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

ФАКУЛЬТЕТ ИУ, Информатика и системы управления

КАФЕДРА ИУ7, Программное обеспечение ЭВМ и информационные технологии

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Моделирование программы детского конструктора

Студент ИУ7-54Б _____ _____
(Группа) (Подпись, дата) (И.О.Фамилия)

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

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2021 г.

Содержание

Введение

Сегодня повсеместно применяются визуальные эффекты, они позволяют создать на компьютере изображения. Процесс создания изображения на компьютере стал частью большого числа сфер деятельности человека: кино, фотография, иллюстрация, видеоигры, печать и т.д.

Целью проекта является разработка программы для создания трехмерных графических сцен из трехмерных геометрических объектов, схожих с детским конструктором и их визуализация с учетом выбранного цвета, позиции камеры. Предусмотреть условия размещения каждого объекта с учетом остальных, уже присутствующих в сцене объектов — деталей.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ существующих алгоритмов удаления невидимых линий и поверхностей, закраски, а также моделей освещения и выбрать из них подходящие для выполнения проекта;
- произвести основные математические расчеты для реализации выбранных алгоритмов;
- выбрать подходящую программную платформу для реализации поставленной задачи;
- реализовать программное обеспечение для моделирования детского конструктора и его интерфейс.

1. Аналитический раздел

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

1.1 Объекты сцены

Объекты сцены представляют собой размещенные пользователем в пространстве сцены трехмерные объекты, которые программа визуализирует, алгоритмами, соответствующим выбранному режиму рендера изображения. Для размещения на сцене пользователю предлагается выбор из стандартных фигур: Куб, Цилиндр, Пирамида, Конус, Сфера, а также возможность загрузить собственную фигуру из указанного файла. Стандартные фигуры также хранятся в файлах по системному пути, заданному в конфигурации программы, что позволяет единообразно работать со всеми объектами на этапе их размещения.

1.1.1 Отражения

В реализации программе, в согласии с техническим заданием, не производится расчет и отображение отражений света от объектов и все объекты не обладают параметрами отражения. Это позволяет ускорить процесс расчета изображения сцены.

1.1.2 Размещение объектов

При выборе стандартного или загрузке пользовательского объекта, пользователю отображаются его фантомы - копии размещаемого объекта, размещенные во всех доступных для размещения местах: для стандартных объектов - где существует достаточная плоскость, прилегающая к границе объекта, к которой размещаемый объект может “прикрепиться”. В случае, если размещаемый объект единственный на сцене, он помещается в начало координат без размещения фантомов.

1.1.3 Ограничения на взаимное расположение объектов

Объекты могут пересекаться друг с другом при изменении параметров поворота, масштаба и смещения. При этом не должно возникать каких-либо артефактов изображения при наложении двух объектов друг на друга. При размещении объекта, объекты должны размещаться на достаточном расстоянии (в случае размещения пользовательского объекта) от исходного, чтобы не иметь с ним пересечений. Размещать объекты возможно только при помощи “прилипания” к другим объектам.

1.2 Алгоритмы построения трехмерного изображения

Алгоритмы построения трехмерного изображения позволяют преобразовывать трехмерные объекты сцены, отсекая невидимые грани при пересечении объектов с лучом зрения наблюдателя в двухмерное изображения сцены для отображения на экране пользователя.

1.2.1 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 $AnX + BnY + CnZ + D = 0$ в объектной системе координат XYZ:

$$T = \begin{bmatrix} A_1 & \dots & A_n & \dots & A_N \\ B_1 & \dots & B_n & \dots & B_N \\ C_1 & \dots & C_n & \dots & C_N \\ D_1 & \dots & D_n & \dots & D_N \end{bmatrix} \quad (1)$$

Минимальное число граней N = 4 наблюдается у тетраэдра.

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

$f(X, Y, Z) = AnX + BnY + CnZ + Dn$, 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:

$$V_n = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ X_K & Y_K & Z_K & 1 \end{bmatrix} \quad (2)$$

Не все грани, входящие в состав объекта, будут видны наблюдателю. Невидимой будет та, для которой выполняется условие — $\left(\vec{R}, \vec{N}\right) < 0$, 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 нелицевых 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 сразу 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.

1.2.2 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, может быть отнесен к следующим типам.

1. Внешний, если он целиком находится вне окна (см. Рисунок 1.1a).
2. Внутренний, если он целиком внутри окна (см. Рисунок 1.1 b).
3. Пересекающий, если он пересекает границу окна (см. Рисунок 1.1c).
4. Охватывающий, если окно целиком внутри него (см. Рисунок 1.1 d).

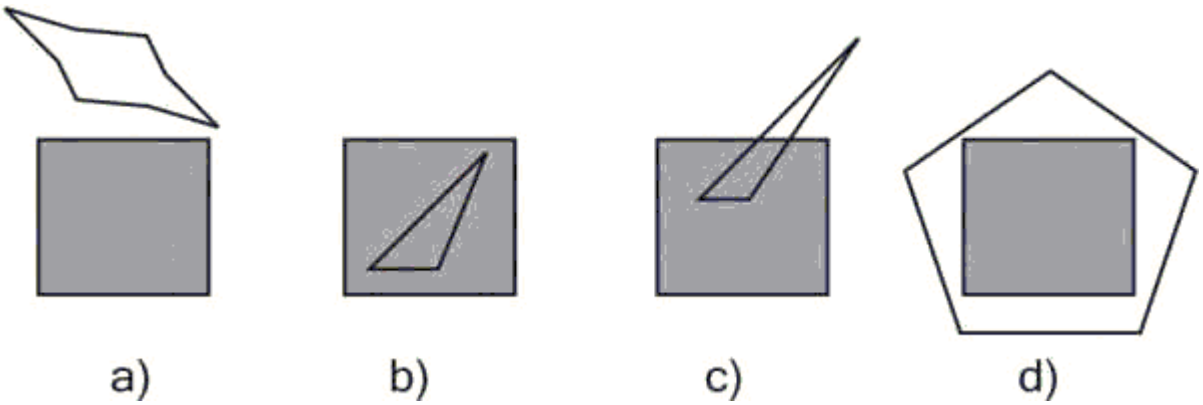

Рисунок 1.1. Варианты расположения многоугольника по отношению к окну

Для каждого окна надлежит выполнить следующие шаги.

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.

2. 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.

3. 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.

4. Если только один многоугольник охватывает окно и нет других многоугольников, имеющих общие точки с окном, то окно заполняется цветом этого многоугольника.

5. 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.

1.2.3 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).

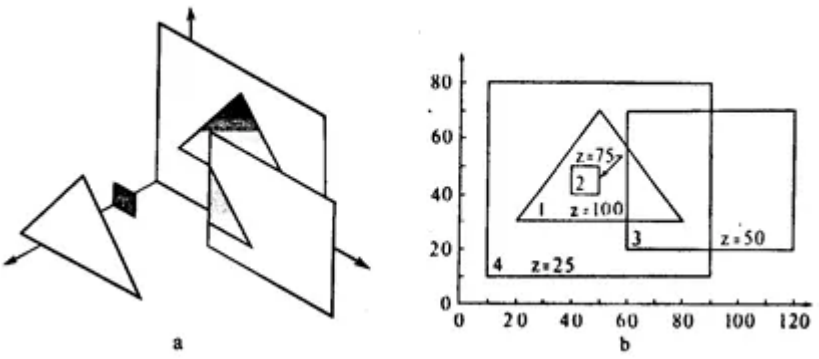

Рисунок 1.2. Алгоритм Вейлера-Азертона. Трехмерный вариант

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.

1.2.4 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 1.3.

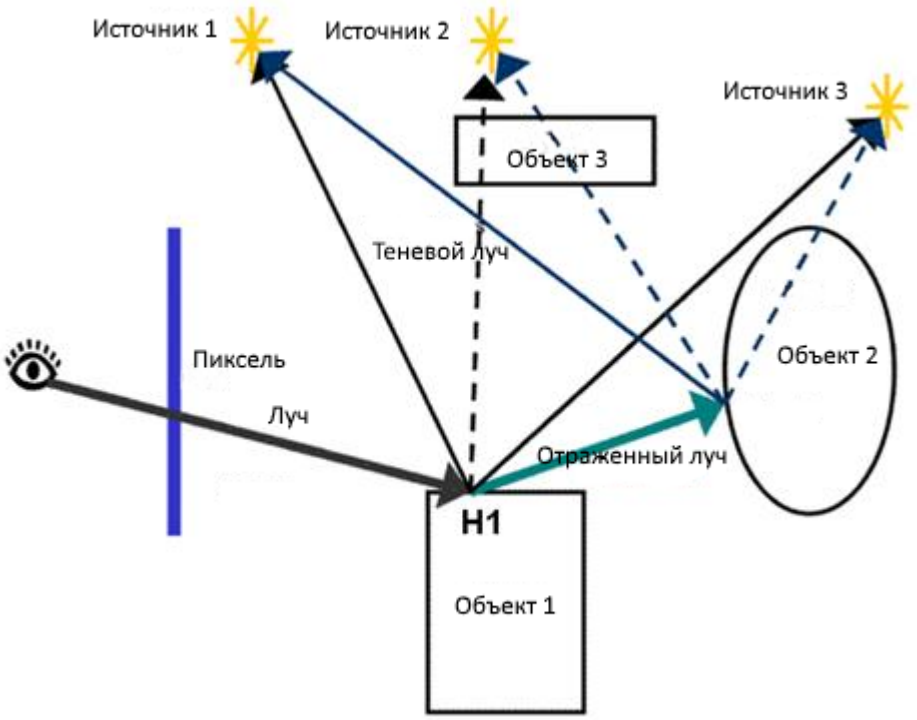

Рисунок 1.3. Алгоритм обратной трассировки лучей

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].

