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

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:
«Разработка программного обеспечения для визуализации моделей музейных экспонатов»

Студент ИУ7-53Б (Группа) _____ (Подпись, дата) И.А. Гринкевич (И.О. Фамилия)

Руководитель _____ (Подпись, дата) А.Л. Исаев (И.О. Фамилия)

2022 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Музеи в современном мире представляют собой неотъемлемую часть культурной инфраструктуры практически любого города на планете. Важно чтоб экспонаты в музее были расставлены так, чтоб человеку было максимально удобно воспринимать информацию и не заблудиться. Этого можно достигнуть эмпирическим способом, но вживую продвигать экспонаты может быть тяжело физически и опасно для сохранности самих экспонатов, поэтому хорошим решением проблемы будет реализация программного обеспечения для визуализации сцены с музейными экспонатами.

Целью данной работы является разработка программного обеспечения для визуализации моделей музейных экспонатов. Для ее достижения были поставлены следующие цели:

- проведение анализа предметной области;
- разработка программных инструментов, которые позволяют масштабировать, перемещать, вращать музейные экспонаты;
- разработка программных инструментов, которые позволяют вращать камеру относительно сцены;
- разработка отчета о проделанной работе в виде расчетно-пояснительной записки;
- подготовка презентации к защите работы.

1. Аналитическая часть

В данном разделе проведен обзор предметной области: рассмотрены существующие методы удаления невидимых линий и поверхностей, методы закрашивания и методы удаления теней. Из рассмотренных методов были выбраны оптимальные для поставленной задачи. Также в данном разделе формализованы объекты синтезируемой сцены и рассмотрены существующие решения.

1.1 Формализация объектов синтезируемой сцены

Сцена состоит из:

- Источников света – является материальной точкой, излучающей свет во все стороны. Положение задается координатами X, Y, Z.
- Пол – горизонтальная поверхность, на которой размещаются экспонаты.
- Экспонат – модель. Положение задается координатами X, Y, Z.

В качестве варианта представления экспонатов была выбрана полигональная модель, то есть множества многоугольников, состоящих из прямых ребер и вершин. Образованная плоскость называется гранью, которая обычно представляет собой трехстороннюю геометрическую форму, или «треугольный многоугольник». Также существуют четырехсторонние «четверки» и «n-угольники» с несколькими вершинами. Каждый полигон соединяется с другими полигонами, и вместе они создают полигональную сетку, которая по сути является 3D-моделью.

1.2 Анализ алгоритмов удаления невидимых линий и поверхностей

В данном подразделе будут рассмотрены алгоритмы удаления невидимых линий и поверхностей.

1.2.1 Алгоритм обратной трассировки лучей

Обратная трассировка лучей — это метод рендеринга, который может реалистично имитировать освещение сцены и ее объектов путем рендеринга физически точных отражений, преломлений, теней и непрямого освещения. Трассировка лучей генерирует изображения компьютерной графики путем отслежи-

вания пути света от камеры обзора (которая определяет ваш взгляд на сцену) через плоскость 2D-просмотра (плоскость пикселей), в 3D-сцену и обратно к источникам света. Проходя через сцену, свет может отражаться от одного объекта к другому (вызывая отражение), блокироваться объектами (вызывая тени) или проходить сквозь прозрачные или полупрозрачные объекты (вызывая преломление). Все эти взаимодействия объединяются для получения окончательного цвета и освещения пикселя, который затем отображается на экране. Этот обратный процесс трассировки глаза/камеры к источнику света выбран потому, что он намного эффективнее, чем трассировка всех световых лучей, испускаемых источниками света в нескольких направлениях.

Несомненным плюсом является универсальность данного алгоритма (работа с несколькими источниками освещения, реализация различных оптических явлений). Так же при использовании данного алгоритма не приходится делать дополнительных вычислений для нахождения теней.

Положительной стороной так же является возможность алгоритма работать на параллельных вычислительных системах, т.к. вычисление каждого пикселя не зависит от остальных.

Минусом является большой объём вычислений, требуемый алгоритмом. Но есть оптимизации позволяющие сократить время выполнения, например погружение объектов в прямоугольную или сферическую «оболочку», позволяющую быстро отбрасывать большинство объектов, через которые не проходит, рассматриваемый луч.

1.2.2 Алгоритм, использующий Z буфер

Его также называют «алгоритмом буфера глубины». Алгоритм буфера глубины является простейшим алгоритмом пространства изображения. Для каждого пикселя на экране дисплея записывается глубина объекта в пределах пикселя, который находится ближе всего к наблюдателю. Помимо глубины, также записывается интенсивность, которая должна отображаться, чтобы пока-

зать объект. Алгоритм буфера глубины требует 2 массива, интенсивности и глубины, каждый из которых индексируется координатами пикселя (x, y).

Серьезным плюсом данного алгоритма являются его простота. Сцена может быть произвольной сложности. Алгоритм не требует сортировки объектов, а сложность линейно зависит от количества рассматриваемых поверхностей. Данный алгоритм требует относительно много памяти, но малый для современных систем.

Недостатками является трудность устранения лестничного эффекта и реализации эффекта прозрачности.

1.2.3 Алгоритм Робертса

Оператор Робертс Кросс выполняет простую, быструю для вычислений двумерную измерение пространственного градиента на изображении. Таким образом, выделяет области высокой пространственной частоты, которые часто соответствуют края. В наиболее распространенном использовании входные данные для оператора представляют собой изображение в градациях серого, как и вывод. Значения пикселей в каждой точке изображения выходные данные представляют собой предполагаемую абсолютную величину пространственного градиент входного изображения в этой точке.

Существенным недостатком данного алгоритма является его трудоемкость, она имеет квадратичную зависимость от количества объектов на сцене. В следствии чего алгоритм будет занимать много времени при большом количестве экспонатов.

Алгоритм работает в объектном пространстве, что обеспечивает высокую точность и реалистичность картинки.

Существуют некоторые оптимизации алгоритма, например сортировка объектов по оси z. Однако некоторые реализации оптимизаций сложны, что затрудняет разработку

1.2.4 Алгоритм Варнока

Данный алгоритм основан на методе «разделяй и властвуй», когда видимая (просматриваемая) область последовательно делится на все меньшие и меньшие прямоугольники, пока не будет обнаружена упрощенная область.

Алгоритм Варнока имеет высокую производительность при достаточной однородности картинки, что не подходит для задачи отображения многополигональных экспонатов.

Вывод

Алгоритм, использующий Z буфер, является оптимальным для данной задачи, так как обеспечивает достаточную реалистичность изображения при относительно небольших требованиях к ресурсам.

