СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В XXI веке с возрастанием компьютерных возможностей и с расширением применения компьютерных технологий в различных сферах растёт так же необходимость в графическом моделировании. Компьютерная графика используется практически во всех научных и инженерных дисциплинах для наглядности восприятия и передачи информации, а так же в игровой индустрии и кинематографе. [1]

Построение реалистичного изображения мыльных пузырей может понадобиться для моделирования их физической природы.

Целью работы является разработка программного обеспечения для создания реалистичного изображения мыльных пузырей.

Для достижения данной цели требуется решить следующие задачи:

1) описать физическую модель мыльных пузырей;
2) проанализировать и выбрать модели представления объектов;
3) проанализировать и выбрать алгоритмы решения основных задач компьютерной графики: удаления невидимых линий и поверхностей, учёта теней и освещения;
4) спроектировать программное обеспечение;
5) выбрать средства реализации и реализовать спроектированное программное обеспечение;
6) обеспечить возможность тестирования, создать наборы тестов и продемонстрировать работоспособность программы;
7) исследовать характеристики разработанного программного обеспечения.

1 Аналитическая часть

1.1 Физическая модель мыльных пузырей

Мыльный пузырь – это слой воды, ограниченный молекулами мыла с внешней и внутренней стороны, внутри которого заключён газ [2].

Поверхность мыльного пузыря может переливаться, что продемонстрировано на рисунке 1.1.

Рисунок 1.1 – Мыльный пузырь с переливчатой поверхностью [3]

Переливание возникает благодаря физическому явлению, называемому интерференцией в тонких плёнках [2].

Чтобы объяснить физику данного явления, необходимо ввести следующие понятия [4]:

— свет – в физической оптике электромагнитное излучение, распространяемое в виде волны;

— световой луч (далее просто луч) – линия, вдоль которой распространяется световая волна;

— фаза луча – состояние волны в данный момент времени (функция координат и времени);

— длина волны – расстояние между двумя ближайшими друг к другу точками в пространстве, в которых колебания происходят в одинаковой фазе;

— разность хода – разница фаз между двумя волнами;

— интенсивность луча – энергия, переносимая лучом в заданном направлении (от неё зависит яркость излучения).

На рисунке 1.2 смоделирована интерференция в тонких плёнках.

Световой луч B_1 падает на внешний слой плёнки, от него отражается (луч B_2) и преломляется (луч B_3). В свою очередь на внутренний слой падает преломленный луч B_3 и так же отражается (луч B_4) и преломляется (луч B_5). После этого отражённый луч B_4 падает вновь на внешний слой, после чего отражается и преломляется (луч B_6). Лучи B_2 и B_6 оказываются параллельными, таким образом они будут взаимодействовать.

За счёт разности фаз лучей B_2 и B_6 суммарная интенсивность отличается от исходной и наблюдатель видит цвет, отличный от цвета источника [2].

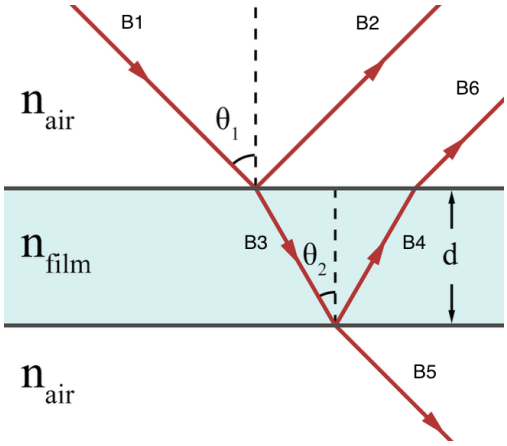

Рисунок 1.2 – Интерференция в тонких плёнках [4]

На рисунке 1.2 присутствуют следующие обозначения:

— n_{air} – коэффицент преломления воздуха;

— n_{film} – коэффицент преломления мыльной воды;

— θ_1 – угол падения луча;

— θ_2 – угол преломления луча при вхождении в мыльную воду;

— d – расстояние между тонкими плёнками

.

Пусть свет, поступающий вдоль B_1 имеет длину волны λ. Пленка имеет толщину d, а показатель преломления пленки равен n_{film}. Угол падения между B_2 и нормалью к поверхности равен θ_1, а угол преломления – θ_2. Свет по лучу B_6 будет сдвинут по фазе относительно света по лучу B_2, а разность фаз будет определять, усиливают или гасят эти волны друг друга. Чтобы найти разность фаз, нужно сначала найти оптическую разность хода.

Свет всегда распространяется медленнее в более плотной среде, поэтому расстояние, пройденное светом в мыльной плёнке, нужно умножить на показатель преломления мыльной воды. Также когда свет переходит из одной среды в среду с более высоким показателем преломления, он претерпевает фазовый сдвиг, равный половине длины его волны.

Разность хода Δ выражается следующим соотношением:

$$\Delta = 2 \cdot d \cdot n_{film} \cdot \cos(\theta_2) - \frac{\lambda}{2}. \tag{1.1}$$

Если в разность хода укладывается чётное число полуволн, то в точке падения луча будет максимум интенсивности света. Если в разность хода укладывается нечётное число полуволн, то в точке падения луча будет минимум интенсивности света.

Разность фаз δ выражается следующим соотношением:

$$\delta = \frac{2 \cdot \pi \cdot \Delta}{\lambda}. \tag{1.2}$$

Для волн с одинаковой интенсивностью итоговая интенсивность I в результате интерференции выражается следующим соотношением:

$$I = 2 \cdot I_0 \cdot (1 + \cos(\delta)). \tag{1.3}$$

1.2 Формализация задачи

На рисунке 1.3 представлена формализованная задача.

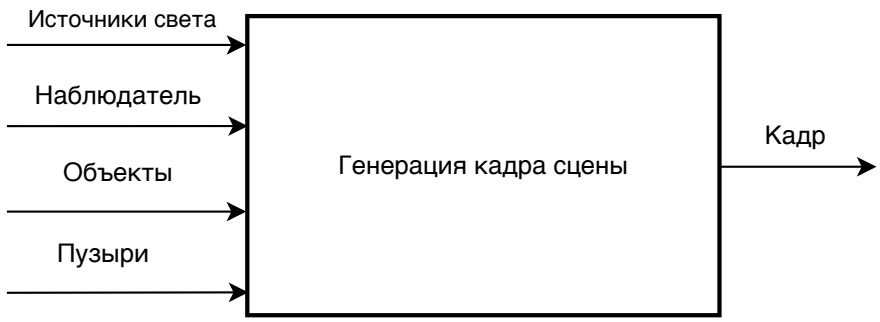

Рисунок 1.3 – Формализованная задача с учётом выбранных алгоритмов

1.3 Формализация объектов сцены

На сцене могут присутствовать следующие типы объектов:

— источники света (задаются расположением и интенсивностью);

— наблюдатель (задаётся расположением и направлением взгляда);

— объекты (задаются моделью описания объектов, набором данных для данной модели, оптическими свойствами, являются выпуклыми);

— пузыри (задаются внешним и внутренним сфрическими слоями, оптическими свойствами).

1.4 Модели описания объектов

Наиболее распространённые модели представления трёхмерных объектов в компьютерной графике: *аналитическая*, *полигональная*, *воксельная*, *равномерная сетка*, *неравномерная сетка* [5], [6], [7].

1.4.1 Аналитическая модель

Аналитической моделью называют описание поверхности с помощью математических формул, таких как $z = f(x, y)$ или $F(x, y, z) = 0$ [5].

Свойствами данной модели являются:

— возможность вычисления пересечения луча с объектом решением системы уравнений;

— возможность вычисления нормали в аналитическом виде при условии гладкости функции;

— необходимось составления объекта из нескольких поверхностей, если поверхность нельзя описать аналитически;

— возможность хранения только коэффициентов формул, задающих поврхности, при условии, что известен общий вид поверхностей;

— отсутствие погрешности при задании сферического объекта.

1.4.2 Полигональная модель

Для описания трёхмерных объектов в данной модели используются следующие элементы [5]:

— точка – объект, размер которого не имеет значения;

— вектор – объект, задаваемый двумя точками;

— полилиния – объект, состоящий из векторов;

— полигон – объект, имеющий площадь, ограниченную замкнутой полилинией;

— полигональная поверхность – объект, состоящий из полигонов.

Свойствами данной модели являются:

— необходимость нахождения пересечения луча со всеми полигонами для поиска пересечения луча с объектом;

— необходимость вычисления нормали для каждого полигона;

— возможность описания объектов произвольной формы;

— присутствие погрешности при задании сферического объекта;

— необходимость хранения информации о каждом полигоне.

1.4.3 Воксельная модель

Воксельной моделью называют модель описания объектов в виде трехмерного массива кубических элементов [5].

Свойствами данной модели являются:

— необходимость нахождения пересечения луча со всеми гранями вокселей для поиска пересечения луча с объектом;

— необходимость вычисления нормалей для граней каждого вокселя;

— возможность описания объектов произвольной формы;

— присутствие погрешности при задании сферического объекта;

— необходимость хранения информации о каждом вокселе.

1.4.4 Равномерная сетка

Неравномерной сеткой называют модель описания поверхности в виде матрицы, каждый элемент которой хранит значение высоты узла сетки.

Свойствами данной модели являются:

— необходимость применения интерполяции для поиска пересечения луча с объектом;

— необходимость вычисления нормали для каждого узла относительно расположения окружающих его узлов;

— невозможность описания замкнутых объектов;

— присутствие погрешности при задании сферического объекта;

— необходимость хранения информации о каждом узле.

1.4.5 Неравномерная сетка

Неравномерной сеткой называют модель описания поверхности в виде множества отдельных точек, принадлежащих поверхности [5].

Свойствами данной модели являются:

— необходимость применения интерполяции для поиска пересечения луча с объектом;

— необходимость вычисления нормали для каждого узла относительно расположения окружающих его узлов;

— возможность описания замкнутых объектов;

— необходимость изменения расположения каждого узла для изменения геометрии объекта;

— необходимость хранения информации о каждом узле.

1.4.6 Сравнение моделей описания объектов

В таблице 1.1 представлены результаты сравнения моделей описания объектов и использованы следующие обозначения:

— А – аналитическая модель;

— П – полигональная модель;

— В – воксельная модель;

— РС – равномерная сетка;

— НС – неравномерная сетка;

— N – количество полигонов;

— M – количество вокселей;

— K – количество соседних троек точек.

Таблица 1.1 – Сравнение моделей

	А	П	В	РС	НС
Временная сложность поиска нормали	$O(1)$	$O(N)$	$O(M)$	$O(K)$	$O(K)$
Временная сложность поиска пересечения	$O(1)$	$O(N)$	$O(M)$	$O(K)$	$O(K)$
Пространственная сложность хранения объектов	$O(1)$	$O(N)$	$O(M)$	$O(K)$	$O(K)$
Возможность описания произвольных объектов	-	+	+	-	+
Отсутствие погрешности при задании сферического объекта	+	-	-	-	-

По всем параметрам, кроме хранения произвольных объектов выигрывает аналитическая модель. Так как пузыри имеют сферическую форму, которую возможно описать аналитически, в работе использована аналитическая модель. Однако для демонстрации оптических свойств мыльных пузырей относительно других объектов, так же введена полигональная модель, так как она позволяет описывать объекты разных форм.

