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 «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Графический редактор композиций тел вращения»

Студент ИУ7-45Б (Группа) _____ (Подпись, дата) Романов С. К. (И. О. Фамилия)

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

2022 г.

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)

УТВЕРЖДАЮ

Заведующий кафедрой
ИУ7
(Индекс)
_____ И.В.Рудаков
(И.О.Фамилия)
« ___ » _____ 2022 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине _____Компьютерная графика_____

_____Графический редактор композиций тел вращения_____
(Тема курсового проекта)

Студент _____Романов Семен Константинович, ИУ7-45Б_____
(ФИО, индекс группы)

График выполнения проекта: 25% к _4_ нед., 50% к _7_ нед., 75% к _11_нед., 100% к 14 нед.

1. Техническое задание

Разработать программное обеспечение для моделирования и редактирования композиции тел вращения. Интерфейс должен позволить выбрать модель для построения из предложенного набора, задавать параметры для построения тела вращения, должна быть возможность изменений физических параметров (ширина, высота), положения в пространстве (перемещения, поворот, масштабирование), редактирования (отсечение, закраска нужным цветом) тел. Проектируемый программный продукт должен позволять производить булевы операции над телами вращения, размещение одного источника света, просмотра сцены (состоит не более чем из 4-ех моделей) путем перемещения и поворота камеры.

2. Оформление курсовой работы

2.1. Расчетно-пояснительная записка на 25-30 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательская часть, заключение, список литературы, приложения.

2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.). На защиту проекта должна быть представлена презентация, состоящая из 10-15 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО, результаты проведенных исследований.

Дата выдачи задания «____» _____ 20__ г.

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

Студент _____ Романов С .К.
(Подпись, дата) (Фамилия И.О.)

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие термины с соответствующими определениями.

Closed Triangular Irregular Network. — Замкнутая ориентированная поверхность, разбитая на треугольники со следующими условиями:
а) каждая точка поверхности принадлежит хотя бы одному треугольнику;
б) два треугольника могут пересекаться только в одной вершине или по целому ребру. [1]

Constructive Solid Geometry — Метод, используемый в твердотельном моделировании. Конструктивная геометрия твердого тела позволяет разработчику моделей создавать сложную поверхность или объект, используя логические операторы для объединения более простых объектов, потенциально создавая визуально сложные объекты путем объединения нескольких примитивных объектов.[2]

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

CSG — Constructive Solid Geometry - конструктивная геометрия твердого тела

CTIN — Closed Triangular Irregular Network - замкнутая Треугольная Нерегулярная Сеть.

ПО — Программное обеспечение

Введение

Целью данного курсового проекта является разработка ПО, визуализирующие различные композиции тел вращения. Это включает в себя такие вещи, как

1. Воссоздание сложных композиций путем объеденения, пересечения и исключения различных примитивов
2. Создание тел вращения путем вращения выше обозначенных композиций вокруг установленной оси
3. Представление полноценной освещенной сцены с композициями объектов
4. Обзор рабочей сцены из различных ее участков посредством нескольких камер
5. Интерактивность с различными объектами сцены, такими как:
   (a) Камеры
   (b) Модели
   (c) Композиты

Для достижения поставленных целей необходимо решить следующие задачи:

- Описать структуру трехмерной сцены;
- Реализовать оптимальные алгоритмы представления, преобразования и визуализации твердотельной модели;
- Реализовать аппаратную обработку всех элментов сцены;
- Спроектировать процесс моделирования сцены;
- Описать использующиеся структуры данных;
- Определить средства программной реализации;

- Провести экспериментальные замеры временных характеристик разработанного ПО.

В ходе курсовой работы будут затронуты такие темы, как:

– Понятия о телах и поверхностях вращения

– Удаление невидимых линий и поверхностей

– Бинарные операции над телами

– Методы закрашивания объемных тел

– Преимущества и особенности языка Rust

1 Аналитический раздел

1.1 Описание объектов сцены

Сцена состоит из следующих объектов:

- **Камера** – объект, с которого осуществляется наблюдение за сценой.
- **Источник света** – объект, который освещает сцену.
  - **источник света с фиксированным направлением** – источник света, направление освещения которого не меняется. Задается вектором направления освещения и интенсивностью.
  - **источник рассеянного света** – источник света, освещение которого не меняется от положения наблюдателя. Задается интенсивностью (Используется для освещения всей сцены);
  - **источник точечного света** – источник света, освещение которого меняется от положения наблюдателя. Задается положением и интенсивностью;
- **Модель** – сущность, которая может быть отображена на сцене.
  - **Замечание: Следует отметить, что в данной работе одной из составляющей модели будет являться ось вращения, хотя и отображаться оная не будет**
- **Композит** – объект, который может содержать в себе другие объекты.

1.2 Анализ методов создания моделей

Тело вращения — это поверхность в евклидовом пространстве, образованная вращением кривой (образующей) вокруг оси вращения.[3]

На данном этапе стоит рассмотреть существующие схемы представления тел вращения.

