МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ



Кафедра СМАРТ технологий

Лабораторная работа № 3

«Аппаратная поддержка отображения пространственных данных»

По дисциплине «Технологии визуализации данных систем управления»

Группа: 221-327

Студент: Шурова Д. С.

Дата: 13.05.2025

Преподаватель: Идиатуллов Т. Т.

Москва, 2025

**Задача**

* Реализовать загрузку набора данных, заданных как тройки чисел (X, Y, Z) из файлов формата CSV (разделитель – точка с запятой) и генерацию заданного (через текстовое поле) количества случайных точек, где X,Y и Z – равномерно распределенные случайные величины на диапазоне [ -1 ÷ 1].
* Выполнить статистический анализ набора точек, выполнив построение частотной диаграммы (гистограммы) 10х10 ячеек в координатной плоскости XY.
* Выполнить статистический анализ набора точек, выполнив расчет плотности заполнения вокселей (voxel) как пространственной матрицы 10х10х10 вокселей (пространственных ячеек).
* Разработать систему отображения данных в виде облака точек средствами OpenGL (SharpGL) с указанием степени прореживания (отображать с шагом по номеру) и функцией анимации со сдвигом по номеру отображения.
* Разработать систему отображения данных в виде пространства вокселей, значение плотности заполнения каждого из которых отображать размером кубика, помещенного в центр вокселя.
* Разработать систему отображения гистограммы распределения точек в виде поверхности, где координаты вычисляются следующим образом: координаты X и Y соответствуют координатам ячеек (центрам) гистограммы, а координата Z вычисляется как доля частоты попадания точек в данную ячейку по отношению к самому большому значению среди всех ячеек.
* Реализовать систему поворота базовой системы координат при отображении данных с помощью матрицы трансляции-поворота, управляемого позиционным манипулятором (мышью).
* Реализовать отображение системы координат и ребер описывающего куба (стороны: -1 и 1 по каждой координате)
* Реализовать сохранение полученного изображения в файл.

**Ход работы**

Метод buttonLoad\_Click реализует загрузку точек из CSV файла, содержащего строки формата (X;Y;Z). Метод btnGenerateParametric\_Click генерирует набор точек: pointCloud – равномерное распределённые по X, Y, Z в диапазоне [-1, 1].

**Гистограмма:** Построена частотная диаграмма 10x10 в плоскости XY. Каждая ячейка окрашена в зависимости от количества точек (чем больше точек, тем темнее цвет). (Рисунок 1)

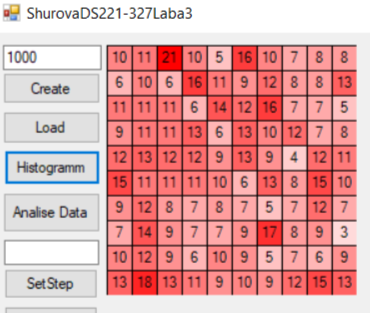


Рисунок 1 – Гистограмма

**Воксели:** Рассчитана плотность заполнения для матрицы 10x10x10. Плотность отображается через размер кубиков в центре каждого вокселя. (Рисунок 2)

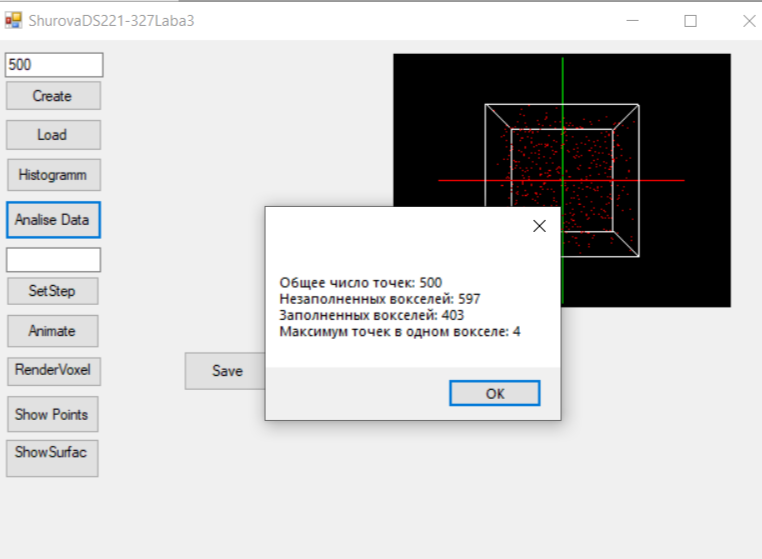


Рисунок 2 – Анализ данных

**Облако точек:** Реализовано отображение точек с возможностью прореживания (шаг задается пользователем) и анимации (постепенное смещение отображаемых точек). (Рисунок 3)