1.5 Модель освещения

Модель освещения определяет интенсивность I в каждой точке [8].

Наиболее распространёнными являются 2 модели освещения: *локальная* и *глобальная* [9], [8], [10], [11].

Локальная модель освещения рассматривает каждый объект отдельно и не учитывает взаимное расположение между ними [8]. Так как для генерации мыльных пузырей необходимо учитывать взаимодействие между внешним и внутренним слоем мыльной плёнки для создания интерференции, данная модель освещения не подходит.

Глобальная модель освещения учитывает взаимное расположение между объектами, поэтому будет использоваться она [9], [10], [11].

Интенсивность I в точке P в глобальной модели освещения складывается из следующих компонент:

— фоновое освещение (ambient), существующее в любой области сцены и не зависящее от координат точки P и источников света;

— рассеянный свет (diffuse), распространяющийся равномерно во все стороны при попадании на поверхность и зависящий от ориентация поверхности (нормали N), направления на каждый источник света L и интенсивности источников света I;

— зеркальная составляющая (specular), зависящая от того, насколько близки направления на наблюдателя V и отраженного луча R, интенсивности источников света I, а также учитывающая интенсивность отражённого луча I_s;

— преломлённая составляющая (refract), учитывающая интенсивность преломлённого луча I_r;

Тогда итоговая интенсивность в глобальной модели освещения рассчитывается следующим образом [11]:

$$I = k_a I_a + k_d \sum_j I_j (N \cdot L_j) + k_s \sum_j I_j (V \cdot R_j)^\alpha + k_s I_s + k_r I_r, \qquad (1.4)$$

где

k_a – коэффициент фонового освещения;

k_d – коэффициент диффузного отражения;

k_s – коэффициент зеркального отражения;

k_r – коэффициент пропускания;

I_a – интенсивность фонового освещения;

I_j – интенсивность j-го источника света;

I_s – интенсивность отражённого луча;

I_r – интенсивность преломлённого луча;

$\vec{L_j}$ – вектор, направленный к j-му источнику света;

$\vec{N}$ – вектор нормали в точке;

$\vec{R}$ – вектор отраженного луча;

$\vec{V}$ – вектор, направленный на наблюдателя;

α – коэффициент блеска.

1.6 Алгоритмы удаления невидимых линий и поверхностей

Наиболее распространённые алгоритмы удаления невидимых линий и поверхностей: *Робертса, с использованием Z-буфера, художника, Варнока, Вейлера — Азертона, трассировки лучей* [11—17].

1.6.1 Алгоритм Робертса

Этапы алгоритма Робертса [11]:

1) Определить грани тела, перекрываемые самим телом.

2) Определить рёбера, перекрываемые другими телами.

3) В случае наличия перекрытия найти видимую часть ребра.

Работа с гранями и рёбрами говорит о том, что данный алгоритм можно использовать только для тел, заданных полигональной моделью.

Временная сложность алгоритма равна $O(N^2)$, где N – количество граней.

1.6.2 Алгоритм с использованием z-буфера

В алгоритме с использованием z-буфера, есть *буфер кадра* ($frame\_buf$), хранящий интенсивность для каждого пикселя и *z-буфер* (z_buf), хранящий глубину видимого пикселя.

Этапы алгоритма с использованием z-буфера [11]:

1) Заполнить $frame\_buf$ фоновым значением интенсивности.

2) Заполнить z_buf минимальным значением z.

3) Преобразовать каждое тело в растровую форму.

4) Для каждого пикселя (x, y) в теле вычислить его глубину $z(x, y)$.

5) Сравнить глубину $z(x, y)$ со значением $z\_buf(x, y)$.

6) Если $z(x, y) > z\_buf(x, y)$, то записать атрибут этого тела (интенсивность, цвет и т. д.) в $frame\_buf(x, y)$ и заменить $z\_buf(x, y)$ на $z(x, y)$. В противном случае никаких действий не производить.

Данный алгоритм можно использовать для обработки как аналитически заданных тел, так и для тел, заданных полигональной моделью.

Так как размер экрана ограничен, временная сложность алгоритма равна $O(WHN)$, где W – ширина экрана в пикселях, H – высота экрана в пикселях, N – количество граней и аналитических поверхностей.

1.6.3 Алгоритм художника

Этапы алгоритма алгоритма художника [11]:

1) Отсортировать грани по глубине (от дальней до ближайшей к наблюдателю).

2) Отрисовать грани в отсортированном порядке.

Данный алгоритм не справляется с циклическим перекрытием тел друг друга. Кроме того, необходимо учитывать неоднозначность выбора порядка сортировки.

Так как алгоритм работает с гранями, его можно использовать только для тел, заданных полигональной моделью.

Сложность алгоритма равна $O(N)$, где N – количество граней, однако необходимо также учитывать предварительную сортировку.

1.6.4 Алгоритм Варнока

В алгоритме Варнока используется прямоугольная область, которая изначально равна размеру экрана. В ходе работы алгоритма область может разбиться на 4 равые части, потом каждая часть аналогично рассматривается отдельно и может разбиться ещё на 4 равные части и так далее.

Этапы алгоритма Варнока [13]:

1) Проверить область на соответствие каждому из случаев:

   — Проекция ни одной грани не попадает в область, тогда закрасить область цветом фона.

   — Проекция только одной грани содержится в области или пересекает её, тогда закрасить проекцию грани, а остальное закарсить цветом фона.

   — Существует грань, проекция которой полностью накрывает данную область, и эта грань расположена к картинной плоскости ближе, чем все остальные грани, проекции которых пересекают данную область, тогда закрасить область цветом данной грани.

2) Если область не соответствует ни одному пункту, то разбить её на 4 части и повторить предыдущие шаги.

Так как алгоритм работает с гранями, его можно использовать только для тел, заданных полигональной моделью.

Временная сложность алгоритма равна $O(WHN)$, где W – ширина экрана в пикселях, H – высота экрана в пикселях, N – количество граней.

1.6.5 Алгоритм Вейлера — Азертона

Этапы алгоритма Вейлера — Азертона [11]:

1) Отсортировать грани по глубине (от дальней до ближайшей к наблюдателю).

2) Из списка выбирается ближайшая грань и по границам её проекции остальные грани разбиваются на части, которые лежат полностью внутри (F_in) выбранной грани или снаружи (F_{out}) её.

3) Если в F_{in} есть части, которые ближе текущей грани (циклическое экранирование), тогда каждая такая часть используется для разбиения всех граней из множества F_{in}.

4) Повторить предыдущие шаги с F_{out}.

Так как алгоритм работает с гранями, его можно использовать только для тел, заданных полигональной моделью.

Временная сложность алгоритма равна $O(N^2)$, где N – количество граней.

1.6.6 Алгоритм трассировки лучей

Идея, лежащая в основе алгоритма трассировки лучей, заключается в том, что наблюдатель видит любой объект посредством испускаемого неким источником света лучом, который падает на этот объект и затем доходит до наблюдателя. Однако обычно используют алгоритм обратной трассировки лучей, в котором лучи пускаются от наблюдателя до источника света [11].

Этапы алгоритма трассировки лучей для каждого пикселя:

1) Найти уравнение трассируемого луча, проходящего через рассматриваемый пиксель и положение взгляда наблюдателя.

2) Найти точку пересечения луча с ближайшим объектом сцены, закрасить пиксель цветом данного объекта и перейти к следующему пикселю.

3) Если нет пересечений, то закрасить пиксель цветом фона и перейти к следующему пикселю.

Данный алгоритм позволяет учесть преломления и отражения. Для этого необходимо не останавливаться на достижении лучом объекта, а трассировать отражённый и преломлённый лучи.

Данный алгоритм можно использовать для обработки как аналитически заданных тел, так и для тел, заданных полигональной моделью.

Сложность алгоритма равна $O(WHN)$, где W – ширина экрана в пикселях, H – высота экрана в пикселях, N – количество граней и аналитических поверхностей..

1.6.7 Сравнение алгоритмов

В таблице 1.2 представлены результаты сравнения алгоритмов и использованы следующие обозначения:

— Р – алгоритм Робертса;

— ЗБ – алгоритм с использованием z-буфера;

— Х – алгоритм художника;

— В – алгоритм Варнока;

— ВА – алгоритм Вейлера — Азертона;

— ОТ – алгоритм обратной трассировки лучей;

— W – ширина изображения в пикселях;

— H – высота изображения в пикселях;

— N – количество граней и аналитических поверхностей.

Таблица 1.2 – Сравнение алгоритмов

	Р	ЗБ	Х	В	ВА	ОТ
Возможность построения отражений и преломлений	-	-	-	-	-	+
Возможность использования без сортировки	+	+	-	+	-	+
Возможность использования для аналитических объектов	-	+	-	-	-	+
Временная сложность	$O(N^2)$	$O(WHN)$	$O(N)$	$O(WHN)$	$O(N^2)$	$O(WHN)$

Только алгоритм обратной трассировки лучей (далее трассировки лучей) позволяет реализовать отражение и преломление, может работать

с объектами, заданными аналитически, а также учитывать глобальную модель освещения, поэтому в данной работе будет использован именно он, хотя он не является самым эффективным из рассмотренных алгоритмов.

1.7 Построение теней

Алгоритмы затенения в случае точечных источников света идентичны алгоритмам удаления невидимых линий и поверхностей, однако точка наблюдения перемещается в источник света [18], [19]. Тогда точки, невидимые из источников света, являются затенёнными.

Так как был выбран алгоритм трассировки лучей, можно использовать его и для поиска теней. В отличие от задачи удаления невидимых линий и поверхностей, для построения теней луч пускается от рассматриваемой точки до источника света. При наличии объектов на пути луча, даннная точка затенена.

1.8 Формализация задачи с учётом выбранных алгоритмов

На рисунке 1.4 представлена формализованная задача с учётом выбранных алгоритмов.

