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:

Разработка универсальной программы, которая позволит редактировать трехмерные модели на уровне вершин, ребер, полигонов(граней)

Студент ИУ7-53Б _____ **Калашников С. Д.**
(Группа) (Подпись, дата) (И.О.Фамилия)

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

Консультант _____ **Павельев А. А.**
(Подпись, дата) (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)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине _____Компьютерная графика_____

Студент группы _ИУ7-53Б__

_____Калашников Сергей Дмитриевич_____
(Фамилия, имя, отчество)

Тема курсовой работы: Редактор трехмерных поверхностей

Направленность КР (учебная, исследовательская, практическая, производственная, др.)
_____учебная _____
Источник тематики (кафедра, предприятие, НИР) _____кафедра__

График выполнения работы: 25% к _4_ нед., 50% к _7_ нед., 75% к 11 нед., 100% к 14 нед.

Задание Разработать универсальную программу, которая позволит редактировать трехмерные модели на уровне вершин, ребер, полигонов(граней). Каждый элемент модели может подвергаться изменениям переноса, масштабирования и поворота. Должна присутствовать возможность выбора изменяемого элемента при помощи мыши и/или выпадающего списка. Изменяемых элементов может быть несколько или один. Так же изменять можно и модель целиком меняя ее масштаб и положение, добавляя или удаляя вершины, грани и ребра. Предусмотреть возможность визуализации модели как с отображением невидимых граней и ребер, так и без них. Обеспечить просмотр модели с любого положения наблюдателя без изменения положения модели. Предусмотреть точечный источник света. Реализовать интерфейс, который позволит задавать исходные данные в виде файла, содержащего в себе описание модели и ее положение на сцене. Реализовать интерфейс, который позволит сохранять описание модели и ее положение на сцене в файл.

Оформление курсовой работы:
Расчетно-пояснительная записка на 25-30 листах формата А4.

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

Перечень графического материала (плакаты, схемы, чертежи и т.п.)_____

На защиту проекта должна быть представлена презентация, состоящая из 15-20 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 _____ _____Павельев А.А.
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____Калашников С.Д.
(Подпись, дата) (И.О.Фамилия)

СОДЕРЖАНИЕ

Введение

Целью курсовой работы является разработка универсального программного обеспечения, которе позволит:

1) загружать параметры трехмерной модель из файла;
2) редактировать трехмерные модели на уровне вершин, ребер, полигонов;
3) просматривать трехмерную модель как с отсечением невидимых ребер и граней, так и без отсечения с любой позиции наблюдателя.

Для достижения поставленной цели требуется решить ряд задач:

1) провести анализ алгоритмов для удаления невидимых линий, закраски, освещения моделей и выделить наиболее подходящие для будущей программы;
2) реализовать выбранные алгоритмы и структуры данных;
3) разработать программное обеспечение для отображения трехмерной сцены;
4) выполнить исследование на основе разработанной программы.

1 Аналитическая часть

1.1 Описание объектов сцены

Сцена состоит из следующих объектов:

- камера — объект, которым может управлять пользователь, с которого осуществляется наблюдение за сценой;
- точечный источник света — объект, положение которого совпадает с положением камеры, освещает сцену;
- модель — объект, который может быть отображен на сцене и который может быть отредактирован пользователем.

1.2 Анализ и выбор формы задания трехмерных моделей

В компьютерной графике существует три основных способа задания трехмерной модели.

1. Каркасная (проволочная) модель — простейшая форма задания трехмерной модели. В этом случае необходимо хранить только вершины — точки, привязанные к координатной сетке и ребра — линии, связывающие вершины.
2. Поверхностная модель — самый популярный вид трехмерной модели в компьютерной графике. Она состоит из вершин, ребер и полигонов. Однако минусом данного вида является незнание того, на какой стороне располагается текстура.

3. Объемная (твердотельная) модель — данная форма отличается от поверхностной тем, что у нас есть информация о том, где расположен материал, это делается с помощью указания направления внутренней нормали.

Для выполнения задания выбраны поверхностные модели. Полигоны, ребра и вершины будут храниться в хэш-таблицах для уменьшения времени доступа к ним.

1.3 Анализ методов вывода трехмерных объектов на экран

Существует три основных алгоритма отображения объемного тела на плоскости.

1. Трассировка лучей — из точки наблюдения на объекты сцены направляются лучи, с помощью которых определяется цвет пикселя на двумерном экране при попадании луча в модель; при этом луч не прекращает своё распространение, а разделяется на три луча-компонента, каждый из которых вносит свой вклад в цвет пикселя на двумерном экране: отражённый, теневой и преломлённый. Благодаря своим особенностям, метод позволяет получить очень фотореалистичные изображения, однако из-за большой ресурсоёмкости процесс визуализации занимает значительное время.

2. Рэйкастинг — метод похожий на трассировку лучей, различие состоит в том, что при нахождении первого пересечения с моделью лучи прекращают своё распространение. Возможно использование каких-либо очень простых способов добавления оптических эффектов. Эффект перспективы получается естественным образом в случае, когда бросаемые лучи запускаются под углом, зависящим от положения пикселя на экране и максимального угла обзора камеры.

3. Растеризация — метод, состоящий из двух этапов: вершинный шейдер, при котором каждый полигон и грань модели проходят процесс

переспективного преобразования, и самого этапа растеризации, где по известным вершинам (в данном случае уже пикселям на экране) необходимо получить все остальные пиксели модели и их атрибуты (цвет, текстура, глубина).

Будут реализованы два алгоритма: растеризация — это ускорит отрисовку сцены в режиме реального времени, и рэйкастинг — данный алгоритм будет так же использован в выборе элемента щелчком мыщи с тем отличием, что будет рассматриваться не весь экран, а только выбранный пиксель.

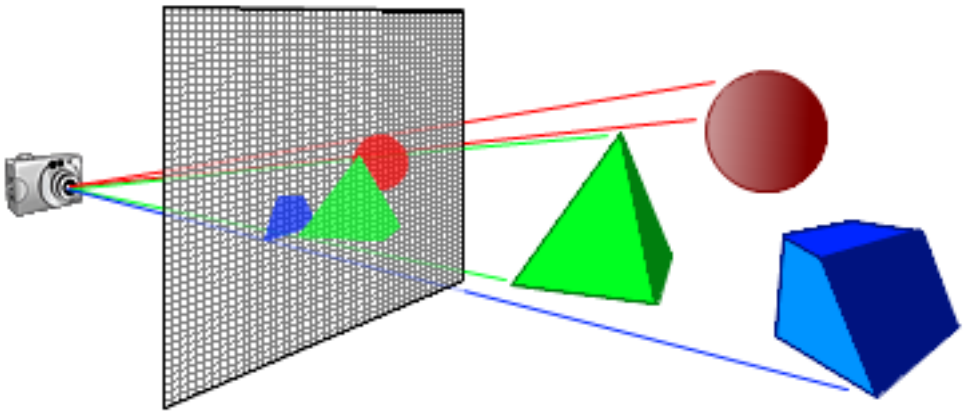

Рис. 1.1 – Процесс работы рэйкастинга

1.4 Описание трёхмерных преобразований

1.4.1 Способ хранения декартовых координат

Координаты можно хранить в форме вектор-столбца x, y, z. Однако в этом случае неудобно применять преобразования поворота, так как такой вектор нельзя умножить на соответствующие квадратные матрицы трансформации размерности четыре на четыре. Целесообразнее использовать вектор-столбцы размерности четыре - [x, y, z, w], где координата $w = 1$. Преобразования координат выполняются умножением слева преобразуемого вектора-строки на соответствующую матрицу линейного оператора.

