Содержание

Введение

Компьютерная графика является неотъемлемой частью современного мира в особенности она востребована в сфере кинопроизводства и игровой промышленности. Именно поэтому синтез реалистического изображения является одной из важнейших задач.

Подобные условия требуют создания средств и методик, учитывающих такие оптические явления как преломление, отражение и рассеивание света, зависящие от свойств визуализируемых объектов. Но рост точности и детальности разрабатываемых алгоритмов зачастую приводит к более высоким затратам по времени и памяти.

Целью данной работы является разработка программы, позволяющей создать реалистическое изображение с учетом подповерхностного рассеивания, на основе трехмерной сцены наполненной объектами (сфера, куб, конус, цилиндр, полигональная модель) и источниками света, положение, количество и свойства которых задается пользователем.

Задачи:

1) выявить неотъемлемые качества реалистического изображения;
2) провести анализ существующих алгоритмов;
3) выбрать оптимальные пути для решения основной задачи;
4) реализовать выбранные алгоритмы;
5) создать программный продукт для решения задачи;
6) исследовать время работы полученной программы в случае распараллеливания алгоритма.

1 Аналитическая часть

В этом разделе будут рассмотрены модели представления трехмерных тел, формализованы объекты, наполняющие сцену, и требования, предъявляемые к работе программы и конечному результату. Будут рассмотрены алгоритмы построения трехмерного изображения, методы закраски, модели освещения.

1.1 Анализ необходимых качеств реалистического изображения

Большинство алгоритмов, изученных в курсе компьютерной графики нацелены на корректное отображение геометрии объектов и их положения в сцене, однако для построения реалистического изображения этих аспектов недостаточно.

Первым аспектом, наполняющим сцену объемом можно считать тень. Однако большинство моделей подразумевает источник света материальной точкой, из-за чего все тени получаются четкими и однородными, что в общем случае неверно.

Подобным образом, для построения изображения стоит учитывать такую важную составляющую оптических систем, как линза. Изображения предметов на фотографии или у нас перед глазами получаются четкими только в случае, если они находятся в фокальной плоскости, иначе из представление размывает, что также необходимо учитывать.

Не стоит забывать, что объекты в сцене состоят из различных материалов, что вносит на рассмотрение блестящие и прозрачные предметы, с присущими явлениями отражения, преломления и рассеивания.

1.2 Формализация объектов синтезируемой сцены

Сцена состоит из следующего набора объектов.

1) Геометрический объект – непосредственное представление объектов реального мира внутри сцены. В число рассматриваемых тел входят параметрические модели, такие как куб, сфера, конус, цилиндр, а также тела произвольной формы. Данная модель предназначена исключительно для хранения геометрического представления и иерархии составного тела.

   Для описания каждого объекта используются:

   — куб – координаты двух диагональных вершин;

   — сфера – координаты центра и радиус;

— конус – координаты центра основания, радиус основания, высота;

— цилиндр – координаты центра нижнего основания, радиус, высота;

— тело произвольной формы – набор полигонов с указанием нормали к поверхности, определяющей наружную часть объекта. В качестве основной единицы такой модели был выбран треугольник, так как с его помощью можно представить полигон любой формы.

2) Источник света – геометрическое тело, обладающее свойством свечения. Для источника света присущи такие вырожденные случаи геометрического тела, как точка (точечный источник света, направленный свет).

3) Карта окружения – объект (как правило куб или сфера), опоясывающий все объекты сцены и бесконечно удаленный от них. Служит для определения фонового изображения и создания дополнительных отражений.

4) Камера – геометрическое тело, определяющее видимую часть сцены. Положение задается координатой центра, направление взгляда совпадает с осью Z тела.

1.3 Анализ алгоритмов удаления невидимых линий и ребер поверхностей

Прежде всего предстоит выбрать основной алгоритм визуализации изображаемой сцены. При выборе необходимо обращать внимание на следующие аспекты: быстродействие, требовательность к памяти, возможность реализации эффектов, описанных в пункте 1.1.

1.3.1 Алгоритм, использующий Z-буфер

Смысл алгоритма заключается в использовании двух буферов: буфера кадра и Z-буфера, хранящих атрибуты и информацию о координате Z каждого пикселя соответственно [1].

Z-буфер инициализируется минимальным значением координаты, а буфер кадра – информацией о пикселе, описывающем фон. Глубина каждого нового пикселя при подсчете сравнивается со значением, которое уже есть в Z-буфере. В случае, если новый пиксель расположен ближе к наблюдателю, то информация о нем заносится в буфер кадра и происходит редактирование Z-буфера.

Данный алгоритм достаточно прост в реализации и, в силу отсутствия сортировок, не требует больших затрат по времени и памяти (для современных компьютеров), однако требуются дополнительные техники для учета отражений, прозрачности и теней.

1.3.2 Алгоритм обратной трассировки лучей

Алгоритм берет свое начало из законов оптики: наблюдатель видит объект, только если до него доходят лучи испускаемого света, прошедшие некоторый путь от источника [2].

Однако, вопреки физической модели, трассировка лучей происходит в обратном направлении – от наблюдателя к источнику. В противном случае приходится рассматривать множество лучей, которые никогда не попадут в наблюдаемую область.

Предполагая, что наблюдатель располагается непосредственно за экраном, в процессе работы алгоритма для каждого пикселя выпускаетя луч. Если луч попадает в объект сцены, то из найденной точки пересечения в направлении источников света посылаются так называемые «теневые» лучи, благодаря которым определяется итоговая интенсивность точки, а в следствии и пикселя. Иначе пиксель высвечивается цветом фона.

Алгоритм построен на физически верной модели, что дает широкий простор для реализации различных визуальных явлений, но такая гибкость и обобщенность требует большого количества вычислений (возможность усовершенствования засчет параллельных вычислений, так как каждый пиксель в данном методе не зависим от окружающих [3]).

1.3.3 Алгоритм обратной трассировки путей

Описанный выше алгоритм учитывает только прямое освещение, что ограничивает возможности воссоздания оптических эффектов. Как следует из названия, алгоритм производит построение пути, пройденного светом от источника до наблюдателя, путем рекурсивной работы алгоритма трассировки лучей, для очередной найденной точки пересечения [2].

Несмотря на кажущуюся простоту алгоритма, он является достаточно затратным в плане вычислений, так как для получения корректного результата в конкретной точке необходим расчет нескольких путей в различных направлениях. Помимо этого, возникают проблемы, связанные с множественным отражением и глубиной рекурсии.

1.3.4 Алгоритм Робертса

Данный алгоритм решает задачу удаления невидимых ребер и граней только с выпуклыми телами, работает только в объектном пространстве [4].

У следующие этапы выполнения:

— удаление ребер, экранируемых самим телом;

— удаление невидимых ребер, экранируемых другими телами сцены;

— удаление невидимых ребер, образованных в результате протыкания.

Алгоритм обладает высокой точностью, в силу того, что все вычисления происходят в объектном пространстве, однако тела должны быть выпуклыми, что не подходит для нашей задачи.

1.3.5 Алгоритм художника

Данный алгоритм работает аналогично тому, как художник рисует картину - то есть сначала рисуются дальние объекты, а затем более близкие. Наиболее распространенная реализация алгоритма - сортировка по глубине, которая заключается в том, что произвольное множество граней сортируется по ближнему расстоянию от наблюдателя, а затем отсортированные грани выводятся на экран в порядке от самой дальней до самой ближней.

Данный алгоритм требует меньше памяти в сравнении с остальными, однако так же, как и алгоритм Z-буфера требует дополнительных методов отображения визуальных эффектов.

Вывод

Так как цель работы – создание реалистического изображения, то лучшим выбором станет алгоритм трассировки путей. Данный алгоритм предоставляет удобную базу для реализации оптических эффектов, а затраты по времени не окажут особого значения, так как объекты в сцене – статические, поэтому можно сконцентрироваться на качестве получаемого изображения.

1.4 Реализация теней

Выбранный в предыдущем пункте алгоритм трассировки лучей, позволяет легко реализовать отрисовку тени.

Корректное построение достигается засчет того, что после нахождения ближайшей точки пересечения очередного луча, из нее посылаются «теневые» лучи к каждому источнику света в сцене. Если теневой луч не достигает источника света (имеется более близкое пересечение с каким-либо объектом сцены), то точка не освещена, иначе – в точке рассчитывается интенсивность, согласно модели освещения.

Помимо этого, также решается проблема построения мягких теней, так как имеется возможность испускания множества теневых лучей к одному объекту, что позволит определить

видимую часть и итоговую интенсивность.

1.5 Построение отражений и преломления

Нахождение отражений и преломления также является логическим продолжением алгоритма трассировки лучей. Используя нормаль к поверхности и направление отраженного луча, нетрудно найти падающий и преломленный лучи, для которого аналогично применяется трассировка. Пример расположения лучей на границе объекта представлен на рисунке 1.1.

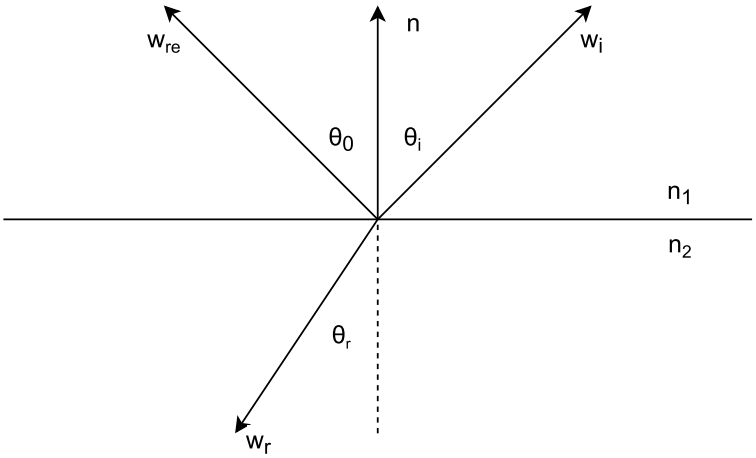

Рисунок 1.1 – Пример расположения лучей на границе объекта: w_i - направление падающего луча, w_{re} - направление отраженного луча, w_r - направление преломленного луча

Интенсивность света, распространяющегося в направлении взгляда w_0 из точки пространства p, может быть описана при помощи следующей формулы

$$L_0(p, w_0) = \int_{S^2} \mathrm{f}(p, w_0, w_i) \mathrm{L_i}(p, w_i) \cos(\theta_i) \mathrm{d}w_i, \tag{1.1}$$

где w_i - направление падающего света в точке p; f - функция рассеяния; L_i - интенсивность света, падающего по направлению w_i; θ_i - угол между нормалью к поверхности и вектором направления падающего света [5]. Значение интеграла вычисляется по поверхности единичной сферы. Такой метод описания дает возможность реализовать любую модель освещения и легко произвести подмену при необходимости, засчет введения функции рассеяния, показывающую какая часть падающего света отражается в указанном направлении.

$$\mathrm{f}(p, w_0, w_i) \in [0, 1] \tag{1.2}$$

1.5.1 Модель освещения Ламберта

Данная модель предназначена для представления идеального диффузного рассеивания, то есть подразумевается, что свет при попадании на поверхность равномерно распространяется во всех направлениях

$$\mathrm{f_d}(p, w_0, w_i) = k_d, \tag{1.3}$$

где k_d - коэффициент диффузного отражения.

1.5.2 Модель освещения Фонга

Данная модель базируется на модели Ламберта и добавляет зеркальную составляющую, то есть блик на поверхности объекта. Для достижения этого эффекта к формуле (1.3) добавляется следующая функция зеркального рассеивания

$$\mathrm{f_s}(p, w_0, w_i) = k_s \cos^{\alpha}(\theta_{re}) = k_s \left(\frac{w_0 w_{re}}{|w_0||w_{re}|} \right)^{\alpha}, \tag{1.4}$$

где θ_{re} – угол между отраженным лучом и лучом взгляда, k_s - коэффициент зеркального отражения, α - коэффициент блеска. Пример расположения лучей представлен на рисунке 1.2.

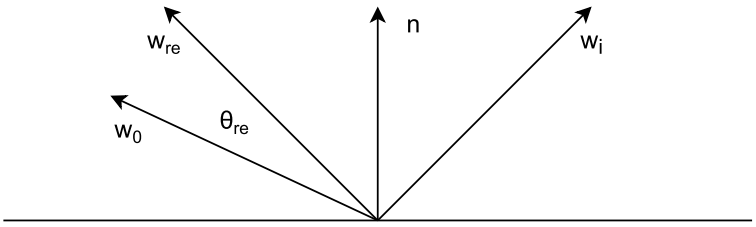

Рисунок 1.2 – Расположение лучей при расчете модели Фонга: w_0 - направление взгляда, w_{re} - направление отраженного луча, w_i - направление падающего луча

1.5.3 Модель освещения Френеля

В отличие от представленных эмпирических моделей, данная – физически корректна и опирается на решение уравнений Максвелла [5]

$$\mathrm{F}(p, w_0, w_i) = \frac{1}{2}(r_{\perp}^2 + r_{\parallel}^2), \tag{1.5}$$

где $r_{\parallel}$ - коэффициент отражения для света с параллельной поляризацией, $r_{\perp}$ - коэффициент отражения для света с перпендикулярной поляризацией. Реализация метода разделяет материалы на два типа: проводники и диэлектрики (согласно их физическим свойствам).

1) Отражение для диэлектрика

$$r_{\parallel} = \frac{n_2\cos(\theta_i) - n_1\cos(\theta_r)}{n_2\cos(\theta_i) + n_1\cos(\theta_r)} \tag{1.6}$$

$$r_{\perp} = \frac{n_1\cos(\theta_i) - n_2\cos(\theta_r)}{n_1\cos(\theta_i) + n_2\cos(\theta_r)}, \tag{1.7}$$

где θ_r - угол между нормалью и преломленным лучом, который может быть вычислен по закону Снеллиуса.

2) Отражение для проводника

В отличие от диэлектриков коэффициент преломления для металлов - комплексное число $\overline{n} = n + ik$, где n – показатель преломления, k - коэффициент экстинкции.

$$r_{\perp} = \frac{a^2 + b^2 - 2a\cos(\theta_i) + \cos^2(\theta_i)}{a^2 + b^2 + 2a\cos(\theta_i) + \cos^2(\theta_i)} \tag{1.8}$$

$$r_{\parallel} = r_{\perp}\frac{\cos^2(\theta_i)(a^2+b^2) - 2a\cos(\theta_i)\sin^2(\theta_i) + \sin^4(\theta_i)}{\cos^2(\theta_i)(a^2+b^2) + 2a\cos(\theta_i)\sin^2(\theta_i) + \sin^4(\theta_i)}, \tag{1.9}$$

где $a^2 = \frac{\sqrt{(n^2-k^2-\sin^2(\theta_i))^2+4n^2k^2}+(n^2-k^2-\sin^2(\theta_i))}{2}$, $b^2 = \frac{\sqrt{(n^2-k^2-\sin^2(\theta_i))^2+4n^2k^2}-(n^2-k^2-\sin^2(\theta_i))}{2}$, тогда $a^2 + b^2 = \sqrt{(n^2-k^2-\sin^2(\theta_i))^2+4n^2k^2}$, при этом $\overline{n} = \frac{\overline{n_2}}{\overline{n_1}} = \frac{n_2+ik_2}{n_1+ik_1} = \frac{n_1n_2+k_1k_2}{n_1^2+k_1^2} + i\frac{k_2n_1-n_2k_1}{n_1^2+k_1^2}$, то есть $n = \frac{n_1n_2+k_1k_2}{n_1^2+k_1^2}$, а $k = \frac{k_2n_1-n_2k_1}{n_1^2+k_1^2}$.

Однако в таком виде не получится добиться зеркального блика, для этого введем функцию $\delta_{w_0} = \begin{cases} 1, & \text{если} w = w_0 \\ 0, & \text{иначе.} \end{cases}$ Тогда

$$\mathrm{L_0}(p, w_0) = \int_{S^2} \delta_{w_0}(w_i)\mathrm{F}(p, w_0, w_i)\mathrm{L_i}(p, w_i)\cos(\theta_i)\mathrm{d}w_i =$$
$$= \mathrm{F}(p, w_0, w_{re})\mathrm{L_{re}}(p, w_{re})\cos(\theta_{re}),$$

где w_{re} - зеркально отраженный луч для w_0. Но желаемый результат

$$\mathrm{L_0} = \mathrm{F}(p, w_0, w_{re})\mathrm{L_{re}}(p, w_{re}),$$

поэтому необходимо нормировать функцию $\delta_{w_{re}}$

$$\mathrm{f_s}(p, w_0, w_i) = \frac{\delta_{w_{re}}}{\cos(\theta_{re})}\mathrm{F}(p, w_0, w_i) \tag{1.10}$$

1.6 Реализация глубины резкости

Желаемый результат по достижению глубины резкости может быть достигнут следующими способами.

1.6.1 Трассировка лучей по поверхности линзы

Трассировка лучей также позволяет достаточно очевидно реализовать глубину резкости, засчет симуляции множества лучей проходящих через оптическую систему (линзу) [6].

К примеру можно реализовать тонкую линзу, пропуская луч из точки картинной плоскости в произвольные точки на поверхности диска, представляющего собой линзу. Если объект находятся в фокусе, то все лучи попадут в одну точку, в результате чего изображение будет четким, иначе – множество различных значений излучения создадут размытое изображение. Пример прохождения лучей через линзу представлен на рисунке 1.3.

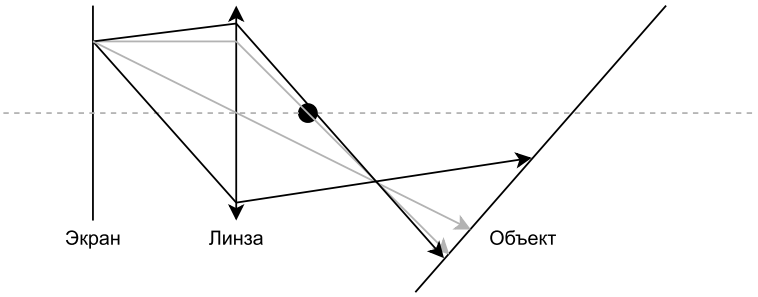

Рисунок 1.3 – Лучи, попадающие в одну точку, после прохождения через тонкую линзу

1.6.2 Использование накопительного буфера

Метод подразумевает, что для получения необходимого эффекта, нужно будет получить множество буферов кадра, для каждого из которых положение камеры будет изменяться согласно параметрам камеры. Результирующее изображение получается суммированием и усреднением полученных значений в накопительном буфере на каждой новой итерации алгоритма [6].

1.6.3 Слоистая глубина резкости

Алгоритм реализует поставленную задачу, путем разделения сцены на «слои», согласно глубине изображения в точке сцены. Далее к полученным слоям применяется алгоритм фильтрации, для достижения необходимого размытия текущего слоя. Это метод менее затратен в плане вычислений, так как для сцены требуется вычислить только один буфер кадра, разбитый на несколько слоев [6].

Вывод

В программе будет использоваться метод распространения лучей по поверхности линзы, так как он позволит реализовать практически любую оптическую систему и произвести подмену в случае необходимости.

1.7 Реализация подповерхностного рассеивания

Подповерхностное рассеивание описывает механизм распространения света, при котором свет, проникая внутрь полупрозрачного тела через его поверхность, рассеивается внутри самого тела, многократно отражаясь от частиц тела в случайном направлении и на нерегулярные углы. В итоге свет выходит из объекта в выходной точке, отличной от точки вхождения в объект.

Нетрудно заметить, что необходимый результат можно достичь, произведя дополнительное интегрирование соотношения (1.1) по поверхности тела A, что потребует введения дополнительной *функции поверхностного рассеивания* $\mathrm{S}(p_0, w_0, p_i, w_i)$ [7]

$$\mathrm{L}_0(p_0, w_0) = \int_A \int_{S^2} \mathrm{S}(p_0, w_0, p_i, w_i)\mathrm{L}_\mathrm{i}(p_i, w_i)\cos(\theta_i)\mathrm{d}w_i\mathrm{d}A. \tag{1.11}$$

Однако задание функции рассеивания даже для плоского случая достаточно проблематично, даже без учета составных тел. Поэтому на практике применяется другое соотношение для S , называемое разделяемой *функцией поверхностного рассеивания* [5]

$$\mathrm{S}(p_0, w_0, p_i, w_i) = (1 - \mathrm{F}_\mathrm{r}(\cos\theta_0))\mathrm{S}_\mathrm{p}(p_0, p_i)\mathrm{S}_\mathrm{w}(w_i), \tag{1.12}$$

где $1 - \mathrm{F}_\mathrm{r}$ - формула отражения Френеля (1.5), показывающая часть света, доходящую до наблюдателя после выхода за границы объекта; $\mathrm{S}_\mathrm{p}(p_0, p_i)$ - функция, описывающая часть света,

способную пройти расстояние от точки входа до точки выхода внутри тела; $\mathrm{S_w}(w_i) = \frac{1-\mathrm{F_r}(\cos\theta_i}{c\pi}$ - аналог первой части выражения для света, попадающего в границы объекта. Нормализующий коэффициент c служит для выполнения условия $\int\limits_{H^2} \mathrm{S_w}(w_i)\cos\theta \mathrm{d}w = 1$ (интегрирование по половине сферы, отсекаемой поверхностью).

Для демонстрации упрощения подставим (1.12) в (1.11)

$$\begin{aligned}
\mathrm{L_0}(p_0, w_0) &= \int\limits_A \int\limits_{H^2} \mathrm{S}(p_0, w_0, p_i, w_i)\mathrm{L_i}(p_i, w_i)\cos(\theta_i)\mathrm{d}w_i\mathrm{d}A = \\
&= \int\limits_A \int\limits_{H^2} (1 - \mathrm{F_r}(\cos\theta_0))\mathrm{S_p}(p_0, p_i)\mathrm{S_w}(w_i)\mathrm{L_i}(p_i, w_i)\cos(\theta_i)\mathrm{d}w_i\mathrm{d}A = \\
&= (1 - \mathrm{F_r}(\cos\theta_0)) \int\limits_A \mathrm{S_p}(p_0, p_i) \int\limits_{H^2} \mathrm{S_w}(w_i)\mathrm{L_i}(p_i, w_i)\cos(\theta_i)\mathrm{d}w_i\mathrm{d}A \qquad (1.13)
\end{aligned}$$

Несмотря на то, что мы разделили интеграл на части, внутренняя часть которого вполне укладывается в имеющееся соотношение для отражения (1.1), остается проблема с поддержкой геометрии произвольных тел, поэтому введем следующее приближение и получим окончательную формулу

$$\mathrm{S_p}(p_0, p_i) \approx \mathrm{S_r}(|\overline{p_0 p_i}|) \qquad (1.14)$$

$$\begin{aligned}
\mathrm{L_0}(p_0, w_0) &= (1 - \mathrm{F_r}(\cos\theta_0)) \int\limits_A \mathrm{S_p}(p_0, p_i) \int\limits_{H^2} \mathrm{S_w}(w_i)\mathrm{L_i}(p_i, w_i)\cos(\theta_i)\mathrm{d}w_i\mathrm{d}A \approx \\
&\approx (1 - \mathrm{F_r}(\cos\theta_0)) \int\limits_A \mathrm{S_p}(|\overline{p_0 p_i}|) \int\limits_{H^2} \mathrm{S_w}(w_i)\mathrm{L_i}(p_i, w_i)\cos(\theta_i)\mathrm{d}w_i\mathrm{d}A \qquad (1.15)
\end{aligned}$$

Вывод

В данном разделе были описаны модели представления моделей в сцене и свойства, необходимые для физически корректной визуализации объектов с учетом подповерхностного рассеивания. В качестве основного алгоритма удаления невидимых линий был выбран алгоритм трассировки путей, на основании которого будет строиться реализация отражений, теней, глубины резкости и подповерхностного рассеивания.

2 Конструкторская часть

В данном разделе будут рассмотрены требования к программному обеспечению, некоторые расчетные соотношения и схемы разработанных алгоритмов.

2.1 Требования к программному обеспечению

Программа должна предоставлять следующий функционал:

— возможность интерактивного изменения положения объектов и источников света в сцене;

— возможность интерактивного добавления/удаления объектов в сцене;

— возможность интерактивного изменения свойств объектов (коэффициент восприятия света, коэффициент преломления, тип материала объекта, интенсивность и цвет излучения);

— возможность интерактивного масштабирования и поворота объектов сцены.

К программе предоставляются следующие требования:

— программа должна корректно реагировать на любые действия пользователя;

— программа должна предоставлять приемлемое время отклика (не более 1 сек) при работе в интерактивном режиме.

2.2 Расчет лучей на поверхности тела

Выражение для нахождения отраженного w_{re} и преломленного w_r лучей, могут быть найдены при помощи заданных лучей падения w_i и нормали n (рисунок 1.1).

Для упрощения вычислений работа будет производиться с единичными векторами $\widehat{w_{re}}$, $\widehat{w_r}$, $\widehat{w_i}$ и $\widehat{n}$.

Отраженный луч

Отраженный луч может быть представлен в виде комбинации векторов падения и нормали

$$\widehat{w_{re}} = \alpha\widehat{w_i} + \beta\widehat{n}.$$

Из равенства углов падения и отражения:

$$\widehat{w_{re}}\widehat{n} = \widehat{w_i}\widehat{n} = \alpha\widehat{w_i}\widehat{n} + \beta\widehat{n}^2 = \alpha\widehat{w_i}\widehat{n} + \beta$$

$$\beta = \widehat{w_i}\widehat{n} - \alpha\widehat{w_i}\widehat{n}.$$

Так как вектора единичные

$$|\widehat{w_{re}}| = |\widehat{w_i}| = 1 \quad \uparrow^2$$
$$|\widehat{w_{re}}|^2 = |\widehat{w_i}|^2 = 1,$$

$$\begin{aligned}
1 = {w_{re}}^2 &= (\alpha\widehat{w_i} + \beta\widehat{n})^2 = \alpha^2{\widehat{w_i}}^2 + 2\alpha\beta\widehat{w_i}\widehat{n} + \beta^2\widehat{n}^2 = \alpha^2 + 2\alpha\beta\widehat{w_i}\widehat{n} + \beta^2 = \\
&= \alpha^2 + 2\alpha(\widehat{w_i}\widehat{n} - \alpha\widehat{w_i}\widehat{n})\widehat{w_i}\widehat{n} + (\widehat{w_i}\widehat{n} - \alpha\widehat{w_i}\widehat{n})^2 = \\
&= \alpha^2 + \underline{2\alpha(\widehat{w_i}\widehat{n})^2} - 2\alpha^2(\widehat{w_i}\widehat{n})^2 + (\widehat{w_i}\widehat{n})^2 - \underline{2\alpha(\widehat{w_i}\widehat{n})^2} + \alpha^2(\widehat{w_i}\widehat{n})^2 = \\
&= (\widehat{w_i}\widehat{n})^2 + (1 - (\widehat{w_i}\widehat{n})^2)\alpha^2,
\end{aligned}$$

$$1 - (\widehat{w_i}\widehat{n})^2 = (1 - (\widehat{w_i}\widehat{n})^2)\alpha^2.$$

Если $1 - (\widehat{w_i}\widehat{n})^2 = 0 \quad \Rightarrow \quad \widehat{w_i}\widehat{n} = \pm 1 \quad \Rightarrow \quad \widehat{w_i} || \widehat{n} \quad \Rightarrow \quad w_{re} = w_i,$
иначе $\alpha^2 = 1 \quad \Rightarrow \quad \alpha = \pm 1$, так как падающий и отраженный луч не могут находиться по одну сторону от нормали, то $\alpha = -1$, $\beta = 2\widehat{w_i}\widehat{n}$, а $\widehat{w_{re}} = 2(\widehat{w_i}\widehat{n})\widehat{n} - \widehat{w_i}$.

Таким образом

$$w_{re} = \widehat{w_{re}}|w_i| = (2(\widehat{w_i}\widehat{n})\widehat{n} - \widehat{w_i})|w_i| = (2\frac{w_i n}{|w_i||n|}\frac{n}{|n|} - \frac{w_i}{|w_i|})|w_i| = 2\frac{w_i n}{n^2}n - w_i. \qquad (2.1)$$

Преломленный луч

Аналогично для преломленного луча

$$\widehat{w_r} = \alpha\widehat{w_i} + \beta\widehat{n}.$$

По закону Снеллиуса

$$\begin{aligned} n_1\sin(\theta_i) &= n_2\sin(\theta_r) \quad \uparrow^2 \\ n_1^2\sin^2(\theta_i) &= n_2^2\sin^2(\theta_r) \\ n_1^2(1-\cos^2(\theta_i)) &= n_2^2(1-\cos^2(\theta_r)). \end{aligned} \tag{2.2}$$

Так как $\cos\theta_i = \widehat{w_i}\widehat{n}$ и $\cos\theta_r = \widehat{w_r}\widehat{n}$, то выражение (2.2) преобразуется в

$$\begin{aligned} n_1^2(1-(\widehat{w_i}\widehat{n})^2) = n_2^2(1-(\widehat{w_r}\widehat{n})^2) = n_2^2(1-(\alpha\widehat{w_i}\widehat{n}+\beta\widehat{n}^2)^2)) = \\ = n_2^2(1-(\alpha^2(\widehat{w_i}\widehat{n})^2 + 2\alpha\beta(\widehat{w_i}\widehat{n}) + \beta^2)) \end{aligned} \tag{2.3}$$

Так как векторы $\widehat{w_i}$ и $\widehat{w_r}$ единичные

$$1 = \widehat{w_r}^2 = (\alpha\widehat{w_i} + \beta\widehat{n})^2 = \alpha^2 + 2\alpha\beta(\widehat{w_i}\widehat{n}) + \beta^2, \tag{2.4}$$

$$2\alpha\beta(\widehat{w_i}\widehat{n}) = 1 - \alpha^2 - \beta^2. \tag{2.5}$$

Подставим (2.5) в (2.3)

$$\begin{aligned} n_1^2(1-(\widehat{w_i}\widehat{n})^2) &= n_2^2(1-(\alpha^2(\widehat{w_i}\widehat{n})^2 + 1 - \alpha^2 - \beta^2 + \beta^2)) \\ n_1^2(1-(\widehat{w_i}\widehat{n})^2) &= n_2^2(1-(\widehat{w_i}\widehat{n})^2)\alpha^2. \end{aligned}$$

Аналогично, если $1-(\widehat{w_i}\widehat{n})^2 = 0$, то $w_{re} = -w_i$, иначе $\alpha = \pm\frac{n_1}{n_2}$. Так как падающий и преломленный лучи не могут находиться по одну сторону от нормали, то $\alpha = -\frac{n_1}{n_2}$. Подставим полученное значение в (2.4)

$$\beta^2 - 2\frac{n_1}{n_2}(\widehat{w_i}\widehat{n})\beta + \frac{n_1^2 - n_2^2}{n_2^2} = 0$$

$$\mathrm{D} = 4\frac{n_1^2}{n_2^2}(\widehat{w_i}\widehat{n})^2 - 4\frac{n_1^2 - n_2^2}{n_2^2} = \frac{4}{n_2^2}(n_2^2 - n_1^2(1-(\widehat{w_i}\widehat{n})^2)).$$

Пусть $\mathrm{D}' = n_2^2 - n_1^2(1-(\widehat{w_i}\widehat{n})^2)$, тогда $\mathrm{D} = \frac{4}{n_2^2}\mathrm{D}$. В силу того, что $\frac{4}{n_2^2} > 0$, сравнение знака дискриминанта можно проводить с величиной D'.

Если $\mathrm{D}' < 0$ - преломления нет.

Если $\mathrm{D}' = 0$

$$\begin{aligned}\beta = \frac{n_1}{n_2}(\widehat{w_i}\widehat{n}) \quad &\Rightarrow \quad \widehat{w_r} = -\frac{n_1}{n_2}\widehat{w_i} + \frac{n_1}{n_2}(\widehat{w_i}\widehat{n})\widehat{n} \quad \Rightarrow \\ &\Rightarrow \quad w_r = -\frac{n_1}{n_2}w_i + \frac{n_1}{n_2}\frac{w_i n}{n^2}n = \frac{n_1}{n_2}\Big(\frac{w_i n}{n^2}n - w_i\Big). \end{aligned} \tag{2.6}$$

Если $\mathrm{D}' > 0$

$$\beta = \frac{2\frac{n_1}{n_2}(\widehat{w_i}\widehat{n}) \pm \sqrt{\mathrm{D}}}{2} = \frac{2\frac{n_1}{n_2}(\widehat{w_i}\widehat{n}) \pm \frac{2}{n_2}\sqrt{\mathrm{D}'}}{2} = \frac{n_1(\widehat{w_i}\widehat{n}) \pm \sqrt{\mathrm{D}'}}{n_2}$$

Дляопределения знака подставим α и β в выражение для $\widehat{w_r}$ и проверим, что $\widehat{w_r}$ и $\widehat{n}$ расположены по разные стороны от границы пересечения, то есть $\widehat{w_r}\widehat{n} < 0$

$$\begin{aligned}\widehat{w_r}\widehat{n} =& (-\frac{n_1}{n_2}\widehat{w_i} + \frac{n_1(\widehat{w_i}\widehat{n}) \pm \sqrt{\mathrm{D}'}}{n_2}\widehat{n})\widehat{n} < 0 \\ &-\frac{n_1}{n_2}(\widehat{w_i}\widehat{n}) + \frac{n_1}{n_2}(\widehat{w_i}\widehat{n}) \pm \frac{\sqrt{\mathrm{D}'}}{n_2} = \pm\frac{\sqrt{\mathrm{D}'}}{n_2} < 0 \quad \Rightarrow \\ &\Rightarrow \quad \beta = \frac{n_1(\widehat{w_i}\widehat{n}) - \sqrt{\mathrm{D}'}}{n_2} \quad \Rightarrow \quad \widehat{w_r} = -\frac{n_1}{n_2}\widehat{w_i} + \frac{n_1(\widehat{w_i}\widehat{n}) - \sqrt{\mathrm{D}'}}{n_2}\widehat{n}.\end{aligned}$$

Таким образом итоговое значение w_r

$$w_r = \frac{n_1}{n_2}\Big(\frac{w_i n}{n^2}n - w_i\Big) - \frac{|w_i|}{|n|}\frac{\mathrm{D}'}{n_2}n. \tag{2.7}$$

2.3 Разработка алгоритмов

На рисунках 2.1 – 2.3 представлены схемы рассматриваемых алгоритмов.

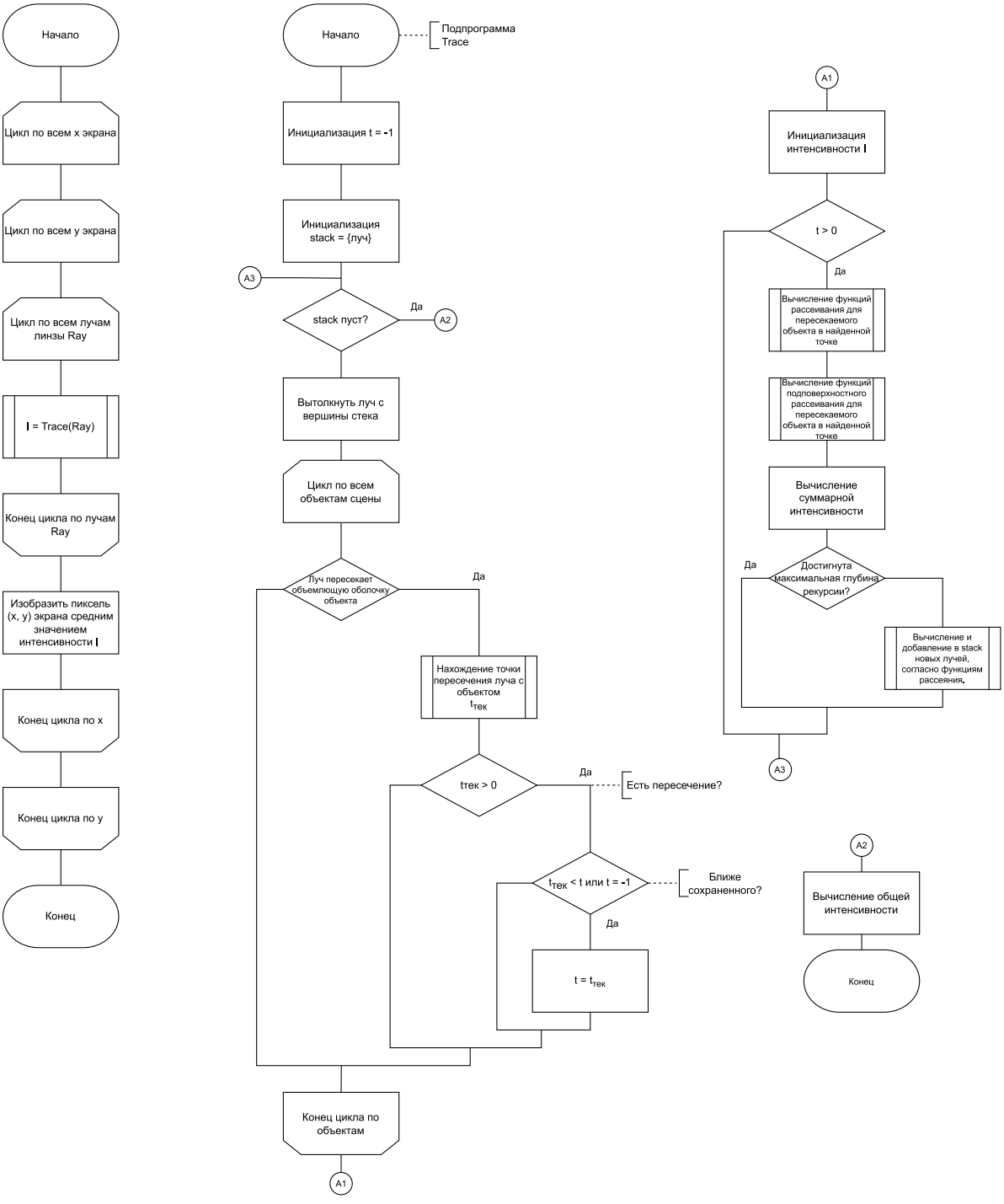

Рисунок 2.1 – Схема алгоритма трассировки путей

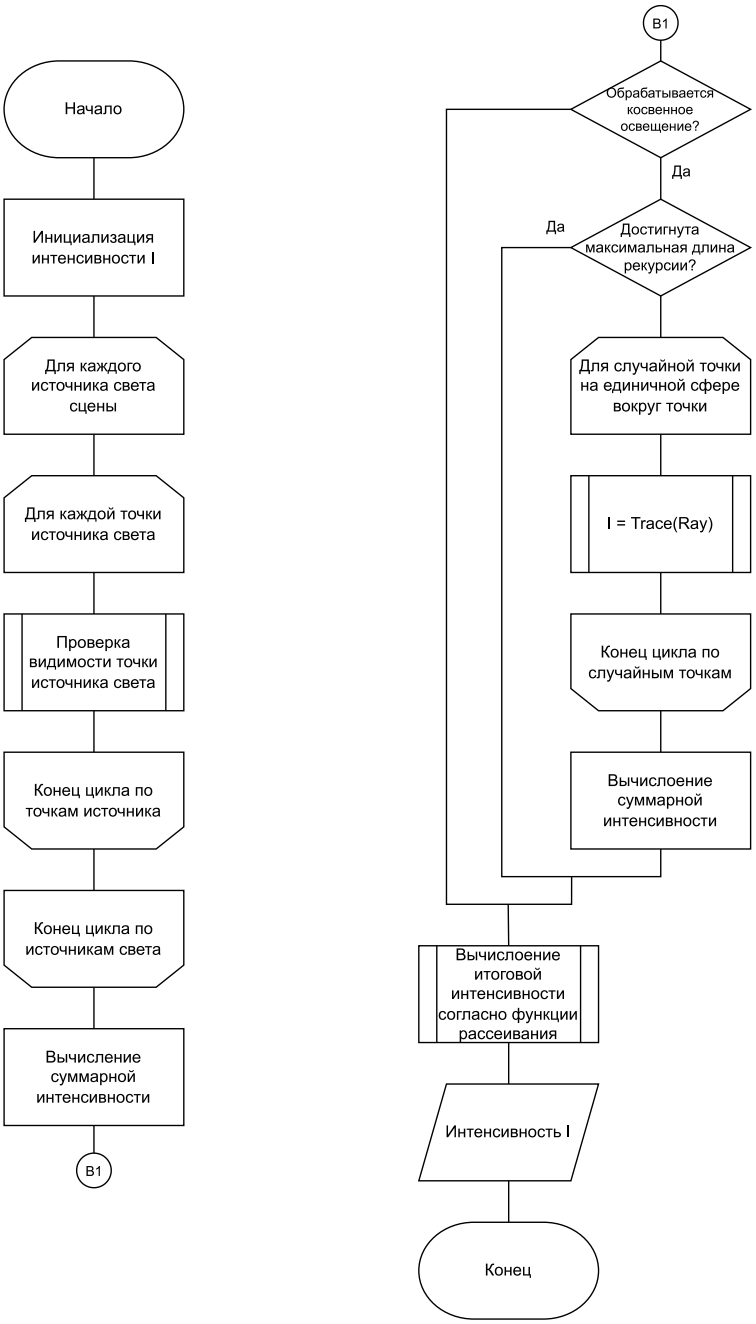

Рисунок 2.2 – Схема алгоритма вычисления функции рассеивания

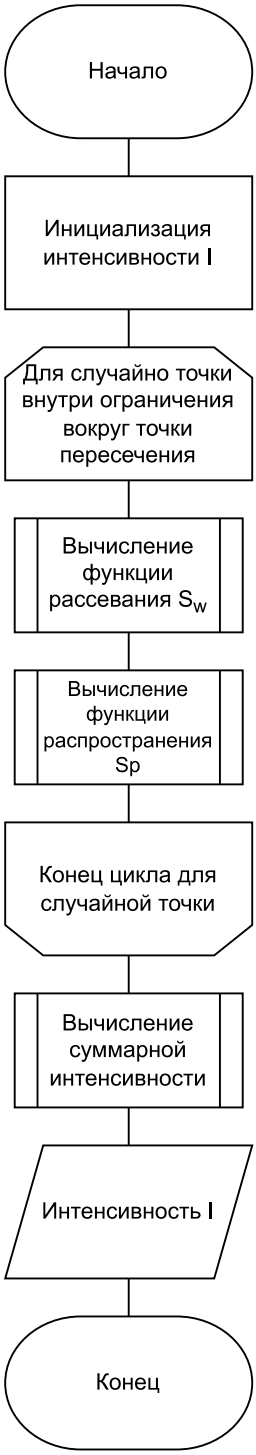

Рисунок 2.3 – Схема алгоритма вычисления функции подповерхностного рассеивания

2.4 Выбор типов и структур данных

Для реализуемого программного обеспечения необходимо реализовать следующия структуры данных:

— точка — набор вещественных координат `x, y[, z]`;

— вектор — набор вещественных координат `x, y[, z]`;

— нормаль — набор вещественных координат `x, y[, z]`;

— луч — совокупность точки и вектора;

— матрица преобразования — матрица размера 4x4 для проведения преобразований в однородных координатах;

— интенсивность — набор вещественных составляющих, соответствующих красному, зеденому и синему цветам соответственно;

— цвет — нормализованная интенсивность;

— объект — набор математических обыектов, соответствующих разделу 1.2 и две матрицы преобразования, соответствующие локальным преобразованиям и преобразованию базиса;

— текстура — матрица инетнсивностей.

Для обеспечения возможности построения сложных объектов из набора примитивов, объекты также должны содержать ссылки на своего родителя и предков.

Вывод

В данном разделе были рассмотрены требования к программному обеспечению, некоторые расчетные соотношения, схемы разработанных алгоритмов и необходимые структуры данных.

3 Технологическая часть

В данном разделе будут рассмотрены реализации алгоритмов визуализации и средства их реализации.

3.1 Средства реализации

Для реализации алгоритмов в данной работе был выбран язык C++, так как данный язык предоставляет возможность решить поставленную задачу и реализовать объектно-ориентированную модель для построения модифицируемой системы. В качестве компилятора для программы используется GNU GCC.

3.2 Диаграмма классов

На рисунке 3.1 предсавлена диаграмма классов разработанного приложения.

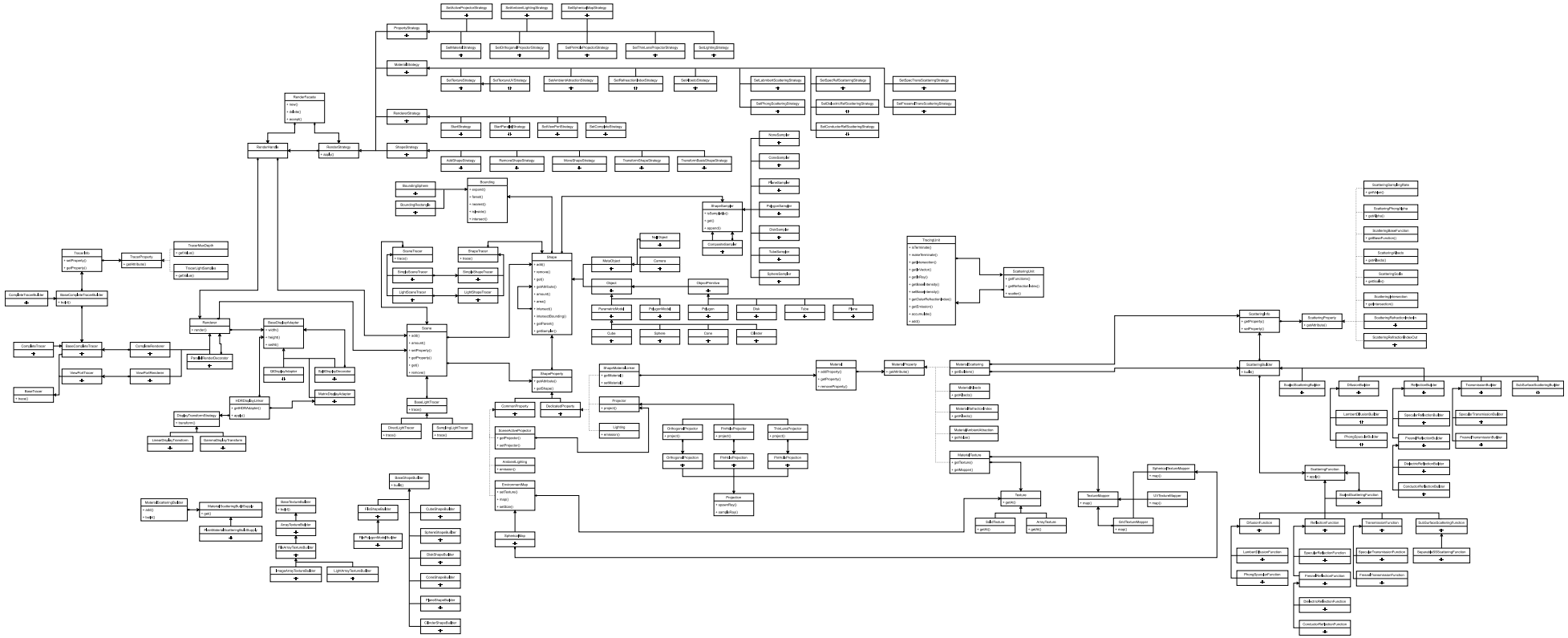

Рисунок 3.1 – Диаграмма классов

3.3 Сведения о модулях программы

Программа состоит из следующих модулей:

— `scattering_propoerties/*` — свойства (аргументы) функций рассеивания;

— `boundings/*` — объемлющие оболочки геметрицеских объектов;

— `display_transform_strategies/*` — стратегии преобразования изображения в расширенном динамическом пространстве;

— `light_tracers/*` — трассировщики источников света;

— `material_properties/*` — свойства материлов;

— `math/*` — математические абстракции (точка, вектор, нормаль, луч, преобразования);

— `matrix/*` — реализация матрицы;

— `projections/*` — проекции из пространства камеры в глобальные координаты;

— `renderers/*` — визуализаторы;

— `scattering_builders/*` — строители функций рассеивания;

— `scattering_functions/*` — функции рассеивания;

— `scene_tracers/*` — трассировщики сцены;

— `shapes/*` — геометрические объекты;

— `shape_builders/*` — строители геометрических объектов;

— `shape_properties/*` — свойства геометрических объектов;

— `shape_properties/projectors/*` — виды камер;

— `shape_samplers/*` —

— `shape_tracers/*` — трвссировщики геометрических объетов;

— `texture/*` — текстуры;

— `texture_builders/*` — строители текстур;

— `texture_mappers/*` — преобразователи текстурных координат;

— `tracers/*` — трассировщики путей;

— `tracer_builders/*` — строители трассировщиков путей;

— `tracer_properties/*` — свойства (аргументы) трассировщика путей;

— `intersection` — пересечение луча с геометрическим объектом;

— `scattering_unit` — модуль рассеивания;

— `tracer_info` — коллекция свойст трассировщика;

— `hdr_display_linker` — модуль связи адаптеров дисплея расширенного динамического диапазона;

— `material` — материал, коллекция свойств материалов;

— `scene` — сцена, коллекия геометрических объектов и их свойств;

— `tracing_unit` – модуль, представляющий взаимодействия на границе сред;

— `intensity` — интенсивность;

— `scattering_info` — коллекция свойств функций рассеивания;

— `tools` — модуль, содержащий прикладные функциии;

— `transform_strategies` — стратегии преобразования геометрических объектов.

3.4 Реализации алгоритмов

На листингах 3.1 – 3.4, 3.3 – 3.4 представлена реализация классов, отвечающих за алгоритм трассировки путей, и вычисления функций рассеивания, а на листингах **??** и **??** — реализация алгоритмов вычисления освещенности в точке.

Листинг 3.1 – Листинг кода визуализатора в алогритме трассировки путей (1)

```
#ifndef _COMPLETE_RENDERER_H_
#define _COMPLETE_RENDERER_H_

#include <memory>

#include "renderer.h"
#include "base_complete_tracer.h"
#include "projection.h"

class CompleteRenderer : public Renderer
{
    private:
        using SpawnMethod = Ray3<double> (Projection::*)(size_t, size_t) const;

    public:
        CompleteRenderer(const std::shared_ptr<BaseCompleteTracer> &tracer,
                         const size_t &samples = 1);
        virtual ~CompleteRenderer(void) override;
        virtual void render(const Scene &scene,
                            BaseDisplayAdapter &adapter,
                            RenderProgress *const progress = nullptr) override;

    private:
        std::shared_ptr<BaseCompleteTracer> tracer;
        SpawnMethod spawn_method;
        size_t samples;
        double isamples;
};

DEF_EX(CommonCompleteRendererException, CommonRendererException,
```

```
31        "General complete renderer exception");
32 DEF_EX(NoActiveProjectorCompleteRenderer, CommonCompleteRendererException,
33        "No active projector in scene");
34 DEF_EX(NoSamplesCompleteRenderer, CommonCompleteRendererException,
35        "Samples amount can't be zero");
36
37 #endif
```

Листинг 3.2 – Листинг кода визуализатора в алогритме трассировки путей (2)

```
1  #include "complete_renderer.h"
2
3  #include "scene_active_projector.h"
4  #include "projector.h"
5
6  CompleteRenderer::CompleteRenderer(const std::shared_ptr<BaseCompleteTracer> &tracer,
7                                     const size_t &samples)
8      : tracer(tracer)
9  {
10     if (0 == samples)
11         throw CALL_EX(NoSamplesCompleteRenderer);
12
13     this->samples = samples;
14     this->isamples = (double)1 / this->samples;
15     this->spawn_method = &Projection::spawnRay;
16
17     if (1 != samples)
18         this->spawn_method = &Projection::sampleRay;
19 }
20
21 CompleteRenderer::~CompleteRenderer(void) {}
22
23 void CompleteRenderer::render(const Scene &scene,
24                               BaseDisplayAdapter &adapter,
25                               RenderProgress *const progress)
26 {
27     std::list<std::shared_ptr<const ShapeProperty>> lst =
       scene.getProperties(SceneActiveProjector::ATTRIBUTE());
28
29     if (0 == lst.size())
30         throw CALL_EX(NoActiveProjectorCompleteRenderer);
31
32     const Projector &projector = std::static_pointer_cast<const
       SceneActiveProjector>(lst.front())->getProjector();
33     std::shared_ptr<Projection> projection = projector.project(adapter);
34
35     for (size_t i = 0; adapter.width() > i; i++)
36         for (size_t j = 0; adapter.height() > j; j++)
37         {
38             Intensity<> avg;
39
40             for (size_t k = 0; this->samples > k; k++)
41                 avg += this->tracer->trace(scene,
42                         (projection.get()->*this->spawn_method)(i, j));
43
44             adapter.setAt(i, j, avg * this->isamples);
45
46             if (nullptr != progress)
47                 progress->step();
48         }
49 }
```

Листинг 3.3 – Листинг кода трассировщика в алогритме трассировки путей (1)

```
#ifndef _COMPLETE_TRACER_H_
#define _COMPLETE_TRACER_H_

#include "base_complete_tracer.h"

class CompleteTracer : public BaseCompleteTracer
{
    public:
        CompleteTracer(size_t max_depth, size_t light_samples);
        virtual ~CompleteTracer(void) override;
        virtual Intensity<> trace(const Scene &scene,
                                  const Ray3<double> &ray) const override;

    public:
        const size_t max_depth;
        const size_t light_samples;
};

#endif
```

Листинг 3.4 – Листинг кода трассировщика в алогритме трассировки путей (2)

```
...
Intensity<> CompleteTracer::trace(const Scene &scene, const Ray3<double> &ray) const
{
    SimpleSceneTracer stracer;
    SamplingLightTracer ltracer (scene, this->light_samples);

    common_prop_t common = get_common_prop(scene);
    ScatteringUnit head (scene,
                         std::list<std::shared_ptr<const ScatteringFunction>>(),
                         ray);
    head.scatter(stracer);

    std::stack<stack_tracing_unit_t> stack;

    for (std::shared_ptr<TracingUnit> &unit : head)
        stack.push({unit, {}, 1, false});

    while (0 != stack.size())
    {
        stack_tracing_unit_t &current = stack.top();

        if (!current.start)
            init_unit_arg(common, scene, current, stracer, ltracer);
        else
            current.unit->accumulate(current.iter++);

        if (current.unit->isTerminate() || current.unit->end() == current.iter)
            stack.pop();
        else if (this->max_depth > current.depth)
            for (std::shared_ptr<TracingUnit> &unit : **current.iter)
                stack.push({unit, {}, current.depth + 1, false});
    }

    Intensity<> out;

    for (std::shared_ptr<TracingUnit> &unit : head)
        out += unit->getBaseIntensity() * unit->getEmission();

```

```
39      return out;
40  }
41  ...
```

3.5 Интерфейс программы

Интерфейс разработанного приложения приведен на рисунке 3.2

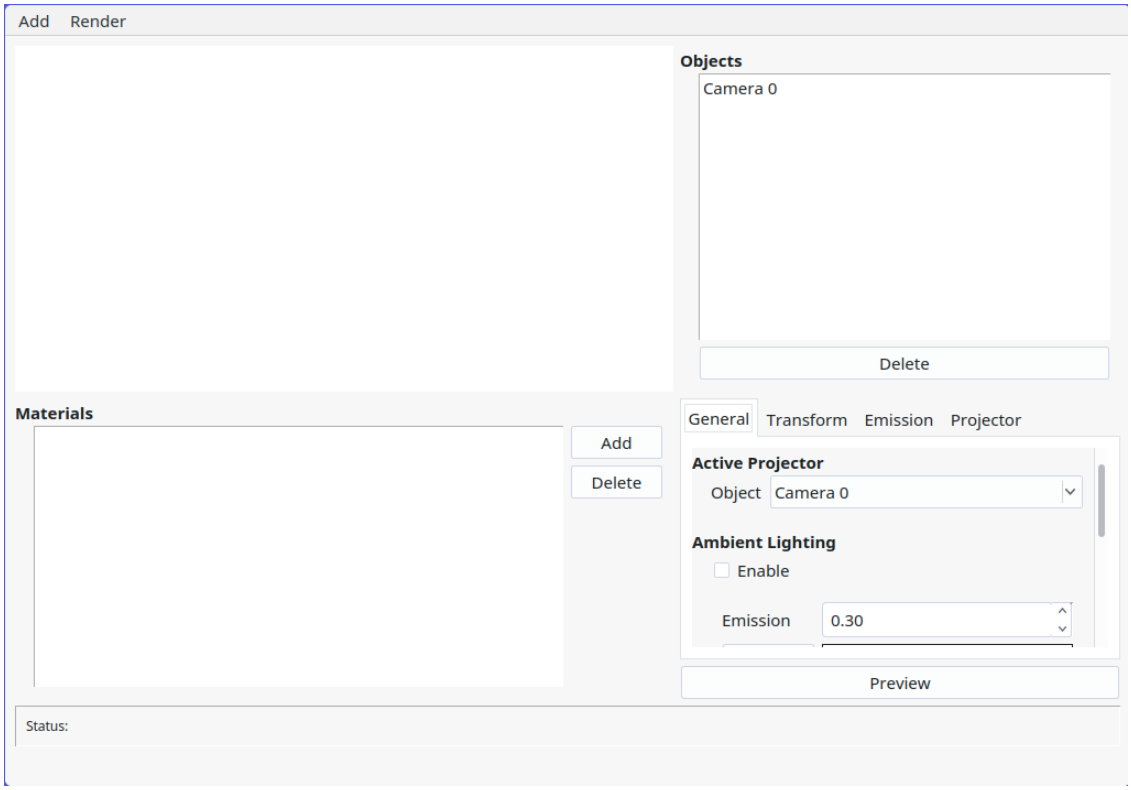

Рисунок 3.2 – Интерфейс программы

Вывод

В данном разделе были рассмотрены реализации алгоритмов визуализации.

4 Экспериментальная часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование.

— Операционная система: Arch 6.1.9 [8] Linux [9] x86_64.

— Память: 16 Гб.

— Процессор: AMD Ryzen™ 5 4600H CPU @ 3.0G ГГц [10].

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Постановка эксперимента

4.2.1 Цель эксперимента

Цель эксперимента - оценить время работы программы под влиянием паараметров таких, как размер изображения, количество потоков, разбиение по горизонтали/вертикали.

Исходя из ожиданий, с увеличение количества потоков время работы алгоритма должно сокращаться в опбратной зависимости.

Для подтверждения данной гипотезы необходимо провести теоретические замеры времени выполнения.

