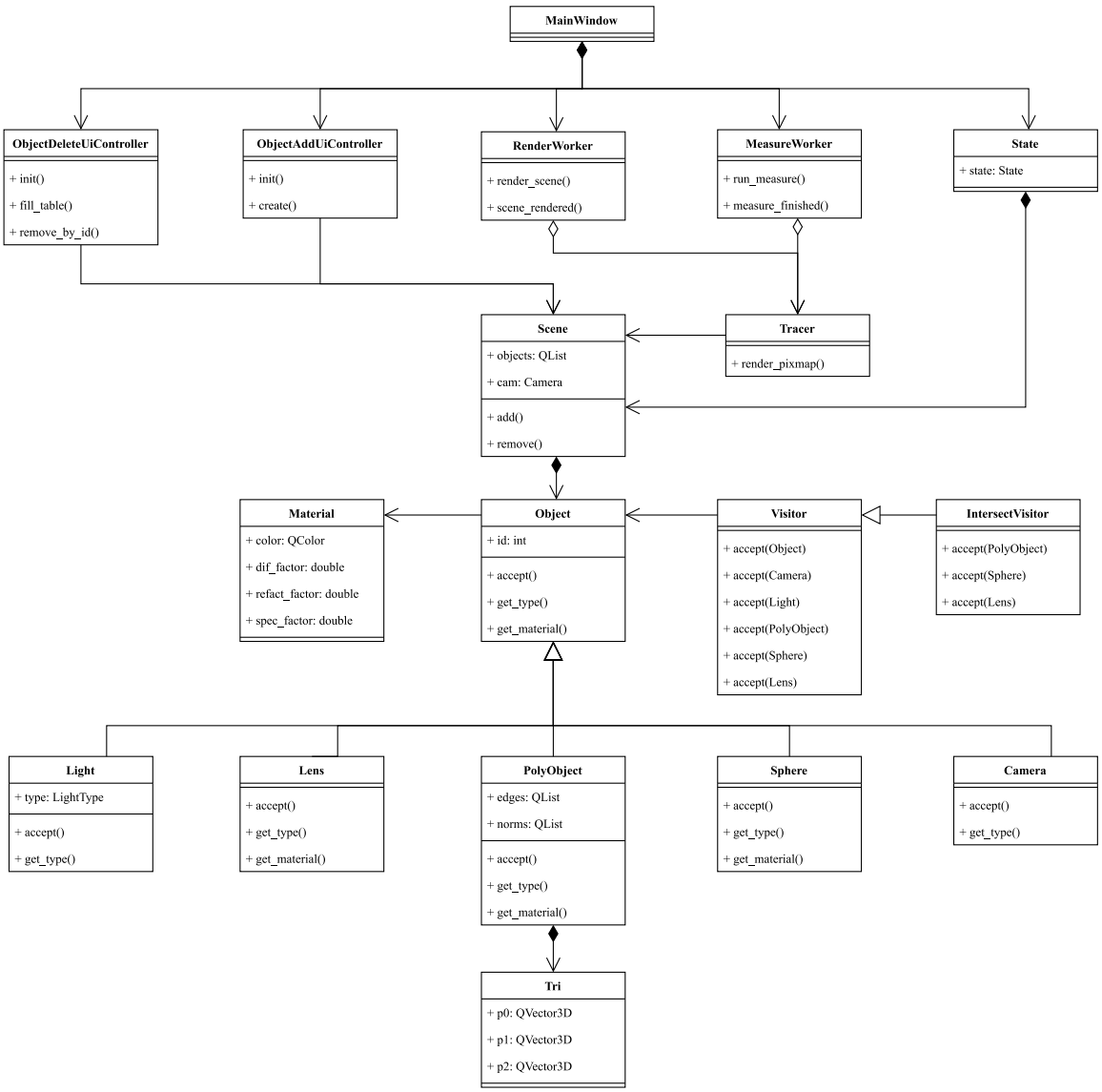

Рисунок 10 – Структура классов программы

— Camera — класс камеры;

— Tri — класс, представляющий полигон (треугольник);

— Material — класс, представляющий материал объекта;

— Visitor — базовый класс для реализации паттерна *Посетитель*;

— IntersectVisitor — класс паттерна *Посетитель* для нахождения пересечения луча с объектами;

— Scene — класс сцены;

— Tracer — класс, содержащий алгоритмы трассировки лучей;

— MeasureWorker — класс для управления замерами;

— RenderWorker — класс для управления отрисовкой изображения;

— ObjectAddUiController — класс для управления виджетами добавления объекта;

— ObjectDeleteUiController — класс для управления виджетами удаления объекта;

— MainWindow — класс главного виджета окна.

