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 (ИУ)_____

КАФЕДРА _____Программное обеспечение ЭВМ и информационные технологии (ИУ7)____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Создание и редактирование трехмерных моделей.

Студент _____ИУ7-51Б_____ _____ ____**Н.А.Котляров**_____
(Группа) (Подпись, дата)
(И.О.Фамилия)

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

2022 г.

Оглавление

Введение

Компьютерная графика (также машинная графика) — область деятельности, в которой компьютеры наряду со специальным программным обеспечением используются в качестве инструмента как для создания (синтеза) и редактирования изображений, так и для оцифровки визуальной информации, полученной из реального мира, с целью дальнейшей ее обработки и хранения.

В современном мире яркими примерами использования компьютерной графики являются: визуализация данных на производстве, создание реалистичных специальных эффектов в кино и для создания видеоигр. При этом из-за разницы в задачах могут быть использованы различные подходы и алгоритмы. Так, компьютерная графика в кино требует реалистичного изображения, в то время как задача визуализации данных в производстве требует лишь понятного представления трехмерного объекта.

В компьютерной графике существует множество разнообразных алгоритмов, помогающих решить эти задачи, однако большинство являются достаточно ресурсоемкими, так как чем более реалистичное изображение мы хотим получить, тем больше нам необходимо времени и памяти на синтез. При этом, если от изображения требуется быть интуитивно понятным представлением трехмерного объекта, также не стоит пренебрегать долей реалистичности.

Целью данной курсовой работы является приложение для создания и редактирования трехмерных объектов, которые впоследствии можно будет использовать в различных нуждах.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- проанализировать существующие представления трехмерных объектов и обосновать выбор тех из них, которые в наибольшей степени подходят для решения поставленной задачи;
- проанализировать существующие алгоритмы построения изображения и обосновать выбор тех из них, которые в наибольшей степени подходят для решения поставленной задачи;

- реализовать выбранные алгоритмы;
- реализовать возможность изменения положения модели в пространстве;
- реализовать возможность создания полигонов как части модели;
- реализовать возможность удаления полигонов модели;
- реализовать возможность изменения формы модели;
- реализовать возможность сохранения модели в файл;
- реализовать возможность чтения модели из файла;
- реализовать возможность изменение цвета грани модели;

1 Аналитический раздел

В данной части проводится анализ объектов сцены и существующих алгоритмов построения изображений и выбор более подходящих алгоритмов для дальнейшего использования.

1.1 Описание и формализация объектов сцены

Источник света – материальная точка в пространстве, из которой исходят лучи во все стороны. Положение на сцене регулируется координатами (x, y, z), может иметь цвет.

Для описания трехмерных геометрических объектов существует три модели: каркасная, поверхностная и объемная. Для реализации поставленной задачи, более подходящей моделью будет поверхностная, так как, по сравнению с каркасной, она может дать более реалистичное и понятное зрителю изображение. В то же время, объёмной модели требуется больше памяти, следовательно, она излишняя.

В свою очередь поверхностная модель может задаваться параметрическим представлением или полигональной сеткой.

При параметрическом представлении поверхность можно получить при вычислении параметрической функции, что очень удобно при просчете поверхностей вращения, однако, ввиду их отсутствия, данное представление будет не выгодно.

В случае полигональной сетки форма объекта задаётся некоторой совокупностью вершин, рёбер и граней, что позволяет выделить несколько способов представления:

1) Вершинное представление — в этом представлении вершины хранят указатели на соседние вершины. В таком случае для рендеринга нужно будет обойти все данные по списку, что может занимать достаточно много времени при переборе.

2) Список граней — в этом представлении объект хранится, как множество граней и вершин. В таком случае достаточно удобно производить

различные манипуляции над данными.

3) Таблица углов — в этом представлении вершины хранятся в предопределенной таблице, такой, что обход таблицы неявно задает полигоны. Это представление более компактное и более производительное для нахождения полигонов, однако операции по замене достаточно медлительны.

Наиболее подходящим вариантом в условиях поставленной задачи будет представление в виде списка граней, т.к. оно позволяет эффективно манипулировать данными, а также позволяет проводить явный поиск вершин грани и самих граней, которые окружают вершину.

1.2 Анализ алгоритмов удаления невидимых линий и поверхностей

Выбирая подходящий алгоритм удаления невидимых линий, необходимо учитывать поставленную задачу и ее особенности, а именно то, что сцена должна быть динамической, а следовательно, алгоритм должен быть достаточно быстрым.В настоящее время существует множество различных алгоритмов построения изображений: алгоритм трассировки лучей, алгоритм Варнока, алгоритм Робертса, алгоритм Z-буфера и т.д. Рассмотрим преимущества и недостатки каждого из алгоритмов.

1.2.1 Алгоритм трассировки лучей

Данный алгоритм обладает следующими преимуществами:

- вычислительная сложность слабо зависит от сложности сцены;
- отсечение невидимых поверхностей, перспектива и корректное изменения поля зрения являются логическим следствием алгоритма;
- возможность рендеринга гладких объектов без аппроксимации их полигональными поверхностями;

- возможность параллельно и независимо трассировать два и более лучей, разделять участки (или зоны экрана) для трассирования на разных узлах кластера и т.д.

Однако, недостатком алгоритма является его производительность. Метод трассирования лучей каждый раз начинает процесс определения цвета пикселя заново, рассматривая каждый луч наблюдения в отдельности [1].

В поставленной задаче не используются явления отражения и преломления света и поэтому некоторые вычисления окажутся излишними, кроме того, скорость синтеза сцены должна быть высокой, из-за чего алгоритм трассировки лучей не подходит.

1.2.2 Алгоритм Варнока

Преимуществом данного алгоритма является то, что он работает в пространстве изображений. Алгоритм предлагает разбиение области рисунка на более мелкие окна, и для каждого такого окна определяются связанные с ней многоугольники и те, видимость которых «легко» определить, изображаются на сцене [2].

К недостаткам можно отнести то, что в противном случае разбиение повторяется, и для каждой из вновь полученных подобластей рекурсивно применяется процедура принятия решения. По предположению, с уменьшением размеров области, её будет перекрывать всё меньшее количество многоугольников. Считается, что в пределе будут получены области, которые содержат не более одного многоугольника, и решение будет принято достаточно просто.

Из-за того, что поиск может продолжаться до тех пор, пока либо остаются области, содержащие не один многоугольник, либо пока размер области не станет совпадать с одним пикселем, алгоритм не подходит под условия моей задачи.

1.2.3 Алгоритм Робертса

Данный алгоритм обладает достаточно простыми и одновременно мощными математическими методами, что, несомненно, является преимуществом. Некоторые реализации, например использующие предварительную сортировку по оси z, могут похвастаться линейной зависимостью от числа объектов на сцене [3].

