МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ижевский государственный технический университет имени

М. Т. Калашникова»

Факультет «Информационные технологии»

Кафедра «Программное обеспечение»

Работа защищена с оценкой

«\_\_\_\_\_\_\_\_\_\_»

Дата \_\_\_\_\_\_\_\_\_\_\_\_

Подпись\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Компьютерная графика»

на тему: «Формирование и визуализация 3D-изображения с использованием однородных координат»

Выполнил:

студент гр. Б22-191-1 И.Ю. Кабалин

Руководитель:

к.т.н., доцент Л.Н. Левицкая

Ижевск 2025

Рецензия:

степень достижения поставленной цели работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

полнота разработки темы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

уровень самостоятельности работы обучающегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

недостатки работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# ПОСТАНОВКА ЗАДАЧИ

Цель курсовой работы — разработка программного средства, обеспечивающего формирование и визуализацию 3D-изображения с использованием математического аппарата однородных координат. Задача реализуется в рамках дисциплины «Компьютерная графика» и направлена на закрепление теоретических знаний и развитие практических навыков по программированию трехмерной графики.

В рамках курсовой работы требуется:

1. Загрузить 3D-модель из файла формата .obj.
2. Построить ортогональную проекцию сцены;
3. Определить положение наблюдателя (камеры);
4. Построить матрицу вида (LookAt) для перехода в систему координат наблюдателя;
5. Построить матрицы линейных преобразований: сдвига, поворота, масштабирования;
6. Построить итоговую матрицу преобразований через перемножение (суперпозицию) всех матриц;
7. Преобразовать координаты модели с учётом итоговой матрицы;
8. Реализовать Z-буфер для удаления невидимых поверхностей;
9. Реализовать освещение с использованием нормалей к треугольникам;
10. Построить матрицу Viewport для перевода нормализованных координат в экранные;
11. Визуализировать 3D-сцену на экране;
12. Реализовать возможность перемещения наблюдателя вокруг объекта.

Программная реализация должна быть выполнена в одной из сред разработки (Visual Studio) с использованием языков программирования C, C++, C# или Java.

# ОПИСАНИЕ ПРОГРАММЫ

# Структура входных данных

Файл в формате .obj, содержащий 3D-модель изображения.

* 1. Структура выходных данных

Сформированное и визуализированное 3D-изображение, доступное для просмотра в окне программы.

* 1. Алгоритм программы

После запуска программы пользователь видит окно приложения, предназначенного для отображения трёхмерной сцены, загружаемой из файла формата .obj. На первом этапе происходит считывание содержимого данного файла. Каждая строка файла анализируется: если строка начинается с v, она содержит координаты вершины, и они добавляются в список всех вершин сцены. При этом одновременно фиксируются минимальные и максимальные значения координат по каждой из трёх осей (X, Y, Z), чтобы в дальнейшем корректно масштабировать и проецировать модель. Если строка содержит ключевое слово vn, то из неё извлекаются координаты нормали, которые также добавляются в отдельный список. При встрече строки, начинающейся с f, производится разбор индексов вершин и нормалей, из которых состоит грань, и на их основе формируется треугольник — минимальный графический примитив, используемый для отображения поверхности. Каждая грань добавляется к соответствующему объекту сцены в зависимости от того, как он обозначен в .obj файле с помощью строки o.

После завершения загрузки модели программа переходит к формированию матрицы преобразований, которая включает в себя несколько этапов. Сначала создаётся матрица масштабирования, уменьшающая модель до нужного размера. Далее формируется матрица камеры (LookAt), определяющая положение наблюдателя и направление взгляда в сцену. Следом создаётся матрица ортогональной проекции, преобразующая трёхмерные координаты модели в плоскость с сохранением параллельности линий. Финальным шагом является создание матрицы viewport-преобразования, которая переводит нормализованные координаты в координаты пикселей окна отображения. Все матрицы объединяются в одну итоговую путём последовательного перемножения: сначала масштаб, затем камера, проекция и, наконец, viewport.

При каждом обновлении сцены (например, при нажатии клавиш для вращения модели), программа создаёт матрицу поворота вокруг оси Y на заданный угол. Эта матрица также умножается на общую матрицу преобразования. Далее для каждой вершины модели вычисляется её новое положение путём умножения на объединённую матрицу. Таким образом, происходит преобразование координат модели в экранные координаты. Полученные полигоны сортируются по глубине (Z-координате), чтобы соблюсти порядок отрисовки от дальнего к ближнему.