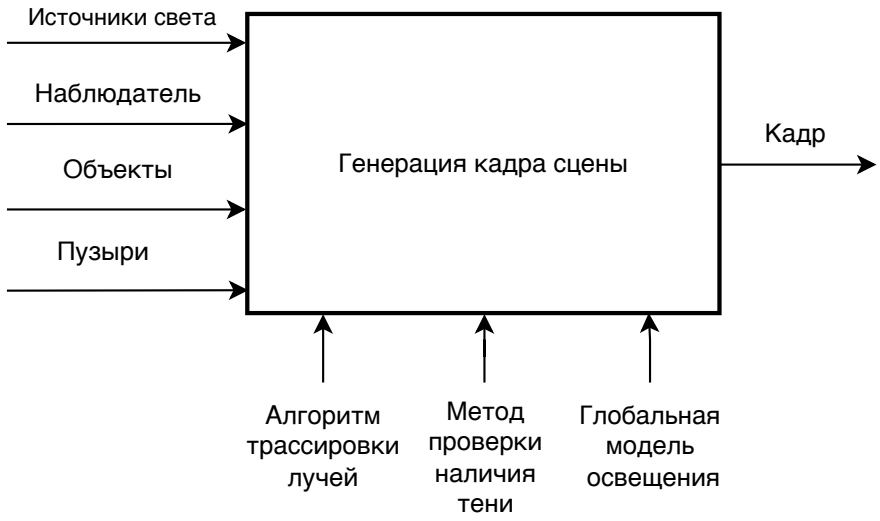

Рисунок 1.4 – Формализованная задача с учётом выбранных алгоритмов

1.9 Выводы из аналитической части

В аналитической части была представлена физическая модель мыльных пузырей, проанализированы модели представления объектов и алгоритмы решения основных задач компьютерной графики. По результатам проведённого анализа были выбраны аналитическая и полигональная модели представления объектов и алгоритм трассировки лучей для удаления невидимых линий и поверхностей и построения теней.

2 Конструкторская часть

2.1 Функциональная модель программного обеспечения

На рисунке 2.1 представлена функциональная модель программного обеспечения.

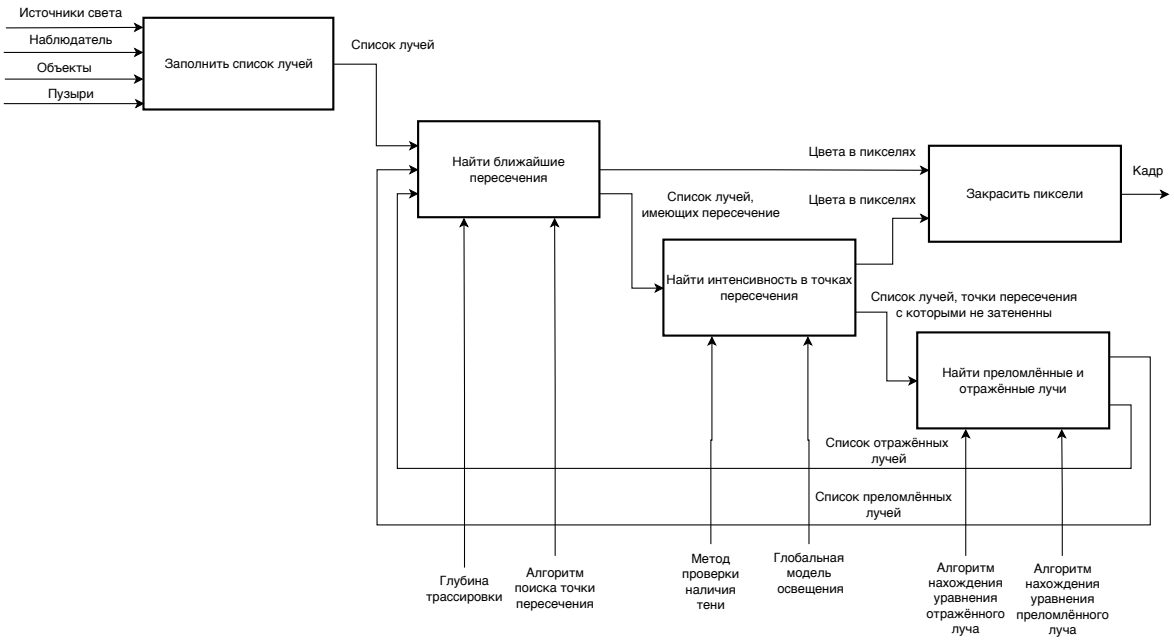

Рисунок 2.1 – Функциональная модель программного обеспечения

2.2 Используемые типы и структуры данных

В данном пункте описываются типы и структуры данных, используемые в программном обеспечении.

1) Сцена, состоящая из:
   - массива объектов;
   - наблюдателя;
   - массива источников света.
2) Объект, состоящий из материала.
3) Триангулированный объект, состоящи из:
   - массива вершин (координаты типа вектор);
   - массива полигонов.
4) Полигон, являющийся массивом из трёх указателей на вершины.
5) Сфера, состоящая из
   - центра (координаты типа вектор);
   - радиуса (положительное вещественное число).
6) Пузырь, состоящий из
   - внешнего слоя (сфера);
   - внутреннего слоя (сфера).
7) Источник света, состоящий из:
   - расположения (координаты типа вектор);
   - интенсивности (вещественное число от 0 до 1).
8) Камера, состоящая из:

— расположения (координаты типа вектор);

— направления взгляда (координаты типа вектор)

9) Материал, состоящий из:

— цвета объекта (три целочисленных переменных в модели RGB);

— коэффициента фонового освещения (вещественное число от 0 до 1);

— коэффициента диффузного освещения (вещественное число от 0 до 1);

— коэффициента зеркального освещения (вещественное число от 0 до 1);

— дисперсия (вещественное неотрицательное число);

— коэффициента преломления (вещественное неотрицательное число);

— показателя отражения (вещественное неотрицательное число);

— показатель преломления (вещественное неотрицательное число).

2.3 Формальное описание алгоритмов

Основные алгоритмы, используемые в программном обеспечении вынесены в схемы, представленные на рисунках 2.2-2.5.