1.2.5 Алгоритм удаления невидимых граней с использованием Z-буфера

Это алгоритм, работающий в пространстве изображения, как показано на рисунке 1.4. 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.

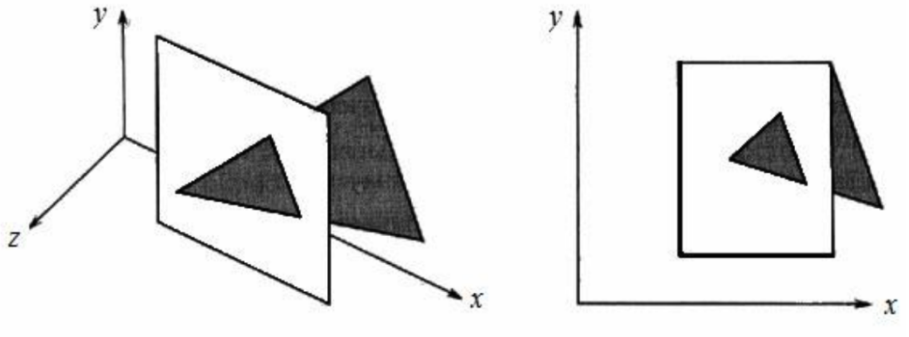

Рисунок 1.4. Пример алгоритма с использованием Z-буфера

Уравнение плоскости имеет вид:

$$Ax + By + Cz + D = 0z = -\frac{Ax + By + D}{C},\ C \neq 0 \qquad (3)$$

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

Глубина — расстояние от наблюдателя до поверхности изображаемого объекта.

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

$$z1 - z = -\frac{ax1 + d}{c} + \frac{ax + d}{c} = \frac{a(x - x1)}{c} \qquad (4)$$

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.

1.3 Анализ алгоритмов закраски

Алгоритм закраски позволяет закрашивать замкнутые области, что дает возможность управлять цветом размещаемых объектов на сцене. Различные методы закраски позволяют использовать разные алгоритмы для разных режимов работы отрисовки программы.

1.3.1 Однотонная закраска полигональной сетки

В данном методе закраски цвет всей поверхности рассчитывается согласно закону Ламберта [5]. Он формулируется так: плоская поверхность, имеющая одинаковую яркость по всем направлениям, отражает свет, интенсивность которого изменяется по закону $I = I_0 \cos\theta$, где I_0 – интенсивность отражается в направлении нормали к поверхности, θ – угол между направлением на наблюдателя и нормалью к поверхности. Метод 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. Все грани тела есть грани многогранника, а не аппроксимирующей поверхности.
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 1.5.

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.

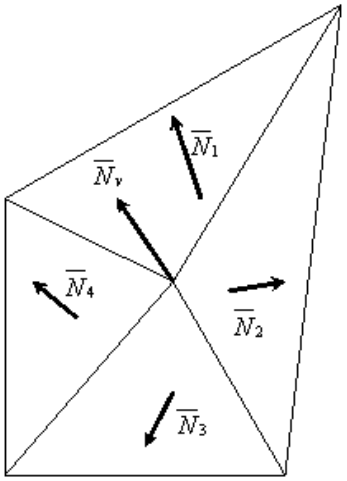

Рисунок 1.5. Определение нормалей

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 [6].

1.4 Выбор модели освещения

Модель освещения играет важную роль в построении реалистичного изображения, созволяя управлять не только глобальным освещением, но и от локальных источников света, размещаемых на сцене.

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.

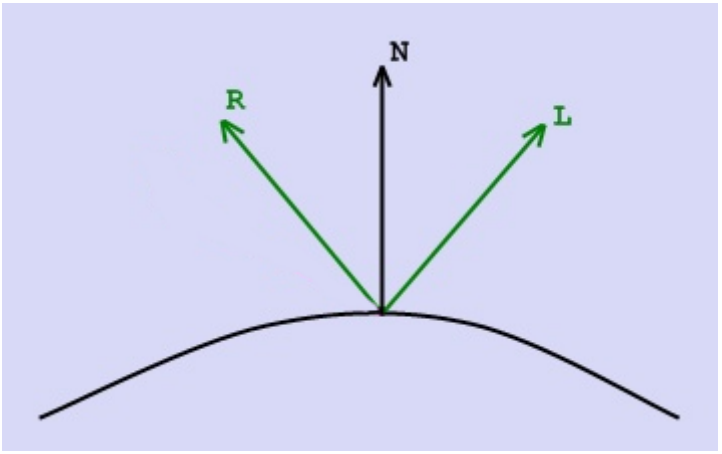

Рисунок 1.6. Отражение света в модели Ламберта

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

$$I_d = k_d \cos cos \left(\vec{L}, \vec{N}\right) i_d = k_d \left(\vec{L} \cdot \vec{N}\right) i_d \quad (5)$$

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 [7]. 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 1.7, где $\vec{V}$ – направление на наблюдателя, $\vec{L}$ – направление на источник, $\vec{R}$ – отраженный луч. Отраженная составляющая освещенности в точке зависит от того, насколько близки направления вектора, направленного на наблюдателя, и отраженного луча.

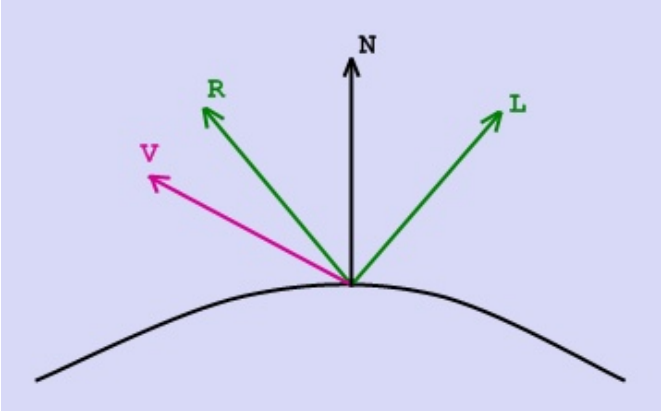

Рисунок 1.7. Модель Фонга

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

$$I = k_a I_a + k_d\left(\vec{N}, \vec{L}\right) + k_S\left(\vec{N}, \vec{V}\right)^p, \qquad (6)$$

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 [7]. 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.

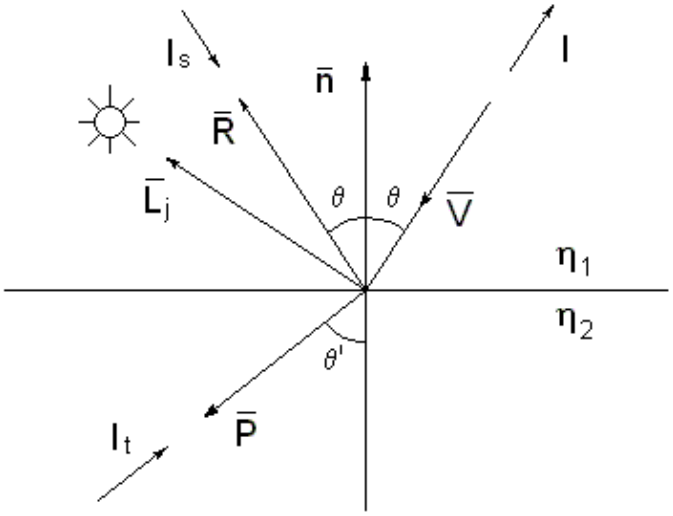

Рисунок 1.8. Зеркальное отражение и преломление
в глобальной модели освещения Уиттеда

Согласно модели Уиттеда наблюдаемая интенсивность определяется суммарной интенсивностью:

$$I = k_a I_a C + k_d I_d C + k_s I_S + k_r I_r + k_t I_t, \qquad (6)$$

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.

1.5 Описание трехмерных преобразований

Трехмерные преобразования позволяют преобразовывать объекты, изменять их угол наклона, позицию в пространстве, масштабировать объекты. Эти преобразования играют важнейшую роль в построении изображения объектов сцены.

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

Рассмотрим необходимые аффинные преобразования декартовых координат.

Сдвиг точки на dx, dy, dz по координатным осям:

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

Масштабирование относительно начала координат с коэффициентами k_x, k_y, k_z:

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

Поворот относительно осей x, y, z на угол φ выполняется следующим образом.

1) Вокруг оси x:

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

2) Вокруг оси y:

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

3) Вокруг оси 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:

$$\begin{aligned} X &= ox * \sin(a / 2) \\ Y &= oy * \sin(a / 2) \\ Z &= oz * \sin(a / 2) \\ W &= \cos(a / 2) \end{aligned} \qquad (7)$$

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:

$$\begin{pmatrix} 1-2*Y^2-2*Z^2 & 2*X*Y-2*Z*W & 2*X*Z+2*Y*W & 0 \\ 2*X*Y+2*Z*W & 1-2*X^2-2*Z^2 & 2*Y*Z-2*X*W & 0 \\ 2*X*Z-2*Y*W & 2*Y*Z+2*X*W & 1-2*X^2-2*Y^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (8)$$

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) [8].

Матрица перспективной проекции выглядит следующим образом:

$$\begin{pmatrix} \frac{1}{ar \cdot \tan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & \frac{-NearZ-FarZ}{NearZ-FarZ} & \frac{2 \cdot FarZ \cdot NearZ}{NearZ-FarZ} \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (9)$$

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:

$$\begin{pmatrix} hW & 0 & 0 & hW \\ 0 & hH & 0 & hH \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (10)$$

где W — ширина области изображения,
H — высота области изображения,
hW — W / 2,
hH — H / 2.