3.3 Реализация алгоритмов

Фрагменты реализации алгоритма представлены на листингах:

— листинг 1 — основная, «высокоуровневая» функция отрисовки сцены;

— листинг 2 — функция трассировки луча, используется для получения цвета пикселя изображения;

— листинг 3 — функция нахождения ближайшего пересечения луча с объектами сцены;

— листинг 4 — функция для рассчета цвета объекта с учетом освещения в точке пересечения с лучем;

— листинг 5 — функция определения направления преломленного луча.

Листинг 1 – Функция отрисовки сцены на изображении

```
1  void render_pixmap(const Scene& scene, QPixmap& pix)
2  {
3      const auto& cam = scene.cam;
4      QMatrix4x4 cam_rot, cam_rot_x, cam_rot_y, cam_rot_z;
5      cam_rot_x.setToIdentity();
6      cam_rot_x.rotate(cam.rot.x(), 1, 0, 0);
7      cam_rot_y.setToIdentity();
8      cam_rot_y.rotate(cam.rot.y(), 0, 1, 0);
9      cam_rot_z.setToIdentity();
10     cam_rot_z.rotate(cam.rot.z(), 0, 0, 1);
11     cam_rot = cam_rot_z * cam_rot_y * cam_rot_x;
12     QPainter paint{&pix};
13     int height = pix.height();
14     int width = pix.width();
15     int rendered_pixels = 0;
16     for (int y = 0; y < height; y++)
17     {
18         for (int x = 0; x < width; x++)
19         {
20             if (should_stop)
21                 return;
```

```
22              QVector3D viewport_point = cam.get_viewport_point(x
                    - width / 2, -y + height / 2, width, height);
23              QVector3D direction = viewport_point - cam.pos;
24              direction = cam_rot.mapVector(direction);
25              QColor col = Tracer::trace_ray(scene.objects,
                    scene.lights, cam.pos, cam.pos + direction, 1,
                    Vars::far_distance);
26              paint.setPen(QPen{QColor{col}});
27              paint.drawPoint(x, y);
28              ++rendered_pixels;
29              if (rendered_pixels % 100 == 0)
30                  emit rendering_progress(rendered_pixels);
31          }
32      }
33  }
```

Листинг 2 – Функция испускания луча

```
1  QColor trace_ray(const QList<Object*>& objects, const
       QList<Light*>& lights, const QVector3D& beg, const
       QVector3D& end, double t_min, double t_max) {
2      auto [closest_object, intersection_point, norm] =
           Tracer::closest_intersection_with_norm(objects, beg,
           end, t_min, t_max);
3      if (closest_object == nullptr)
4      return Vars::bg_color;
5      QColor result_color{};
6      const Material& m = closest_object->get_material();
7      norm.normalize();
8      result_color = Tracer::compute_color_with_lights(objects,
           lights, intersection_point, norm, (end - beg) * -1,
           m.spec_factor, m.dif_factor, m.color);
9      if (m.refact_factor > 0) {
10         QVector3D refact_dir =
               Tracer::compute_refact_direction(end - beg, norm,
               m.refact_factor);
11         refact_dir.normalize();
12         auto refact_color = Tracer::trace_ray(objects, lights,
               intersection_point, intersection_point + refact_dir
               * Vars::small_distance, t_min, t_max);
13         result_color = colorSum(result_color, refact_color);
14     }
```

```
15      return result_color;
16  }
```