1.2.1 Представление тел вращения

Тела вращения характеризуются осью, радиусами оснований и конструктивными точками образующей поверхности тел. Чтобы лучше разобраться в принципах конструктивного построения формы цилиндра и конуса, следует обратить внимание на рис. 1.1 и на рис. 1.2, где они изображены в виде прозрачных проволочных моделей. На рисунках ясно выражены конструктивная основа и объемно-пространственная характеристика формы предметов. Задача состоит в том, чтобы реализовать грамотное и правильное изображение тел на сцене.

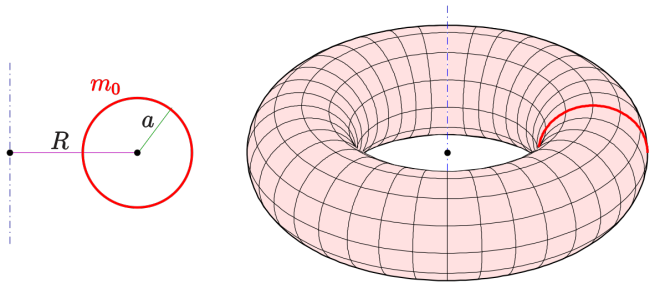

Рисунок 1.1 – Проволочная модель тора

Тело вращения образуется вращательной разверткой кривой профиля C вокруг оси. Если поверхность задается с помощью:

$$P(u,v) = (X(v)cos(u), X(v)sin(u), Z(v)) \tag{1.1}$$

где X и Z - функции, тогда вектор нормали может быть задан через:

$$n(u,v) = X(v)(Z'(v)cos(u), Z'(v)sin(u), -X'(v)) \tag{1.2}$$

где апостроф обозначает первую производную функции.

1.2.2 Понятие поверхности вращения

Поверхность вращения - это один из часто встречающихся типов поверхностей. Сферы и цилиндры можно рассматривать как одни из таких аповерхностей. В общем случае, можно получить тело вращения, вращая тело или набор тел

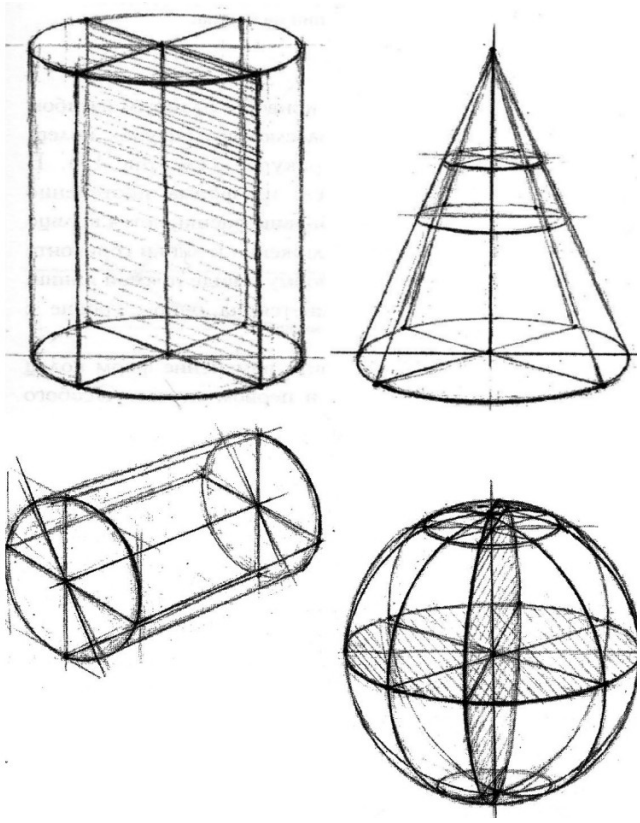

Рисунок 1.2 – Различные проволочные модели

вокруг некоторой произвольной оси. Чтобы более подробно проанализировать, что это значит, рассмотрим самый простой объект для вращения, а именно точку. В этом случае получается окружность в плоскости, ортогональной оси, с центром на оси и радиусом, равным расстоянию точки до оси.

Отсюда следует, что можно представить тело вращения как состоящий из объединения окружностей с центром на оси, по одной для каждой точки вращаемого объекта. Это также предполагает, что способ параметризации точки $\mathbf{P}$ объекта, полученного путем вращения кривой вокруг оси, заключается в использовании двух параметров. Один параметр - это параметр точки на кривой, которая привела к появлению $\mathbf{P}$, а другой - угол, на который она была повернута. Для общих объектов вращения нам понадобилось бы $k + 1$ параметров, где k - количество параметров, необходимых для параметризации вращаемого объекта.

Наше фактическое определение поверхности вращения, которое будет дано в терминах параметризации, ограничится случаем, когда кривая вращается

вокруг оси x. Это упростит определение. Кроме того, из этого можно получить поверхности вращения вокруг произвольной оси, используя жесткие движения.

