Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»
Кафедра «Программное обеспечение ЭВМ и информационные технологии»

Р А С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к курсовому проекту по курсу:

«Компьютерная графика»

на тему:

«Программа для создания трехмерных сцен из трехмерных геометрических примитивов»

Студент _____ _____ И. В. Спасенов
(Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ _____ К.А. Кивва
(Подпись, дата) (И.О.Фамилия)

Москва, 2018 г.

Содержание

Введение

В наши дни в игровой и киноиндустрии повсеместно применяются визуальные эффекты. Специалисты стремятся максимально реалистично изобразить в своих продуктах построенные на компьютере изображения, поэтому сегодня всем тем, кто хочет понять принципы их создания, важно иметь инструменты для разработки собственных реалистичных трехмерных сцен, инструменты, которые позволили бы на первых этапах познакомится с особенностями компьютерной графики.

Целью проекта является разработка программы для создания трехмерных графических сцен из трехмерных геометрических примитивов, а также уже готовых моделей и их визуализация с учетом выбранной текстуры или цвета, а также оптических эффектов отражения, преломления, прозрачности, блеска.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ существующих алгоритмов удаления невидимых линий и поверхностей, закраски, текстурирования, а также моделей освещения и выбрать из них подходящие для выполнения проекта;
- произвести основные математические расчеты для реализации выбранных алгоритмов;
- выбрать подходящий язык программирования для реализации поставленной задачи;
- реализовать интерфейс программного модуля.

1. Аналитический раздел

В этом разделе проводится анализ существующих алгоритмов построения трехмерных изображений и выбираются наиболее подходящие алгоритмы для решения поставленных задач.

1.1 Алгоритмы построения трехмерного изображения

1.1.1 Алгоритм Робертса

Алгоритм Робертса - алгоритм получения изображений одиночных выпуклых объектов, составленных из плоских граней, работающий в объектном пространстве. Алгоритм Робертса может быть применен для изображения множества выпуклых многогранников на одной сцене в виде проволочной модели с удаленными невидимыми линиями [1]. Метод не пригоден непосредственно для передачи падающих теней и других сложных визуальных эффектов.

Многогранник, состоящий из N граней, описывается матрицей T размера 4×N, каждый n-й столбец которой содержит коэффициенты уравнения n-й плоскости:

$\mathrm{AnX} + \mathrm{BnY} + \mathrm{CnZ} + \mathrm{D} = 0$ в объектной системе координат XYZ:

$$T = \begin{bmatrix} A_1 & \dots & A_n & \dots & A_N \\ B_1 & \dots & B_n & \dots & B_N \\ C_1 & \dots & C_n & \dots & C_N \\ D_1 & \dots & D_n & \dots & D_N \end{bmatrix}.$$

Минимальное число граней N = 4 наблюдается у тетраэдра.

Важное требование к модели объекта в алгоритме Робертса заключается в достижении такой функции каждой плоскости:

$f(X, Y, Z) = \mathrm{AnX} + \mathrm{BnY} + \mathrm{CnZ} + \mathrm{Dn}$, при которой справедливо $f(X\text{вн}, Y\text{вн}, Z\text{вн}) \geq 0$ для любой точки $(X\text{вн}, Y\text{вн}, Z\text{вн})$, заведомо принадлежащей телу многогранника. Достижение этого условия осуществляется путем опытной проверки знака функции относительно внутренней точки, в качестве которой может выступать точка со средним

геометрическим положением относительно всех вершин многогранника. При достижении положительного знака функций всех граней автоматически достигается ориентация нормали $Nn = iAn + jBn + kCn$ к любой из граней внутрь фигуры.

Кроме параметров уравнений граней необходимо знать координаты вершин каждой грани. Вершины могут быть заданы или вычислены из матрицы T путем определения общих решений каждой плоскости со всеми остальными.

Для каждой n-й грани должна быть составлена матрица K, содержащая координаты всех вершин, принадлежащих этой грани:

$$V_n = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ X_K & Y_K & Z_K & 1 \end{bmatrix}.$$

Не все грани, входящие в состав объекта, будут видны наблюдателю. Невидимой будет та, для которой выполняется условие - $(\vec{R}, \vec{N}) < 0$, где $\vec{N}$ - нормаль к грани, $\vec{R}$ – вектор, совпадающий с направлением взгляда.

Эти грани исключаются из дальнейшего анализа. Все оставшиеся являются видимыми.

После этапа удаления не лицевых отрезков необходимо выяснить, существуют ли такие отрезки, которые экранируются другими телами в картинке или в сцене. Для этого каждый оставшийся отрезок или ребро нужно сравнить с другими телами сцены или картинки.

Возможны следующие случаи:

1. Грань ребра не закрывает. Ребро остается в списке ребер.
2. Грань полностью закрывает ребро. Ребро удаляется из списка рассматриваемых ребер.
3. Грань частично закрывает ребро. В этом случае ребро разбивается на несколько частей, видимыми из которых являются не более двух. Само ребро удаляется из списка рассматриваемых ребер, но в список

проверяемых ребер добавляются те его части, которые данной гранью не закрываются.

Достоинствами алгоритма являются скорость работы и простота реализации. Основным недостатком метода, определившим ограниченность его распространения, являются неспособность без привлечения других подходов реализовать падающие тени, невозможность передачи зеркальных эффектов и преломления света и, наконец, строгая ориентация метода только на выпуклые многогранники.

1.1.2 Алгоритм Варнока

В алгоритме Варнока и его вариантах делается попытка воспользоваться тем, что большие области изображения когерентны [2].

В пространстве изображения рассматривается окно и решается вопрос о том, пусто ли оно, или его содержимое достаточно просто для визуализации. Если это не так, то окно разбивается на фрагменты до тех пор, пока содержимое фрагмента не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения. В последнем случае информация, содержащаяся в окне, усредняется, и результат изображается с одинаковой интенсивностью или цветом. В оригинальной версии алгоритма каждое окно разбивалось на четыре одинаковых подокна. Многоугольник, входящий в изображаемую сцену, называют:

1. Внешним, если он целиком находится вне окна – Рисунок 1.1a;
2. Внутренним, если он целиком расположен внутри окна – Рисунок 1.1b;
3. Пересекающим, если он пересекает границу окна – Рисунок 1.1c;
4. Охватывающим, если окно целиком расположено внутри него – Рисунок 1.1d.

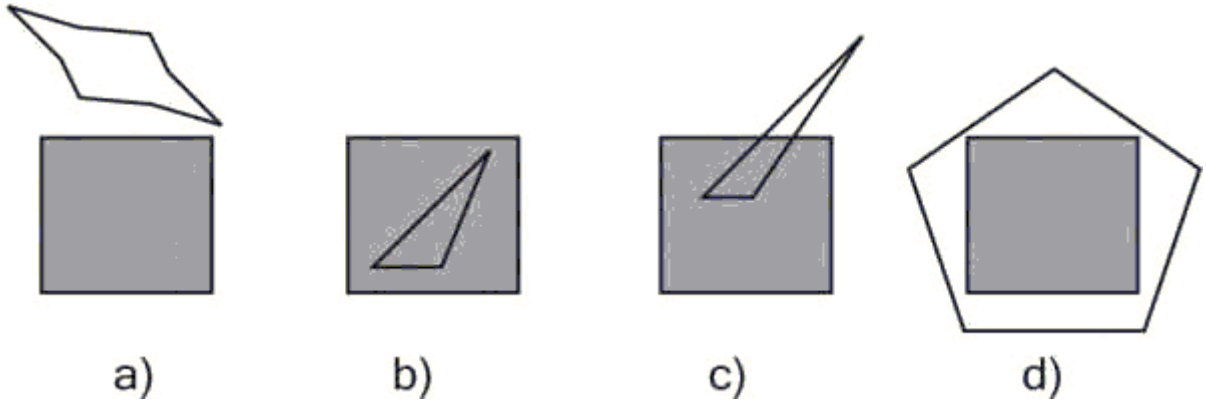

Рисунок 1.1. Варианты расположения многоугольника по отношению к окну

Для каждого окна:

1. Если все многоугольники сцены являются внешними по отношению к окну, то оно пусто; в этом случае окно закрашивается фоновым цветом и дальнейшему разбиению не подлежит.
1. Если только один многоугольник сцены имеет общие точки с окном и является по отношению к нему внутренним, то окно заполняется фоновым цветом, а сам многоугольник заполняется своим цветом.
2. Если только один многоугольник сцены имеет общие точки с окном и является по отношению к нему пересекающим, то окно заполняется фоновым цветом, а часть многоугольника, принадлежащая окну, заполняется цветом многоугольника.
3. Если только один многоугольник охватывает окно и нет других многоугольников, имеющих общие точки с окном, то окно заполняется цветом этого многоугольника.
4. Если существует хотя бы один многоугольник, охватывающий окно, то среди всех таких многоугольников выбирается тот, который расположен ближе всех многоугольников к точке наблюдения, и окно заполняется цветом этого многоугольника.
5. В противном случае производится новое разбиение окна.

Существуют различные реализации алгоритма Варнока. Были предложены варианты оптимизации, использующие предварительную

сортировку многоугольников по глубине, т. е. по расстоянию от точки наблюдения, и другие. Достоинством алгоритма Варнока является учет когерентности, из-за чего скорость его работы повышается при увеличении размеров однородных областей изображения. К недостаткам алгоритма можно отнести невозможность передачи зеркальных эффектов и преломления света, а также несовершенство способа разбиения изображения – в сложных сценах число разбиений может стать очень большим, что приведет к потере скорости.

1.1.3 Алгоритм Вейлера-Азертона

Алгоритм Вейлера-Азертона является попыткой минимизировать количество шагов в алгоритме Варнока путём разбиения окна вдоль границ многоугольника [3]. Метод работает с проекциями граней на картинную плоскость.

Алгоритм выполняет следующую последовательность действий:

1. Предварительная сортировка по глубине (для формирования списка приблизительных приоритетов).
2. Отсечение по границам ближайшего к наблюдателю многоугольника (в качестве отсекателя используется копия первого многоугольника из списка приблизительных приоритетов. Отсекаться будут все многоугольники в этом списке, включая первый. Формируется 2 списка – внутренний и внешний).
3. Удаление многоугольников внутреннего списка, которые экранируются отсекателем.
4. Если глубина многоугольника из внутреннего списка больше, чем Z_{min} отсекателя, то такой многоугольник частично экранирует отсекатель. Нужно рекурсивно разделить плоскость, используя многоугольник, нарушивший порядок, в качестве отсекателя (нужно использовать копию исходного многоугольника, а не остаток после предыдущего

отсечения). Отсечению подлежат все многоугольники из внутреннего списка.

5. По окончании отсечения или рекурсивного разбиения изображаются многоугольники из внутреннего списка (те, которые остались после удаления всех экранируемых на каждом шаге многоугольников – остаются только отсекающие многоугольники).
6. Работа продолжается с внешним списком (шаги 1-5).

Если многоугольники пересекаются, то для корректной работы данного алгоритма нужно плоскость одного разбить другим на две части. Эффективность алгоритма Вейлера-Азертона, как и алгоритма Варнока, зависит от эффективности разбиений. В дальнейшем этот алгоритм был распространен на сплайновые поверхности. К достоинствам алгоритма можно отнести скорость работы, учет когерентности изображения. Недостатками алгоритма является сложность реализации, а также невозможность передачи зеркальных эффектов и преломления света.