Перед непосредственной отрисовкой сцены инициализируется двумерная структура — Z-буфер. В каждой ячейке буфера хранится значение глубины (Z) ближайшего к камере объекта для соответствующего пикселя. Изначально буфер заполняется минимальными значениями, и при отрисовке каждый новый фрагмент сцены сравнивается по глубине с текущим значением буфера.

Далее программа выполняет растеризацию каждого треугольника. Она делит треугольник на горизонтальные строки (scanlines) и вычисляет для каждой строки координаты левой и правой границы. Затем происходит интерполяция координат по X и Z, а также нормалей в вершинах. Для каждого пикселя вдоль строки определяется его глубина (Z), и если она больше текущего значения в Z-буфере, то пиксель перерисовывается, а буфер обновляется.

Для визуального реализма в программе реализовано освещение по модели Фонга. Эта модель включает в себя три компонента: фоновую (ambient), диффузную (diffuse) и зеркальную (specular). В расчётах учитывается нормаль в точке, направление источника света и направление взгляда. Итоговый цвет пикселя формируется как сумма этих трёх составляющих и используется при отрисовке через элементарный прямоугольник.

Таким образом, пользователь может управлять сценой, вращая её с помощью клавиш, а программа будет динамически пересчитывать все преобразования, пересоздавать Z-буфер и перерисовывать сцену с учётом новых координат и освещения. Всё это обеспечивает визуализацию 3D-объекта в реальном времени с базовым освещением и глубиной.

2.4. Результат вывода программы

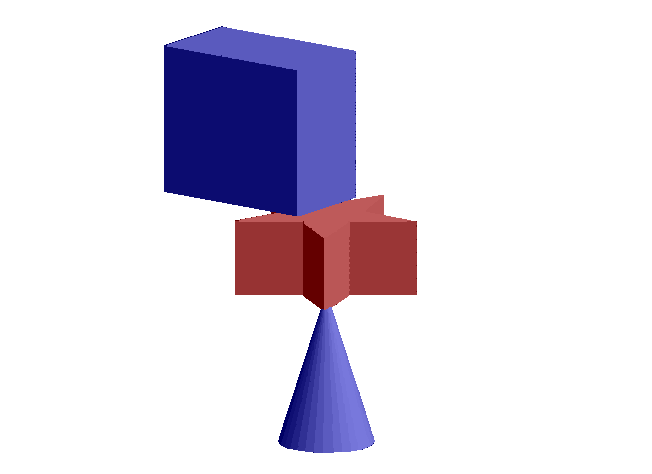


Рис. 1. Вывод графической сцены.

3. ЛИСТИНГ ТЕСКСТА ПРОГРАММЫ

Файл ViewerForm.cs:

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Windows.Forms;

namespace KursovayaKG

{

public class Object3D

{

public List<Point3D[]> faces = new List<Point3D[]>();

public List<Point3D[]> transformedFaces = new List<Point3D[]>();

public Color baseColor;

public Object3D(Color color)

{

baseColor = color;

}

}

public sealed partial class ViewerForm : Form

{

private const string ObjFileName = "9.obj";

private Object3D circleObject = new Object3D(Color.FromArgb(100, 0, 0));

private Object3D pyramidObject = new Object3D(Color.FromArgb(0, 0, 100));

private Mat4 fullTransform;

private static int angleY;

private readonly List<List<double>> depthMap = new List<List<double>>();

private double minX = 1e6, maxX = -1e6;

private double minY = 1e6, maxY = -1e6;

private double minZ = 1e6, maxZ = -1e6;

public ViewerForm()

{

InitializeComponent();

Width = 1280;

Height = 760;

BackColor = Color.White;

SetStyle(ControlStyles.AllPaintingInWmPaint | ControlStyles.OptimizedDoubleBuffer | ControlStyles.UserPaint, true);

ParseObjFile();

BuildFullTransform();

Paint += OnRender;

KeyDown += OnKeyInput;

}

private void OnRender(object sender, PaintEventArgs e)

{

ApplyModelTransformations();

DrawScene(e.Graphics);