Однако алгоритм обладает недостатком, который заключается в вычислительной трудоемкости, растущей теоретически, как квадрат числа объектов, что может негативно сказаться на производительности. В то же время реализация оптимизированных алгоритмов достаточно сложна.

1.2.4 Алгоритм, использующий Z буфер

Главными преимуществами данного алгоритма являются его достаточная простота реализации и функциональность, которая более чем подходит для визуализации динамической сцены. Также алгоритм не тратит время на сортировку элементов сцены, что даёт значительный прирост к производительности [4].

К недостатку алгоритма можно отнести большой объём памяти необходимый для хранения информации о каждом пикселе. Однако данный недостаток является незначительным ввиду того, что большинство современных компьютеров обладает достаточно большим объёмом памяти для корректной работы алгоритма.

Вывод

Алгоритм, использующий Z-буфер, является наиболее подходящим для создания и редактирования трехмерных моделей. На его основе будет просто визуализировать схематичную модель, показывающую требуемые в производстве нюансы.

1.3 Анализ моделей освещения

На сегодняшний день существует две модели освещения, которые используются для построения света на трехмерных сценах: локальная и глобальная. Исходя из поставленной задачи, необходимо выбрать более производительный вариант, который позволит достаточно быстро просчитывать освещения на сцене.

1.3.1 Локальная модель освещения

Является самой простой и не рассматривает процессы светового взаимодействия объектов сцены между собой и рассчитывает освещённость только самих объектов [5].

1.3.2 Глобальная модель освещения

Данная модель рассматривает трехмерную сцену, как единую систему и описывает освещение с учетом взаимного влияния объектов, что позволяет рассматривать такие явления, как многократное отражение и преломление света, а также рассеянное освещение [5].

Вывод

Так как программа не должна выводить реалистичного изображения, рациональнее будет использовать локальную модель освещения, что значительно упростит вычисления и повысит производительность редактора.

1.4 Анализ алгоритмов закрашивания

1.4.1 Алгоритм простой закраски

По закону Ламберта, вся грань закрашивается одним уровнем интенсивности. Метод является достаточно простым в реализации и не требовательным к ресурсам. Однако алгоритм плохо учитывает отражения и при отрисовки тел вращения, возникают проблемы [6].

1.4.2 Алгоритм закраски по Гуро

В основу данного алгоритма положена билинейная интерполяция интенсивностей, позволяющая устранять дискретность изменения интенсивности. Благодаря этому криволинейные поверхности будут более гладкими[6].

1.4.3 Алгоритм закраски по Фонгу

В данном алгоритме за основу берётся билинейная интерполяция векторов нормалей, благодаря чему достигается лучшая локальная аппроксимация кривизны поверхности и, следовательно, изображение выглядит более реалистичным[6].

Однако алгоритм требует больших вычислений, по сравнению с алгоритмом по Гуро, так как происходит интерполяция значений векторов нормалей.

Вывод

В условиях поставленной задачи, рациональнее будет использовать алгоритм простой закраски, так как программа не должна выводить реалистичное изображение и при этом должна быть достаточно быстрой. Кроме

того, алгоритм достаточно хорошо сочетается с выбранным ранее Z буфером.

2 Конструкторская часть

2.1 Описание объектов сцены

На рисунке 2.1 видно, что модель представляет собой список вершин, список ребер, которые хранят в себе две ссылки на разные вершины, и список граней, которые хранят в себе ссылки на три разные вершины и ссылки на три грани, связывающие эти вершины.

