Оглавление

Введение

Физические тела, окружающие нас, обладают различными оптическими свойствами - отражение и преломление световых лучей, отбрасывание теней. Эти и другие свойства нужно уметь наглядно визуализировать на экране компьютера. Этим и занимается компьютерная графика.

Компьютерная графика – это совокупность методов и способов преобразования информации в графическую форму и из графической формы в ЭВМ. Область применения компьютерной графики широка: от воссоздания художественных эффектов в компьютерных играх до построения трехмерных объектов при моделировании сложных технологических продуктов.

Для реализации подобных задач в компьютерной графике существует множество алгоритмов и подходов. Основная проблема заключается в том, что для получения качественного изображения требуется большое количество времени и памяти.

Целью данного курсового проекта является разработка ПО, которое предоставляет трехмерную визуализацию вращения флюгера.

Для достижения поставленной цели необходимо решить следующие задачи:

1. изучить и проанализировать существующие алгоритмы построения реалистичных изображений;
2. выбрать алгоритмы, наиболее подходящие для решения поставленной задачи;
3. спроектировать архитектуру будущего программного продукта и выбрать структуры данных для представления объектов синтезируемой сцены;
4. разработать программу на основе выбранных алгоритмов и структур данных;

5. на основе разработанной программы провести исследование зависимости времени рендеринга изображения от количества используемых потоков и от количества объектов на сцене.

1. Аналитическая часть

В данном разделе приводятся постановка задачи и формализация объектов синтезируемой сцены, анализируются существующие для реализации поставленной задачи алгоритмы и модели, из которых затем выбираются наиболее подходящие.

1.1. Постановка задачи

Необходимо разработать программный продукт, который представит трехмерную визуализацию вращения флюгера, реагирующего на направленные потоки ветра. Сцена должна быть реалистичной и учитывать такие явления, как отражение и отбрасывание теней. Пользователь должен иметь возможность изменения: направления и скорости вращения флюгера, параметров источников освещения, камеры и материалов составных частей флюгера.

1.2. Формализация объектов синтезируемой сцены

Сцена должна состоять из следующих объектов:

1) Направленные источники света. Они описываются вектором направления испускания света и его интенсивностью. Предполагается, что источник расположен в бесконечности.
2) Точечные источники света. Они описывается фиксированной точкой в пространстве (позицией) и интенсивностью. Предполагается, что свет от такого источника распространяется равномерно во всех направлениях.
3) Плоскость основания флюгера и задний фон.
4) Флюгер, состоящий из таких геометрических примитивов, как цилиндр, прямоугольный параллелепипед, сфера, четырехугольная пирамида (пример такого флюгера представлен на рисунке 1.1).

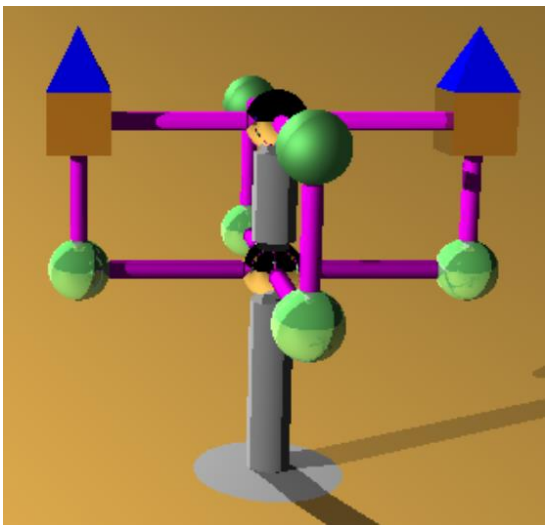

Рисунок 1.1. Пример флюгера из геометрических примитивов.

В компьютерной графике в основном используются 3 вида моделей трехмерных объектов [1]:

1) Каркасная (проволочная) модель. Это простейший вид моделей, содержащий минимум информации - о вершинах и рёбрах объектов. Главный недостаток - такая модель не всегда правильно передает представление об объекте (например, если в объекте есть отверстия).
2) Поверхностная модель. Отдельные участки задаются как участки поверхности того или иного вида. Эта модель решает проблему каркасной, но все еще имеет недостаток - нет информации о том, с какой стороны поверхности находится собственный материал
3) Объемная модель. В отличии от поверхностной, содержит указание расположения материала (чаще всего указанием направления внутренней нормали).

Важнейшие требования к модели – правильность отображения информации об объекте и компактность. В рамках поставленной задачи каркасная модель не удовлетворяет первому критерию, а информация о том, где расположен материал, не является необходимой, что делает объемную модель избыточной, поэтому поверхностная модель является наиболее подходящей.

Поверхность сферы и боковую поверхность цилиндра удобно описывать аналитическими уравнением, а основание цилиндра – как участок плоскости. Параллелепипед, у которого грани параллельны координатным плоскостям проще всего описать уравнениями этих граней. Однако в данном проекте каждый из примитивов должен поддаваться операции поворота, в результате которого параллельность может нарушиться. Поэтому для представления параллелепипеда корректней использовать полигональную аппроксимацию треугольниками. Так же можно описать и четырехугольную пирамиду.

1.3. Анализ алгоритмов удаления невидимых линий и поверхностей

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

В зависимости от того, в каком пространстве решается задача, алгоритмы делят на следующие группы:

1) Алгоритмы, работающие в объектном пространстве (мировая системе координат). Такие алгоритмы предоставляют высокую точность, но требуют много ресурсов.
2) Алгоритмы, работающие в пространстве изображений (в системе координат, связанной с устройством, на котором отображается результат). Менее ресурсоемкие и точные.
3) Иногда отдельно выделяют алгоритмы, работающие в обоих пространствах попеременно.

Для решения задачи, поставленной в данной работе, необходимо уметь строить реалистичные изображения. Поэтому основным критерием выбора алгоритма удаления невидимых линий и поверхностей является возможность учета эффектов отражения и отбрасывания теней.

1.3.1. Алгоритм Робертса

Алгоритм Робертса работает в объектном пространстве и состоит из следующих этапов:

1) (Подготовительный) Формирование исходных данных.
2) Удаление линий, экранируемых самим телом.
3) Удаление линий, экранируемых другими телами.
4) Удаление линий пересечения тел, экранируемых самими телами, связанными отношением протыкания, и другими телами.

Преимущество алгоритма Робертса в том, что он целиком основам на математических предпосылках, которые просты, точны и мощны.

К недостаткам этого алгоритма можно отнести большую трудоемкость, невозможность работы с невыпуклыми телами, а также тот факт, что без модификации и привлечения сторонних методов данный алгоритм не позволяет учитывать тени и зеркальные эффекты.

1.3.2. Алгоритм Варнока

Алгоритм Варнока работает в пространстве изображения и основывается на рекурсивном разбиении экрана. Главная идея - на каждом шаге найти ответ на вопрос о том, что изображать в очередном окне. Если нельзя точно дать ответ, то окно делится на части, пока не сможем решить, что изображать, или окно не дойдёт до размеров в один пиксель.

В простейшей версии алгоритма окно делится на подокна всякий раз, если это окно не пусто. В более сложных версиях делается попытка решения задачи для окон большего размера. Для этого проводится классификация многоугольников по отношению к ячейке: внешний, внутренний, пересекающий или охватывающий. Затем определяются действия, которые нужно предпринять в том или ином случае взаимного расположения ячейки и полигонов.

Достоинством данного алгоритма является простота реализации и высокая эффективность в случае, если размеры перекрываемых областей невелики.

К недостаткам алгоритма относится трудоемкость в случае, когда синтезируемая сцена сложная и число разбиений становится очень большим, а также отсутствие учета оптических свойств объектов.

1.3.3. Алгоритм, использующий Z-буфер

Алгоритм Z буфера решает задачу в пространстве изображений и является одним из самых простых и широко используемых. Его идея заключается в использовании двух буферов: буфера кадра, хранящего интенсивности каждого пикселя в пространстве изображения, и буфера глубины (Z-буфера), в котором запоминается значение координаты Z каждого видимого пикселя [1].

В ходе работы алгоритма значение глубины каждого нового пикселя, заносимого в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в Z-буфер. Если новый пиксель расположен ближе к наблюдателю, чем уже находящийся в буфере кадра, то новый пиксель заносится в буфер кадра и производится корректировка Z-буфера: в него заносится глубина нового пикселя.

Основными достоинствами данного алгоритма являются простота реализации, допустимость сцен любой сложности, линейная зависимость трудоемкости от числа объектов на сцене.

К недостаткам алгоритма Z-буфера относят необходимость выделения памяти под два буфера, каждый из которых имеет размер равный количеству пикселей на экране (что, впрочем, при современном развитии технологий уже не так болезненно). Также алгоритм не учитывает тени, эффекты прозрачности и зеркальности.

1.3.4. Алгоритм трассировки лучей.

Алгоритм трассировки лучей работает в пространстве изображений и имеет 2 подхода: прямой и обратный [2].

Основная идея алгоритма прямой трассировки лучей состоит в том, что наблюдатель видит объекты благодаря световым лучам, которые испускает некоторый источник и которые падают на объект, отражаются, преломляются или проходят через него и в результате достигают зрителя. Если проследить за лучами, то становится понятно, что среди них лишь малая часть дойдет до наблюдателя, что показано на рисунке 1.2 слева, а значит большая часть вычислений произведена напрасно.

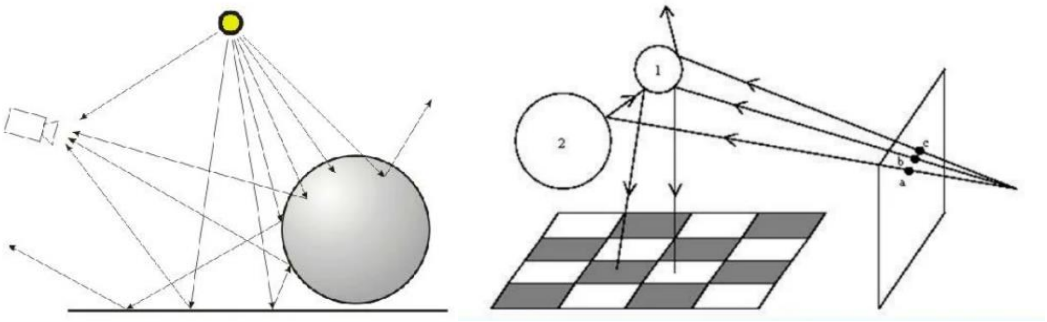

Рисунок 1.2. Прямая и обратная трассировки лучей.

Заменой данному алгоритму служит метод обратной трассировки лучей, который отслеживает лучи в обратном направлении. В ходе работы испускаются лучи от наблюдателя, как показано на рисунке 1.2 справа, и ищутся пересечения луча и всех объектов сцены.

Каждый раз, когда луч пересекает некоторую поверхность, из точки пересечения испускаются новые лучи - отраженный и преломленный. Пути этих лучей отслеживаются по всей модели, и если лучи пересекают другие поверхности, то снова испускаются лучи.