Листинг 3 – Функция нахождения пересечения луча с объектом сцены

```
1  std::tuple<Object*, QVector3D, QVector3D>
2  closest_intersection_with_norm(const QList<Object*>& objects,
       const QVector3D& beg, const QVector3D& end, qreal t_min,
       qreal t_max)
3  {
4      qreal closest_t = Vars::far_distance;
5      Object* closest_obj = nullptr;
6      QVector3D intersection_point, norm;
7      IntersectVisitor v{};
8      v.beg = beg;
9      v.end = end;
10     v.t_min = t_min;
11     v.t_max = t_max;
12     for (const auto& obj : objects) {
13         if (should_stop)
14         break;
15         obj->accept(&v);
16         if (!v.intersects)
17         continue;
18         if (v.t_intersect < closest_t) {
19             closest_t = v.t_intersect;
20             closest_obj = obj;
21             intersection_point = v.intersection_point;
22             norm = v.norm;
23         }
24     }
25     return {closest_obj, intersection_point, norm};
26 }
```

Листинг 4 – Функция рассчета цвета объекта в точке с учетом освещения

```
1  QColor compute_color_with_lights(const QList<Object*>& objects,
       const QList<Light*>& lights, const QVector3D& point, const
       QVector3D& norm, const QVector3D& v, qreal s, qreal dif,
       QColor obj_color) {
2      QColor result_color{};
3      for (const auto& li : lights) {
4          if (should_stop)
```

```
5           break;
6           switch (li->type)
7           {
8               case LightType::AMBIENT: {
9                   result_color = colorSum(result_color,
                        colorMul(colorMul(obj_color, li->color),
                        li->intensity));
10                  break;
11              }
12              case LightType::DIRECTIONAL:
13              case LightType::POINT: {
14                  QVector3D l;
15                  double t_max;
16                  if (li->type == LightType::POINT) {
17                      l = li->position - point;
18                      t_max = 1;
19                  } else {
20                      l = li->direction;
21                      t_max = Vars::far_distance;
22                  }
23                  if (Vars::enable_shadows) {
24                      auto [shadow_object,
                            shadow_intersection_point, shadow_norm] =
25                      Tracer::closest_intersection_with_norm(
26                      objects, point, point + l, 1e-3, t_max);
27                      if (shadow_object != nullptr)
28                      continue;
29                  }
30                  double n_dot_l = QVector3D::dotProduct(norm, l);
31                  if (n_dot_l > 0)
32                  result_color = colorSum(result_color,
                        cmul(cmul(obj_color, li->color),
                        ((li->intensity * n_dot_l / (norm.length() *
                        l.length()))) * dif)));
33                  if (s >= 0) {
34                      QVector3D r = (norm * 2 *
                            QVector3D::dotProduct(norm, l)) - l;
35                      double r_dot_v = QVector3D::dotProduct(r,
                            v);
36                      if (r_dot_v > 0)
37                      result_color = csum(result_color,
```

```
                        cmul(cmul(obj_color, li->color),
                        (li->intensity * pow(r_dot_v /
                        (r.length() * v.length()), s))));
38                  }
39                  break;
40              }
41              default:
42              break;
43          }
44      }
45      return result_color;
46  }
```

Листинг 5 – Функция направления преломленного луча

```
1  QVector3D compute_refact_direction(QVector3D direction,
       QVector3D norm, double refact_index) {
2      direction.normalize();
3      double eta = 1.0 / refact_index;
4      double cos_theta = -1 * QVector3D::dotProduct(norm,
           direction);
5      if (cos_theta < 0) {
6          cos_theta *= -1;
7          norm = norm * -1;
8          eta = 1.0 / eta;
9      }
10     double k = 1.0 - eta * eta * (1.0 - cos_theta * cos_theta);
11     if (k >= 0) {
12         return direction * eta + norm * (eta * cos_theta -
               qSqrt(k));
13     }
14     return QVector3D{0, 0, 0};
15 }
```

3.4 Интерфейс программного обеспечения

На рисунке 11 представлен интерфейс программы. Действия, доступные пользователю разделены на 5 вкладок, в каждой содержатся соответствующие названию вкладки параметры. После произведения каких-либо действий для отрисовки результата требуется нажать на кнопку «Отрисовать», однако также можно установить флаг «Запускать отрисовку при изменении параметров», благодаря которому после каждого действия будет происходить автоматическая перерисовка изображения.