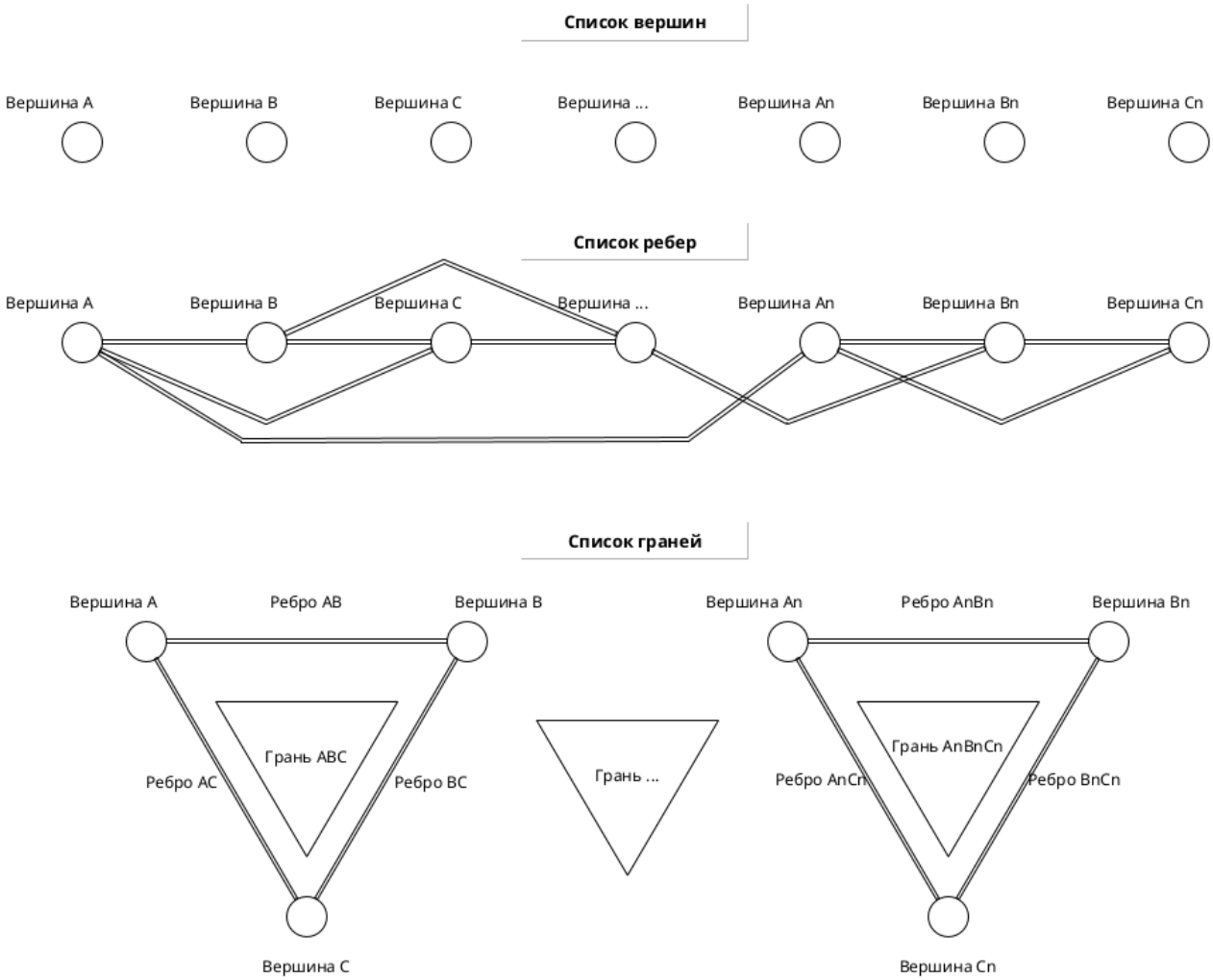

Рисунок 2.1 – Схематичное представление хранения данных в модели

2.2 Редактирование, создание и удаление частей модели

Изменение положения вершин — можно менять как положение отдельных вершин/ребер/граней, так и всей модели. Изменение положения вершин производится с помощью матрицы преобразования. Каждая вершина имеет собственную матрицу преобразования. Для получения нового положения вершины нужно умножить матрицу преобразования этой вершины на матрицу, соответствующую шаблону, описанному ниже.

Получение новой матрицы преобразования:

$$\begin{pmatrix}\text{Новая матрица}\\ \text{преобразования } 4\times 4\end{pmatrix} = \begin{pmatrix}\text{Текущая матрица}\\ \text{преобразования } 4\times 4\end{pmatrix}\cdot\begin{pmatrix}\text{Шаблонная матрица}\\ \text{преобразования } 4\times 4\end{pmatrix}$$

Положение вершины в экранных координатах находится следующим образом:

$$\begin{pmatrix} x_{new} & y_{new} & z_{new} & w_{not\ used}\end{pmatrix} = \begin{pmatrix} x_{old}\\ y_{old}\\ z_{old}\\ 1\end{pmatrix}\cdot\begin{pmatrix}\text{Матрица преобразования}\\ 4\times 4\end{pmatrix}$$

Изменение положения грани или ребра модели производится с помощью изменения положения вершин, ссылками на которые обладает грань или вершина. При этом деформируются грани и ребра, также обладающие ссылками на эти вершины:

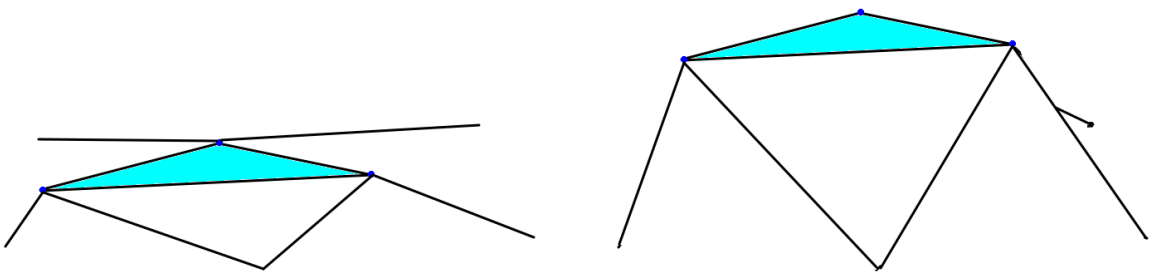

Рисунок 2.2 – Перемещение отдельной грани

Создание новых граней — можно как создать несколько новых вершин, так и использовать уже имеющиеся. В этом случае новые вершины, ребра и новая грань добавляются к соответствующим спискам, хранящимся в объекте модели.

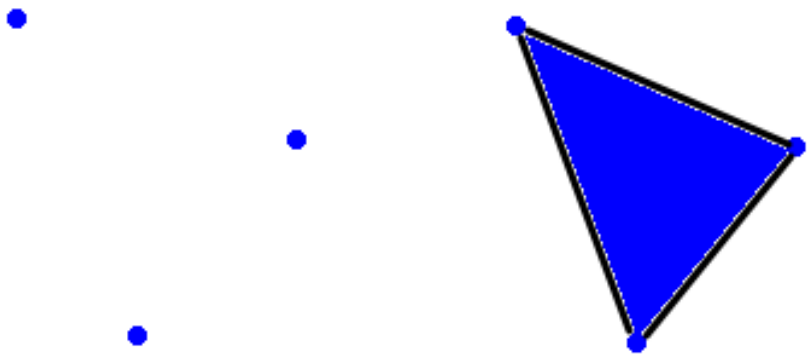

Рисунок 2.3 – Создание грани по трем вершинам, ранее не существующим в модели

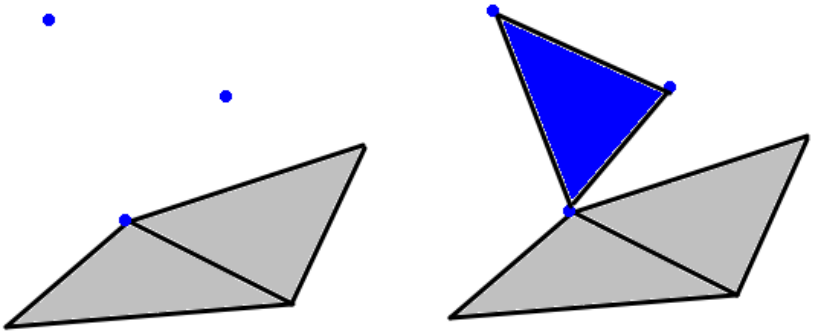

Рисунок 2.4 – Создание грани по двум вершинам, ранее не существующим в модели, и одной вершине, связанной с другими гранями

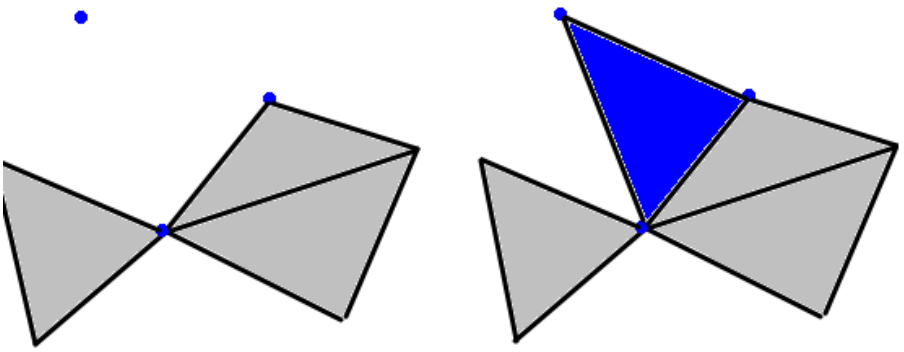

Рисунок 2.5 – Создание грани одной вершине, ранее не существующей в модели, и двум вершинам, связанным с другими гранями