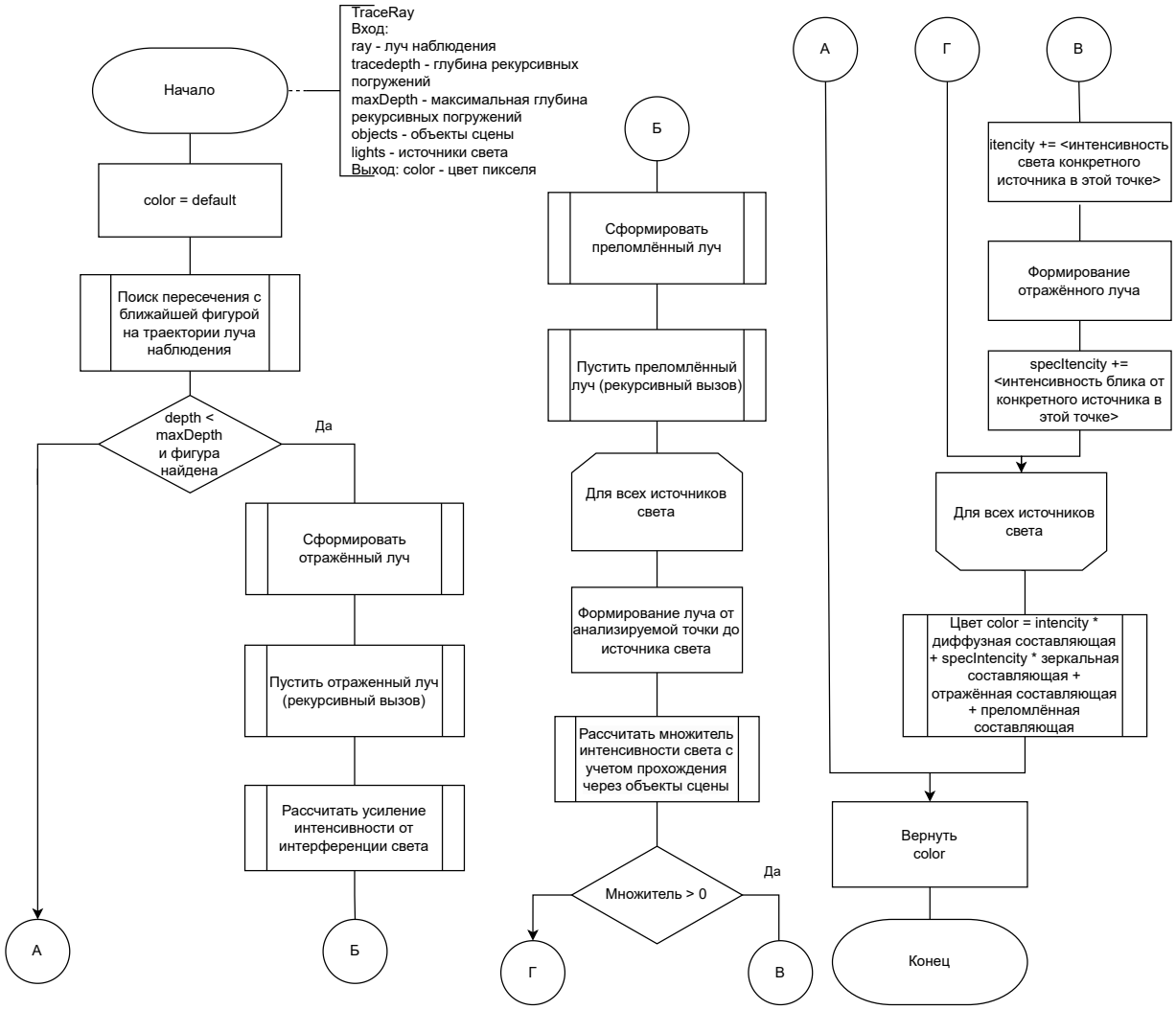

Рисунок 2.2 – Схема алгоритма трассировки лучей

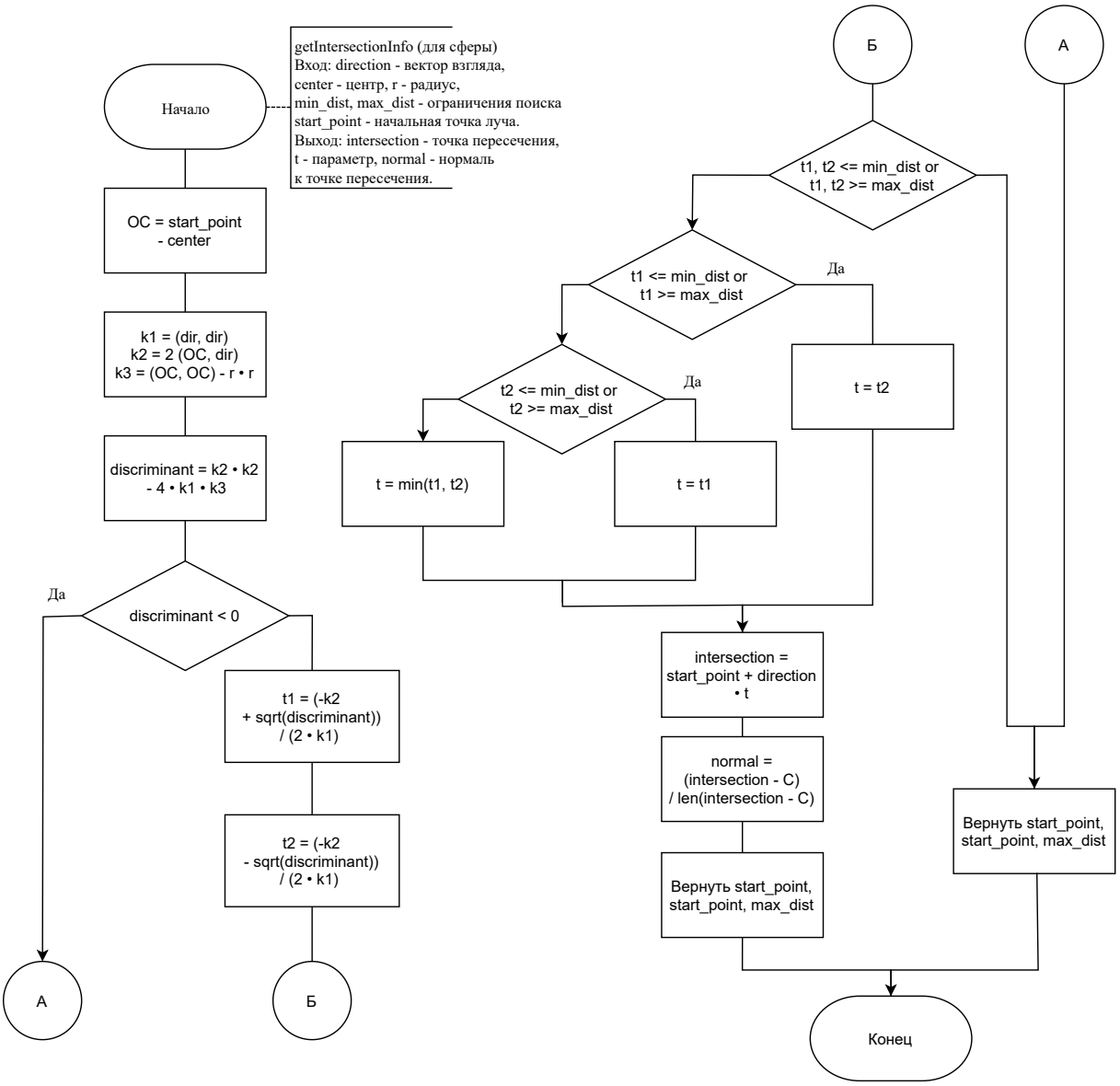

Рисунок 2.3 – Схема алгоритма поиска пересечения луча со сферой

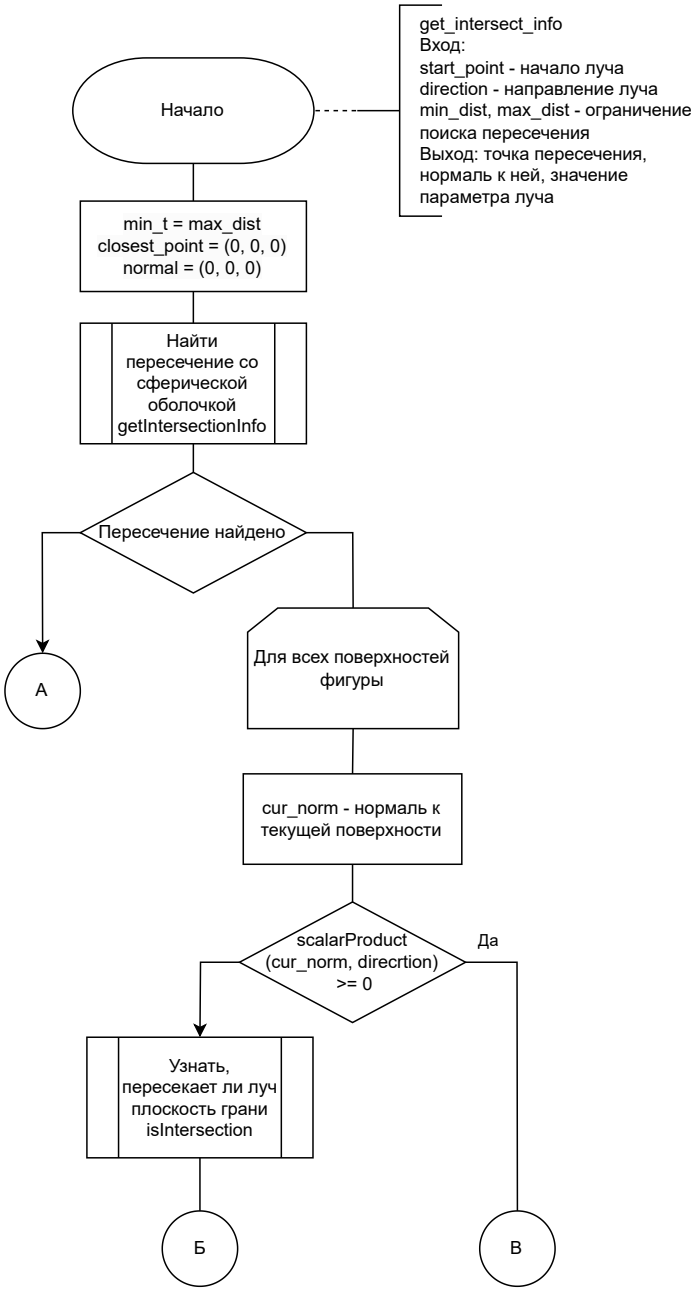

Рисунок 2.4 – Схема алгоритма поиска пересечения луча с триангулированным объектом (часть 1)

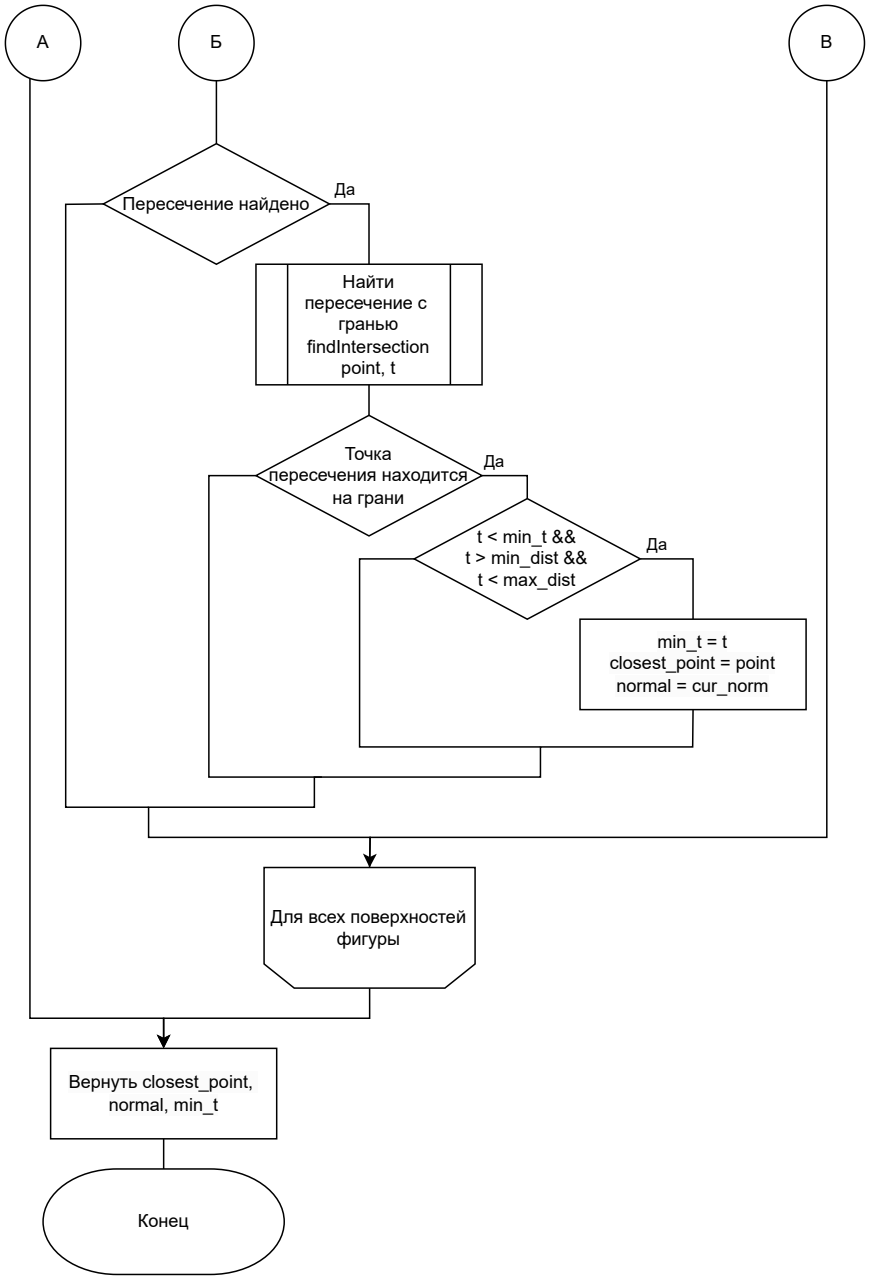

Рисунок 2.5 – Схема алгоритма поиска пересечения луча с триангулированным объектом (часть 2)

2.4 Математические основы алгоритмов

Введём общие обозначения, используемые в формулах:

— V – точка, из которой трассируется луч;

— D – направление трассируемого луча.

Тогда трассируемый луч можно задать параметрическим уравнением:

$$P = V + tD \tag{2.1}$$

Нас интересует только тот случай, когда $t > 0$, потому что тогда объект находится перед наблюдателем.

2.4.1 Поиск пересечения луча со сферой

Для поиска пересечения луча со сферой с центром в точке C и радиусом r необходимо решить следующую систему уравнений:

$$\begin{cases} P = V + tD \\ |P - C| = r \end{cases} \tag{2.2}$$

Второе уравнение можно преобразовать следующим образом:

$$|P - C| = \sqrt{(P - C, P - C)} \tag{2.3}$$

$$(P - C, P - C) = r^2 \tag{2.4}$$

Подставляем первое уравнение во второе:

$$t^2(D, D) + 2t(CO, D) + (CO, CO) - r^2 = 0 \tag{2.5}$$

Введём следующие обозначения:

$$k_1 = (D, D), k_2 = 2(OC, D), k_3 = (CO, CO) - r^2 \tag{2.6}$$

$$k_1 t^2 + k_2 t + k_3 = 0 \tag{2.7}$$

Решениями данного квадратного уравнения являются значения параметра t, при которых луч пересекается со сферой:

$$\{t_1, t_2\} = \frac{-k_2 \pm \sqrt{{k_2}^2 - 4k_1 k_3}}{2k_1} \tag{2.8}$$

2.4.2 Поиск пересечения луча с триангулированным объектом

Для поиска пересечения луча с триангулированным объектом, рассмотрим пересечение с его гранями.