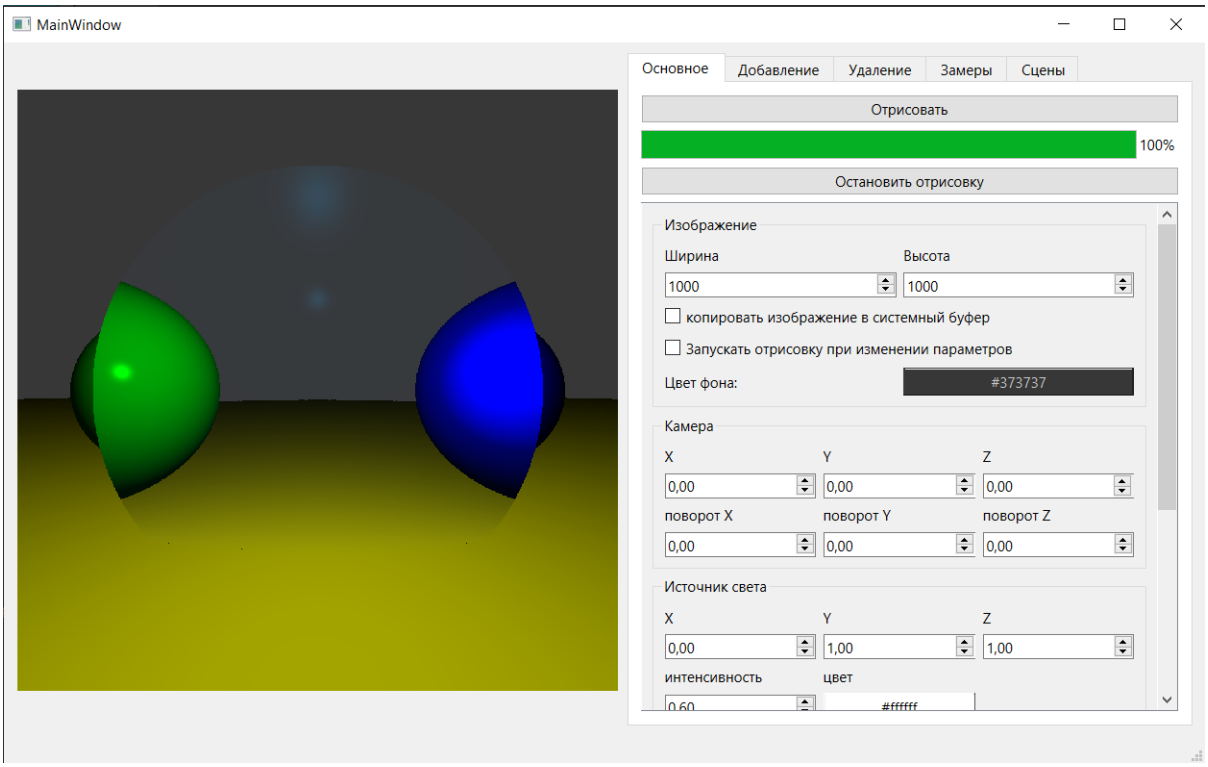

Рисунок 11 – Интерфейс программы

Вывод

В данном разделе были выбраны средства реализации, описана схема классов, реализованы выбранные алгоритмы и разработано программное обеспечение.

4 Исследовательская часть

4.1 Цель исследования

Целью исследования является определение скорости работы реализованного алгоритма генерации изображения.

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

— Процессор: Intel(R) Core(TM) i5-9300H CPU 2.40GHz [15].

— Оперативная память: 24 ГБайт.

— Операционная система: Windows 10 Pro 22H2 [16].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.3 Результаты исследования

Исследование зависимости времени генерации изображения было проведено в двух вариантах:

— в зависимости от размера изображения;

— в зависимости от количества объектов на сцене.

Для обоих вариантов была использована одна из заранее заготовленных и доступных через пользовательский интерфейс сцен — сцена «кубы», содержащая 25 кубов.