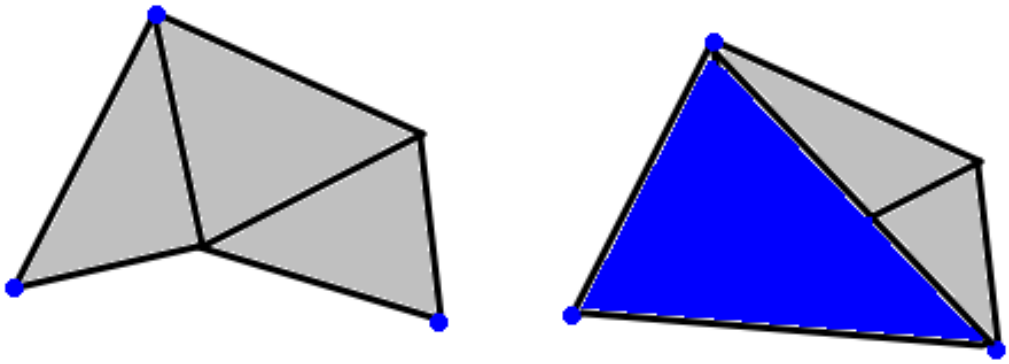

Рисунок 2.6 – Создание грани по трем вершинам, связанными с другими гранями

Удаление граней — удалять можно только грани, однако могут удалиться некоторые связанные с гранью ребра и вершины, если они более ни с одной из граней модели не связаны. Удаление вершин, ребер и граней происходит посредством удаления соответствующих объектов из списков, хранящимся в объекте модели.

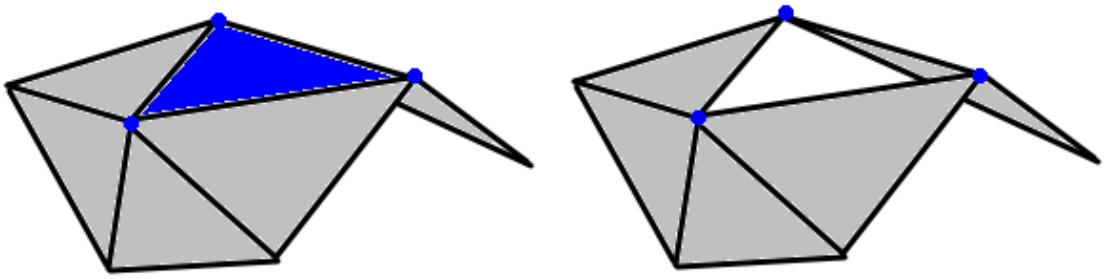

Рисунок 2.7 – Удаление грани в случае, когда не происходит удаления вершин и ребер, принадлежащих этой грани

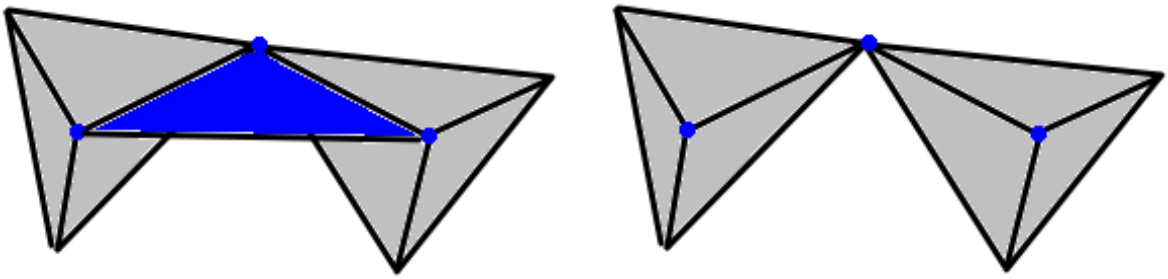

Рисунок 2.8 – Удаление грани в случае, когда удаляется только одно ребро, принадлежащее этой грани

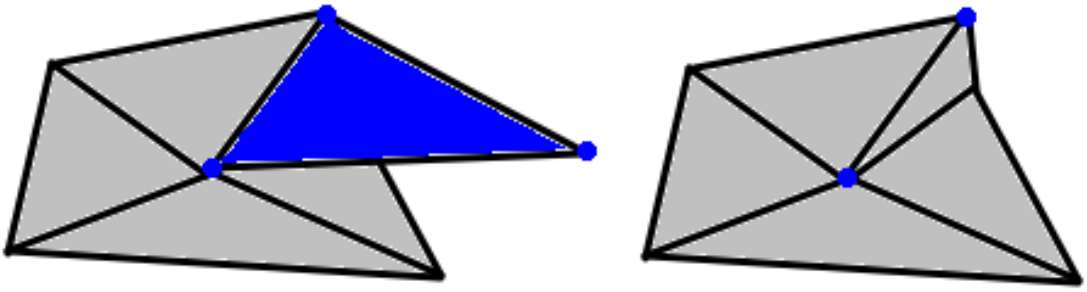

Рисунок 2.9 – Удаление грани в случае, когда удаляется только два ребра и одна вершина, принадлежащие этой грани

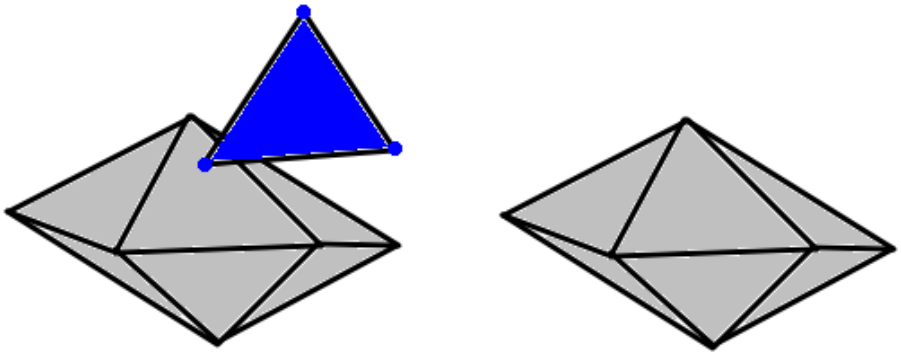

Рисунок 2.10 – Удаление грани в случае, когда удаляются все ребра и вершины, принадлежащие этой грани

2.3 Алгоритм Z-буфера

Данный алгоритм является одним из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Кэтмулом. Алгоритм работает в пространстве изображения, а сама идея z-буфера является простым обобщением идеи о буфере кадра, который используется для запоминания атрибутов или интенсивности каждого пикселя в пространстве изображения.