1.3 Анализ методов закрашивания

В данном подразделе будут рассмотрены известные методы закрашивания.

1.3.1 Простая закраска

Основная идея алгоритма заключается в том, что он использует только одну нормаль к поверхности для каждого полигона. Сам цвет однороден (неизменен) на этом полигоне.

Серьезными плюсами данного алгоритма являются его простата реализации и эффективность по времени.

Однако большим недостатком является нереалистичность картинки при отображении, например в результате работы алгоритма на сфере будут видны грани.

1.3.2 Закраска по Гуро

Основная идея алгоритма заключается в том, что для каждой вершины существует своя нормаль, а цвет вычисляется в вершинном шейдере. Затем этот цвет интерполируется по полигону. Поскольку вершин меньше, чем фрагмен-

тов, вычисление цвета для каждой вершины и его интерполяция более эффективны, чем вычисление для каждого фрагмента.

Закраска по Гуро выполняет сглаживание на основе интерполяции интенсивности. С одной стороны, это решает проблему, присущую алгоритму простой закраски, в следствии которой сфера будет иметь видимые грани. Но с другой стороны, закраска по Гуро не учитывает кривизну поверхности, в следствии чего можно получить плоское изображение.

1.3.3 Закраска по Фонгу

Основная идея алгоритма в том, что нормаль от вершин интерполируется. Цвет рассчитывается по фрагментам с учетом интерполированной нормали. Для аппроксимации сферы это оптимально, потому что интерполированные нормали были бы точно такими же, какие были бы у идеальной сферы. Поскольку цвет рассчитывается на основе нормалей, он будет рассчитываться так, как если бы это была идеальная сфера.

В отличии от закраски по Гуро, закраска по Фонгу производит интерполяцию по нормали, а не по интенсивности. Это решает проблему получения плоской картинки в случае отображения трехмерного объекта с гранями одинакового цвета. Картинка получается более реалистичной.

Серьезным минусом алгоритма является затратность по времени.

Вывод

Для данной задачи оптимальным алгоритмом закрашивания является простая закраска, так как он обеспечивает достаточную реалистичность при использовании на полигональных моделях при относительно небольших затратах ресурсов компьютера.

1.4 Анализ алгоритмов построения теней

При работе алгоритма обратной трассировки лучей тени от объектов строятся по ходу выполнения алгоритма. Пиксель будет затемнен, если испускаемый из камеры луч попадает на объект, но не попадает в источник света. Данный алгоритм не подходит для поставленной задачи, так как алгоритм обратной трассировки лучей не был выбран в качестве алгоритма удаления невидимый ребер и поверхностей.

Поскольку в качестве алгоритма удаления невидимых линий и поверхностей был выбран алгоритм, использующий Z буфер, будет использован алгоритм создания теневых карт. Данный алгоритм будет работать с использованием алгоритма, использующим Z буфер. Построение теней в таком случае аналогично удалению невидимых ребер и поверхностей, только положение камеры находится на месте источника света и найденные невидимые поверхности затемняются, а остальные нет.

1.5 Анализ существующих решений

На данный момент существует большое количество различных решений для работы с объёмными моделями, одно из самых популярных — Blender, его интерфейс представлен на рисунке 1.

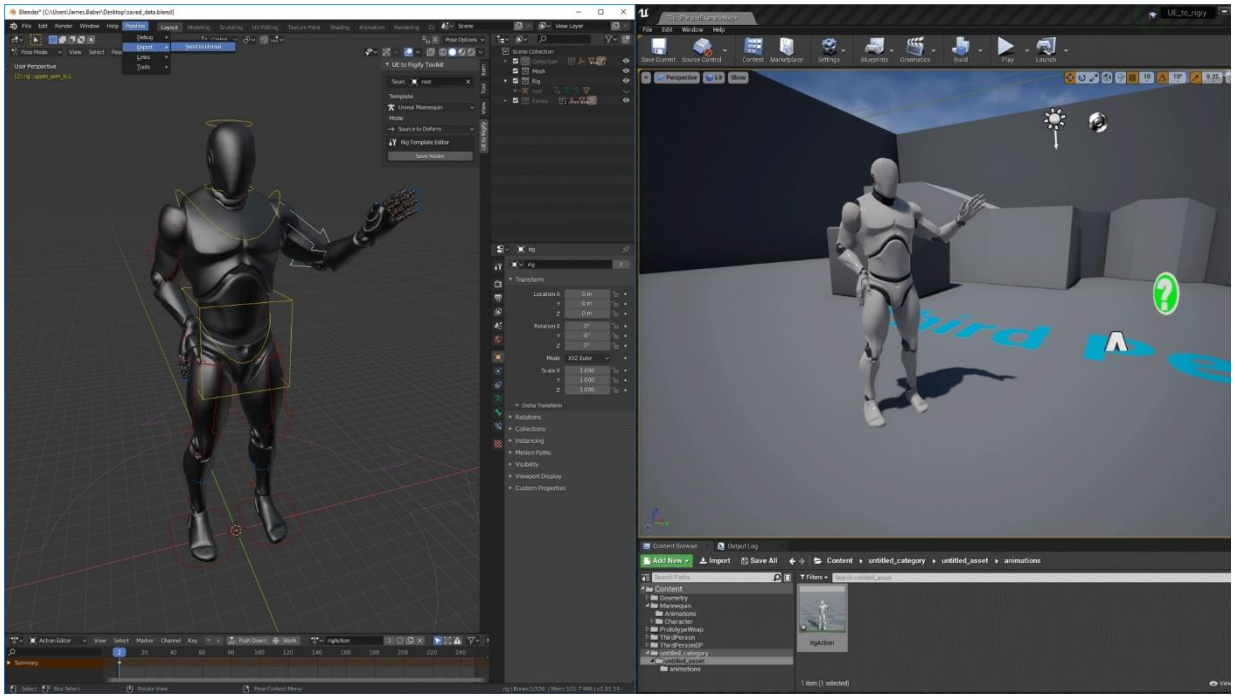


Рисунок 1 — Пример работы программы Blender

Данное программное обеспечение предоставляет много возможностей для манипуляции объёмными моделями, такими как: создание объекта, загрузка стороннего объекта, масштабирование объекта, вращение объекта и перемещение объекта. Главным недостатком данной программы является высокие требования к ресурсам компьютера.

