Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Визуализация дисперсии света на прозрачных предметах»

Студент ИУ7-53Б _____ Д. В. Недолужко
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ Н. В. Новик
(Подпись, дата) (И. О. Фамилия)

2022 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дисперсия - это разбиения белого пучка света на его цветные составляющие при прохождении через прозрачные поверхности. Наиболее известным экспериментом, показывающим это явление, является пропускание белого пучка света через призму и наблюдение светового спектра на экране. Явление дисперсии также иллюстрируют радуга и блеск драгоценных камней.

Явление дисперсии занимало ума людей столетиями. Инженеры, проектирующие оптические приборы, стремились минимизировать ее проявление в своих приборах. В то время как ювелиры находились в постоянном стремлении преумножить блеск своих драгоценных камней.

Дисперсия повсюду встречается в нашей жизни, это явление подробно изучено с физической точки зрения. При построении изображений, претендующих на фотореалистичность, нельзя не учитывать это явление.

Целью моего проекта является разработка программного обеспечения для визуализации трехмерных объектов и наблюдения дисперсии света на прозрачных поверхностях с возможностью выбора пользователем объектов сцены из предложенного списка, а также задания источников освещения по их характеристикам: положению, интенсивности.

Для достижения поставленной цели необходимо решить следующие задачи:

- изучение явления дисперсии с физической точки зрения;
- анализ существующих алгоритмов построения реалистичных изображений;
- выбор алгоритма для решения поставленной задачи;
- проектирование архитектуры и графического интерфейса программы;
- реализация структур данных и алгоритмов;
- описание структуры разрабатываемого ПО;
- написание программы;
- исследование производительности программы.

1 Аналитический раздел

В данном разделе рассматриваются существующие алгоритмы построения реалистичных изображений. Так же, обосновывается выбор реализуемого алгоритма и указывается список ограничений, в рамках которых будет работать разрабатываемое ПО.

1.1 Описание алгоритмов удаления невидимых линий и поверхностей

1.1.1 Некоторые теоретические сведения

Прежде чем описывать алгоритмы, нужно дать некоторые определения, которые будут использованы в данном разделе.

Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства [1].

Решать задачу можно в:

- объектном пространстве - мировая системе координат, высокая точность. Обобщенный подход, основанный на анализе пространства объектов, предполагает попарное сравнение положения всех объектов по отношению к наблюдателю.

- пространстве изображений - в экранных координатах, системе координат, связанной с тем устройством в котором отображается результат. (Графический дисплей).

Под экранированием подразумевается загораживание одного объекта другим.

Под глубиной подразумевается значение координаты Z, направленной от зрителя, за плоскость экрана.

1.1.2 Алгоритм Робертса

Алгоритм Робертса решает задачу удаления невидимых линий. Работает в объектном пространстве. Данный алгоритм работает исключительно с выпуклыми телами. Если тело изначально является не выпуклым, то нуж-

но его разбить на выпуклые составляющие. Алгоритм целиком основан на математических предпосылках [1].

Из-за сложности математических вычислений, используемых в данном алгоритме, а так же из-за дополнительных затраты ресурсов на вычисление матриц данный алгоритм является довольно медленным.

1.1.3 Алгоритм Варнака

Алгоритм Варнака [1—3] позволяет определить, какие грани или части граней объектов сцены видимы, а какие заслонены гранями других объектов. Так же как и в алгоритме Робертса анализ видимости происходит в пространстве изображения. В качестве граней обычно выступают выпуклые многоугольники, алгоритмы работы с ними эффективнее, чем с произвольными многоугольниками. Окно, в котором необходимо отобразить сцену, должно быть прямоугольным. Алгоритм работает рекурсивно, на каждом шаге анализируется видимость граней и, если нельзя легко определить видимость, окно делится на 4 части и анализ повторяется отдельно для каждой из частей (см. рис. 1.1).

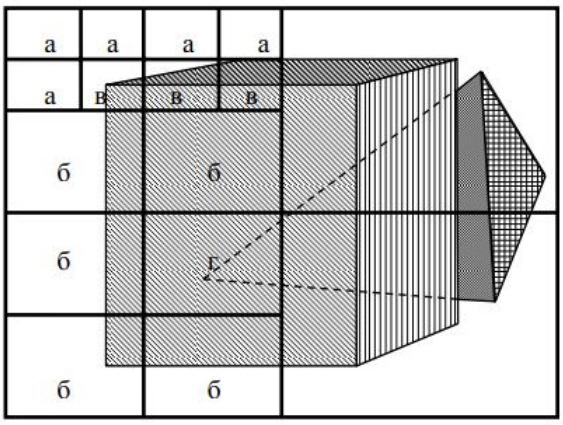

Рисунок 1.1 – Пример разбиения Алгоритмом Варнока

Так как данный алгоритм основывается на рекурсивном разбиении экрана, в зависимости от расположения объектов это может вызвать, как положительной, так и отрицательный эффект. Чем меньше пересечений объектов, тем быстрее алгоритм завершит свою работу.

1.1.4 Алгоритм Z–буфера

Алгоритм Z–буфера [1; 2] позволяет определить, какие пикселы граней сцены видимы, а какие заслонены гранями других объектов. Z–буфер -– это двухмерный массив, его размеры равны размерам окна, таким образом, каждому пикселу окна, соответствует ячейка Z-буфера. В этой ячейке хранится значение глубины пиксела (см. рис. 1.2). Перед растеризацией сцены Z–буфер заполняется значением, соответствующим максимальной глубине. В случае, когда глубина характеризуется значением w, максимальной глубине соответствует нулевое значение. Анализ видимости происходит при растеризации граней, для каждого пиксела рассчитывается глубина и сравнивается со значением в Z–буфере, если рисуемый пиксел ближе (его `w` больше значения в Z–буфере), то пиксел рисуется, а значение в Z–буфере заменяется его глубиной. Если пиксел дальше, то пиксел не рисуется и Z–буфер не изменяется, текущий пиксел дальше того, что нарисован ранее, а значит невидим.

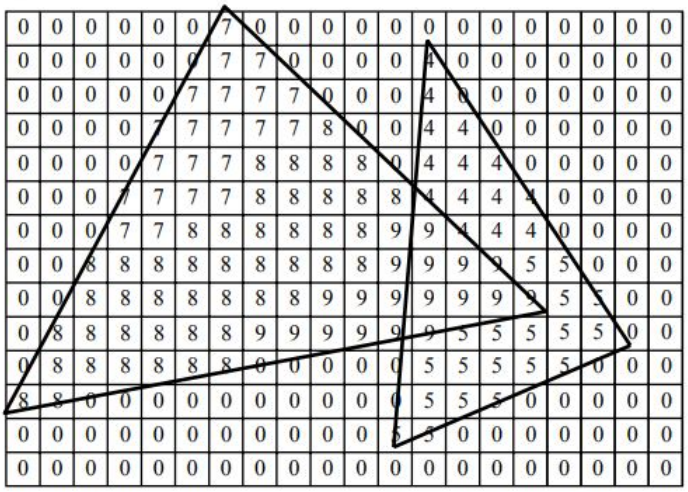

Рисунок 1.2 – Пример работы алгоритма Z–буфера

К недостаткам алгоритма следует отнести довольно большие объемы требуемой памяти, а также имеются другие недостатки, которые состоят в трудоемкости устранения лестничного эффекта и трудности реализации эффектов прозрачности.