Z-буфер — это отдельный буфер глубины, который используется для запоминания координаты z или глубины каждого видимого пикселя в пространстве изображения. В процессе работы глубина или значение z каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в z-буфере.

Если это сравнение показывает, что новый пиксель расположен впереди пикселя, который находится в буфере кадра, то новое значение заносится в буфер и производится корректировка z-буфера новым значением z. Если при сравнении получается противоположный результат, то никаких действий не производится.

Алгоритм представляет из себя поиск по каждой точке наибольшего значения функции $z(x, y)$. Формальное описание алгоритма z-буфера:

1) заполнить буфер кадра фоновым значением интенсивности или цвета;

2) провести инициализацию Z-буфера минимальным значением глубины;

3) преобразовать каждый многоугольник в растровую форму в произвольном порядке;

   (a) для всех пикселей, которые связаны с многоугольником, вычислить его глубину $z(x, y)$;

   (b) глубину пикселя сравнить со значением, которое хранится в буфере: если $z(x, y) > z_{buf}(x, y)$, то $z_{buf}(x, y) = z(x, y)$, цвет(x, y) = цвет пикселя;

4) отобразить результат.

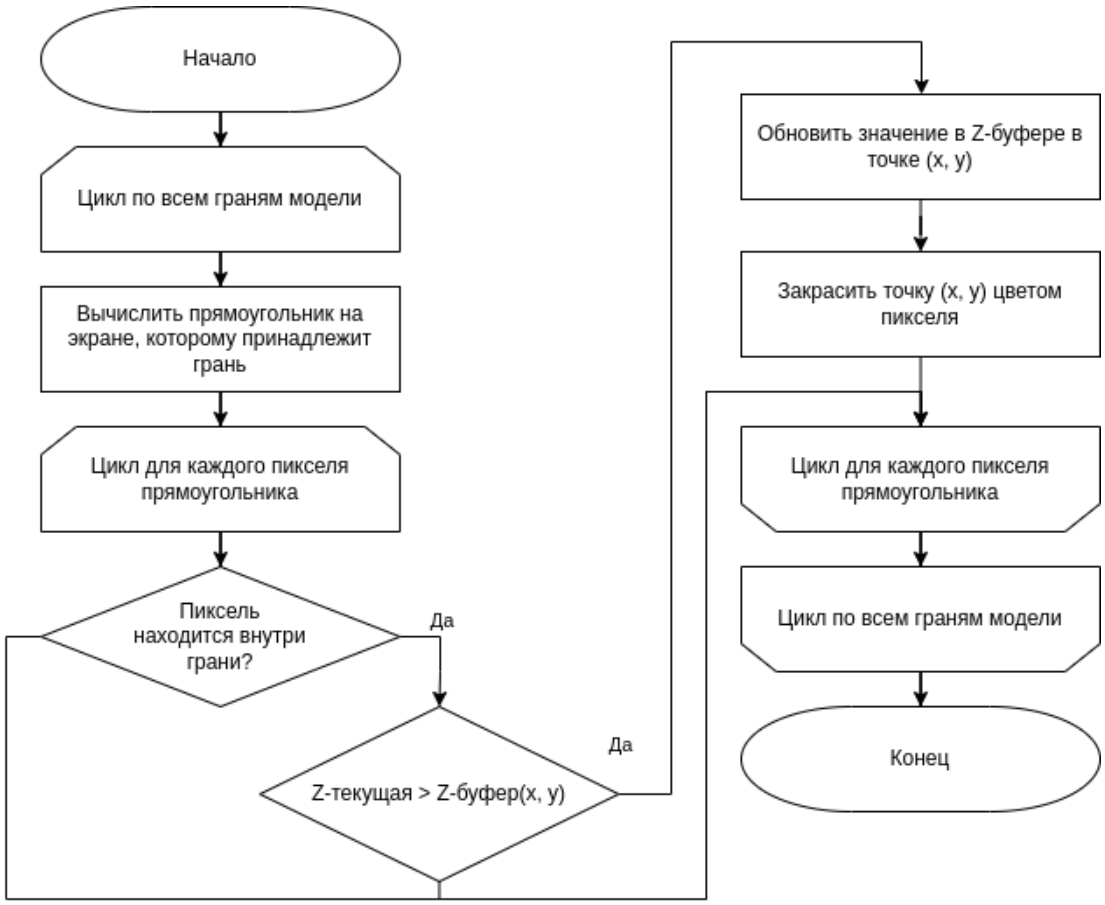

Рисунок 2.11 – Схема алгоритма, использующего Z-буфер

2.4 Определение принадлежности точки треугольнику

В векторном определении принадлежности точки треугольнику методе треугольник и точка являются отображением грани и выбранной точки на область отрисовки.

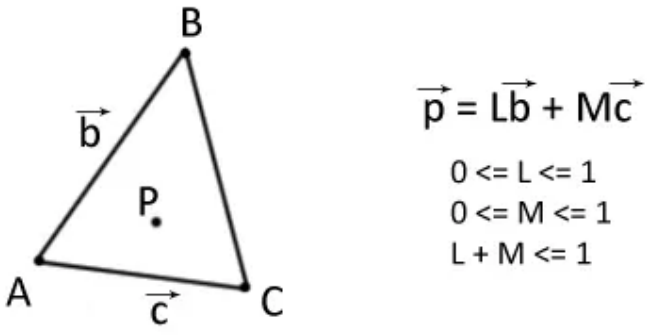

Рисунок 2.12 – Определение принадлежности точки треугольнику

На рисунке 2.12 представлен принцип проверки принадлежности точки треугольнику: вектор $\overrightarrow{AP}$ должен выражаться аффинными координатами $(L; M)$, основанными на векторах $\overrightarrow{AB}$ и $\overrightarrow{AC}$. Положительность координат L и M гарантирует нахождение между сторонами AB и AC соответственно. То, что обе координаты меньше 1 и их сумма также меньше 1 гарантирует, что вектор $\overrightarrow{AP}$ не пересекает сторону BC.

Следовательно, определить, принадлежит ли точка треугольнику можно с помощью вычисления коэффициентов m и l:

$$bx = B_x - A_x, by = B_y - A_y,$$
$$cx = C_x - A_x, cy = C_y - A_y,$$
$$px = P_x - A_x, py = P_y - A_y,$$

$$m = (px \cdot by - bx \cdot py)/(cx \cdot by - bx \cdot cy),$$
$$\textbf{Если } m > 0, \textbf{то:}$$
$$l = (px - m \cdot cx)/bx.$$

В таком случае, если и m и l положительны и в сумме дают не больше 1, то рассматриваемая точка находится внутри треугольника.