Вывод

В данном разделе был проведен обзор предметной области: рассмотрены существующие методы удаления невидимых линий и поверхностей, методы закрашивания и методы удаления теней. Из рассмотренных методов были выбраны оптимальные для поставленной задачи. Также в данном разделе были формализованы объекты синтезируемой сцены и рассмотрены существующие решения.

В качестве алгоритма удаления невидимых линий и поверхностей был выбран алгоритм, использующий Z буфер. В качестве алгоритма закрашивания была выбрана простая закраска. В качестве алгоритма построения теней был

выбран алгоритм построения теневых карт с помощью алгоритма, использующего Z буфер.

2. Конструкторская часть

В данном разделе будут рассмотрены требования алгоритмы визуализации сцены, а также используемые типы и структуры данных.

2.1 Общий алгоритм визуализации трехмерной сцены

Визуализации трехмерной сцены музейных экспонатов происходит поэтапно. Рассмотрим алгоритм визуализации.

1. Задать объекты сцены;
2. Задать источники света;
3. Задать положение наблюдателя;
4. Для каждого полигона высчитать нормаль и интенсивность цвета, найти внутренние пиксели;
5. Используя алгоритм Z буфера получить изображение сцены, и буфер глубины;
6. Используя алгоритм Z буфера получить буфер глубины для источника света;
7. Найти затененные участки при помощи наложения двух буферов.

2.2 Алгоритм, использующий Z буфер

Алгоритм, использующий Z буфер, выглядит следующим образом:

1. Для всех пикселей на экране установить глубину [x, y] на 1,0 и интенсивность [x, y] на значение фона;
2. Для каждого многоугольника в сцене найти все пиксели (x, y), которые лежат в пределах границ многоугольника при проецировании на экран. Для каждого из этих пикселей вычислить глубину z многоугольника в (x, y). Если z < глубины [x, y], этот многоугольник находится ближе к наблюдателю, чем другие, уже записанные для этого пикселя. В этом случае установить для глубины [x, y] значение z, а для интенсивности [x, y] значение, соответствующее затенению полигона. Если вместо этого z > глубины [x, y], многоугольник, уже записанный в (x, y), лежит ближе к наблюдателю, чем этот новый многоугольник, и никаких действий не предпринимается;
3. Все полигоны обработаны; массив интенсивности будет содержать решение.

Схема данного алгоритма продемонстрирована на рисунке 2.

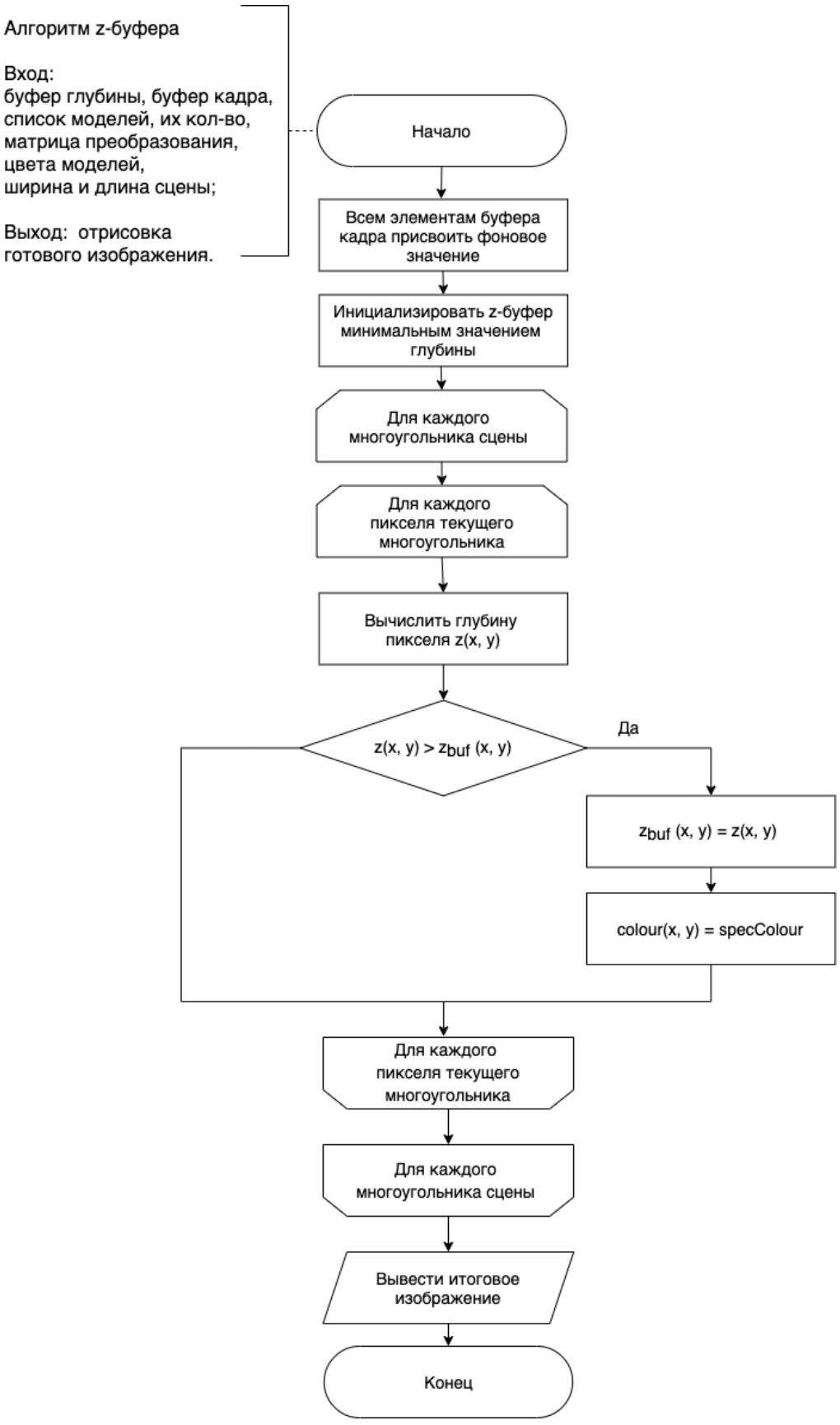


Рисунок 2 — Схема алгоритма, использующего Z буфер.

2.3 Простой метод освещения

В простом методе освещения интенсивность рассчитывается по закону Ламберта:

$$I = I_0 * cos(\alpha), \text{ где} \quad (2.1)$$

I – результирующая интенсивность света в точке,

I_0 – интенсивность источника,

α – угол между нормалью к поверхности и вектором направления света.