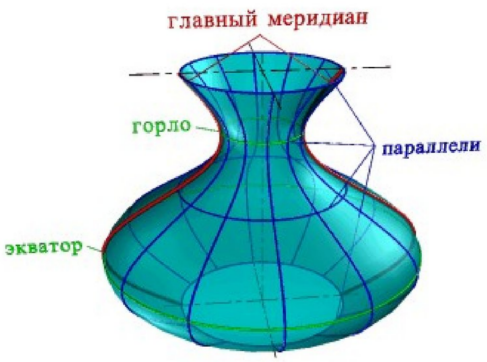

Рисунок 1.3 – Поверхность вращения

1.2.3 Формализация тела вращения

Допустим, что

$$g : [a, b] \longrightarrow R^2 \tag{1.3}$$

будет плоской параметрической кривой и пусть

$$g(t) = (g_1(t), g_2(t)) \tag{1.4}$$

Определим функцию

$$p : [a, b] \times [c, d] \longrightarrow R^3 \tag{1.5}$$

как

$$p(t, \theta) = (g_1(t), g_2(t)cos(\theta), g_2(t)sin(\theta)) \tag{1.6}$$

Подмножество

$$X = p([a, b] \times [c, d]) \subseteq R^3 \tag{1.7}$$

называется поверхностью вращения вокруг оси x для углов c и d относительно g. Кривые

$$\gamma(t) = p(t, \theta) \tag{1.8}$$

для фиксированного θ являются меридианами поверхности вращения и кривые

$$\nu(\theta) = p(t, \theta) \tag{1.9}$$

для фиксированного t называются кругами широты.

Используя стандартную параметризацию $g(t) = (t, f(t))$ для графика f и замена t на x в уравнении 1.6, поверхность, полученная путем вращения графика f вокруг оси x, параметризуется с помощью формулы

$$p(x, \theta) = (x, f(x)cos(\theta), f(x)sin(\theta)) \tag{1.10}$$

Отсюда легко вычислить производные для этой поверхности

$$\frac{\delta p}{\delta x} = (1, f'(x)cos(\theta), f'(x)sin(\theta)) \tag{1.11}$$

$$\frac{\delta p}{\delta \theta} = (0, -f(x)sin(\theta), f(x)cos(\theta)) \tag{1.12}$$

Из этого можно сразу узнать касательные плоскости в каждой точке, потому что перекрестное произведение частных производных является нормальным вектором (при условии, что частные производные не обращаются в нуль).

1.3 Анализ алгоритмов удаления невидимых линий и поверхностей

При выборе алгоритма удаления невидимых линий и поверхностей учитывается особенность поставленной задачи - работа программы будет выполняться в реальном режиме при взаимодействии с пользователем. Этот факт предъявляет к алгоритму требование по скорости работы. Для выбора наиболее подходящего алгоритма следует рассмотреть уже имеющиеся алгоритмы удаления невидимых линий и поверхностей.

1.3.1 Алгоритм обратной трассировки лучей

Алгоритм работает в пространстве изображения[4].

Суть алгоритма: для определения цвета пиксела экрана через него из точки наблюдения проводится луч, ищется пересечение первым пересекаемым объектом сцены и определяется освещенность точки пересечения. Эта освещенность складывается из отраженной и преломленной энергий, полученных от источников света, а также отраженной и преломленной энергий, идущих от других объектов сцены. После определения освещенности найденной точки учитывается ослабление света при прохождении через прозрачный материал и в результате получается цвет точки экрана.

Преимущества:

- изображение, которое строится с учётом явлений дисперсии лучей, преломления, а также внутреннего отражения;
- возможность использования в параллельных вычислительных системах.

Недостатки:

- трудоёмкие вычисления[5];

1.3.2 Алгоритм, использующий Z-буфер

Алгоритм работает в пространстве изображения[6].

Сущность алгоритма: имеется 2 буфера - буфер кадра, который используется для запоминания цвета каждого пиксела изображения, а также z-буфер - отдельный буфер глубины, используемый для запоминания координаты z (глубины) каждого видимого пиксела изображения. В процессе работы глубина или значение z каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксел расположен выше пиксела, находящегося в буфере кадра ($z > 0$), то новый пиксел заносится в цвет рассматриваемого пиксела заносится в буфер кадра, а координата z - в z-буфер. По сути, алгоритм является поиском по x и y наибольшего значения функции $z(x, y)$.

Преимущества:

- возможность обработки произвольных поверхностей, аппроксимируемых полигонами;

- отсутствие требования сортировки объектов по глубине.

Недостатки:

- отсутствие возможности работы с прозрачными и просвечивающими объектами (в классической версии).

1.3.3 Алгоритм Робертса

Алгоритм работает в объектном пространстве[7].

Суть алгоритма: алгоритм прежде всего удаляет из каждого тела те ребра или грани, которые экранируются самим телом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, экранируются этими телами.

Преимущества:

- реализации алгоритма, использующие предварительную приоритетную сортировку вдоль оси z и простые габаритные или минимаксные тесты, демонстрируют почти линейную зависимость от числа объектов[7].

Недостатки:

- вычислительная трудоёмкость алгоритма теоретически растет, как квадрат числа объектов[7];

- отсутствие возможности работы с прозрачными и просвечивающими объектами.

Вывод

В таблице 1.1 представлено сравнение алгоритмов[8] удаления невидимых линий и поверхностей (по каждому параметру составлен рейтинг: 1 - лучший алгоритм, 3 - худший). Так как главным требованием к алгоритму является скорость работы, алгоритмы были оценены по следующим критериям:

- скорость работы (С);

- масштабируемость с ростом количества моделей (ММ);

- масштабируемость с увеличением размера экрана (МЭ);
- работа с фигурами вращения (ФВ).