1.1.4 Алгоритм обратной трассировки лучей

Алгоритм обратной трассировки лучей выглядит следующим образом: из камеры через каждый пиксел изображения испускается луч и находится точка его пересечения с поверхностью сцены. Лучи, выпущенные из камеры, называют первичными. Пусть, первичный луч пересекает некий объект 1 в точке H1, как показано на Рисунке 1.2.

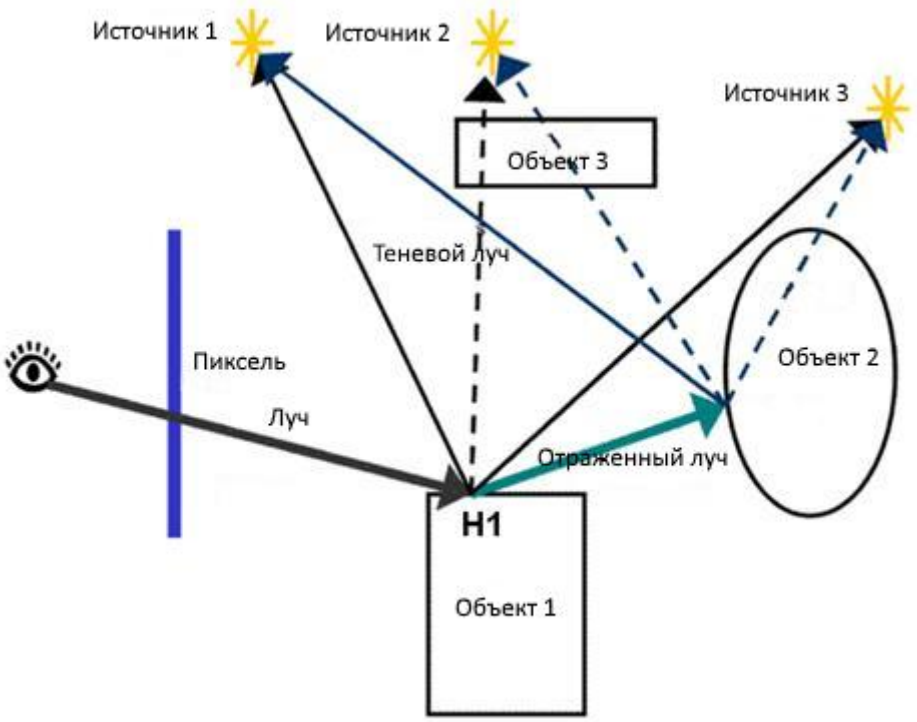

Рисунок 1.2. Алгоритм обратной трассировки лучей

Далее необходимо определить для каждого источника освещения, видна ли из него эта точка. Тогда в направлении каждого точечного источника света испускается теневой луч из точки H1. Это позволяет определить, освещается ли данная точка конкретным источником. Если теневой луч находит пересечение с другими объектами, расположенными ближе к точке H1, чем источник света, значит, точка H1 находится в тени от этого источника и освещать ее не надо. Иначе считаем освещение по некоторой локальной модели. Освещение со всех видимых (из точки H1) источников света складывается. Далее, если материал объекта 1 имеет отражающие свойства, из точки H1 испускается отраженный луч и для него вся процедура трассировки рекурсивно повторяется. Аналогичные действия должны быть выполнены, если материал имеет преломляющие свойства [3].

Техника рендеринга с трассировкой лучей отличается высоким реализмом получаемого изображения. Она позволяет изобразить гладкие объекты без аппроксимации их полигональными поверхностями, а также воссоздать тени, отражения и преломления света. Алгоритм трассировки позволяет параллельно и независимо трассировать два и более лучей,

разделять участки для трассирования на разных узлах кластера и т.д. Также присутствует отсечение невидимых поверхностей, перспектива.

Недостатком метода обратного трассирования является производительность. Трассировка лучей каждый раз начинает процесс определения цвета пикселя заново, рассматривая каждый луч наблюдения в отдельности [4].

1.1.5 Алгоритм удаления невидимых граней с использованием Z-буфера

Это алгоритм, работающий в пространстве изображения, как показано на Рисунке 1.3. Буфер кадра используется для запоминания интенсивности каждого пиксела в пространстве изображения, z-буфер — это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пиксела в пространстве изображения.

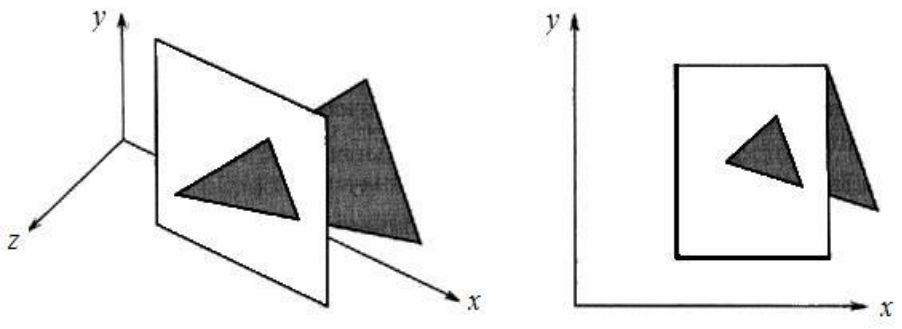

Рисунок 1.3. Пример алгоритма с использованием Z-буфера

Уравнение плоскости имеет вид:

$$Ax + By + Cz + D = 0$$

$$z = -\frac{Ax + By + D}{C}, \qquad C \neq 0$$

При $C = 0$ плоскость многоугольника параллельна оси Z.

У сканирующей строки $y = const$ и глубина пиксела на этой строке, у которого $x1 = x + \Delta x$, равна:

$$z1 - z = -\frac{ax1 + d}{c} + \frac{ax + d}{c} = \frac{a(x - x1)}{c}$$

Так как $\Delta x = 1$, то $z1 = z - \frac{A}{C}$

В процессе работы значение z каждого пиксела, который нужно занести в буфер кадра, сравнивается с глубиной уже занесенного в z-буфер пиксела. Если новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксел заносится в этот буфер и производится корректировка z-буфера новым значением z.

К достоинствам алгоритма, использующего z-буффер можно отнести простоту реализации, а также отсутствие предварительной сортировки элементов сцены. Недостатками алгоритма являются трудоёмкость реализации эффектов прозрачности, а также перерасход по памяти - алгоритм предполагает хранение двух двумерных массивов, размер которых увеличивается с увеличением размеров изображения.

1.1.6 Выводы

В программе будет предусмотрено два режима работы. В первом режиме пользователь сможет добавлять новые объекты, редактировать их размер, положение, выбирать цвет и текстуры, поэтому для обеспечения высокой скорости работы в качестве алгоритма отсечения невидимых линий и поверхностей, был выбран алгоритм z-буфера. Во втором режиме программа должна будет строить реалистичное изображение, учитывать тени, прозрачность, преломление, отражение объектов, поэтому в этом режиме будет использоваться алгоритм обратной трассировки лучей, как позволяющий достичь наибольшей реалистичности построенного изображения. Так как недостатком алгоритма обратной трассировки лучей является низкая производительность, целесообразно подумать о ее улучшении, например, реализовать его многопоточное выполнение.

1.2 Анализ алгоритмов наложения текстур на объекты трёхмерной сцены

1.2.1 Аффинное текстурирование

Этот метод текстурирования основан на приближении u, v линейными функциями, где u, v – координаты текстуры [5]. Пусть u - линейная функция, u = k1*sx+k2*sy+k3, где sx, sy - координаты принадлежащие проекции текстурируемого треугольника. Можно посчитать k1, k2, k3 исходя из того, что в вершинах грани u, v известны — это даст три уравнения, из которых находятся эти коэффициенты. Однако в таком случае вычисление цвета пикселя получается медленным. Оптимизацией подхода является расчет начальных значений координат текстур для каждого полигона, а затем билинейная интерполяция значений x и y текстуры. Основным недостатком данного метода является игнорирование координаты z, вследствие чего данные искажаются, и текстура накладывается нереалистично.

1.2.2 Перспективно-корректное текстурирование

Этот метод основан на приближении u, v кусочно-линейными функциями [5]. При отрисовке каждая сканирующая строка разбивается на части, в начале и конце каждого куска считаются точные значения u, v, а в каждой части они интерполируется линейно.

Точные значения u и v можно считать по формулам точного текстурирования, но обычно используют более простой путь. Он основан на том факте, что значении 1/Z, u/Z и v/Z зависят от sx, sy линейно. Таким образом, достаточно для каждой вершины посчитать 1/Z, u/Z, v/Z и линейно их интерполировать - точно так же, как интерполируются u и v в аффинном текстурировании. Причем, поскольку эти значения зависят от sx, sy строго линейно, то интерполяция дает не приближенные результаты, а точные.

Сами же точные значения u, v считаются, как

$$u = (u/Z) / (1/Z),$$
$$v = (v/Z) / (1/Z).$$

1.2.3 Параболическое текстурирование

Метод параболического текстурирования основан на приближении u, v квадратичными функциями - параболами. Для каждой сканирующей строки строятся приближающие u, v квадратичные функции. С помощью этих функций координаты текстур интерполируются по сканирующей строке. Для этого необходимы точные значения u, v в трех точках - начале, середине и конце строки. Они считаются точно так же, как в предыдущем методе.

1.2.4 Выводы

Наиболее приемлемой является перспективно-корректная реализация, поскольку в ее основе лежит точное текстурирование, что делает ее результаты более приближенными к действительным, нежели у аффинной, к тому же она учитывает перспективу изображения.

1.3 Анализ алгоритмов закраски

1.3.1 Однотонная закраска полигональной сетки

В данном методе закраски цвет всей поверхности рассчитывается согласно закону Ламберта [7]. Он формулируется так: плоская поверхность, имеющая одинаковую яркость по всем направлениям, отражает свет, интенсивность которого изменяется по закону косинуса - $I = I_0 \cos\theta$, где I_0 – интенсивность отражается в направлении нормали к поверхности, θ – угол между направлением на наблюдателя и нормалью к поверхности. Метод гранения позволяет получать изображения, сравнимые

по качеству с реальными объектами, лишь при выполнении следующих условий:

1. источник света находится на большом расстоянии от объекта;
2. наблюдатель находится на большом расстоянии от объекта;
3. каждая грань тела является гранью многогранника, а не аппроксимирующей поверхностью;
4. поверхность аппроксимирована большим числом небольших плоских граней.

Преимуществами данного метода закраски является простая реализация, а также небольшие требования к ресурсам. В то же время данный метод имеет ряд существенных недостатков. Так, например, он плохо подходит для гладких объектов и плохо учитывает отраженный свет, ярко выражает края фигур.

1.3.2 Метод закраски Гуро

Метод закраски Гуро основан на интерполяции интенсивности. Он позволяет устранить дискретность изменения интенсивности и создать иллюзию гладкой криволинейной поверхности [5].

Процесс закраски по методу Гуро осуществляется в четыре этапа:

1) Вычисляются нормали ко всем полигонам.