Сначала необходимо проверить, пересекает ли луч плоскость, в которой лежит рассматриваемая грань (не пересекает только если параллелен).

$$(D, n) \neq 0 \tag{2.9}$$

Далее ищем пересечение с плоскостью грани, вершинами которой являются точки A, B и C.

$$t = \frac{(AV, n)}{(D, n)} \tag{2.10}$$

Тогда точка пересечения P будет равна:

$$P = V + tD \tag{2.11}$$

После этого проверяем, лежит ли точка пересечения в пределах грани. Для этого сводим задачу к 2х мерной, так как уже известно, что пересечение лежит в плоскости грани. Для этого координаты по оси OZ у вершин грани и точки пересечения обнуляем.

Далее считаем векторные произведения векторов стороны и векторов, соединяющих первую вершину с точкой пересечения.

$$v_1 = [AB, AP], v_2 = [BC, BP], v_3 = [CA, CP] \tag{2.12}$$

Если все произведения имеют одинаковый знак, то точка лежит внутри грани.

Однако можно сразу отбросить ситуации, когда луч точно не пересечёт триангулированный объект. Для этого проверяется, пересекает ли луч сферу, описанную вокрг него. Тогда если луч не пересекает данную сферу, то он не может пересеч триангулированный объект.

Центр такой сферы совпадает с центром триангулированного объекта. А радиус совпадает с расстоянием от центра до наиболее удалённой от него точки.

2.4.3 Поиск внешней нормали

Внешней нормалью в точке P сферы с центром в точке C является вектор CP.

В триангулированном объекте каждая грань имеет свою нормаль. Найти нормаль грани можно следующим образом:

$$n = [BC, BA] \tag{2.13}$$

Однако нормаль может оказаться внутренней, поэтому необходимо провести корректировку.

Нормаль является внешней если угол между ней и вектором проведённы от вершины к центру объекта тупой, это можно выразить следующим неравенством:

$$(BI, n) < 0 \tag{2.14}$$

Иначе необходимо домножить нормаль на -1.

2.4.4 Поиск отражённого и преломлённого лучей

Если вещество, из которого состоит объект, имеет ненулевой коэффициент зеркального отражения, трассируется луч, начинающийся в точке

пересечения и имеющий направление отражённого луча относительно луча падения.

Если объект имеет ненулевой коэффициент пропускания, трассируется луч, начинающийся в точке пересечения и имеющий направление преломлённого луча относительно луча падения.

Луч падения, отражения, преломления и нормаль к поверхности лежат в одной плоскости. Луч падения и луч отражения имеют одинаковые углы с нормалью.

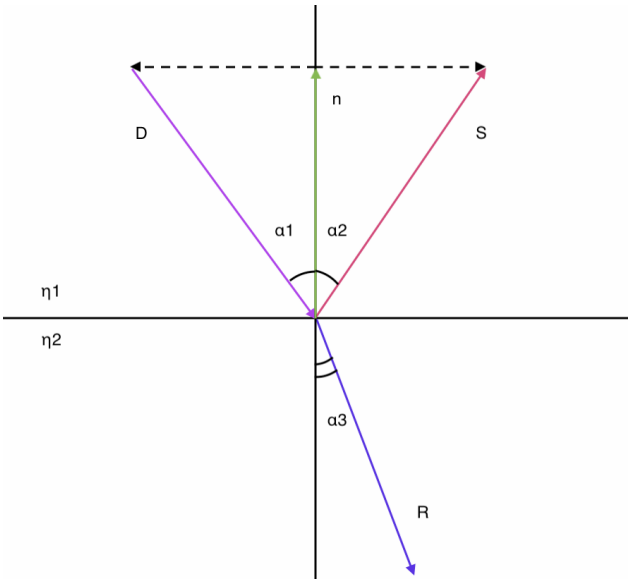

Рисунок 2.6 – Отражённый и преломлённый лучи

Отражённый луч находится следующим образом:

$$S = D - 2n(D, n). \tag{2.15}$$

Пусть η_i – показатели преломления сред, причём $i = \overline{1,2}$. Применяя закон Снеллиуса, преломлённый луч можно найти следующим образом:

$$R = \frac{\eta_1}{\eta_2} D + (\frac{\eta_1}{\eta_2} \cos(\alpha_1) - \cos(\alpha_3))n, \tag{2.16}$$

$$\cos(\alpha_3) = \sqrt{1 - (\frac{\eta_1}{\eta_2})^2 (1 - \cos(\alpha_1))^2}. \tag{2.17}$$

2.5 Выводы из конструкторской части

В конструкторской части была представлена функциональная модель, спроектированно программное обеспечение для генерации изображения и представлены схемы алгоритмов.

3 Технологическая часть

3.1 Формат входных и выходных данных

Входные данные располагаются в текстовом файле определённого формата.

Каждый тип объектов представляется в файле следующим образом:

— Сфера:

1) тип объекта (s) и название;
2) координаты центра (три вещественных числа, задающих координаты соответственно по x, y и z);
3) радиус (вещественное положительное число);
4) оптические параметры вещества.

— Пузырь:

1) тип объекта (b) и название;
2) описание внешней сферы;
3) описание внутренней сферы.

— Триангулированный объект:

1) тип объекта (t) и название;
2) координаты вершины (три вещественных числа, задающих координаты соответственно по x, y и z), перед которыми v;
3) поверхность, описанная номерами трёх вершин (целые числа от 0 до N, где N – количество вершин), перед которыми f;
4) оптические параметры вещества.

— Источник света:

1) тип объекта (l);
2) координаты расположения (три вещественных числа, задающих координаты соответственно по x, y и z);
3) интенсивность (вещественное число от 0 до 1).

Оптические параметры вещества задаются следующим образом:

1) цвет объекта (три целочисленных переменных в модели RGB);
2) коэффициенты фоновый, диффузный, зеркальный и пропускания (вещественные числа от 0 до 1);
3) показатель отражения (вещественное неотрицательное число);
4) показатель преломления (вещественное неотрицательное число);
5) дисперсия (вещественное неотрицательное число).

Листинг 3.1 – Пример входного файла с триангулированным объектом и источником света

```
t тетраэдр
v 500.0 300.0 0
v 700.0 300.0 -300.0
v 300.0 400.0 0
v 300.0 350 -100.0
f 1 3 4
f 1 2 4
f 2 3 4
f 1 2 3
-
255 255 255
0.3 1 0 0
100
1
0

1
1000 1000 1000
0.5
```

Листинг 3.2 – Пример входного файла со сферой и пузырём

```
s сфера
300 400 150
100
0 0 255
0.4 0.1 0 1
100
1
0

b пузырь
450 150 -100
100
100 100 100
0.1 0.4 0 1
100
1.3
100000
450 150 -100
99.9
0 0 0
0 0 0.5 1
0
1
0
```

На выходе пользователь получает изображение в цветовой модели RGB с разрешением 960×960 и расширением PNG.

3.2 Средства реализации

В качестве языка программирования для реализации программного обеспечения был выбран $C++$ [20], так как он позволяет реализовать все алгоритмы, выбранные в результате проектирования и поддерживает все структуры данных.

Был выбран фреймворк Qt [21] для реализации интерфейса программного обеспечения, так как в нём присутствуют инструменты для работы с изображениями и интерфейсом.

3.3 Описание используемых классов

При реализации программы использовался объектно-ориентированный подход. Реализованы следующие классы и структуры:

— класс MainWindow – содержит информацию об интерфейсе и связи кнопок с действиями;

— класс Loader – описывает действия загрузки данных из файла и построения соответствующих объектов на основе прочитанной информации;

— класс Scene – описывает сцену (объекты, источники света), методы взаимодействия со сценой и ее объектами;

— класс Light – описывает расположение и интенсивность источника света, а также методы взаимодействия с ним;

— класс Object – описывает представление трёхмерного объекта в программе и методы работы с ним;

— класс TriangulatedObject – описывает трехмерный объект, состоящий из треугольных полигонов и методы работы с ним;

— класс Triangle – описывает полигон для представления трёхмерного объекта и методы работы с ним;

— класс Sphere – описывает сферу и методы работы с ней;

— класс Bubble – описывает пузырь, содержащий две сферы, и методы работы с ним;

— класс Substance – описывает свойства материала объекта;

— класс Ray – описывает трассируемый луч.

3.4 Примеры реализации алгоритов

В листингах 3.3-3.4 представлена реализация алгоритма трассировки лучей. В листинге 3.5 представлена реализация алгоритма поиска пересечения луча с триангулированным объектом. В листинге 3.6 представлена реализация алгоритма поиска пересечения луча со сферой.

Листинг 3.3 – Трассировка лучей (часть 1)

```
1  QColor Scene::TraceRay(Ray ray, double min_dist, double max_dist, int
       trace_depth, bool where)
2  {
3      QColor color = QColor(0, 0, 0);
4      auto [flag, closest, intersection, normal] =
           findClosestIntersection(ray.start_point, ray.direction,
           min_dist, max_dist);
5      if (flag == NOTHING) return color;
6      if (trace_depth == max_trace_depth) return color;
7      double reflectK = 0.0;
8      QColor reflectColor = QColor(0, 0, 0);
9
10     if (objects[closest]->getSubstance().getCoefficients() .specular >
           EPS)
11     {
12         QVector3D reflectDir = getReflectVector(ray.direction,
               normal).normalized();
13         Ray reflectRay = {
14             .start_point = intersection,
15             .direction = reflectDir,
16             .color = ray.color,
17             .len = ray.len,
18             .n = ray.n,
19         };
20         reflectColor = TraceRay(reflectRay, min_dist, max_dist,
               trace_depth + 1, where);
21         double rC = getInterferenceColor(ray.color,
               objects[closest]->getSubstance());
22         reflectK = getInterferenceStrengthCoef(objects[closest]
               ->getThickness(), rC, ray.direction, normal, ray.len);
23     }
24
25     QColor refractColor = QColor(0, 0, 0);
```

Листинг 3.4 – Трассировка лучей (часть 2)

```
1     if (objects[closest]->getSubstance().getCoefficients(). refract >
          EPS)
2     {
3         double rC = getInterferenceColor(ray.color,
              objects[closest]->getSubstance());
4         QVector3D refractDir = getRefractVector(ray.direction, normal,
              rC, ray.n, where).normalized();
5         Ray refractRay = {
6             .start_point = intersection,
7             .direction = refractDir,
8             .color = ray.color,
9             .len = ray.len,
10            .n = rC,
11        };
12        refractColor = TraceRay(refractRay, min_dist, max_dist,
              trace_depth + 1, !where);
13    }
14
15    double intensity = 0.0;
16    double specIntensity = 0.0;
17
18    for (int i = 0; i < lights.size(); i++)
19    {
20        QVector3D light = intersection - lights[i].getView();
21        double k = findLightIntensity(intersection, (-1) * light,
              MIN_DIST_LIGHT, MAX_DIST_LIGHT);
22        if (!k) continue;
23        light.normalize();
24        intensity += k * lights[i].getIntensity() * std::max(0.,
              scalarProduct(normal, (-1) * light));
25        QVector3D reflect = (getReflectVector(light,
              normal)).normalized();
26        specIntensity += pow(std::max(0., scalarProduct(reflect, (-1)
              * ray.direction)),
              objects[closest]->getSubstance().getSpecularIndex()) *
              lights[i].getIntensity() * k;
27    }
28
29    QColor diffColor = objects[closest]->getSubstance().getColor();
30    color = getColor(intensity, specIntensity,
          objects[closest]->getSubstance().getCoefficients(), diffColor,
          reflectColor, refractColor, reflectK, trace_depth);
31    return color;
32 }
```