1.4.2 Матрицы аффинных преобразований декартовых координат

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

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix}. \tag{1.1}$$

2. Масштабирование относительно начала координат с коэффициентами sx, sy, sz:

$$\begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.2}$$

3. Поворот относительно осей x, y, z соответственно на угол α :

$$Rx(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) & 0 \\ 0 & -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{1.3}$$

$$Ry(\alpha) = \begin{pmatrix} \cos(\alpha) & 0 & -\sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{1.4}$$

$$Rz(\alpha) = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.5}$$

1.4.3 Преобразование трехмерного пространства в двумерное пространство экрана

Нижеописанные пункты применимы к методу растеризации.

Алгоритм перевода координат из трехмерного пространства к двумерному, следующий [1]:

1) перевести объект из мирового пространства в пространство камеры;

2) найти все проекции точек из пространства камеры в видимые точки, где координаты точек x, y, z находятся в диапазоне [−w, w], а w находится в диапазоне [0, 1];

3) масштабировать все точки, полученные в пункте 2, на картинку необходимого разрешения.

1.4.4 Преобразования трехмерной сцены в пространство камеры

Для того, чтобы преобразовать сцену в пространство камеры нужно умножить каждую вершину всех полигональных моделей на матрицу камеры. Сама камера задается набором следующих атрибутов: положение центра камеры в мировом пространстве, вектора направления взгляда, направления верха камеры. Матрица камеры представляет собой результат перемножения двух матриц:

$$\begin{pmatrix} Right_x & Right_y & Right_z & 0 \\ Up_x & Up_y & Up_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 & -PX \\ 0 & 1 & 0 & -PY \\ 0 & 0 & 1 & -PZ \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{1.6}$$

где D — вектор ”взгляда”камеры, Up — вектор, который указывает куда направлен верх камеры, Right — ортогональный вектор к векторам направления ”взгляда”и вектору направления верха камеры.

1.4.5 Матрица перспективной проекции

После перехода в пространство камеры следует умножить каждую вершину всех полигональных моделей на матрицу проекции (1.7). Матрица проекции отображает заданный диапазон усеченной пирамиды 1.2 в пространство отсечения, и при этом манипулирует w-компонентой каждой вершины

таким образом, что чем дальше от наблюдателя находится вершина, тем больше становится это w-значение. После преобразования координат в пространство отсечения x, y попадают в диапазон от -w до w, а вершина z от 0 до w (вершины, находящиеся вне этого диапазона, отсекаются). Следующий этап — спроецировать все координаты на одну плоскость, разделив всё на координату z. После умножения вектора координат на матрицу перспективной проекции, реальная координата z заносится в w-компоненту, так что вместо деления на z делят на w.

$$\begin{pmatrix} \frac{\cot\frac{fov}{2}}{aspect} & 0 & 0 & 0 \\ 0 & \cot\frac{fov}{2} & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & 1 \\ 0 & 0 & \frac{-2fn}{f-n} & 0 \end{pmatrix}. \tag{1.7}$$

В (1.7):

- aspect — отношение ширины изображения к его высоте;
- fov — угол обзора камеры;
- f — координата z дальней от камеры плоскости отсечения пирамиды видимости;
- n — координата z ближней к камере плоскости отсечения пирамиды видимости

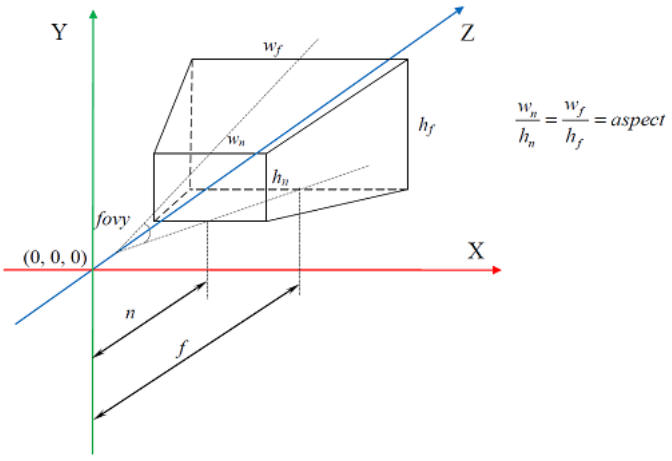

Рис. 1.2 – Усеченная пирамида проекции

1.4.6 Преобразования трехмерной сцены в пространство области изображения

Чтобы преобразовать спроецированные координаты в координаты области изображения, нужно инвертировать ось Y и применить операции переноса и маштабирования к каждой вершине.

1.5 Анализ алгоритма удаления невидимых ребер и поверхностей

1.5.1 Z-буфер

Суть данного алгоритма — это использование Z-буфера, в котором хранятся информация о координате Z для каждого пикселя. Первоначально в Z-буфере находятся максимально возможные значения Z. Каждый многоугольник преобразуется в растровую форму. В процессе подсчета глубины нового пикселя, он сравнивается с тем значением, которое уже лежит в Z-буфере. Если новый пиксель расположен ближе к наблюдателю, чем предыдущий, то онотрисовывается и происходит корректировка Z-буфера. Необходимо отметить, что в данном алгоритме может возникнуть следующая ситуация: если два объекта имеют близкую Z-координату, то они могут перекывать друг друга. Это называется Z-конфликт (рис. 1.3). Решаются Z-конфликты сдвигом одного объекта относительно другого на величину, превышающую погрешность Z-буфера.

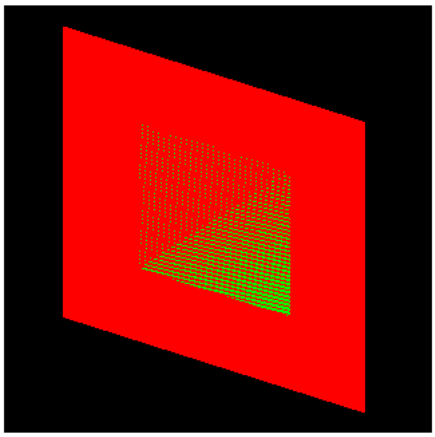

Рис. 1.3 – Z-конфликт

1.6 Алгоритмы растреризации

1.6.1 Интерполяция

Интерполяция [2] — в вычислительной математике нахождение неизвестных промежуточных значений некоторой функции, по имеющемуся дискретному набору её известных значений, определенным способом. Для каждого полигона необходимо провести интерполяцию по двум осям: X и Y. Сначала происходит интерполяция по каждому ребру для нахождения промежуточных значений, затем для каждой сканирующий строки происходит интерполяция по X, что позволяет получить данные о каждой точке внутри полигона. Данный алгоритм позволяет довольно быстро определить цвет, глубину и остальные атрибуты для элемента.