Алгоритм	С	ММ	МЭ	ФВ
Z-буфера	1	2	1	1
Трассировка лучей	3	1	3	2
Робертса	2	3	1	3

Таблица 1.1 – Сравнение алгоритмов удаления невидимых линий и поверхностей.

С учётом результатов в таблице 1.1 был выбран алгоритм **Z-буфера** удаления невидимых линий и поверхностей.

1.4 Бинарные операции над телами

1.4.1 CTIN

Для описания гомогенных объектов можно использовать граничную модель, определяя границу как множество многоугольников. Частным случаем граничной модели является модель трехмерного тела, у которого граница представляет собой замкнутое множество нерегулярных треугольников. Замкнутую ориентированную поверхность P разобьем на треугольники со следующими условиями:

1. каждая точка поверхности P принадлежит хотя бы одному треугольнику;
2. два треугольника могут пересекаться только в одной вершине или по целому ребру.

Такую модель будем называть CTIN-представлением или CTIN-поверхностью (CTIN — closed triangular irregular network).[1]

На рис. 1.4 изображены верхняя и боковая триангулированные поверхности слоя.

В результате любой булевой операции над двумя CTIN-поверхностями можно получить новую CTIN-поверхность

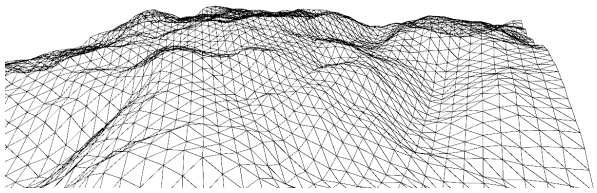

Рисунок 1.4 – Триангулированная поверхность слоя

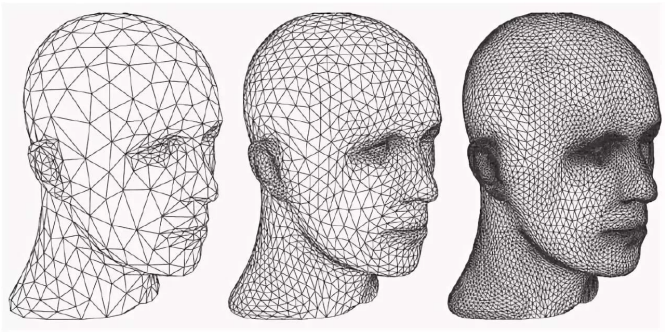

Рисунок 1.5 – Сложная триангулированная поверхность

1.4.2 Улучшенный Z-буффер

Другим подходом является путь улучшенной Z-буфферизации. Конструктивная геометрия твердого тела (CSG) - это подход к геометрическому моделированию. CSG упорядочивает логические операции и примитивные объекты в виде дерева. Узлы (или нетерминалы) дерева представляют логические операции, а листья (или терминалы) представляют объекты. Логическими операциями, используемыми в CSG, являются объединение ($\cup$), пересечение ($\cap$) и разница (-). Аффинные преобразования, такие как масштабирование, перемещение и поворот, также могут быть связаны с каждым узлом дерева. На рис. 1.6 показан абстрактный объект CSG, заданный в терминах коробок, сфер и цилиндров. [2]

В связи с выбранным ранее алгоритмом удаления невидимых линий в виде Z-буффера, был выбран именно этот способ

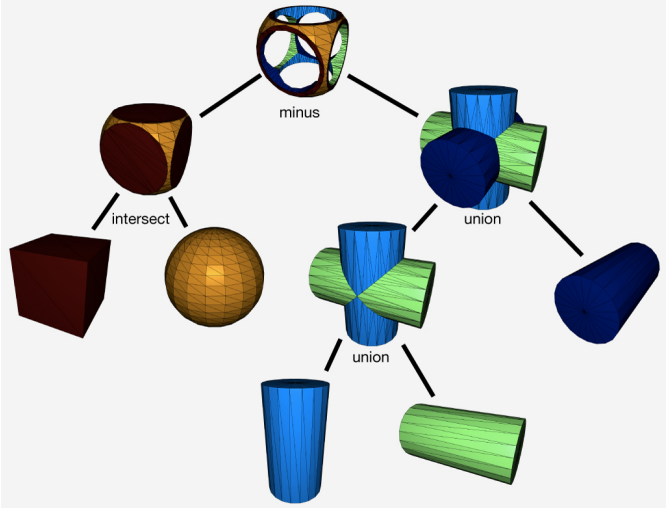

Рисунок 1.6 – CSG-дерево

1.5 Анализ методов закрашивания

Методы закрашивания используются для затенения полигонов (или поверхностей, аппроксимированных полигонами) в условиях некоторой сцены, имеющей источники освещения.

1.5.1 Простая закраска

Суть алгоритма: вся грань закрашивается одним уровнем интенсивности, который зависит высчитывается по закону Ламберта[8]. При данной закраске все плоскости (в том числе и те, что аппроксимируют фигуры вращения), будут закрашены однотонно, что в случае с фигурами вращения будет давать ложные ребра.

Преимущества:

- используется для работы с многогранниками, обладающими преимущественно диффузным отражением.