Листинг 3.5 – Поиск пересечения луча с триангулированным объектом

```
1  std::tuple<QVector3D, QVector3D, double>
       TriangulatedObject::getIntersectionInfo(QVector3D start_point,
       QVector3D direction, double min_dist, double max_dist)
2  {
3      double min_t = max_dist;
4
5      QVector3D closest_point(0, 0, 0);
6      QVector3D closest_normal(0, 0, 0);
7
8      auto [intersection, normal, t] =
           sphere.getIntersectionInfo(start_point, direction, min_dist,
           max_dist);
9
10     if (t == max_dist)
11         return { closest_point, closest_normal.normalized(), min_t };
12
13     for (int i = 0; i < faces.size(); i++)
14     {
15         Triangle cur_face = faces[i];
16         if (scalarProduct(cur_face.getNormal(), direction) >= 0)
               continue;
17         if (!cur_face.isIntersection(direction)) continue;
18         auto [point, t] = cur_face.findIntersection(start_point,
               direction);
19         if (!cur_face.isInside(point)) continue;
20
21         if (t < min_t && t > min_dist && t < max_dist)
22         {
23             min_t = t;
24             closest_point = point;
25             closest_normal = cur_face.getNormal();
26         }
27     }
28
29     return { closest_point, closest_normal.normalized(), min_t };
30 }
```

Листинг 3.6 – Поиск пересечения луча со сферой

```
std::tuple<QVector3D, QVector3D, double>
    Sphere::getIntersectionInfo(QVector3D start_point, QVector3D
    direction, double min_dist, double max_dist)
{
    QVector3D C = center;
    double r = radius;
    QVector3D OC = start_point - C;
    double k1 = scalarProduct(direction, direction);
    double k2 = 2 * scalarProduct(OC, direction);
    double k3 = scalarProduct(OC, OC) - r * r;
    double discriminant = k2 * k2 - 4 * k1 * k3;

    if (discriminant < 0)
        return { start_point, start_point, max_dist } ;

    double t1 = (-k2 + sqrt(discriminant)) / (2 * k1);
    double t2 = (-k2 - sqrt(discriminant)) / (2 * k1);
    double t;

    if ((t1 <= min_dist || t1 >= max_dist) && (t2 <= min_dist || t2 >=
        max_dist))
        return { start_point, start_point, max_dist };
    else if (t1 <= min_dist || t1 >= max_dist) t = t2;
    else if (t2 <= min_dist || t2 >= max_dist) t = t1;
    else t = std::min(t1, t2);

    QVector3D intersection = start_point + direction * t;
    QVector3D normal = (intersection - C).normalized();
    return { intersection, normal, t };
}
```

3.5 Интерфейс программы

На рисунке 3.1 представлен пример работы программы.

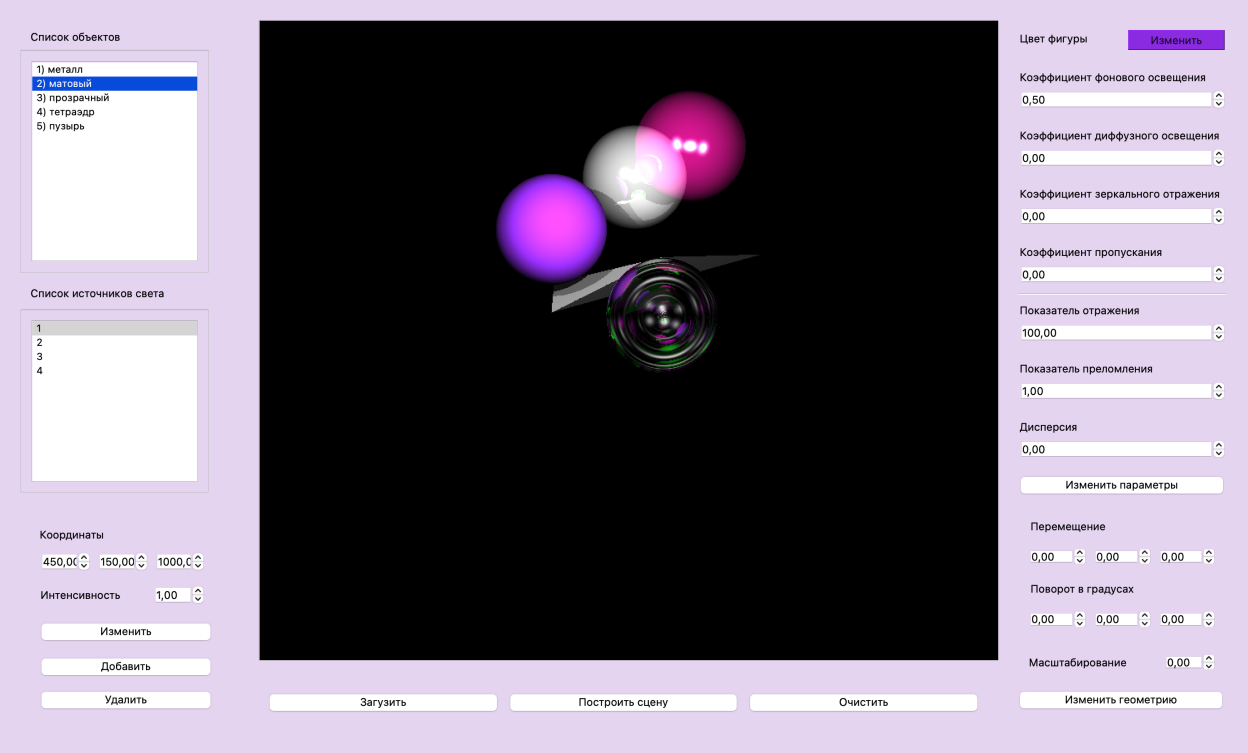

Рисунок 3.1 – Пример работы программы

С помощью реализованного интерфейса можно:

— загрузить объекты из файла, нажad на кнопку «Загрузить» (названия объектов высветятся в окне «Список объектов», а номера источников света в окне «Список источников света»);

— изменить положение источника и интенсивность, выделелив строку в окне «Список источников света», записав новые значения и нажав на кнопку «Изменить»;

— добавить новый источник, введя его координаты и интенсивность и нажав на кнопку «Добавить»;

— удалить выделенный источник, нажав на кнопку «Удалить».

— изменить оптические параметры объекта, выделив строку в окне «Список объектов», указав новые значения и нажав на кнопку «Изменить параметры»;

— изменить расположение данного источника или увеличить его, указав коэффициенты преобразования и нажав на кнопку «Изменить геометрию».

— очистить экран, нажав на кнопку «Очистить».

3.6 Модульное тестирование

Было проведено модульное тестирование с помощью фреймворка *GoogleTest* [22].

В листинге 3.7 пример тестирования функции нахождения пересечения луча со сферой, код которой был представлен на листинге 3.5.

Листинг 3.7 – Тест для функции нахождения пересечения луча со сферой

```
TEST(SphereIntersectionTest, TwoIntersections) {
    Sphere sphere(QVector3D(0, 0, 0), 2.0);
    QVector3D start_point(0, 3, 0);
    QVector3D direction(0, -1, 0);
    double min_dist = 0.0;
    double max_dist = 10.0;

    auto intersectionInfo = sphere.getIntersectionInfo(start_point,
       direction, min_dist, max_dist);

    EXPECT_EQ(std::get<0>(intersectionInfo), QVector3D(0, 2, 0));
    EXPECT_EQ(std::get<2>(intersectionInfo), 1.0);
}
```

С помощью макроса `TEST` создаются тесты, а с помощью макроса `EXPECT_EQ` проверяется совпадение результата с заранее вычесленным.

Были созданы наборы тестов для функций сцены, сферы и триангулированного объекта. Все тесты были пройдены успешно.

3.7 Функциональное тестирование

Этапы проведения функционального тестирования изображены на рисунке 3.2.

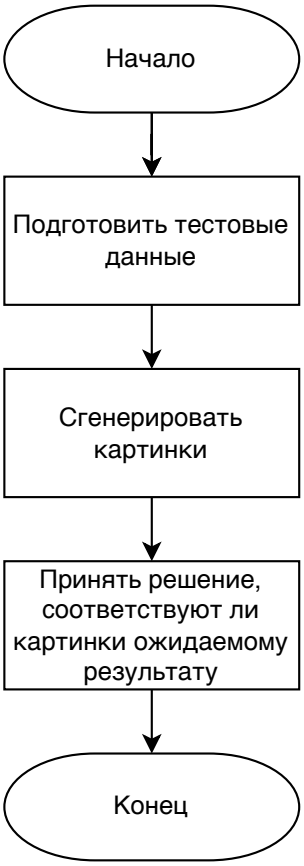

Рисунок 3.2 – Схема алгоритма проведения функционального тестирования

В качестве функционального тестирования били построены все типы объектов, которые есть в программе: сфера, мыльный пузырь, триангулированный объект и источник света.

На рисунке 3.3 продемонстрировано отражение фиолетовой сферы, перекрытой розовой сферой, в тетраэдре.

На рисунке 3.4 продемонстрировано рекурсивное отражение фиолетовой сферы и тетраэдра и прозрачная розовая сфера, за которой видно тетраэдр.

На рисунке 3.5 продемонстрирован мыльный пузырь и сферы, имеющие следующие оптические характеристики:

— розовая сфера – стеклянная (присутствуют отражение и преломление);

— серая сфера – металлическая (присутствует только отражение);

— фиолетовая сфера – матовая (отсутствуют отражение и преломление).