2.4 Выбор используемых типов и структур данных

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

- Источник света – задается вектором, обозначающим с какой стороны от сцены находится источник;
- Сцена – задается объектами сцены;
- Объекты сцены (экспонаты) – задаются вершинами и гранями;
- Математические абстракции:
  - Точка – хранит координаты x, y, z;
  - Вектор – хранит направление по x, y, z;
  - Многоугольник – хранит вершины, нормаль, цвет;
- Интерфейс – используются библиотечные классы для предоставления доступа к интерфейсу.

Вывод

В данном разделе были рассмотрены требования алгоритмы визуализации сцены, общий алгоритм визуализации сцены, используемые типы и структуры данных. Также был рассмотрен простой метод освещения, используемый в программном обеспечении.

3. Конструкторская часть

В данном разделе будут представлены требования к программному обеспечению, средства реализации, структура и состав классов, сведенья о модулях программы, а также реализации выбранных ранее алгоритмов.

3.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

- визуальное отображение сцены;
- вращение камеры относительно сцены;
- перемещение источников;
- перемещение объектов;
- вращение объектов;
- масштабирование объектов.

3.2 Средства реализации

В качестве языка программирования был выбран C#, так как:

- данный язык является объектно-ориентированным, что позволяет удобно реализовать программное обеспечение;
- язык имеет простой и в то же время удобный конструктор графического интерфейса Windows Forms;
- C# интересен для изучения и использования в будущем, как современный язык.

В качестве среды разработки был выбран JetBrains Rider, так как он бесплатен для студентов и обладает всеми необходимыми средствами для реализации данного программного обеспечения.

3.3 Структура и состав классов

В этом разделе будут рассмотрена структура и состав классов (представлены на рисунках 3.1, 3.2).

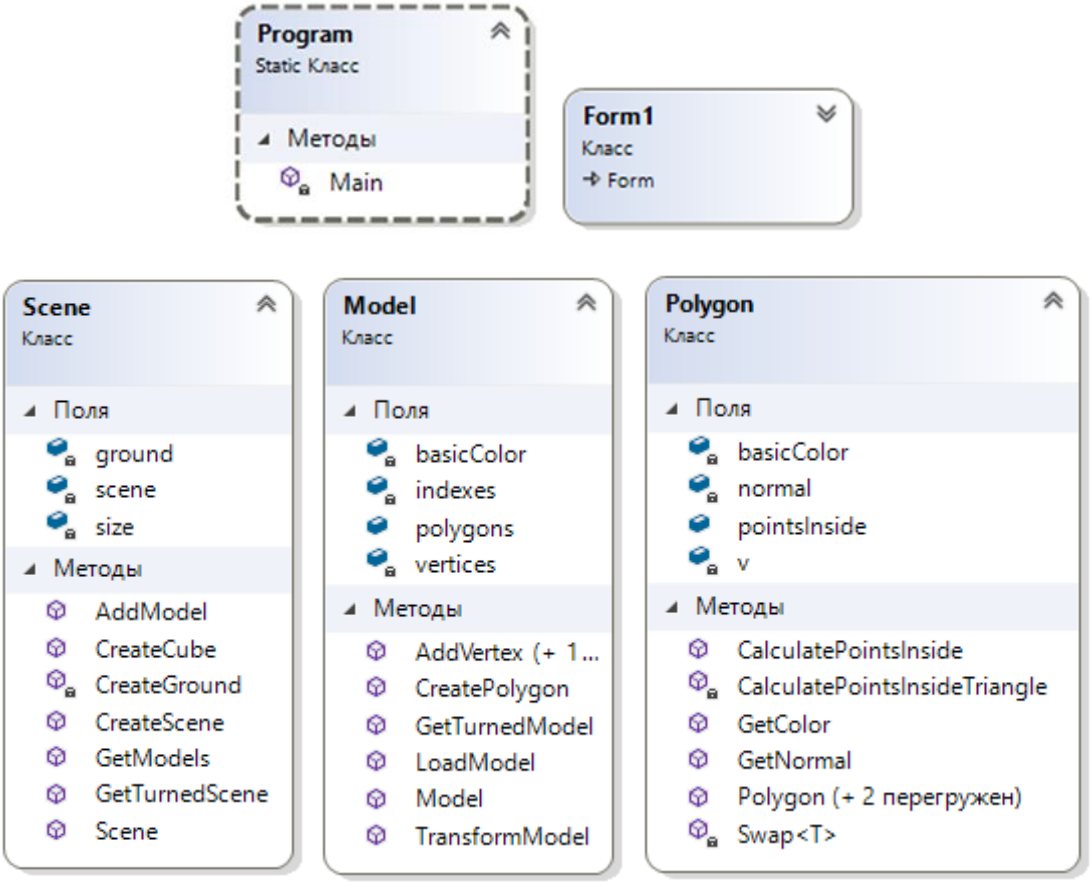


Рисунок 3.1 — Структура классов Program, Form1, Scene, Model, Polygon.

На рисунке 3.1 представлены следующие классы:

– Program – входная точка в программу;

– Form1 – класс пользовательского интерфейса;

– Scene – хранит информацию о сцене: размеры, модели внутри сцены, имеет методы создания сцены, ее преобразования;

– Model – хранит информацию о модели: ее вершины, индексы вершин для создания многоугольников, многоугольники, имеет методы загрузки и преобразования модели;
– Polygon – класс многоугольника, хранит цвет, вершины, нормали и точки внутри, при необходимости, имеет методы вычисления нормали и получения точек внутри многоугольника.