1.1.5 Алгоритм прямой трассировки лучей

Основная идея алгоритма прямой трассировки лучей [2] состоит в том, что наблюдатель видит объекты благодаря световым лучам, испускаемым

некоторым источником, которые падают на объект, отражаются, преломляются или проходят сквозь него и в результате достигают зрителя.

Основным недостатком алгоритма является излишне большое число рассматриваемых лучей, приводящее к существенным затратам вычислительных мощностей, так как лишь малая часть лучей достигает точки наблюдения. Данный алгоритм подходит для генерации статических сцен и моделирования зеркального отражения, а так же других оптических эффектов [4].

1.1.6 Алгоритм обратной трассировки лучей

Алгоритм обратной трассировки лучей отслеживает лучи в обратном направлении (от наблюдателя к объекту) [2]. Такой подход призван повысить эффективность алгоритма в сравнении с алгоритмом прямой трассировки лучей. Обратная трассировка позволяет работать с несколькими источниками света, передавать множество разных оптических явлений [5].

Пример работы данного алгоритма приведен на рисунке 1.3.

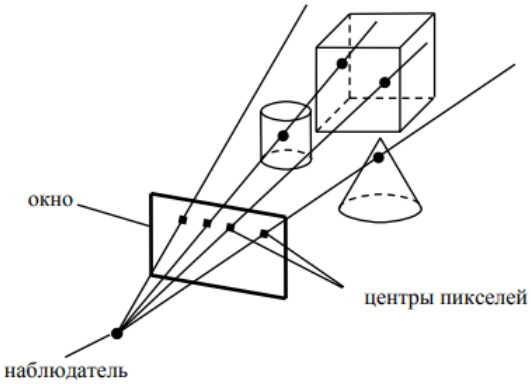

Рисунок 1.3 – Пример работы алгоритма обратной трассировки лучей

Считается, что наблюдатель расположен на положительной полуоси z в бесконечности, поэтому все световые лучи параллельны оси z. В ходе работы испускаются лучи от наблюдателя и ищутся пересечения луча и всех объектов сцены [3]. В результате пересечение с максимальным значением z является видимой частью поверхности и атрибуты данного объекта используются для определения характеристик пикселя, через центр которого проходит данный световой луч.

Для расчета эффектов освещения сцены проводятся вторичные лучи от точек пересечения ко всем источникам света. Если на пути этих лучей встречается непрозрачное тело, значит, данная точка находится в тени.

Несмотря на более высокую эффективность алгоритма в сравнении с прямой трассировкой лучей, данный алгоритм считается достаточно медленным, так как в нем происходит точный расчет сложных аналитических выражений для нахождения пересечения с рассматриваемыми объектами.

1.2 Описание алгоритмов построения теней

Для создания реалистичного изображения в компьютерной графике применяются различные алгоритмы освещения.

Модель освещения предназначена для расчета интенсивности отраженного к наблюдателю света в каждой точке изображения.

Модель освещения может быть:

- локальной - в данной модели учитывается только свет от источников и ориентация поверхности.
- глобальной - в данной модели, помимо составляющих локальной, учитывается еще и свет, отраженный от других поверхностей или пропущенный через них.

Локальная модель включает 3 составляющих:

1. Диффузную составляющую отражения.
2. Отражающую составляющую отражения.
3. Рассеянное освещение.

Выбор алгоритма построения теней напрямую зависит от выбора алгоритма отсечения невидимых ребер и поверхностей, а так же - от выбора модели освещения.

Так в алгоритме трассировки лучей тени получаются практически без дополнительных вычислений, а в алгоритме с Z буфером, к примеру, можно получить тени, используя второй буфер, полученный подменой точки наблюдения на точку источника света.

1.3 Вывод

Оценив все изложенные выше алгоритмы, можно сделать вывод, что для данной работы лучше всего подходит алгоритм обратной трассировки лучей,

так как он точно отражает суть физических явлений, таких как отражение и преломление лучей.

2 Конструкторский раздел

В данном разделе будут рассмотрены требования к программному обеспечению, а также схемы алгоритмов, выбранных для решения поставленной задачи. Так же, будут описаны пользовательские структуры данных и приведена структура реализуемого программного обеспечения.

2.1 Разработка алгоритмов

2.1.1 Основные физические соотношения для дисперсии

Дисперсия света обусловлена зависимостью показателей преломления света от длины волны. Закон Снеллиуса (также Снелля или Снелла) описывает преломление света на границе двух прозрачных сред.

Угол падения света на поверхность связан с углом преломления соотношением (2.1):

$$n_1 sin(\theta_1) = n_2 sin(\theta_2) \tag{2.1}$$

где n_1 — показатель преломления среды, из которой свет падает на границу раздела;

θ_1 — угол падения света — угол между падающим на поверхность лучом и нормалью к поверхности;

n_2 — показатель преломления среды, в которую свет попадает, пройдя границу раздела;

θ_2 — угол преломления света — угол между прошедшим через поверхность лучом и нормалью к поверхности.

Формула Зельмейера (2.2) — это эмпирическая формула описывающая зависимость между показателем преломления и длиной волны для конкретной прозрачной среды. Уравнение используется для определения дисперсии света в этой среде.

$$n^2(\lambda) = 1 + \sum_i \frac{B_i \lambda^2}{\lambda^2 - C_i} \tag{2.2}$$

где n - показатель преломления;

λ - длина волны;

B_i, C_i - экспериментально определяемые коэффициенты Селлмейера.

2.1.2 Пересечение луча и сферы

Уравнение луча представлено ниже:

$$P = O + t\overrightarrow{D}, t \geq 0, \tag{2.3}$$

где $\overrightarrow{D}$ – направление луча.

Сфера — это множество точек P, лежащих на постоянном расстоянии r от фиксированной точки C. Тогда можно записать уравнение, удовлетворяющее этому условию:

$$distance(P, C) = r \tag{2.4}$$

Запишем расстояние (2.4) между P и C как длину вектора из P в C.

$$|P - C| = r \tag{2.5}$$

Заменим на скалярное произведение вектора на себя:

$$\sqrt{\langle P - C\rangle, \langle P - C\rangle} = r \tag{2.6}$$

Избавимся от корня:

$$\langle P - C\rangle, \langle P - C\rangle = r^2 \tag{2.7}$$

В итоге есть два уравнения - уравнение луча и сферы. Найдем пересечение луча со сферой. Для этого подставим (2.3) в (2.7)

$$\langle O + t\overrightarrow{D} - C\rangle, \langle O + t\overrightarrow{D} - C\rangle = r^2 \tag{2.8}$$

Разложим скалярное произведение и преобразуем его. В результате получим:

$$t^2\langle\overrightarrow{D}, \overrightarrow{D}\rangle + 2t\langle\overrightarrow{OC}, \overrightarrow{D}\rangle + \langle\overrightarrow{OC}, \overrightarrow{OC}\rangle - r^2 = 0 \tag{2.9}$$

Представленное квадратное уравнение (2.9) имеет несколько возможных случаев решения. Если у уравнения одно решение, это обозначает, что луч

касается сферы. Два решения обозначают, то что луч входит в сферу и выходит из неё. И если нет решений, значит, луч не пересекается со сферой.