2) Определяются нормали в вершинах путем усреднения нормалей по всем полигональным граням, которым принадлежит рассматриваемая вершина, как показано на Рисунке 1.4.

3) Используя нормали в вершинах и, применяя определенную модель освещения, вычисляют значения интенсивностей в вершинах многоугольника.

4) Каждый многоугольник закрашивается путем линейной интерполяции значений интенсивностей в вершинах сначала вдоль каждого ребра, а затем и между ребрами вдоль каждой сканирующей строки

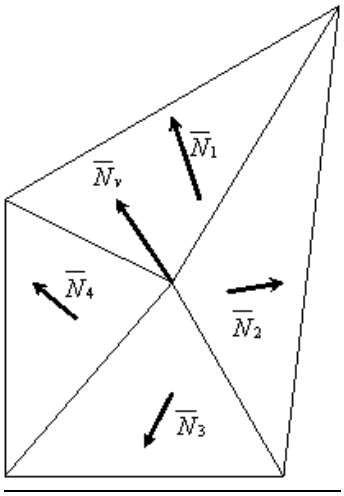

Рисунок 1.4. Определение нормалей

Метод Гуро применим только для небольших граней, расположенных на значительном расстоянии от источника света. Если же размер грани достаточно велик, то расстояние от источника света до ее центра будет значительно меньше, чем до ее вершин, и, согласно закону освещенности, центр грани должен быть освещен сильнее ребер. Однако модель изменения освещенности, принятая в методе Гуро, предполагает линейное изменение яркости в пределах грани и не позволяет сделать середину грани ярче, чем ее края. В итоге на изображении появляются участки с неестественной освещенностью.

1.3.3 Метод закраски Фонга

Метод закраски Фонга основан на интерполяции вектора нормали, который затем используется в модели освещения для вычисления интенсивности пиксела.

Процесс закраски по методу Фонга осуществляется в четыре этапа:

1. Определяются нормали к граням.

2. По нормалям к граням определяются нормали в вершинах.

3. В каждой точке закрашиваемой грани определяется интерполированный вектор нормали.

4. По направлению векторов нормали определяется цвет точек грани.

Закраска Фонга требует больших вычислительных затрат, однако при этом достигается лучшая локальная аппроксимация кривизны поверхности, получается более реалистичное изображение, правдоподобнее выглядят зеркальные блики [9].

1.3.4 Выводы

Исходя из поставленной задачи, вместе с методом z-буфера предпочтительнее использовать алгоритм Гуро, сочетая приемлемую скорость работы и качество получаемого изображения.

1.4 Выбор модели освещения

1.4.1 Модель освещения Ламберта

Модель Ламберта моделирует идеальное диффузное освещение. Свет при попадании на поверхность рассеивается равномерно во все стороны. При расчете такого освещения учитывается только ориентация поверхности (нормаль $\vec{N}$) и направление на источник света (вектор $\vec{L}$), как показано на Рисунке 1.5. Рассеянная составляющая рассчитывается по закону Ламберта: интенсивность отражения пропорциональна косинусу угла между внешней нормалью к поверхности и направлением к источнику света.

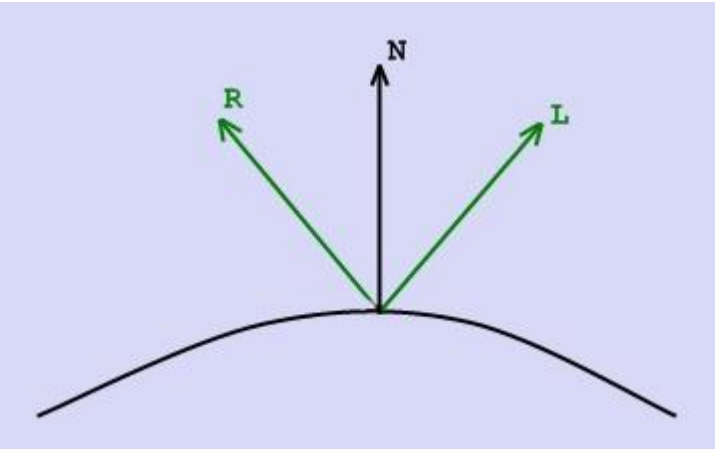

Рисунок 1.5. Отражение света в модели Ламберта

Если векторы являются единичными, то косинус угла между ними совпадает со скалярным произведением:

$$I_d = k_d\, cos(\vec{L}, \vec{N}) i_d = k_d(\vec{L} \cdot \vec{N}) i_d$$

где

I_d – рассеянная составляющая освещенности в точке;

k_d – свойство материала воспринимать рассеянное освещение;

i_d – мощность рассеянного освещения;

$\vec{L}$ – направление из точки на источник;

$\vec{N}$ – вектор нормали в точке.

1.4.2 Модель освещения Фонга

Модель Фонга – модель освещения, представляющая собой комбинацию диффузной составляющей (модели Ламберта) и зеркальной составляющей, и работает таким образом, что кроме равномерного освещения на материале может еще появляться блик [10]. Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части, как показано на Рисунке 1.6, где $\overrightarrow{\mathrm{V}}$ – направление на наблюдателя, $\overrightarrow{\mathrm{L}}$ – направление на источник, $\overrightarrow{\mathrm{R}}$ – отраженный луч. Отраженная составляющая освещенности в точке зависит от того, насколько близки направления вектора, направленного на наблюдателя, и отраженного луча.

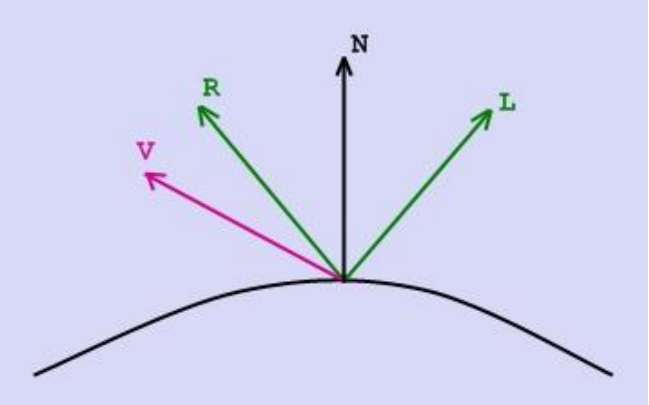

Рисунок 1.6. Модель Фонга

В модели учитываются интенсивности фоновой, рассеянной компонент освещения, а также глянцевые блики.

$$I = k_a I_a + k_d(\vec{N}, \vec{L}) + k_S(\vec{N}, \vec{V})^p,$$

где

$\vec{N}$ — вектор нормали к поверхности в точке;

$\vec{L}$ — направление проецирования (направление на источник света);

$\vec{V}$ — направление на наблюдателя;

k_a — коэффициент фонового освещения;

k_s — коэффициент зеркального освещения;

k_d — коэффициент диффузного освещения;

p — степень блеска.

Модель Фонга учитывает только свойства заданной точки и источников освещения, игнорируя эффекты рассеивания, линзирования, отражения от соседних тел.

1.4.3 Модель освещения Уиттеда

Модель освещения предназначена для того, чтобы рассчитать интенсивность отраженного к наблюдателю света в каждом пикселе изображения. Она может быть локальной или глобальной [10]. В первом случае во внимание принимается только свет, падающий от источника

(источников), и ориентация поверхности. Во втором учитывается также свет, отраженный от других объектов сцены или пропущенный сквозь них, как показано на Рисунке 1.7, где $\vec{V}$ – направление взгляда, $\vec{R}$ – отраженный луч направления взгляда, $\vec{P}$ – преломленный луч направления взгляда, L – направление на источник освещения, n1, n2 – показатели преломления сред.

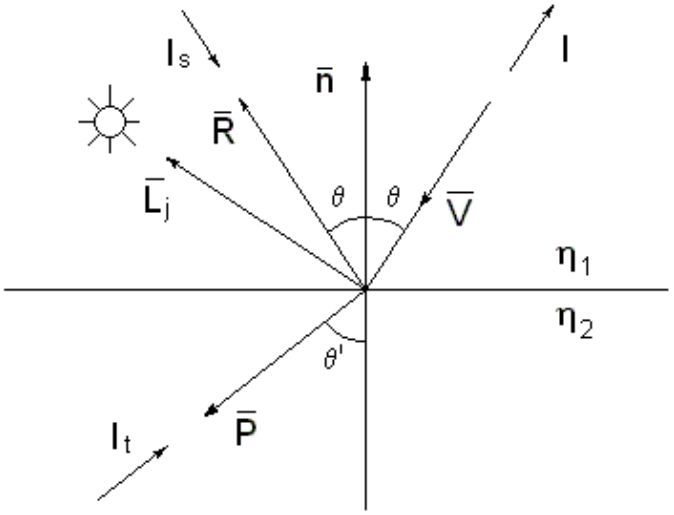

Рисунок 1.7. Зеркальное отражение и преломление

Согласно модели Уиттеда наблюдаемая интенсивность определяется суммарной интенсивностью:

$$I = k_a I_a C + k_d I_d C + k_s I_S + k_r I_r + k_t I_t,$$

где

k_a - коэффициент рассеянного отражения;

k_d - коэффициент диффузного отражения;

k_s- коэффициент зеркальности;

k_r - коэффициент отражения;

k_t - коэффициент преломления;

I_a- интенсивность фонового освещения;

I_d - интенсивность, учитываемая для диффузного рассеивания;

I_S - интенсивность, учитываемая для зеркальности;

I_r - интенсивность излучения, приходящего по отраженному лучу;

I_t - интенсивность излучения, приходящего по преломленному лучу;

C - цвет поверхности.

Модель Уиттеда учитывает эффекты преломления и отражения, зеркальности.

1.4.4 Выводы

При отрисовке с помощью z-буфера нет необходимости выбирать сложную модель освещения, поэтому было решено выбрать локальную модель Ламберта. При отрисовке с помощью обратной трассировки с учетом поставленной задачи необходимо реализовать эффекты преломления, отражения, прозрачности, затенения, поэтому, чтобы добиться реалистичного изображения решено было, несмотря на сложность реализации, использовать глобальную модель освещения Уиттеда. В ней будут учитываются только точечные источники различного цвета и источники рассеянного освещения.

1.5 Описание трёхмерных преобразований

1.5.1 Способы хранения и обработки декартовых координат

Координаты можно хранить в форме вектор-столбца [x, y, z]. Однако в этом случае неудобно применять преобразования поворота, так такой вектор нельзя умножить на соответствующие матрицы трансформации размерности четыре. Целесообразнее использовать вектор-столбцы размерности четыре - [x,y,z,w], где w для точки равно одному. Преобразования координат выполняются умножением слева преобразуемого вектора-столбца на соответствующую матрицу линейного оператора.