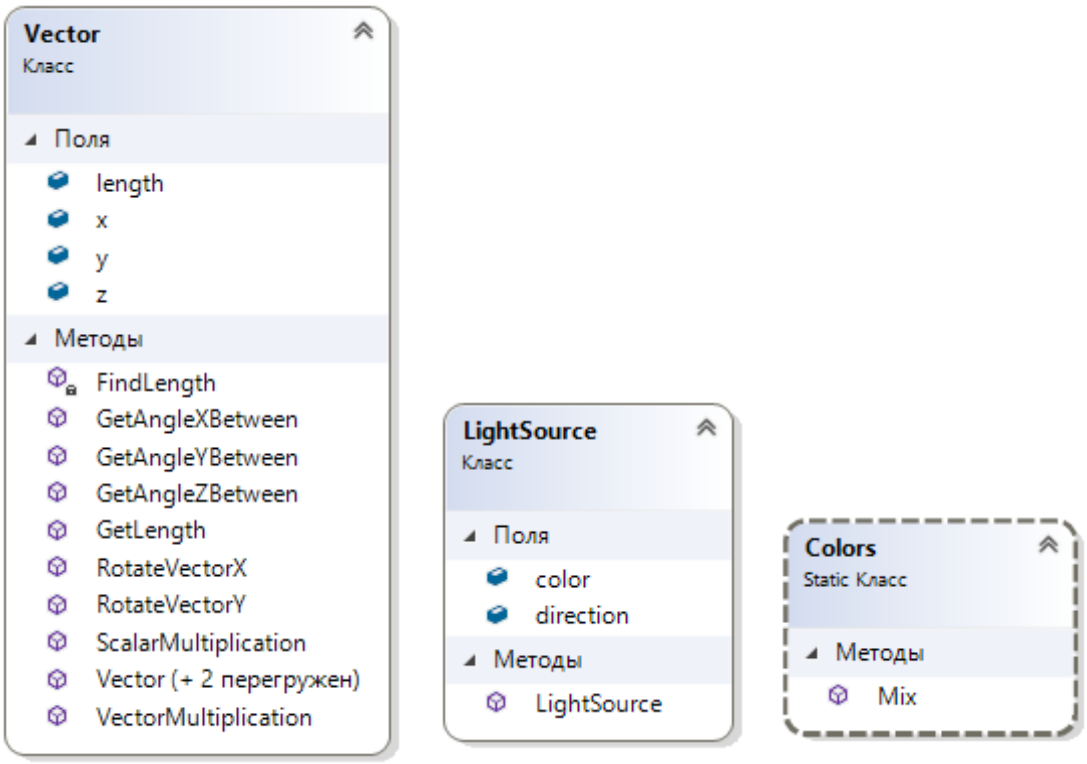


Рисунок 3.2 — Структура классов Vector, LightSource, Colors.

На данном рисунке представлены схемы классов:
– Vector – хранит направление вектора и его длину, имеет методы поиска длины вектора, нахождение углов между двумя векторами, векторное и скалярное умножение векторов;
– LightSource – хранит в себе цвет света и направление света;
– Colors – статический класс имеющий метод смешивания цветов;

3.4 Сведенья о модулях программы

В программе используются следующие модули:

– Program.cs – главная точка входа в приложение;
– Form1.cs – интерфейс;
– Light.cs – описание источников света;
– Scene.cs – описание сцены, методы взаимодействия с ней;
– Model.cs – описание объектов сцены, методы взаимодействия с моделью и ее частями;
– Zbuffer.cs – алгоритм Z буфера;
– Colors.cs – взаимодействие цветов;
– Transformation.cs – функции преобразования координат.

3.5 Реализация алгоритмов

В данном подразделе представлены реализации алгоритмов. В листинге 3.1 представлена реализация алгоритма, использующего Z буфер. В листинге 3.2 представлен алгоритм добавления теней. В листинге 3.3 представлен алгоритм нахождения цвета пикселя с учетом затемнения. В листингах 3.4 – 3.6 представлены алгоритмы вращения, масштабирования и перемещения модели соответственно.

Листинг 3.1— Реализация алгоритма, использующего Z буфер.

```
    class Zbuffer
    {
        private Bitmap img;
        private Color[][] imgPar;
        private Bitmap imgFromSun;
        private Bitmap imgFromSun2;
        private int[][] Zbuf;
        private int[][] ZbufFromSun;
        private int[][] ZbufFromSun2;
        LightSource sun;
        LightSource sun2;
        Size size;
        double tettax, tettay, tettaz;
        double tettax2, tettay2, tettaz2;

        private static readonly int zBackground = -10000;


        public Zbuffer(Scene s, Size size, LightSource sun,
LightSource sun2)
        {
            img = new Bitmap(size.Width, size.Height);
            imgFromSun = new Bitmap(size.Width, size.Height);
            imgFromSun2 = new Bitmap(size.Width, size.Height);

            InitBuf(ref Zbuf, size.Width, size.Height, zBack-
ground);
            InitBuf(ref ZbufFromSun, size.Width, size.Height,
zBackground);
            InitBuf(ref ZbufFromSun2, size.Width, size.Height,
zBackground);

            this.sun = sun;
            this.sun2 = sun2;
            this.size = size;
            InitTeta();

            imgPar = new Color[size.Width][];
            for (int i = 0; i < size.Width; i++)
            {
                imgPar[i] = new Color[size.Height];
            }

            foreach (Model m in s.GetModels())
            {
                ProcessModel(Zbuf, img, m);
                ProcessModelForSun(ZbufFromSun, imgFromSun,
m.GetTurnedModel(tettax, tettay, tettaz, new Point3D(0, 0, 0)),
sun);
```

```
                ProcessModelForSun(ZbufFromSun2,      imgFromSun2,
m.GetTurnedModel(tettax2, tettay2, tettaz2, new Point3D(0, 0, 0)),
sun2);
            }
        }


        private void InitTeta()
        {
            tettax = sun.tetax;
            tettay = sun.tetay;
            tettaz = sun.tetaz;

            tettax2 = sun2.tetax;
            tettay2 = sun2.tetay;
            tettaz2 = sun2.tetaz;
        }


        private void InitBuf(ref int[][] buf, int w, int h, int
value)
        {
            buf = new int[h][];
            for (int i = 0; i < h; i++)
            {
                buf[i] = new int[w];
                for (int j = 0; j < w; j++)
                    buf[i][j] = value;
            }
        }


        private void ProcessModel(int[][] buffer, Bitmap image,
Model m)
        {
            Color draw;
            foreach (Polygon polygon in m.polygons)
            {
                polygon.CalculatePointsInside(img.Width,
img.Height);
                draw  =  Colors.Mix(polygon.GetColor(sun),  poly-
gon.GetColor(sun2), 0.5f);
                foreach (Point3D point in polygon.pointsInside)
                {
                    ProcessPoint(buffer, image, point, draw);
                }
            }
        }


        private void ProcessModelForSun(int[][] buffer, Bitmap
image, Model m, LightSource sun)
        {
```

```
            Color draw;
            foreach (Polygon polygon in m.polygons)
            {
                if (polygon.ignore)
                    continue;
                polygon.CalculatePointsInside(img.Width,
img.Height);
                draw = polygon.GetColor(sun);
                foreach (Point3D point in polygon.pointsInside)
                {
                    ProcessPoint(buffer, image, point, draw);
                }
            }
        }


        private void ProcessPoint(int[][] buffer, Bitmap image,
Point3D point, Color color, int w = 1000, int h = 500)
        {

            if (!(point.x < 0 || point.x >= w || point.y < 0 ||
point.y >= h))
            {
                if (point.z > buffer[(int)point.y][(int)point.x])
                {
                    buffer[(int)point.y][(int)point.x]          =
(int)point.z;
                    image.SetPixel((int)point.x,     (int)point.y,
color);
                }
            }
        }
    }
```