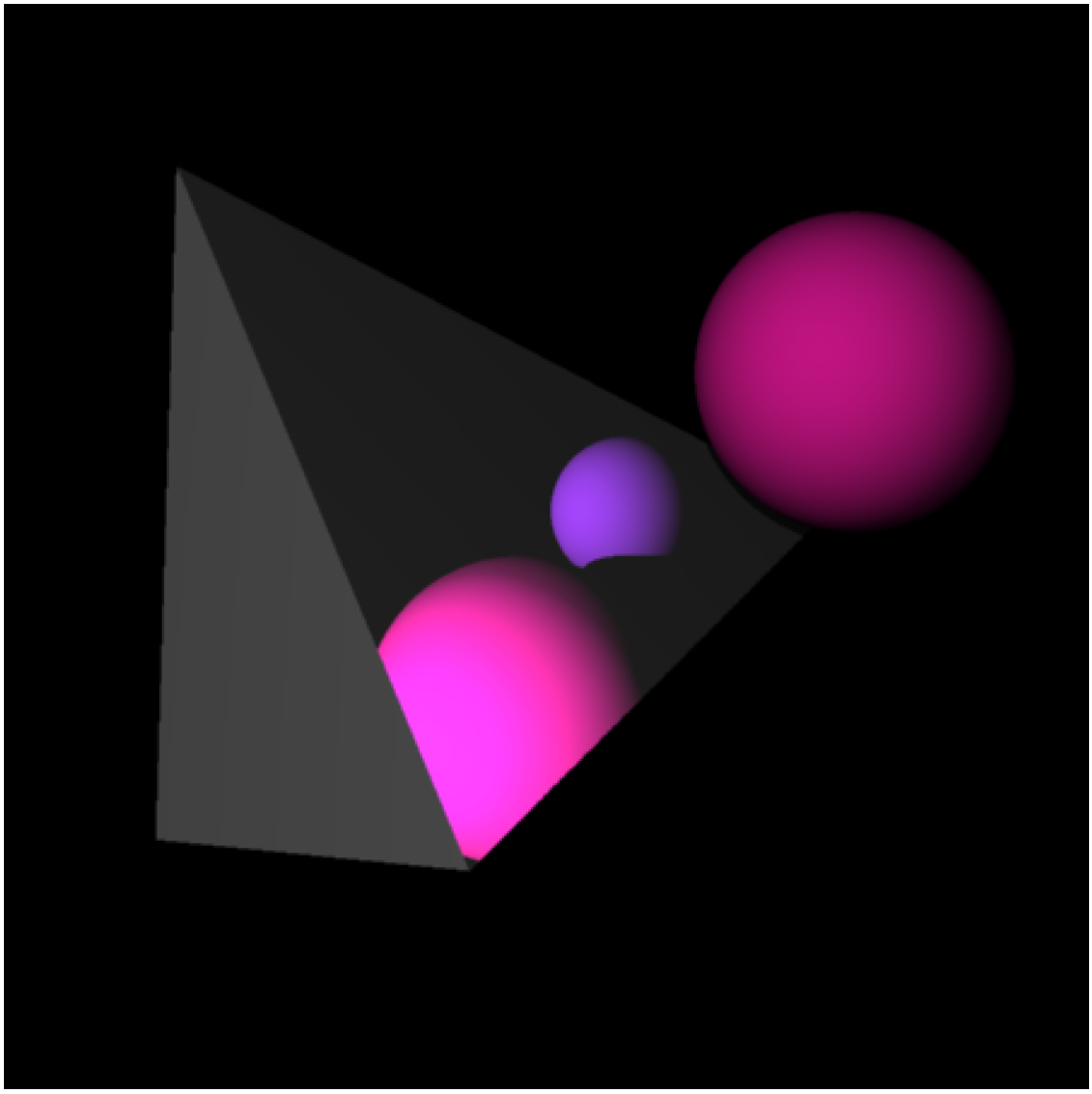

Рисунок 3.3 – Пример отражения перекрытого объекта

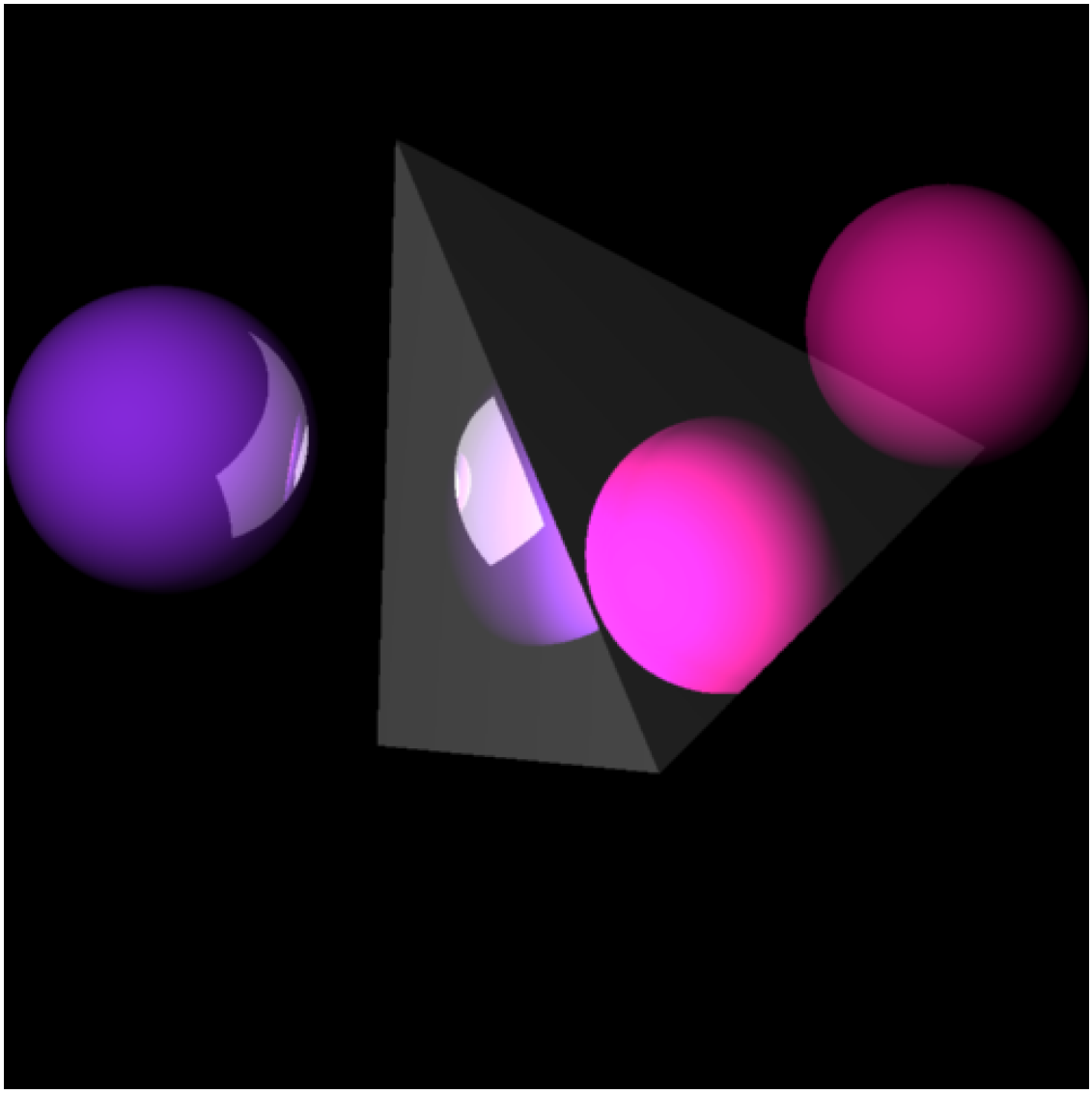

Рисунок 3.4 – Пример рекурсивного отражения и прозрачности

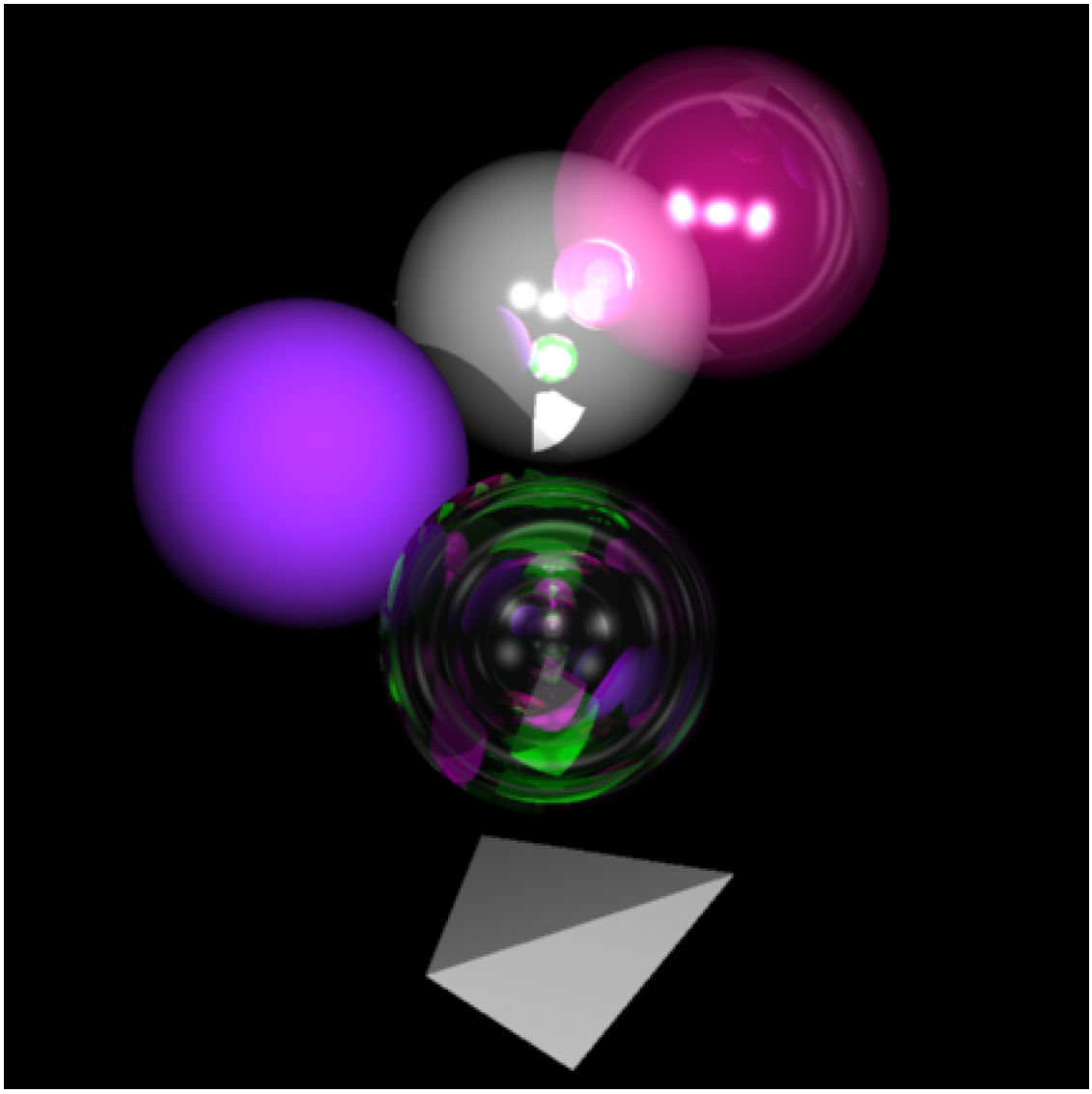

Рисунок 3.5 – Пример мыльного пузыря и сфер из различных материалов

3.8 Интерфейс управления из командной строки

Для автоматизации некоторых процессов в программе реализовано управление из командной строки:

— ключ «-image», после которого идет название текстового файла, в котором описаны объекты, а далее название изображения, получаемого при работе программы;

— ключ «-film», после которого идет название текстового файла, в котором описаны объекты, далее название фильма, получаемого при работе программы и в конце название текстового файла, в котором хранятся команды для модификации сцены;

— ключ «-research», после которого идет название текстового файла, в котором описаны объекты, далее название изображения, получаемого при работе программы и в конце значение глубины трассировки (целое неотрицательное число).