1.6.2 Traversal алгоритм

Барицентрические координаты — это координаты, в которых точка треугольника описывается как линейная комбинация вершин. Чаще всего используют нормализованный вариант — т.е. суммарный вес трех вершин равен единице

Барицентрические координаты равны отношению площадей треугольников, образованных на 1.4, к общей площади треугольника:

$$b_0 = \frac{\delta_0}{\delta}, b_1 = \frac{\delta_1}{\delta}, b_2 = 1 - b_0 - b_1. \tag{1.8}$$

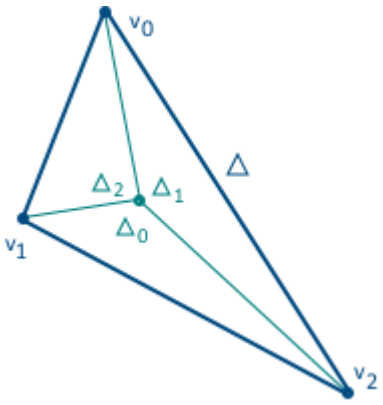

Рис. 1.4 – Площади полигона

Значение атрибута в заданной точке треугольника равно линейной комбинации барицентрических координат и значений атрибута в соответствующих вершинах:

$$T = b_0 * T_0 + b_1 * T_1 + b_2 * T_2 \tag{1.9}$$

Данный алгоритм работает в соответствии с нижележащими пунктами:

- определить наименьший прямоугольник, описывающий полигон;
- для каждого пикселя выбранного прямоугольника определить барицентрицеские координаты
- если значение каждой из них находится в отрезке от 0 до 1, то находим атрибуты рикселя и закрашиваем его.

Вывод

Не смотря на то, что алгоритм интерполяции выполняется быстрее по времени из-за того, что он работает исключительно в пространстве полигона, будут реализованы оба алгоритма по условию задания.

1.7 Модель освещения

В программе будет использована простая модель освещения, суть которой заключается в следующем: вся грань закрашивается одним уровнем интенсивности,который высчитывается по закону Ламберта.

2 Конструкторская часть

2.1 Требования к разрабатываемой программе

Программа должна предоставлять следующий доступ к функционалу:

- перемещения, маштабирования, поворота ребер, вершин, полигонов модели и самой модели целиком;
- добавления, удаления вершин, ребер и полигонов;
- изменения координат вершин;
- выбора различных алгоритмов вывода модели на экран;
- загрузки и сохранению модели в файл;
- свободному перемещению по сцене без перемещения модели;
- возможности прекратить и возобновить отрисовку в любой момент.

Требования к программе:

- программа должна корректно обрабатывать действия пользователя;
- программа должна реагировать на действия пользователя с задержкой не более секунды, либо выводить соответствующее сообщение об обработке запроса.

2.2 Выбор используемых типов и структур данных

Для разрабатываемо программного обеспечения необходимо реализовать следующие типы:

- математические абстракции — точки, линии, вектора, матрицы;
- модель, задаваемая вершинами, полигонами, ребрами;
- камера — наблюдатель;
- интерфейс — позволяет пользователю взаимодействовать со сценой;
- сцена — хранит камеру и модель, реализует обработку отрисовки;
- посетители — для унифицирования разработки программы и упрошения дальнейшего развития и дополнения.

Нужно разработать следующие структуры данных:

- хэш таблицы — для хранения элементов модели;
- списки активных элементов;
- контейнерный класс — для хранения связей между элементами модели.

2.3 Описание алгоритмов

На рисунках 2.1 и 2.2 представлены схемы двух алгоритмов отсечения невидимых ребер — z-буфера и рэйкастинга соответственно.