1.6 Методы сглаживания

Методы сглаживания позволяют устранить так называемый “лестничный эффект”, возникающий на краях отображаемых объектов вследствие его отображения без сглаживания краев. Различные реализации алгоритма сглаживания позволяют получить результат различного качества за соответствующее время, что дает возможность выбрать алгоритм сглаживания для разных задач и режимов работы программы.

1.6.1 Избыточная выборка сглаживания

Избыточная выборка сглаживания (SSAA) — простой и прямолинейный метод сглаживания. 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.

Вывод из аналитического раздела

В программе будет предусмотрено два режима работы. В первом режиме пользователь сможет добавлять новые объекты, редактировать их размер, положение, выбирать цвет, 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, поэтому в этом режиме будет использоваться алгоритм обратной трассировки лучей, как позволяющий достичь наибольшей реалистичности построенного изображения.

При отрисовке с помощью z-буфера нет необходимости выбирать сложную модель освещения, поэтому было решено выбрать локальную модель Ламберта. Учитывая простоту реализации и высокое качество получаемого изображения, предпочтительнее использовать метод избыточной выборки сглаживания, несмотря на низкую скорость работы.

Для отрисовки с помощью обратной трассировки в режиме рендера решено выбрать глобальную модель освещения Уиттеда, для достижения реалистичного изображения.

2. Конструкторский раздел

Для перехода к разработке программного обеспечения сначала необходимо сформулировать требования к разрабатываемой программе, описать схему алгоритмов, используемых в программе, выделить ключевые объекты и представления данных, спроектировать условия к работе интерфейса, а также описать процесс рендера изображения.

2.1 Алгоритм удаления невидимых граней с использованием z-буфера

В качестве алгоритма визуализации объекта был выбран алгоритм z-буфер. На Рисунке 2.1 приведена схема алгоритма z-буфера.

В процессе работы данного алгоритма, можно выделить следующие основные этапы:

1. Инициализация буфера кадра фоновым значением интенсивности, Z-буфера максимальным значением Z.

2. Инициализация и заполнение z-буфера максимальным значением z.

3. Преобразование объекта в растровую форму.

4. Для каждого пиксела в многоугольнике вычислить его глубину z(x, y) или расстояние по оси z. Сравнить вычисленную глубину z(x, y) со значением в zbuffer(x, y).

5. Если значение z(x, y) меньше значения zbuffer(x, y), тогда заменяем значение zbuffer(x, y) на z(x, y) и вычисляем новое значение интенсивности цвета для точки с координатами (x, y, z(x, y)).

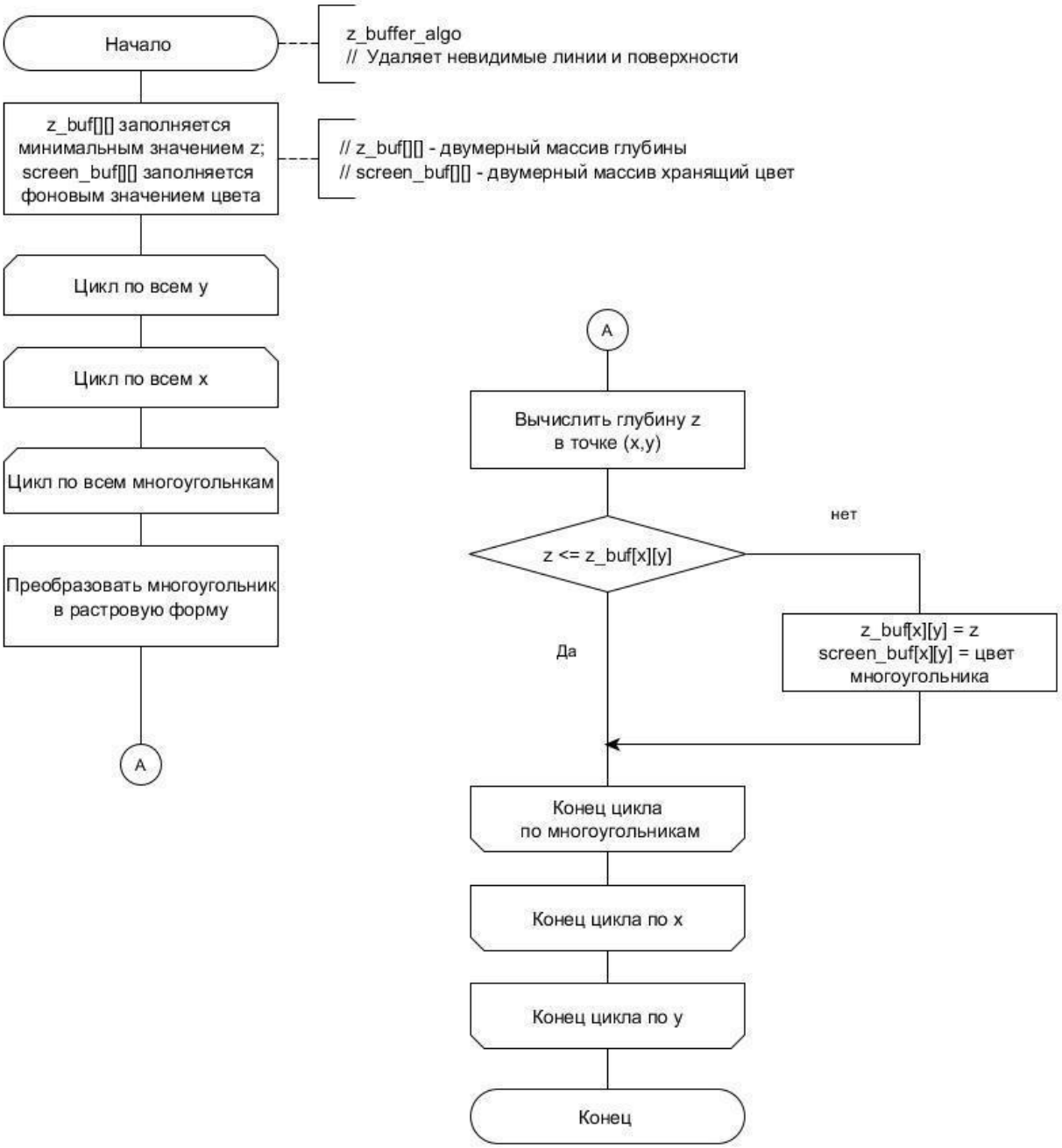

Рисунок 2.1. Схема алгоритма Z-буфера

2.2 Алгоритм обратной трассировки лучей

Алгоритм обратной трассировки лучей использует лучи света, которые уже попали в камеру и позволяет не обрабатывать не попадающие в камеру лучи, которых большинство.

2.2.1 Общий алгоритм

На Рисунке 2.2 приведена схема алгоритма обратной трассировки лучей.

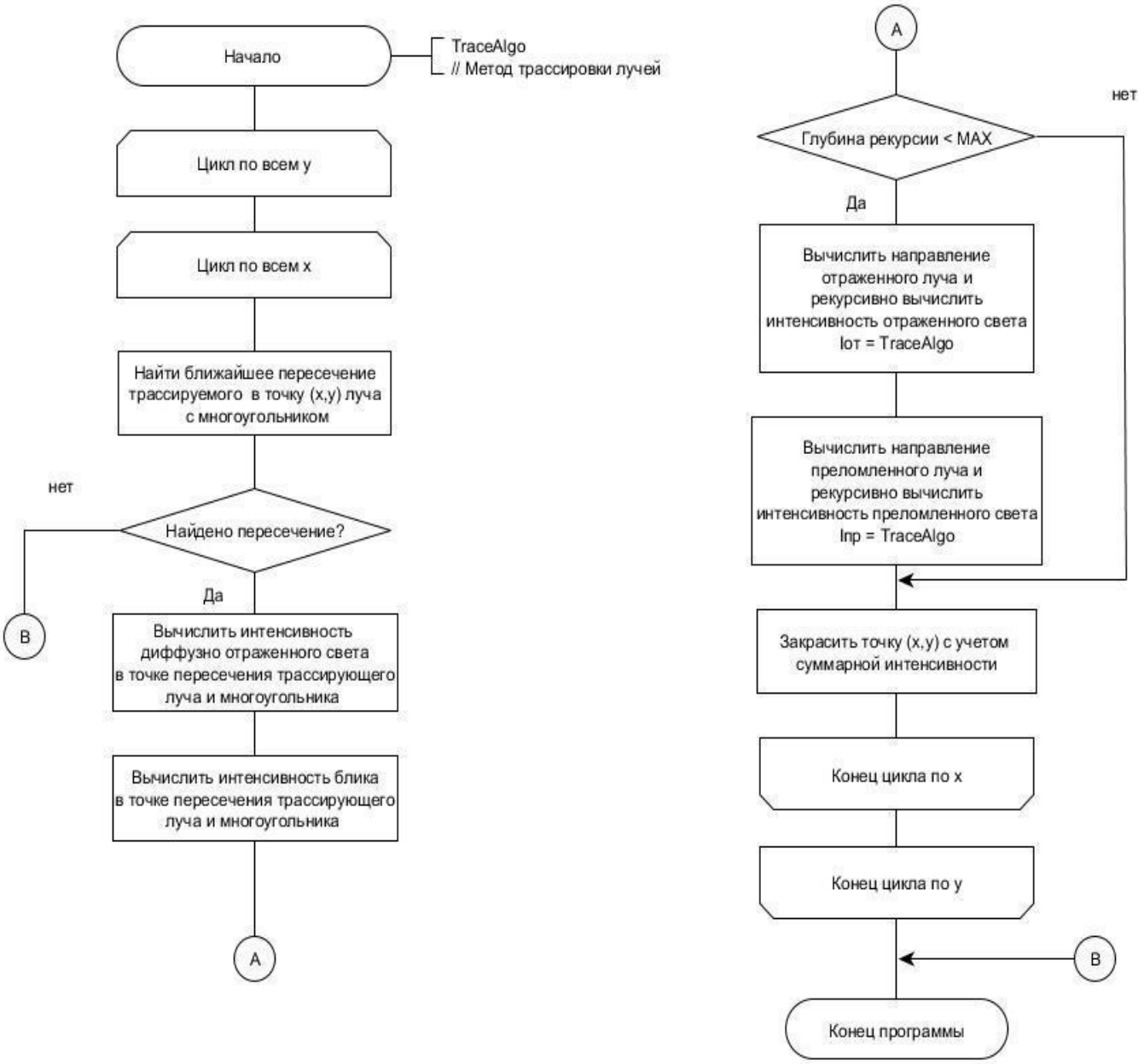

Рисунок 2.2. Схема алгоритма обратной трассировки лучей

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.