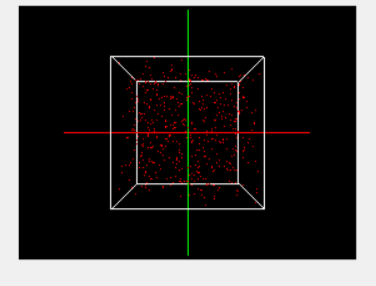


Рисунок 3 – Отображение облака точек

**Воксели:** Кубики отображаются с размером, пропорциональным плотности точек в соответствующем вокселе. (Рисунок 4)

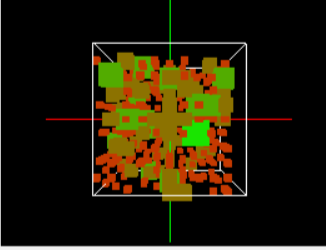


Рисунок 4 - Воксели

**Гистограмма в 3D:** Поверхность построена так, что координаты X и Y соответствуют центрам ячеек гистограммы, а Z — нормированной частоте попадания точек. Гистограмма обновляется каждый раз при анимации. (Рисунок 5)

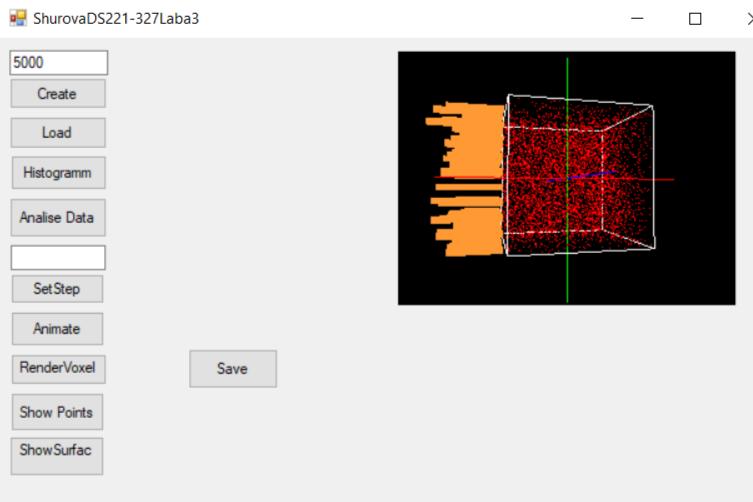


Рисунок 5 – Гистограмма 3D

Вращение системы координат мышью (Drag&Drop) реализовано через события и метод пересчета матрицы. Определяется смещение мыши, вычисляются углы поворота вокруг осей X и Y. Формируются матрицы поворота для X и Y.

Реализовано Отображение базисных векторов в методах отображения проекция. Цвета: Красный (X), зеленый (Y), синий (Z). Поворот осей синхронизируется с вращением точек через матрицы.

Происходит сохранение bitmap в файл (путь указан в коде), как картинки png.

**Вывод**

В ходе работы было разработано приложение для визуализации пространственных данных с использованием библиотеки SharpGL. Приложение демонстрирует эффективное использование аппаратной поддержки для визуализации данных и предоставляет удобные инструменты для анализа.