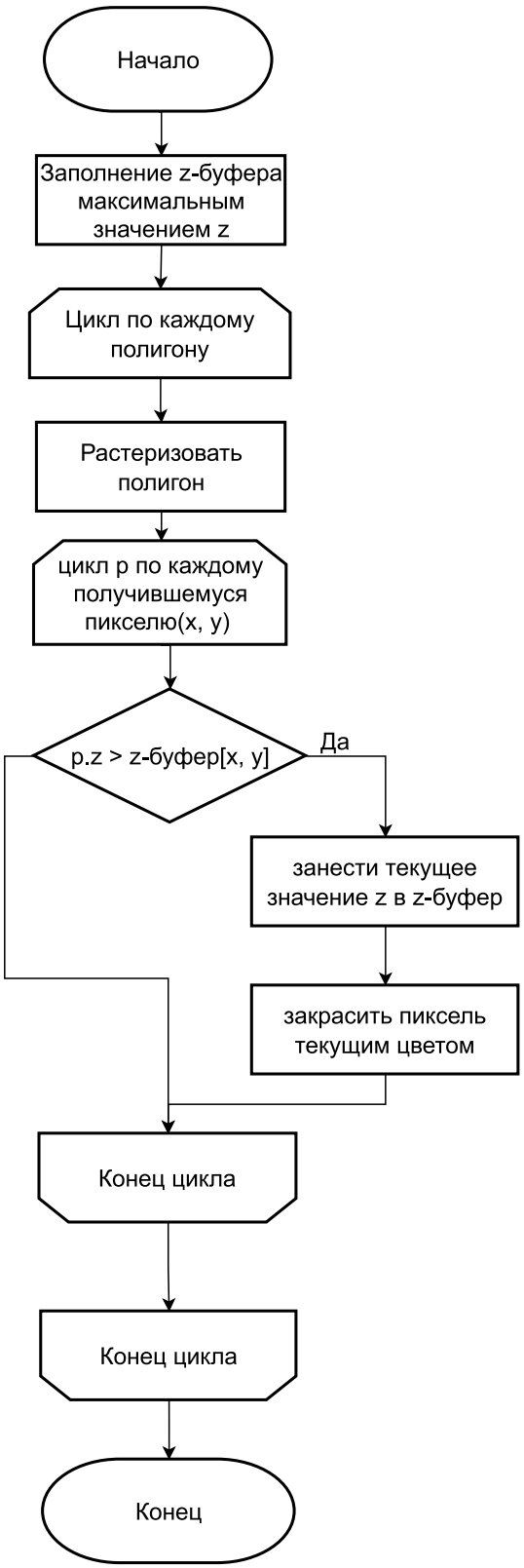

Рис. 2.1 – Схема алгоритма z-буфера

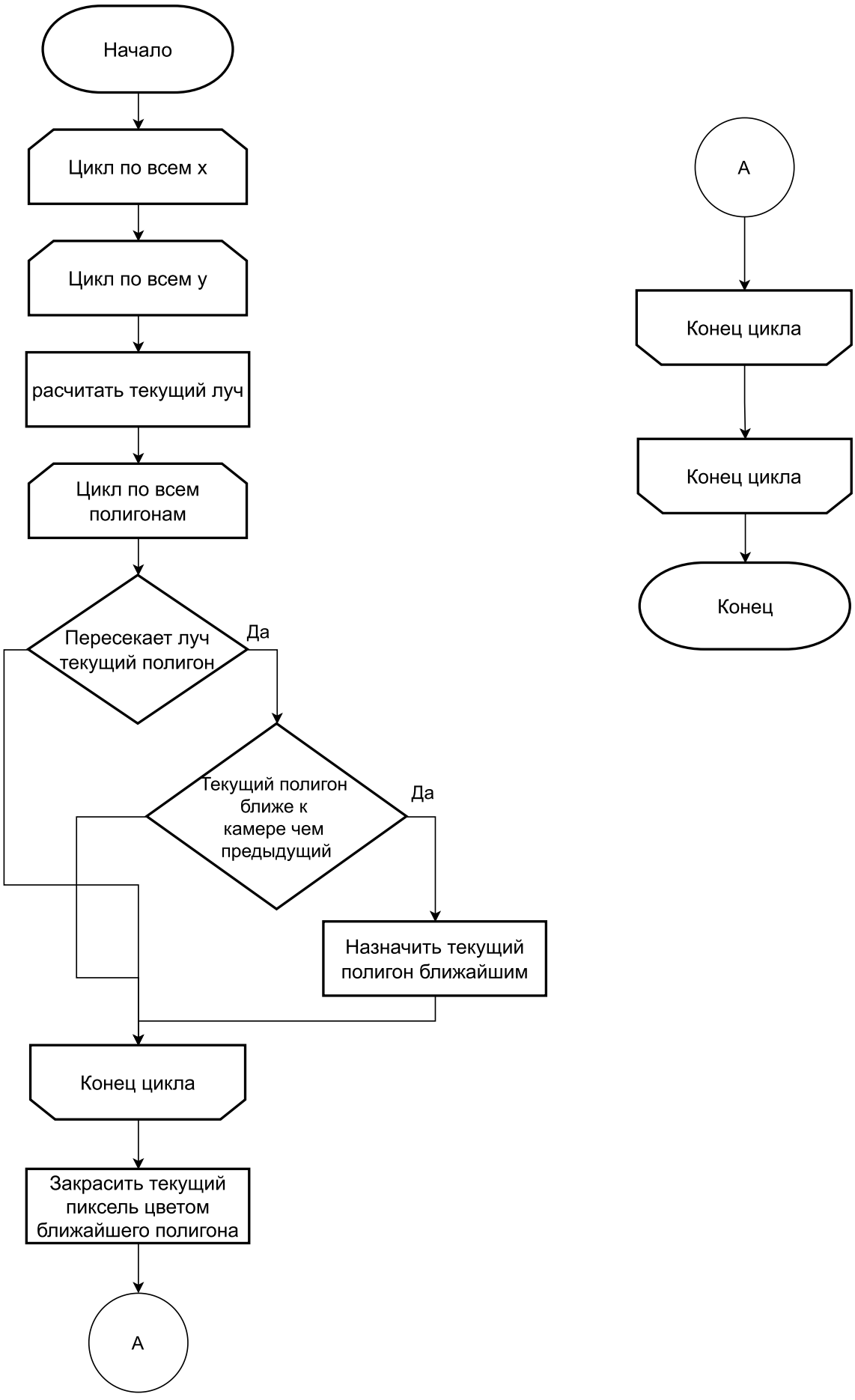

Рис. 2.2 – Схема алгоритма рэйкастинга

2.4 Описание способа хранения элементов

Для корректного взаимодействия элементов необходимо хранить связи между объектами. Для этого создана структура контейнера, котрая выполняет данную работу. Связь между объектами аналогична связи между детьми и родителями так как для каждого из трех типов элементов легко выделить такие отношения:

- для полигона детьми являются три ребра и три вершины;
- у ребра есть родители — полигоны — и потомоки — две вершины;
- для вершины родителями являются полигоны и ребра.

Контейнер включает в себя три хэш таблицы — для хранения элементов, для хранения родителей элемента, для хранения детей элемента. Если у элемента нет родителей или детей, соответствующие таблицы не заполняются.

Удаление частей модели из контейнера происходит по двум принципам:

1) если у элемента не осталось ни одного родителя, то он подлежит удалению;
2) если у элемента пропал хоть один потомок, то он так же подлежит удалению.

При добавлении элемента в контейнер добавляется сам элемент и индетификаторы связанных объектов.

Отдельно стоит отметить, что в контейнере не может храниться одновременно несколько абсолютно одинаковых объектов.

3 Технологическая часть

3.1 Средства реализации

Для решения поставленных в курсовом проекте задач был выбран язык *c#* [3], поскольку:

- этот язык предоставляет программисту широкие возможности реализации самых разнообразных алгоритмов, он обладает высокой эффективностью и большим набором стандартных классов и процедур;
- *c#* является полностью объектно-ориентированным и позволяет использовать наследование, абстрактные и параметризованные классы;
- трехмерные объекты, также как и математические абстракции, естественным образом представляются в виде объектов классов, что позволяет легко и эффективно организовывать их взаимодействие, при этом сохраняется читаемый и легко изменяемый код;
- В *c#* реализована многопоточность, что может быть полезным при реализации алгоритмов, требующих больших затрат по времени.

В качестве среды разработки была выбрана Visual Studio 2022. Некоторые факторы, по которым была выбрана данная среда:

- включает весь основной функционал — параллельная сборка, отладчик, поддержка точек останова, сборки и т.д;
- разработчики имеют возможность расширить любой функционал, включая компиляцию, отладку;

- работает с интерфейсом Windows Forms, который очень удобен в использовании, а также позволяет без проблем создавать приложения.

3.2 Структура классов программы

На рисунке 3.1 представлена структура классов программы.

- MatrixCoord3D — класс, реализующий точку в пространстве.
- MatrixTransformation3D — класс, реализующий матрицы преобразований.
- ModelComponent — класс, предоставляющий интерфейс для реализации компонентов модели.
- ContainerHash — класс, реализующй контейнер, хранящий элементы.
- DrawVisitor — класс, отрисовывающий сцену.
- ReadVisitor — класс, расчитывающий попадание мыши.
- EasyTransformVisitor — класс, реализующий операции над моделью.
- Rasterizator — класс, реалзующий алгоритмы растризации.
- RayTraicing — класс, реалзующий трассировку лучей.
- ModelHash — класс, реалзующий модель.
- Scene — класс, реалзующий сцену.