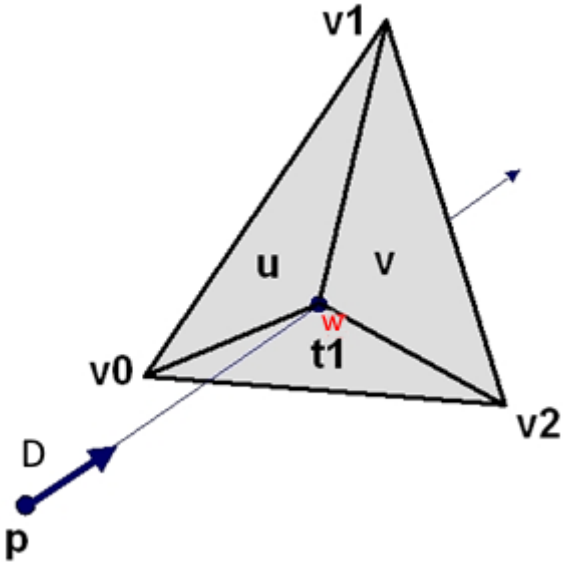

Рисунок 2.3: Барицентрический тест

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.

$$w(u,v) = (1 - u - v) * v1 + u * v2 + v * v0 \qquad (10)$$

$$w(t) = p + t * \vec{D} \qquad (11)$$

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).

$$p + t * d = (1 - u - v) * v1 + u * v2 + v * v0 \quad (12)$$

Проведя алгебраические преобразования, ответ можно получить в следующем виде:

$$[t\ u\ v\ ] = \frac{1}{dot(P, E1)} * [dot(Q, E2)\ dot(P, T)\ dot(Q, D)\ ] \quad (13)$$

где $\vec{E1} = v1 - v0$

$\vec{E2} = v2 - v0$

$\vec{T} = p - v0$

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

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

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:

$$dot(P - C, P - C) = r^2 \quad (14)$$

$$P = O + t\vec{D} \quad (15)$$

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:

$$dot\left(\vec{OC} + t\vec{D}, \vec{OC} + t\vec{D}\right) = r^2 \quad (16)$$

$$t^2 dot\left(\vec{D},\ \vec{D}\right) + 2 * t * dot(\vec{OC}, \vec{D}) + dot(\vec{OC}, \vec{OC}\ \big) - r^2 = 0 \qquad (17)$$

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 закраски

Алгоритм закраски позволяет закрашивать замкнутые области, что дает возможность управлять цветом размещаемых объектов на сцене. Различные методы закраски позволяют использовать разные алгоритмы для разных режимов работы отрисовки программы.

2.3.1 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 2.3 приведена схема алгоритма закраски методом Гуро.

Рисунок 2.3. Схема алгоритма закраски методом Гуро

2.4 Модель освещения

Модель освещения играет важную роль в построении реалистичного изображения, созволяя управлять не только глобальным освещением, но и от локальных источников света, размещаемых на сцене.

2.4.1 Нахождение отраженного и преломленного лучей

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

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

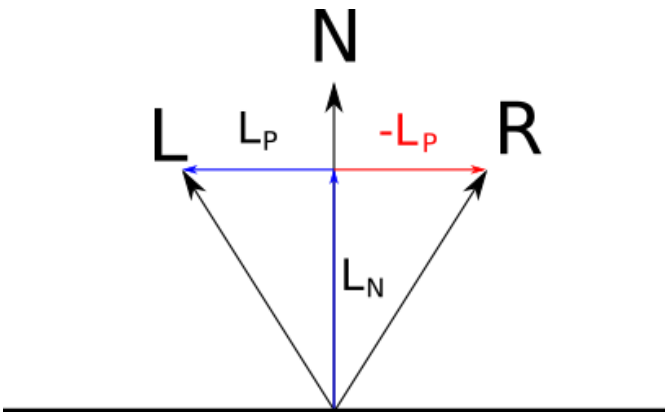

Рисунок 2.4. Разложение вектора падающего луча

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

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

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

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:

$$\eta_i \sin sin\, \theta_i = \eta_t \sin sin\, \theta_t \qquad (18)$$

где 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:

$$\vec{P} = \frac{n_1}{n_2} * \vec{L} + \left(\frac{n_1}{n_2} * \cos cos\left(\theta_i\right) - \sqrt{1 - sin^2\left(\theta_t\right)}\right) * \vec{N} \quad (19)$$

При этом $sin^2\left(\theta_t\right) = \left(\frac{n_1}{n_2}\right)^2 * sin^2(\theta_i)$, где θ_i – угол падения, θ_t – угол преломления, n_2 и n_1 – абсолютные показатели преломления двух сред.

2.4.2 Расчет интенсивностей

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

$$I_d = k_d * \sum I_i * (\vec{n}, \vec{L}_i) \quad (20)$$

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;

$\vec{L}_i$ — 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):

$$I_s = k_s \sum I_i * (\vec{S}, \vec{R}_i)^N \quad (21)$$

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;

$\vec{R}_i$ — единичный вектор, задающий направление отраженного луча от i-го источника;

N — степень, аппроксимирующая пространственное распределение зеркально отраженного света.

Общую интенсивность можно определить по формуле:

$$I_r = k_a \sum I_{ia} + k_d \sum I_i * (\vec{n}, \vec{L}_i) + k_s \sum I_i * (\vec{S}, \vec{R}_i)^N + k_r I_r + k_t I_t \quad (22)$$

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.

2.5 Алгоритм устранения лестничного эффекта

На Рисунке 2.6 приведена схема алгоритма сглаживания

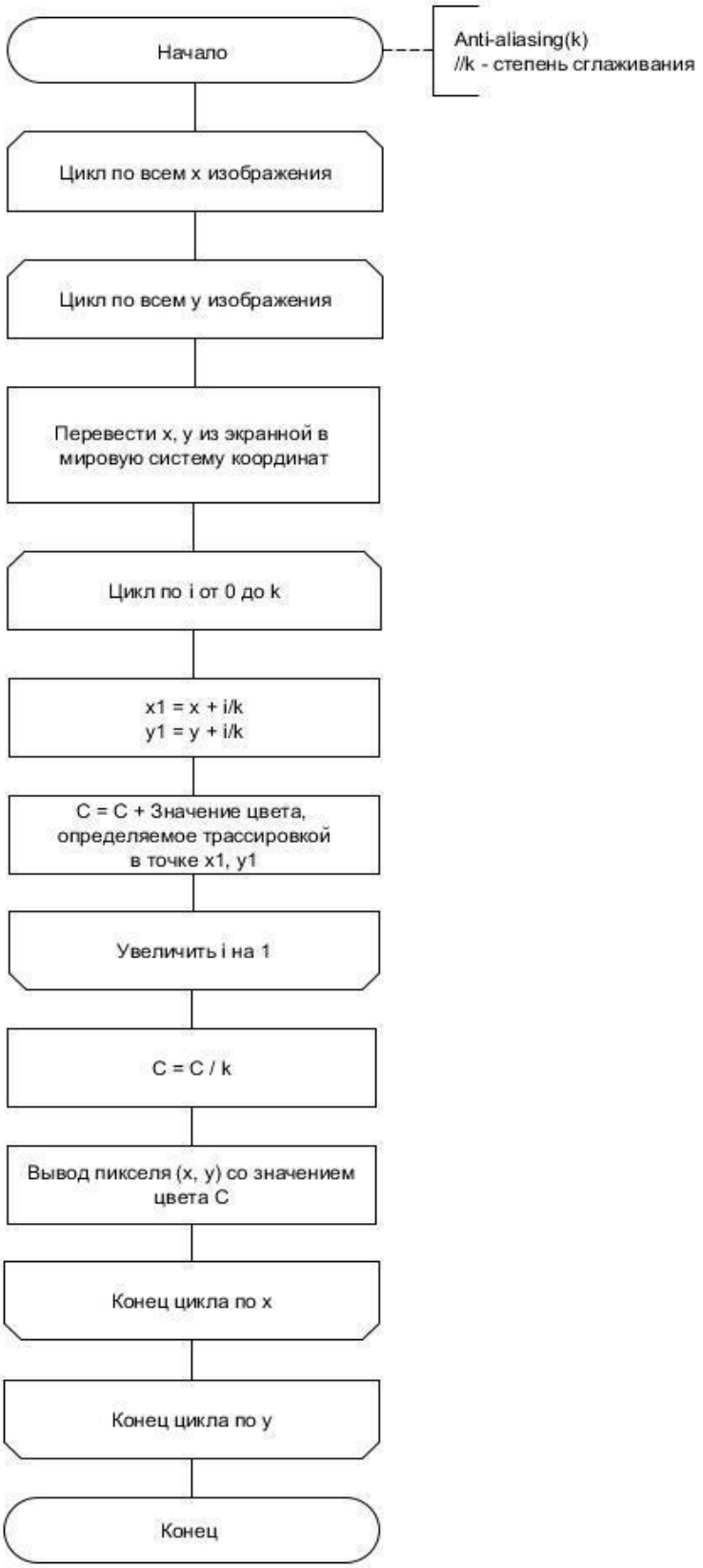

Рисунок 2.6. Схема алгоритма сглаживания

2.6 Выделенные типы и структуры данных

Для удобства разработки и систематизации программного продукта в рамках поставленной задачи выделены программные объекты (классы), каждый из которых наделен определенным функционал в соответствии с парадигмой ООП [3]: полная схема доступна на Рисунок 2.5, детальное описание каждого класса на рисунке 2.7.1.1-2.7.9.2 .