**Приложение А**

Листинг 1 – Код программы

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using SharpGL;

using SharpGL.WinForms;

using System.IO;

using System.Windows.Forms.DataVisualization.Charting;

using System.Drawing.Imaging;

namespace Laba3

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private List<(double X, double Y, double Z)> pointCloud = new List<(double X, double Y, double Z)>();

private int thinningStep = 1;

private int displayOffset = 0;

private Timer animationTimer;

private void Form1\_Load(object sender, EventArgs e)

{

animationTimer = new Timer();

animationTimer.Interval = 100; // скорость анимации в мс

animationTimer.Tick += AnimationTimer\_Tick;

}

private void AnimationTimer\_Tick(object sender, EventArgs e)

{

displayOffset += 1;

if (displayOffset >= thinningStep) displayOffset = 0;

openGLControl1.Invalidate(); // перерисовать

}

private void openGLControl1\_OpenGLDraw(object sender, SharpGL.RenderEventArgs args)

{

var gl = openGLControl1.OpenGL;

// Очистка экрана

gl.Clear(OpenGL.GL\_COLOR\_BUFFER\_BIT | OpenGL.GL\_DEPTH\_BUFFER\_BIT);

gl.LoadIdentity();

// Трансляция назад (камера отдаляется)

gl.Translate(0.0f, 0.0f, -5.0f); // или -zoom, если у тебя есть переменная приближения

// Поворот базовой системы координат

gl.Rotate(rotationX, 1.0f, 0.0f, 0.0f); // вокруг X

gl.Rotate(rotationY, 0.0f, 1.0f, 0.0f); // вокруг Y

RenderAxesAndBoundingCube(gl);

// Отрисовка нужного режима

if (showVoxels)

{

RenderVoxels(gl);

}

else

{

if (showHistogramSurface)

{

RenderHistogramBars(gl);

}

gl.Begin(OpenGL.GL\_POINTS);

gl.Color(1.0f, 0.0f, 0.0f);

for (int i = displayOffset; i < pointCloud.Count; i += thinningStep)

{

var (x, y, z) = pointCloud[i];

gl.Vertex(x, y, z);

}

gl.End();

}

gl.Flush();

}

private void btnSetStep\_Click(object sender, EventArgs e)

{

if (int.TryParse(txtStep.Text, out int step) && step > 0)

{

thinningStep = step;

displayOffset = 0;

openGLControl1.Invalidate();

}

else

{

MessageBox.Show("Введите положительное число.");

}

}

private void btnAnimate\_Click(object sender, EventArgs e)

{

if (animationTimer.Enabled)

{

animationTimer.Stop();

}

else

{

animationTimer.Start();

}

}

private void loadBtn\_Click(object sender, EventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog

{

Filter = "CSV files (\*.csv)|\*.csv"

};

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

var points = new List<(double X, double Y, double Z)>();

foreach (var line in File.ReadAllLines(openFileDialog.FileName))

{

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

if (parts.Length == 3 &&

double.TryParse(parts[0], out double x) &&

double.TryParse(parts[1], out double y) &&

double.TryParse(parts[2], out double z))

{

points.Add((x, y, z));

}

}

pointCloud = points;

}

}

private void createBtn\_Click(object sender, EventArgs e)

{

if (!int.TryParse(txtCount.Text, out int N) || N <= 0)

{

MessageBox.Show("Введите корректное положительное число точек.");

return;

}

pointCloud.Clear();

Random rand = new Random();

// Первый список: равномерно распределенные точки

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

{

double x = rand.NextDouble() \* 2 - 1;

double y = rand.NextDouble() \* 2 - 1;

double z = rand.NextDouble() \* 2 - 1;

pointCloud.Add((x, y, z));

}

}

private void histBtn\_Click(object sender, EventArgs e)

{

int[,] histogram = new int[10, 10];

int maxCount = 1; // нужно для нормализации цветов

// Подсчёт количества точек

foreach (var point in pointCloud)

{

int binX = (int)((point.X + 1) / 2 \* 10);

int binY = (int)((point.Y + 1) / 2 \* 10);

if (binX == 10) binX = 9;

if (binY == 10) binY = 9;

histogram[binX, binY]++;

if (histogram[binX, binY] > maxCount)

maxCount = histogram[binX, binY];

}

// Рисуем картинку

int cellSize = 20;

Bitmap bmp = new Bitmap(10 \* cellSize, 10 \* cellSize);

using (Graphics g = Graphics.FromImage(bmp))

{

g.Clear(Color.White);

for (int x = 0; x < 10; x++)

{

for (int y = 0; y < 10; y++)

{

int count = histogram[x, y];

// Чем выше count — тем темнее цвет

int intensity = (int)(255 - (count / (double)maxCount) \* 255);