2.1.3 Описание алгоритма обратной трассировки

Суть алгоритма состоит в следующем: Из некоторой точки пространства, называемой виртуальным глазом, или камерой, через каждый пиксель изображения испускается луч и находится точка пересечения с объектом сцены. При обнаружении точки пересечения, формируется отраженный луч и испускается. Данный алгоритм повторяется рекурсивно, пока не достигнем максимальной установленной глубины рекурсии. Полученные цвета перемножаются.

На рисунке 2.1 представлена схема синтеза изображения с применением данного алгоритма.

2.2 Описание используемых типов и структур данных

В данной работе используются следующие типы и структуры данных:

- источник света – задается расположением, направленностью и интенсивностью света;
- математические абстракции:
  - точка – хранит координаты x, y, z
  - вектор – хранит направление по x, y, z
- цвет – хранит три составляющие RGB модели цвета

2.3 Описание структуры программного обеспечения

На рисунке 2.2 представлена диаграмма классов реализуемого программного обеспечения.

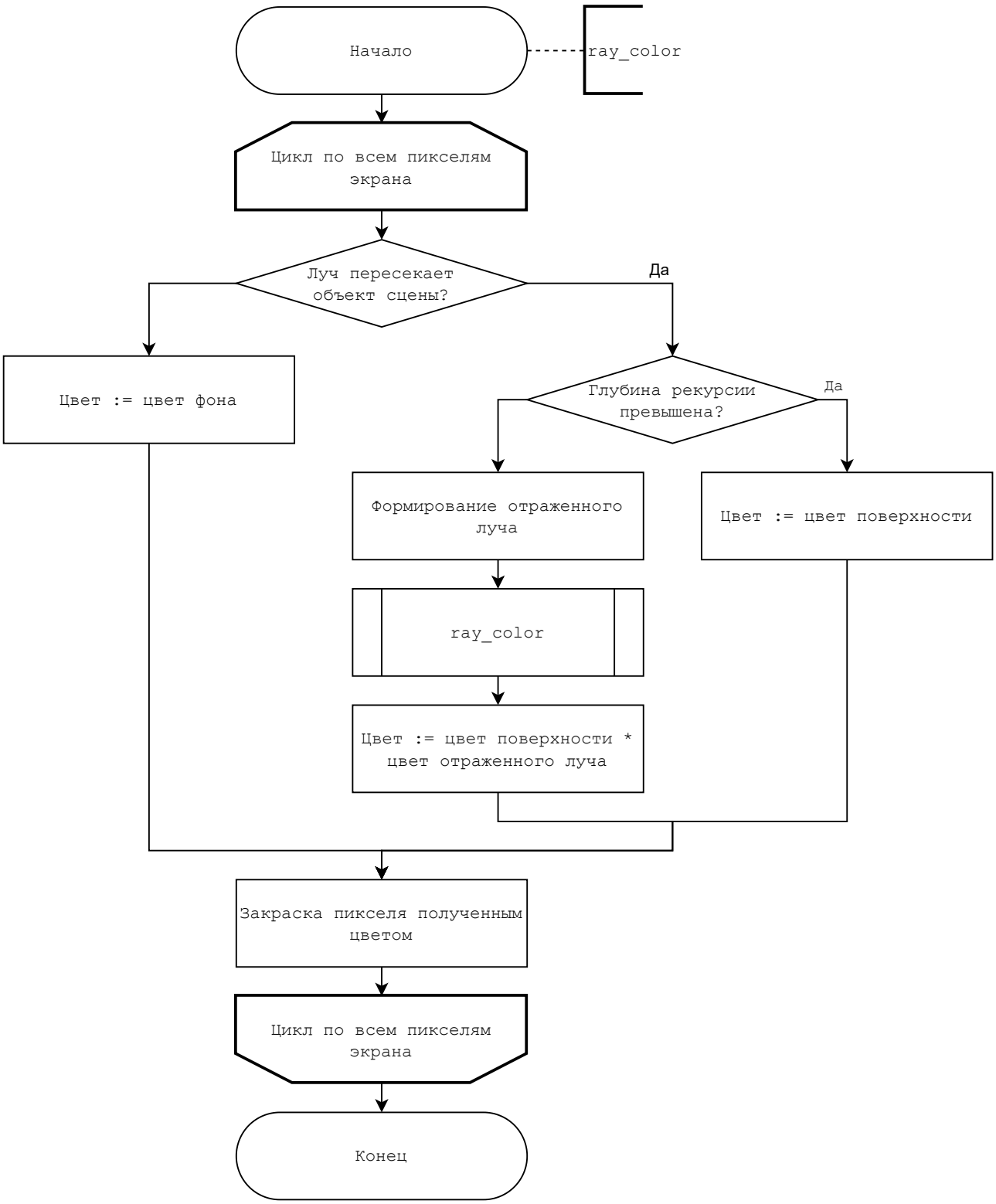

Рисунок 2.1 – Схема алгоритма синтеза изображения с применением алгоритма обратной трассировки лучей