В каждой точке, где луч пересекает поверхность, рисуется луч тени из точки пересечения к каждому источнику света. Если этот луч пересекает другую поверхность перед тем, как достигнуть источника света, то на ту поверхность, с которой был послан луч, падает тень с поверхности, блокирующей свет

К достоинствам данного алгоритма можно отнести предоставление возможности визуализировать оптические эффекты: тени, прозрачность, отражение, что делает полученное изображение очень реалистичным.

Серьёзным недостатком алгоритма трассировки является производительность: необходимо создавать большое число лучей, проходящих через сцену, которые могут раздваиваться и требовать повторных вычислений. Однако этот негативный аспект можно решить путем распараллеливания: поскольку каждый луч, исходящий из камеры, независим от всех остальных, лучи можно трассировать одновременно, а также ограничением глубины разбиения.

1.4. Анализ моделей освещения

Модели освещения используются для вычисления интенсивности света для данной точки на поверхности модели. Существует множество моделей, каждый из которых учитывает различное число параметров, но их все можно разделить на локальные и глобальные.

В локальной модели учитывается только свет от источников (первичные лучи) и ориентация поверхности. В глобальной модели учитывается еще и свет, отраженный от других поверхностей или пропущенный через них (вторичные источники).

Самая простая модель освещения представляет собой сумму трех световых составляющих: фоновая (ambient), рассеянная, или диффузная (diffuse) и зеркальная (specular). Вклад каждой из них показан на рисунке 1.3.

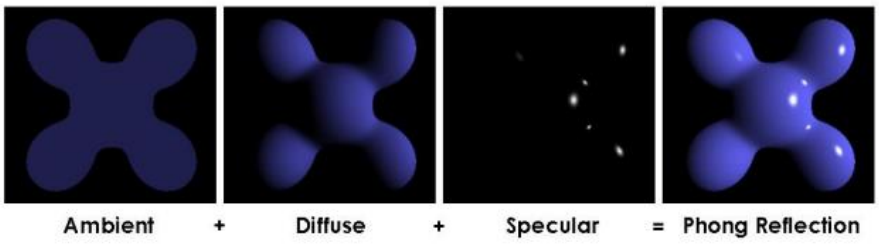

Рисунок 1.3. Три компоненты освещения.

Так как в данной задаче необходимо построить реалистичное изображение, то модель освещения должна учитывать все эти составляющие и при этом быть физически корректной.

Первая компонента - фоновое освещение - присутствует в любом уголке сцены и не зависит от пространственных координат освещаемой точки и источника, поэтому ее интенсивность обычно задается константой ia для всей сцены. Фоновая составляющая освещенности в точке Ia зависит только от ia и ka - свойства материала воспринимать фоновое освещение, и рассчитывается по следующей формуле:

$$Ia = ka * ia \quad (1.1)$$

1.4.1. Модель Ламберта

Модель освещения Ламберта позволяет рассчитывать вторую компоненту - рассеянное освещение, - и является базовой для большинства остальных моделей.

Считается, что свет, падающий в точку, одинаково рассеивается по всем направлением полупространства. Рассеянная составляющая освещения в точке Id зависит только от угла α между вектором падения света $\vec{L}$ и вектором нормали $\vec{N}$ в этой точке, что показано на рисунке 1.4, интенсивности рассеянного освещения id и свойства материала воспринимать рассеянное освещение kd.

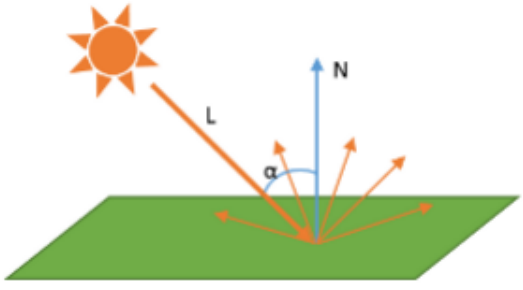

Рисунок 1.4. Вычисление рассеянной составляющей в модели Ламберта.

Формула для расчета диффузной составляющей:

$$Id = kd * \cos(\vec{L}, \vec{N}) * id \quad (1.2)$$

Наибольшая интенсивность достигается при перпендикулярном падении света на поверхность и убывает с увеличением угла α. Угол более 90 градусов означает, что источник света находится ниже освещаемой поверхности, и, следовательно, вклад диффузного освещения должен равняться нулю.

Поверхность, освещение которой рассчитывается по модели Ламберта, выглядит одинаково яркой со всех направлений и не позволяет передавать блики на телах сцены, так как не учитывает зеркальную составляющую освещения.

1.4.2. Модель Фонга

Модель Фонга добавляет в модель Ламберта зеркальную составляющую Is [3]. Падающий $\vec{L}$ и отраженный $\vec{R}$ лучи лежат в одной плоскости с нормалью $\vec{N}$ к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части.

Отраженная составляющая освещенности в точке Is зависит от того, насколько близки направления отраженного вектора $\vec{R}$ и вектора $\vec{V}$, направленного на наблюдателя, показанных на рисунке 1.5, а также от интенсивности зеркального освещения is и коэффициентов зеркального отражения ks и блеска p материала.

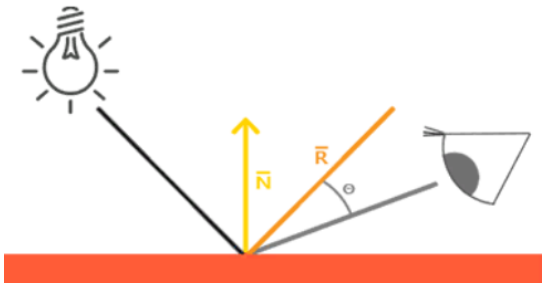

Рисунок 1.5. Вычисление зеркальной составляющей в модели Фонга.

Формула для расчета зеркальной составляющей в модели Фонга:

$$Is = ks * (\cos(\vec{R}, \vec{V}))^{p} * is \quad (1.3)$$

При этом угол между вектором обзора и вектором отражения не должен превышать 90 градусов, иначе их скалярное произведение становится отрицательным. И если при расчете диффузной составляющей освещения подобное ограничение имеет место и в реальном мире, то в данном случае часть зеркальной компоненты может теряться.

Таким образом, хоть модель Фонга и включает все необходимые компоненты, она не предоставляет вполне реалистичную картину.

1.4.3. Модель Блинна-Фонга

Озвученную выше проблему решает модель Блинна-Фонга, которая во многом схожа с моделью Фонга, но использует другой подход к расчету зеркальной компоненты: угол между вектором на наблюдателя $\vec{V}$ и отраженного луча $\vec{R}$ заменяется на угол между нормалью к поверхности $\vec{N}$ и вектором $\vec{H}$, средним между $\vec{V}$ и вектором на источник света $\vec{L}$ (см. рисунок 1.6) [3].

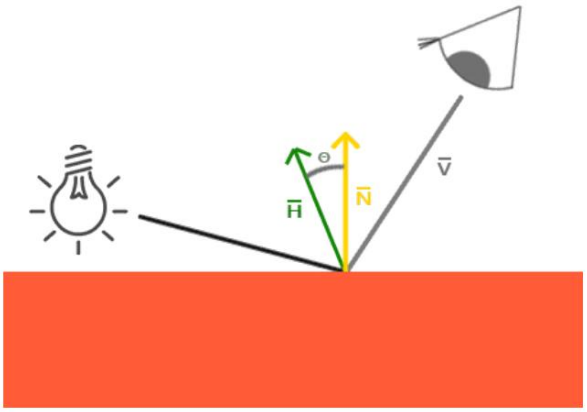

Рисунок 1.6. Вычисление зеркального света в модели Блинна-Фонга.

Формула для расчета зеркальной составляющей в модели Блинна-Фонга:

$$Is = ks * (\cos(\vec{N}, \vec{H}))^p * is \text{ , где } \vec{H} = \frac{\vec{L}+\vec{V}}{||\vec{L}+\vec{V}||} \qquad (1.4)$$

Чем ближе вектор $\vec{H}$ к нормали поверхности, тем больше будет вклад зеркальной компоненты. Вне зависимости от направления, с которого смотрит наблюдатель, угол между медианным вектором и нормалью к поверхности

превысит 90 градусов, только если источник света находится ниже поверхности. В такой ситуации модель Блинна-Фонга останется физически корректной: такой же результат (отсутствие бликов) будет наблюдаться и в реальном мире. Это предоставляет возможность получить правдоподобную картину, показанную на рисунке 1.7 справа, в сравнении с Фонговским освещением, показанным на том же рисунке слева.

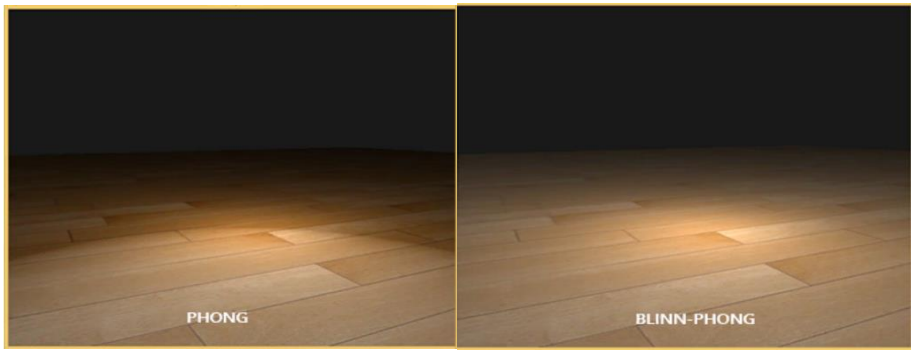

Рисунок 1.7. Сравнение моделей Фонга и Блинна-Фонга.

Таким образом, модель Блинна-Фонга включает все необходимые компоненты освещения и дает более реалистичную картину, чем модель Фонга. Более того, теперь нет необходимости в вычислении вектора отражения.

Однако важно учесть, что угол, вычисляемый в модели Фонга, часто больше угла в модели Блинна-Фонга, поэтому значение силы зеркального блеска во второй должно быть немного выше, чем в первой (эмпирически установлено, что оно должно быть примерно в 2-4 раза больше).

Вывод

В соответствии со сформулированными критериями выбора алгоритма удаления невидимых линий и модели освещения, для реализации поставленной задачи были выбраны:

1) Алгоритм обратной трассировки лучей для удаления невидимых линий и поверхностей. Он позволяет получить реалистичное изображение высокого

качества, в котором учитываются такие оптические эффекты, как отражение и отбрасывание теней.

2) Модель Блинна-Фонга выбрана в качестве основной для расчета интенсивности в точке. Она дает возможность учитывать необходимые компоненты освещения - диффузную и отражающую составляющую отражения, а также рассеянное освещение. При этом она позволяет получить более реалистичное изображение, чем в модели Фонга, которая будет реализована как дополнение.

2. Конструкторская часть

В данном разделе приводятся требования к функционалу программного обеспечения. Затем подробнее рассматриваются выбранные в предыдущем разделе решения и нужные в них вспомогательные расчеты. Также приводятся способ ускорения работы программы и диаграмма классов.

2.1. Требования к программе

Программа должна предоставлять следующие возможности:

1) Визуализация динамической сцены, состоящей из описанных в пункте 1.2 объектов.
2) Запуск, останов флюгера, изменение направления, скорости его вращения.
3) Изменение материалов, из которых изготовлены части флюгера.
4) Изменение положения камеры и ее поворот.
5) Изменение параметров источников освещения и добавление новых источников.

2.2. Модели освещения

Модель Блинна-Фонга выбрана в качестве основной для расчета интенсивности в точке, а модель Фонга – для сравнения. Алгоритм FindIntensity расчета интенсивности I освещения в точке поверхности P с нормалью N, вектором взгляда V, степенью блеска s и коэффициентом ее увеличения coef в зависимости от выбранной модели освещения model, представлен на рисунке 2.1:

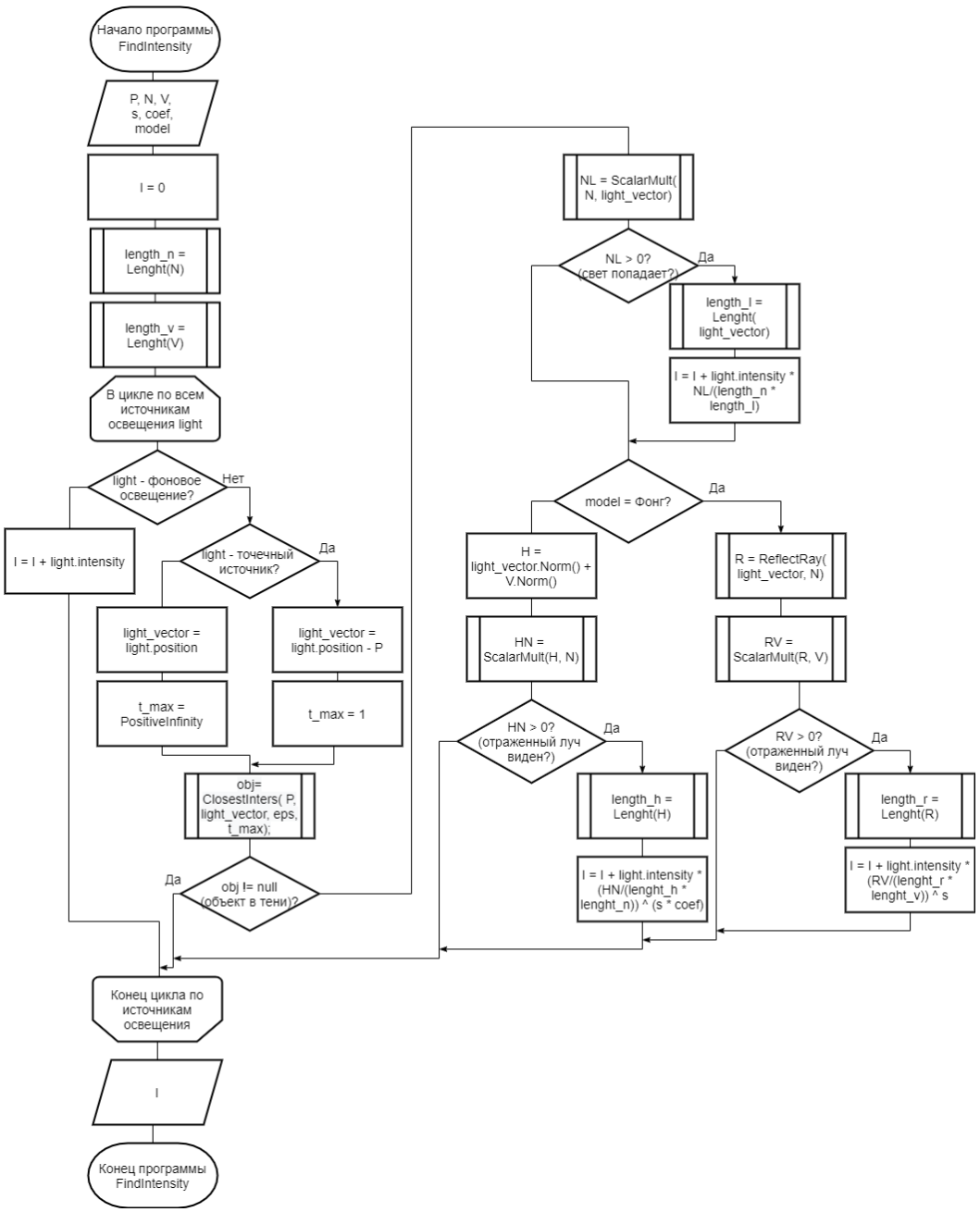

Рисунок 2.1. Схема алгоритма расчета интенсивности.

2.3. Алгоритм обратной трассировки лучей

В алгоритме обратной трассировки лучей определяется цвет каждого пиксела экрана, независимо от других пикселей. На рисунке 2.2 представлена схема алгоритма TraceRay обратной трассировки одного луча, испущенного из camera_position в направлении view_vector и ограниченного t_min и t_max, при максимальной глубине рекурсии depth.

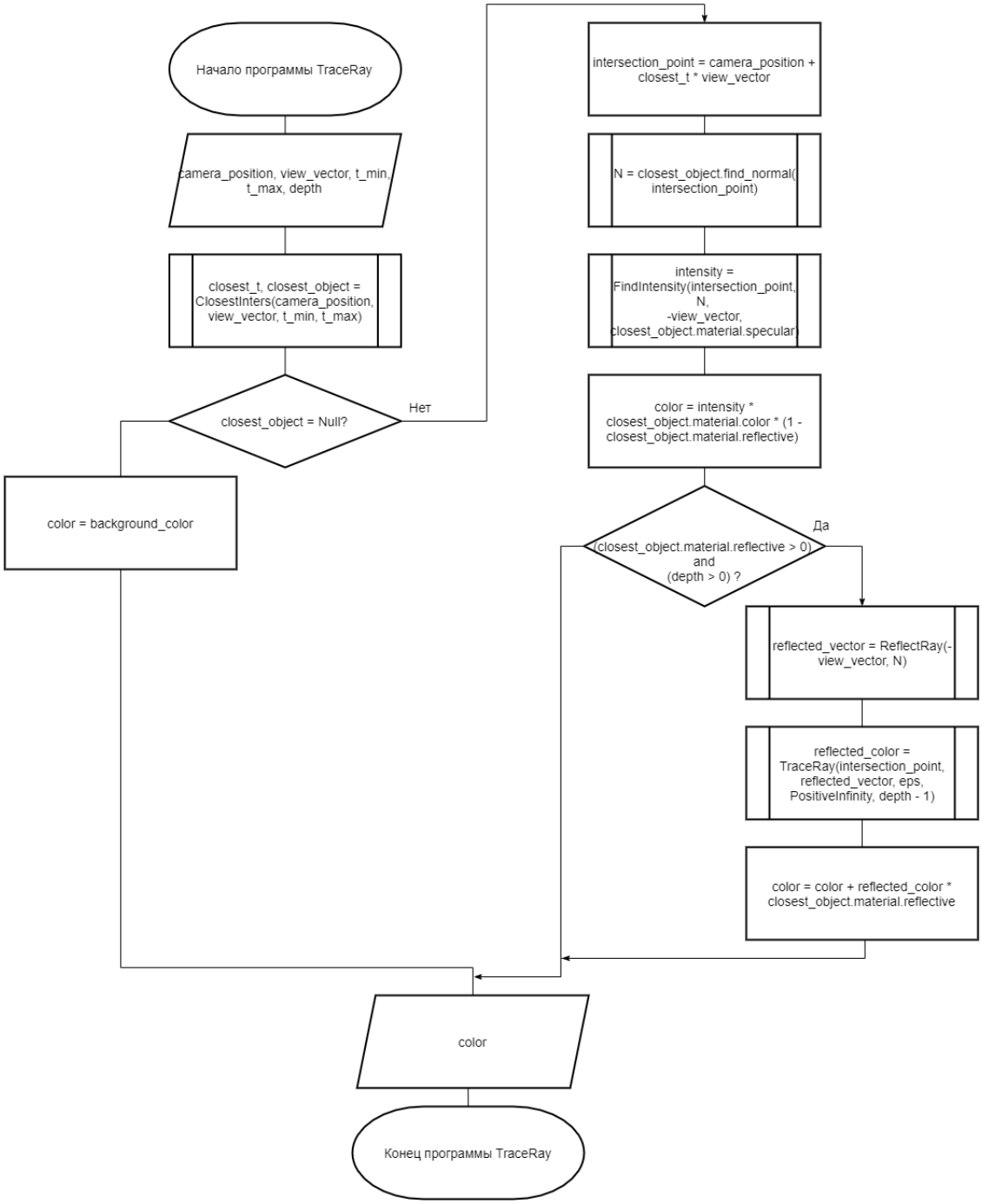

Рисунок 2.2. Схема алгоритма обратной трассировки луча.

2.4. Нахождение отраженного луча

Чтобы воспользоваться алгоритмом трассировки лучей, необходимо уметь находить направление отраженного луча.

По закону отражения вектор падающего света, нормаль к поверхности и вектор отражения лежат в одной плоскости, а угол падения равен углу

отражения. Пусть $\vec{L}$ – вектор падающего света, $\vec{R}$ – вектор отражения, $\vec{N}$- нормаль к поверхности (все векторы единичные), t – угол падения, как показано на рисунке 2.3.

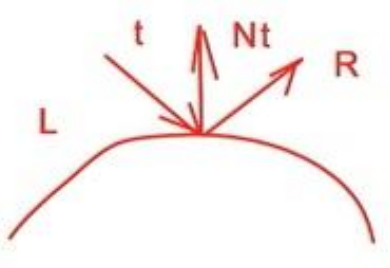

Рисунок 2.3. Определение направления отраженного луча.

Вектор $\vec{R}$ представляется суммой векторов $\vec{L}$ и $\vec{N}$ с некоторыми коэффициентами α и β.

$$\vec{R} = \alpha\vec{L} + \beta\vec{N} \tag{2.1}$$

Далее составляется и решается система из двух уравнений и находятся коэффициенты. Из условия, что угол падения равен углу отражения:

$$\cos(t) = \left(-\vec{L}, \vec{N}\right) = \left(\vec{N}, \vec{R}\right) \tag{2.2}$$

Подставляя выражение для $\vec{R}$ из 2.1 в 2.2, получаем выражение для β:

$$\vec{N}\left(\alpha\vec{L} + \beta\vec{N}\right) = \alpha(-\cos(t)) + \beta = \cos(t)$$

$$\beta = \cos(t)\,(1 + \alpha) \tag{2.3}$$

Скалярно умножим единичный вектор $\vec{R}$ на себя :

$$\left(\alpha\vec{L} + \beta\vec{N}\right)^2 = \alpha^2 + 2\alpha\beta(-\cos(t)) + \beta^2 = 1 \tag{2.4}$$

Подставляя 2.3 в 2.4, после преобразований, получаем выражение для α^2:

$$\alpha^2 - 2\alpha\,(\cos(t))^2\,(1 + \alpha) + (\cos(t))^2\,(1 + \alpha)^2 = 1$$

$$\alpha^2 + (\cos(t))^2\,(1 - \alpha^2) = 1$$

$$\alpha^2(1 - (\cos(t))^2) = 1 - (\cos(t))^2 \tag{2.5}$$

Откуда находим два значения α: 1 или -1. Второй корень не подходит из физических соображений. Итак,

$$\alpha=1 \tag{2.6}$$

Подставляя 2.6 в 2.3, находим β:

$$\beta = 2\cos(t) \tag{2.7}$$

Подставляя значения α и β в выражение для отраженного луча (2.1), находим вектор $\vec{R}$:

$$\vec{R} = \vec{L} + 2\cos(t)\,\vec{N} \tag{2.8}$$

Представляя cos(t) через скалярное произведение единичных векторов $\vec{N}$ и $\vec{L}$, окончательно получаем:

$$\vec{R} = \vec{L} + 2\left(-\vec{L}, \vec{N}\right)\vec{N} \tag{2.9}$$

2.5.Пересечение луча с объектами сцены

Чтобы воспользоваться алгоритмом трассировки лучей, также нужно уметь находить пересечение луча с объектами сцены: сферой, плоскостью, цилиндром, прямоугольным параллелепипедом и четырехугольной пирамидой.

Для этого луч представляют с помощью параметрического уравнения. Пусть O – начало луча (позиция камеры), V-текущий пиксель на экране, $\vec{D} = V - O$ – вектор, показывающий направление луча. Тогда любую точку P луча можно представить так:

$$P(t) = O + t\vec{D}, t \geq 0 \tag{2.10}$$

Так как точка пересечения луча и объекта сцены принадлежит и первому, и второму, она должна удовлетворять двум уравнениям: уравнению луча и уравнению, которое задает поверхность объекта сцены. Поэтому поиск точки пересечения сводится к составлению системы перечисленных уравнений и поиску ее решения.

При этом, если для найденного параметра не выполняется условие $t \geq 0$, значит пересечение принадлежит продолжению луча за камеру и оно отбрасывается. Далее в формулах оно будет опущено, но подразумевается.

2.5.1. Сфера

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

$$|\overrightarrow{P - C}| = r \text{, или } (\overrightarrow{P - C}, \overrightarrow{P - C}) = r^2 \qquad (2.11)$$

Имея уравнения, описывающие точки сферы (2.11) и точки луча (2.10), решаем систему, подставляя второе уравнение в первое:

$$(\overrightarrow{O + t\vec{D} - C}, \overrightarrow{O + t\vec{D} - C}) = r^2 \qquad (2.12)$$

Разложим скалярное произведение и преобразуем его. В результате получим квадратное уравнение:

$$(\vec{D}, \vec{D})t^2 + 2(\overrightarrow{CO}, \vec{D})t + (\overrightarrow{CO}, \overrightarrow{CO}) - r^2 = 0 \qquad (2.13)$$

Остается решить уравнение 2.13 чтобы найти параметры t точек пересечения. При этом, если у уравнения нет решений, значит луч не пересекается со сферой (рис. 2.4, слева), если решение одно, то луч касается сферы (рис. 2.4, в центре), два - луч входит в сферу и выходит из неё, и нужно выбрать наименьшее решение – где луч входит в сферу (рис. 2.4, справа).

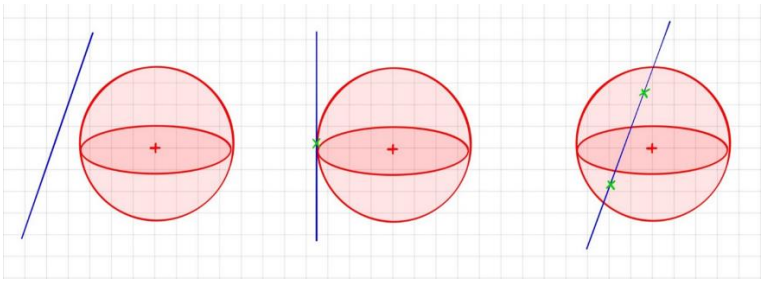

Рисунок 2.4. Взаимное расположение луча и сферы.

2.5.2. Плоскость

Плоскость можно задать с помощью вектора нормали к ней $\vec{V}$ и одной точки C, принадлежащей этой плоскости. Вектор из точки C до любой другой точки на плоскости P перпендикулярен нормали $\vec{V}$:

$$\left(\vec{V}, \overrightarrow{P - C}\right) = 0 \tag{2.14}$$

Имея уравнения, описывающие точки плоскости (2.14) и точки луча (2.10), решаем систему, подставляя второе уравнение в первое:

$$\left(\vec{V}, \overrightarrow{O + t\vec{D} - C}\right) = 0 \tag{2.15}$$

После преобразований получаем линейное уравнение:

$$t\left(\vec{V}, \vec{D}\right) - \left(\vec{V}, \overrightarrow{OC}\right) = 0 \tag{2.16}$$

Остается решить уравнение 2.16, чтобы найти параметр t точки пересечения:

$$t = \frac{\left(\vec{V}, \overrightarrow{OC}\right)}{\left(\vec{V}, \vec{D}\right)}, \left(\vec{V}, \vec{D}\right) \neq 0 \tag{2.17}$$

Если $\left(\vec{V}, \vec{D}\right) = 0$, значит векторы направления луча и нормали к плоскости перпендикулярны, и, следовательно, вектор направления луча параллелен самой плоскости и не пересекает ее.

2.5.3. Цилиндр

Цилиндр можно задать с помощью следующих параметров: радиус r, центр основания C, единичный вектор оси $\vec{V}$, высота H. Поверхность цилиндра условно делится на 2 части: боковая поверхность и два круга-основания.

Чтобы найти пересечение луча и оснований цилиндра достаточно найти пересечение луча с плоскостями $\left(C, -\vec{V}\right)$ и $\left(C + H\vec{V}, \vec{V}\right)$, которые содержат основания, по алгоритму из предыдущего пункта, а затем проверить, что найденные точки находятся на расстоянии не более r от оси цилиндра.

В этом пункте рассмотрим пересечение луча с боковой поверхностью цилиндра. Пусть P – точка пересечения луча с боковой поверхностью, h – расстояние от плоскости основания до этой точки, M – точка на оси цилиндра на том же расстоянии h от основания:

$$\mathrm{M} = \mathrm{C} + \mathrm{h}\vec{\mathrm{V}} \tag{2.18}$$

Тогда вектор $\overrightarrow{\mathrm{MP}}$ параллелен основанию и, соответственно, перпендикулярен вектору оси цилиндра:

$$\left(\overrightarrow{\mathrm{MP}}, \vec{\mathrm{V}}\right) = 0 \tag{2.19}$$

Выразим вектор $\overrightarrow{\mathrm{MP}}$ через уравнения для M (2.18) и P (2.10):

$$\overrightarrow{\mathrm{MP}} = \overrightarrow{\mathrm{P} - \mathrm{M}} = \overrightarrow{\mathrm{O} + \mathrm{t}\vec{\mathrm{D}} - \mathrm{C} - \mathrm{h}\vec{\mathrm{V}}} = \overrightarrow{\mathrm{CO}} + \mathrm{t}\vec{\mathrm{D}} - \mathrm{h}\vec{\mathrm{V}} \tag{2.20}$$

Подставляя 2.20 в 2.19, имеем:

$$\left(\overrightarrow{\mathrm{CO}} + \mathrm{t}\vec{\mathrm{D}} - \mathrm{h}\vec{\mathrm{V}}, \vec{\mathrm{V}}\right) = 0$$

$$\left(\overrightarrow{\mathrm{CO}} + \mathrm{t}\vec{\mathrm{D}}, \vec{\mathrm{V}}\right) = \mathrm{h}\left(\vec{\mathrm{V}}, \vec{\mathrm{V}}\right) = \mathrm{h}$$

$$\mathrm{h} = \left(\overrightarrow{\mathrm{CO}}, \vec{\mathrm{V}}\right) + \left(\vec{\mathrm{D}}, \vec{\mathrm{V}}\right)\mathrm{t} \tag{2.21}$$

Длина вектора $\overrightarrow{\mathrm{MP}}$ равна радиусу цилиндра:

$$\left(\overrightarrow{\mathrm{MP}}, \overrightarrow{\mathrm{MP}}\right) = \mathrm{r}^2$$

$$\left(\overrightarrow{\mathrm{CO}} + \mathrm{t}\vec{\mathrm{D}} - \mathrm{h}\vec{\mathrm{V}}, \overrightarrow{\mathrm{CO}} + \mathrm{t}\vec{\mathrm{D}} - \mathrm{h}\vec{\mathrm{V}}\right) = \mathrm{r}^2$$

$$(\overrightarrow{\mathrm{CO}}, \overrightarrow{\mathrm{CO}}) + \mathrm{t}^2\left(\vec{\mathrm{D}}, \vec{\mathrm{D}}\right) + \mathrm{h}^2 + 2\mathrm{t}\left(\overrightarrow{\mathrm{CO}}, \vec{\mathrm{D}}\right) - 2\mathrm{h}(\overrightarrow{\mathrm{CO}}, \vec{\mathrm{V}}) - 2\mathrm{ht}(\vec{\mathrm{D}}, \vec{\mathrm{V}}) = \mathrm{r}^2 \tag{2.22}$$

Подставляя в последнее уравнение выражение для h (2.21), получаем квадратное уравнение:

$$\mathrm{t}^2\left[\left(\vec{\mathrm{D}}, \vec{\mathrm{D}}\right) - \left(\vec{\mathrm{D}}, \vec{\mathrm{V}}\right)^2\right] + 2\mathrm{t}\left[\left(\overrightarrow{\mathrm{CO}}, \vec{\mathrm{D}}\right) - \left(\overrightarrow{\mathrm{CO}}, \vec{\mathrm{V}}\right)\left(\vec{\mathrm{D}}, \vec{\mathrm{V}}\right)\right] + \left[\left(\overrightarrow{\mathrm{CO}}, \overrightarrow{\mathrm{CO}}\right) -\right.$$

$$\left. -\left(\overrightarrow{\mathrm{CO}}, \vec{\mathrm{V}}\right)^2 - \mathrm{r}^2\right] = 0 \tag{2.23}$$

Остается решить это уравнение и проверить, что каждая найденная точка находится в плоскости (параллельной плоскости основания) на расстоянии не более H от плоскости основания. Рассуждения о геометрическом смысле количества корней аналогичны случаю со сферой.

2.5.4. Прямоугольный параллелепипед и четырехугольная пирамида.

Прямоугольный параллелепипед и четырехугольную пирамиду удобно представить с помощью группы треугольников: 12 для параллелепипеда (по 2 на каждую грань) и 6 для пирамиды (по одному на каждую боковую грань и 2 на четырехугольное основание). Тогда задача поиска точки пересечения луча и одного из этих примитивов сводится к поиску точки пересечения луча и треугольников.

Пусть треугольник задан тремя своими вершинами V0, V1, V2, не лежащими на одной прямой. Точки на треугольника P можно задать с помощью барицентрических координат (u, v):

$$P(u,v) = (1-u-v)V0 + uV1 + uV2, u \geq 0, v \geq 0, u+v \leq 1 \qquad (2.24)$$

Подставим в это уравнение выражение для точек луча P (2.10)

$$O + t\vec{D} = (1-u-v)V0 + uV1 + uV2 \qquad (2.25)$$

После перегруппировки членов получаем систему линейных уравнений:

$$[-\vec{D} \quad \overrightarrow{V1-V0} \quad \overrightarrow{V2-V0}] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = \overrightarrow{O-V0} \qquad (2.26)$$

Обозначим $\overrightarrow{E1} = \overrightarrow{V1-V0}, \overrightarrow{E2} = \overrightarrow{V2-V0}, \vec{T} = \overrightarrow{O-V0}$, и решим систему по правилу Крамера:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{|[-\vec{D} \quad \overrightarrow{E1} \quad \overrightarrow{E2}]|} \begin{bmatrix} |\vec{T} & \overrightarrow{E1} & \overrightarrow{E2}| \\ |-\vec{D} & \vec{T} & \overrightarrow{E2}| \\ |-\vec{D} & \overrightarrow{E1} & \vec{T}| \end{bmatrix} \qquad (2.27)$$

Используя свойство смешанного произведения:

$$|[\vec{A} \quad \vec{B} \quad \vec{C}]| = -(\vec{A} \times \vec{C})\vec{B} = -(\vec{C} \times \vec{B})\vec{A},$$

перепишем систему так:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(\vec{D} \times \overrightarrow{E2})\overrightarrow{E1}} \begin{bmatrix} (\vec{T} \times \overrightarrow{E1})\overrightarrow{E2} \\ (\vec{D} \times \overrightarrow{E2})\vec{T} \\ (\vec{T} \times \overrightarrow{E1})\vec{D} \end{bmatrix} \tag{2.28}$$

Обозначим $\vec{P} = (\vec{D} \times \overrightarrow{E2}), \vec{Q} = \vec{T} \times \overrightarrow{E1}$ и перепишем систему:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(\vec{P}, \overrightarrow{E1})} \begin{bmatrix} (\vec{Q}, \overrightarrow{E2}) \\ (\vec{P}, \vec{T}) \\ (\vec{Q}, \vec{D}) \end{bmatrix} \tag{2.29}$$

Остается найти параметры t, u и v и проверить выполнение для них соответствующих условий. Если хотя бы одно из них не выполняется, значит пересечения нет [4].

2.6. Уменьшение времени работы алгоритма.

Поскольку алгоритм обратной трассировки лучей обрабатывает каждый пиксель экрана независимо, можно использовать параллельные вычисления для уменьшения времени его работы, разбив экран на некоторые части. Наиболее часто используется горизонтальное или вертикальное разбиение [5].

Также можно ускорить программу, приняв во внимание тот факт, что у всех точек плоскости или треугольника (как части плоскости) нормали совпадают, поэтому достаточно вычислить их один раз перед началом трассировки, чтобы не повторять эту процедуру каждый раз.

2.7. Выбор используемых типов и структур данных

На рисунке 2.5 приведена диаграмма классов программы.

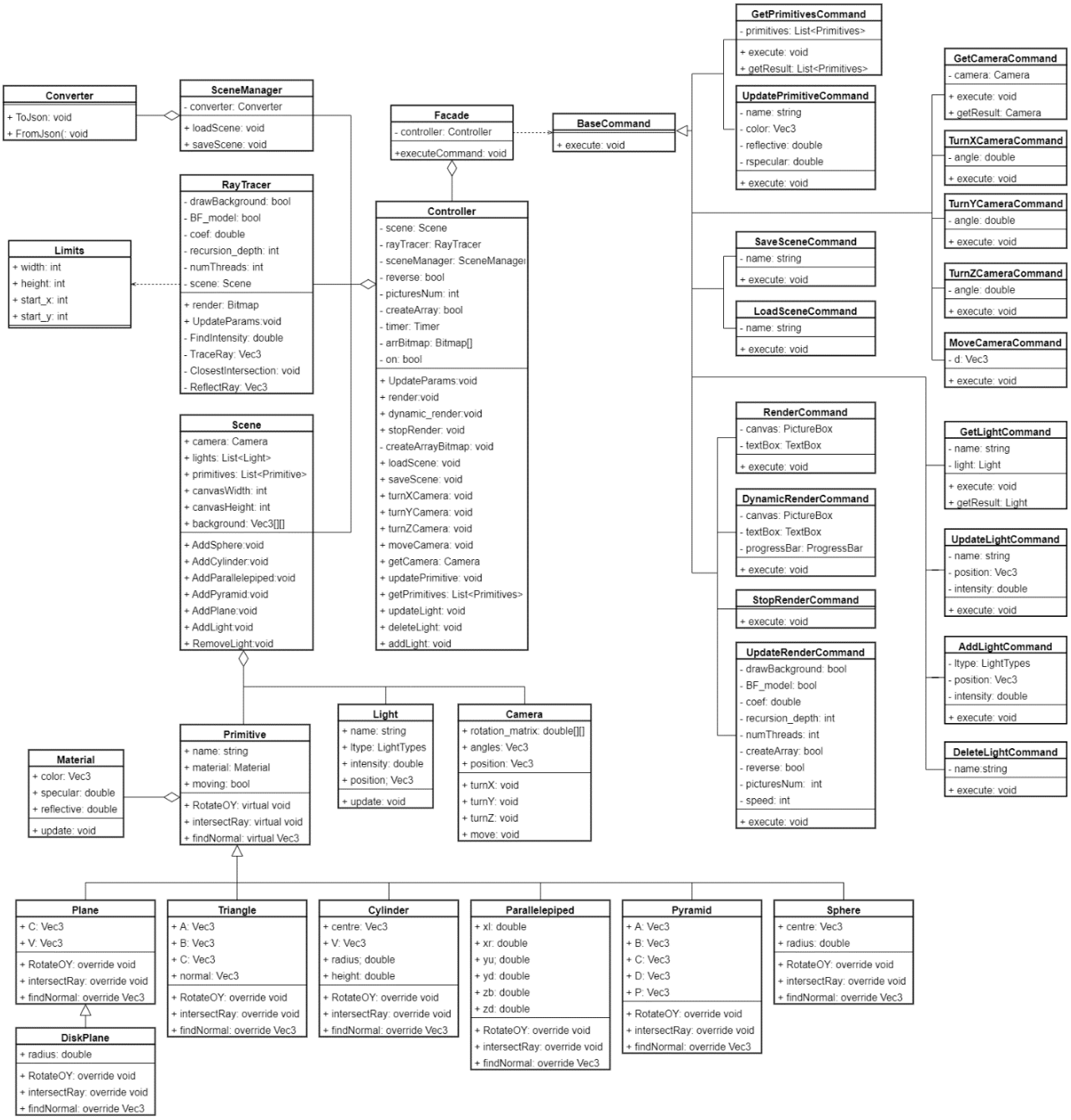

Рисунок 2.5. Диаграмма классов программы.

Вывод

В данном разделе были подробно рассмотрены выбранные решения - алгоритм обратной трассировки лучей и вычисление интенсивности света в точке по моделям Фонга и Блинна-Фонга, приведены соответствующие блок-схемы и расчеты, а также была представлена диаграмма классов.

3. Технологическая часть

В данном разделе производится выбор средств реализации, описывается интерфейс программы и нетривиальные моменты реализации.

3.1. Выбор средств реализации

Для решения поставленной задачи был выбран язык программирования C#, так как он:

1) поддерживает объектно-ориентированный подход к программированию, который дает возможность создавать четко структурированные и легко модифицируемые программы;
2) является строго типизированным, что позволяет защититься от многих ошибок;
3) является нативным и дает возможность легко использовать распараллеливание для уменьшения времени работы алгоритмов;
4) предоставляет широкий набор стандартных библиотек и шаблонов, что позволяет эффективно использовать ресурсы системы.

Для замеров времени используется предоставляемый класс Stopwatch, а для организации распараллеливания - пространство имен System.Threading, которое содержит в себе классы, поддерживающие многопоточное программирование, и самостоятельно реализованный вспомогательный класс Limits.

В качестве среды разработки выбрана “Visual Studio 2019” так как она:

1) позволяет создавать классические приложения для компьютера под управлением операционной системы Windows;
2) предоставляет умную проверку кода и его автоматический рефакторинг, что делает возможным быстрое выявление и исправление ошибок.

3.2. Описание основных моментов программной реализации

На листинге 3.1. представлен код алгоритма TraceRay обратной трассировки луча, испущенного из camera_position в направлении view_vector и ограниченного t_min и t_max, при максимальной глубине рекурсии depth.

Листинг 3.1. Код алгоритма трассировки луча TraceRay

```
private Vec3 TraceRay(Vec3 camera_position, Vec3 view_vector, double t_min, double t_max, int
depth, int x, int y)
{
    double closest_t = Double.PositiveInfinity;
    Primitive closest_object = null;

    ClosestIntersection(ref closest_object, ref closest_t, camera_position, view_vector,
t_min, t_max);

    if (closest_object == null)
    {
        if (drawBackground)
            return scene.background[scene.canvasWidth / 2, scene.canvasHeight / 2];
        else
            return new Vec3(0, 0, 0);
    }

    Vec3 intersection_point = camera_position + closest_t * view_vector;
    Vec3 N = closest_object.findNormal(intersection_point).Normalize();

    double intensity = FindIntensity(intersection_point, N, -view_vector,
closest_object.material.specular);
    Vec3 currentColor = intensity * closest_object.material.color * (1 -
closest_object.material.reflective);

    if (depth <= 0 || closest_object.material.reflective <= 0)
        return currentColor;

    Vec3 reflected_vector = ReflectRay(-view_vector, N);
    Vec3 reflectedColor = TraceRay(intersection_point, reflected_vector, 0.01,
Double.PositiveInfinity, depth - 1, x, y);

    currentColor += reflectedColor * closest_object.material.reflective;

    return currentColor;
}
```

На листинге 3.2 представлен код алгоритма FindIntensity, который вычисляет интенсивность intensity освещения в точке поверхности P с нормалью N в зависимости от вектора взгляда V и степени блеска материала specular. При этом зеркальная составляющая освещения рассчитывается либо по модели Фонга (если поле BF_model класса RayTracer установлено false), либо по модели Блинна-Фонга (если поле BF_model класса RayTracer установлено true, и тогда specular умножается на коэффициент, записанный в поле coef класса RayTracer).

Листинг 3.2. Код алгоритма расчета интенсивности FindIntensity.

```
private double FindIntensity(Vec3 P, Vec3 N, Vec3 V, double specular)
{
    double intensity = 0;
    double length_n = Vec3.Length(N);
    double length_v = Vec3.Length(V);
    foreach (var light in scene.lights)
    {
        // фоновая составляющая
        if (light.ltype == LightTypes.Ambient)
            intensity += light.intensity;
        else
        {
            Vec3 light_vector;
            double t_max;
            if (light.ltype == LightTypes.Point) // точечный источник
            {
                light_vector = light.position - P;
                t_max = 1.0;
            }
            else // направленный источник
            {
                light_vector = light.position;
                t_max = Double.PositiveInfinity;
            }
            // проверка на нахождение в тени
            double shadow_t = Double.PositiveInfinity;
            Primitive shadow_object = null;

            ClosestIntersection(ref shadow_object, ref shadow_t, P,
light_vector, 0.001, t_max);
            if (shadow_object != null)
                continue;
            // Диффузная составляющая
            double NL = Vec3.ScalarMultiplication(N, light_vector);
            if (NL > 0)
                intensity += light.intensity * NL / (length_n *
Vec3.Length(light_vector));
            // Зеркальная составляющая
            // по модели Фонга
            if (!BF_model)
            {
                Vec3 R = ReflectRay(light_vector, N);
                double RV = Vec3.ScalarMultiplication(R, V);
                if (RV > 0)
                    intensity += light.intensity * Math.Pow(RV / (Vec3.Length(R)
* length_v), specular);
            }
            // по модели Блинна-Фонга
            else
            {
                Vec3 H = (light_vector.Normalize() + V.Normalize());
                double HN = Vec3.ScalarMultiplication(H, N);
                if (HN > 0)
                    intensity += light.intensity * Math.Pow(HN / (Vec3.Length(H)
* length_n), specular * coef);
            }
        }
    }
    return intensity;
}
```

3.3.Интерфейс программы

Пользовательский интерфейс для управления программным обеспечением предоставлен на рисунке 3.1:

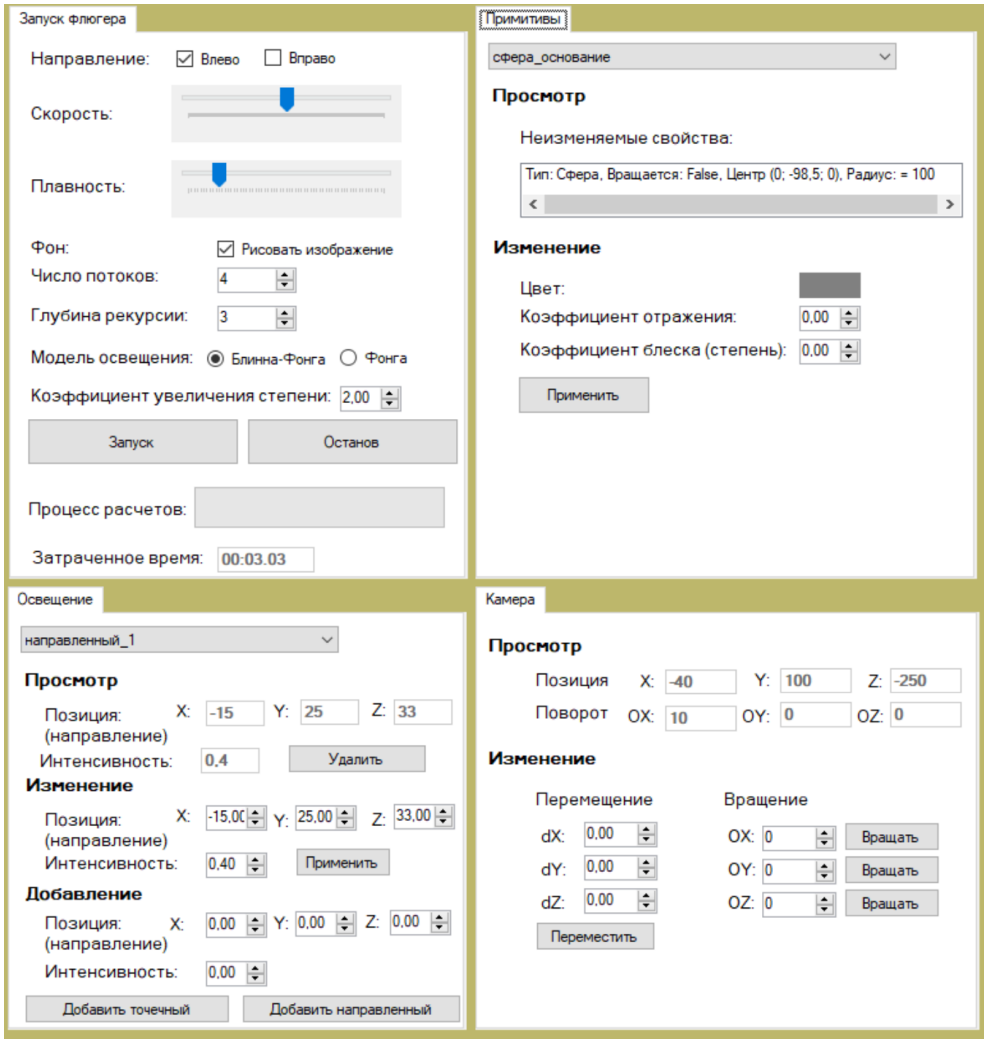

Рисунок 3.1. Пользовательский интерфейс управления программой.

Часть интерфейса, отвечающая за запуск флюгера, позволяет задать такие параметры, как: направление, скорость и плавность вращения флюгера; тип фона; количество потоков, используемых для расчетов; глубина рекурсии в алгоритме трассировки лучей; модель, по которой рассчитывается освещение, и коэффициент увеличения степени блеска в случае, если выбрана модель Блинна-Фонга. Здесь же можно запустить или остановить вращение флюгера, следить за процессом расчетов и затраченным на них временем.

Часть интерфейса, отвечающая за примитивы, позволяет просматривать текущие параметры примитивов (частей флюгера) и изменять параметры

материала, из которого они изготовлены: цвет, коэффициент отражения, степень блеска.

Часть интерфейса, отвечающая за освещение, позволяет добавлять и удалять источники освещения, а также изменять их параметры: для фонового – интенсивность, для точечного – позицию и интенсивность, для направленного – направление и интенсивность.

Часть интерфейса, отвечающая за камеру, позволяет просматривать и изменять положение камеры: ее позицию в пространстве и углы поворота относительно координатных осей.

Вывод

В данном разделе был обоснован выбор средств реализации, приведена структура программы и описан ее интерфейс, а также был продемонстрирован листинг основных моментов программной реализации.

4. Исследовательская часть

В данном разделе будут приведены результаты работы разработанного программного обеспечения, а также проведен эксперимент по сравнению времени работы алгоритма трассировки лучей при его последовательной и параллельной реализациях.

4.1. Результаты работы программного обеспечения

На рисунке 4.1 приведены результаты работы разработанного ПО, визуализирующего вращение флюгера с помощью метода обратной трассировки лучей (слева и справа показаны изображения флюгера в разные моменты вращения). В сцене используется фоновое освещение, а также по одному точечному и направленному источнику. Глубина рекурсии равна 3. Для расчета интенсивности в точке используется модель Блинна-Фонга. На рисунках видны эффекты отражения и отбрасывания теней, присущие объектам реального мира. Также видно, что при вращении изменяется положение не только примитивов, но и их теней.

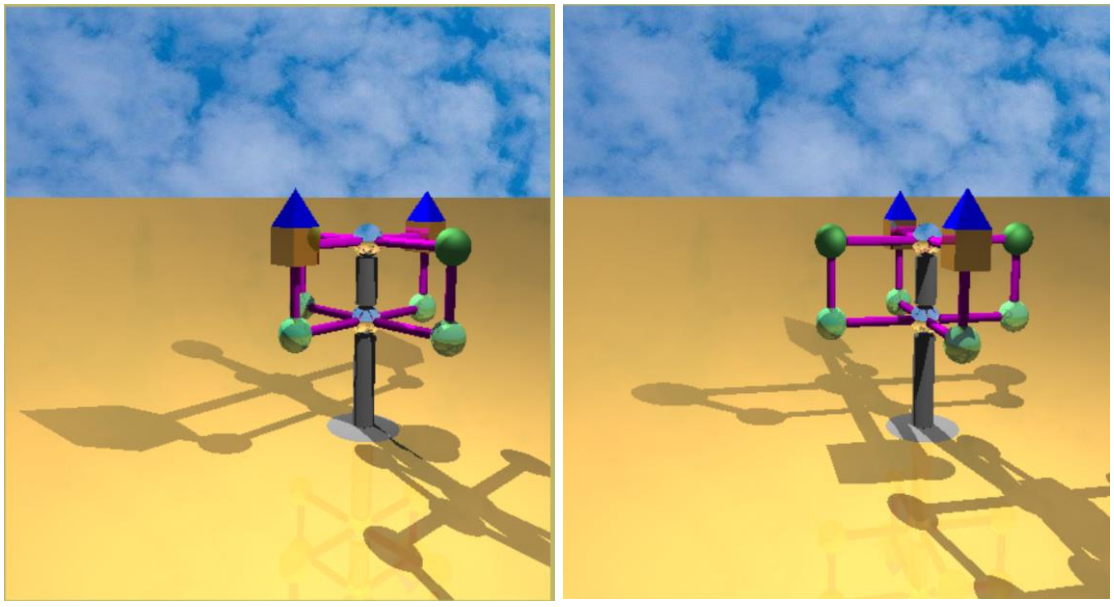

Рисунок 4.1. Результаты работы программы.

Далее будут приведены примеры результатов работы программы при изменении параметров.

На рисунке 4.2 приведено изображение флюгера с начальными параметрами материалов, камеры и источников освещения.

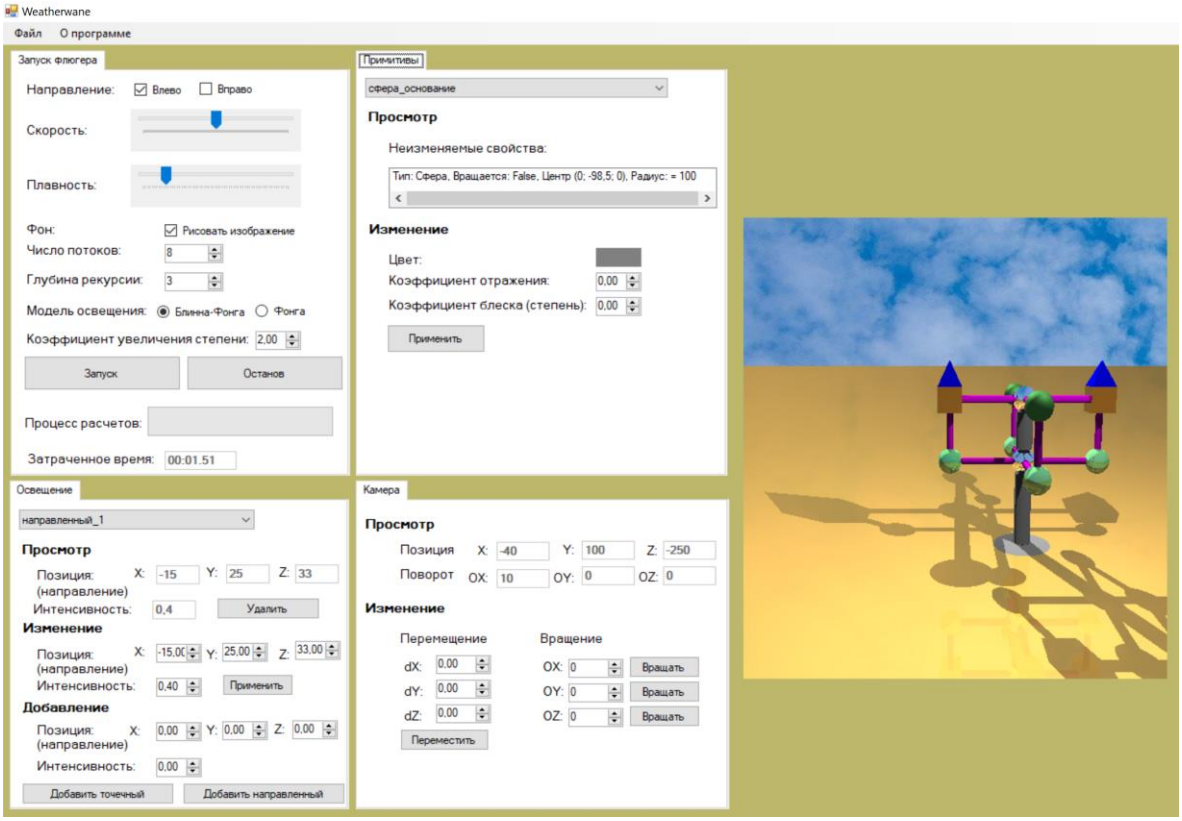

Рисунок 4.2. Изображение с начальными параметрами.

На рисунке 4.3 приведено изображение после изменения параметров материалов:

- для сферы верхнего уровня, наиболее близкой к наблюдателю изменен цвет (с зеленого на красный);
- для плоскости основания увеличен коэффициент блеска (с 1 до 2);
- для нижней правой сферы увеличен коэффициент отражения (с 0.3 до 0.8).

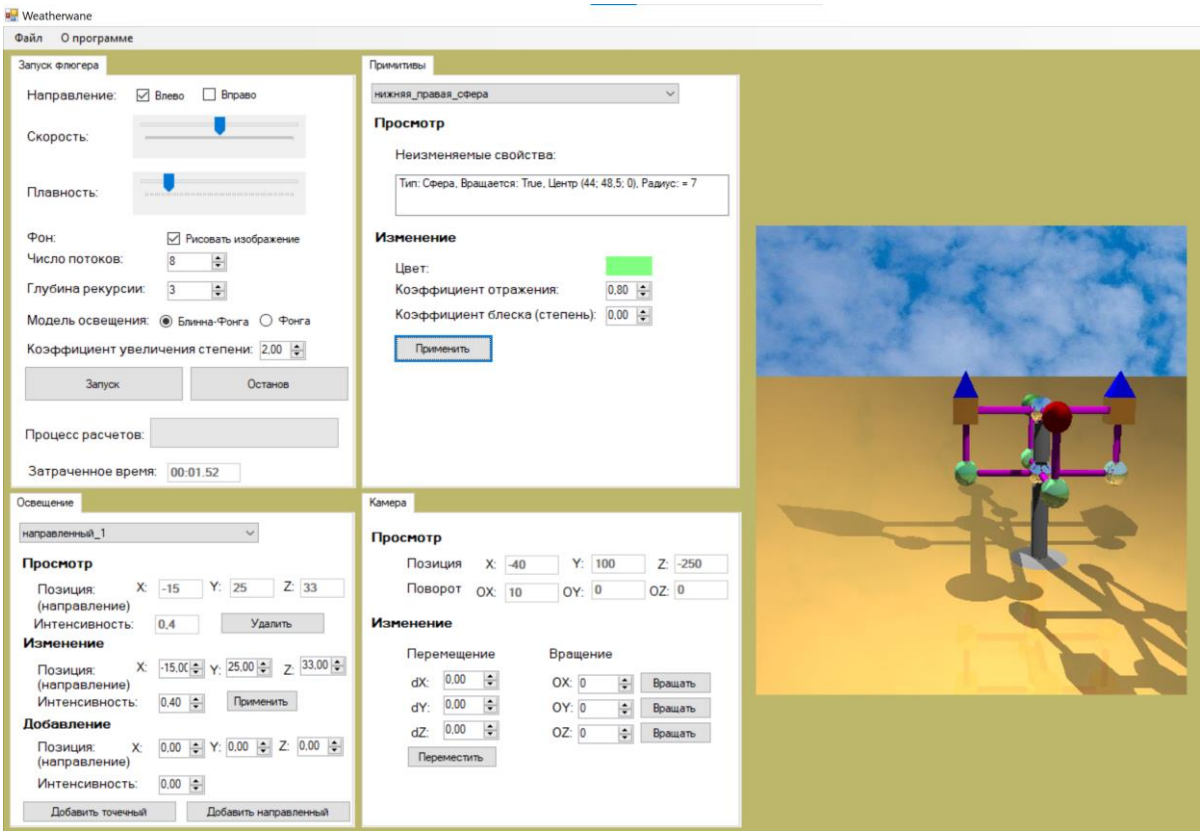

Рисунок 4.3. Изображение после изменения параметров материалов.

На рисунке 4.4 приведено изображение после изменения положения камеры: она была сначала перемещена на вектор (20, -20, 20), затем повернута на 5 градусов вокруг оси OX (направлена вправо) и на 10 градусов вокруг оси OZ (направлена вглубь изображения).

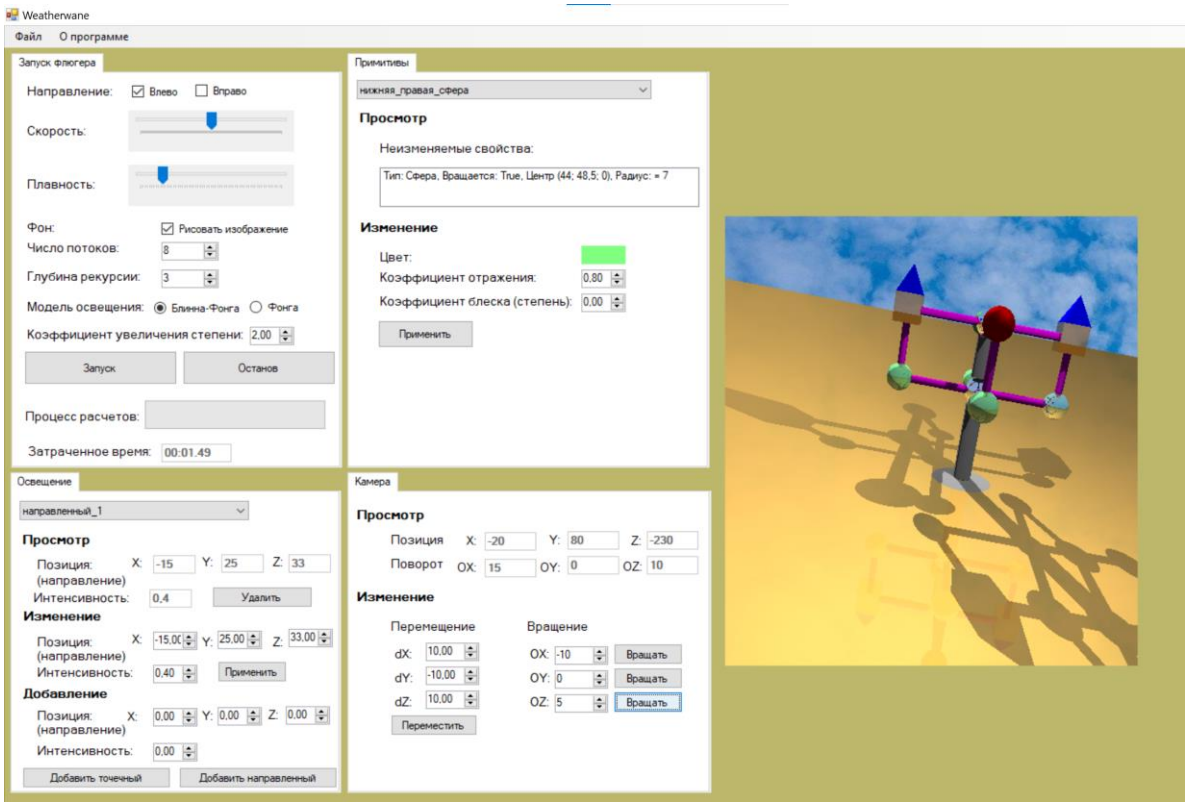

Рисунок 4.4. Изображение после изменения положения камеры.

На рисунке 4.5 приведено изображение после изменения параметров освещения: точечный источник был удален, интенсивность фонового освещения увеличена с 0.2 до 0.4, а направление направленного источника изменено с (-15, 25, 33) на (30, 20, 10).

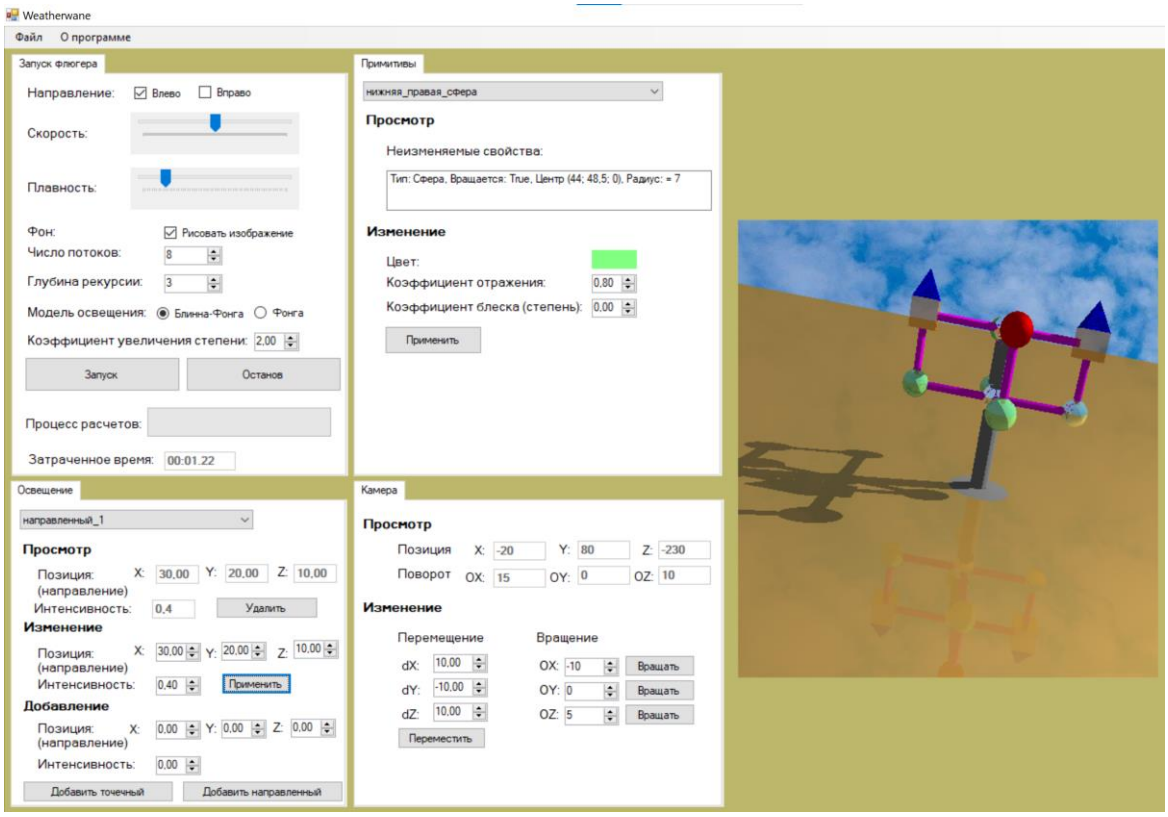

Рисунок 4.5. Изображение после изменения параметров освещения.

На рисунке 4.6 приведено изображение с теми же параметрами, что и в предыдущем случае, но при глубине рекурсии, равной 0. Из рисунка видно, что с изображения пропали эффекты отражения.

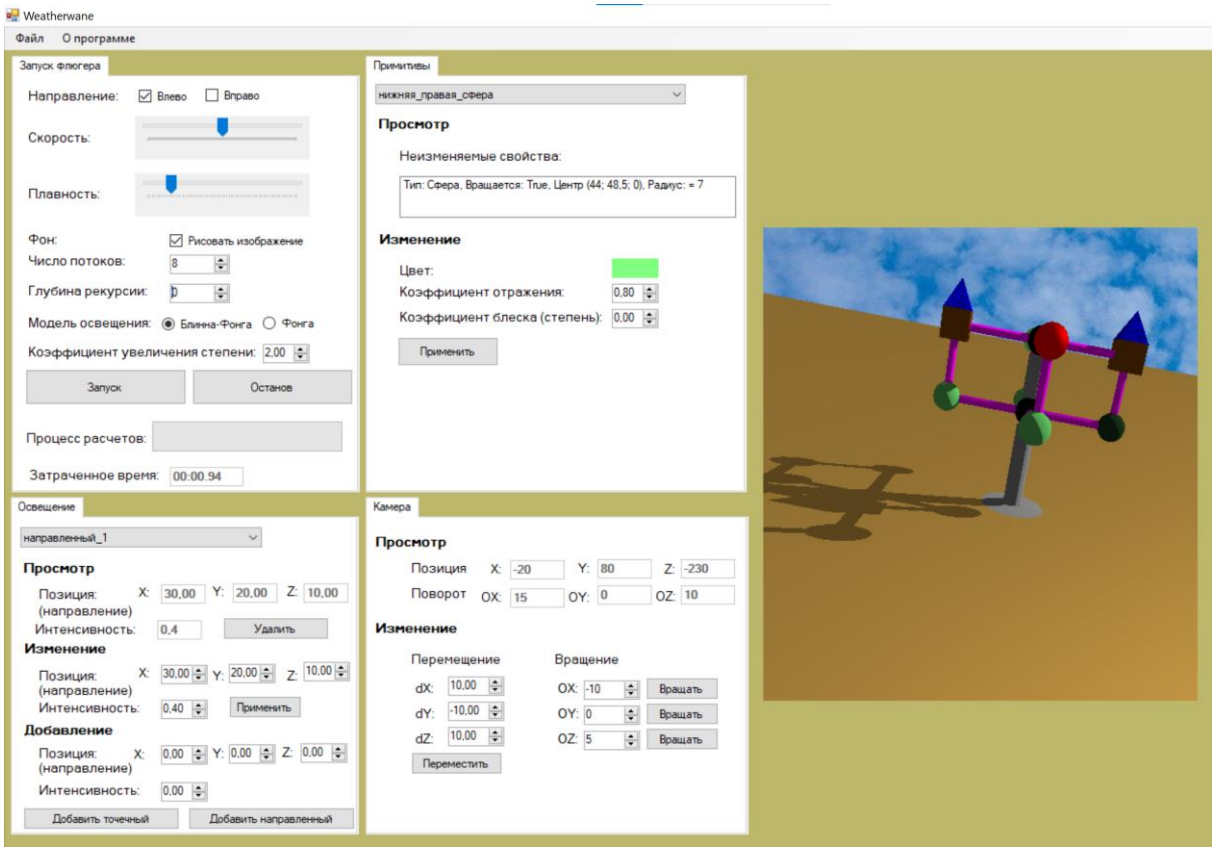

Рисунок 4.6. Изображение, построенное при нулевой глубине рекурсии.

4.2. Технические характеристики

Технические характеристики устройства, на котором выполнялись исследования:

- операционная система: Windows 10;
- оперативная память: 16 Гб;
- процессор: Intel® Core i5-8259U;
- количество ядер: 4;
- количество логических процессоров: 8.

Во время исследований ноутбук был включен в сеть питания и нагружен только встроенными приложениями окружения и системой тестирования.

4.3. Сравнение времени выполнения реализаций алгоритмов

Сравнивалось время работы последовательной и параллельной реализаций алгоритма обратной трассировки лучей, причем во втором случае сравнивалось также время работы реализации в зависимости от количества потоков (1, 2, 4, ..., (4*количество логических ядер) = 32).

Перечисленные реализации сравнивались по времени обработки сцены в зависимости от количества примитивов (прямоугольных параллелепипедов) в ней: от 5 до 35 с шагом 5. Во всех случаях использовалась модель Блинна-Фонга, глубина рекурсии была равна 3, в сцене присутствовало фоновое освещение, а также по одному точечному и направленному источнику.

Так как некоторые реализации выполняются достаточно быстро, а замеры времени имеют некоторую погрешность, они для каждой реализации и каждого количества элементов на сцене выполнялись 10 раз, а затем вычислялось среднее время работы. На рисунке 4.2 приведены результаты сравнения времени работы всех реализаций на всех данных (в легенде количество потоков указано как n_threads).

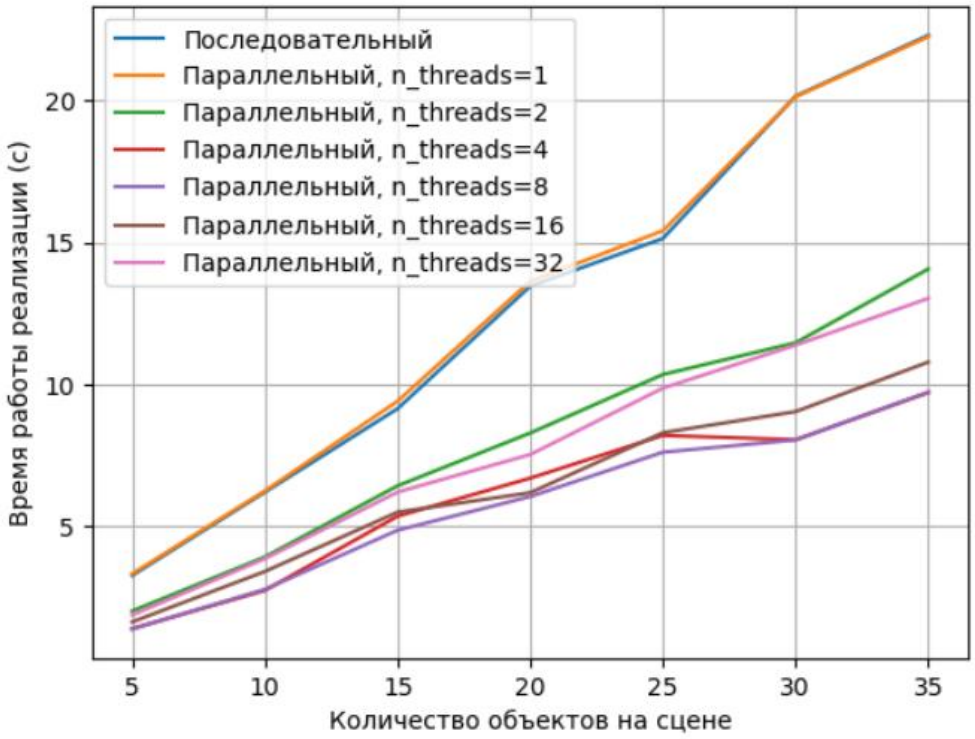

Рисунок 4.2. Сравнение времени работы реализаций в зависимости от количества примитивов на сцене.

Вывод

Как видно из рисунка 4.2, последовательная реализация и параллельная реализация с одним потоком, как и ожидалось, работают примерно одинаковое количество времени, при этом вторая немного дольше в связи с накладными расходами на создание потока. Эти две реализации затратили наибольшее количество времени из всех сравниваемых. Далее с ростом числа потоков время работы соответствующей параллельной реализации уменьшается, так как независимые вычисления производятся одновременно на разных ядрах. Это происходит вплоть до момента, когда используются 8 потоков, то есть их количество равно количеству логических ядер в компьютере.

При дальнейшем увеличении числа потоков время работы параллельной реализации больше, чем в описанной выше наилучшей точке, так как количество потоков становится больше количества логических ядер в компьютере, и, соответственно, некоторые из них вынуждены ожидать освобождения занятого другим потоком процессора, который смог бы провести необходимые вычисления. В результате теряется смысл в выделении этих потоков, так как одновременной обработки каждого из них не происходит, а дополнительное время на их организацию затрачивается. Поэтому рекомендуемым числом потоков можно назвать число, равное количеству логических процессоров.

Заключение

В результате выполнения данного курсового проекта было разработано программное обеспечение, позволяющее генерировать трехмерную визуализацию вращения флюгера.

В ходе выполнения работы были решены следующие задачи:

1. изучены и проанализированы существующие алгоритмы построения реалистичных изображений;
2. выбраны алгоритмы, наиболее подходящие для решения поставленной задачи;
3. спроектирована архитектура программного продукта и выбраны структуры данных для представления объектов синтезируемой сцены;
4. разработана программа на основе выбранных алгоритмов и структур данных;
5. проведено исследование на основе разработанной программы.

При проведении работы были получены знания в области компьютерной графики и параллельного программирования, были закреплены навыки проектирования программного обеспечения. Также удалось лучше изучить язык программирования C# и среду разработки “Visual Studio 2019”.

Список использованной литературы

1. Роджерс, Д. Математические основы машинной графики / Д. Роджерс, Дж. Адамс – М.: Мир, 1989. – 512с.
2. Боресков, А. В. Программирование компьютерной графики. Современный OpenGL / А. В. Боресков – М.: ДМК Пресс, 2019. – 372 с.
3. Свойства источника света и материала. Типы источников света. Суммарное освещение, Иван Андреев. [Электронный ресурс]. – Режим доступа: https://compgraphics.info/3D/lighting/light_sources.php (дата обращения: 22.07.2021)
4. Компьютерная графика. Практические занятия, лабораторный практикум, В. А. Ярошевич [Электронный ресурс]. – Режим доступа: http://miet.aha.ru/cg/cg_2015.pdf (дата обращения: 10.07.2021)
5. Параллельная обработка в алгоритме визуализации с трассировкой лучей. [Электронный ресурс]. – Режим доступа: https://cyberleninka.ru/article/n/parallelnaya-obrabotka-v-algoritmah-vizualizatsii-s-trassirovkoy-luchey (дата обращения: 18.07.2021)