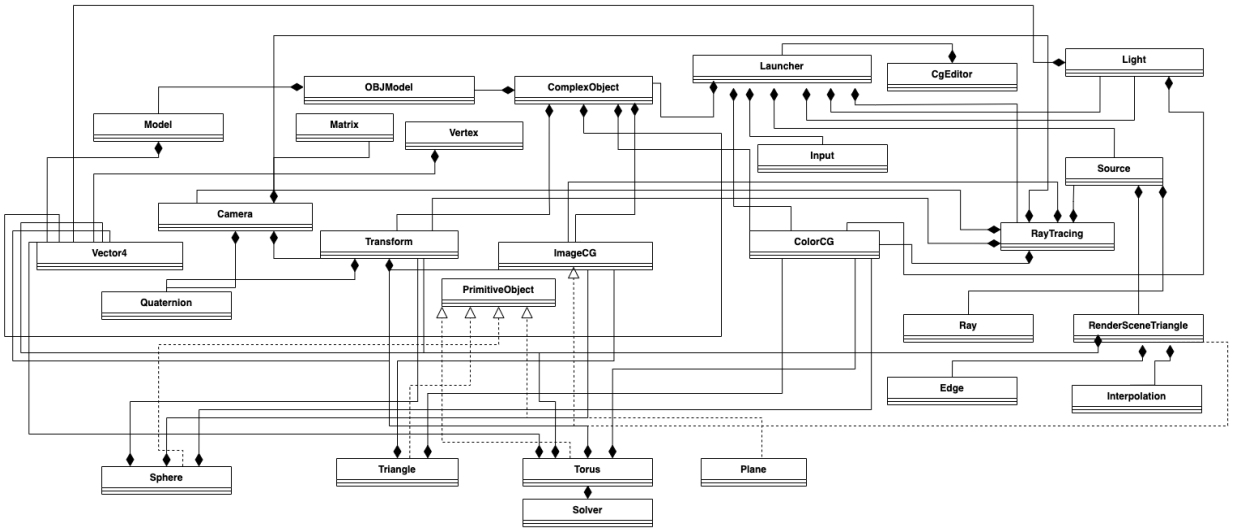

Рисунок 2.7. Диаграмма взаимодействия объектов

2.6.1 Описание объекта ComplexObject

ComplexObject — класс, представляющий размещаемый и отображаемый графический объект, считываемый из файла формата .obj (см пункт 2.7 и листинг 2.8) (см. Рисунок 2.7.1).

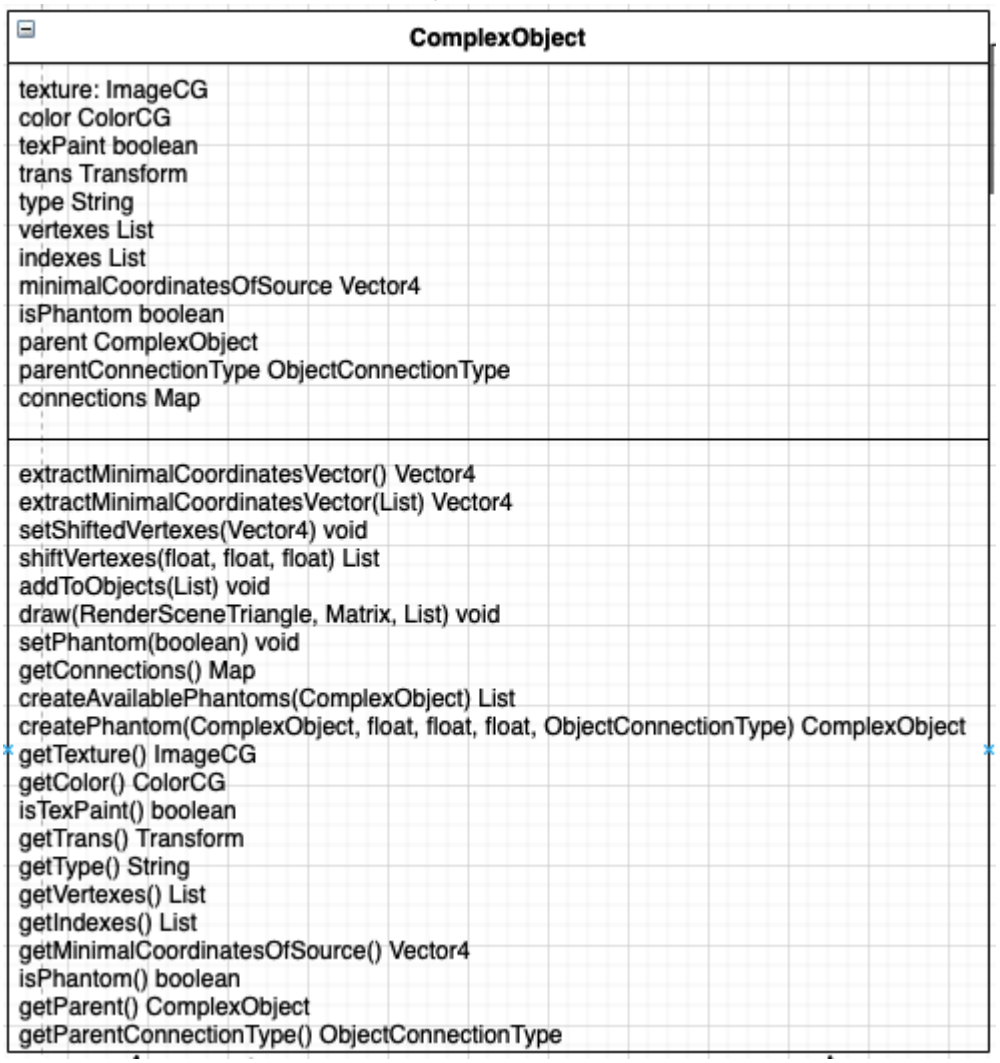

Рисунок 2.7.1. Описание полей и методов класса ComplexObject

2.6.2 Описание объекта Launcher

Launcher — класс, создающий скелет интерфейса и устанавливающий обработчики действий на его компоненты. (см. рис 2.7.2).

Рисунок 2.7.2. Описание полей и методов класса Launcher

2.6.3 Описание объекта Vertex

Vertex — класс, назначение которого хранить и предоставлять информацию о точке в пространстве, являющейся вершиной объекта (см. рис 2.7.3). Представляется объектами класса Vector4 (см. рис 2.7.3.1).

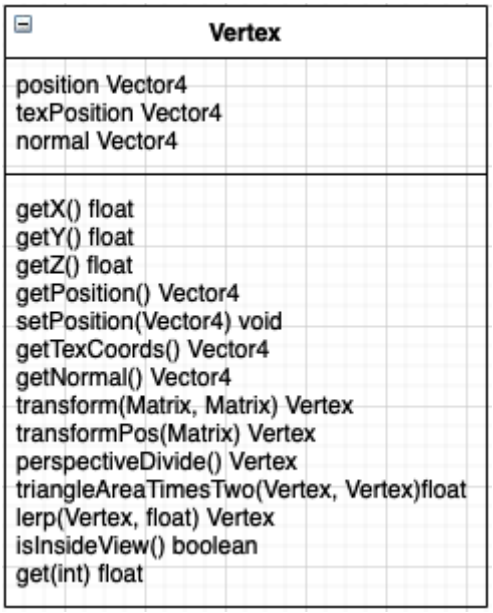

Рисунок 2.7.3. Описание полей и методов класса Vertex

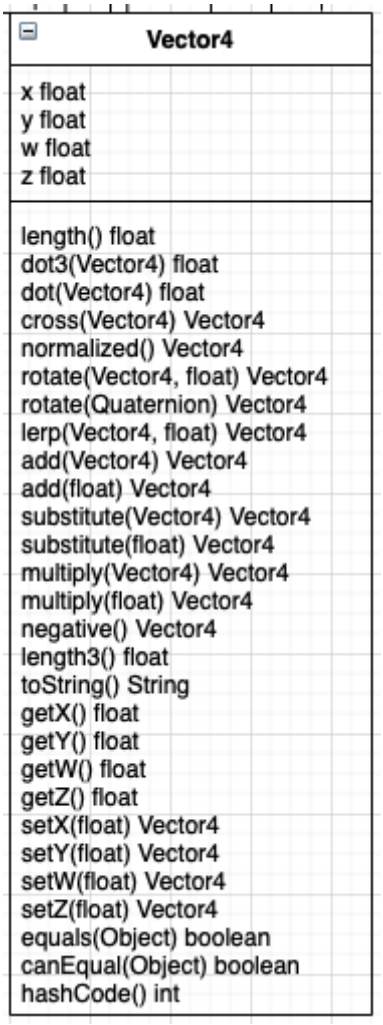

Рисунок 2.7.3.1. Описание полей и методов класса Vector4

2.6.4 Описание объектов OBJModel и Model

OBJModel — класс, загружающий объект из файла и представляющий его в удобного для дальнейшей работы формате, определяемый классом Model. (см. рис 2.7.4).