Недостатки:

- плохо подходит для фигур вращения: видны ребра.

1.5.2 Закраска по Гуро

Суть алгоритма: билинейная интерполяция в каждой точке интенсивности освещения в вершинах[9].

Нормаль к вершине можно найти несколькими способами:

- интерполировать нормали прилегающих к вершине граней;
- использовать геометрические свойства фигуры (так, например, в случае со сферой ненормированный вектор нормали будет в точности соответствовать вектору от центра сферы до рассматриваемой точки).

После нахождения нормали ко всем вершинам находится интенсивность в каждой вершине по закону Ламберта. Затем алгоритм проходится сканирующими строками по рассматриваемому полигону для всех $y : y \in [y_{min}; y_{max}]$. Каждая сканирующая строка пересекает 2 ребра многоугольника, пусть для определённости это будут ребра через одноименные вершины: MN и KL. В точках пересечения высчитывается интенсивность путём интерполяции интенсивности в вершинах. Так, для точки пересечения с ребром MN интенсивность будет рассчитана как (1.13):

$$I_{MN} = \frac{l_1}{l_0} \cdot I_M + \frac{l_2}{l_0} \cdot I_N \tag{1.13}$$

где l_1 - расстояние от точки пересечения до вершины N, l_2 - расстояние от точки пересечения до вершины M, l_0 - длина ребра MN. Для точки пересечения сканирующей строки с ребром KL интенсивность высчитывается аналогично.

Далее, после нахождения точек пересечения, алгоритм двигается по Ox от левой точки пересечения X_{left} до правой точки пересечения X_{right} и в каждой точке $\mathcal{X}$ интенсивность рассчитывается как (1.14):

$$I_{\mathcal{X}} = \frac{\mathcal{X} - X_{left}}{X_{right} - X_{left}} \cdot I_{X_{right}} + \frac{X_{right} - \mathcal{X}}{X_{right} - X_{left}} \cdot I_{X_{left}} \tag{1.14}$$

Преимущества:

- преимущественно используется с фигурами вращения с диффузным отражением, аппроксимированными полигонами.

Недостатки:

- при закраске многогранников ребра могут стать незаметными.

1.5.3 Закраска по Фонгу

Суть алгоритма: данный алгоритм работает похожим на алгоритм Гуро образом, однако ключевым отличием является то, что интерполируются не интенсивности в вершинах, а нормали[9]. Таким образом, закон Ламберта в данном алгоритме применяется в каждой точке, а не только в вершинах, что делает этот алгоритм гораздо более трудоёмким, однако с его помощью можно гораздо лучше изображаются блики.

Преимущества:

- преимущественно используется с фигурами вращения с зеркальным отражением, аппроксимированными полигонами.

Недостатки:

- самый трудоёмкий алгоритм из рассмотренных[8].

Вывод

В таблице 1.2 представлено сравнение алгоритмов[8] закраски (по каждому параметру составлен рейтинг: 1 - лучший алгоритм, 3 - худший). Так как требованиями к алгоритму являются высокая скорость работы, а также возможность закраски фигур вращения с диффузными свойствами отражения, алгоритмы были оценены по следующим критериям:

- скорость работы (С);
- работа с фигурами вращения (ФВ);
- работа с фигурами со свойствами диффузного отражения (ДО).

С учётом результатов в таблице 1.2 был выбран алгоритм закраски **Гуро**.

Алгоритм	С	ФВ	ДО
Простой	1	3	1
Гуро	2	1	1
Фонга	3	1	3

Таблица 1.2 – Сравнение алгоритмов закраски.

Вывод

В данном разделе были формально описаны тела и поверхности вращения, их структурные характеристики, по которым эти модели строятся, были рассмотрены алгоритмы удаления невидимых линий и поверхностей, методы закрашивания поверхностей. В качестве алгоритма удаления невидимых линий и поверхностей был выбран алгоритм Z-буфера, в качестве метода закрашивания был выбран алгоритм закраски Гуро.

2 Конструкторский раздел

2.1 Требования к программному обеспечению

Программа должна предоставлять доступ к функционалу:

- Добавление, удаление объектов/композитов;
- Вращение, масштабирование, перемещение объектов/композитов;
- Редактирование параметров (ширина, высота) объектов;
- Изменение положения и типа источника света;
- Редактирование объектов (отсечение, закраска нужным цветом);
- Редактирование композитов (добавление, удаление объектов);
- Осуществление булевых функций над телами вращения;
- Передвижение по сцене (перемещение и вращение камеры);

К программе предъявляются следующие требования:

- время отклика программы должно быть менее 1 секунды для корректной работы в интерактивном режиме;
- программа должна корректно реагировать на любые действия пользователя.

2.2 Разработка алгоритмов

2.2.1 Алгоритм Z-буфера

1. Всем элементам буфера кадра присвоить фоновое значение
2. Инициализировать Z буфер минимальными значениями глубины
3. Выполнить растровую развертку каждого многоугольника сцены:

(a) Для каждого пикселя, связанного с многоугольником вычислить его глубину z(x, y)

(b) Сравнить глубину пискселя со значением, хранимым в Z буфере.

Если $z(x, y) > z\_buf(x, y)$, тогда
$z\_buf(x, y) = z(x, y), color(x, y) = colorOfPixel.$

4. Отобразить результат.

На рис. 2.1 изображена схема алгоритма Z-буфера

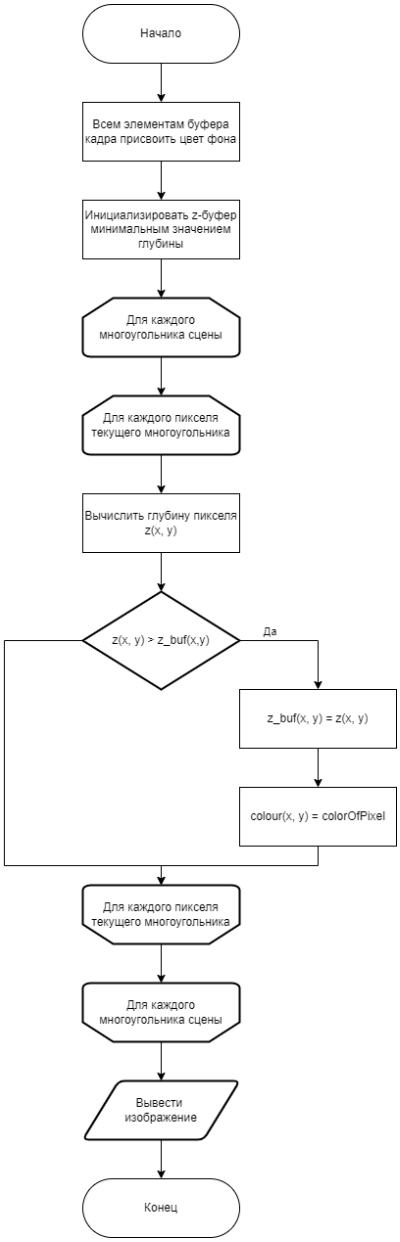

Рисунок 2.1 – Алгоритм Z-буфера

2.2.2 Модифицированный алгоритм, использующий z-буфер

1. Для каждого направленного источника света:
   - Инициализировать теневой z-буфер минимальным значением глубины;
   - Определить теневой z-буфер для источника.
2. Выполнить алгоритм z-буфера для точки наблюдения. При этом, если некоторая поверхность оказалась видимой относительно текущей точки наблюдения, то проверить, видима ли данная точка со стороны источников света
3. Для каждого источника света:
   (a) координаты рассматриваемой точки (x, y, z) линейно преобразовать из вида наблюдателя в координаты $(x0, y0, z0)$ на виде из рассматриваемого источника света;
   (b) сравнить значение $z\_shadowBuf(x0, y0)$ со значением $z0(x0, y0)$. Если $z0(x0, y0) < zshadowBuf(x0, y0)$, то пиксел высвечивается с учетом его затемнения, иначе точка высвечивается без затемнения
4. Отобразить результат.

2.3 Выбор используемых типов и структур данных

Для разрабатываемого ПО нужно будет реализовать следующие типы и структуры данных.

1. **Источник света** – направленностью света.
2. **Сцена** – задается объектами сцены.
3. **Объекты сцены** – задаются вершинами и гранями.
4. **Математические абстракции**.

(a) **Точка** – хранит координаты x, y, z.

(b) **Вектор** – хранит направление по x, y, z.

(c) **Фигура** – хранит вершины, нормаль, цвет.

5. **Интерфейс** – используются библиотечные классы для предоставления доступа к интерфейсу.

6. **Графический обработчик** - абстрактная структура, выполянющая реализацию алгоритмов.

7. **Фабрики** для сцены, интерфейса, обработчика - для возможной подмены в ходе разработки или дополнения в дальнейшем.

8. **Композит** - объект, который будет содержать в себе другие объекты

3 Технологический раздел

3.1 Средства реализации

Основным языком программирования является мультипарадигменный язык Rust[10].

– Одно из главных достоинств данного языка это гарантия безопасной работы с памятью при помощи системы статической проверки ссылок, так называемый Borrow Checker[11].

– Отсутствие сброщика мусора, как следствие, более экономная работа с ресурсами

– Встроенный компилятор

– Кросс-платформенность, от UNIX и MacOS до Web

– Крайне побдробные коды ошибки и документация от разработчиков языка

– Важно отметить, что язык программирования Rust сопоставим по скорости с такими языками как C и C++, предоставляя в то же время более широкий функционал для тестирования кода и контроля памяти.

Также были выбраны следующие библиотеки:

– В качестве графического интерфейса была выбрана библиотека Slint[12] (или иначе crate в контексте языка Rust)

– Для рендера изображения была выбрана библиотека tiny-skia[13], предоставляющий быстрый CPU-рендеринг

– Помимо этого Slint дает инструментарий для запуска приложения в браузере при непосредственном участии WebAssembly при практически нулевых затратах со стороны программиста.

– Для тестирования ПО использовались инструменты Cargo[14] - пакетного менеджера языка Rust, поставляемого вместе с компилятором из официального источника.

Среда разработки:

- Работа была проведена в среде разработки CLion[15] от компании JetBrains[16]
- Дополнительный плагин "Rust"для поддержки синтаксиса языка.

3.2 Структура классов

На рисунках 3.1 - 3.2 представлена структура реализуемых классов.

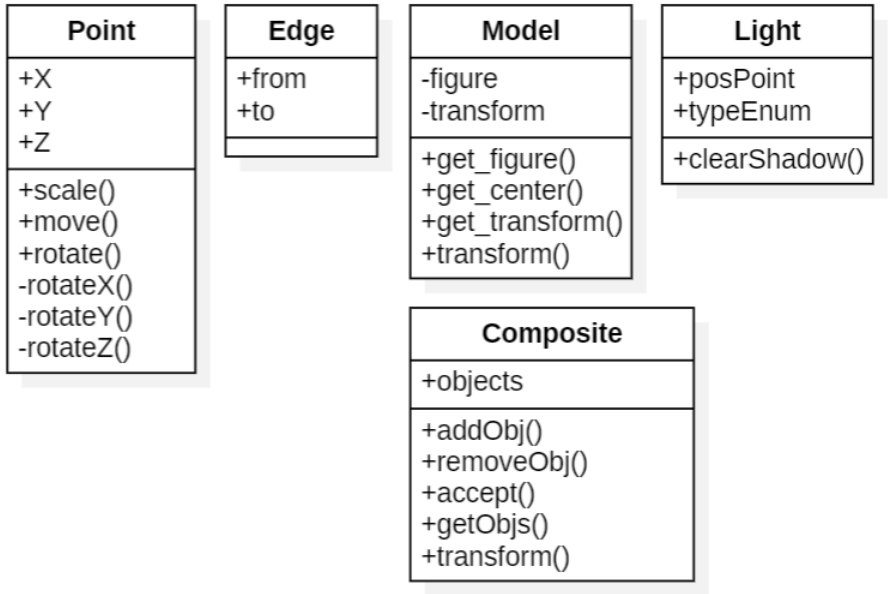

Рисунок 3.1 – Структура классов-объектов

- Point – класс точки трехмерного пространства. Хранит координаты в пространстве, владеет методами преобразований точки.
- Edge – класс грани. Хранит номера задействованных в грани вершин.
- Light – класс источника света.
- Model - класс модели. Скрывает конкретную реализацию модели(фигуры) и предоставляет единый интерфейс для работы с ней. Владеет методами преобразования модели, а также методами для получения информации о модели.

- Composite - класс композита. Хранит в себе набор моделей, владеет методами для работы с ними.

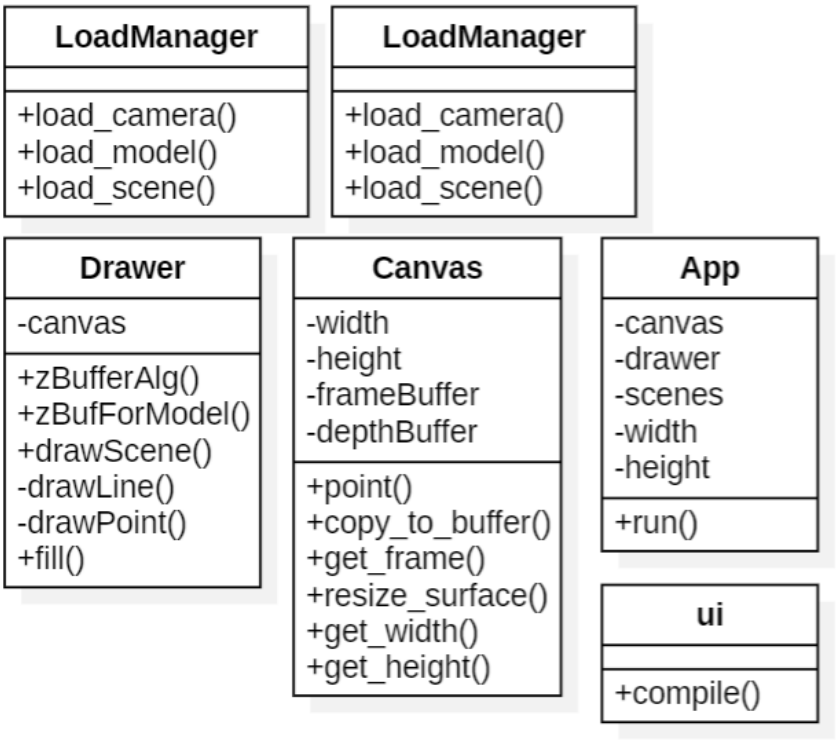

Рисунок 3.2 – Структура классов

- Drawer – класс, отвечающий за растеризацию сцены. Хранит полотно для отрисовки. Владеет методами алгоритма теневого z-буфера и формирования объекта для отображения рисунка в главном приложении.
- App – точка входа в программу.
- Ui - класс, отвечающий за отображение графического интерфейса.
- TransformManager – абстракция, содержащия методы трансформмации объектов.
- TransformManager - абстракция, содержащия методы загрузки объектов.
- Canvas - класс, отвечающий за отображение сцены.

3.3 Реализация алгоритмов

В листинге 3.1 представлена реализация Z-буффера на языке Rust.

Листинг 3.1 – Реализация алгоритма Z-буффера

```
#[derive(Copy, Clone)]
pub struct Vector3D<T> {
    pub x: T,
    pub y: T,
    pub z: T,
}
impl<T> Vector3D<T> {
    pub fn new(x: T, y: T, z: T) -> Vector3D<T> {
        Vector3D {
            x: x,
            y: y,
            z: z,
        }
    }
}
impl<T: NumCast> Vector3D<T> {
    pub fn to<V: NumCast>(self) -> Vector3D<V> {
        Vector3D {
            x: NumCast::from(self.x).unwrap(),
            y: NumCast::from(self.y).unwrap(),
            z: NumCast::from(self.z).unwrap(),
        }
    }
}
impl Vector3D<f32> {
    pub fn norm(self) -> f32 {
        return (self.x*self.x+self.y*self.y+self.z*self.z).sqrt();
    }
    pub fn normalized(self, l: f32) -> Vector3D<f32> {
        return self*(l/self.norm());
    }
}
impl<T: fmt::Display> fmt::Display for Vector3D<T> {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        write!(f, "({},{},{})", self.x, self.y, self.z)
    }
}
impl<T: Add<Output = T>> Add for Vector3D<T> {
    type Output = Vector3D<T>;
    fn add(self, other: Vector3D<T>) -> Vector3D<T> {
```

```
41          Vector3D { x: self.x + other.x, y: self.y + other.y, z: self.z + other.z}
42      }
43  }
44  impl<T: Sub<Output = T>> Sub for Vector3D<T> {
45      type Output = Vector3D<T>;
46      fn sub(self, other: Vector3D<T>) -> Vector3D<T> {
47          Vector3D { x: self.x - other.x, y: self.y - other.y, z: self.z - other.z}
48      }
49  }
50  impl<T: Mul<Output = T> + Add<Output = T>> Mul for Vector3D<T> {
51      type Output = T;
52      fn mul(self, other: Vector3D<T>) -> T {
53          return self.x*other.x + self.y*other.y + self.z*other.z;
54      }
55  }
56  impl<T: Mul<Output = T> + Copy> Mul<T> for Vector3D<T> {
57      type Output = Vector3D<T>;
58      fn mul(self, other: T) -> Vector3D<T> {
59          Vector3D { x: self.x * other, y: self.y * other, z: self.z * other}
60      }
61  }
62  impl<T: Mul<Output = T> + Sub<Output = T> + Copy> BitXor for Vector3D<T> {
63      type Output = Vector3D<T>;
64      fn bitxor(self, v: Vector3D<T>) -> Vector3D<T> {
65          Vector3D { x: self.y*v.z-self.z*v.y, y: self.z*v.x-self.x*v.z, z: self.x*v.y-self.y*v.x}
66      }
67  }
```

Вывод

В данном разделе были рассмотрены средства, с помощью которых было реализовано ПО, а также представлены структуры классов и листинги кода с реализацией алгоритмов компьютерной графики.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Тюкачев Н. А.* Бинарные операции над трехмерными телами // Вестник ВГУ, серия: Системный анализ и информационные технологии. — 2010.
2. *Sabu John N. S. G. L.* An Improved Z-Buffer CSG Rendering Algorithm / RMIT University, Melbourne, Australia.
3. Middlemiss; Marks; Smart. '15-4. Surfaces of Revolution'. Analytic Geometry (3rd ed.). p. 378. — 1968. — https://lccn.loc.gov/68015472.
4. Трассировка лучей в реальном времени. — Режим доступа: https://www.ixbt.com/3dv/directx-raytracing.html (дата обращения: 13.08.2022).
5. Cost Analysis of a Ray Tracing Algorithm. — Режим доступа: https://www.graphics.cornell.edu/~bjw/mca.pdf (дата обращения: 05.09.2022).
6. Алгоритм Z-буфера. — Режим доступа: http://compgraph.tpu.ru/zbuffer.htm (дата обращения: 21.08.2022).
7. Алгоритм Робертса. — Режим доступа: http://compgraph.tpu.ru/roberts.htm (дата обращения: 14.05.2022).
8. *Д. Р.* Алгоритмические основы машинной графики. — 1989.
9. Модели затенения. Плоская модель. Затенение по Гуро и Фонгу. — Режим доступа: https://compgraphics.info/3D/lighting/shading_model.php (дата обращения: 18.06.2022).
10. Rust Programming Language [Электронный ресурс]. — 2021. — Режим доступа: https://doc.rust-lang.org/std/index.html (дата обращения: 20.07.2022).
11. Rust Borrow Checker [Электронный ресурс]. — 2021. — https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html.
12. Slint - The UI Toolkit [Электронный ресурс]. — 2021. — https://slint-ui.com/releases/0.2.5/docs/rust/slint/.
13. A tiny Skia subset ported to Rust [Электронный ресурс]. — 2021. — https://github.com/RazrFalcon/tiny-skia.

14. The Cargo Book. — 2021. — Режим доступа: https://doc.rust-lang.org/cargo/ (дата обращения: 21.07.2022).

15. Кросс-платформенная среда разработки CLion [Электронный ресурс]. — 2022. — https://www.jetbrains.com/ru-ru/clion/.

16. JetBrains [Электронный ресурс]. — https://www.jetbrains.com/.