}

private void OnKeyInput(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Right)

angleY -= 10;

else if (e.KeyCode == Keys.Left)

angleY += 10;

Refresh();

}

private void ParseObjFile()

{

var vertexList = new List<Vec3>();

var normalList = new List<Vec3>();

string currentObject = "";

foreach (var line in File.ReadAllLines(ObjFileName))

{

var parts = line.Replace('.', ',').Split(' ');

switch (parts[0])

{

case "o":

currentObject = parts[1];

break;

case "v":

var vx = double.Parse(parts[1]);

var vy = double.Parse(parts[2]);

var vz = double.Parse(parts[3]);

vertexList.Add(new Vec3(vx, vy, vz));

minX = Math.Min(minX, vx); maxX = Math.Max(maxX, vx);

minY = Math.Min(minY, vy); maxY = Math.Max(maxY, vy);

minZ = Math.Min(minZ, vz); maxZ = Math.Max(maxZ, vz);

break;

case "vn":

normalList.Add(new Vec3(

double.Parse(parts[1]),

double.Parse(parts[2]),

double.Parse(parts[3])));

break;

case "f":

var indices = parts.Skip(1)

.Select(p => p.Split('/'))

.ToArray();

var polygon = indices.Select(index =>

{

int vi = int.Parse(index[0]) - 1;

int ni = int.Parse(index[2]) - 1;

return new Point3D(vertexList[vi], normalList[ni]);

}).ToArray();

if (currentObject.Contains("Окружность"))

circleObject.faces.Add(polygon);

else

pyramidObject.faces.Add(polygon);

break;

}

}

}

private void BuildFullTransform()

{

var viewport = Mat4.CreateViewport(250, -200, Width, Height, maxZ - minZ);

var projection = Mat4.CreateOrthographic(minX, maxX, minY, maxY, maxZ, minZ);

var cameraPosition = new Vec3(0.6, 0.4, 0.8);

var lookAtPoint = new Vec3(0, 0, 0);

var forward = cameraPosition - lookAtPoint;

var right = Vec3.Cross(new Vec3(0, 1, 0), forward);

var up = Vec3.Cross(forward, right);

var view = Mat4.CreateLookAt(right, up, forward, cameraPosition);

var scale = Mat4.CreateScale(0.2, 0.5, 0.2);

fullTransform = viewport \* projection \* view \* scale;

}

private void ApplyModelTransformations()

{

var rotationY = Mat4.CreateRotationY(angleY);

circleObject.transformedFaces = circleObject.faces

.Select(face => face

.Select(pt => new Point3D(fullTransform \* rotationY \* pt.Position, pt.NormalVec))

.ToArray())

.OrderBy(face => face.Average(v => v.Position.Z))

.ToList();

pyramidObject.transformedFaces = pyramidObject.faces

.Select(face => face

.Select(pt => new Point3D(fullTransform \* rotationY \* pt.Position, pt.NormalVec))

.ToArray())

.OrderBy(face => face.Average(v => v.Position.Z))

.ToList();

}

private void DrawScene(Graphics g)

{

InitDepthMap();

foreach (var face in circleObject.transformedFaces)

RenderTriangle(face[0], face[1], face[2], g, circleObject.baseColor);

foreach (var face in pyramidObject.transformedFaces)

RenderTriangle(face[0], face[1], face[2], g, pyramidObject.baseColor);

}

private void InitDepthMap()

{

depthMap.Clear();

for (int x = 0; x < Width; x++)

{

var row = new List<double>();

for (int y = 0; y < Height; y++)

row.Add(double.MinValue);

depthMap.Add(row);

}

}

private static Color ComputePhongColor(Vec3 normal, Color baseCol)

{

float rf = baseCol.R / 255f;

float gf = baseCol.G / 255f;

float bf = baseCol.B / 255f;

var lightDir = new Vec3(1, 1, 0).Normalize();

var reflection = new Vec3(0, 0, 0);

var viewDir = lightDir;

var dotNL = Vec3.Dot(normal, lightDir);

if (dotNL >= 0)

reflection = (lightDir - 2 \* dotNL \* normal).Normalize();

float ambient = 1.0f;

float diffuse = (float)Math.Max(Vec3.Dot(normal.Normalize(), lightDir), 0.0) \* 0.5f;

double specular = Math.Pow(Math.Max(0, Vec3.Dot(reflection, lightDir)), 32) \* 0.05;

int r = Clamp((int)(255 \* (ambient \* rf + diffuse + specular)));

int g = Clamp((int)(255 \* (ambient \* gf + diffuse + specular)));

int b = Clamp((int)(255 \* (ambient \* bf + diffuse + specular)));

return Color.FromArgb(r, g, b);