3.9 Выводы из технологической части

В технологической части были выбраны средства реализации и реализовано спроектированное программное обеспечение. Кроме того, были продемонстрированы примеры работы программы и проведено тестирование.

4 Исследовательская часть

В данной части описано проведённое исследование и его результаты, а так же технические характеристики устройства, на котором проводились замеры.

4.1 Технические характеристики устройства

Технические характеристики устройства, на котором выполнялись замеры [23]:

— операционная система MacOC Senoma 14.2;

— 18 ГБ оперативной паяти;

— процессор Apple M3 Pro (6 ядер производительности с частотой 4.06 ГГц и 6 ядер эффективности с частотой 2.8 ГГц).

4.2 Зависимость времени работы программы от глубины трассировки

Глубиной трассировки называется максимальная глубина рекурсии при вычислении интенсивности пикселей. Чем больше глубина трассировки, тем больше отражений и преломлений учитывается в итоговой интенсивности, однако как влияет глубина трассировки на время выполнения программы?

В таблице 4.1 представлена зависимость времени работы программы от глубины трассировки. Исследование проводилось на картинках, содержащих 1, 2 или 3 пузыря.

Таблица 4.1 – Зависимость времени работы программы от глубины трассировки

Глубина трассировки	Количество сфер		
	Одна	Две	Три
1	17.48	24.98	33.80
2	24.26	32.40	43.06
3	38.13	48.18	61.22
4	58.93	71.29	88.72
5	95.66	117.03	148.76
6	153.81	189.97	227.51
7	250.60	324.76	384.53
8	419.12	559.50	643.89
9	682.20	949.00	1113.02
10	1191.09	1631.23	1889.56

По таблице 4.1 был построен график, изображенный на рисунке 4.1. Также на график была нанесена аппроксимация полученных значений полиномами четвертой степени.

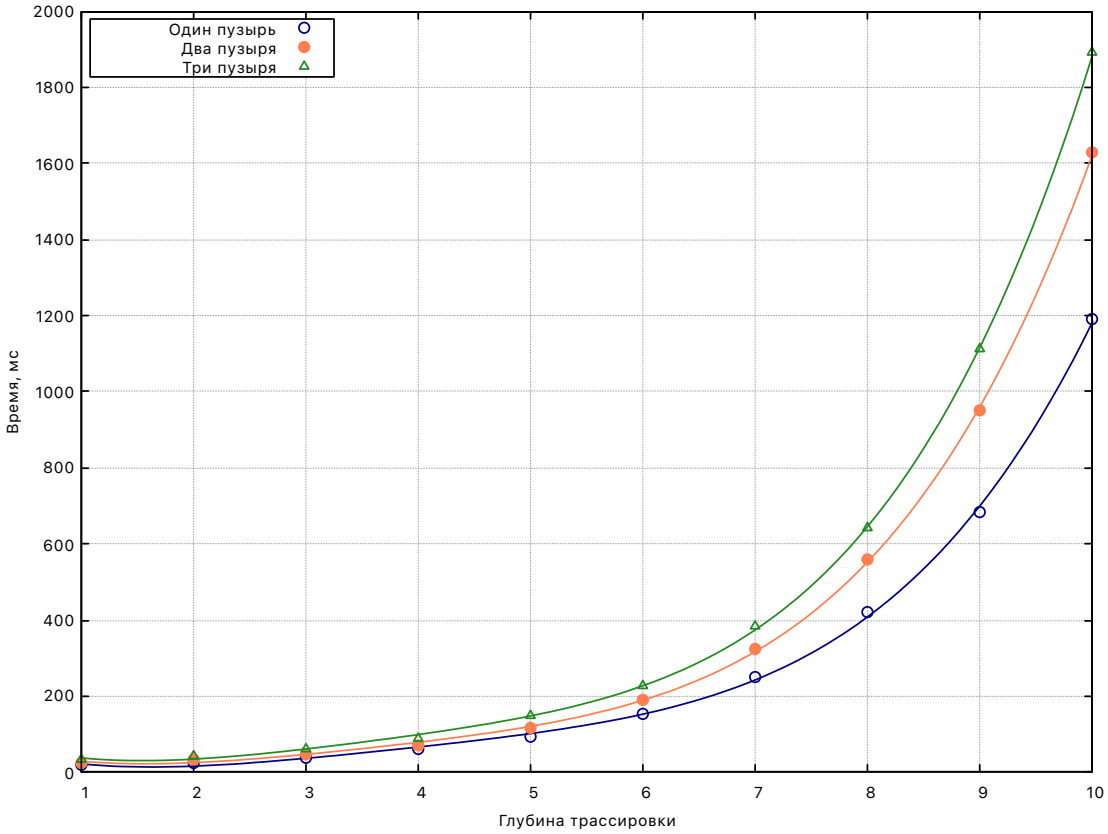

Рисунок 4.1 – Зависимость времени работы программы от глубины трассировки

По итогу исследования можно сделать вывод, что при увеличении глубины трассировки возрастает время выполнения программы. Кроме того, чем больше объектов, тем быстрее растёт время выполнения программы.

4.3 Выводы из исследовательской части

В исследовательской части были описаны характеристики устройства, на котором проводились замеры времени, а также проведённое исследование. Результаты исследования показывают, что с возрастанием глубины трассировки так же возрастает время выполнения программы. Кроме того, чем больше объектов на сцене, тем быстрее растёт время выполнения программы при увеличении глубины трассировки.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была достигнута поставленная цель: было разработано программное обеспечение для создания реалистичного изображения мыльных пузырей.

Также были решены все поставленные задачи:

1) описана физическая модель мыльных пузырей;
2) проанализированы и выбраны модели представления объектов;
3) проанализированы и выбраны алгоритмы решения основных задач компьютерной графики: удаления невидимых линий и поверхностей, учёта теней и освещения;
4) спроектировано программное обеспечение;
5) выбраны средства реализации и реализовано спроектированное программное обеспечение;
6) обеспечена возможность тестирования, созданы наборы тестов и продемонстрирована работоспособность программы;
7) исследована зависимость времени работы программы от глубины рекурсии.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Перемитина Т.* Компьютерная графика. — 2012.
2. Understanding Interference. Patterns in Soap Films. Eric Tompkins. Stony Brook Laser Teaching Center. [Электронный ресурс]. — — Режим доступа: `https://laser.physics.sunysb.edu/_ett/report/index.html` (дата обращения: 17.09.2023).
3. Фото мыльного пузыря [Электронный ресурс]. — — Режим доступа: `https://photo.99px.ru/photos/107854/`(дата обращения: 13.11.2023).
4. *Berne B. J.*, *Pecora R.* Dynamic light scattering: with applications to chemistry, biology, and physics. — Courier Corporation, 2000.
5. *Порев В.* Компьютерная графика // СПб.: БХВ-Петербург. — 2002. — Т. 432. — С. 3.
6. *Falcidieno B.*, *Kunii T. L.* Modeling in computer graphics: methods and applications. — Springer Science & Business Media, 2012.
7. *Salomon D.* Computer graphics and geometric modeling. — Springer Science & Business Media, 2012.
8. *Жаксылык А.*, *Ахметова Н.*, *Ибрагимов Р.* ПРИМЕНЕНИЕ ЕСТЕСТВЕННОГО И ИСКУССТВЕННОГО ОСВЕЩЕНИЯ И ТРАССИРОВКИ ЛУЧЕЙ ПРИ ТРЕХМЕРНОМ МОДЕЛИРОВАНИИ // Вестник Казахстанско-Британского технического университета. — 2021. — Т. 16, № 3. — С. 253—258.
9. *Будак В.*, *Галлямов И.*, *Киселёв Д.* Сравнительный анализ моделей цветового восприятия для реалистической визуализации в программах глобального освещения // Графикон-конференции по компьютерной графике и зрению. Т. 33. — 2023. — С. 116—124.
10. Система интерактивного расчета глобального освещения для гибридных сцен / Д. Боголепов [и др.] // Труды. — 2012.
11. *Демин А.* Основы компьютерной графики: учебное пособие // Томск: Изд-во Томского политехнического университета. — 2011.
12. *Dévai F.* The Future of Graphics Processors. —.

13. *Warnock J. E.* A hidden surface algorithm for computer generated halftone pictures. — The University of Utah, 1969.

14. *Foley J. D.* Computer graphics: principles and practice. T. 12110. — Addison-Wesley Professional, 1996.

15. *Кутенев И.* Разработка и реализация алгоритма трассировки лучей в реальном времени на основе динамического воксельного октодерева // Информационные технологии. — 2019. — С. 10—10.

16. *Половинина А. А.*, *Егорова Ю. Г.* ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ АЛГОРИТМОВ УДАЛЕНИЯ НЕВИДИМЫХ ПОВЕРХНОСТЕЙ ДЛЯ ПОСТРОЕНИЯ РЕАЛИСТИЧНЫХ ИЗОБРАЖЕНИЙ // НАУЧНО-ТЕХНИЧЕСКОЕ ТВОРЧЕСТВО АСПИРАНТОВ И СТУДЕНТОВ. — 2018. — С. 227—229.

17. *Головнин А.* Базовые алгоритмы компьютерной графики // Проблемы качества графической подготовки студентов в техническом вузе: традиции и инновации. — 2016. — Т. 1. — С. 13—30.

18. *Crow F. C.* Shadow algorithms for computer graphics // Acm siggraph computer graphics. — 1977. — Т. 11, № 2. — С. 242—248.

19. Алгоритмы построения теней / А. Романюк [и др.]. — 2000.

20. Документация по языку C++. [Электронный ресурс]. — — Режим доступа: `https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170` (дата обращения: 17.09.2023).

21. Документация по фреймворку Qt [Электронный ресурс]. — — Режим доступа: `https://doc.qt.io`(дата обращения: 13.08.2023).

22. Документация по библиотеке Google Test [Электронный ресурс]. — — Режим доступа: `https://google.github.io/googletest/`(дата обращения: 13.08.2023).

23. macOS Sonoma [Электронный ресурс]. — — Режим доступа: `https://www.apple.com/macos/sonoma/`(дата обращения: 23.11.2023).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе.