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 и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Визуализация тел вращения»

Студент ИУ7-52 _____ А. А. Новиков
(Группа) (Подпись, дата) (И. О. Фамилия)

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

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире компьютерная графика является важным инструментом для создания реалистичных изображений в различных областях — от компьютерных игр и кино до научных исследований и инженерных задач. Одной из ключевых задач графики является моделирование и визуализация трехмерных объектов с высокой степенью детализации. Это особенно актуально при создании сложных объектов, требующих учета текстур, неровностей поверхности и взаимодействия с внешними условиями.

Одним из эффективных подходов для создания реалистичных моделей является визуализация тел вращения — объектов, форма которых определяется вращением плоской кривой (образующей) вокруг оси (направляющей). Для более детальной и правдоподобной визуализации на полученное тело можно нанести фактуру и текстуру материала. Это требует применения специализированных алгоритмов, таких как рельефное текстурирование, что позволяет сделать модель более естественной и реалистичной.

Цель работы — разработка программного обеспечения для визуализации тел вращения с добавлением фактуры и текстуры материала. Для достижения поставленной цели необходимо решить следующие задачи:

— описать предметную область работы;

— рассмотреть и выбрать алгоритмы построения реалистичного тела вращения;

— на основе выбранных алгоритмов спроектировать программное обеспечение;

— реализовать спроектированное программное обеспечение;

— провести исследование на основе разработанной программы.

1 Аналитический раздел

1.1 Описание предметной области

Тело вращения — это объект, образованный вращением плоской линии (образующей) вокруг фиксированной оси (направляющей). Таким образом, любая точка образующей описывает окружность, центр которой лежит на оси вращения, а само тело получается симметричным относительно этой оси [1].

1.2 Методы задания кривой

Образующая задается как плоская кривая, поэтому необходимо рассмотреть методы задания кривых линий:

1.2.1 Полиномиальные кривые

Полиномиальные кривые задаются аналитическими уравнениями вида:

$$y = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0, \tag{1.1}$$

где n — степень полинома, а a_i — его коэффициенты. Метод задания кривой заключается в определении коэффициентов полинома a_i. Для этого для заданного количества точек k необходимо решить систему из k уравнений:

$$\begin{cases} y_1 = a_{k-1} x_1^{k-1} + a_{k-2} x_1^{k-2} + ... + a_0 \\ y_2 = a_{k-1} x_2^{k-1} + a_{k-2} x_2^{k-2} + ... + a_0 \\ \vdots \\ y_k = a_{k-1} x_k^{k-1} + a_{k-2} x_k^{k-2} + ... + a_0 \end{cases} . \tag{1.2}$$

После нахождения коэффициентов уравнения можно вычислить значение кривой для любого x в заданном диапазоне. Однако не всегда получается найти точное решение системы, также при высоких степенях полинома может возникнуть эффект Рунге, из-за которого кривая будет задана отличным от ожидаемого образом, с выраженными осцилляциями [2].

1.2.2 Кубические сплайны

Кубические сплайны — это составные полиномиальные кривые третьей степени, обеспечивающие гладкость соединения сегментов. Каждый сегмент сплайна на интервале $[x_i, x_{i+1}]$ описывается уравнением:

$$S_i(x) = a_i x^3 + b_i x_2 + c_i x + d_i, \tag{1.3}$$

где a_i, b_i, c_i, d_i — коэффициенты, определяемые для каждого сегмента. Для задания кривой вычисляются коэффициенты для каждого из сегментов кривой путем решения системы уравнений, которая учитывает непрерывность на стыках сегментов функции, первой производной и второй производной:

$$\begin{cases} S_i(x_{i+1}) = S_{i+1}(x_{i+1}) \\ S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \\ S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \end{cases} . \tag{1.4}$$

После вычисления коэффициентов кривая задается как последовательность сегментов.

1.2.3 Кривая Безье

Кривая Безье определяется контрольными точками $P_0, P_1, ..., P_n$. Эти точки задают форму кривой, но сама кривая не обязательно будет проходить через все точки, кроме крайних.

Для построения кривой используется алгоритм де Кастельжо [3]. Для каждого значения параметра $t \in [0, 1]$, итеративно вычисляются новые точки $P_{i,j}$ по формуле:

$$P_{i,j} = (1 - t)P_{i-1,j} + tP_{i,j+1}, \tag{1.5}$$

где $P_{i-1,j}, P_{i,j-1}$ – соседние точки на предыдущем уровне итерации, t — параметр интерполяции, определяющий положение точки между ними. Процесс начинается с заданного набора контрольных точек (нулевой уровень). На каждом последующем уровне число точек уменьшается, так как каждая новая точка вычисляется как интерполяция двух точек предыдущего уровня. Алгоритм продолжается до тех пор, пока не останется единственная точка,

которая и будет являться точкой кривой для данного значения параметра t.

Использование кривых Безье позволяет задавать кривые различных форм, которые можно легко построить на плоскости по заданным точкам с помощью алгоритма де Кастельжо.

1.2.4 Выбор метода задания кривой

В качестве метода задания кривых были выбраны кривые Безье, так как с их помощью можно задавать кривые различных форм, а с помощью алгоритма де Кастельжо их легко строить на плоскости без необходимости производить сложные вычисления.

1.3 Алгоритмы построения тела вращения

Рассмотрим несколько существующих алгоритмов построения тела вращения:

— **Построение при дискретном представлении тела.** При дискретном представлении тело вращения задается множеством примитивов (полигонов), которые являются математическим приближением тела. Образующая разбивается на множество точек, каждая точка вращается вокруг выбранной направляющей. Полученные точки являются вершинами полигонов, которые задают тело;

— **Построение при аналитическом представлении**. При аналитическом представлении тело вращения описывается с использованием параметрических уравнений, которые выражают положение каждой точки поверхности через параметры. Образующая задается аналитической функцией $f(x)$, описывающей ее форму. Для построения тела вращения вычисляются координаты точек поверхности путем вращения образующей вокруг оси с использованием параметрических формул;

— **Построение при произвольной направляющей.** Если направляющая задается как произвольная кривая, то необходимо рассмотреть движение образующей вдоль этой прямой в пространстве. Образующая сдвигается и вращается вдоль направляющей, формируя тело вращения.

1.3.1 Выбор алгоритма построения тела вращения

Для визуализации тел вращения в данной работе был выбран алгоритм построения при дискретном представлении тела. Поскольку направляющая задается одной из координатных осей, отсутствует необходимость учитывать сложные преобразования, связанные с произвольной направляющей. Кроме того, данный подход предпочтителен по сравнению с алгоритмом аналитического построения, так как он обеспечивает большую гибкость в представлении сложных форм. Алгоритм дискретного представления позволяет работать с любыми кривыми, заданными конечным набором точек, что упрощает реализацию и делает его более универсальным.

1.4 Алгоритмы удаления невидимых линий

Для построения реалистичного трехмерного изображения необходимо определить, какие линии или поверхности объектов видимы для наблюдателя, а какие нет.

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

1.4.1 Алгоритм Варнока

Основная идея алгоритма Варнока заключается в решении вопроса: что нужно отображать в каждом конкретном окне изображения. Изначально окно охватывает весь экран. Если невозможно сразу определить, что отображать, окно делится на четыре части. В результате получается 4 окна меньших размеров. Если и для них невозможно дать точный ответ, процесс деления продолжается до тех пор, пока каждое окно не станет достаточно малым, чтобы дать однозначный ответ, или пока размер окна не достигнет одного пикселя [4].

На рисунке 1.1 показан пример разбиения по алгоритму Варнока.

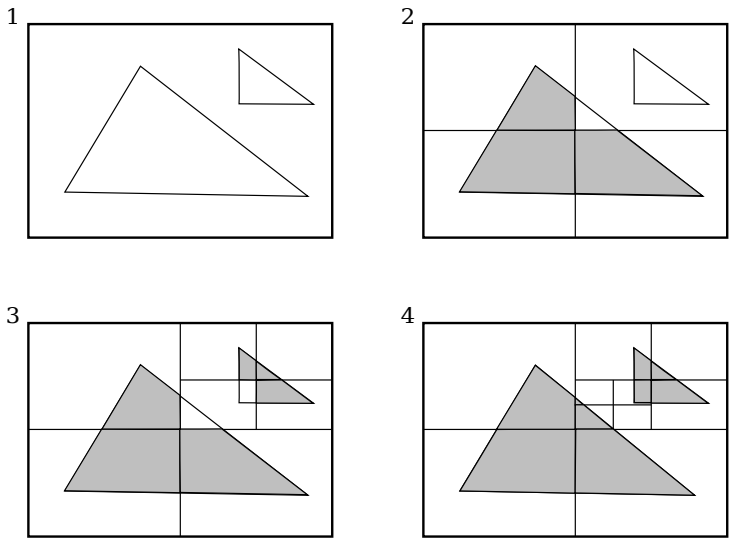

Рисунок 1.1 – Пример разбиения алгоритмом Варнока

Из недостатков алгоритма можно отметить необходимость производить большое количество разбиений при визуализации сложной сцены.

1.4.2 Алгоритм Z-буфера

Алгоритм Z-буфера решает задачу скрытых поверхностей, используя два буфера: буфер кадра, который хранит информацию о цвете и яркости пикселей, и Z-буфер, фиксирующий координаты глубины. Процесс начинается с инициализации Z-буфера максимальными значениями глубины, а буфера кадра — значениями фона. Для каждого многоугольника сцены его пиксели сравниваются с пикселями в Z-буфере, и если значение глубины для текущего пикселя меньше соответствующего значения в Z-буфере, то происходит обновление как в буфере кадра, так и в Z-буфере.