1.5.2 Матрицы аффинных преобразований декартовых координат

1) сдвиг точки на dx, dy, dz по координатным осям:

$$\begin{cases} X = x + dx \\ Y = y + dy \\ Z = z + dz \end{cases} \quad \begin{pmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{pmatrix};$$

2) масштабирование относительно начала координат с коэффициентами k_x, k_y, k_z:

$$\begin{cases} X = x \cdot k_x \\ Y = y \cdot k_y \\ Z = z \cdot k_z \end{cases} \quad \begin{pmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix};$$

3) поворот относительно осей x, y, z на угол φ:

- ось x:

$$\begin{cases} X = x \\ Y = y \cdot \cos\phi + z \cdot \sin\phi \\ Z = -y \cdot \sin\phi + z \cdot \cos\phi \end{cases} \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- ось y:

$$\begin{cases} X = x \cdot \cos\phi - z \cdot \sin\phi \\ Y = y \\ Z = x\sin\phi + z\cos\phi \end{cases} \quad \begin{pmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- ось z:

$$\begin{cases} X = x \cdot \cos\phi + y \cdot \sin\phi \\ Y = -x \cdot \sin\phi + y \cdot \cos\phi \\ Z = z \end{cases} \quad \begin{pmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

1.5.3 Кватернионы

Кватернионы расширяют понятие вращения в трёх измерениях на вращение в четырёх измерениях. Они разрешают проблему т.н. "блокировки осей" и позволяют выполнить плавное и непрерывное вращение. Кватернионы определяются четырьмя действительными числами [x y z w]. Они вычисляются из комбинации оси и угла вращения. Пусть ось имеет координаты (ox, oy, oz), а угол равен a, тогда кватернион хранится в виде:

$$X = ox * \sin(a / 2)$$
$$Y = oy * \sin(a / 2)$$
$$Z = oz * \sin(a / 2)$$
$$W = \cos(a / 2)$$

С помощью кватернионов можно повернуть объект вокруг любой оси на заданный угол. Для того, чтобы применить несколько операций поворота последовательно, достаточно перемножить соответствующие кватернионы между собой. И затем результирующий кватернион можно преобразовать в матрицу поворота следующим образом:

$$\begin{pmatrix} 1-2*Y^2-2*Z^2 & 2*X*Y-2*Z*W & 2*X*Z+2*Y*W & 0 \\ 2*X*Y+2*Z*W & 1-2*X^2-2*Z^2 & 2*Y*Z-2*X*W & 0 \\ 2*X*Z-2*Y*W & 2*Y*Z+2*X*W & 1-2*X^2-2*Y^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Таким образом, кватернионы мощный инструмент, который позволяет сделать вращение более реалистичным и при этом позволяет продолжить работу с привычными матрицами поворота.

1.5.4 Преобразования трехмерной сцены в пространство камеры

Для того, чтобы преобразовать сцену в пространство камеры, и при этом сохранить перспективу, необходимо координату каждой точки сцены умножить на матрицу перспективной проекции. Матрица проекции отображает заданный диапазон усеченной пирамиды в пространство отсечения, и при этом манипулирует w-компонентой каждой вершины

таким образом, что чем дальше от наблюдателя находится вершина, тем больше становится это w-значение. После преобразования координат в пространство отсечения, все они попадают в диапазон от -w до w (вершины, находящиеся вне этого диапазона, отсекаются) [11].

Матрица перспективной проекции выглядит следующим образом:

$$\begin{pmatrix} \frac{1}{ar \cdot \tan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & \frac{-NearZ - FarZ}{NearZ - FarZ} & \frac{2 \cdot FarZ \cdot NearZ}{NearZ - FarZ} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

где

ar - отношение ширины изображения к его высоте,

a - угол обзора камеры,

NearZ - координата z ближней к камере плоскости отсечения пирамиды видимости,

FarZ - координата z дальней от камеры плоскости отсечения пирамиды видимости.

После перевода в пространство камеры, все координаты необходимо спроецировать на одну плоскость путем деления на координату z. Стоит отметить, что после применения умножения вектора координат на матрицу перспективной проекции, истинная координата z автоматически заносится в координату w, поэтому вместо деления z делят на w.

1.5.5 Преобразования трехмерной сцены в пространство области изображения

Для того, чтобы преобразовать спроецированные координаты в координаты области изображения, достаточно умножить вектор координат на следующую матрицу:

$$\begin{pmatrix} hW & 0 & 0 & hW \\ 0 & hH & 0 & hH \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

где

W - ширина области изображения,

H - высота области изображения,

hW - W / 2,

hH - H / 2

1.6 Методы сглаживания

1.6.1 Избыточная выборка сглаживания

Избыточная выборка сглаживания (SSAA) - простой и прямолинейный метод сглаживания [12]. Он заключается в том, что изображение рассчитывается в виртуальном разрешении, в несколько раз превосходящем реальное разрешение монитора ПК, после чего масштабируется и фильтруется до итогового разрешения. При этом цвет каждого пикселя реального разрешения вычисляется на основе нескольких субпикселей виртуального. Это позволяет значительно повысить качество

изображения, но при этом нагрузка на вычислитель возрастает в несколько раз, а скорость, соответственно, падает.

1.6.2 Множественная выборка сглаживания

Метод множественной выборки сглаживания (MSAA) пришел на смену SSAA [12]. Он потребляет меньше ресурсов, так как происходит сглаживание только краев объекта, а не всей картинки, как в SSAA. Из минусов метода стоит отметить, что на прозрачных полигонах (стекла, вода..) данный метод не работает. И так как сглаживается только часть изображения, то можно наблюдать еще и артефакты. MSAA выгоднее использовать на низких разрешениях, чем оно выше, тем накладнее по ресурсам сглаживание.

1.6.3 Быстрое приближенное сглаживание

Метод быстрого приближенного сглаживания (FXAA) метод сглаживания, созданный Nvidia и представляющий собой однопроходный пиксельный шейдер, который обсчитывает результирующий кадр на этапе постобработки. Является более производительным решением по сравнению с MSAA, что, однако, сказывается на точности работы и качестве изображения [12].

1.6.4 Выводы

Учитывая простоту реализации и высокое качество получаемого изображения, предпочтительнее использовать метод избыточной выборки сглаживания, несмотря на низкую скорость работы.

2. Конструкторский раздел

В данном разделе приводятся схемы реализованных алгоритмов и математические выкладки.

2.1 Алгоритм удаления невидимых граней с использованием z-буфера

На Рисунке 2.1 приведена схема алгоритма z-буфера.

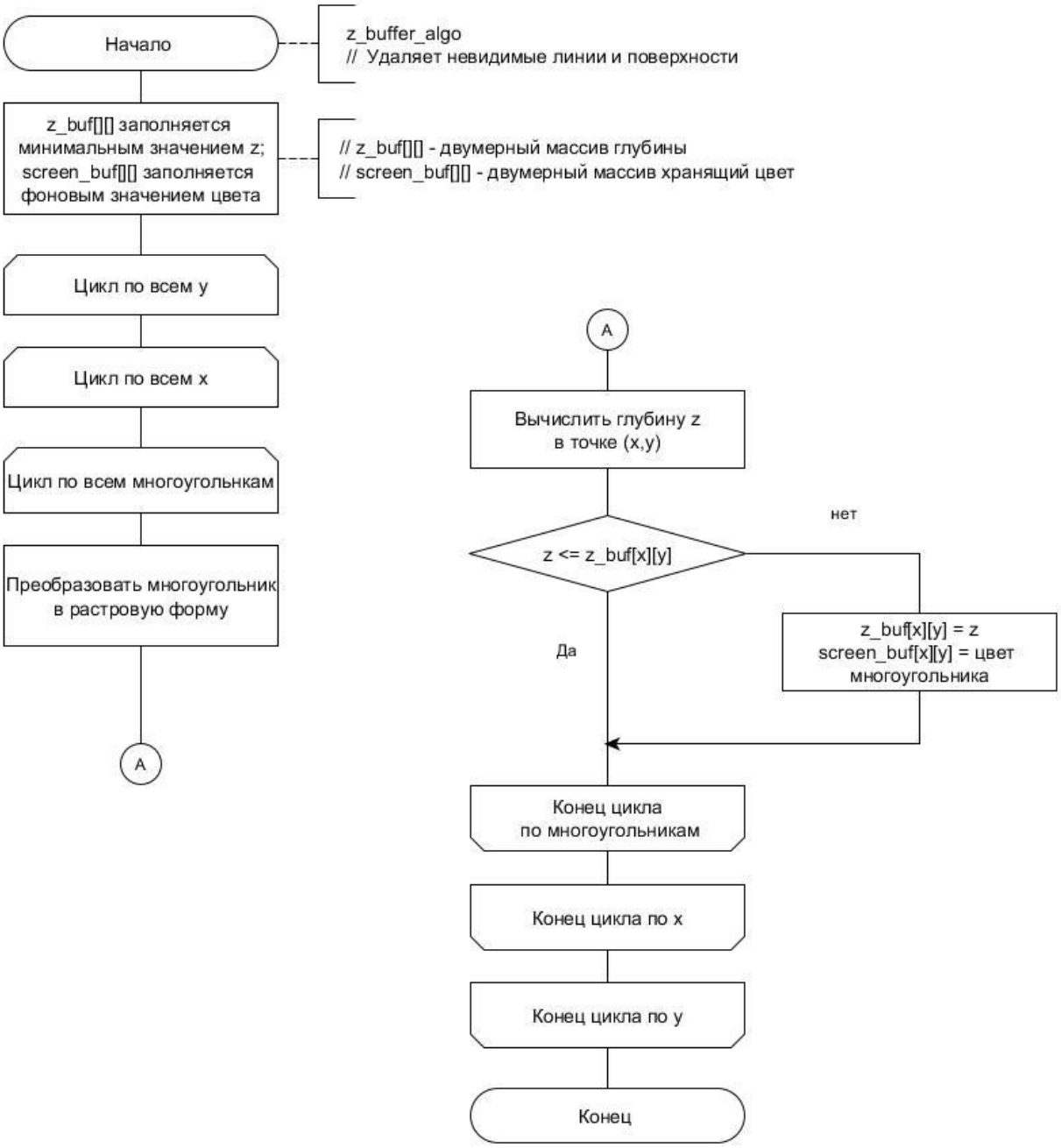

Рисунок 2.1. Алгоритм z-буфера

2.2 Алгоритм обратной трассировки лучей

2.2.1 Общий алгоритм

На Рисунке 2.2 приведена схема алгоритма обратной трассировки лучей.