2.5 Геометрические преобразования

Для осуществления поворота и перемещения вершин используются аффинные преобразования.

Перемещение точки в пространстве:

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

Поворот точки в пространстве:

$$\text{ось } x: \begin{cases} X = x, \\ Y = y \cdot \cos(\phi) + z \cdot \sin(\phi), \\ Z = -y \cdot \sin(\phi) + z \cdot \cos(\phi). \end{cases} \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) & 0 \\ 0 & -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{ось } y: \begin{cases} X = x \cdot \cos(\phi) + z \cdot \sin(\phi), \\ Y = y, \\ Z = -x \cdot \sin(\phi) + z \cdot \cos(\phi). \end{cases} \quad \begin{pmatrix} \cos(\phi) & 0 & \sin(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{ось } z: \begin{cases} X = x \cdot \cos(\phi) + y \cdot \sin(\phi), \\ Y = -x \cdot \sin(\phi) + y \cdot \cos(\phi), \\ Z = z. \end{cases} \quad \begin{pmatrix} \cos(\phi) & \sin(\phi) & 0 & 0 \\ -\sin(\phi) & \cos(\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Масштабирование точки в пространстве:

$$\begin{cases} X = kx \cdot x, \\ Y = ky \cdot y, \\ Z = kz \cdot z. \end{cases} \quad \begin{pmatrix} kx & 0 & 0 & 0 \\ 0 & ky & 0 & 0 \\ 0 & 0 & kz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2.6 Модель освещения

Для достижения наибольшей производительности в локальная модель освещения делится на две составляющие интенсивности, а именно фоновую освещенность I_{amb} и диффузное отражение I_{diff}. Фоновое освещение это постоянная в каждой точке величина надбавки к освещению. Вычисляется следующим образом:

$$I_{amb} = k_a \cdot I_a,$$

где I_{amb} — интенсивность отраженного Ambient-освещения, k_a — коэффициент в пределах от 0 до 1, характеризующий отражающие свойства поверхности для Ambient-освещения, I_a — исходная интенсивность Ambien-освещения, которое падает на поверхность.

Диффузное отражение это свет, отраженный гранью модели:

$$I_{diff} = I_d \cdot k_{diff} \cdot \cos(\theta),$$

где I_d — интенсивность падающего на поверхность света, k_{diff} — коэффициент в пределах от 0 до 1, характеризующий рассеивающие свойства поверхности, $\cos(\theta)$ — угол между направлением на источника света и нормалью поверхности.

Конечная формула принимает вид:

$$I = I_{amb} + I_{diff} = k_a \cdot I_a + I_d \cdot k_{diff} \cdot \cos(\theta),$$

где для более красивого и приятного изображения $k_a = 1$, $k_{diff} = 0.8$.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к программе

Программа должна предоставлять следующие возможности:

- визуальное отображение модели;
- изменение положения модели в пространстве;
- создание полигонов как части модели;
- удаление полигонов модели;
- изменение формы модели;
- сохранение модели в файл;
- чтение модели из файла;
- изменение цвета грани модели;

3.2 Язык программирования и среда разработки

Для реализации проекта в качестве языка программирования был выбран Kotlin [7] — статически типизированный, объектно-ориентированный язык программирования, работающий поверх Java Virtual Machine и разрабатываемый компанией JetBrains.

Создатели Kotlin изначально делали ставку на компактность: чем меньше ненужного кода, тем проще работать программисту и тем быстрее идёт разработка.

Таким образом Kotlin сочетает в себе преимущества годами увеличивающегося списка готовых решений для Java и лаконичности, безопасности и быстродействия современных языков программирования, к которым и принадлежит Kotlin.

В качестве среды разработки была выбрана среда IntelliJ IDEA[8], запуск происходил через средство разработки.

3.3 Структура программы

Так как разработка велась на структурно-ориентированном языке программирования, особое внимание уделено структуре классов.

- Математические абстракции.
  - Vector3 — четырехмерный вектор (x, y, z, w).
  - Matrix — матрица 4×4.
- Объекты.
  - SimpleObject — абстрактный класс объекта сцены.
  - Vertex — вершина, задается вектор Vector3 и матрицей поворота Matrix.
  - Edge — ребро, заданное двумя вершинами Vertex.
  - Facet — грань, заданная тремя вершинами Vertex и тремя ребрами Edge.
  - PartsOfModel — контейнер, содержащий все грани Facet, ребра Edge и вершины Vertex модели.
  - Model — модель, хранилище PartsOfModel.
- Классы сцены.
  - Scene — класс, хранящий модель.
  - Render — класс для создания изображений.
- Интерфейс пользователя.

– MainVeiw — окно интерфейса программы.

– InfoView — окно для отображения координат вершины, выбранной пользователем.

– WarningView — окно для отображения сообщений об ошибке.

– NewFileView — окно для сохранения модели в файл.

– Styles — класс управления стилем программы.

– MyApp — файл запуска приложения.

3.4 Реализация алгоритма, использующего Z-буфер

В листинге 3.1 представлена реализация алгоритма отрисовки грани, использующего Z-буфер.

Листинг 3.1 – Реализация алгоритма отрисовки грани использующего Z-буфер

```
1     private fun renderFacet (facet: Facet, scene: Scene,
          screenCenter: Vector2D, visible: Boolean) {
2         val z = facet.getNormal(scene.camera, screenCenter).z
3         if (z < 0) return
4         val screenFacets = mutableListOf<Vector3>()
5         for (v in facet.dots) {
6             screenFacets += v.getScreenPos(scene.camera,
                  screenCenter)
7         }
8         val frameRect = calculateFrameRect(screenFacets)
9         var faceColor = calculateFacetColor(facet, scene.camera,
              screenCenter)
10        if (facet.selected) {
11            if (facet.color == Color.BLUE)
12            faceColor = Color.ORANGE
13            else
14            faceColor = Color.BLUE
15        }
16        processFacet(screenFacets, frameRect, faceColor, visible)
17    }
```

```
18
19      private fun processFacet(facets: MutableList<Vector3>, rect:
            IntArray, color: Color, visible: Boolean) {
20          val square = (facets[0].y - facets[2].y) * (facets[1].x -
                facets[2].x) +
21          (facets[1].y - facets[2].y) * (facets[2].x - facets[0].x)
22          for (y in rect[1] .. rect[3]) {
23              for (x in rect[0]..rect[2]) {
24                  val barCoords = calcBarycentric(x, y, facets,
                        square)
25                  if (barCoords.x >= 0 && barCoords.y >= 0 &&
                        barCoords.z >= 0) {
26                      val z = baryCentricInterpol(facets[0],
                            facets[1], facets[2], barCoords)
27                      processPixel(Vector3(x.toDouble(),
                            y.toDouble(), z), color, visible)
28                  }
29              }
30          }
31      }
32
33      fun processPixel (p: Vector3, c: Color, visible: Boolean) {
34          val x = p.x.roundToInt()
35          val y = p.y.roundToInt()
36          if (x <= 0 || x >= width - 1)
37          return
38          if (y <= 1 || y >= height - 1)
39          return
40          if (p.z >= zBuffer[x][y]) {
41              zBuffer[x][y] = p.z
42              if (visible)
43              pw.setColor(x, height - y, c)
44          }
45      }
```

3.5 Интерфейс

На рисунке 3.1 представлен интерфейс программы.