Model — класс, назначение которого представлять загруженный объект в удобной для дальнейшей работы формате. (см. рис 2.7.4.1)

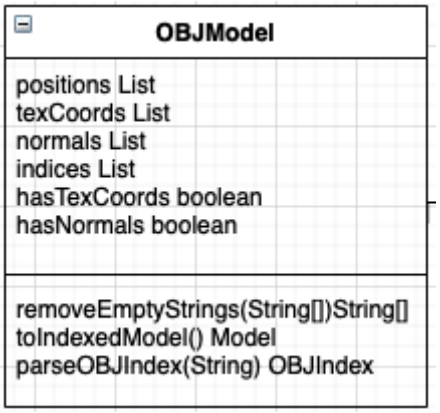

Рисунок 2.7.4. Описание полей и методов класса OBJModel

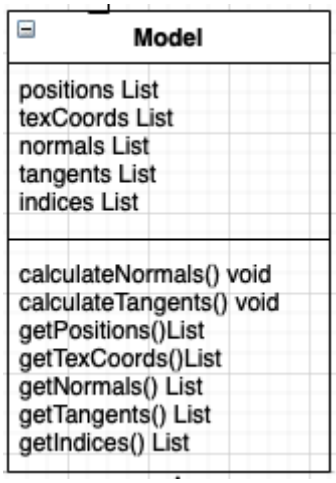

Рисунок 2.7.4.1. Описание полей и методов класса Model

2.6.5 Описание объекта ImageCG

ImageCG — класс, задающий текстуру объекта сцены. Может быть как однотонным, так и быть пользовательской текстурой, загруженной из файла. (см. рис 2.7.5).

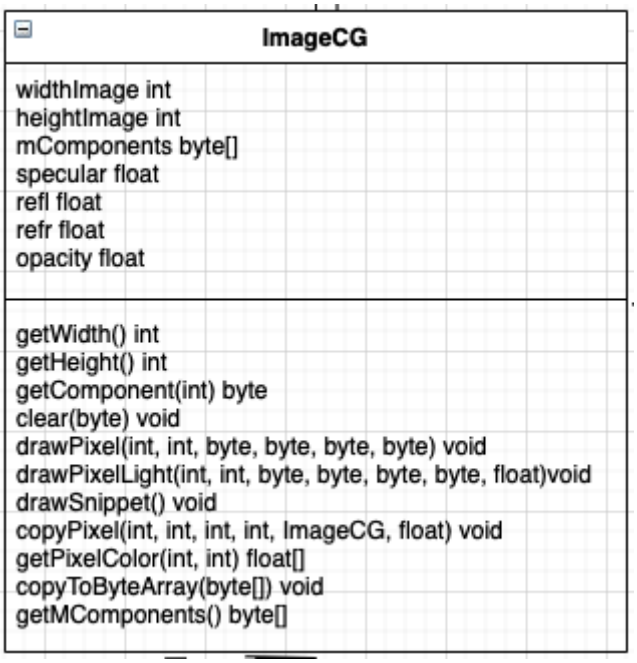

Рисунок 2.7.5. Описание полей и методов класса ImageCG

2.7 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 нормалях.

Формат файла следующий. В файле должны быть последовательно расположены три сегмента.

1. Список вершин с координатами (x,y,z), пример: *v 0.74 0.26 1*

2. Координаты нормалей (x,y,z), пример: *vn 0.7 0.000 0.3*

3. Определения поверхности (сторон) задаются в формате i1/i2, где i1 — индекс координаты вершины, i2 — индекс координаты нормали: *f 2/4/4 3/5/5 1/3/5*

Пример задания треугольника приведен в листинге 2.8.

Листинг 2.8. Пример входного файла с расширением .obj.

```
1. v 1.000000 0.000000 -1.000000
2. v 1.000000 0.000000 1.000000
3. v -1.000000 0.000000 1.000000
4.
5. vt 0.748573 0.750412
6. vt 0.500149 0.750166
7. vn 0.000000 1.000000 0.000000
8.
9. f 1/1/1 2/9/1 3/1/1
```

Вывод из конструкторского раздела

В данном разделе были описаны на основе сформулированных требований к программе выделенные объекты представления данных и их функционал, приведены схемы работы алгоритмов Z-буфера и обратной трассировки лучей, алгоритм закраски и модели освещения, которые позволяют перейти в разработке программного обеспечения, решающего задачу на курсовой проект.

3. Технологический раздел

После разработки программы на основе анализа, проведенного для алгоритмов и структур данных и выделения объектов и абстракций, необходимо перейти к рассмотрению требований разрабатываемого программного обеспечению, средств, использующихся в процессе разработки для реализации поставленных задач.

3.1 Требования к программному обеспечению

Программное обеспечение обязано реализовывать поставленную на задачу. В программном продукте обязан присутствовать пользовательский интерфейс для управления объектами сцены. Программа не должна аварийно завершаться и не должна задействовать иррационально большой объем оперативной памяти или времени и ресурсов процессора.

3.2 Обоснование выбора языка, среды и платформы программирования

Для реализации поставленной задачи был выбран язык Java, 8 версии. Это мощный объектно-ориентированный язык, 8 версия которого распространяется по лицензии GNU GPL. Был выбран по следующим причинам.

1. Высокая скорость выполнения
2. Присутствует поддержка ООП
3. Широкий функционал стандартных библиотек для построения интерфейса

Благодаря возможностям настройки размера доступной программе памяти в heap, возможность работы с off-heap памятью, и поддержки многопоточности, возможно произвести полноценный анализ производительности программы в разных условиях работы.

3.3 Описание интерфейса программы

Пользователю доступна возможность добавить готовый объект из списка примитивных деталей. Для каждого из объектов пользователь может назначать положение, углы поворота, масштаб и цвет при настройке параметров при размещении, а также при помощи курсора разместить объект на сцене, прикрепив к имеющемуся. Управление камерой осуществлено с помощью клавиш клавиатуры “W”, “A”, “S”, “D”, “Space”, “Ctrl”. При выборе места размещения детали-объекта, пользователем выбирается место из доступных в сцене, а также отображаются фантомы размещаемой детали в доступных к размещению местах.

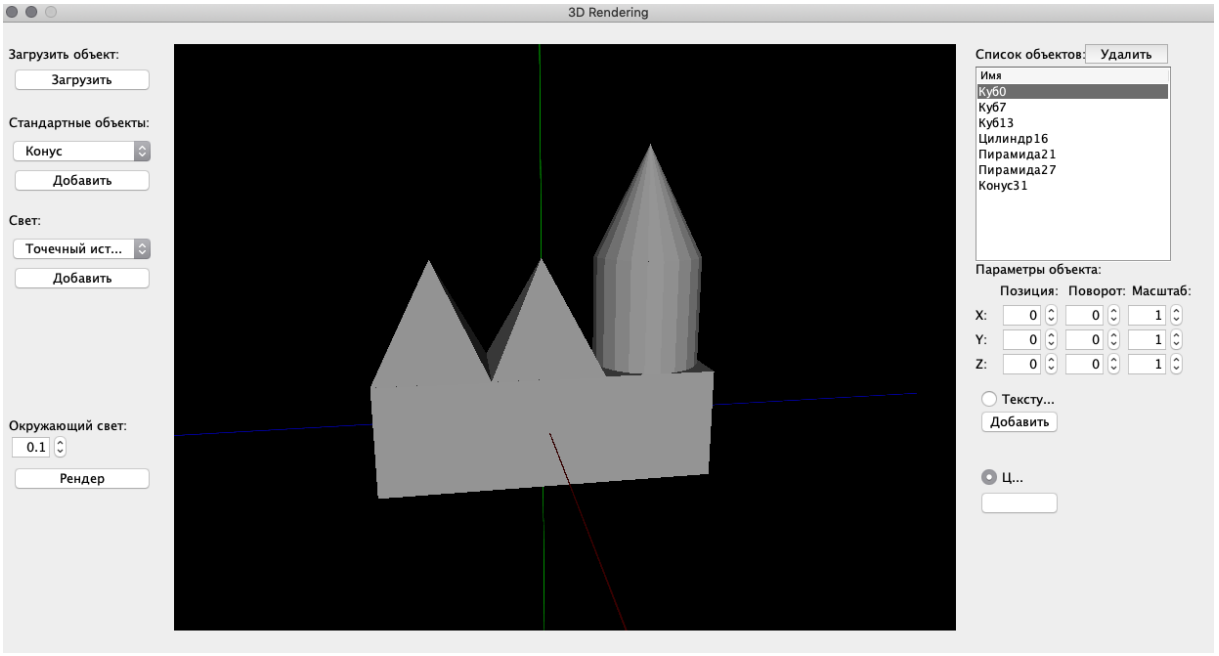

Рисунок 3.1. Пример работы программы

3.4 Пользовательский интерфейс

Интерфейс программы представляется на русском языке, Разбитым на несколько областей: левая область создания (см. рисунок 3.2.1), сцена (см. рисунок 3.2.3) и правая область для редактирования объектов сцены (см. рисунок 3.2.2).

После запуска программы пользователю отображается пустое пространство области отображения объектов и все доступные элементы интерфейса для добавления, удаления и редактирования объектов.

Интерфейс левой панели (см. рисунок 3.2.1) представлен следующими секциями.

1. Секцией “Загрузить объект” с кнопкой для загрузки объекта, . Через диалоговое окно системы, пользователь выбирает файл формата .obj и программа загружает этот файл, предлагая пользователю разместить объект, считанный из файла в доступном месте в пространстве холста.