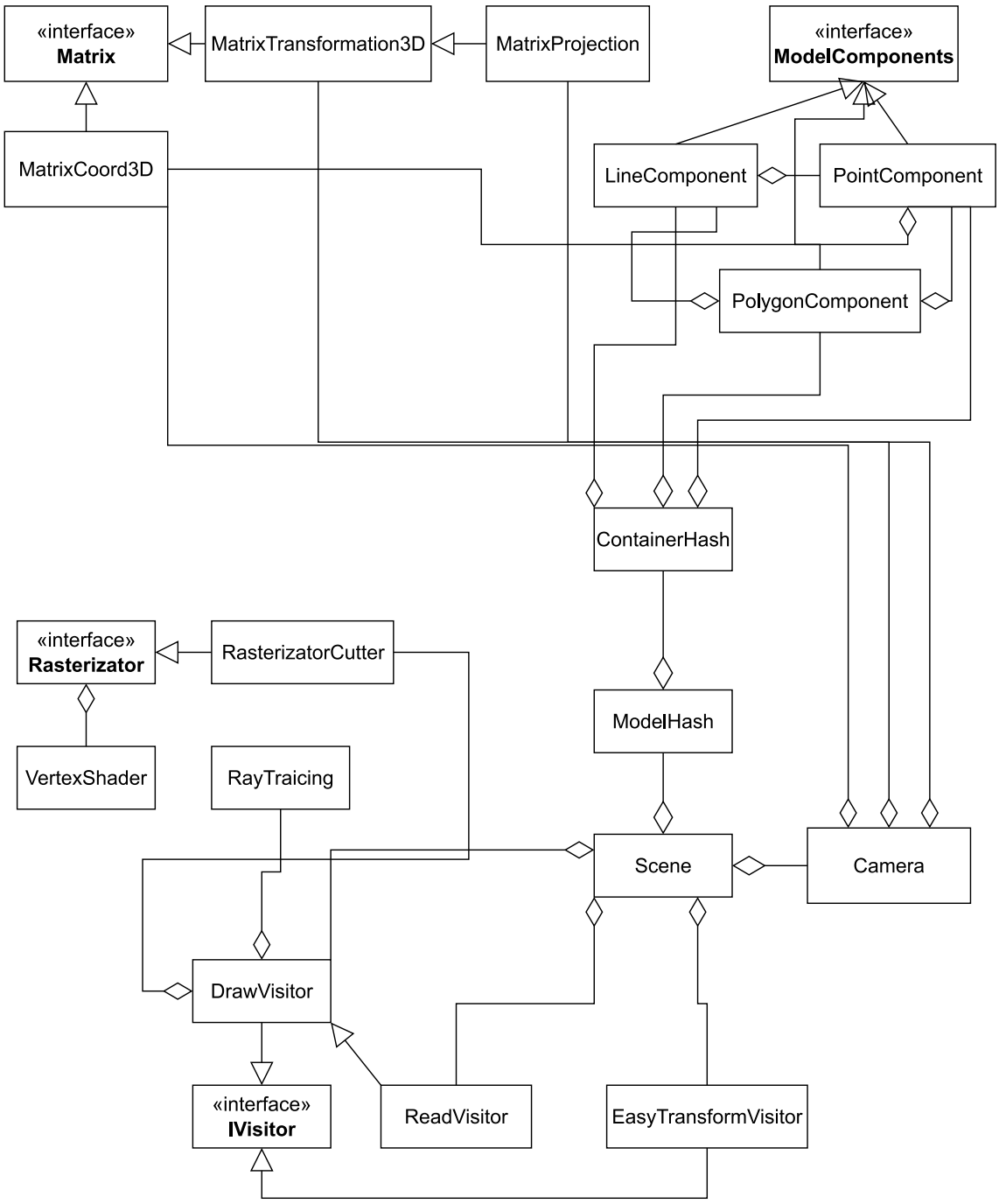

Рис. 3.1 – Схема структуры классов программы

3.3 Реализация алгоритмов

В листингах представлены реализации алгоритмов

Листинг 3.1 — Алгоритм рэйкастинга

```
1 public virtual void RayTrasing(IModel model){
2  Tuple<MatrixCoord3D, double> sphere =FoundCenter(model.Points);
3  MatrixCoord3D CamPosition = cam.Position.Coords;
```

```
4   MatrixTransformation3D RotateMatrix = cam.RotateMatrix.
        InversedMatrix();
5   double aspect = screen.Width / (double)screen.Height;
6   double field = Math.Tan(cam.Fovy / 2 * Math.PI / 180.0f);
7   for (int x = 0; x < screen.Width; x++)
8     for (int y = 0; y < screen.Height; y++)
9     {
10      MatrixCoord3D D = CanvasToVieport(x, y, aspect, field) *
            RotateMatrix;
11      D.Normalise(); Color c = Color.White;
12      if (RaySphereIntersection(CamPosition, D, sphere.Item1,
            sphere.Item2) != double.MaxValue)
13        c = RayT(model, D, CamPosition);
14      PictureBuff.SetPixel(x, y, c.ToArgb());
15    }}
16 protected Color RayT(IModel model, MatrixCoord3D D, MatrixCoord3D
       position){
17  PolygonComponent closest = null;
18  double closest_t = double.MaxValue;
19    foreach (PolygonComponent p in model.Polygons)
20      if(p!=null)
21      {
22        MatrixCoord3D tt = GetTimeAndUvCoord(position, D, p.
              Points[0].Coords, p.Points[1].Coords, p.Points[2].
              Coords);
23        if (tt != null)
24        {
25          if (tt.X < closest_t && tt.X > 1)
26          {
27            closest_t = tt.X; closest = p;
28          }}}
29    if (closest == null)
30      return Color.White;
31    double cos = Math.Abs(MatrixCoord3D.scalar(closest.Normal,
          cam.Direction));
32    Color c = Color.FromArgb(255, Convert.ToInt32(closest.ColorF.
          R * cos), Convert.ToInt32(closest.ColorF.G * cos), Convert
          .ToInt32(closest.ColorF.B * cos));
33    return c;}
```

Листинг 3.2 — Алгоритм Z-буфера

```
public override void DrawPolygon(PolygonComponent polygon){
  MatrixCoord3D p1 = shader.VertexTransform(polygon.Points[0]);
  MatrixCoord3D p2 = shader.VertexTransform(polygon.Points[1]);
  MatrixCoord3D p3 = shader.VertexTransform(polygon.Points[2]);
  double cos = Math.Abs(MatrixCoord3D.scalar(polygon.Normal,
     shader.up.Direction));
  Color c = Color.FromArgb(255, Convert.ToInt32(polygon.ColorF.R
     * cos), Convert.ToInt32(polygon.ColorF.G * cos), Convert.
     ToInt32(polygon.ColorF.B * cos));
  if (p1 != null && p2 != null && p3 != null)
  drawTriangleFill(new List<PointComponent> { new PointComponent(
     p1), new PointComponent(p2), new PointComponent(p3) }, c);}
private void drawTriangleFill(List<PointComponent> vertices,
   Color color){
  var points = new List<PointComponent> { vertices[0], vertices
     [1], vertices[2] };
  foreach (var p in Fill.FillTriangle(points))
  drawPoint(p, color);
}
void drawPoint(PointComponent point, Color color){
  var p2D = point;
  if (zBuffer[point.X, point.Y] <= point.Z)
  return;
  zBuffer[point.X, point.Y] = point.Z;
  if (color != Color.White)
  PictureBuff.SetPixel((int)p2D.X, (int)p2D.Y, color.ToArgb());}
```