Color color = Color.FromArgb(255, intensity, intensity); // красный оттенок

Rectangle rect = new Rectangle(x \* cellSize, (9 - y) \* cellSize, cellSize, cellSize);

using (Brush b = new SolidBrush(color))

g.FillRectangle(b, rect);

g.DrawRectangle(Pens.Black, rect);

if (count > 0)

{

var sf = new StringFormat { Alignment = StringAlignment.Center, LineAlignment = StringAlignment.Center };

g.DrawString(count.ToString(), this.Font, Brushes.Black, rect, sf);

}

}

}

}

pictureBox1.Image = bmp;

}

private int[,,] ComputeVoxelDensity()

{

int[,,] voxelGrid = new int[10, 10, 10];

foreach (var point in pointCloud)

{

int i = (int)((point.X + 1) / 2 \* 10);

int j = (int)((point.Y + 1) / 2 \* 10);

int k = (int)((point.Z + 1) / 2 \* 10);

if (i == 10) i = 9;

if (j == 10) j = 9;

if (k == 10) k = 9;

voxelGrid[i, j, k]++;

}

return voxelGrid;

}

private void analiseDataBttn\_Click(object sender, EventArgs e)

{

var voxels = ComputeVoxelDensity();

int nonEmpty = 0;

int maxCount = 0;

int total = 0;

for (int x = 0; x < 10; x++)

{

for (int y = 0; y < 10; y++)

{

for (int z = 0; z < 10; z++)

{

int count = voxels[x, y, z];

if (count > 0)

{

nonEmpty++;

total += count;

if (count > maxCount)

maxCount = count;

}

}

}

}

MessageBox.Show($"Общее число точек: {pointCloud.Count}\n" +

$"Незаполненных вокселей: {1000 - nonEmpty}\n" +

$"Заполненных вокселей: {nonEmpty}\n" +

$"Максимум точек в одном вокселе: {maxCount}");

}

// 1. Массив для хранения плотности вокселей

private int[,,] voxelGrid = new int[10, 10, 10]; // Сетка 10x10x10 вокселей

// 2. Функция для подсчета плотности точек в вокселях

private void ComputeVoxelDensity(List<(double X, double Y, double Z)> pointCloud)

{

foreach (var point in pointCloud)

{

// Нормализуем координаты, чтобы они попадали в диапазон от 0 до 9 для вокселей

int i = (int)((point.X + 1) / 2 \* 10);

int j = (int)((point.Y + 1) / 2 \* 10);

int k = (int)((point.Z + 1) / 2 \* 10);

// Обеспечиваем, чтобы индексы не выходили за пределы

if (i == 10) i = 9;

if (j == 10) j = 9;

if (k == 10) k = 9;

// Увеличиваем счетчик точек в этом вокселе

voxelGrid[i, j, k]++;

}

}

// 3. Функция для отрисовки вокселей с кубиками

private void RenderVoxels(SharpGL.OpenGL gl)

{

float maxDensity = voxelGrid.Cast<int>().Max(); // Максимальная плотность для нормализации

// Отображаем все воксели

for (int x = 0; x < 10; x++)

{

for (int y = 0; y < 10; y++)

{

for (int z = 0; z < 10; z++)

{

int density = voxelGrid[x, y, z];

if (density > 0)

{

float size = (density / (float)maxDensity) \* 0.4f; // Нормализованный размер кубика

// Вычисляем позицию вокселя

float voxelX = (x / 10.0f) \* 2 - 1; // Преобразуем воксель в диапазон от -1 до 1

float voxelY = (y / 10.0f) \* 2 - 1;

float voxelZ = (z / 10.0f) \* 2 - 1;

// Отображаем куб в центре вокселя с нормализованным размером

gl.PushMatrix();

gl.Translate(voxelX, voxelY, voxelZ);

gl.Scale(size, size, size); // Масштабируем кубик по размеру

// Отрисовываем куб

gl.Begin(OpenGL.GL\_QUADS);

// Куб из 6 граней

for (int i = 0; i < 6; i++)

{

gl.Color(1.0f - (density / (float)maxDensity\*0.9), density / (float)maxDensity\*0.9, 0.0f); // Цвет зависит от плотности

DrawCubeFace(i);

}

gl.End();

gl.PopMatrix();

}

}

}

}

}

private void DrawCubeFace(int faceIndex)