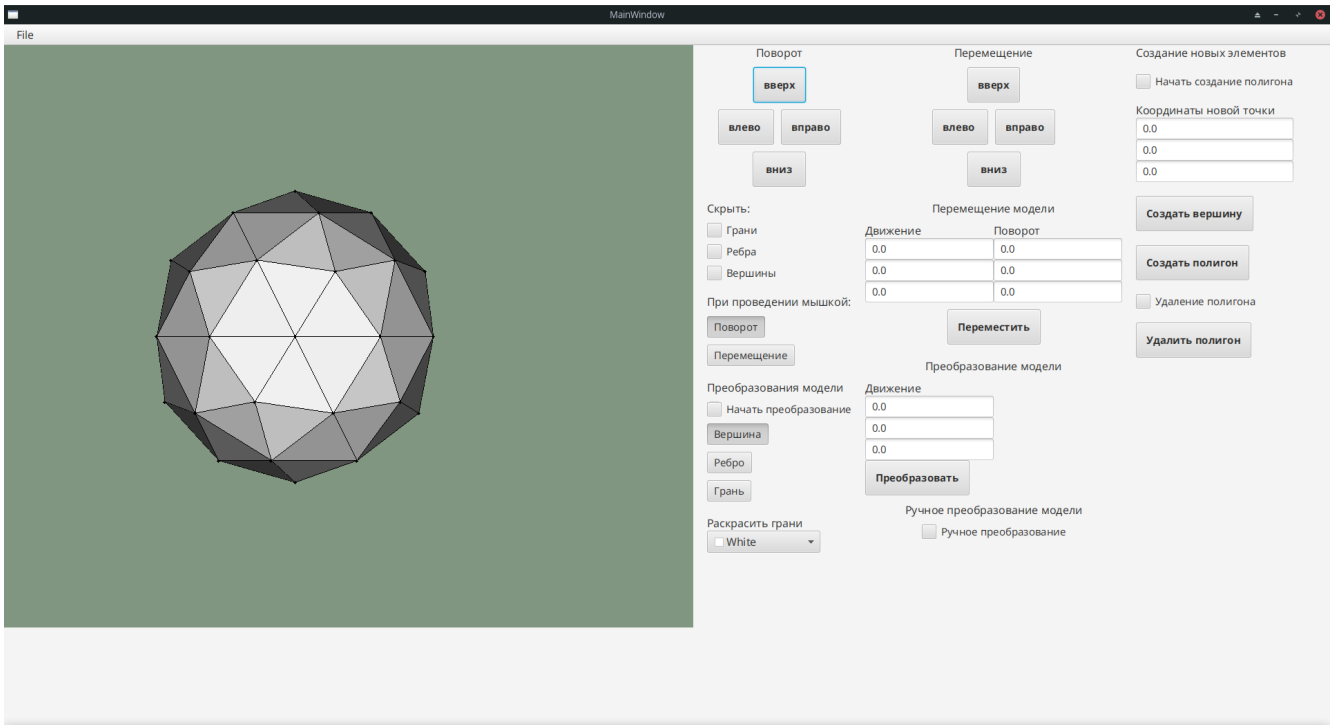

Рисунок 3.1 – Интерфейс программы

Функции представленных полей следующие:

- кнопки «Поворот» — изменение угла поворота модели:
  - кнопка «Вверх» — поворот в плоскости $y0z$ на $10°$;
  - кнопка «Вниз» — поворот в плоскости $y0z$ на $-10°$;
  - кнопка «Влево» — поворот в плоскости $x0z$ на $10°$;
  - кнопка «Вправо» — поворот в плоскости $x0z$ на $-10°$;
- кнопки «Перемещение» — изменение положения модели:
  - кнопка «Вверх» — перемещение по y на 10 пикселей;
  - кнопка «Вниз» — перемещение по y на -10 пикселей;
  - кнопка «Влево» — перемещение по x на -10 пикселей;
  - кнопка «Вправо» — перемещение по x на 10 пикселей;
- тоггл-кнопки «Скрыть»:
  - проверочная кнопка «Грани» — перестают отображаться грани модели;

  - проверочная кнопка «Ребра» — перестают отображаться ребра модели;
  - проверочная кнопка «Вершины» — перестают отображаться вершины модели;

- тоггл-кнопки «При проведении мышкой»:
  - кнопка «Поворот» — при проведении мышкой модель поворачивается;
  - кнопка «Перемещение» — при проведении мышкой модель перемещается;

- «Преобразование модели»:
  - проверочная кнопка «Начать преобразование» — включает режим преобразования модели;
  - тоггл-кнопка «Вершина» — становится возможным выбрать вершину модели;
  - тоггл-кнопка «Ребро» — становится возможным выбрать ребро модели;
  - тоггл-кнопка «Грань» — становится возможным выбрать грань модели;

- всплывающая палитра «Раскрасить грани» — выбор цвета указанной грани;

- «Перемещение модели»:
  - «Движение» — три поля ввода для перемещения всей модели по координатам (x, y, z) соответственно;
  - «Поворот» — три поля ввода для поворота всей модели по координатам (x, y, z) в градусах соответственно;
  - кнопка «Переместить» — изменение положения модели по указанным параметрам;

- «Преобразование модели»:

  - «Движение» — три поля ввода для перемещения выбранных вершин/ребер/граней модели по координатам (x, y, z) соответственно;
  - кнопка «Переместить» — изменение положения выбранных вершин/ребер/граней модели по указанным параметрам;

- проверочная кнопка «Ручное преобразование» — перемещение выбранных вершин/ребер/граней модели с помощью мышки;

- «Создание новых элементов»:
  - проверочная кнопка «Начать создание полигона» — включает режим создания полигона: становится возможным выбирать вершины для создания полигона;
  - три поля ввода «Координаты новой точки» — (x, y, z) новой точки соответственно;
  - кнопка «Создать вершину» — создает вершину по указанным координатам, они автоматически выбраны для создания полигона;
  - кнопка «Создать полигон» — создает полигон по трем выбранным для создания полигона вершинам;

- кнопка «Удалить полигон» — удаляет выбранный полигон;

Вывод

В этот разделе был выбран язык программирования и среда разработки, рассмотрены роли основных классов, подробно рассмотрен интерфейс приложения.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система Manjaro xfce [9] Linux [10] x86_64;
- память 8 ГБ;
- мобильный процессор AMD Ryzen™ 7 3700U @ 2.3 ГГц [11].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Влияние площади, занимаемой моделью на эране, на время отрисовки

В алгоритме, который использует Z-буфер, происходит попиксельный анализ. В таблице 4.1 продемонстрировано пользовательское время программы при разной площади модели.