3.4 Описание интерфейса программы

На рисунке 3.2 представлен интерфейс разработанного програмного обеспечения. По умолчанию на экран выводится куб, однако нажав на кнопку <файл> можно выбрать модель из файла obj. Интерфейс позволяет выбирать отдельные элементы модели левой кнопкой мыши. Выбранный элемент можно добавить в активные и работать с ним отдельно. Присутствует возможность перетаскивать активные элементы мышью. Добавление полигонов производится в окне справа; там же можно поменять координату выбраной

точки. Правой кнопкой мыши можно добавть новую точку и соответственно линию на модель. Для обзора сцены используется камера, управление которой осуществляется посредством нажатия клавиш на клавиатуре. При нажатии кнопки <выбор трансформирования> откроется окно (рис. 3.3), в котором можно задать необходимые матрицы аффиных преобразований.

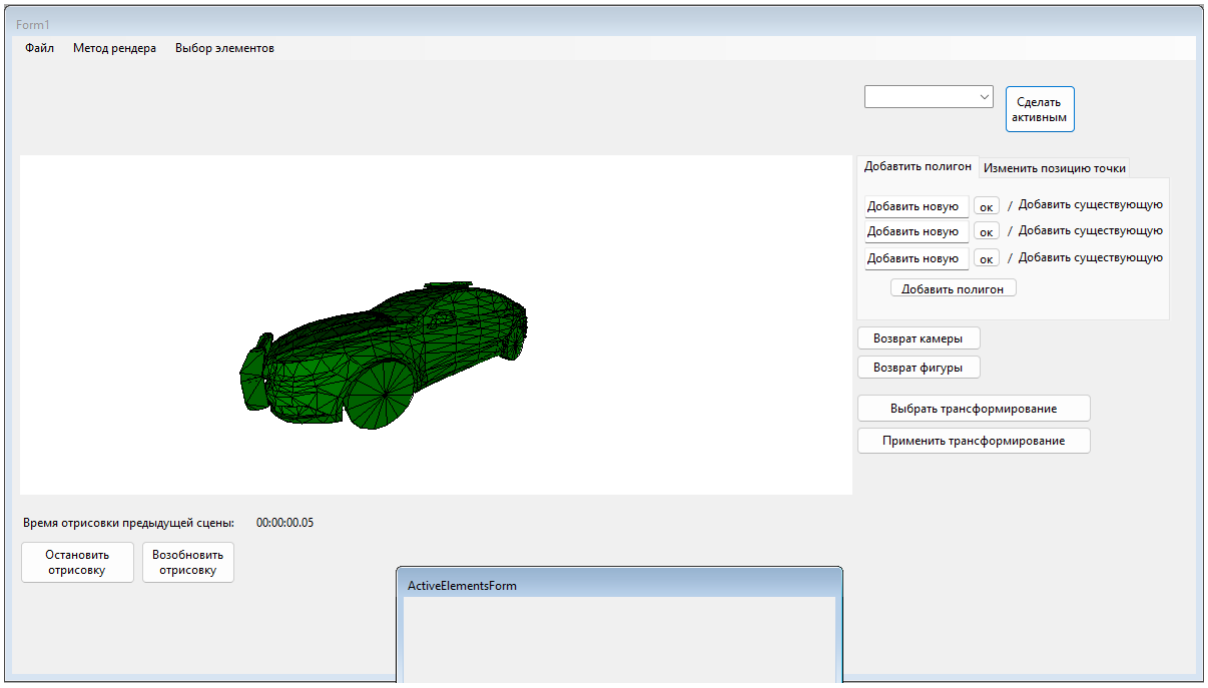

Рис. 3.2 – Интерфейс программы

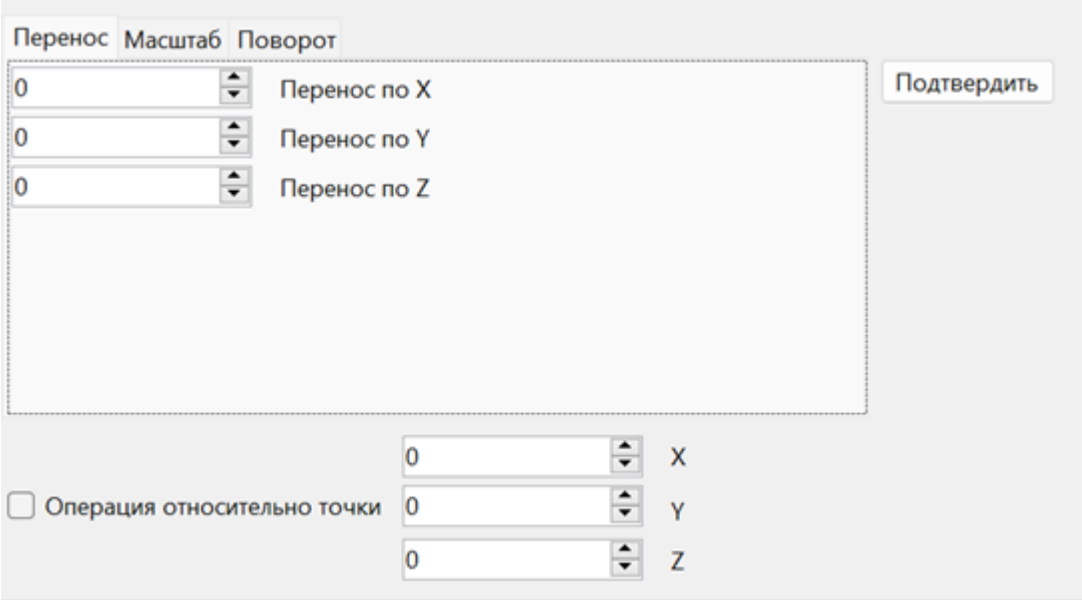

Рис. 3.3 – Интерфейс программы

4 Исследовательская часть

В данном разделе будет проведен сравнительный анализ времени работы реализаций алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени представлены далее:

- операционная система Windows 11 Pro Версия 22H2 (22621.674) [4];
- память 16 ГБ;
- процессор 11th Gen Intel(R) Core(TM) i5-11400 2.59 ГГц [5].

При тестировании компьютер был включен в сеть электропитания. Во время замеров времени выполнения реализаций алгоритмов устройство было нагружено только встроенными приложениями окружения, а также системой тестирования.

4.2 Описание проводимых исследований

Так как в данной работе реализовано несколько алгоритмов вывода трехмерной модели на экран, то можно провести несколько опытов.