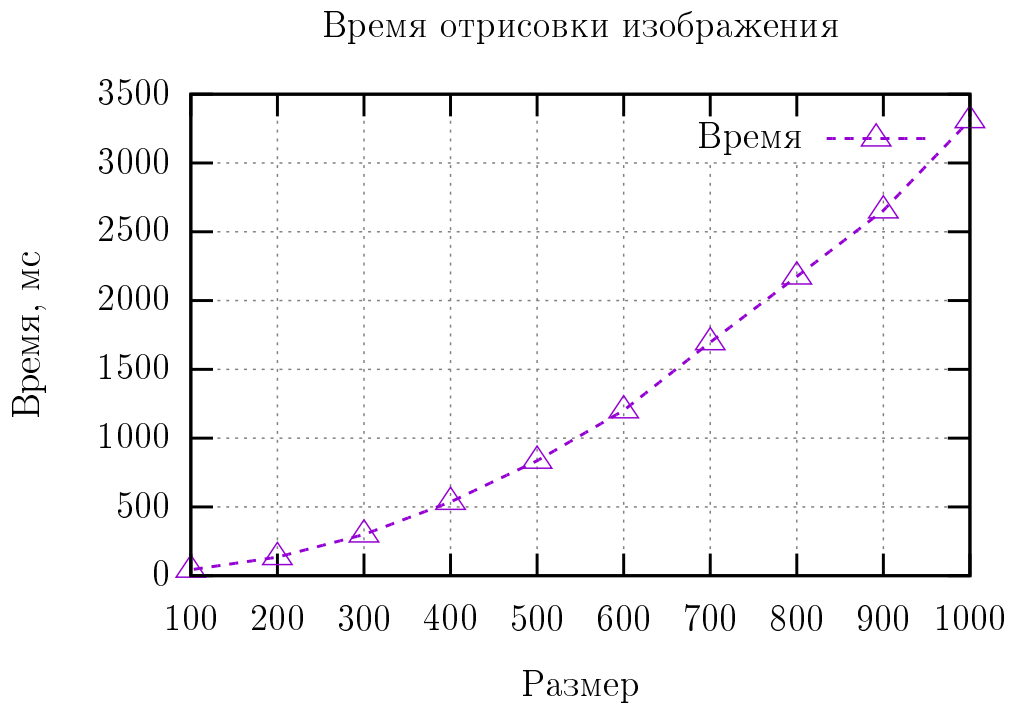

Рисунок 12 – График зависимости времени отрисовки от размера изображения

Результаты исследования зависимости времени генерации от размера изображения представлены на рисунке 12. Соотношение сторон изображения — один к одному, то есть количество пикселей пропорционально квадрату размера.

Результаты второго варианта исследования представлены на рисунке 13. Для получения зависимости из сцены по одному удалялись объекты и измерялось время отрисовки для полученной сцены. Исследование проводилось с шириной и высотой изображения, равными 400.