Этот метод прост и эффективен для работы с произвольными сценами, но требует значительных объемов памяти для хранения глубины каждого пикселя. Также он имеет ограничения при работе с прозрачными объектами и сглаживании лестничного эффекта на границах многоугольников.

1.4.3 Алгоритм прямой трассировки лучей

Алгоритм прямой трассировки лучей заключается в том, что лучи света испускаются от источника и распространяются по сцене, взаимодействуя с объектами. Когда свет достигает поверхности объекта, он может отражаться, поглощаться или преломляться в зависимости от свойств материала. Для каждого такого взаимодействия необходимо просчитать, куда направляется

луч, и что происходит с его интенсивностью и цветом. Задача алгоритма — вычислить траекторию света от источника до наблюдателя, моделируя эффекты отражения, преломления и рассеивания, что позволяет учесть такие явления, как тени, отражения и прохождение света через полупрозрачные объекты.

Основная сложность прямой трассировки заключается в том, что лишь небольшая часть лучей, испущенных от источников света, достигает поверхности камеры или экрана, что делает метод вычислительно затратным. Этот алгоритм требует большого количества лучей для получения реалистичного изображения, что значительно увеличивает время рендеринга сложных сцен [5].

1.4.4 Алгоритм обратной трассировки лучей

Алгоритм обратной трассировки лучей работает по принципу, противоположному прямой трассировке. Вместо того, чтобы испускать лучи от источников света, алгоритм начинает с наблюдателя. Каждый луч проходит через пиксели изображения и пересекают объекты в сцене. Когда луч достигает поверхности объекта, он может отразиться, переломиться или поглотиться. Отраженные и преломленные лучи продолжают свое движение по сцене, пока не встретят другие объекты или не будут отброшены. Для расчета эффектов освещения проводятся вторичные лучи от точек пересечения до всех источников света. Если вторичный луч пересекается с непрозрачным телом, то точка, из который вторичный луч был выпущен, находится в тени.

Этот метод более эффективен, поскольку вычисления ведутся только для тех лучей, которые попадают в камеру, что позволяет существенно сократить число обрабатываемых лучей по сравнению с прямой трассировкой. Кроме того, обратная трассировка лучей может точно учитывать такие эффекты, как отражения, преломления и тени, делая изображение более реалистичным. Однако, несмотря на свою эффективность по сравнению с прямой трассировкой, обратная трассировка все же требует значительных вычислительных ресурсов для сложных сцен с большим количеством объектов.

Для оптимизации количества вычислений можно поместить объект в выпуклую оболочку: сферическую или в форме параллелепипеда. Таким образом, если луч не пересекает оболочку тела, то он не пересекает и само

тело, значит не нужно искать пересечение луча и тела.

Визуализация работы алгоритма обратной трассировки лучей приведена на рисунке 1.2.

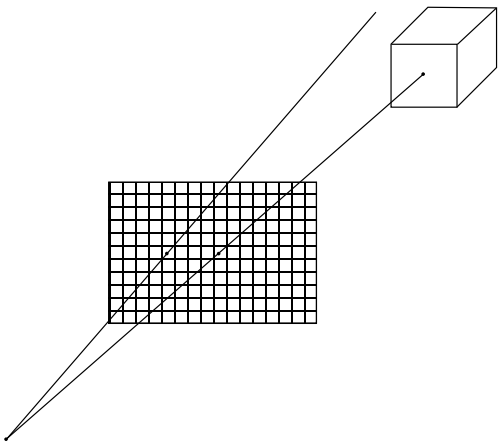

Рисунок 1.2 – Пример трассировки лучей

1.4.5 Выбор алгоритма удаления невидимых линий

В данной работе для удаления невидимых линий был выбран алгоритм Z-буфера благодаря его простоте и скорости работы, что важно для взаимодействия с полученным телом в реальном времени с минимальными задержками.

1.5 Алгоритмы закрашивания

Существует несколько алгоритмов закрашивания:

— **Однотонная закраска.** Этот метод довольно прост в своей реализации. Для каждой грани объекта вычисляется уровень освещенности, который применяется одинаково ко всей поверхности грани. Это обеспечивает быструю обработку, однако изображение выглядит плоским, так как не учитывается плавный переход интенсивности света между различными частями объекта;

— **Закраска по Гуро.** В данном методе интенсивность света вычисляется для каждой вершины грани, а затем интерполируется. Этот метод позволяет получить гораздо более визуально-приятное изображение. Это позволяет сгладить переходы между различными частями поверхности, что улучшает визуальное восприятие объекта. Однако при низком

уровне детализации блики могут быть смазанными, так как интенсивность света не пересчитывается для каждой точки поверхности, а лишь интерполируется;

— **Закраска по Фонгу.** В этом методе используется интерполяция вектора нормали к поверхности вместо интерполяции интенсивности. Интерполяция выполняется между начальной и конечной нормалями, которые сами тоже являются результатом интерполяции вдоль ребер многоугольника между нормалями в вершинах. Данный метод требует гораздо большего количества вычисление, чем метод закраски по Гуро или метод однотонной закраски, но дает значительно лучшие результаты, а также позволяет применять методы рельефного текстурирования для визуализации неровностей [6].

В методах Гуро и Фонга необходимо рассчитывать значение нормали в вершинах многогранника. Нормаль в точке вычисляется путем путем усреднения нормали по всем полигональным граням, которым принадлежит вершина:

$$\vec{N} = \frac{\sum_{i=1}^{n} \vec{N}_i}{n}. \tag{1.6}$$

1.5.1 Выбор алгоритма закрашивания

В качестве алгоритма закрашивания был выбран алгоритм закраски по Фонгу, так как данный алгоритм позволяет максимально точно передать освещение благодаря детальной интерполяции нормалей на поверхности. Кроме того, этот метод идеально подходит для реализации рельефного текстурирования.

1.6 Анализ моделей освещения

Существует две модели освещения: локальная и глобальная. Локальная модель не рассматривает процессы светового взаимодействия объектов сцены между собой, рассчитывает освещенность только самих объектов. Глобальная модель рассматривает трехмерную сцену, как единую систему и описывает освещение с учетом взаимного влияния объектов. Для поставленной задачи лучше подходит локальная модель освещения, так как она является более

быстродействующей. Также в сцене отсутствуют объекты, обладающие зеркальными или преломляющими свойствами, поэтому использование более качественной глобальной модели не требуется. Рассмотри наиболее популярные локальные модели освещения.

1.6.1 Модель Ламберта

Модель Ламберта моделирует идеальное диффузное освещение [7]. Считается, что свет при попадании на поверхность рассеивается равномерно во все стороны. При расчете такого освещения учитывается только ориентация поверхности (нормаль) и направление на источник света. Рассеянную составляющую можно рассчитать по формуле 1.7:

$$I_d = k_d \cdot \cos(\vec{L}, \vec{N}) \cdot i_d, \tag{1.7}$$

где I_d - рассеянная составляющая освещенности в точке, k_d - свойство материала воспринимать рассеянное освещение, i_d - мощность рассеянного освещения, L - направление из точки на источник, N - вектор нормали в точке.

1.6.2 Модель Фонга

Модель Фонга —- классическая модель освещения. Модель представляет собой комбинацию диффузной составляющей (модели Ламберта) и зеркальной составляющей и работает таким образом, что кроме равномерного освещения на материале может еще появляться блик [7]. Местонахождение блика на объекте, освещенном по модели Фонга, определяется из закона равенства углов падения и отражения.

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. Т.о. отраженная составляющая освещенности в точке зависит от того, насколько близки направления на наблюдателя и отраженного луча. Это можно выразить следующей формулой 1.8:

$$I_s = k_s \cdot \cos^{\alpha}(\vec{R}, \vec{V}) \cdot i_s, \tag{1.8}$$

где I_s - зеркальная составляющая освещенности в точке, k_s - коэффициент зеркального отражения, i_s - мощность зеркального освещения, R - направление

отраженного луча, V - направление на наблюдателя, α - коэффициент блеска, свойство материала.

1.6.3 Выбор модели освещения

Так как сцена не предполагает наличия на ней зеркальных объектов, в данной курсовой работе используется модель освещения Ламберта. Модель обеспечивает приемлемое качество освещения, легко реализуется и интегрируется с закраской по Фонгу, а также наименее требовательна к вычислительным ресурсам.

1.7 Анализ алгоритмов визуализации неровностей

Существует несколько алгоритмов рельефного текстурирование:

— **Bump mapping.** Этот метод создает иллюзию рельефа на поверхности без изменения ее геометрии. Bump mapping использует карту высот, где каждый пиксель задает отклонение нормали поверхности. На основе этих нормалей пересчитываются эффекты освещения, что создает ощущение шероховатости [8];

— **Normal mapping.** Метод является развитием bump mapping и моделирует рельеф точнее за счет использования карты нормалей. В normal mapping для каждой точки поверхности хранятся три значения, соответствующие координатам вектора нормали, что дает более точные результаты при расчете освещения. Данный метод также не изменяет геометрию объекта, а лишь корректирует его освещенность [9];

— **Parallax mapping.** Этот метод создает иллюзию глубины на поверхности, изменяя текстурные координаты в зависимости от угла зрения наблюдателя. Parallax mapping использует карту высот, где значения представляют относительные изменения высоты. На основе этой карты текстурные координаты смещаются, чтобы имитировать перспективу и визуализировать рельеф. Метод не изменяет геометрию объекта и используется для повышения реализма текстур при низких вычислительных затратах;

— **Displacement mapping.** Этот метод физически изменяет геометрию

объекта, преобразуя его поверхность в соответствии с картой высот. Каждая вершина модели смещается вдоль нормали поверхности на величину, указанную в карте. Это позволяет создавать реалистичный рельеф с правильным взаимодействием света и теней. Метод требует значительных вычислительных ресурсов и применяется в задачах, где важен высокий уровень реализма.

1.7.1 Выбор алгоритма визуализации неровностей

В качестве алгоритма визуализации неровностей был выбран normal mapping, так как данный алгоритм позволяет получить реалистичное отображение мелких деталей поверхности без увеличения количества геометрических примитивов.

ВЫВОД

В данном разделе был проведен анализ существующих алгоритмов построения построения изображения. Для задания кривой была выбрана кривая Безье. Для получения тела вращения был выбран алгоритм построения при дискретном представлении тела. Для построения реалистичного изображения были выбраны алгоритм Z-буфера, модель освещения Ламберта, закраска по Фонгу, normal mapping.

2 Конструкторский раздел

В данном разделе представлены требования к программному обеспечению, рассмотрены структуры данных, алгоритмы и математические уравнения, выбранные для построения сцены.

2.1 Требования к программному обеспечению

Программное обеспечение должно обеспечивать следующую функциональность:

— Возможность ввода пользователем кривой и выбора оси вращения;

— Вывод полученного тела вращения;

— Возможность наложения фактуры или текстуры на тело по запросу пользователя;

— Возможность изменения положения камеры в пространстве для обзора объекта под разными углами;

— Освещение объекта статическим источником света.

2.2 Используемые структуры данных

Для реализации работы программы разработаны следующие структуры данных:

1. Сцена содержит:
   — Тело вращения;
   — Источник света.
2. Тело вращения состоит из:
   — Массива вершин;
   — Массива граней;
   — Цвета тела вращения;
   — Нормалей точек.

3. Вершина объекта содержит:

   — Положение вершины в пространстве.

4. Грань объекта содержит:

   — Индексы трех вершин из списка вершин тела вращения, образующих грань.

5. Источник света содержит:

   — Вектор направления источника;

   — Интенсивность источника.

2.3 Алгоритм построения изображения

Алгоритм генерации изображения представлен в виде диаграммы, оформленной в соответствии с нотацией IDEF0 и отражающей общую декомпозицию алгоритма:

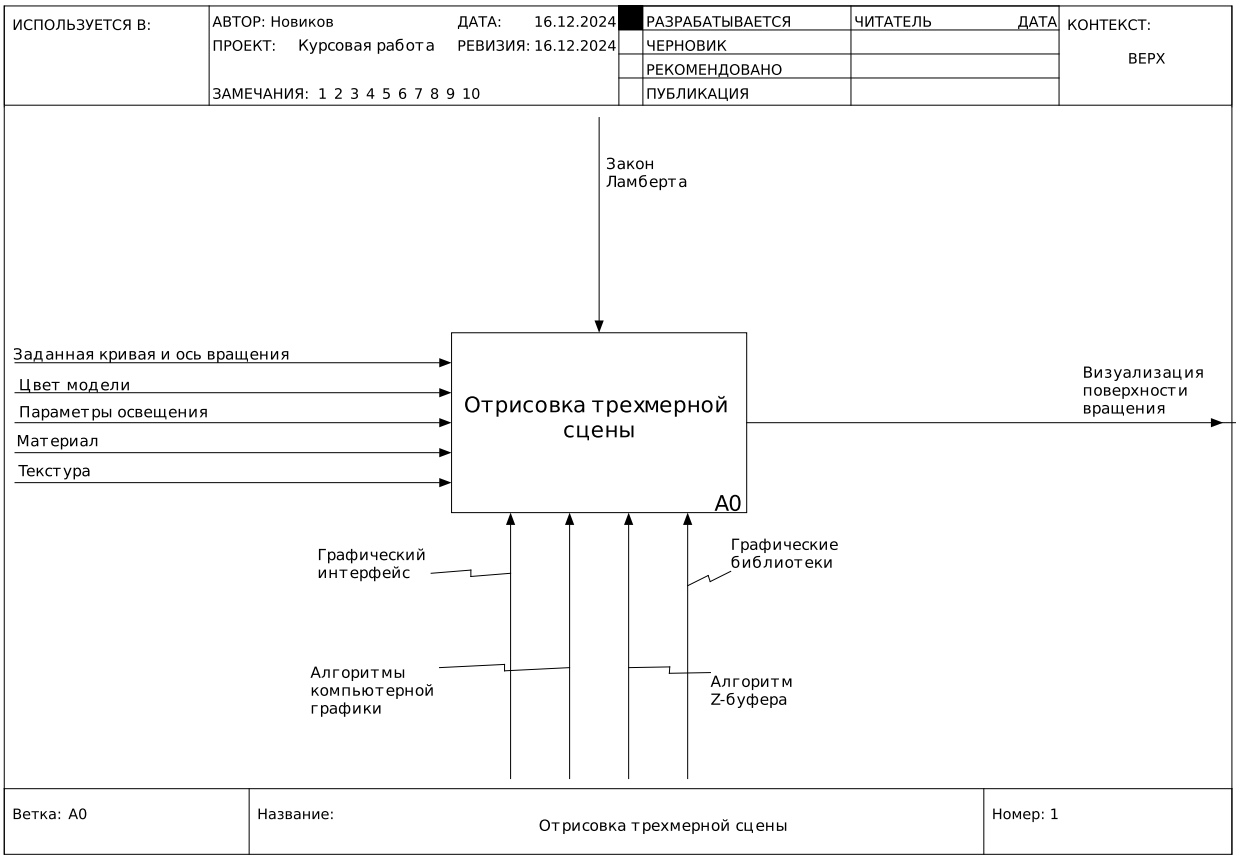

Рисунок 2.1 – Контекстная диаграмма верхнего уровня в нотации IDEF0

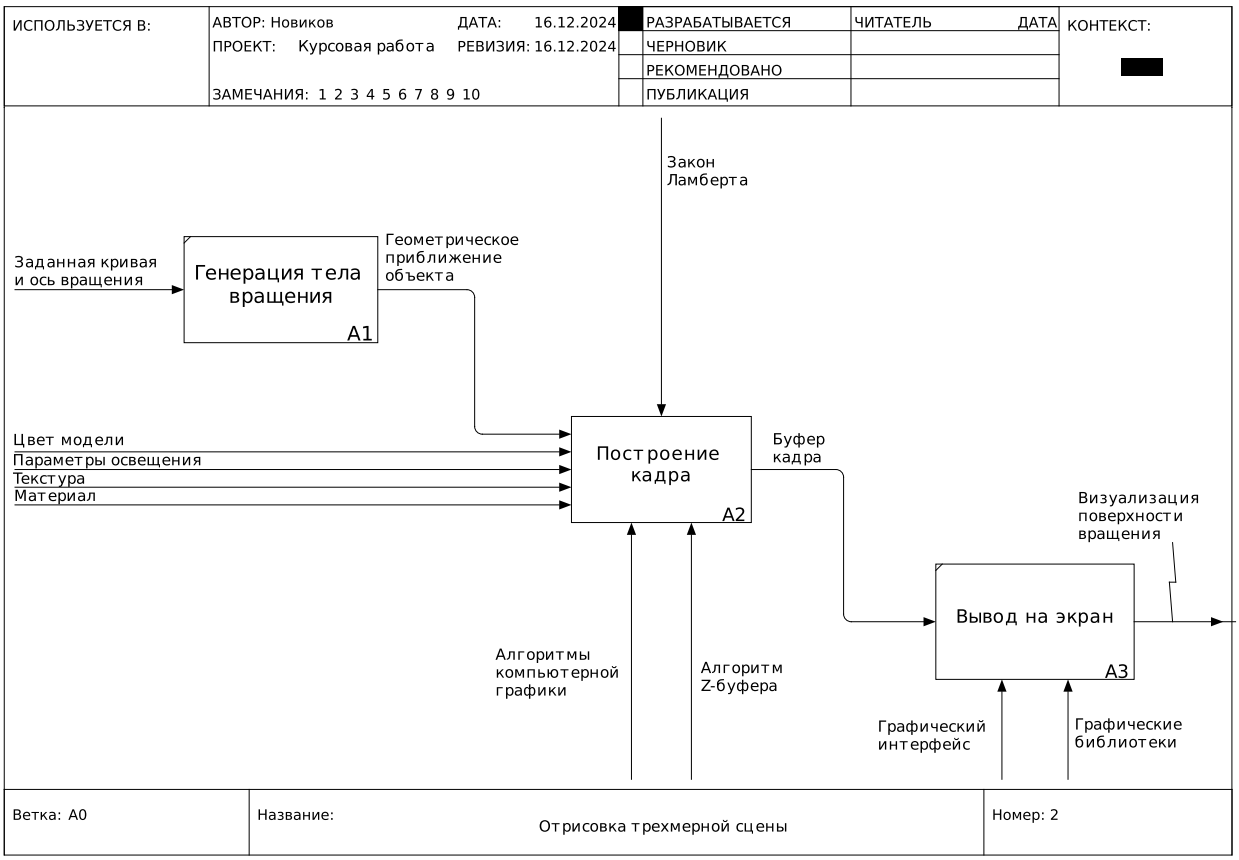

Рисунок 2.2 – Основной цикл ПО в нотации IDEF0

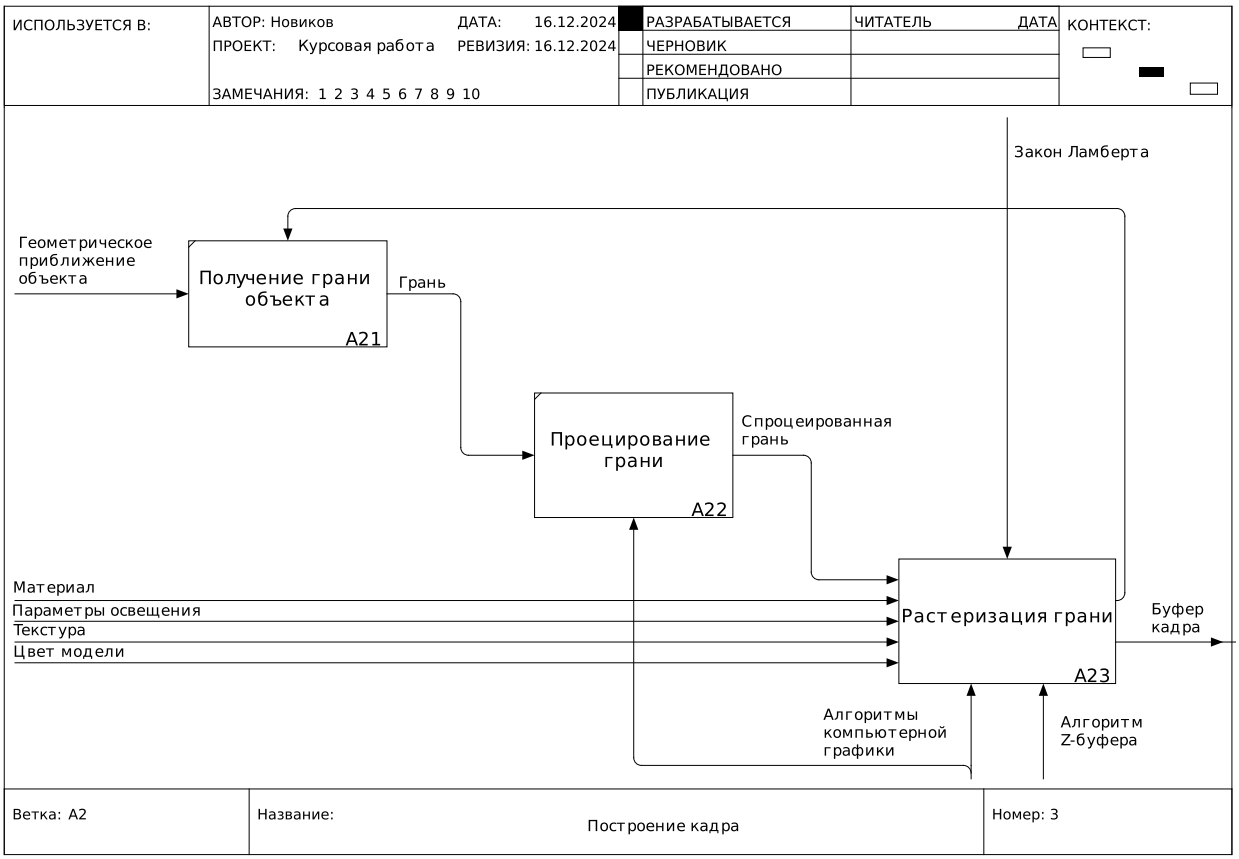

Рисунок 2.3 – Построение кадра в нотации IDEF0

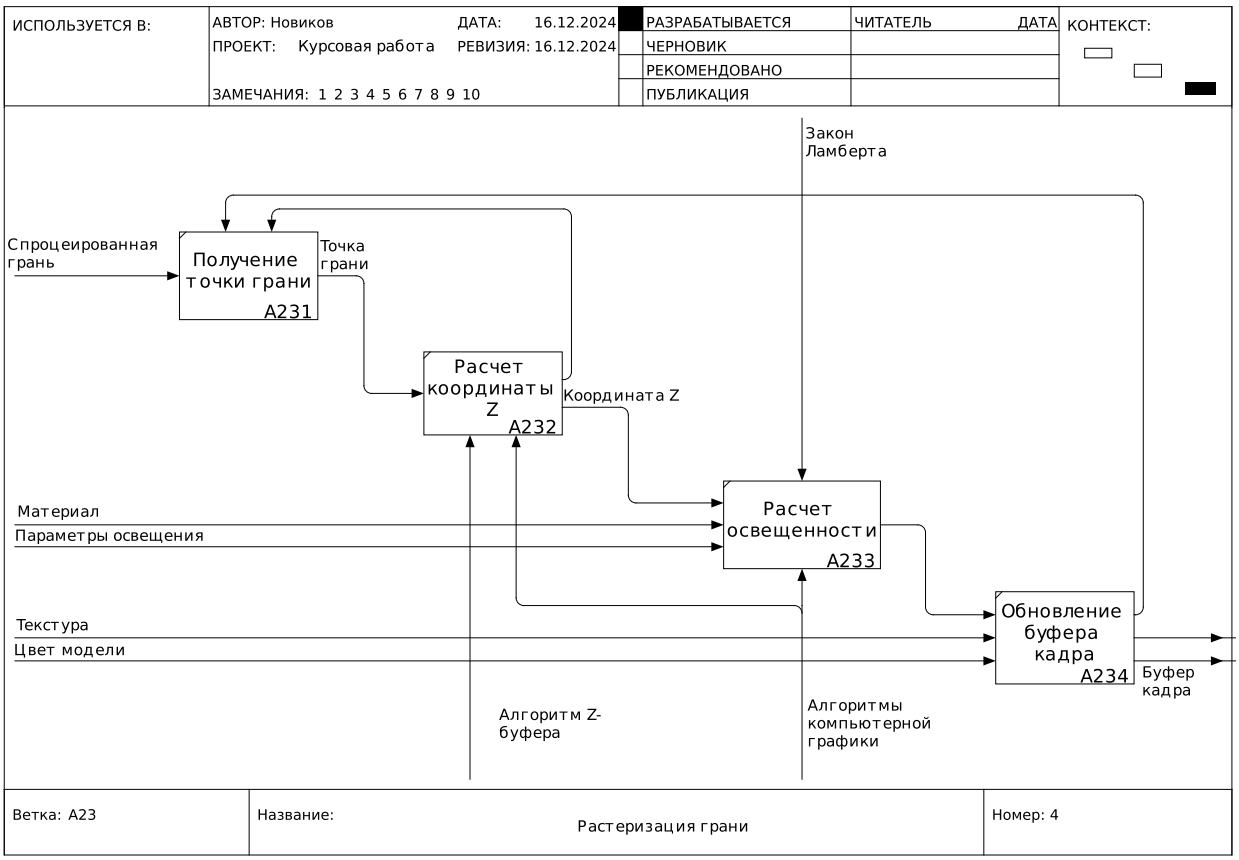

Рисунок 2.4 – Растерицазция грани в нотации IDEF0

2.4 Алгоритм построения кривой

Для построения кривой Безье по заданным точкам $P_1, P_2, ..., P_n$, где P_1, P_n — начальная и конечная точки, $P_2, P_3, ..., P_{n-1}$ — необязательные контрольные точки, задающие форму кривой, используется алгоритм де Кастельжо. Для заданных контрольных точек итеративно вычисляются новые промежуточные точки, для этого отрезки $P_1P_2, P_2P_3, ..., P_{n-1}P_n$ разделяются в отношении $\frac{t}{1-t}$, где $t \in [0, 1]$. Разбиение продолжается до тех пор, пока не останется одна точка, которая будет являться точкой кривой для заданного параметра t.

2.5 Алгоритм построения тела вращения

Для построения тела вращения для каждой точки кривой $C = (P_1, P_2, ..., P_n)$ необходимо вычислить соответствующие точки на поверхности, полученные путем вращения этих точек вокруг выбранной оси. А также сформировать из полученных точек полигоны.

Для каждой точки кривой $P_j = (x_j, y_j)$ генерируются точки вращения

для каждого сегмента (где j – индекс точки на кривой). Вращение каждой точки P_j на угол α, делящий окружность на m частей можно записать как:

$$x_i = x_j + r_j \cdot \cos(\alpha_i),\ y_i = y_j + r_j \cdot \sin(\alpha_i), \tag{2.1}$$

где r_j — расстояние от оси вращения до точки P_j, при вращении вокруг оси Ox $r_j = y_j$, для оси Oy $r_j = x_j$, α_i — угол, соответствующий каждому сегменту окружности:

$$\alpha_i = \frac{2\pi}{m} i,\ i = \overline{0, m-1}. \tag{2.2}$$

Для объединения полученных точек вращения в треугольники точки рассматриваются четверками: две соседние точки на текущем уровне окружности и две соответствующие им точки на следующем уровне окружности. Для каждой четверки точек получается два треугольника.

2.6 Алгоритм, использующий Z-буфер

Алгоритм Z-буфера применяется для удаления невидимых поверхностей. На первом этапе Z-буфер инициализируется максимальными значениями глубины:

$$z_{buffer}[x][y] = z_{max}. \tag{2.3}$$

Далее каждый треугольник сцены проецируется в экранные координаты. Для каждого пикселя треугольника вычисляется значение координаты z по формуле:

$$z = \frac{Ax + By + Cz}{D}, \tag{2.4}$$

где A, B, C, D - коэффициенты, задающие плоскость треугольника. Если значение z для очередного пикселя меньше соответствующего значения в Z-буфере, то значение Z-буфера и цвет пикселя обновляются:

$$z_{buffer}[x][y] = z,\ color[x][y] = color_z. \tag{2.5}$$

2.7 Вычисление нормали

Нормали необходимы для расчета освещенности в точке. Нормаль в вершине треугольника считается как среднее арифметическое между всеми

нормалями треугольников, которые содержат данную вершину:

$$\vec{N} = \frac{\sum_{i=1}^{n} \vec{N}_i}{n}. \tag{2.6}$$

Нормаль треугольника вычисляется по формуле:

$$\vec{N} = \frac{(\vec{V_2} - \vec{V_1}) \times (\vec{V_3} - \vec{V_1})}{|(\vec{V_2} - \vec{V_1}) \times (\vec{V_3} - \vec{V_1})|}, \tag{2.7}$$

где $\vec{V_1}, \vec{V_2}, \vec{V_3}$ — это вектора, определяющие положение вершин треугольника в пространстве. Операции выполняются над самими векторами, которые представляют массивы координат. После подсчета нормали, ее необходимо нормализовать, т.е. привести к единичной длине:

$$\vec{N}_{normalize} = \frac{\vec{N}}{|\vec{N}|}, \tag{2.8}$$

где $\vec{N}_{normalize}$ — нормализованный вектор вершины.

2.8 Расчет освещения

В программе используется направленный источник света, расположенный в заданной точке. В качестве модели освещения была выбрана модель Ламберта. Освещенность I можно вычислить по формуле:

$$I = I_{ambient} + I_{diffuse}, \tag{2.9}$$

где $I_{ambient}$ – фоновая составляющая освещения, $I_{diffuse}$ — рассеянная составляющая освещения.

Фоновое освещение — это постоянная в каждой точке величина надбавки к освещению, вычисляемая по формуле:

$$I_{ambient} = k_{ambient} \cdot i_{ambient}, \tag{2.10}$$

где $k_{ambient}$ — коэффициент, характеризующий способность материала воспринимать фоновое освещение, $i_{ambient}$ — мощность фонового освещения.

Рассеянная составляющая рассчитывается по формуле:

$$I_{diffuse} = k_{diffuse} \cdot i_{diffuse} \cdot \cos(\vec{N} \cdot \vec{L}), \tag{2.11}$$

где $k_{diffuse}$ — коэффициент, характеризующий способность свойство материала воспринимать рассеянное освещение, $i_{diffuse}$ — мощность рассеянного освещения, $\vec{N}$ — единичный вектор нормали, $\vec{L}$ — единичный вектор направления из точки на источник света.

2.9 Наложение текстуры

Для наложение текстуры на поверхность тела вращения необходимо воспользоваться UV-преобразованием. UV-преобразование — это соответствие между координатами трехмерного объекта и координатами на текстуре. Значения U, V изменяются в пределах от 0 до 1. Значение координаты V соответствует вертикальной оси текстуры, U — горизонтальной. Зная UV-координаты для точки, можно найти соответствующий ей пиксель текстуры. Для этого необходимо значение U умножить на ширину изображения, а V — на высоту изображения.

Для каждой точки тела вращения необходимо найти соответствующее значение UV-координаты. Для вычисления V необходимо сначала определить длину кривой, которая задается суммой длин всех ее сегментов:

$$L_{total} = \sum_{i=0}^{n-2} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}. \tag{2.12}$$

Для каждой точки P_j на кривой вычисляется расстояние от начала кривой до P_j:

$$L_{point} = \sum_{k=0}^{j-1} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2}. \tag{2.13}$$

Значение V координаты вычисляется как отношение L_{point} к L_{total}:

$$V_j = \frac{L_{point}}{L_{total}}. \tag{2.14}$$

Координата U определяется для каждой точки, возникающей в процессе вращения. Поскольку точки генерируются равномерно вдоль окружности, U

считается по формуле:

$$U_i = \frac{i}{m}, \tag{2.15}$$

где i – номер текущего сегмента вращение ($i = \overline{0, m-1}$, m — общее число сегментов.

2.10 Визуализация фактуры

Для визуализации фактуры на теле вращения был выбран метод normal mapping, который предполагает использование карты нормалей для изменения текущей нормали в точке. Карта нормалей — это тип текстурной карты, в которой хранятся данные о нормалях к поверхности в виде RGB-изображения [10]. Значения R, G и B соответствуют значению отклонению нормали по координатам x, y, z соответственно.

Так как значения каналов R, G, B в текстуре находятся в диапазоне $[0, 255]$, а координаты нормали обычно описываются в диапазоне $[-1, 1]$, необходимо произвести следующие преобразования:

$$N_{map} = \frac{2}{255} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \tag{2.16}$$

где N_{map} — значение отклонения нормали в точке, R, G, B — значения каналов цвета в точке.

Для каждой точки на теле вращения при подсчете освещения необходимо выбрать соответствующий ей пиксель из карты нормалей, найти значение N_{map} и прибавить его к текущей нормали в точке.

ВЫВОД

В данном разделе были представлены требования к программному обеспечению, рассмотрены структуры данных, алгоритмы и математические уравнения, выбранные для построения сцены.

3 Технологический раздел

В данной части рассматривается выбор средств реализации, описывается структура классов программы и приводится интерфейс программного обеспечения.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования C++ [11].

В качестве среды разработки была выбрана Qt Creator [12].

Выбор обсуловлен наличием проработанной стандартной библиотеки, включающей в себя классы для работы с векторами и матрицами, большим количеством инструментов для работы с пользовательским интерфейсом. Используемые инструменты обладают полной функциональностью для разработки, профилирования и отладки необходимой программы.

3.2 Структура программы

Разработанное программное программное обеспечение состоит из следующих классов. Математические классы:

— QPointF — класс для работы с точкой в двухмрном пространстве;

— QVector3D — класс для работы с трехмерными векторами;

— QMatrix4x4 — класс для работы с матрицами 4x4,

Классы для работы с объектами сцены:

— AbstractModel — абстрактный класс объекта сцены;

— Point — класс точки в трехмерном пространстве;

— Model — класс представляющий трехмерное тело;

— AbstractLight — абстрактный класс источника света;

— DirectionalLight — класс направленного источника света;

— AmbientLight — класс фонового освещения.

Классы для визуализации сцены:

— Scene — класс сцены;

— AbstractDrawVisitor — абстрактный класс посетителя для отрисовки объектов сцены;

— DrawVisitor — класс для отрисовки объекта заданным цветом;

— DrawMappedVisitor — класс для отрисовки объекта с визуализацией фактуры объекта;

— DrawTextureVisitor — класс для отрисовки с визуализацией текстуры объекта;

— ProjectStrategy — абстрактный класс стратегии процеирования трехмерной точки на плоскость;

— PerspectiveStrategy — класс перспективной процекуии точки на плоскость.

Классы интерфейса:

— MainWindow — основное окно программы;

— ModelViewer — окно для просмотра тела вращения;

— CurveCanvas — холст для ввода пользователем направляющей.

Вспомогательные классы:

— Curve — класс представления направляющей, введенной пользователем;

— Facade — класс для унификации доступа к сцене.

На рисунке 3.1 представлена диаграмма разработанных классов:

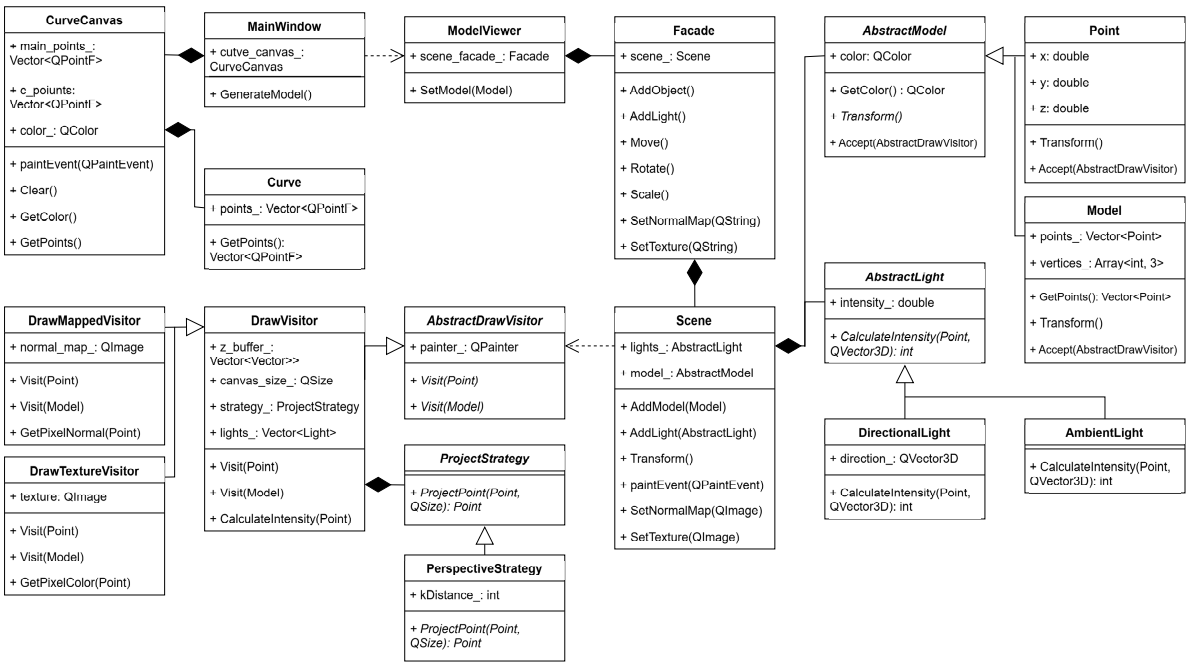

Рисунок 3.1 – Диаграмма классов программы

3.3 Схемы алгоритмов

Схемы алгоритмов представлены на рисунках 3.2- 3.5

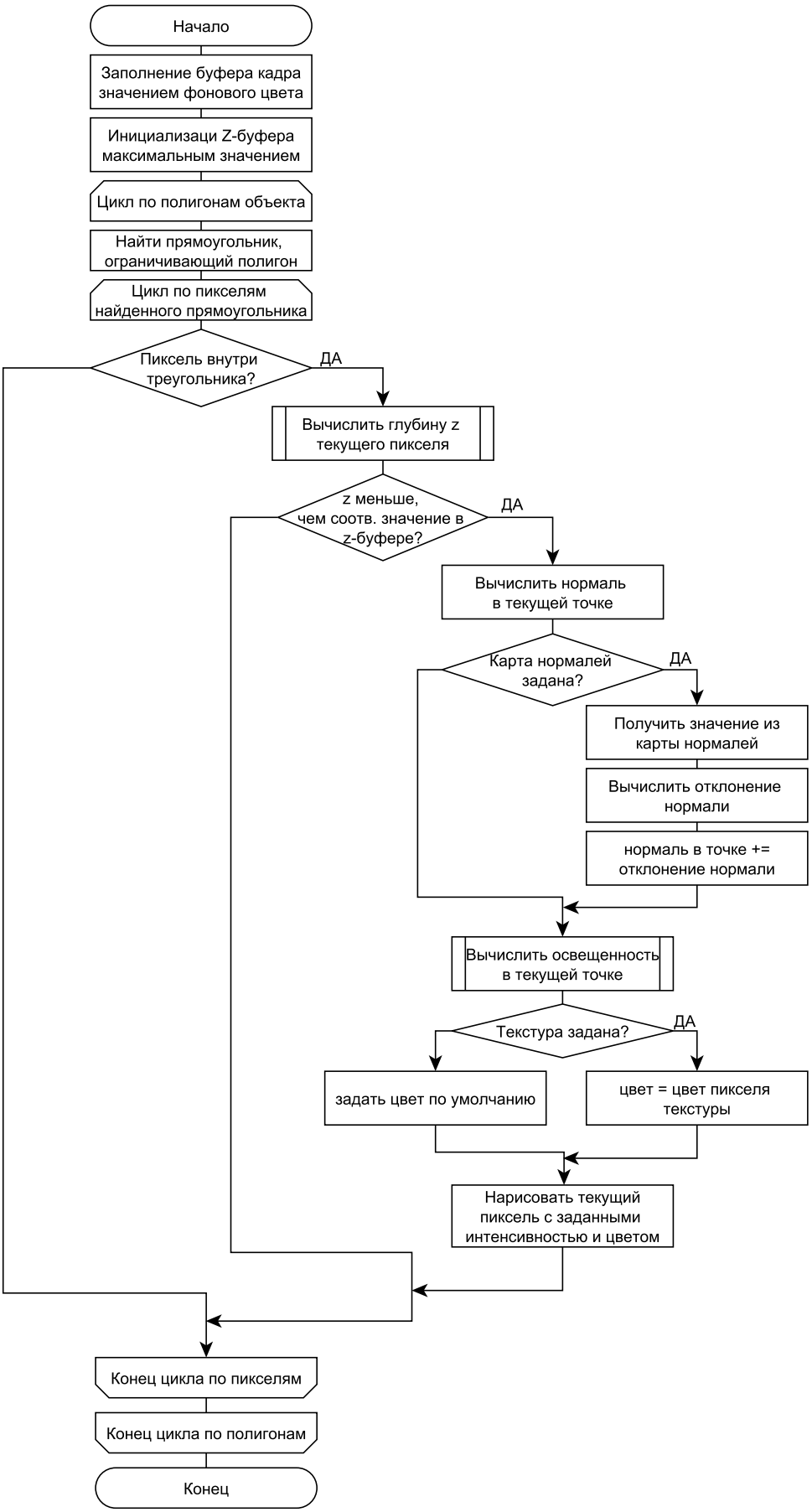

Рисунок 3.2 – Схема алгоритма с Z-буфером

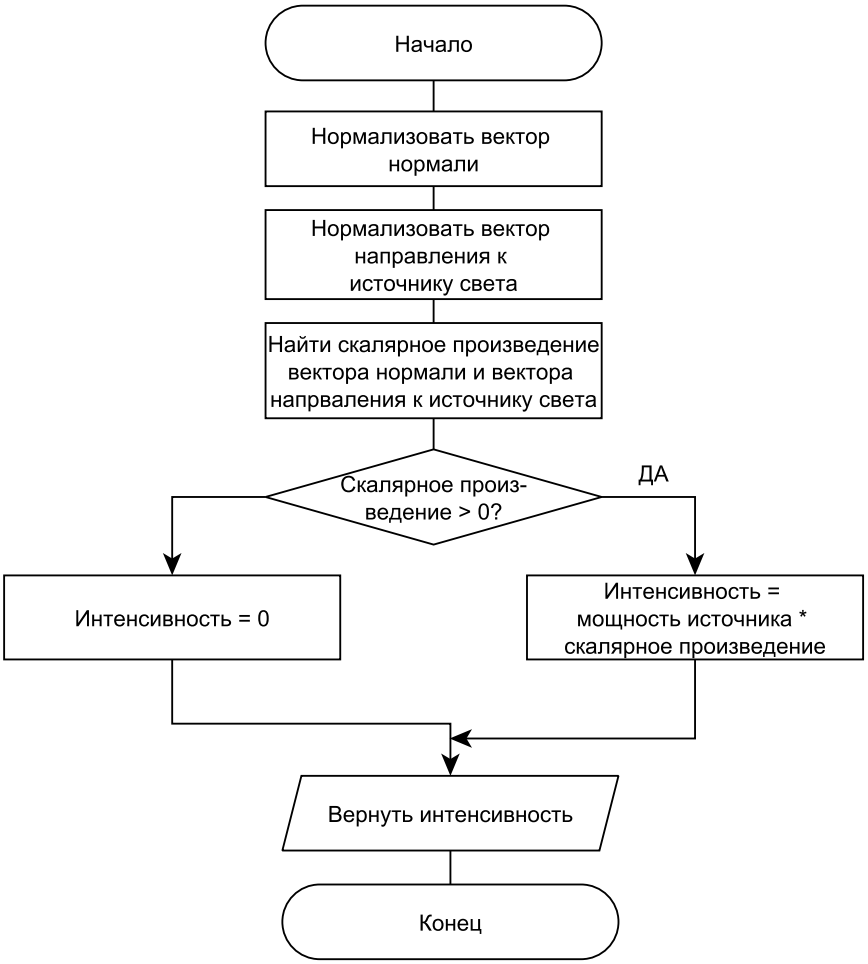

Рисунок 3.3 – Схема алгоритма подсчета интенсивности света в точке

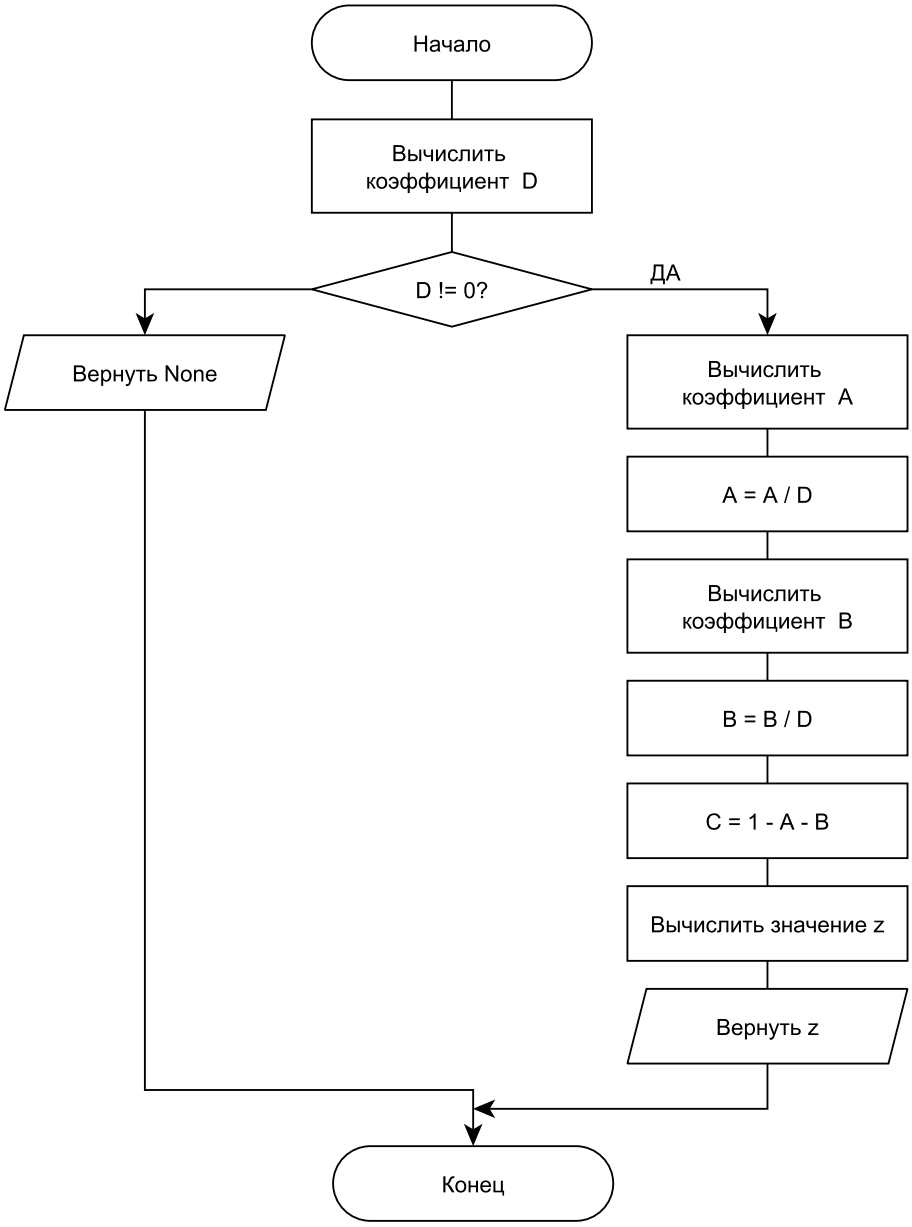

Рисунок 3.4 – Алгоритм вычисления значения глубины в точке

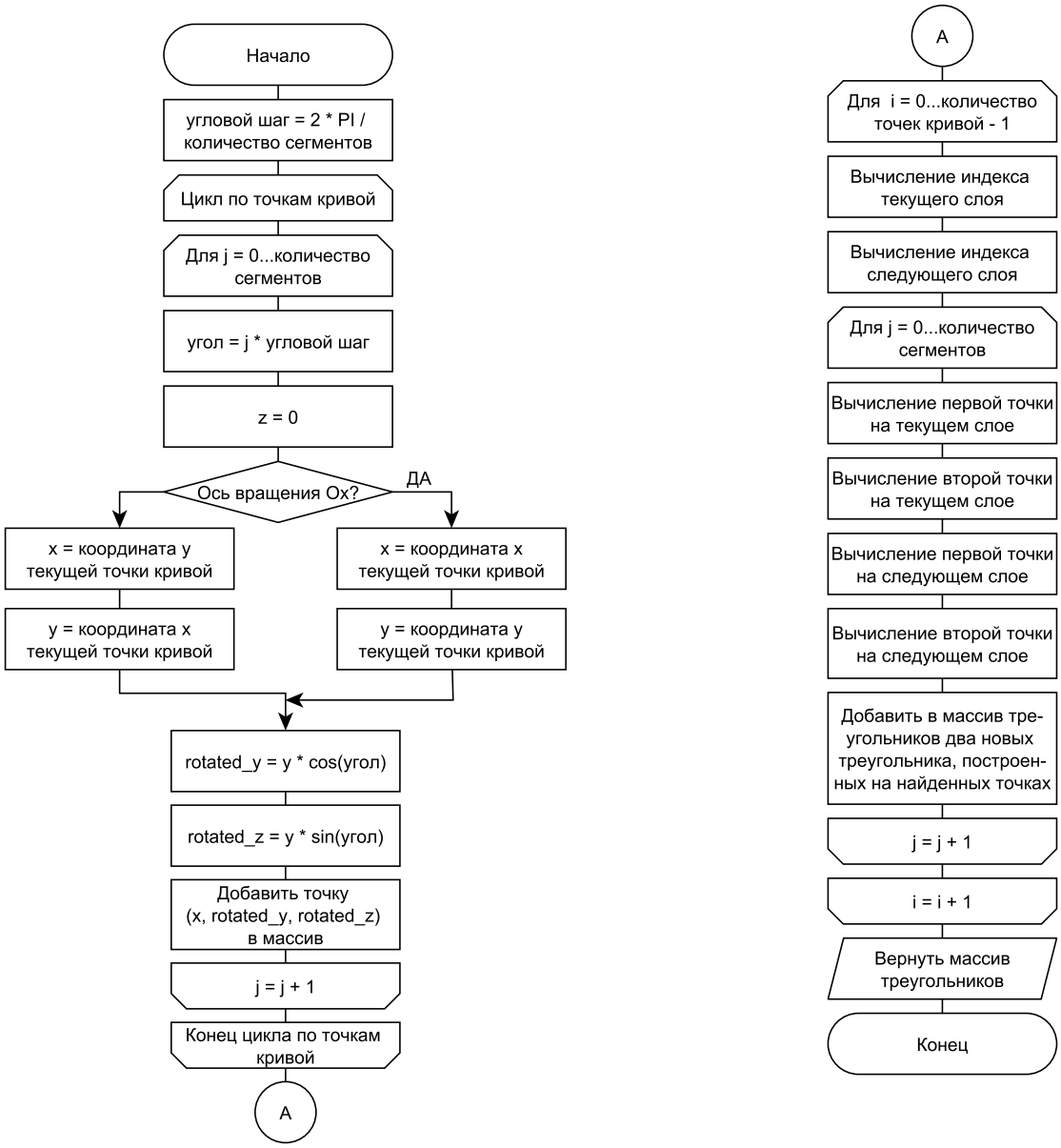

Рисунок 3.5 – Алгоритм получения тела вращения

3.4 Интерфейс программного обеспечения

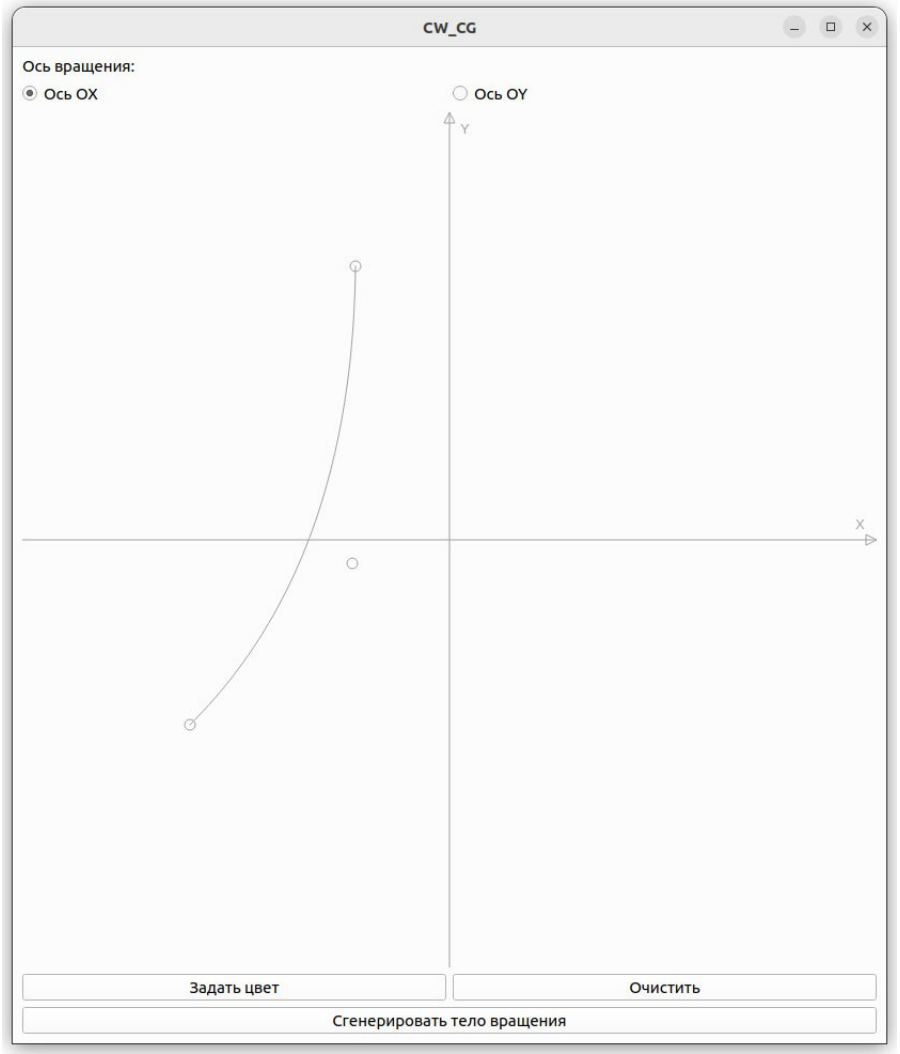

Рисунок 3.6 – Графический интерфейс программы

Главное окно программы представлено на рисунке 3.6. При запуске программы в центре находится координатная плоскость для ввода пользователем направляющей. В верхней части окна находится список для выбора пользователем оси вращения. В нижней части окна находятся кнопки управления программой: кнопка задания цвета тела, кнопка очистки введенных точек, кнопки генерации тела вращения.

Пользователь вводит начальную и конечную точки нажатием левой кнопки мыши, а также необязательно задает контрольные точки правой кнопкой мыши, которые задают кривизну. При попытке ввести более двух начальных точек или ввести контрольные точки до задания начальных точек пользователь будет уведомлен ошибкой.