1. Одной из положительных сторон рейкастинга является возможность параллельного выполнения данного алгоритма. Разные части изображения могут обрабатываться независимо от других. По этой причине можно обрабатывать части холста одновременно, используя потоки, таким образом уменьшая общее время работы реализации алгоритма. Разделение на потоки будет происходить по слеующему принципу: холст разбивается по оси X на равные интервалы, их количество равно количеству логических ядер. В данном эксперементе будет проведено сравнение параллельной версии алгоритма на количестве потоков равному количеству логических ядер ЭВМ, на которой будут проводится замеры времени, и последовательной версии.

2. Во втором эксперементе будет проведено сравнение времени и качества работы реализаций алгоритмов растеризации. Будут сравниваться следующие алгоритмы: использующий интерполяцию, использующий барицентрические координаты, оптимизированные алгоритм использующий барицентрические координаты. Оптимизиция последнего является изменением способа прохода по пикселям — если в оригинальном алгоритме рассматриваются все пиксели холста, то в текущем они будут рассматриваться змейкой(от границы до границы полигона).

   Все эксперименты будут проведены на кубе, имеющим 12 полигонов.

4.3 Использованные средства для замеров времени

Время работы было замерено с помощью класса *Stopwatch* из библиотеки $System.Diagnostics$ [6]. Перед началом выполнения алгоритма включается таймер методом $Start()$, после выполнения таймер останавливается методом $Stop()$. Для получения времени работы реализаций алгоритмов используется свойство $Elapsed$ того же класса. Возвращаемым значение является структура типа $TimeSpan$, содержащая время работы таймера в часах, минутах, секундах, миллисекундах.

4.4 Время выполнения реализаций алгоритмов

На рисунке 4.1, приведены графические результаты замеров времени работы алгоритма рейкастинга для параллельного (при количестве потоков, равном количеству логических ядер ЭВМ, на которой проводились замеры времени, затрачиваемого реализацией трассировки лучей) и последовательного случаев при разной доле заполнения экрана. На рисунке 4.2, приведены графические результаты замеров времени работы алгоритмов растеризации при разной доле заполнения экрана.