Листинг 3.2 — Реализация алгоритма добавления теней.

```
    public Bitmap AddShadows()
    {
        Bitmap hm = new Bitmap(size.Width, size.Height);

        for (int i = 0; i < size.Width; i++)
        {
            for (int j = 0; j < size.Height; j++)
            {
                int z = GetZ(i, j);
                if (z != zBackground)
                {
                    Point3D newCoord = Transfor-
mation.Transform(i, j, z, tettax, tettay, tettaz);
                    Point3D newCoord2 = Transfor-
mation.Transform(i, j, z, tettax2, tettay2, tettaz2);
```

```
                    Color curPixColor = img.GetPixel(i, j); ;
                    if (newCoord.x < 0 || newCoord.y < 0 || new-
Coord.x >= size.Width || newCoord.y >= size.Height)
                    {
                        hm.SetPixel(i, j, curPixColor);
                        continue;
                    }

                    Color c1, c2;

                    if (Zbuf-
FromSun[(int)newCoord.y][(int)newCoord.x] > newCoord.z + 5)
                    {
                        c1 = Colors.Mix(Color.Black, curPixColor,
0.4f);
                    }
                    else
                    {
                        c1 = curPixColor;
                    }

                    if (Zbuf-
FromSun2[(int)newCoord2.y][(int)newCoord2.x] > newCoord2.z + 5)
                    {
                        c2 = Colors.Mix(Color.Black, curPixColor,
0.4f);
                    }
                    else
                    {
                        c2 = curPixColor;
                    }

                    hm.SetPixel(i, j, Colors.Mix(c1, c2, 0.5f));
                }
            }
        }

        return hm;
    }
```

Листинг 3.3 — Реализация алгоритма нахождения цвета пикселя с учетом затемнения.

```
          public static double ScalarMultiplication(Vector a, Vec-
tor b)
     {
         return a.x * b.x + a.y * b.y + a.z * b.z;
     }

     public Color GetColor(LightSource light)
     {
         double cos = Vector.ScalarMultiplication(light.direction,
normal) /
              (light.direction.length * normal.length);

         if (cos <= 0 || cos != cos)
             return Colors.Mix(basicColor, Color.Black, 0.2f);

         return Colors.Mix(basicColor, Color.Black, (float)cos);

     }
```

Листинг 3.4 — Реализация алгоритма вращения модели.

```
      public static void Transform(ref double x, ref double y, ref
double z, double cosTetX, double sinTetX, double cosTetY, double
sinTetY, double cosTetZ, double sinTetZ, Point3D centre)
     {
         double x_tmp = x;
         double y_tmp = y;
         double z_tmp = z;
         RotateX(ref y_tmp, ref z_tmp, cosTetX, sinTetX, centre);
         RotateY(ref x_tmp, ref z_tmp, cosTetY, sinTetY, centre);
         RotateZ(ref x_tmp, ref y_tmp, cosTetZ, sinTetZ, centre);

         x = (int)x_tmp;
         y = (int)y_tmp;
         z = (int)z_tmp;
     }

     public void TransformModel(double tetax, double tetay, double
tetaz, Point3D cent)
     {
         tetax = tetax * Math.PI / 180;
         tetay = tetay * Math.PI / 180;
         tetaz = tetaz * Math.PI / 180;
         double cosTetX = Math.Cos(tetax), sinTetX =
Math.Sin(tetax);
         double cosTetY = Math.Cos(tetay), sinTetY =
Math.Sin(tetay);
```

```
        double cosTetZ = Math.Cos(tetaz), sinTetZ =
Math.Sin(tetaz);
        foreach (Point3D v in vertices)
        {
            Transformation.Transform(ref v.x, ref v.y, ref v.z,
cosTetX, sinTetX, cosTetY, sinTetY, cosTetZ, sinTetZ, cent);
        }
    }
```

Листинг 3.5 — Реализация алгоритма масштабирования модели.

```
    public void ScaleModel(double k, Point3D centre)
    {
        foreach (Point3D v in vertices)
        {
            Point3D changed = (v - centre) * k + centre;
            v.x = changed.x;
            v.y = changed.y;
            v.z = changed.z;
        }
    }
```

Листинг 3.6 — Реализация алгоритма перемещения модели.

```
         public void MoveModel(double tetax, double tetay, double
tetaz)
    {
        foreach (var v in vertices)
        {
            v.x += (int)tetax;
            v.y += (int)tetay;
            v.z += (int)tetaz;
        }
    }
```

Вывод

В данном разделе были представлены требования к программному обеспечению в соответствии с описанным ранее задание. Также были представлены средства реализации и обоснование их выбора в контексте данной работы. Была представлена структура и состав классов, используемых в программном обеспечении, реализующим визуализацию сцены, содержащей музейные экспонаты. Также были рассмотрены сведенья о модулях программы (их предназначение в

данной разрабатываемой программе) и реализации выбранных ранее алгоритмов на выбранном языке программирования.

4. Технологическая часть

В данном разделе будут приведены примеры работы программ, технические характеристики, постановка и результаты эксперимента.

4.1 Демонстрация работы программы

На рисунках 4.1 – 4.5 представлены результаты работы программы.

Рисунок 4.1 — Результат работы программы (вид спереди).