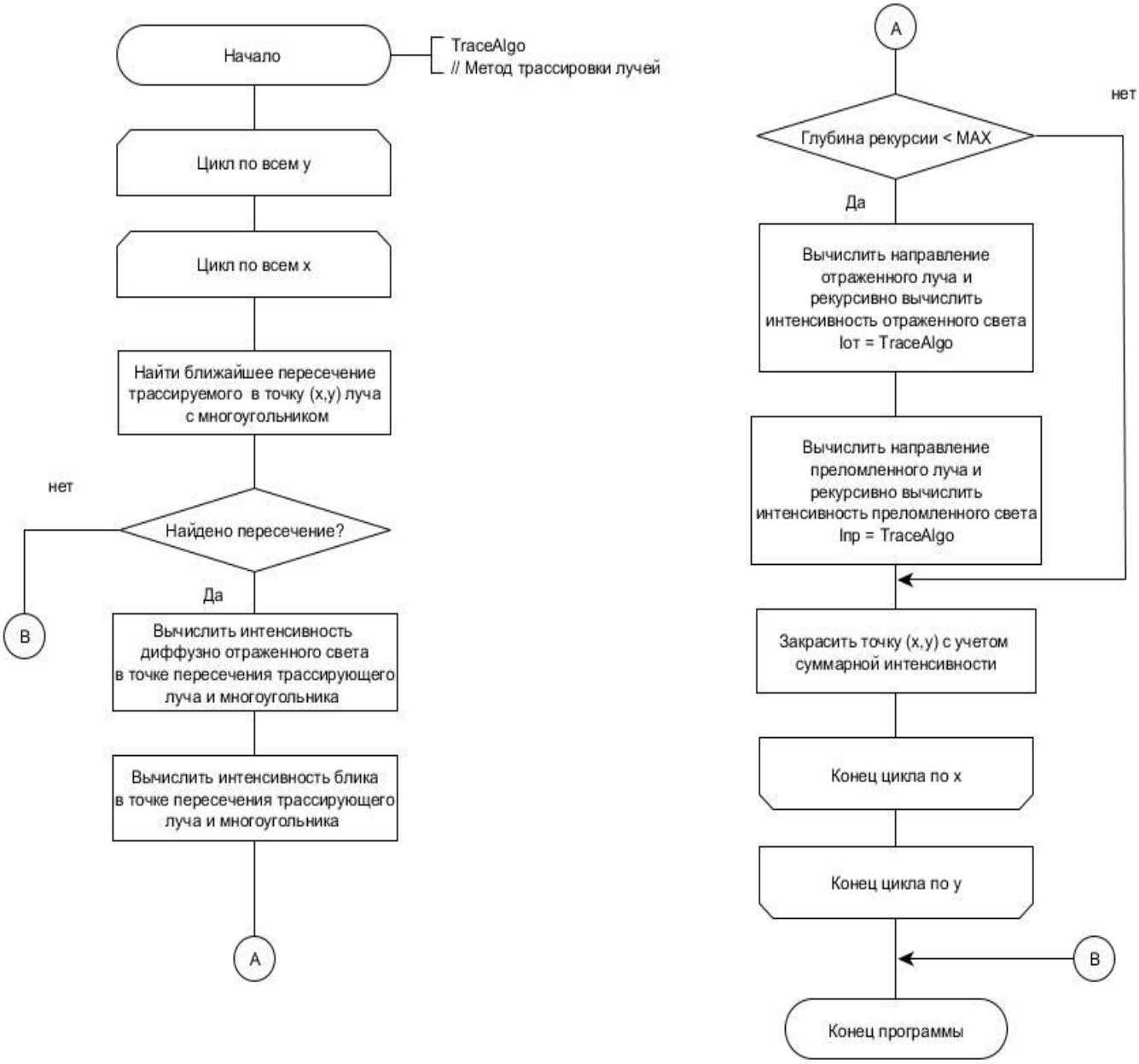

Рисунок 2.2 Схема алгоритма трассировки лучей

2.2.2 Пересечение трассирующего луча с треугольником

Вычислить пересечение трассирующего луча с полигоном можно, проведя барицентрический тест, схема которого изображена на Рисунке 2.3.

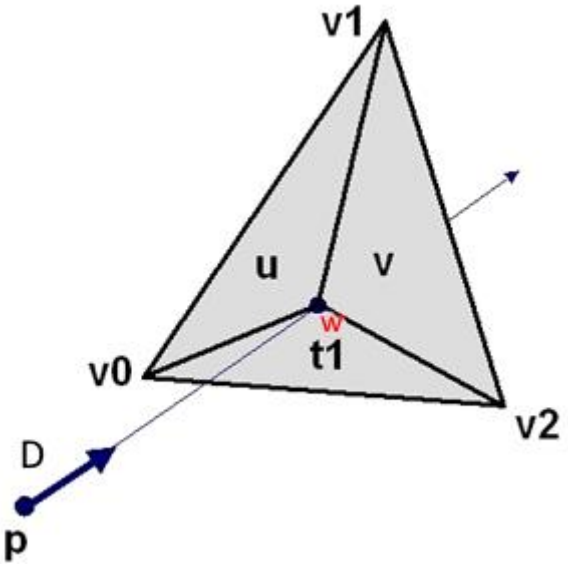

Рисунок 2.3. Барицентрический тест

Имея три точки на плоскости, можно выразить любую другую точку через ее барицентрические координаты. Первое уравнение берется из определения барицентрических координат, выражая точку пересечения w. С другой стороны, эта же точка w лежит на прямой. Второе уравнение, таким образом, — это параметрическое уравнение прямой. Второе уравнение, таким образом, — это параметрическое уравнение прямой.

$$1)\ \ w(u,v) = (1-u-v) * v1 + u * v2 + v * v0,$$

$$2)\ \ w(t) = p + t * \vec{D},$$

где

u, v - барицентрические координаты,

v0, v1, v2 - координаты вершин треугольника,

p - начало луча,

$\vec{D}$ - вектор совпадающий с направлением луча

t - параметр параметрического уравнения прямой

Приравняв правые части уравнений 1 и 2 можно получить третье уравнение, которое является системой трех уравнений с тремя неизвестными (u,v,t).

$$3)\ \ p + t * d = (1-u-v) * v1 + u * v2 + v * v0,$$

Проведя алгебраические преобразования, ответ можно получить в следующем виде:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{dot(P, E1)} * \begin{bmatrix} dot(Q, E2) \\ dot(P, T) \\ dot(Q, D) \end{bmatrix},$$

где

$\overrightarrow{E1} = v1 - v0$

$\overrightarrow{E2} = v2 - v0$

$\vec{T} = p - v0$

$\vec{P} = cross(\vec{D}, \overrightarrow{E2})$

$\vec{Q} = cross(\vec{T}, \overrightarrow{E1})$,

p – начало луча, $\vec{D}$ – вектор совпадающий с направлением луча, dot(a, b) – скалярное произведение векторов, cross(a,b) – векторное произведение.

Найдя отсюда u, v и (1-u-v), необходимо проверить их на принадлежность интервалу [0;1]; t – будет являться расстоянием до точки пересечения.

2.2.3 Пересечение луча со сферой

Пусть C – центр сферы, r – ее радиус, P – точка пересечения луча со сферой, O – начало луча, $\vec{D}$ – вектор, показывающий направление луча, dot(a, b) – скалярное произведение векторов, cross(a,b) – векторное

произведение. Тогда есть два уравнения, одно из которых описывает точки сферы, а другое — точки луча:

$$dot(P - C, P - C) = r^2$$

$$P = O + t\vec{D}$$

Точка P, в которой луч падает на сферу, является одновременно и точкой луча, и точкой на поверхности сферы, поэтому она должна удовлетворять обоим уравнениям одновременно. Получим:

$$dot\left(\overrightarrow{OC} + t\vec{D}, \overrightarrow{OC} + t\vec{D}\right) = r^2$$

$$t^2 dot\left(\vec{D}, \vec{D}\right) + 2 * t * dot(\overrightarrow{OC}, \vec{D}) + dot(\overrightarrow{OC}, \overrightarrow{OC}\ ) - r^2 = 0$$

Если решений у этого квадратного уравнения нет – то луч не пересекает сферу. При получении двух корней выбирается меньший из них, он и будет расстоянием от начала луча до первого пересечения.

2.2.4 Вычисление нормалей

Чтобы обеспечить реалистичность наложения света на треугольник, нормаль интерполируется с помощью барицентрических координат методом, описанным в пункте 2.3.2. Нормалью к сфере будет являться вектор с началом в центре сферы, проходящий через точку пересечения луча со сферой.

2.3 Алгоритмы текстурирования

2.3.1 Текстурирование в режиме создания сцены

На вход программе поступает модель, для каждой вершины которой уже указаны соответствующие ей координаты текстур, поэтому остается только найти значение координат в промежутке между этими точками.

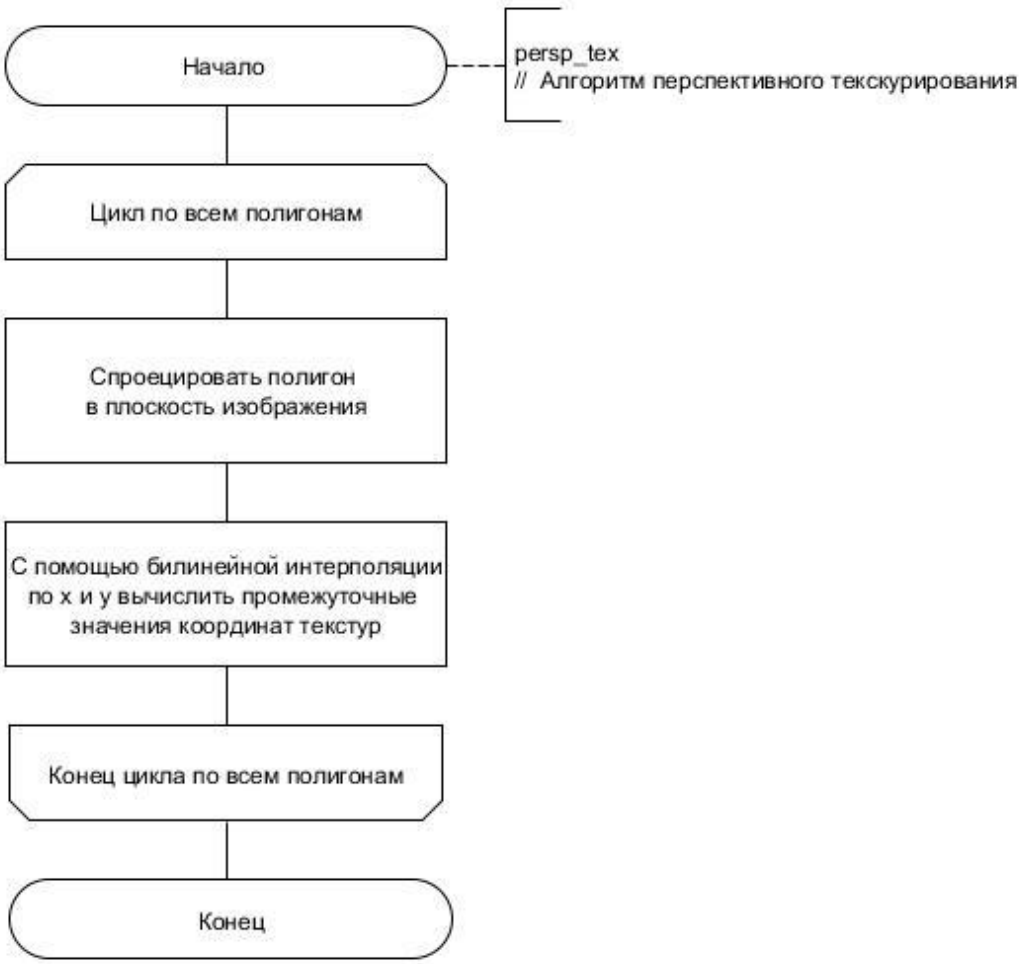

Рисунок 2.4. Алгоритм перспективного текстурирования

2.3.2 Текстурирование в режиме рендеринга с помощью обратной трассировки лучей

При нахождении пересечения луча с полигоном, зная значения координат текстур в его вершинах, координату в точке пересечения можно найти интерполяцией с помощью барицентрических координат.