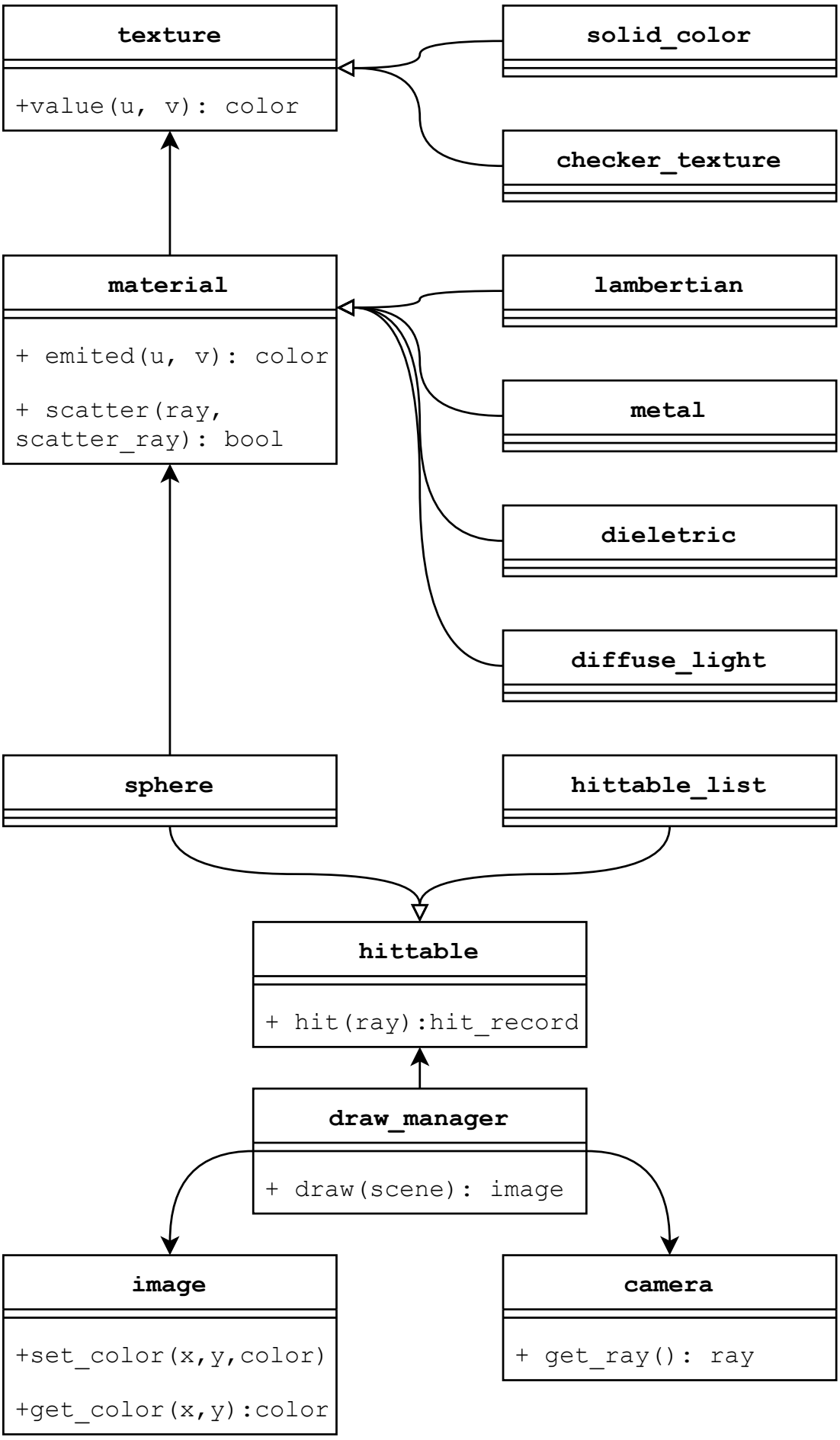

Рисунок 2.2 – UML диаграмма разрабатываемого программного обеспечения

2.4 Вывод

В данном разделе были приведены основные физические соотношения для дисперсии, приведена схема алгоритма обратной трассировки лучей, описаны используемые структуры данных, приведена UML диаграмма разрабатываемого приложения.

3 Технологический раздел

В данном разделе будут представлены средства разработки программного обеспечения, детали реализации и процесс сборки разрабатываемого программного обеспечения.

3.1 Выбор средств программной реализации

В качестве языка программирования для разработки программного обеспечения был выбран язык `C++` [6]. Данный выбор обусловлен тем, что данный язык предоставляет весь функционал требуемый для решения поставленной задачи.

Для создания пользовательского интерфейса ПО был использован фреймворк `QT` [7]. Данный фреймворк содержит в себе объекты, позволяющие напрямую работать с пикселями изображения, а так же возможности создания интерактивных пользовательских интерфейсов, что позволит в интерактивном режиме управлять изображением.

В качестве стиля кода был выбран стиль Mozilla [8]. Для проведения автоформатирования был выбран инструмент `clang-format` [9], так как он поддерживает работу в командной строке, а так же реализован в качестве плагина для популярных ide.

Для отслеживания утечек памяти был выбран инструмент `Valgrind` [10].

Для сборки программного обеспечения использовался инструмент `CMake` [11].

В качестве среды разработки был выбран текстовый редактор `vim` [12], поддерживающий работу в командной строке, а так же установку плагинов [13], в том числе для работы с `C++` и `CMake`.

3.2 Процесс сборки приложения

Для сборки программного обеспечения использовался инструменты `CMake` [11].

Для сборки приложения необходимо в командной строке, находясь в директории проекта, выполнить следующие команды.

Листинг 3.1 – Сборка реализуемого программного обеспечения

```
$ cmake -B build
$ cmake --build build -j $(nproc)
```

3.3 Пользовательский интерфейс

Интерфейс реализуемого ПО представлен на рисунках 3.1 – 3.5.