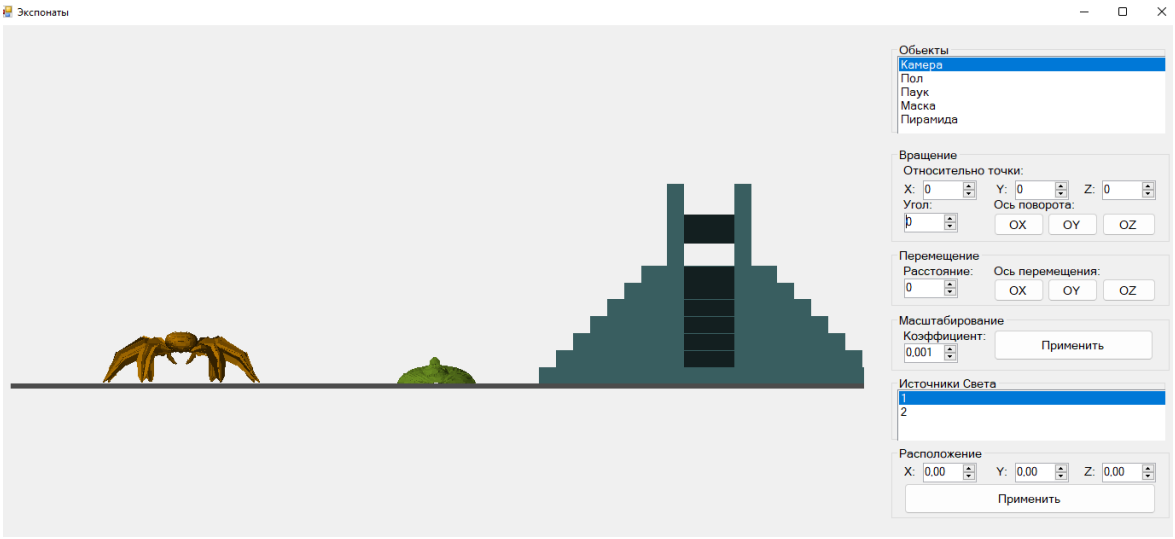


Рисунок 4.2 — Результат работы программы (вид сверху).

Рисунок 4.3 — Результат работы программы (вид сбоку).

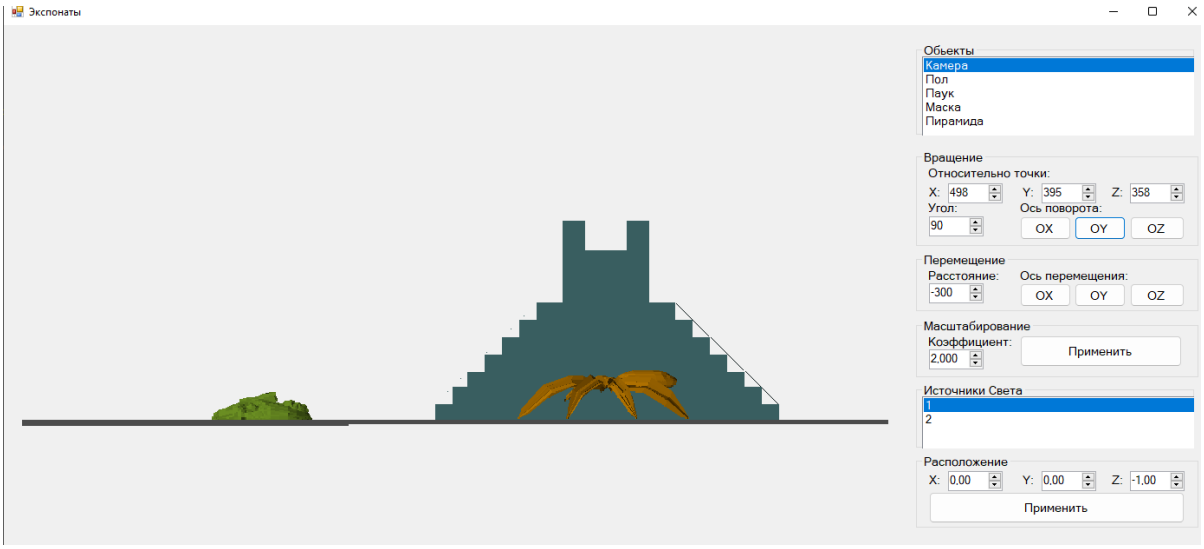


Рисунок 4.4 — Результат работы программы (вид под углами 45 градусов по осям OX, OY и OZ).

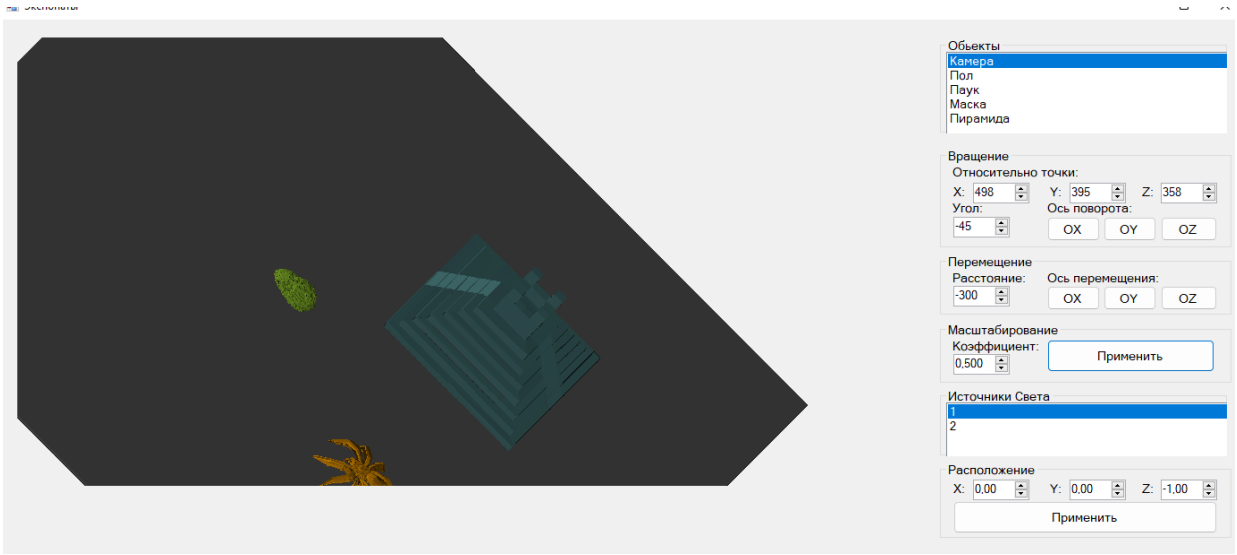
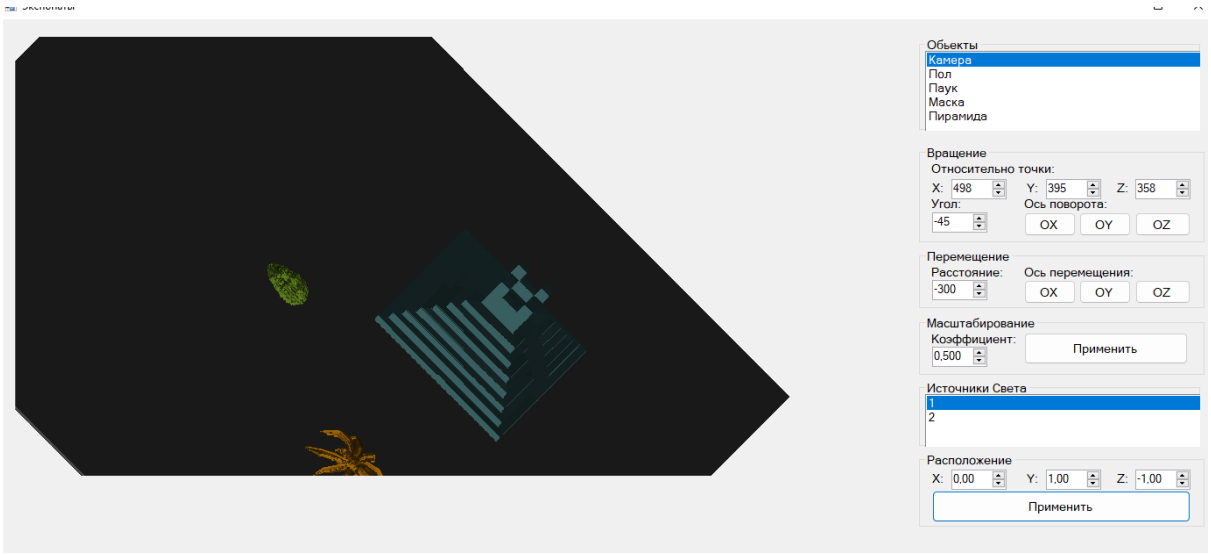