Обозначим скалярное произведение как:

$$\left(\overrightarrow{V1}, \overrightarrow{V2}\right) = V1_x * \ V2_x + \ V1_y * \ V2_y + \ V1_z * \ V2_z$$

Пусть

vA, vB, vC - вершины треугольника,

v – точка пересечения луча с треугольником,

tA, tB, tC – координаты текстур в вершинах,

$v = \overrightarrow{vAB} * a + \overrightarrow{vAC} * b + vA$ – из определения барицентрических координат.

$\overrightarrow{(vAB}, \overrightarrow{vAB}) * a + \overrightarrow{(vAC}, \overrightarrow{vAB}) * b = (\overrightarrow{v - vA}, \overrightarrow{vAB});$

$\overrightarrow{(vAB}, \overrightarrow{vAC}) * a + \overrightarrow{(vAC}, \overrightarrow{vAC}) * b = (\overrightarrow{v - vA}, \overrightarrow{vAC});$

тогда

$A1 = \overrightarrow{(vAB}, \overrightarrow{vAB}); \; A2 = \overrightarrow{(vAC}, \overrightarrow{vAB}); \; A3 = (\overrightarrow{v - vA}, \overrightarrow{vAB});$

$B1 = \overrightarrow{(vAB}, \overrightarrow{vAC}); \; B2 = \overrightarrow{(vAC}, \overrightarrow{vAC}); \; B3 = (\overrightarrow{v - vA}, \overrightarrow{vAC});$

отсюда значения a, b:

$$a = \frac{A3 * B2 - A2 * B3}{A1 * B2 - A2 * B1};$$

$$b = \frac{A1 * B3 - A3 * B1}{A1 * B2 - A2 * B1};$$

координата текстуры в точке пересечения луча с треугольником:

$t = tAB * a + tAC * b + tA;$

Отдельно осуществляется текстурирование сферы путем сферического текстурирования. В этом случае координаты текстуры вычисляются следующим образом:

$$x = (0.5 + \text{atan2}(\vec{N}_z, \vec{N}_x) / (\pi * 2.0f)) * W;$$

$$y = (0.5 - \text{asin}(\vec{N}_y) / \pi) * H;$$

где

$\vec{N}$ – нормаль в точке падения луча

W и H – ширина и высота текстуры.

2.4 Алгоритмы закраски

2.4.1 Метод Гуро

Суть метода заключается в интерполяции интенсивности между вершинами полигона. Так как на вход программе приходит модель, для

каждой вершины которой уже указана нормаль, вычислять нормаль не нужно. Интенсивность в вершинах полигона определяется как скалярное произведение вектора светового луча и нормали в вершине. На Рисунке 2.5 приведена схема алгоритма закраски методом Гуро.

Рисунок 2.5. Алгоритм закраски методом Гуро

2.5 Модель освещения

2.5.1 Нахождение отраженного и преломленного лучей

Для нахождения направлений преломленного и отраженного лучей достаточно знать направление падающего луча $\vec{L}$ и нормаль к поверхности $\overrightarrow{N}$ в точке падения луча. Направление падающего луча и нормаль к поверхности на данном этапе известны [6].

Можно разложить $\vec{L}$ на два вектора $\overrightarrow{Lp}$ и $\overrightarrow{Ln}$, таких, что $\vec{L} = \overrightarrow{Lp} + \overrightarrow{Ln}$, где $\overrightarrow{Ln}$ параллелен $\vec{N}$, а $\overrightarrow{Lp}$ перпендикулярен, как изображено на Рисунке 2.6.

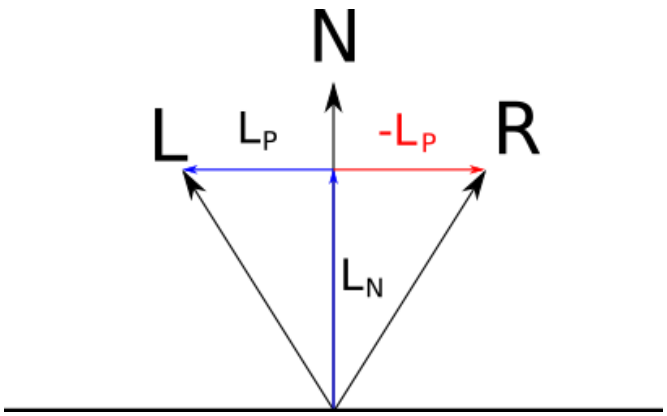

Рисунок 2.6. Разложение вектора падающего луча

$\overrightarrow{Ln}$ – проекция $\vec{L}$ на $\vec{N}$; по свойствам скалярного произведения и исходя из того, что $|\vec{N}| = 1$, длина этой проекции равна $(\vec{N}, \vec{L})$, поэтому $\overrightarrow{Ln} = \vec{N}(\vec{N}, \vec{L})$.

Отсюда $\overrightarrow{Lp} = \vec{L} - \overrightarrow{Ln} = \vec{L} - \vec{N}(\vec{N}, \vec{L})$.

Очевидно, что $\vec{R} = \overrightarrow{Ln} - \overrightarrow{Lp}$.

Подставим полученные ранее выражения и упростим, получим:

$\vec{R} = 2\vec{N}(\vec{N}, \vec{L}) - \vec{L}$. – Направление отраженного луча.

Преломленный луч $\vec{P}$ можно рассчитать из закона преломления. Он звучит следующим образом - луч падающий, луч преломленный и нормаль к поверхности в точке преломления лежат в одной плоскости. Углы падения и преломления при этом связаны следующим соотношением, называемым законом Снеллиуса:

$$\eta_i \sin\theta_i = \eta_t \sin\theta_t$$

Здесь:

η_i — показатель преломления среды, из которой свет падает на границу раздела двух сред;

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

η_t — показатель преломления среды, в которую свет попадает, пройдя границу раздела двух сред;

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

Можно получить:

$$\overrightarrow{\mathrm{P}} = \frac{n_1}{n_2} * \overrightarrow{\mathrm{L}} + \left(\frac{n_1}{n_2} * \cos(\theta_i) - \sqrt{1 - sin^2(\theta_t)}\right) * \overrightarrow{\mathrm{N}}$$

При этом $sin^2(\theta_t) = \left(\frac{n_1}{n_2}\right)^2 * \sin^2(\theta_i)$

Где θ_i – угол падения, θ_t – угол преломления, n_2 и n_1 – абсолютные показатели преломления двух сред.

2.5.2 Расчет интенсивностей

Интенсивность света, диффузно отражающегося в точке поверхности, можно вычислить следующим образом (не зависит от положения наблюдателя).

$$I_d = k_d * \sum I_i * (\vec{n}, \overrightarrow{L_i}),$$

где

k_d – коэффициент диффузного отражения;

I_i – интенсивность света, попадающего в точку от i-го источника освещения;

$\vec{n}$ - нормаль к поверхности в данной точке;

$\overrightarrow{L_i}$ - единичный вектор, совпадающий по направлению с вектором, проведенным из i-го источника в рассматриваемую точку.

Интенсивность света, отраженного зеркально, может быть вычислена следующим образом (зависит от положения наблюдателя):

$$I_s = k_s \sum I_i * (\vec{S}, \overrightarrow{R_i})^N,$$

где

k_s – коэффициент зеркального отражения;

I_i – интенсивность света, попадающего в точку от i-го источника освещения;

$\vec{S}$ - единичный вектор, совпадающий по направлению с вектором из рассматриваемой точки в точку наблюдения;

$\overrightarrow{R_i}$- единичный вектор, задающий направление отраженного луча от i-го источника;

N - степень, аппроксимирующая пространственное распределение зеркально отраженного света.

Общую интенсивность можно определить по формуле:

$$I_r = k_a \sum I_{ia} + k_d \sum I_i * (\vec{n}, \overrightarrow{L_i}) + k_s \sum I_i * (\vec{S}, \overrightarrow{R_i})^N + k_r I_r + k_t I_t,$$

где

I_r и I_t – интенсивности, принесенные составляющими оттрассированных отраженного и преломленного лучей, I_{ia} - составляющие рассеянного освещения от i-го источника,

k_a - коэффициент рассеянного отражения,

k_d - коэффициент диффузного отражения,

k_s - коэффициент зеркальности,

k_r - коэффициент отражения,

k_t - коэффициент преломления.

2.7 Алгоритм устранения лестничного эффекта

На Рисунке 2.6 приведена схема алгоритма сглаживания

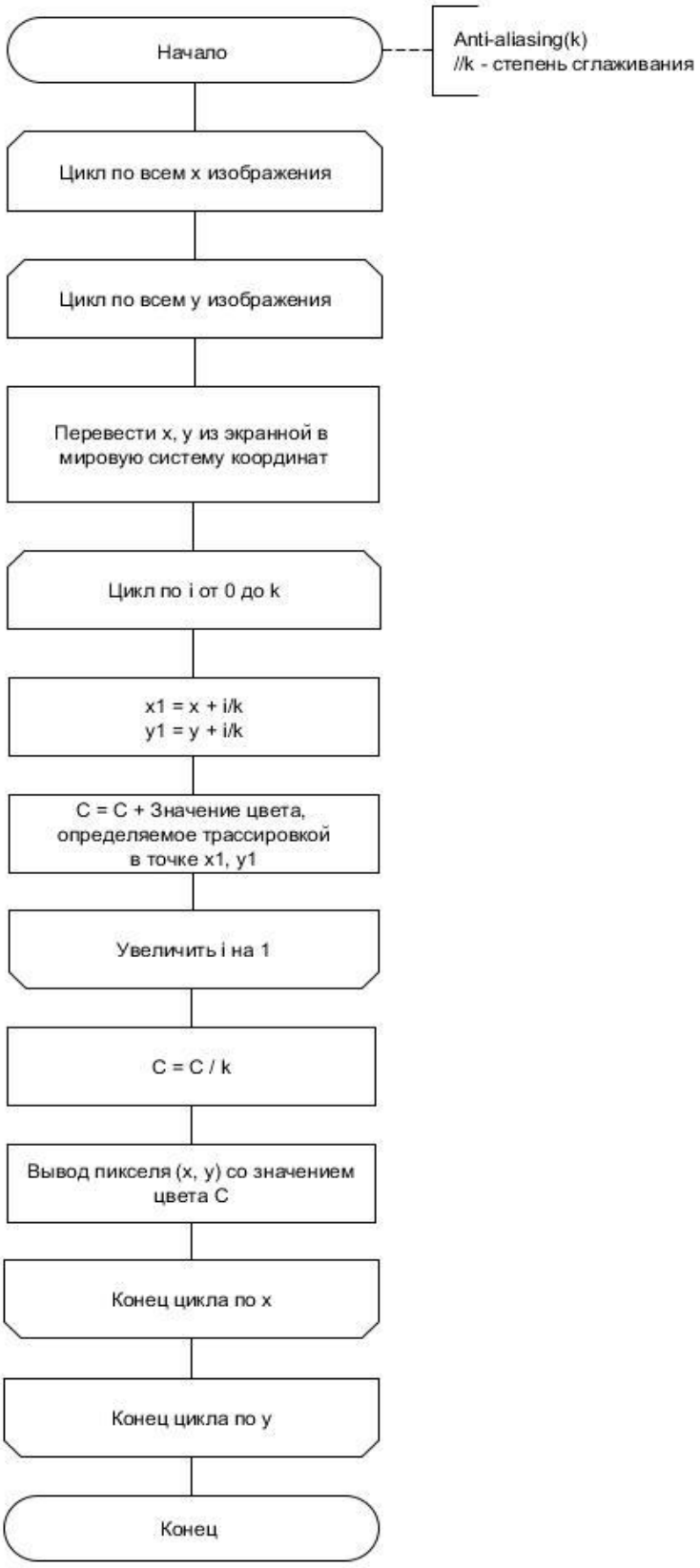

Рисунок 2.6. Алгоритм сглаживания

2.8 Описание входных данных

В данной программе входные данные подаются в виде файла с расширением .obj. В данном формате помимо координат вершин можно передавать информацию о текстурах и нормалях.