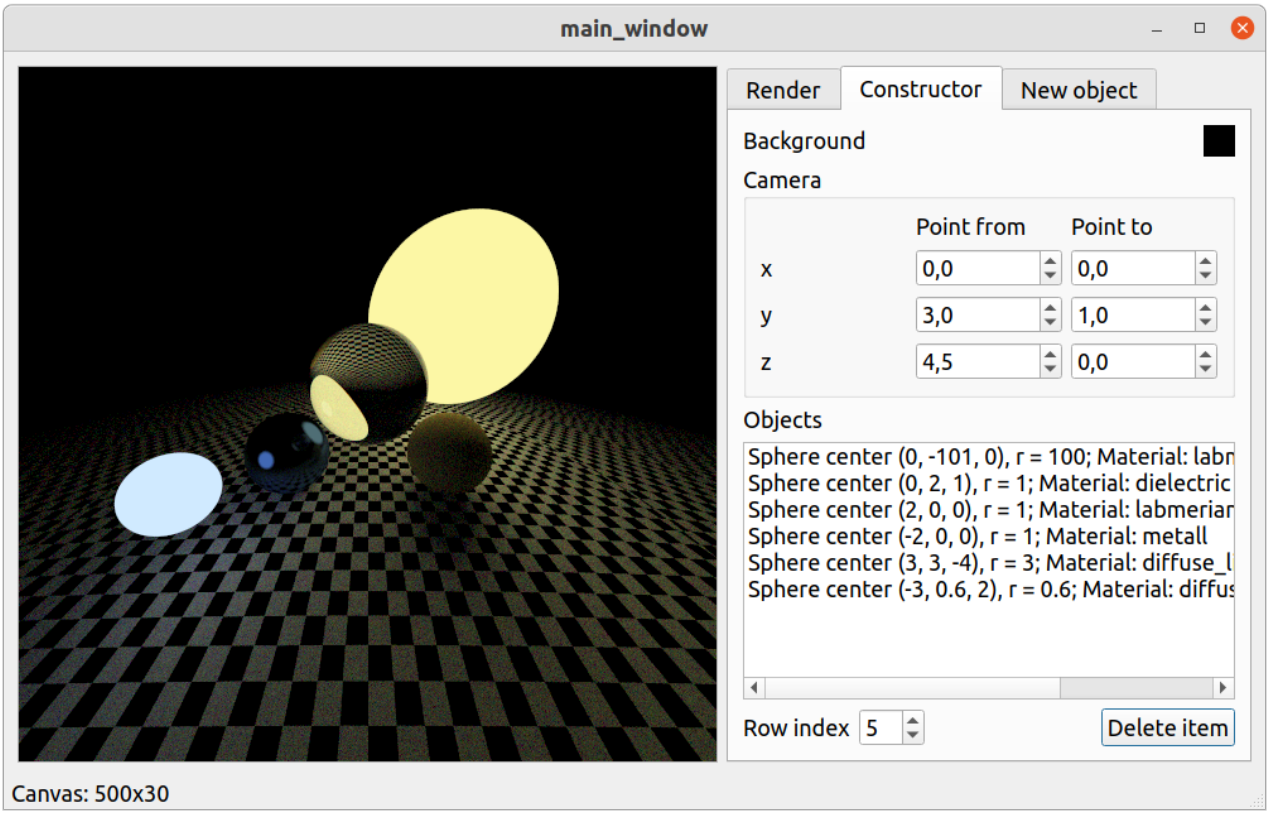

Рисунок 3.1 – Интерфейс программы. Общий план

На рисунке 3.1 изображен общий вид программы. Слева находится экран для вывода изображения. Справа – интерфейс настройки параметров изображения. Рассмотрим каждую из вкладок настройки отдельно.

На рисунке 3.2 представлена вкладка генерирования изображения. На ней доступен выбор качества генерируемого изображения и кнопка запуска генерирования изображения.

На рисунке 3.3 представлена вкладка настройки параметров сцены. Она включает в себя задание цвета заднего фона, настройку положения камеры, а также содержит список объектов сцены с возможностью их удаления.

На рисунках 3.4 – 3.5 представлена вкладка добавления новых объектов для сцены. Она включает в себя выбор фигуры, ее материала и текстуры. Для

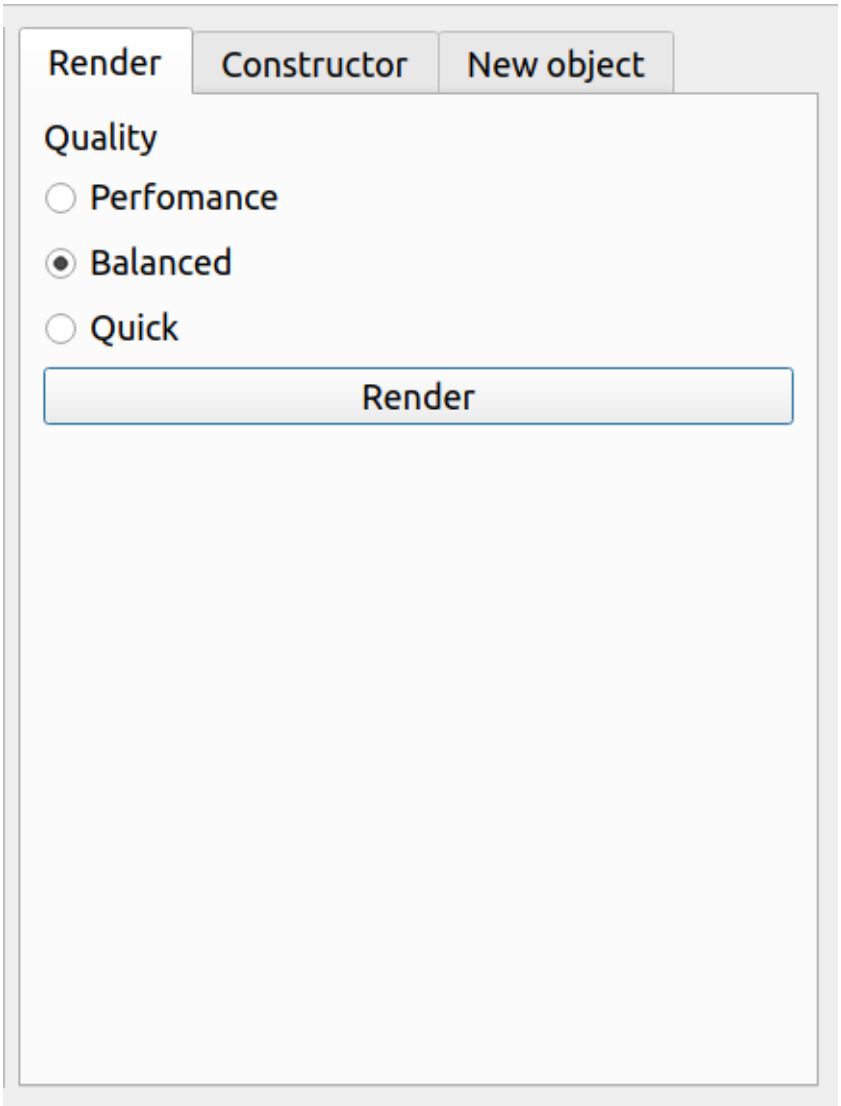

Рисунок 3.2 – Интерфейс программы. Вкладка настроек рендера

прозрачного материала дана возможность ввода коэффициентов формула Зельмейера.