4.2.2 Проведение замеров

На рисунке 4.1 преведено время работы алгоритма от размера визуализируемого изображения в случае однопоточной работы приложения.

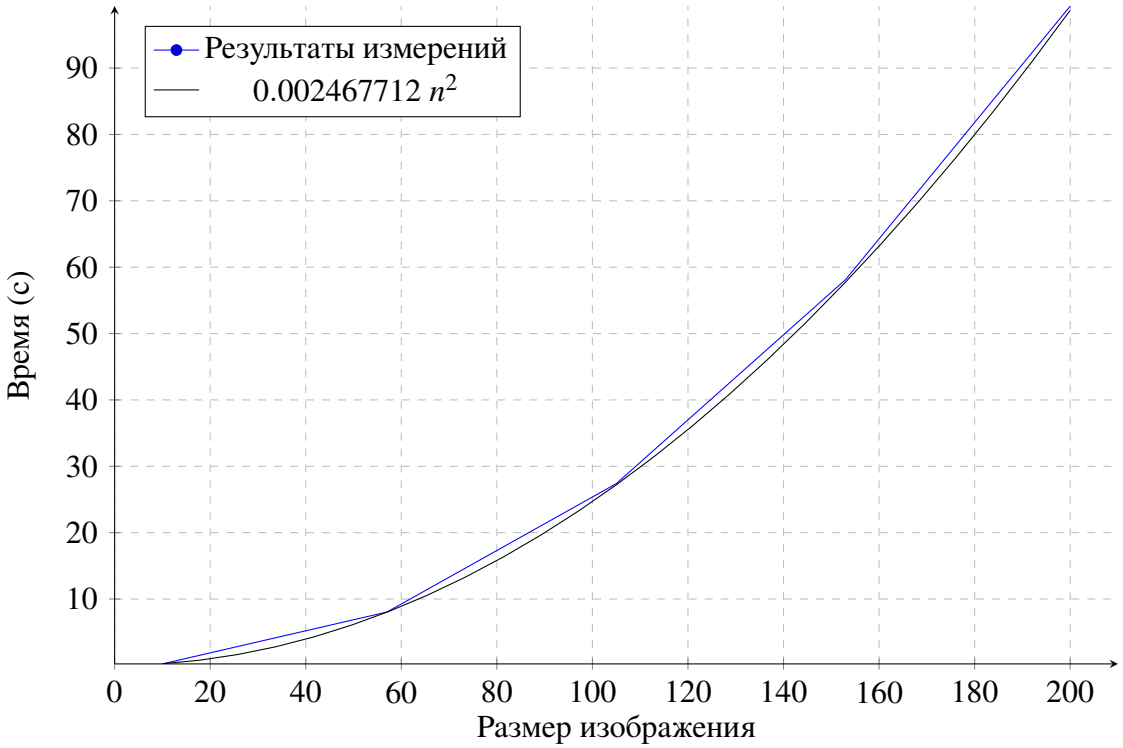

Рисунок 4.1 – Замеры времени работы программы в случае линейного увеличения количества пикселей экрана

На рисунках 4.2 – 4.6 представлены соответсвующие результаты замеров времени.

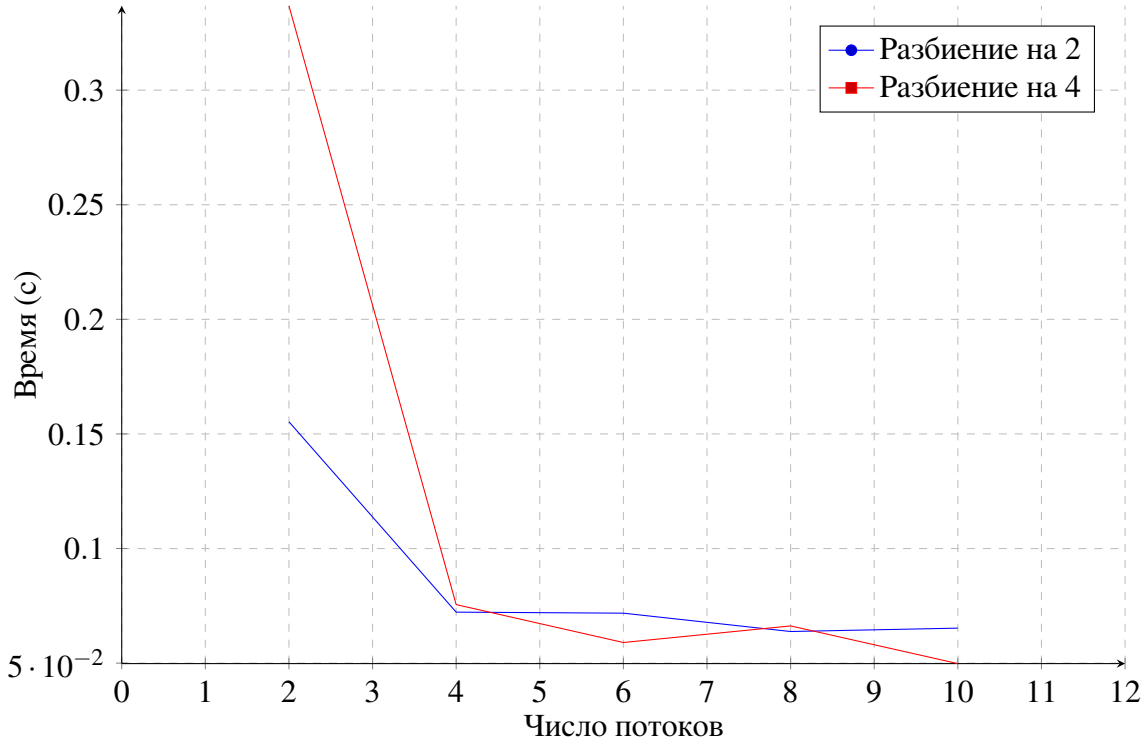

Рисунок 4.2 – Замеры времени работы программы в случае увеличения числа потоков для изображения размером 10 пикселей

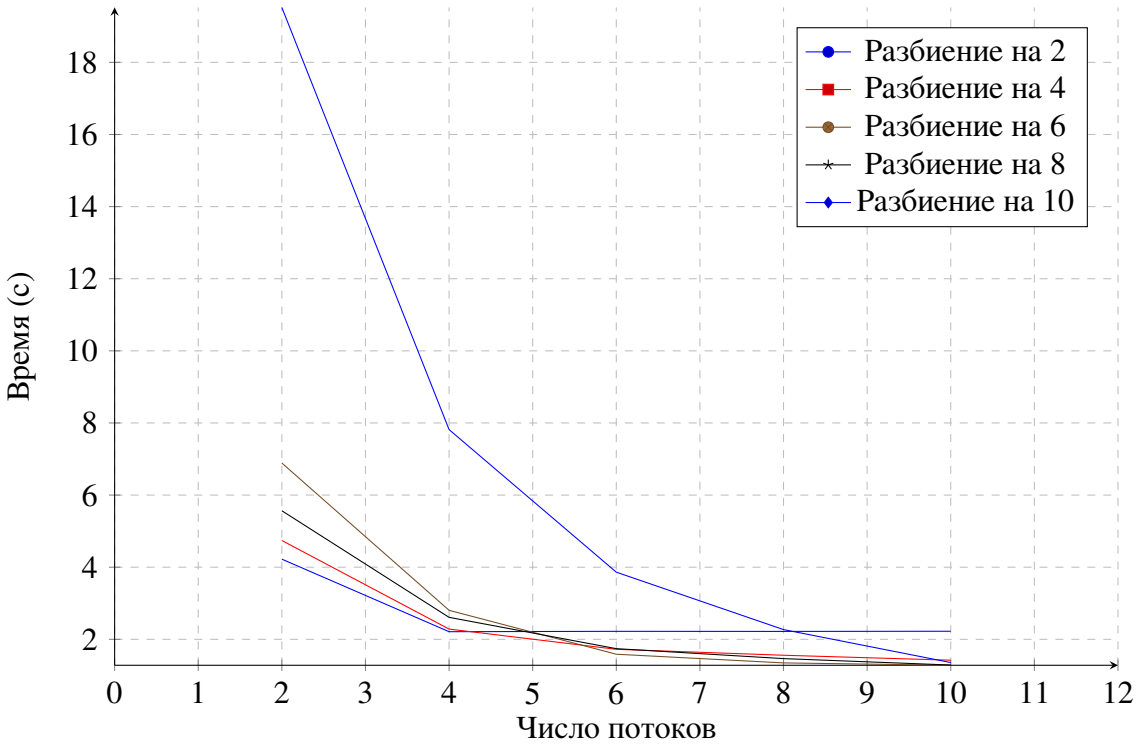

Рисунок 4.3 – Замеры времени работы программы в случае увеличения числа потоков для изображения размером 57 пикселей