}

private static int Clamp(int val, int min = 0, int max = 255) =>

Math.Min(max, Math.Max(min, val));

private void RenderTriangle(Point3D v0, Point3D v1, Point3D v2, Graphics g, Color col)

{

if (v0.Position.Y < v2.Position.Y) (v0, v2) = (v2, v0);

if (v0.Position.Y < v1.Position.Y) (v0, v1) = (v1, v0);

if (v1.Position.Y < v2.Position.Y) (v1, v2) = (v2, v1);

double x1 = 0, x2 = 0, z1 = 0, z2 = 0;

Vec3 n1 = new Vec3(0, 0, 0), n2 = new Vec3(0, 0, 0);

for (double y = v0.Position.Y; y >= v2.Position.Y; y--)

{

bool upper = y >= v1.Position.Y;

bool lower = y <= v1.Position.Y;

if (upper)

{

double t0 = (v0.Position.Y - y) / (v0.Position.Y - v2.Position.Y);

double t1 = (v0.Position.Y - y) / (v0.Position.Y - v1.Position.Y);

x1 = Lerp(v0.Position.X, v2.Position.X, t0);

x2 = Lerp(v0.Position.X, v1.Position.X, t1);

z1 = Lerp(v0.Position.Z, v2.Position.Z, t0);

z2 = Lerp(v0.Position.Z, v1.Position.Z, t1);

n1 = v0.NormalVec + t0 \* (v2.NormalVec - v0.NormalVec);

n2 = v0.NormalVec + t1 \* (v1.NormalVec - v0.NormalVec);

}

else if (lower)

{

double t0 = (v2.Position.Y - y) / (v2.Position.Y - v0.Position.Y);

double t1 = (v2.Position.Y - y) / (v2.Position.Y - v1.Position.Y);

x1 = Lerp(v2.Position.X, v0.Position.X, t0);

x2 = Lerp(v2.Position.X, v1.Position.X, t1);

z1 = Lerp(v2.Position.Z, v0.Position.Z, t0);

z2 = Lerp(v2.Position.Z, v1.Position.Z, t1);

n1 = v2.NormalVec - t0 \* (v2.NormalVec - v0.NormalVec);

n2 = v2.NormalVec - t1 \* (v2.NormalVec - v1.NormalVec);

}

for (double x = Math.Min(x1, x2); x <= Math.Max(x1, x2); x++)

{

double alpha = (x - x1) / (x2 - x1);

var normal = n1 + alpha \* (n2 - n1);

double z = Lerp(z1, z2, alpha);

if (depthMap[(int)x][(int)y] <= z)

{

depthMap[(int)x][(int)y] = z;

var pixelColor = ComputePhongColor(normal.Normalize(), col);

g.FillRectangle(new SolidBrush(pixelColor), (int)x - 1, (int)y - 1, 2, 2);

}

}

}

}

private static double Lerp(double a, double b, double t) => a + (b - a) \* t;

}

}

Файл Mat4.cs:

using System;

namespace KursovayaKG

{

public class Mat4

{

private readonly double[,] data = new double[4, 4];

private Mat4() { }

public double this[int row, int col]

{

get => data[row, col];

private set => data[row, col] = value;

}

public static Mat4 operator \*(Mat4 a, Mat4 b)

{

var result = new Mat4();

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

for (int k = 0; k < 4; k++)

result[i, j] += a[i, k] \* b[k, j];

return result;

}

public static Mat4 CreateRotationY(double angleDeg)

{

double angleRad = angleDeg \* Math.PI / 180.0;

return new Mat4

{

[0, 0] = Math.Cos(angleRad),

[0, 2] = Math.Sin(angleRad),

[2, 0] = -Math.Sin(angleRad),

[2, 2] = Math.Cos(angleRad),

[1, 1] = 1,

[3, 3] = 1

};

}

public static Mat4 CreateOrthographic(double left, double right, double bottom, double top, double near, double far)