Формат файла:

1. Список вершин с координатами (x,y,z).

v 0.74 0.26 1

2. Текстурные координаты (u,v).

vt 0.830 0.5

3. Координаты нормалей (x,y,z).

vn 0.7 0.000 0.3

4. Определения поверхности (сторон) задаются в формате i1/i2/i3, где i1 - индекс координаты вершины, i2 - индекс координаты текстуры, i3 - индекс координаты нормали.

f 2/4/4 3/5/5 1/3/5

Пример задания треугольника:

v 1.000000 0.000000 -1.000000
v 1.000000 0.000000 1.000000
v -1.000000 0.000000 1.000000

vt 0.748573 0.750412
vt 0.500149 0.750166
vn 0.000000 1.000000 0.000000

f 1/1/1 2/9/1 3/1/1

3. Технологический раздел

3.1 Обоснование выбора языка, среды и платформы программирования

Для реализации поставленной задачи был выбран язык Java. Это мощный объектно-ориентированный язык, который широко распространен, что подразумевает наличие для него немалого числа библиотек, что в свою очередь также существенно упрощает разработку. К тому же язык Java использует управляемую память, благодаря которой упрощается процесс разработки, а именно: не требуется следить самостоятельно за очисткой кучи.

Для реализации проекта была выбрана среда программирования NetBeans IDE 8.2.

3.2 Описание структуры программы

3.2.1 Описание основных модулей программы

Launcher. java – создает элементы GUI и запускает программу;

ComlexObject.java – содержит функции отрисовки полигональных объектов;

RenderSceneTriangle.java – содержит функции для отрисовки треугольника;

Edge. java – содержит функции для интерполяции значений по ребру полигона;

ImageCG. java – содержит функции для работы с изображениями;

Interpolation. java – содержит функции для расчета значений интерполяции;

Light. java – содержит функции для работы с источниками света;

Ray. java – содержит функции для работы с лучом;

OBJModel.java – содержит функции для считывания obj модели;

Model.java – класс, хранящий модель;

PrimitiveObject.java – родительский класс примитивов, используемых в трассировке;

Quanternion.java – содержит функции для работы с кватернионами;

Sphere. java – содержит функции для отрисовки сферы в трассировке;

Triangle. java – содержит функции для отрисовки треугольника в трассировке;

ColorCG. java – содержит функции для работы с цветом;

Transform. java – содержит функции для работы трансформациями;

Matrix4. java – содержит функции для работы с матрицами;

Vector4. java – содержит функции для работы с вектором;

Vertex. java – содержит функции для работы с вершиной;

Camera. java – содержит описание параметров камеры и функции для работы с ней;

RayTracing.java – содержит функции для работы трассировки лучей.

Диаграмма классов проекта представлена на Рисунке 4.1.

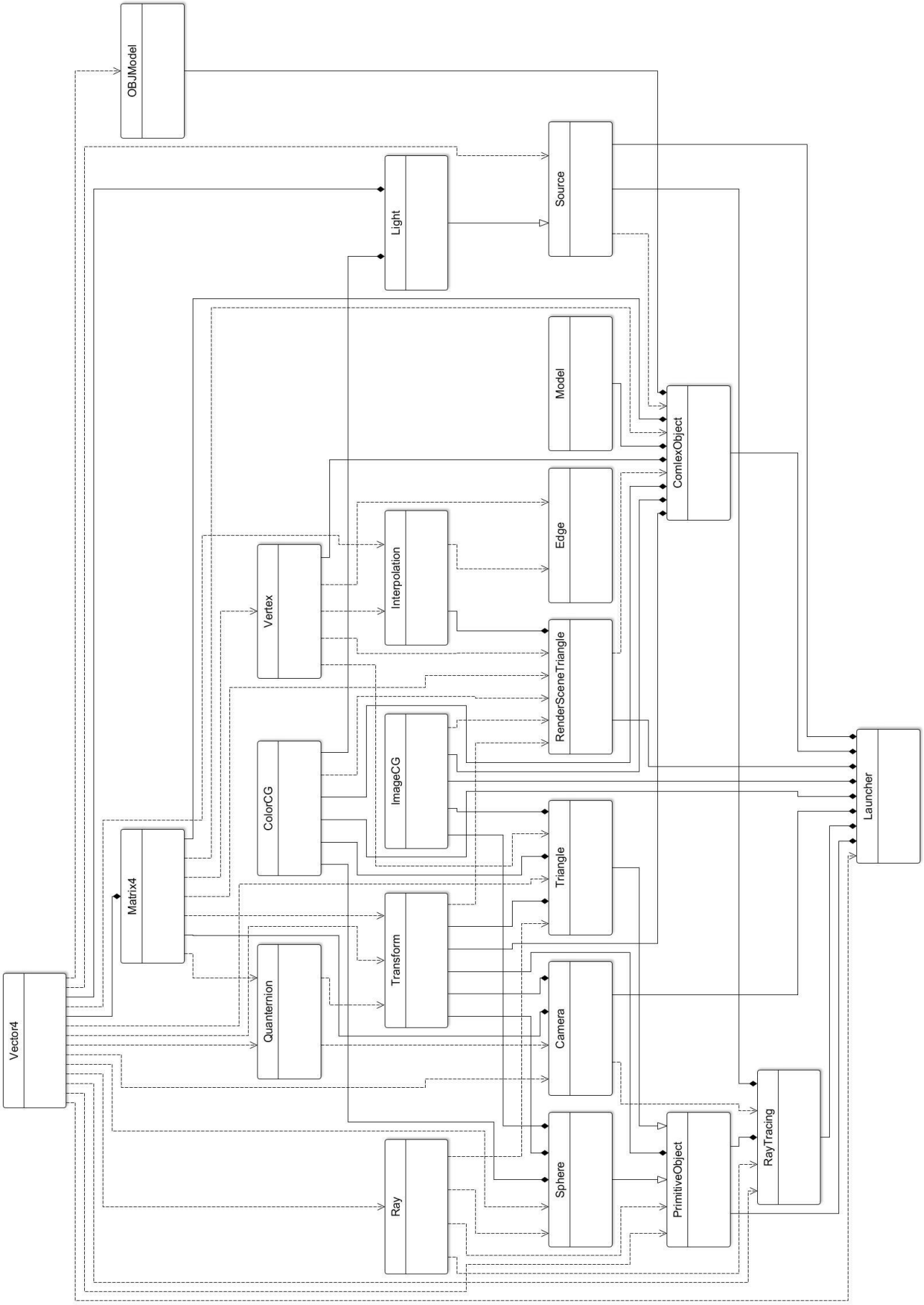

Рисунок 4.1. Диаграмма классов

3.2.2 Описание интерфейса программы

На рисунках 3.2 и 3.3 изображено главное окно программы. Пользователю доступна возможность загрузить 3D объект, добавить готовый объект из списка примитивов, а также добавить точечные источники света. Для каждого из объектов пользователь может назначать размеры, положение, углы поворота, выбирать цвет и текстуры. Кроме того, доступна возможность установить коэффициенты отражения, преломления, блеска и прозрачности объектов. Для источника света доступна настройка интенсивности и цвета. Кнопка “Рендер” отрисовывает изображение с учетом введённых коэффициентов. Пользователь может установить степень сглаживания (позволяет устранить лестничный эффект), а также интенсивность фонового освещения. Управление камерой осуществлено с помощью кнопок “w”, “a”, “s”, “d”, а также стрелок “вправо”, “влево”, “вверх”, “вниз”.