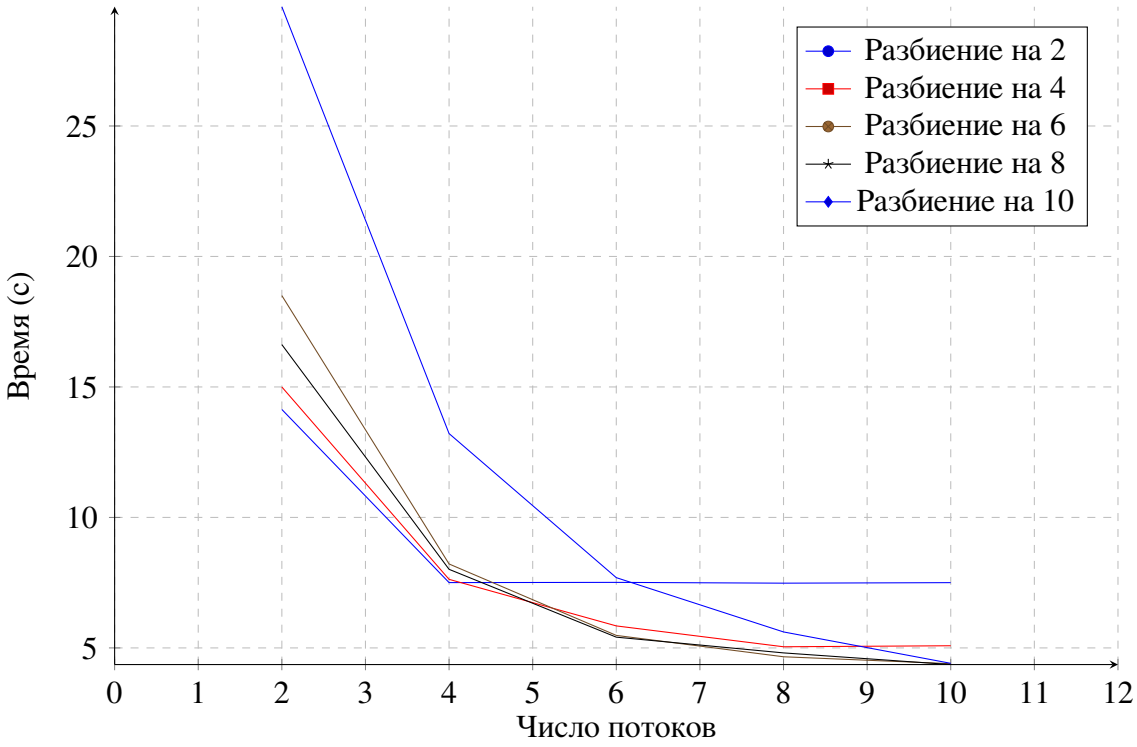

Рисунок 4.4 – Замеры времени работы программы в случае увеличения числа потоков для изображения размером 105 пикселей

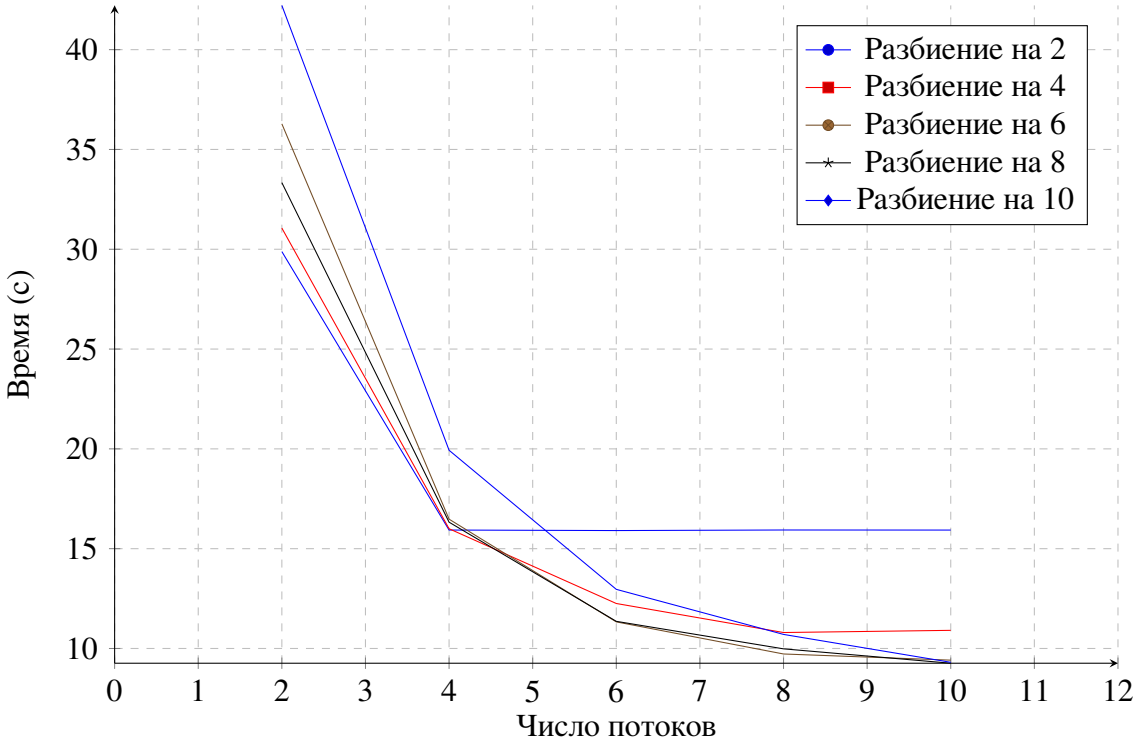

Рисунок 4.5 – Замеры времени работы программы в случае увеличения числа потоков для изображения размером 153 пикселей

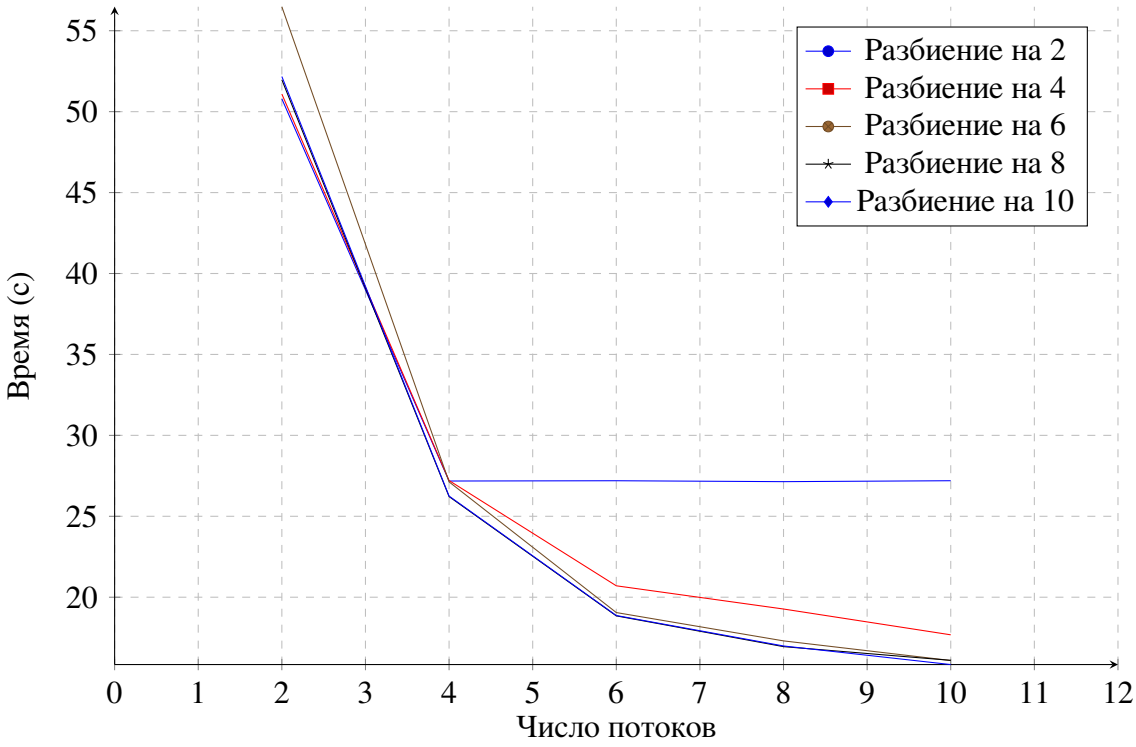

Рисунок 4.6 – Замеры времени работы программы в случае увеличения числа потоков для изображения размером 200 пикселей

Вывод

В результате проведения эксперимента было выявлено, что время работы программы в случае применеия распараллеливания действительно стремится к обратной зависимости от числа потоков. При этом, было замечено, что с ростом размера изображения, затрачиваемое время увеличивается согласно закону $O(n^2)$.

Заключение

В ходе выполнения данной работы была разработана программа, позволяющей создать реалистическое изображение без учета подповерхностного рассеивания, на основе трехмерной сцены наполненной объектами (сфера, куб, конус, цилиндр, полигональная модель) и источниками света, положение, количество и свойства которых может задаваться пользователем интерактивно.

Были решены следующие задачи:

— выявлены неотъемлемые качества реалистического изображения;

— проведен анализ существующих алгоритмов;

— выбран путь для решения основной задачи;

— реализованы выбранные алгоритмы;

— создан программный продукт для решения задачи.

— исследовано время работы полученной программы в случае распараллеливания алгоритма.

Список использованных источников

1. В.И. Гонахчян. Алгоритм удаления невидимых поверхностей на основе программных проверок видимости // Труды ИСП РАН. 2018. Т. 2.

2. Юрьевна Янова Регина. Метод прямой и обратной трассировки // Вестник науки и образования. 2016. Т. 5.

3. Фролов В. А. Игнатенко А. В. Фотореалистичная визуализация с помощью трассировки путей на графических процессорах // Новые информационные технологии в автоматизированных системах. 2010. Т. 13. С. 149–150.

4. Емельянова Татьяна Владимировна Аминов Леонид Анатольевич Емельянов Владислав Андреевич. Реализация алгоритма удаления невидимых граней // Актуальные проблемы военно-научных исследований. 2021. Т. 2.

5. Matt Pharr Wenze Jakob G. H. Phisically based rendering. Morgan Kaufman publishers, 2017. P. 1233.

6. Fernando R. GPU Gems Programming Techniques, Tips, and Trick for Real-Time Graphics. Addison-Wesley, 2004. P. 765.

7. Henrik Wann Jensen Stephen R. Marschner M. L. P. H. A Practical Model for Subsurface Light Transport // Staford University. 2001. P. 8.

8. Arch 6.1.9 [Электронный ресурс]. Режим доступа: URL – https://archlinux.org/. (Дата обращения: 22.03.23).

9. Linux – Getting Started [Электронный ресурс]. Режим доступа:
URL – https://linux.org. (Дата обращения: 22.03.23).

10. Процессор AMD Ryzen™ 5 4600H [Электронный ресурс]. Режим доступа:
URL – https://www.amd.com/ru/products/apu/amd-ryzen-5-4600h.
(Дата обращения: 22.03.23).