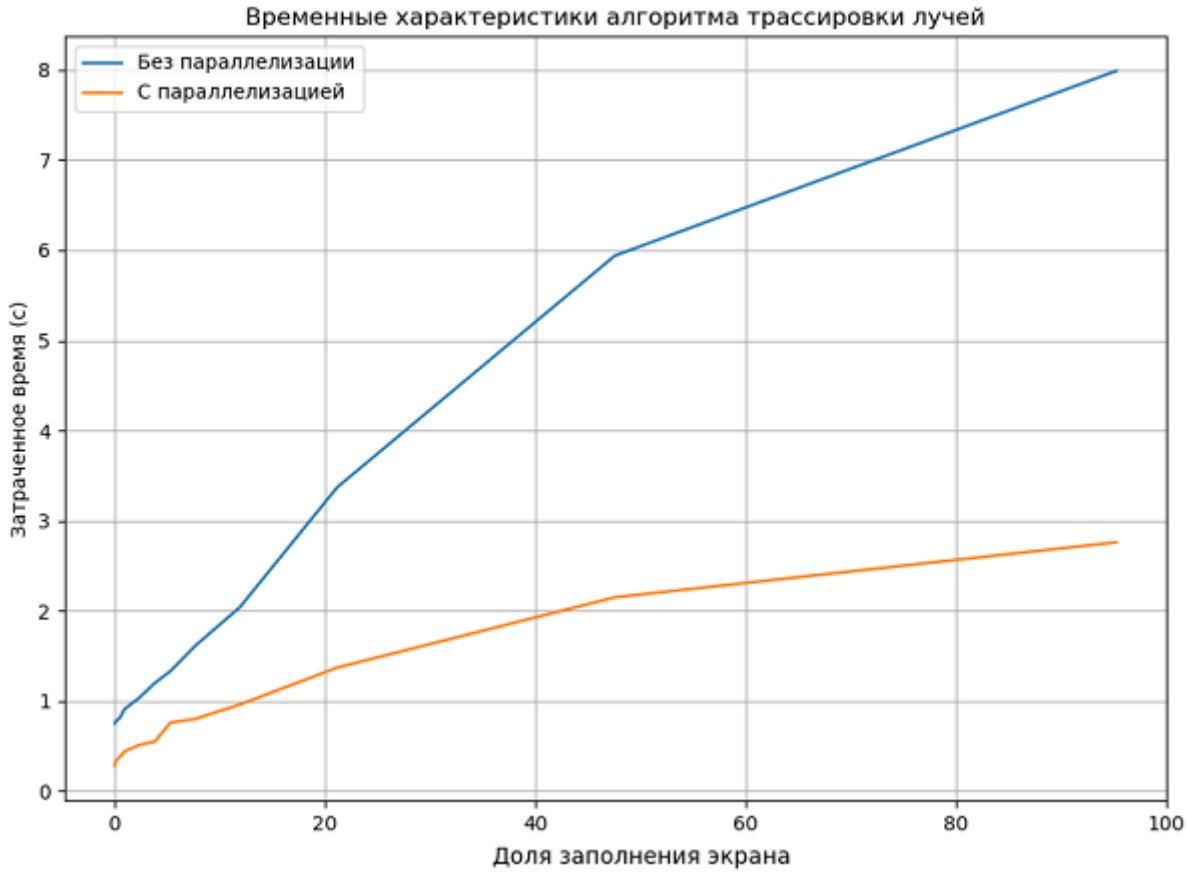

Рис. 4.1 – Время работы реализаций алгоритма

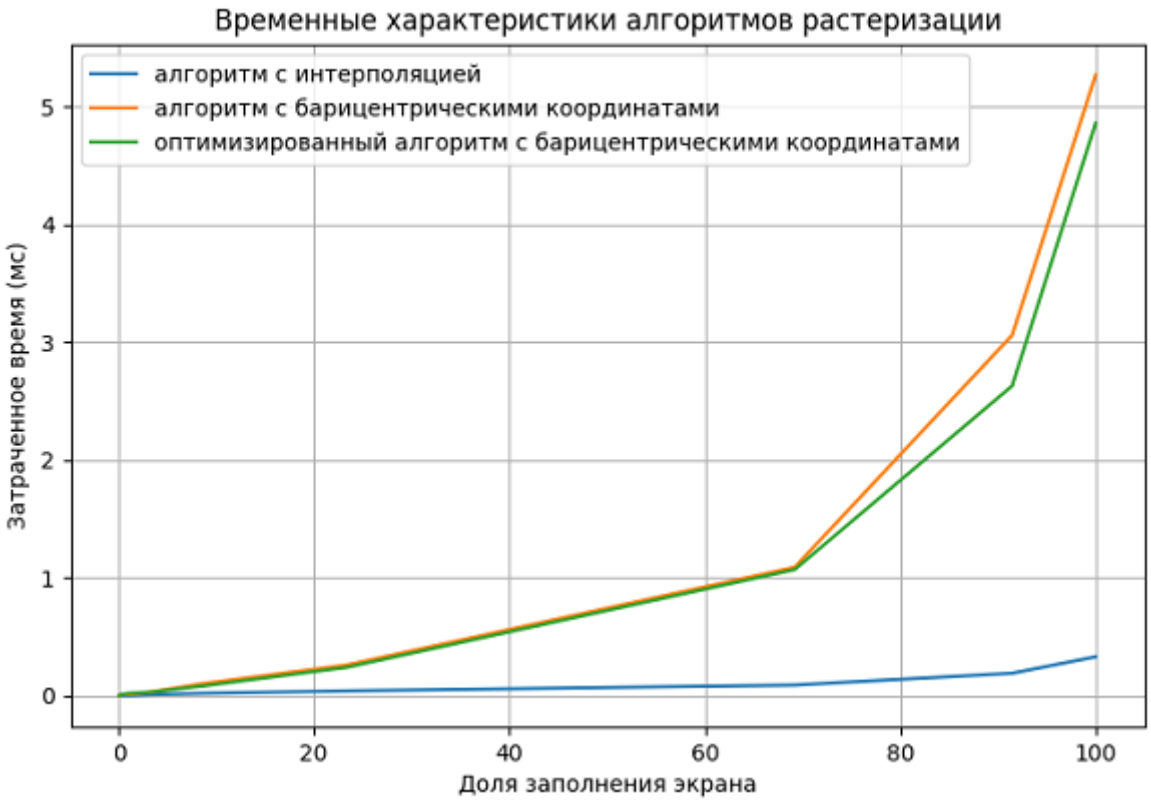

Рис. 4.2 – Время работы реализаций алгоритма

4.5 Примеры работы программы

На рисунках 4.3 и 4.4 приведены прменры работы программы для алгоритмов растеризации использующих интреполяцию и барицентрические координаты соответственно. На экран выводиться лицевая сторона куба.

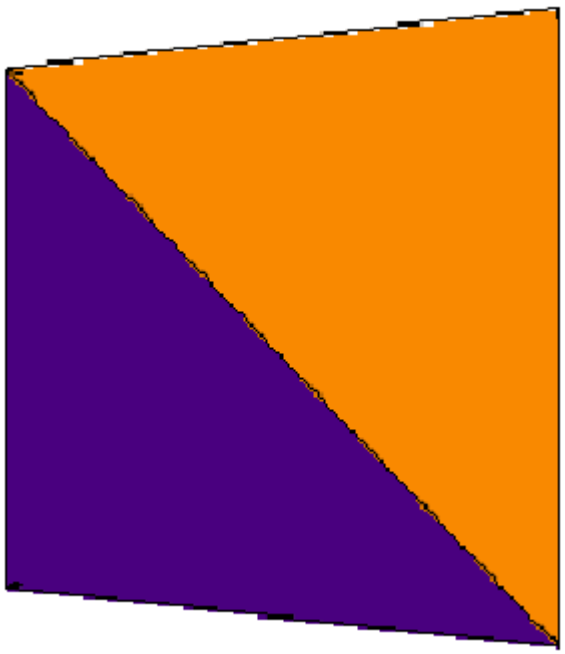

Рис. 4.3 – Пример работы программы

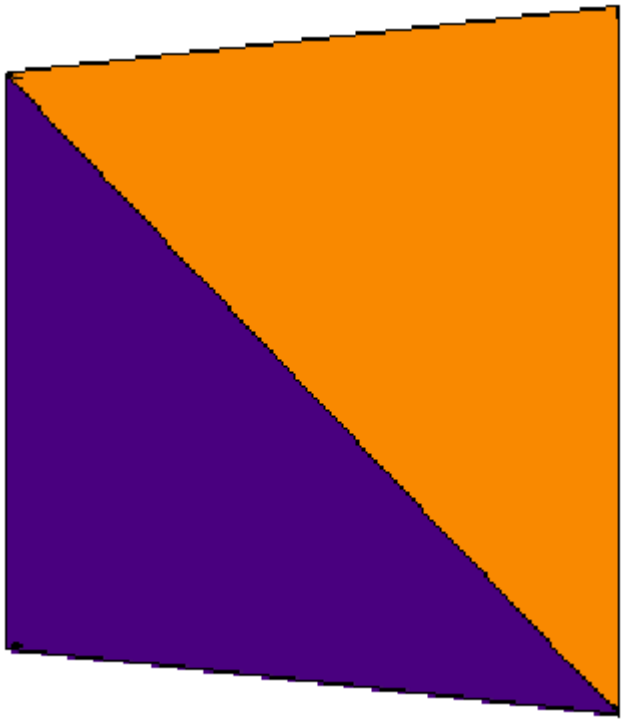

Рис. 4.4 – Пример работы программы

Вывод

Исходя из полученных результатов, трассировка лучей с параллелизацией оказалась быстрее в 2.6 раза на большем заполнении экрана и в 2 раза на меньшем.

Алгоритмы растеризации использующие баррицентрические координаты оказались значительно медленее при большем заполнении экрана, однако результат выдаваемый ими выглядит лучше с визуальной точки зрения. Так же стоит отметить, что оптимизация алгоритма с барицентрическими координатами не дает особого прироста к скорости выполнения на малой доле заполнения экрана.

Заключение

В ходе проведенной работы было разработано программное обеспечение, которе позволяет:

1) загружать параметры трехмерной модель из файла;
2) редактировать трехмерные модели на уровне вершин, ребер, полигонов;
3) просматривать трехмерную модель как с отсечением невидимых ребер и граней, так и без отсечения с любой позиции наблюдателя.

Были выполнены следующие задачи:

1) проведен анализ алгоритмов для удаления невидимых линий и граней, закраски, освещения моделей и выделены наиболее подходящие для программы;
2) реализованы выбранные алгоритмы и структуры данных;
3) разработано программное обеспечение для отображения трехмерной сцены;
4) выполнено исследование на основе разработанной программы.

Список использованных источников

1. *Преобразования в OpenGL [Эл. ресурс]*. Режим доступа: http://www.songho.ca/opengl/gl_transform.html (дата обращения: 16.07.2022).
2. *Е.В. Шикин Компьютерная графика. Динамика, реалистические изображения / Е.В. Шикин, А.В. Боресков. – М.: Диалог-МИФИ, 1995. – 288 с.*
3. *Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# / Дж. Рихтер. – 3-е изд. – СПб : Питер, 2012. – 928 с.*
4. *Windows 11, version 22H2 [Эл. ресурс]*. Режим доступа: https://www.microsoft.com/software-download/windows11 (дата обращения: 14.10.2022).
5. *Процессор Intel® Core™ i5 [Эл. ресурс]*. Режим доступа: https://www.intel.com/content/www/us/en/support/ru-banner-inside.html?123 (дата обращения: 14.10.2022).
6. *Stopwatch [Эл. ресурс]*. Режим доступа: https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics?view=windowsdesktop-7.0 (дата обращения: 13.10.2022).