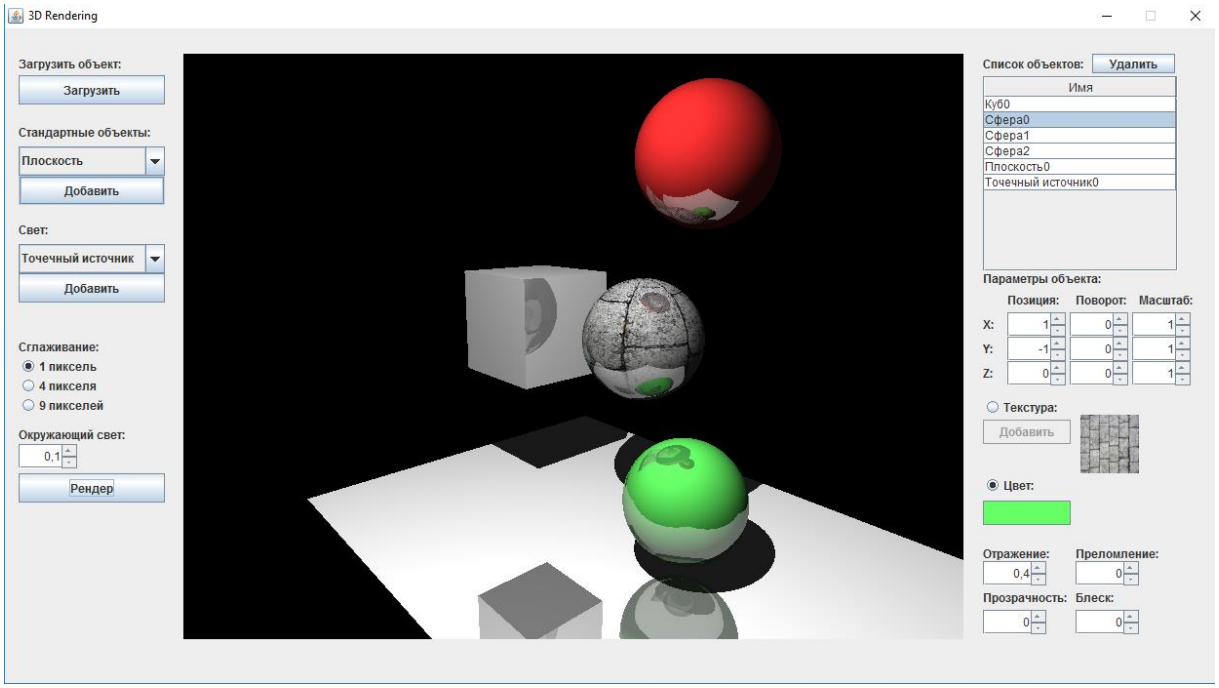

Рисунок 3.2. Снимок экрана, изображающий интерфейс программы в режиме рендеринга

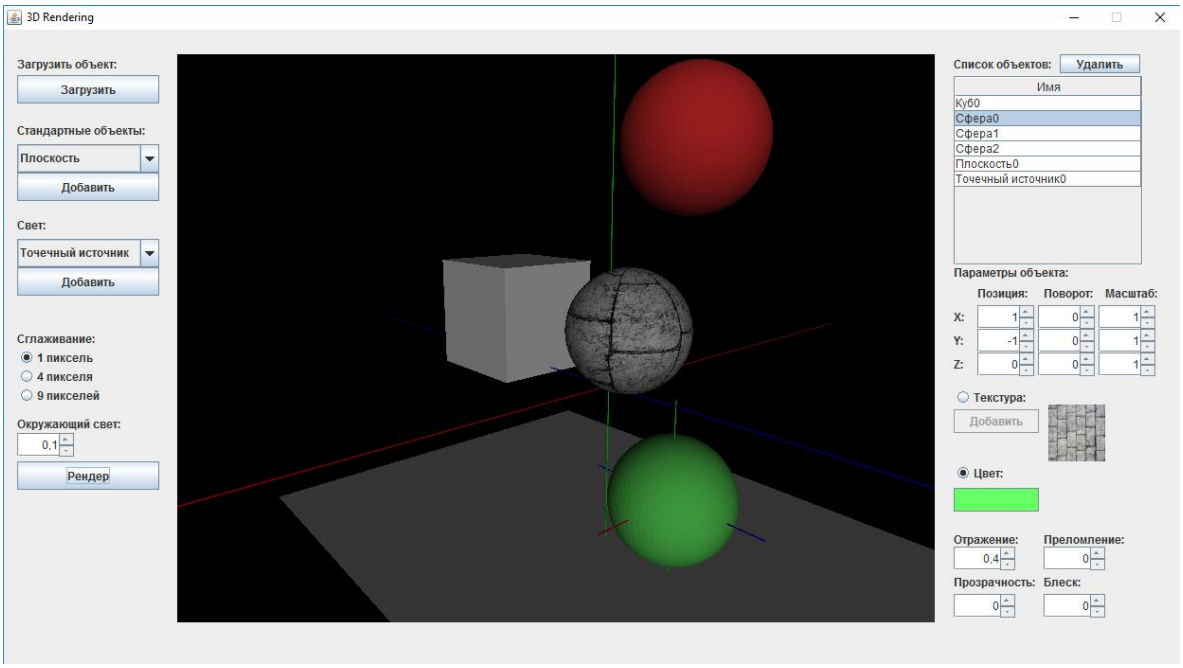

Рисунок 3.3. Снимок экрана, изображающий интерфейс программы в режиме создания сцены

4. Исследовательский раздел

4.1 Цель исследований

Целью проведенных исследований являлась оценка производительности программы в зависимости от количества потоков, выполняющих трассировку.

4.2 Описание эксперимента

Исследования проводились на компьютере следующей конфигурации:

– Процессор Intel Core i3-4150 3.5 GHz, 4 логических ядра

– ОЗУ 8Гб

– ОС Windows 10

Для исследования быстродействия программы была создана сцена, изображение которой представлено на Рисунке 4.1:

На ней присутствуют куб, плоскость, три сферы, к которым применены эффекты отражения, и один источник света.

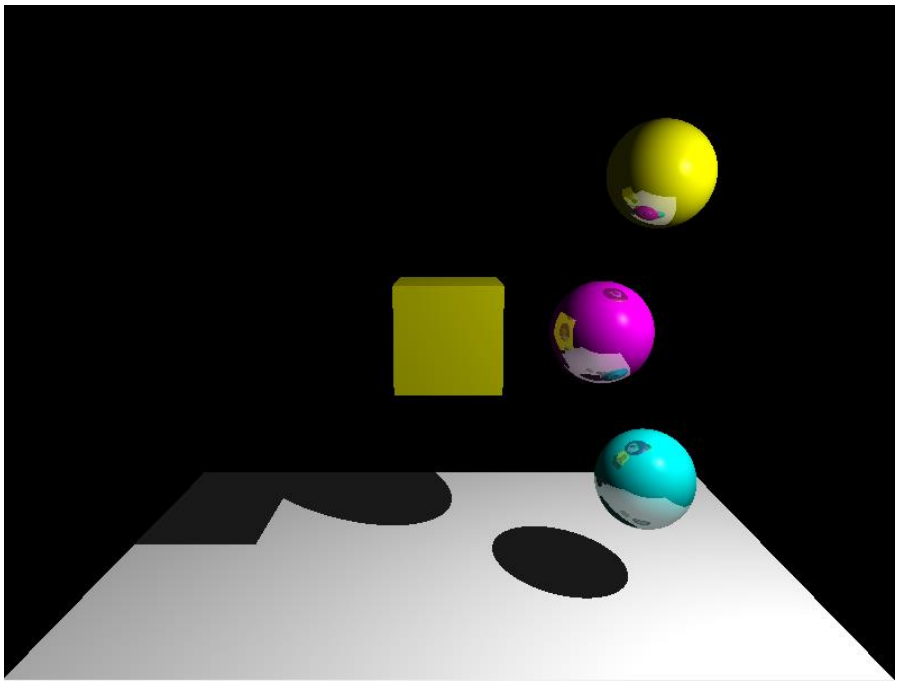

Рисунок 4.1. Сцена эксперимента

В ходе эксперимента производились замеры времени работы алгоритма трассировки лучей при различном количестве потоков: 1, 2, 4, 8, 16. Каждый эксперимент проводился 100 раз, значения времени, получаемые в результате этих экспериментов, усреднялись.

4.3 Результаты исследований

В таблице 4.1, представленной ниже, указано время в наносекундах визуализации сцены в зависимости от количества задействованных потоков.

1 поток	2 потока	4 потока	8 потоков	16 потоков
$98 * 10^7$	$76 * 10^7$	$63 * 10^7$	$65 * 10^7$	$64* 10^7$

Таблица 4.1. Время визуализации сцены в наносекундах

Для удобства анализа полученные результаты представлены на графике, изображенном на Рисунке 4.2.

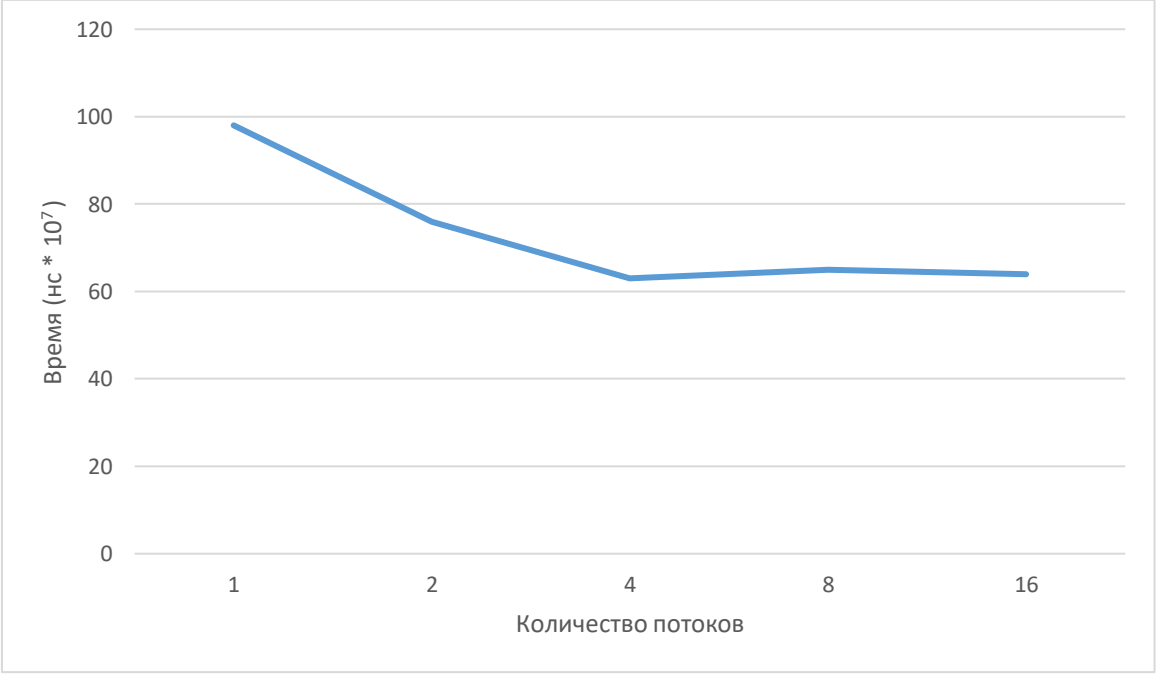

Рисунок 4.2. Зависимость времени работы от количества потоков

4.4 Вывод

Из проведенных исследований можно сделать вывод о том, что наилучшего времени удалось добиться при количестве потоков, равном четырем, что равно количеству логических ядер процессора. При дальнейшем увеличении количества потоков, прироста скорости не наблюдается.

Заключение

В ходе работы были проанализированы существующие алгоритмы удаления невидимых линий и поверхностей, закраски, текстурирования, сглаживания, а также модели освещения, указаны их преимущества и недостатки. Для решения поставленных задач были выбраны и реализованы алгоритмы обратной трассировки лучей с применением модели Уиттеда, алгоритм с применением z-буфера, перспективного текстурирования, метод Гуро, алгоритм сглаживания на основе избыточной выборки сглаживания. Также проведена оптимизация алгоритма обратной трассировки путем его распараллеливания для выполнения на нескольких ядрах процессора.

Реализованная в результате программа позволяет создавать трехмерные сцены из геометрических примитивов, а также пользовательских моделей, загружаемых в виде obj файлов, назначать им коэффициенты прозрачности, блеска, отражения и преломления, добавлять точечные источники света, указывать их цвет и интенсивность, кроме того имеется возможность перемещения, вращения, изменение размера обектов, наложения на них текстур.

Было проведено исследование, в результате которого была выявлена зависимость производительности программы от количества потоков. Наибольшая производительность наблюдается при выполнении трассировки количеством потоков, равных количеству ядер компьютера.

В качестве перспектив дальнейшего развития данной программной системы можно предложить дальнейшую оптимизацию алгоритма обратной трассировки введением октантных деревьев, добавление новых примитивов, расширение функционала программы – введение функции редактирования вершин и ребер.

Литература

1. Иванов В.П., Батраков А.С. Трехмерная компьютерная графика / Под ред. Г.М. Полищука. — М.: Радио и связь, 1995. — 224 с.
2. https://www.intuit.ru/studies/courses/70/70/lecture/2102?page=2
3. http://compgraph.tpu.ru/weiler_atherton.htm
4. http://ray-tracing.ru/articles164.html
5. http://www.codenet.ru/progr/video/tex/
6. http://www.realtimerendering.com/resources/RTNews/html/
7. https://studbooks.net/2248060/informatika/odnotonnaya_zakraska_metod_graneniya
8. http://compgraph.tpu.ru/guro.htm
9. http://stratum.ac.ru/education/textbooks/kgrafic/additional/addit26.html
10. http://grafika.me/node/344
11. https://triplepointfive.github.io/ogltutor/tutorials/tutorial12.html
12. https://soft-tuning.ru/zhelezo/40-%D1%81%D0%B3%D0%BB%D0%B0%D0%B6%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5.html
13. Д. Роджерс, Дж. Адамс «Математические основы компьютерной графики» - Москва, «Мир», 2001г.- 604 с.