2. Секцией “Стандартные объекты” с выпадающим списком и кнопкой “Добавить”. В списке выбирается один из стандартных объектов и после нажатия кнопки “Добавить”, предлагается к размещению в пространстве холста.

3. Секцией “Свет” с выпадающим списком и кнопкой “Добавить”. При нажатии кнопки “Добавить”, на сцену добавляется источник света, который будет отображаться как объект в режиме проектирования и будет использован как источник освещения в режиме рендера.

4. Секцией “Окружающий свет” с полем ввода числового значения и кнопкой “Рендер”. При нажатии кнопки “Рендер”, программа переходит в режим рендера, если была в режиме проектирования и обратно при повторном нажатии. В режиме рендера будет учитываться значение в числовом поле.

Интерфейс правой панели (см. рисунок 3.2.2) представлен следующими секциями.

1. Секцией “Список объектов” с кнопкой “Удалить” и списком размещенных объектов. При выборе объекта из списка, по нажатии кнопки “Удалить”, можно удалить объект из пространства сцены.

2. Секцией “Параметры объекты” с полями для ввода доступных для ввода параметров объекта. Для редактирования параметров, пользователю необходимо выбрать желаемый объект из списка выше.

3. Секцией “Текстура” с кнопкой “Добавить”. При нажатии кнопки “Добавить”, к объекту добавляется текстура, из выбранного файла.

4. Секцией “Цвет” с кнопкой выбора цвета. При нажатии кнопки, открывается меню выбора цвета для выбранного объекта.

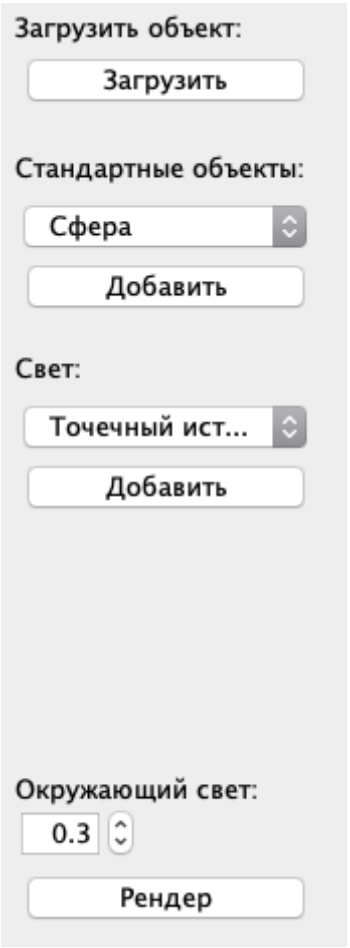

Рисунок 3.2.1. Левые секции интерфейса программы

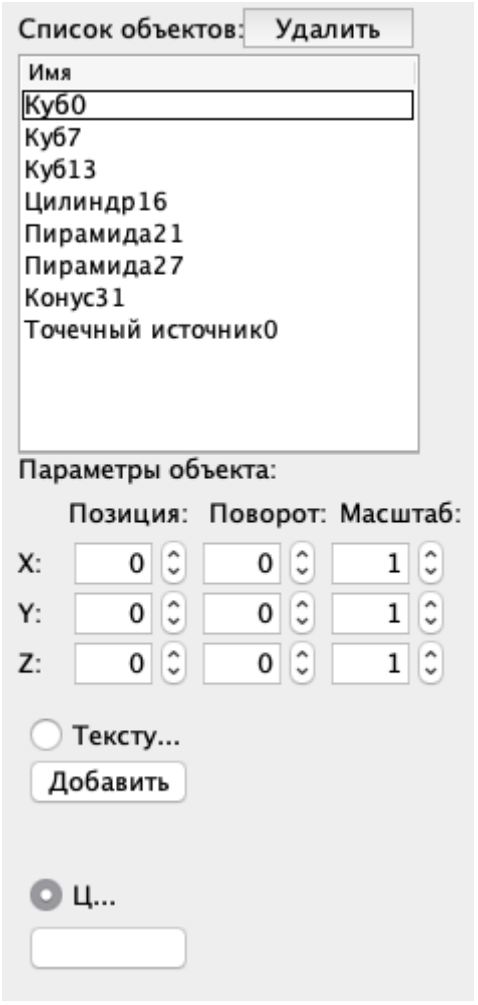

Рисунок 3.2.2. Правые секции интерфейса программы

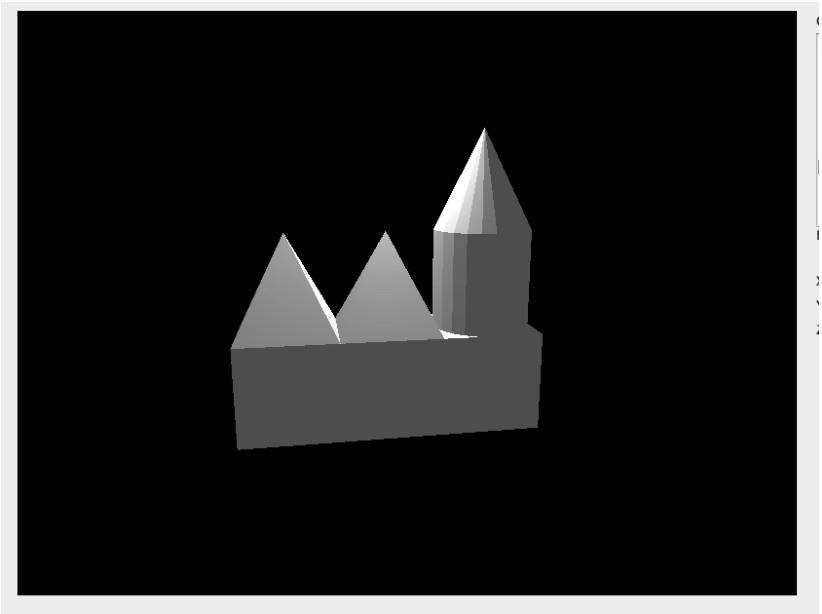

Рисунок 3.2.3. Холст сцены в режиме рендера
(с источником света над моделью)

3.5 Реализация алгоритмов

На основе алгоритмов, рассмотренных в аналитическом разделе реализованы алгоритмы отрисовки, перемещения, поворота, масштабирования объектов, а также закраска и применение трассировки лучей для освещения.

3.5.1 Перемещение объекта в пространстве

Перемещение объекта в пространстве происходит путем изменения координат вектора, описывающего его позицию (см. листинг 3.5.1).

Листинг 3.5.1. Метод перемещения объекта ComplexObject

```
1. // Перемещение выбранного объекта в пространстве
2. complexObjectList.get(i).trans =
   complexObjectList.get(i).trans.setPos(
3. new Vector4(Float.parseFloat(spinXtrans.getValue().toString()),
4.              Float.parseFloat(spinYtrans.getValue().toString()),
5.              Float.parseFloat(spinZtrans.getValue().toString())));
```

3.5.2 Поворот объекта в пространстве

Поворот объекта в пространстве происходит путем нормализации кватерниона из заданных значений (см. листинг 3.5.2).

Листинг 3.5.2. Метод поворота объекта ComplexObject

```
1. // Поворот выбранного объекта в пространстве
2. complexObjectList.get(i).trans = complexObjectList.get(i)
3.           .trans.rotateFromNull(
4. Float.parseFloat(spinXrot.getValue().toString()),
5. Float.parseFloat(spinYrot.getValue().toString()),
6. Float.parseFloat(spinZrot.getValue().toString()));
```

3.5.3 Поворот объекта в пространстве

Масштабирование объекта в пространстве происходит путем задания вектора из заданных значений (см. листинг 3.5.3).

Листинг 3.5.3. Метод масштабирования объекта ComplexObject

```
1. // Масштабирования выбранного объекта в пространстве
2. complexObjectList.get(i).trans =
   complexObjectList.get(i).trans.setScale(
3. new Vector4(Float.parseFloat(spinXscale.getValue().toString()),
4. Float.parseFloat(spinYscale.getValue().toString()),
5. Float.parseFloat(spinZscale.getValue().toString())))
```

3.5.4 Применение всех трансформаций объекта в пространстве

Поворот, масштабирование и перемещение объекта при отрисовки происходят умножением соответствующих матриц и созданием матрицы трансляций (см. листинг 3.5.4).

Листинг 3.5.4. Метод применения трансформаций объекта ComplexObject

```
1. // Применение трансформаций объекта в пространстве
2. // Создание матрицы трансляций из положения объекта
3. Matrix translationMatrix = new Matrix()
4.                 .createMovement(position.getX(),
   position.getY(), position.getZ());
5. // Матрицы поворота
6. Matrix rotationMatrix = rotation.toRotationMatrix();
7. // Матрица масштабирования
8. Matrix scaleMatrix = new Matrix()
9.                 .createScale(scale.getX(), scale.getY(),
   scale.getZ());
10.    // Умножение всех матриц
11.   return
   translationMatrix.multiply(rotationMatrix.multiply(scaleMatrix)
   );
```

3.5.5 Отрисовка объекта в пространстве

При отрисовке сцены, для всех объектов вызывается метод отрисовки (см. листинг 3.5.5,).