Таблица 4.1 – Пользовательское время работы программы при разной площади, занимаемой моделью

Площадь модели в пикселях	Время в наносекундах
50×50	945657
100×100	2044827
150×150	4729681
200×200	6306256
250×250	11384766
300×300	17261890
350×350	23556005
400×400	30835890

На рисунке 4.1 представлено время работы алгоритма, использующего z-буфер.

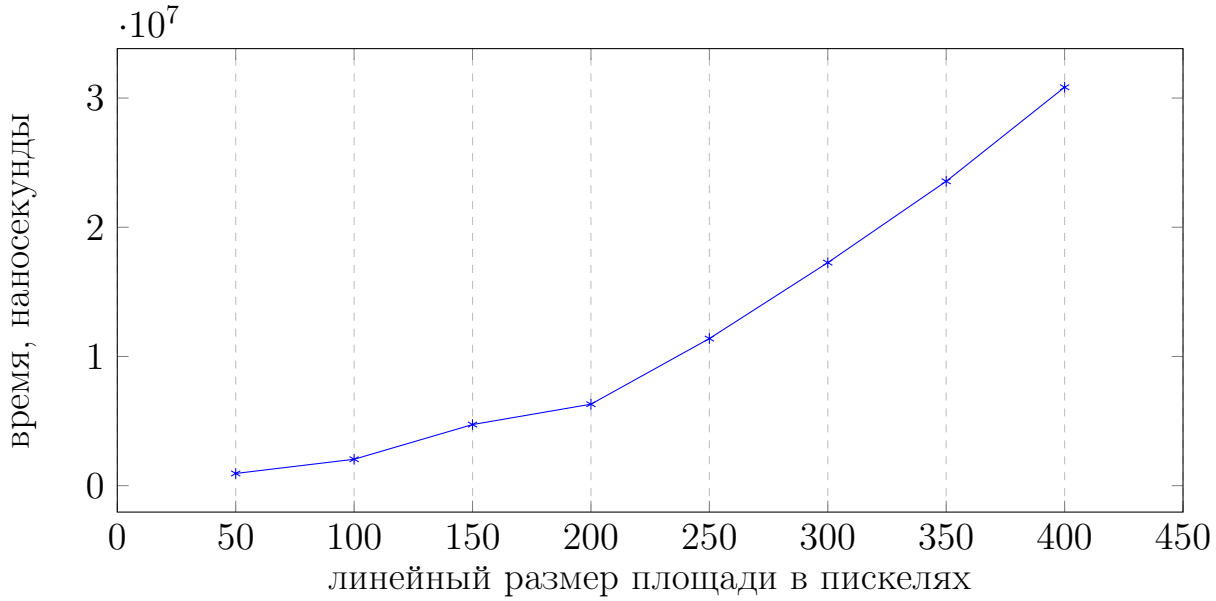

Рисунок 4.1 – Время работы алгоритма, использующего z-буфер

Вывод

В результате было выявлено, что время работы реализации алгоритма растет линейно относительно площади на экране отрисовки, занимаемой моделью — квадратично относительно сторон площади модели.

Заключение

В рамках данного курсового проекта были:

- проанализированы существующие представления трехмерных объектов и обосновать выбор тех из них, которые в наибольшей степени подходят для решения поставленной задачи;
- проанализированы существующие алгоритмы построения изображения и обосновать выбор тех из них, которые в наибольшей степени подходят для решения поставленной задачи;
- реализованы выбранные алгоритмы;
- реализованы возможность изменения положения модели в пространстве;
- реализованы возможность создания полигонов как части модели;
- реализованы возможность удаления полигонов модели;
- реализованы возможность изменения формы модели;
- реализованы возможность сохранения модели в файл;
- реализованы возможность чтения модели из файла;
- реализованы возможность изменение цвета грани модели;

Цель курсового проекта достигнута.

Список использованных источников

1. Трассировка лучей из книги Джефа Проузиса [Электронный ресурс]. Режим доступа: https://www.graphicon.ru/oldgr/courses/cg99/notes/lect12/prouzis/raytrace.htm (дата обращения: 28.10.2021).

2. Алгоритм Варнока. [Электронный ресурс]. Режим доступа: http://compgraph.tpu.ru/Oglavlenie.htm (дата обращения: 09.10.2021).

3. Алгоритм Робертса. [Электронный ресурс]. Режим доступа: http://compgraph.tpu.ru/Oglavlenie.htm (дата обращения: 09.10.2021).

4. Алгоритм Z-буфера. [Электронный ресурс]. Режим доступа: http://compgraph.tpu.ru/Oglavlenie.htm (дата обращения: 09.10.2021).

5. Модели освещения. [Электронный ресурс]. Режим доступа: https://devburn.ru/2015/09/ (дата обращения: 09.10.2021).

6. Закраска, освещение и удаление невидимых поверхностей [Электронный ресурс]. Режим доступа: https://www.graphicon.ru/oldgr/courses/cg99/notes/lect12/prouzis/shading.htm (дата обращения: 27.10.2021).

7. Kotlin Документация [Электронный ресурс]. Режим доступа: https://kotlinlang.org/docs/home.html (дата обращения: 24.09.2022).

8. IntelliJ IDEA Документация[Электронный ресурс]. Режим доступа: https://www.jetbrains.com/idea/ (дата обращения: 24.09.2022).

9. Manjaro [Электронный ресурс]. Режим доступа: https://manjaro.org/ (дата обращения: 03.10.2022).

10. Linux – Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Linux (дата обращения: 04.10.2022).

11. Мобильный процессор AMD Ryzen™ 7 3700U [Электронный ресурс]. Режим доступа: https://www.amd.com/ru/products/apu/amd-ryzen-7-3700u (дата обращения: 04.10.2022).