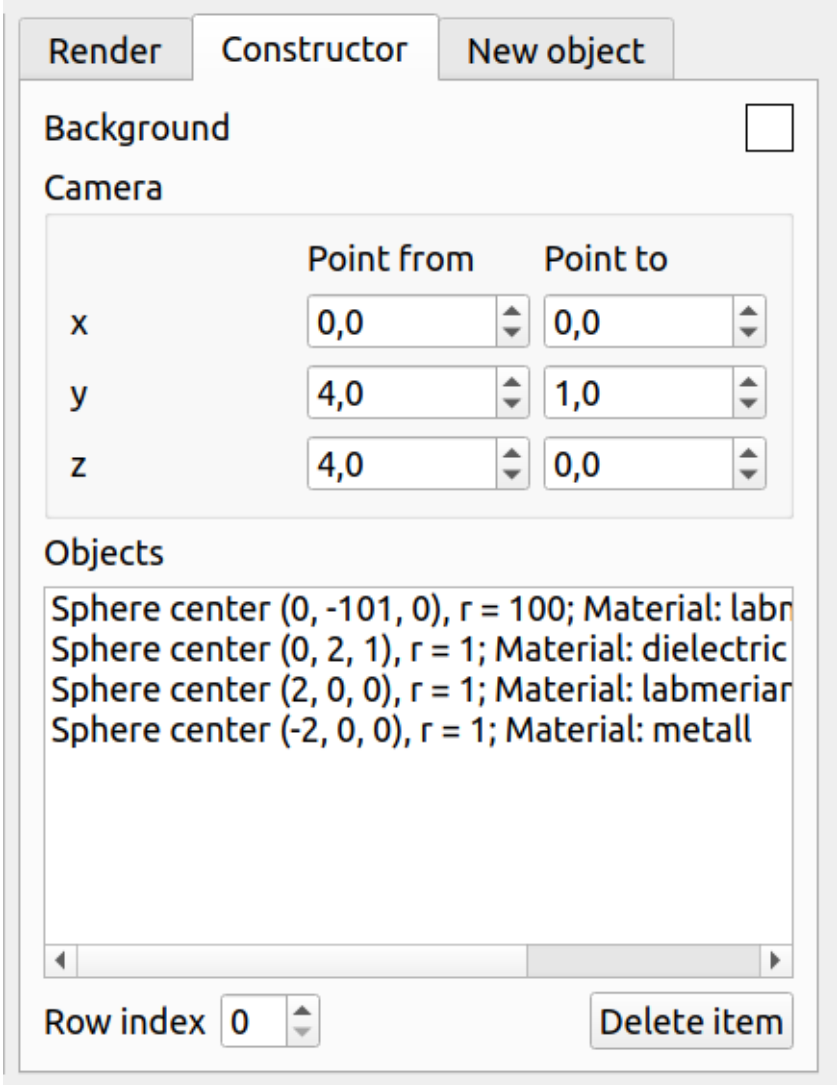

Рисунок 3.3 – Интерфейс программы. Вкладка настроек сцены

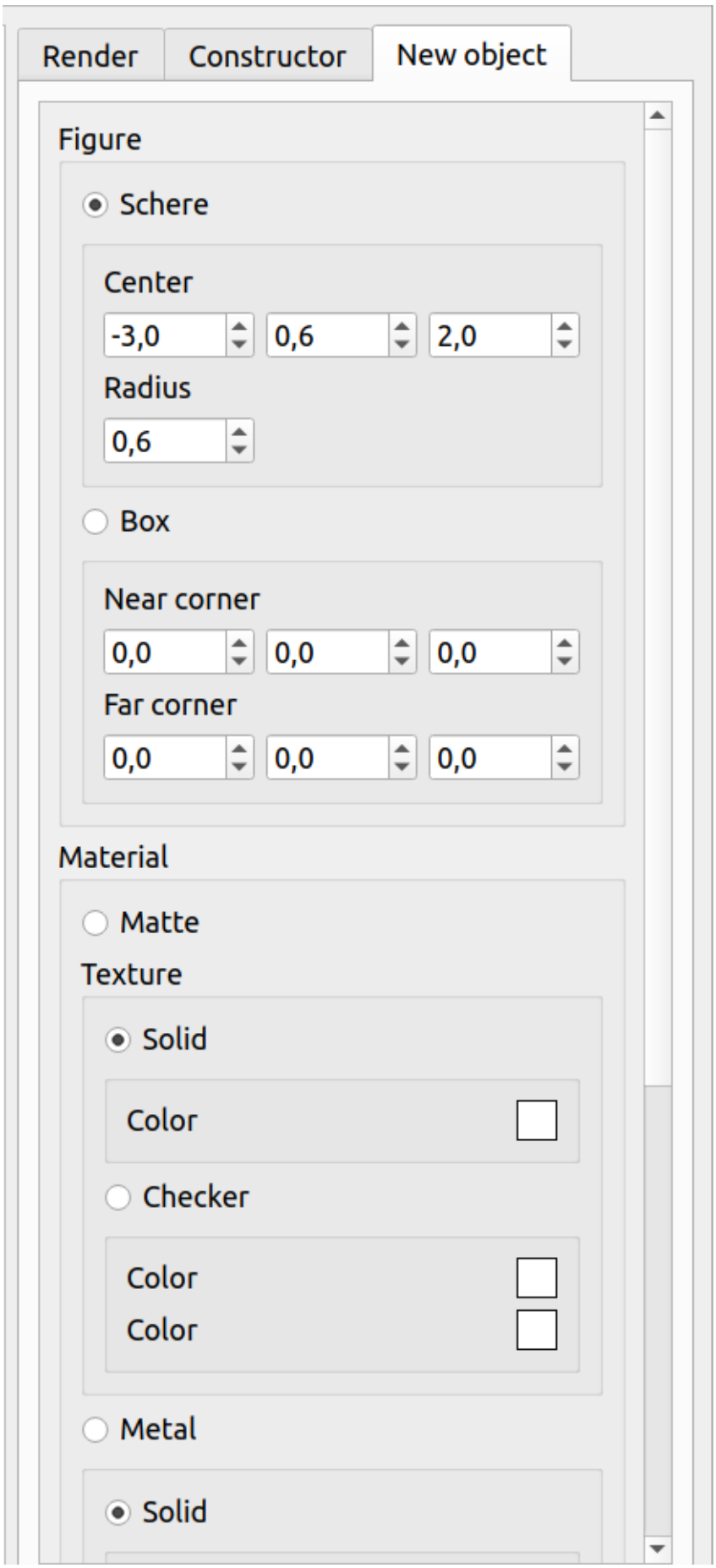

Рисунок 3.4 – Интерфейс программы. Вкладка добавления нового объекта. Часть 1

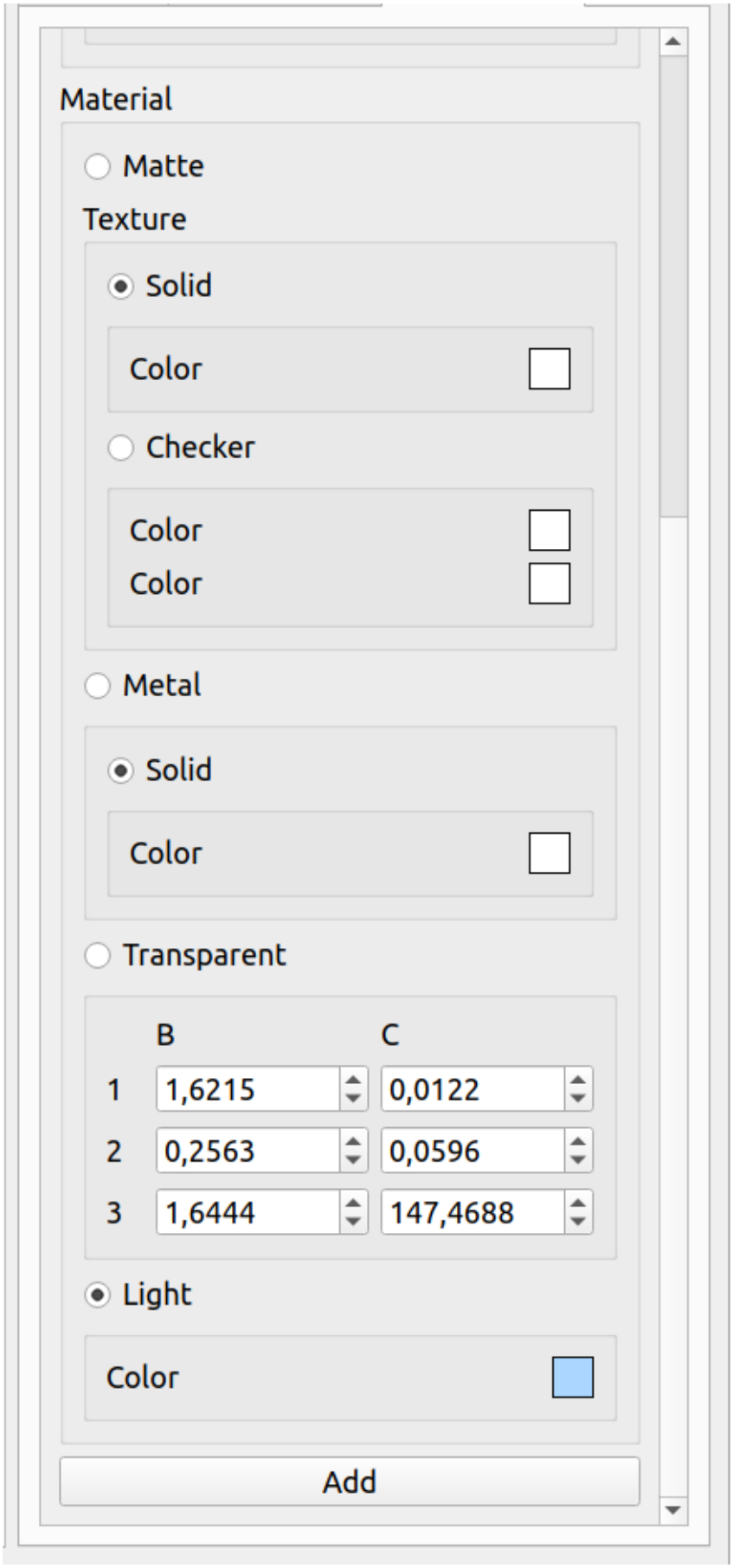

Рисунок 3.5 – Интерфейс программы. Вкладка добавления нового объекта. Часть 2

3.4 Примеры работы приложения

На рисунках 3.6 – 3.8 представлены примеры работы приложения.

На рисунке 3.6 представлена сцена с видимой дисперсией. В сцене используется стеклянный шар на фоне сферы с шахматным узором. Для наблюдения дисперсия камера расположена близко с стеклянной сфере.

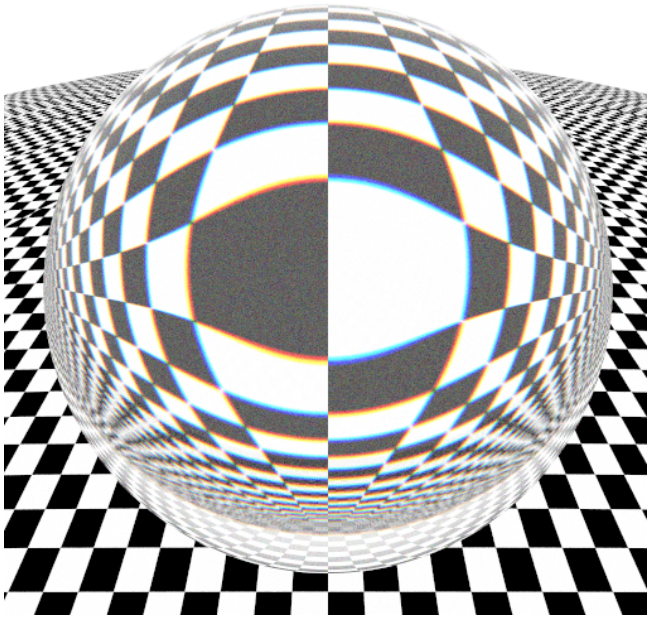

Рисунок 3.6 – Пример работы приложения. Сцена с дисперсией