Рисунок 3.7 – Окно просмотра полученного объекта

После генерации тела вращения оно актуализируется в новом окне. Окно визуализации представлено на рисунке 3.7. В верхней части экрана располагаются элементы управления визуализацией текстуры и фактуры материала.

ВЫВОД

В данном разделе были выбраны средства реализации, описаны структуры классов программы, описаны модули, а также рассмотрен интерфейс программы.

4 Исследовательский раздел

В данном разделе приведены характеристики устройства, на котором проводилось измерение времени работы программного обеспечение, результаты проведенных замеров.

4.1 Технические характеристики

Технические характеристики используемого устройства:

— операционная система — Ubuntu Linux x86_64 [13];

— память — 16 Гб;

— процессор — AMD Ryzen 5 5500U (6x2.10 ГГц) [14].

4.2 Замеры времени

Время обрисовки затерялось на компьютере с указанными техническими характеристиками и использованием библиотеки $<ctime>$ [15]. Были проведены замеры времени отрисовки одного кадра в зависимости от количества точек задаваемой пользователем кривой и времени отрисовки одного кадра в зависимости от числа разбиений окружности при построении тела.

На рисунке 4.1 приведены результаты замеров времени отрисовки одного кадра в зависимости от числа разбиений окружности при построении тела вращения. Для каждого числа разбиений время посчитано как среднее от 100 проведенных замеров. Замеры проводились при фиксированном количестве точек кривой.

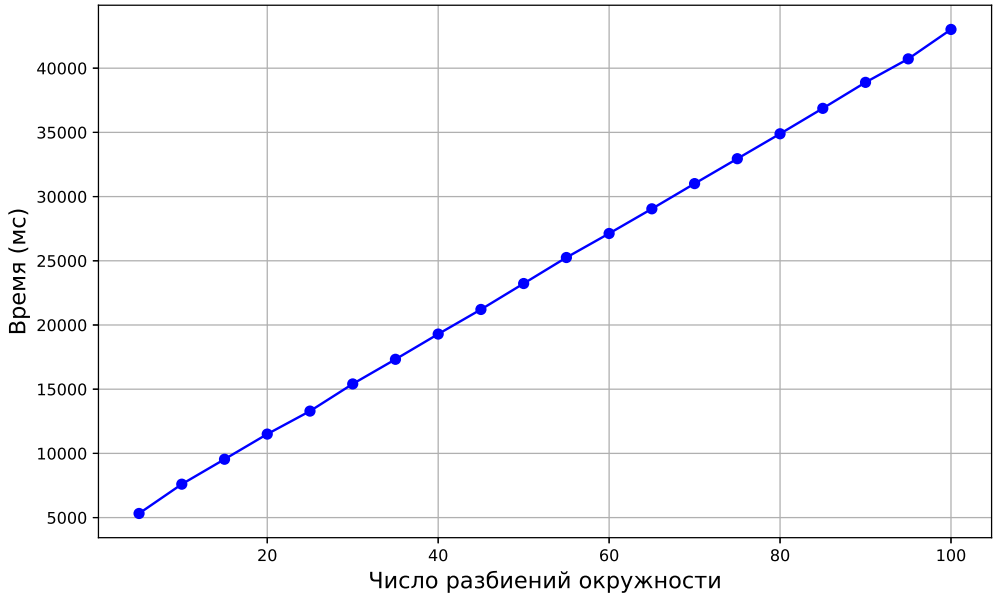

Рисунок 4.1 – Зависимость времени отрисовки одного кадра от числа разбиений окружности

На рисунке 4.2 приведены результаты замеров времени отрисовки одного кадра в зависимости от количества точек кривой. Для каждого числа точек время посчитано как среднее от 100 проведенных замеров. Замеры проводились при фиксированном числе разбиений окружности.

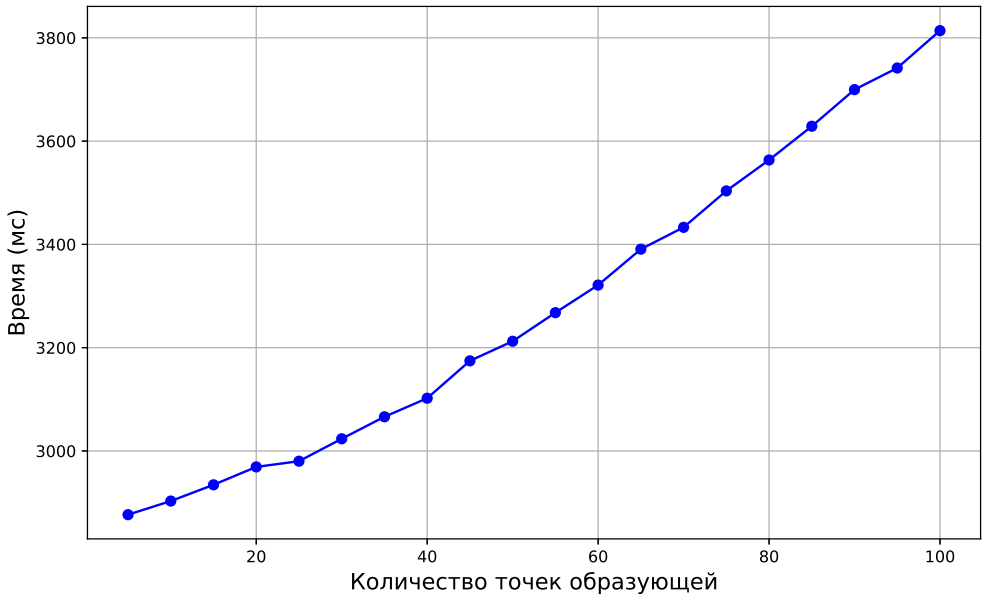

Рисунок 4.2 – Зависимость времени отрисовки одного кадра от количества точек кривой

В результате исследования была получено, что зависимость времени отрисовки кадра от числа разбиений окружности и количества точек кривой линейна. Такая зависимость наблюдается вследствие пропорционального увеличения количества треугольников в модели, которые необходимо обработать, при увеличении данных параметров.

ВЫВОД

В данном разделе приведены результаты работы программного обеспечения и результаты исследования. Результаты исследования совпали с ожидаемыми, так как время отрисовки сцены прямо пропорционально числу разбиений окружности и количеству точек кривой.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была достигнута поставленная цель: было разработано программное обеспечение для визуализации тел вращения с добавлением фактуры и текстуры материала. В процессе работы были решены следующие задачи:

— описана предметная область работы;

— рассмотрены и выбраны алгоритмы построения реалистичного тела вращения;

— на основе выбранных алгоритмов было спроектировано программное обеспечение;

— было реализовано спроектированное программное обсечение;

— проведено исследование на основе разработанной программы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Ханов Г.*, *Безрукова Т.* 3D-моделирование в инженерной графике: учебное пособие // Волгоград. — 2015. — С. 55.
2. *Скворцов Л. М.* Точность методов Рунге–Кутты при решении жестких задач // Журнал вычислительной математики и математической физики. — 2003. — Т. 43, № 9. — С. 1374—1384.
3. *Покорная И.*, *Суворова Е.* Основы построения: сплайны и кривые Безье // Некоторые вопросы анализа, алгебры, геометрии и математического образования. — 2017. — № 6. — С. 149—150.
4. Алгоритм Варнока. [Электронный ресурс]. — URL: `http://compgraph.tpu.ru/Oglavlenie.htm` (дата обращения 01.10.2024).
5. Ю.М.Баяковский. Трассировка лучей из книги Джефа Проузиса [Электронный ресурс]. — URL: `http://www.graphicon.ru/oldgr/courses/cg99/notes/lect12/prouzis/raytrace.htm` (дата обращения 01.10.2024).
6. *Задорожный А. Г.* Модели освещения и алгоритмы затенения в компьютерной графике: учебное пособие // Новосибирск: Изд-во НГТУ. — 2020. — С. 80.
7. *Пауков Н.*, *Соломатин Д.* Сравнение и реализация моделей освещения в приложениях компьютерной графики реального времени // Сборник студенческих научных работ факультета компьютерных наук ВГУ. — 2019. — С. 169—177.
8. *Серегин К. В.* Особенности моделирования света: рельефное текстурирование (bump mapping) // Труды Дальневосточного государственного технического университета. — 2004. — № 137. — С. 221—223.
9. 3D Modeling Fundamentals - Part 2 [Электронный ресурс]. — URL: `https://web.archive.org/web/20131212064631/gameindustry.about.com/od/game-development/a/3d-Modeling-Fundamentals-Part-2.htm#` (дата обращения 04.10.2024).
10. *Канаев Д. К.*, *Чернова С. В.* Детализация текстурой. Карты нормалей в игровом 3D-моделировании // Научный электронный журнал Меридиан. — 2021. — № 2. — С. 153—155.

11. C++ zprogramminglanguage [Электронный ресурс]. — URL: https://isocpp.org/ (дата обращения 12.11.2024).

12. Qt Creator documentation [Электронный ресурс]. — URL: https://doc.qt.io/qtcreator/ (дата обращения 12.11.2024).

13. Ubuntu technical documentation for developers and IT pros [Электронный ресурс]. — URL: https://ubuntu.com/tutorials (дата обращения 25.10.2024).

14. AMD Ryzen 5 5500U Processor [Электронный ресурс]. — URL: https://www.amd.com/en/products/apu/amd-ryzen-5-5500u (дата обращения 25.10.2024).

15. C Time Library [Электронный ресурс]. — URL: https://cplusplus.com/reference/ctime/ (дата обращения 26.11.2024).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе состоит из 15 слайдов.