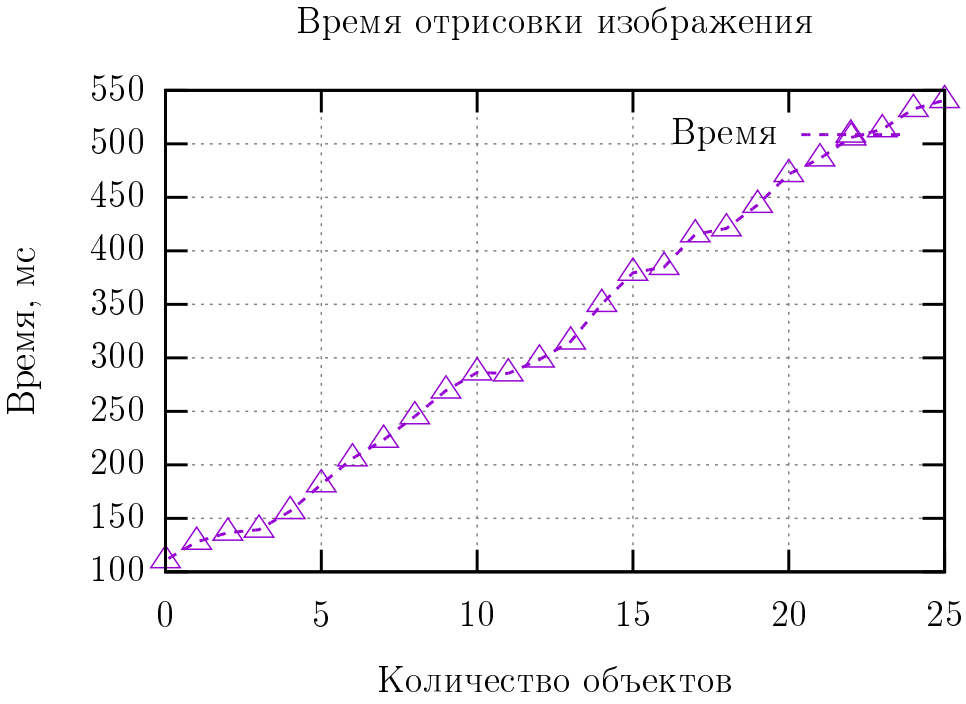

Рисунок 13 – График зависимости времени отрисовки от количества объектов

Вывод

В данном разделе были проведены исследования производительности и приведены результаты работы полученного программного обеспечения.

ЗАКЛЮЧЕНИЕ

Цель данной курсовой работы была достигнута. Было разработано программное обеспечение для визуализации эффектов искажения изображения, получаемых с помощью двояковыпуклой линзы. Пользователю предоставляются возможности для изменения положения камеры и источника света, изменения спектральных характеристик источника.

Для достижения цели были выполнены следующие задачи:

— рассмотрены существующие алгоритмы визуализации сцены, учитывающие такие свойства объектов как преломление и отражение света, выбран достаточный для решения поставленной задачи;

— спроектирован выбранный алгоритм;

— разработано программное обеспечение и реализованы выбранные алгоритмы;

— проведено исследование производительности полученного программного обеспечения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Боресков А.В. Компьютерная графика : учебник и практикум для среднего профессионального образования / А. В. Боресков, Е. В. Шикин. — Москва: Издательство Юрайт, 2017.

2. Лекции по физике СУНЦ МГУ, Г. 14. Геометрическая оптика. — СУНЦ МГУ, 2013.

3. А. Лебедев. История развития алгоритмов глобального освещения. — Компьютерная графика и мультимедиа, №9, 2011.

4. Gambetta G. Computer graphics from scratch : a programmer's introduction to 3D rendering / Gabriel Gambetta. — San Francisco : No Starch Press, 2021.

5. Никулин Е. А. Компьютерное моделирование оптических эффектов. — Труды НГТУ им. Р. Е. Алексеева, 2011. — С. 77–87.

6. Д. Роджерс. Алгоритмические основы машинной графики. — Москва: Издательство Мир, 1989.

7. Whitted T. An improved illumination model for shaded display. — Communications of the ACM, 1980. — P. 343–349.

8. Меженин А.В. Технологии разработки 3D-моделей. Учебное пособие. — СПб: Университет ИТМО, 2018.

9. Szirmay-Kalos L. Photorealistic image synthesis using ray-bundles, PhD dissertation. — Department of Control Engineering and Information Technology Technical University of Budapest, 2000.

10. Jensen H. A. Practical guide to global illumination using ray tracing and photon mapping. — ACM SIGGRAPH 1999 Course Notes, 1999.

11. Ильин А.А. Моделирование отражательных свойств материалов плоских объектов по фотоизображениям / А.А. Ильин, А.С. Лебедев, В.А. Синявский, А.В. Игнатенко. — Proc. of Graphicon'2009, 2009. — С. 198–202.

12. Документация по C++ [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170/ (дата обращения: 22.12.2023).

13. QtCreator Manual [Электронный ресурс]. — Режим доступа: https://doc.qt.io/qtcreator/ (дата обращения: 22.12.2023).

14. Документация по Qt [Электронный ресурс]. — Режим доступа: https://doc.qt.io/ (дата обращения: 22.12.2023).

15. Intel [Электронный ресурс]. — Режим доступа: https://ark.intel.com/content/www/us/en/ark/products/191075/intel-core-i5-9300h-processor-8m-cache-up-to-4-10-ghz.html (дата обращения: 22.12.2023).

16. Windows 10 Pro 22H2 [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/windows/release-health/status-windows-10-22h2 (дата обращения: 22.12.2023).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе

Презентация содержит 12 слайдов.