Листинг 3.5.5. Метод применения трансформаций объекта ComplexObject

```
1. // Применение трансформаций объекта в пространстве
2. public void draw(RenderSceneTriangle context, Matrix
   viewProjection, List<Source> lightsArray) {
3. Matrix transform = trans.getTransformation();
4. // Умножение проекции на матрицу трансформаций
5. Matrix mvp = viewProjection.multiply(transform);
6. for (int i = 0; i < indexes.size(); i += 3) {
7.     // Отрисовка очередного треугольника
8.     context.drawTriangle(
9.    vertexes.get(indexes.get(i)).transform(mvp, transform),
10.      vertexes.get(indexes.get(i + 1)).transform(mvp,
   transform),
11.      vertexes.get(indexes.get(i + 2)).transform(mvp,
   transform),
12.      texture,
13.      color,
14.      texPaint,
15.      lightsArray,
16.      isPhantom);
17.          }
18.      }
```

Листинг 3.5.6. Метод применения трансформаций объекта ComplexObject

```
19.// Применение трансформаций объекта в пространстве
20.   public void drawTriangle(Vertex v1, Vertex v2, Vertex v3,
   ImageCG texture, ColorCG color, boolean texPaint, List<Source>
   lightsArray, boolean isPhantom) {
21.   // Проверка на полностью внутри
22.   if (v1.isInsideView() && v2.isInsideView() &&
   v3.isInsideView()) {
23.         fillTriangle(v1, v2, v3, texture, color, texPaint,
   lightsArray, isPhantom);
24.          return;
25.    }
26.    // Массив вершин
27.    List<Vertex> vertices = new ArrayList<>();
28.    List<Vertex> additionalList = new ArrayList<>();
29.
30.    vertices.add(v1);
31.    vertices.add(v2);
32.    vertices.add(v3);
33.    // Проверка на обрезанность
34.    if (clipPolygon(vertices, additionalList, 0) &&
35.        clipPolygon(vertices, additionalList, 1) &&
36.        clipPolygon(vertices, additionalList, 2)) {
37.        Vertex initialVertex = vertices.get(0);
38.
39.        for (int i = 1; i < vertices.size() - 1; i++) {
40.           // Закраска треугольника
41.            fillTriangle(initialVertex, vertices.get(i),
   vertices.get(i + 1), texture, color, texPaint, lightsArray,
   isPhantom);
42.              }
43.          }
44.      }
```

Вывод из технологического раздела

В данном разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, приведены описания структур и поведения объектов, приведены листинги операций, вызываемых интерфейсом.

4 Исследовательская часть

Исследования, выполненные и приведенные в данном разделе, были проведены 10 раз, результаты являются средним арифметическим всех измерений. Измерялось время выполнения конкретного потока, программы (соответствующего системному) для достижения наиболее точных показателей потребления процессорного времени при работе программы.

4.1 Технические характеристики устройства

В Таблице 4.1 приведены технические характеристики устройства (ЭВМ), использованного для проведения экспериментов на базе разработанного программного обеспечения. Стоит заметить, что на ЭВМ используется технология Hyper-Threading [9], расширяющая количество потоков процессора до 4.

Таблица 4.1: Технические характеристики устройства, на котором проводилось тестирование программного обеспечения

ОС	macOS 10.14.6 64-bit
ОЗУ	2x4 Gb 1600 MHz DDR3
Процессор	2,3 GHz Intel Core i7 (2 ядра, 4 с технологией Hyper-Threading)

4.2 Зависимость времени генерации детали от количества треугольников для стандартного объекта Сфера

Задача эксперимента — наблюдение зависимости изменения времени отрисовки сцены от количества треугольников стандартных объектов Сфера, размещенных на сцене. В стандартном объекте типа Сфера - 7080 треугольников. Данные аппроксимированы из 100 экспериментов для каждого количества. Точные результаты исследования представлены в таблице 4.2. Графическое представление таблицы доступны на рисунок 4.2.1.

Таблица 4.2. Зависимость времени визуализации сцены от количества треугольников

Количество треугольников	Время визуализации, мс
7080	13
14160	14
28320	19
56640	29
113280	63

Рисунок 4.2.1. Исследование зависимости времени визуализации сцены от количества треугольников объектов сцены Сфера (ось абсцисс - логарифмическая)

4.3 Зависимость времени генерации детали от количества треугольников для стандартного объекта Куб

Задача эксперимента — наблюдение зависимости изменения времени отрисовки сцены от количества треугольников стандартных объектов Куб, размещенных на сцене. В стандартном объекте типа Куб - 12 треугольников. Данные аппроксимированы из 100 экспериментов для каждого количества. Точные результаты исследования представлены в таблице 4.3. Графическое представление таблицы доступны на рисунок 4.3.1.

Таблица 4.3. Исследование зависимости времени визуализации сцены от количества треугольников стандартных объектов Куб

Количество треугольников	Время визуализации, мс
12	5
24	5
36	6
48	6
96	6
192	6
384	7

804	9
1668	10
3108	12

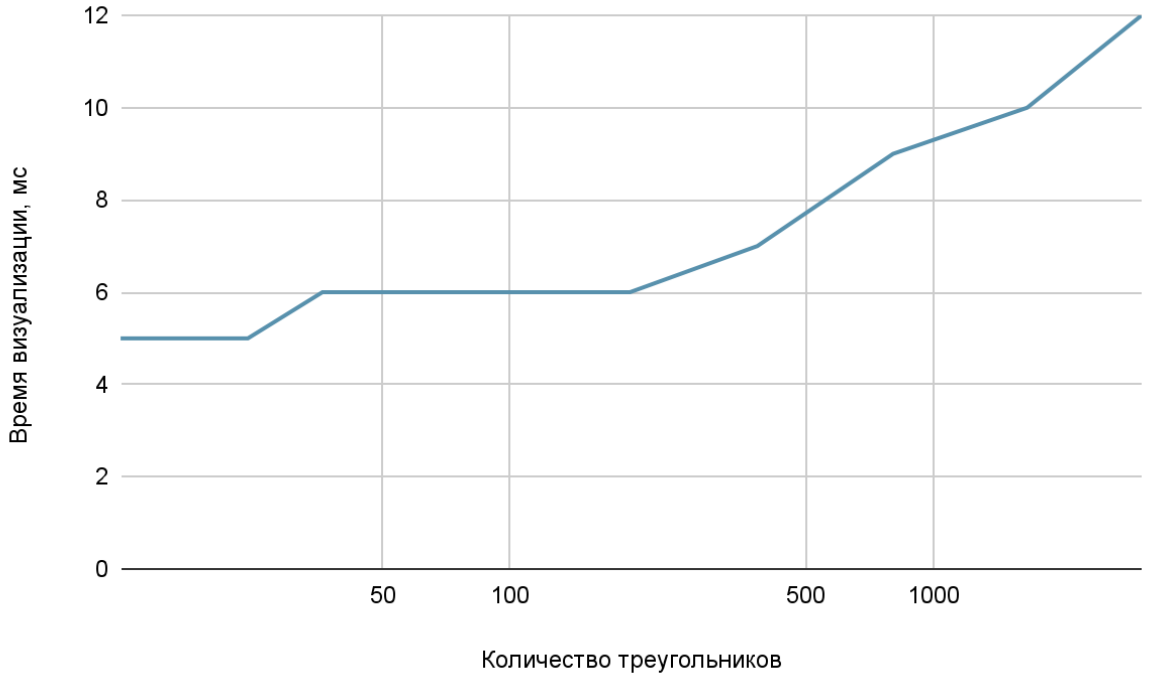

Рисунок 4.3.1. Исследование зависимости времени визуализации сцены от количества треугольников объектов сцены Куб (ось абсцисс - логарифмическая)

4.4 Вывод из исследовательского раздела

Из проведенных исследований можно сделать вывод о том, что увеличении количества треугольников, скорость визуализации сцены увеличивается не линейно. При увеличении количества треугольников более 10 000, наблюдающийся прирост времени визуализации становится подобен экспоненциальному.

Заключение

Цель проекта была достигнута - разработана программа для создания трехмерных графических сцен из трехмерных геометрических объектов, схожих с детским конструктором и их визуализация с учетом выбранного цвета, позиции камеры. Предусмотрены условия размещения каждого объекта с учетом остальных, уже присутствующих в сцене объектов.

Все поставленные задачи были решены:

- проведен анализ существующих алгоритмов удаления невидимых линий и поверхностей, закраски, а также моделей освещения и выбраны подходящие для выполнения проекта;
- произведены основные математические расчеты для реализации выбранных алгоритмов;
- выбрана подходящая программная платформа для реализации поставленной задачи;
- реализовано программное обеспечение для моделирования детского конструктора и его интерфейс.

Литература

1. Иванов В.П., Батраков А.С. Трехмерная компьютерная графика / Под ред. Г.М. Полищука. — М.: Радио и связь, 1995. — 224 с.
2. Д. Роджерс, Дж. Адамс «Математические основы компьютерной графики» - Москва, «Мир», 2001г.- 604 с.
3. Шикин Е. В., Боресков А. В. Зайцев А. А. Начала компьютерной графики. – М.: ДИАЛОГ-МИФИ, 1993. – 138 с.
4. Обратная трассировка лучей. Фролов. В, Фролов А. [Электронный ресурс]. – Режим доступа: http://ray-tracing.ru/articles164.html
5. Построение изображений ландшафта в реальном времени [Электронный ресурс]. – Режим доступа: https://studbooks.net/2248060/informatika/odnotonnaya_zakraska_metod_graneniya
6. Закраска Фонга [Электронный ресурс]. – Режим доступа: http://stratum.ac.ru/education/textbooks/kgrafic/additional/addit26.html
7. Простые модели освещения [Электронный ресурс]. – Режим доступа: http://grafika.me/node/344
8. Проекция перспективы [Электронный ресурс]. – Режим доступа: https://triplepointfive.github.io/ogltutor/tutorials/tutorial12.html
9. Ватутин Э. И., Титов В. С. Особенности реализации технологии «hyper-threading» в процессорах Intel «Pentium 4» на примере выполнения кода разного типа, 2005