{

return new Mat4

{

[0, 0] = 2 / (right - left),

[1, 1] = 2 / (top - bottom),

[2, 2] = -2 / (far - near),

[0, 3] = -(right + left) / (right - left),

[1, 3] = -(top + bottom) / (top - bottom),

[2, 3] = -(far + near) / (far - near),

[3, 3] = 1

};

}

public static Mat4 CreateViewport(double offsetX, double offsetY, double width, double height, double depth)

{

return new Mat4

{

[0, 0] = width / 2.0,

[1, 1] = -height / 2.0,

[2, 2] = depth / 2.0,

[0, 3] = offsetX + width / 2.0,

[1, 3] = offsetY + height / 2.0,

[2, 3] = depth / 2.0,

[3, 3] = 1

};

}

public static Mat4 CreateLookAt(Vec3 right, Vec3 up, Vec3 back, Vec3 position)

{

return new Mat4

{

[0, 0] = right.X,

[0, 1] = right.Y,

[0, 2] = right.Z,

[1, 0] = up.X,

[1, 1] = up.Y,

[1, 2] = up.Z,

[2, 0] = back.X,

[2, 1] = back.Y,

[2, 2] = back.Z,

[0, 3] = -position.X,

[1, 3] = -position.Y,

[2, 3] = -position.Z,

[3, 3] = 1

};

}

public static Mat4 CreateScale(double scaleX, double scaleY, double scaleZ)

{

return new Mat4

{

[0, 0] = scaleX,

[1, 1] = scaleY,

[2, 2] = scaleZ,

[3, 3] = 1

};

}

}

}

Файл Vec3.cs:

using System;

namespace KursovayaKG

{

public class Vec3

{

private readonly double[] components = { 0, 0, 0, 1 };

public double X { get => components[0]; private set => components[0] = value; }

public double Y { get => components[1]; private set => components[1] = value; }

public double Z { get => components[2]; private set => components[2] = value; }

private double Length { get; }

public Vec3(double x, double y, double z)

{

X = x;

Y = y;

Z = z;

Length = Math.Sqrt(x \* x + y \* y + z \* z);

}

private Vec3() { }

private double this[int index]

{

get => components[index];

set => components[index] = value;

}

public static Vec3 operator -(Vec3 a, Vec3 b) =>

new Vec3(a.X - b.X, a.Y - b.Y, a.Z - b.Z);

public static Vec3 operator +(Vec3 a, Vec3 b) =>

new Vec3(a.X + b.X, a.Y + b.Y, a.Z + b.Z);

public static Vec3 operator \*(double scalar, Vec3 v) =>

new Vec3(v.X \* scalar, v.Y \* scalar, v.Z \* scalar);

public static Vec3 operator /(Vec3 v, double scalar) =>

new Vec3(v.X / scalar, v.Y / scalar, v.Z / scalar);

public static double Dot(Vec3 a, Vec3 b) =>

a.X \* b.X + a.Y \* b.Y + a.Z \* b.Z;

public static Vec3 Cross(Vec3 a, Vec3 b) =>

new Vec3(

a.Y \* b.Z - a.Z \* b.Y,

a.Z \* b.X - a.X \* b.Z,

a.X \* b.Y - a.Y \* b.X

);

public static Vec3 operator \*(Mat4 matrix, Vec3 v)

{

var result = new Vec3();

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

result[i] += matrix[i, j] \* v[j];

return result;

}

public Vec3 Normalize() =>

new Vec3(X / Length, Y / Length, Z / Length);

}

}

Файл Point3D.cs:

namespace KursovayaKG

{

public class Point3D

{

public Vec3 Position;

public Vec3 NormalVec;

public Point3D(Vec3 pos, Vec3 normal)

{

Position = pos;

NormalVec = normal;

}

}

}

Файл Program.cs:

using System;

using System.Windows.Forms;

namespace KursovayaKG

{

static class Program

{

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new ViewerForm());

}

}

}

ВЫВОД

В ходе курсовой работы была разработана программа для визуализации трёхмерной сцены на языке C# с использованием Windows Forms. Реализована загрузка 3D-модели из файла .obj, применение матричных преобразований, Z-буфера и освещения по модели Фонга. Программа позволяет интерактивно вращать сцену и корректно отображает глубину и освещённость объектов. Цели работы достигнуты, поставленные задачи выполнены.