Рисунок 4.5 — Результат работы программы с другим расположением источников света.

Рисунок 4.6 — Результат работы программы с одним источником света.



4.2 Постановка эксперимента

Целью эксперимента является анализ времени, требующегося для визуализации сцены, содержащей разное количество экспонатов.

4.3 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, следующие:

- Операционная система Windows 11;
- Оперативная память 8 Гб;
- Процессор AMD Ryzen 5 4500U with Radeon Graphics 2.38 ГГц.

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

4.4 Результаты эксперимента

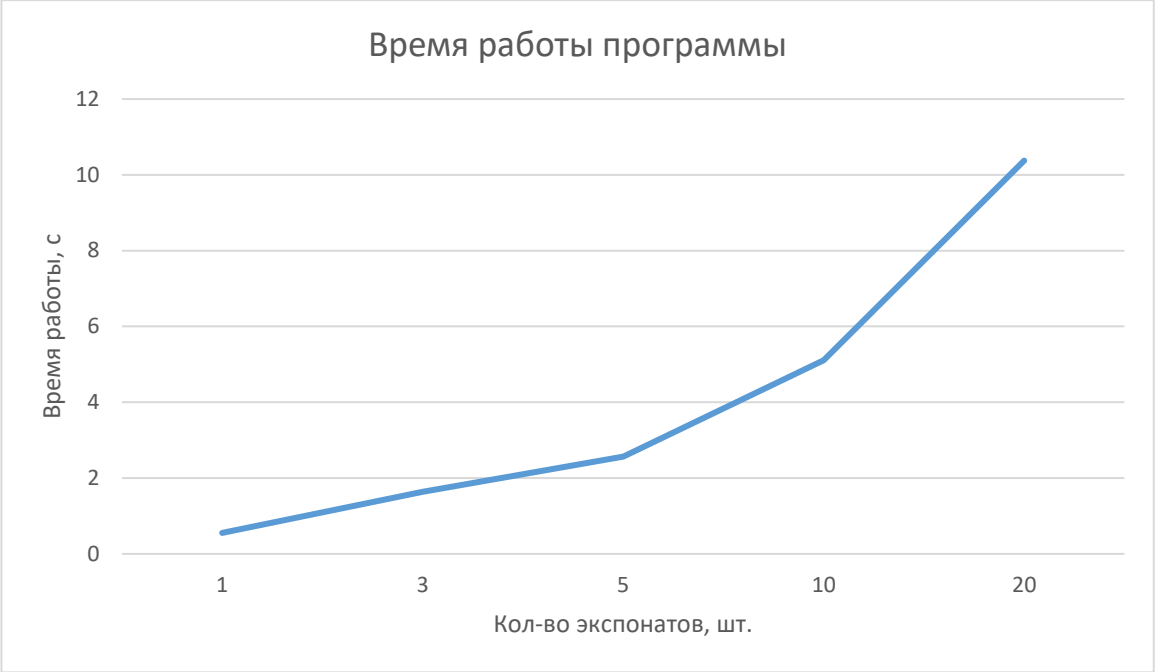
Результаты эксперимента представлены в таблице 4.1

Таблица 4.1 — Результаты эксперимента

Количество объектов, шт.	Время, с
1	0,5542
3	1,6345
5	2,5656
10	5,1124
20	10,3759

На рисунке 4.7 представлен график зависимости времени визуализации сцены от количества объектов на сцене.

Рисунок 4.6 — График зависимости времени визуализации сцены от количества объектов на сцене.



Вывод

В данном разделе были приведены примеры работы программ, технические характеристики, постановка и результаты эксперимента.

По результатам проводимого эксперимента найдена линейная зависимость между количеством объектов на сцене и временем, требуемым для построения изображения сцены.

ЗАКЛЮЧЕНИЕ

В ходе данной работы было разработано программное обеспечение для визуализации моделей музейных экспонатов.

Так же были достигнуты следующие цели:

- проведение анализа предметной области;
- разработка программных инструментов, которые позволяют масштабировать, перемещать, вращать музейные экспонаты;
- разработка программных инструментов, которые позволяют вращать камеру относительно сцены;
- разработка отчета о проделанной работе в виде расчетно-пояснительной записки;
- подготовка презентации к защите работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Семенов В. А., Алексеева Е. В., Морозов С. В., Тарлапан О. А. Композиционный подход к построению программных приложений визуализации // Труды ИСП РАН. 2004. №.

2. Асеева Е.Н., Авдеюк О.А., Асеева С.Д., Авдеюк Д.Н. Алгоритмы и программные средства машинной графики для построения и визуализации твердотельных моделей // ИВД. 2019. №1 (52).

3. Русанова Я. М., Чердынцева М. И. Некоторые проблемы визуализации трехмерных сцен в реальном времени // Прикладная информатика. 2008. №6.

4. Семенов В. А., Крылов П. Б., Морозов С. В., Роминов М. Г., Тарлапан О. А. Объектно-ориентированная методология разработки интегрированных приложений моделирования и визуализации // Труды ИСП РАН. 2000.

5. Попов Андрей Сергеевич Обзор методов реалистичной визуализации и постановка задачи исследования // Радиоэлектроника и информатика. 2006. №4.