На рисунке 3.7 представлена сцена с металлическим и матовым шаром. В металлическом фаре наблюдается отражаются другие объекты сцены.

На рисунке 3.8 представлена сцена с двумя источниками света: желтого и синего.

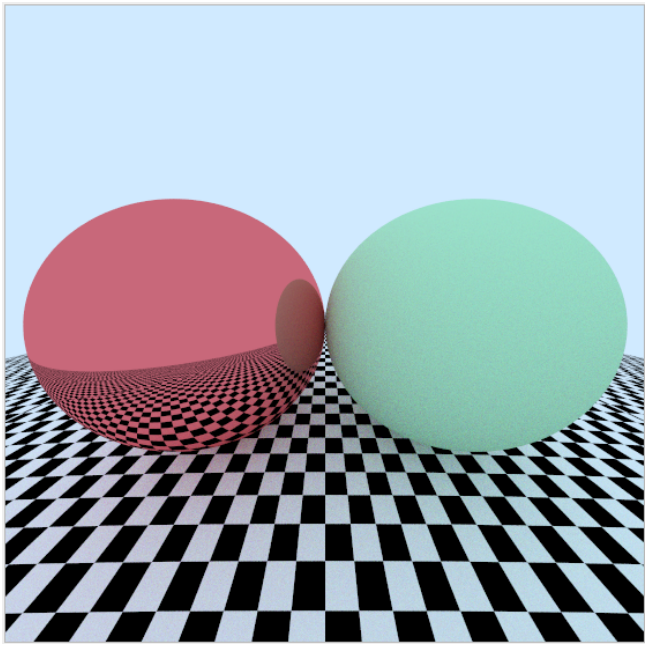

Рисунок 3.7 – Пример работы приложения. Сцена с металлическим и матовым шаром

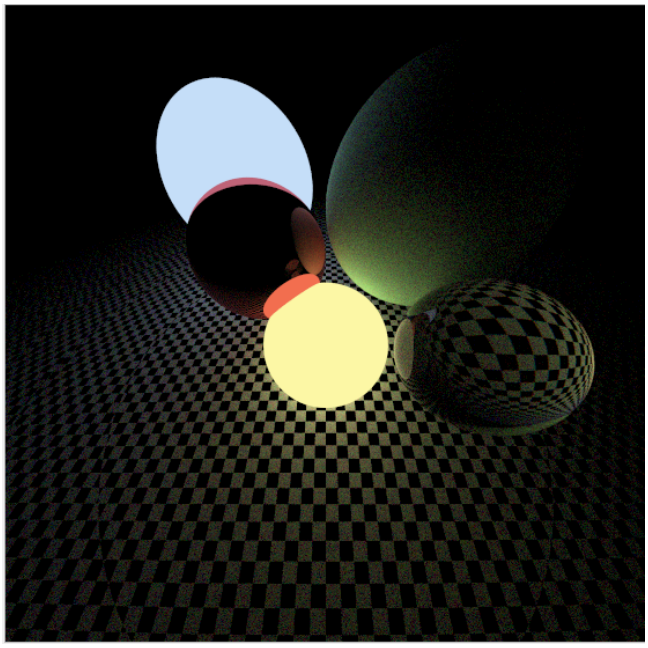

Рисунок 3.8 – Пример работы приложения. Сцена с источниками света

3.5 Вывод

В данном разделе были представлены средства разработки программного обеспечения, детали реализации, пользовательский интерфейс и процесс сборки разрабатываемого программного обеспечения.

4 Экспериментальный раздел

В данном разделе будет поставлен эксперимент, в котором будут сравнены временные характеристики работы реализованного программного обеспечения в различных конфигурациях.

4.1 Цель эксперимента

Рассматривая схему алгоритма обратное трассировки лучей можно заметить, вычисление значения цвета каждого пикселя происходит независимо от других значений. Таким образом данный алгоритм можно распараллелить.

Целью эксперимента является оценка временной эффективности параллельной реализации алгоритма обратной трассировки лучей.

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

- процессор: Intel Core™ i5-8250U [14] CPU @ 1.60GHz;
- память: 8 GiB;
- операционная система: Fedora [15] Linux [16] 21.1.4 64-bit.

Исследование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования.

4.3 Описание эксперимента

Была реализована функция параллельного синтеза сцены. Для этого была использована библиотека `OpenMP` [17], директива препроцессора `#pragma omp parallel for`. Данная директива препроцессора преобразует код для выполнения итераций цикла параллельно.

В рамках данного эксперимента будет производиться оценка влияния размерности изображения и количества объектов сцена на время работы

алгоритма. Для этого будем синтезировать квадратные изображения с размерностями равными [100, 200, 500, 1000, 2000] элементов. Количество объектов сцены будет задано равным [2, 4, 8, 16] штук.

4.4 Результат эксперимента

В таблице 4.1 приведены экспериментально полученные значения временных характеристик работы алгоритма в зависимости от размерности синтезируемого изображения и количества объектов сцены.

Таблица 4.1 – Замеры времени для изображений с различными размерностями

Кол-во объектов	Размерность сцены	Время (мс)	
		Послед. реал.	Паралл. реал.
2	100x100	49	21
	200x200	301	83
	500x500	1606	442
	1000x1000	5068	1853
	2000x2000	20428	8349
4	100x100	135	49
	200x200	380	168
	500x500	2024	672
	1000x1000	7840	2672
	2000x2000	32666	10859
8	100x100	135	38
	200x200	544	165
	500x500	3341	1072
	1000x1000	13009	4233
	2000x2000	51708	17864
16	100x100	272	74
	200x200	1098	302
	500x500	6693	1864
	1000x1000	26108	7451
	2000x2000	102328	31217

Согласно данным, приведенным в таблице 4.1, время синтеза изображения зависит от размерности данного изображения как $O(mn)$ или $O(n^2)$ где m, n - размерности изображения.

Также время синтеза изображения прямо пропорционально количеству объектов сцены, то есть зависит как $O(n)$, где n - количество объектов сцены.

Стоит отметить, что многопоточная реализация алгоритма оказалась более эффективной при всех рассматриваемых размерностях изображения. Связано это с большими размерностями синтезируемых изображений (от

300 до 1000 пикселей). Для изображения малых размерностей, однопоточный алгоритм окажется более эффективным в связи с отсутствием дополнительных затрат по времени и памяти, требуемых для реализации многопоточности (создание потоков, совместный лоступ к ресурсам).

4.5 Вывод

В данном разделе было произведено экспериментально сравнение временных характеристик реализованного программного обеспечения.

Время работы алгоритма имеет квадратичную зависимость от размерности синтезируемого изображения и линейную зависимость от количества объектов сцены.

Наибоее эффективной по времени оказалась многопоточная реализация алгоритма. В среднем время ее работы меньше 3 раза, чем последовательной реализации.

ЗАКЛЮЧЕНИЕ

В ходе курсового проекта было разработано программное обеспечение, предоставляющее возможность визуализации дисперсии света на прозрачных предметах. Разработанное программное обеспечение предоставляет функционал для изменения интервалов построения поверхностей, задания цвета и свойств материала поверхности, а так же задания и изменения в процессе работы положения точки наблюдения и источников света по их характеристикам (положению, интенсивности) в интерактивном режиме. В процессе выполнения данной работы были выполнены следующие задачи:

- изучение явления дисперсии с физической точки зрения;
- определение зависимостей влияющей на преломление лучей света при прохождении через прозрачную поверхность;
- описание существующих алгоритмов построения реалистичных изображений;
- выбор реализуемого алгоритмы;
- приведение схемы реализуемых алгоритмов;
- проектирование архитектуры и графического интерфейса программы;
- реализация структур данных и алгоритмов;
- описание структуры разрабатываемого ПО;
- исследование производительности программы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Роджерс Д.* Алгоритмические основы машинной графики. — 1-е изд. — Д. Роджерс. — Москва «Мир», 1989.
2. *Шикин Е. В.* Компьютерная графика. Динамика, реалистические изображения //. — М.: ДИАЛОГ–МИФИ, 1998. — С. 288.
3. Ю.М. Баяковский. Трассировка лучей из книги Джефа Проузиса [Электронный ресурс]. — Режим доступа: `https://www.graphicon.ru/oldgr/courses/cg99/notes/lect12/prouzis/raytrace.htm` дата обращения: 03.11.2021).
4. Проблемы трассировки лучей – из будущего в реальное время [Электронный ресурс]. — Режим доступа: `https://nvworld.ru/articles/ray-tracing/3/` дата обращения: 03.11.2021).
5. *Снижко Е. А.* Компьютерная графика. Динамика, реалистические изображения //. — Балт. гос. техн. ун-т. – СПб., 2005. — С. 132.
6. Standard C++ [Электронный ресурс]. — Режим доступа: `https://isocpp.org/` (дата обращения: 24.10.2021).
7. Qt | Cross-platform software development for embedded & desktop [Электронный ресурс]. — Режим доступа: `https://www.qt.io/` (дата обращения: 24.10.2021).
8. C++ Coding style – Firefox Source Docs documentation. — Режим доступа: `https://firefox-source-docs.mozilla.org/code-quality/coding-style/coding_style_cpp.html` (дата обращения: 24.12.2021).
9. Clang-Format Style Options – Clang 13 documentation. — Режим доступа: `https://clang.llvm.org/docs/ClangFormatStyleOptions.html` (дата обращения: 24.12.2021).
10. Valgrind Home. — Режим доступа: `https://valgrind.org/` (дата обращения: 2.11.2021).
11. CMake. — Режим доступа: `https://cmake.org/` (дата обращения: 3.11.2021).
12. welcome home : vim online. — Режим доступа: `https://vim.org` (дата обращения: 24.12.2021).

13. Vim Awesome. — Режим доступа: https://vimawesome.com/ (дата обращения: 24.12.2021).

14. Процессор Intel® Core™ i5-8250U [Электронный ресурс]. — Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/124967/intel-core-i5-8250u-processor-6m-cache-up-to-3-40-ghz.html (дата обращения: 24.10.2021).

15. Fedora [Электронный ресурс]. — Режим доступа: https://getfedora.org/en/ (дата обращения: 24.12.2021).

16. LINUX.ORG.RU – Русская информация об ОС Linux [Электронный ресурс]. — Режим доступа: https://www.linux.org.ru/ (дата обращения: 24.10.2021).

17. Директивы OpenMP | Microsoft Docs. — Режим доступа: https://docs.microsoft.com/ru-ru/cpp/parallel/openmp/reference/openmp-directives?view=msvc-170 (дата обращения: 24.10.2021).