{

var gl = openGLControl1.OpenGL;

float[][] vertices = new float[][]

{

new float[] { -0.5f, -0.5f, -0.5f },

new float[] { 0.5f, -0.5f, -0.5f },

new float[] { 0.5f, 0.5f, -0.5f },

new float[] { -0.5f, 0.5f, -0.5f },

new float[] { -0.5f, -0.5f, 0.5f },

new float[] { 0.5f, -0.5f, 0.5f },

new float[] { 0.5f, 0.5f, 0.5f },

new float[] { -0.5f, 0.5f, 0.5f },

};

int[][] faces = new int[][]

{

new int[] {0, 1, 2, 3}, // back

new int[] {1, 5, 6, 2}, // right

new int[] {5, 4, 7, 6}, // front

new int[] {4, 0, 3, 7}, // left

new int[] {3, 2, 6, 7}, // top

new int[] {4, 5, 1, 0}, // bottom

};

int[] face = faces[faceIndex];

foreach (var idx in face)

{

gl.Vertex(vertices[idx]);

}

}

private bool showVoxels = false;

private void buttonRenderVoxels\_Click(object sender, EventArgs e)

{

Array.Clear(voxelGrid, 0, voxelGrid.Length);

ComputeVoxelDensity(pointCloud);

showVoxels = true; // переключиться на режим отображения вокселей

openGLControl1.Invalidate();

}

private void buttonShowPoints\_Click(object sender, EventArgs e)

{

showVoxels = false;

showHistogramSurface = false;

openGLControl1.Invalidate();

}

private bool showHistogramSurface = false;

private void btnShowSurface\_Click(object sender, EventArgs e)

{

showHistogramSurface = true;

showVoxels = false;

openGLControl1.Invalidate();

}

private void RenderHistogramBars(OpenGL gl)

{

// Считаем 2D-гистограмму по осям Z и Y (на плоскости YZ)

int[,] histogram = new int[10, 10];

int maxCount = 1;

for (int i = displayOffset; i < pointCloud.Count; i += thinningStep)

{

var (x, y, z) = pointCloud[i];

int binY = (int)((y + 1) / 2 \* 10);

int binZ = (int)((z + 1) / 2 \* 10);

if (binY == 10) binY = 9;

if (binZ == 10) binZ = 9;

histogram[binY, binZ]++;

if (histogram[binY, binZ] > maxCount)

maxCount = histogram[binY, binZ];

}

// Рисуем столбики вдоль оси X, начиная с X = -1 (левая грань)

for (int y = 0; y < 10; y++)

{

for (int z = 0; z < 10; z++)

{

int count = histogram[y, z];

if (count == 0) continue;

float length = (count / (float)maxCount) \* 1.0f; // длина по X

// Координаты в диапазоне [-1, 1]

float posY = (y / 10.0f) \* 2 - 0.95f;

float posZ = (z / 10.0f) \* 2 - 0.95f;

float posX = -1.05f - length / 2f;

gl.PushMatrix();

gl.Translate(posX, posY, posZ);

gl.Scale(length, 0.08f, 0.08f); // длина по X, остальные — тонкие

gl.Color(1.0f, 0.6f, 0.2f); // оранжевые столбики

gl.Begin(OpenGL.GL\_QUADS);

for (int i = 0; i < 6; i++)

{

DrawCubeFace(i);

}

gl.End();

gl.PopMatrix();

}

}

}

private float rotationX = 0.0f;

private float rotationY = 0.0f;

private Point lastMousePosition;

private bool isRotating = false;

private void openGLControl\_MouseDown(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

isRotating = true;

lastMousePosition = e.Location;

}

}

private void openGLControl\_MouseMove(object sender, MouseEventArgs e)

{

if (isRotating)

{

int deltaX = e.X - lastMousePosition.X;

int deltaY = e.Y - lastMousePosition.Y;

rotationX += deltaY;

rotationY += deltaX;

lastMousePosition = e.Location;

openGLControl1.Invalidate(); // перерисовать

}

}

private void openGLControl\_MouseUp(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

isRotating = false;

}

private void RenderAxesAndBoundingCube(OpenGL gl)

{

// Оси координат

gl.Begin(OpenGL.GL\_LINES);

// Ось X — красная

gl.Color(1.0, 0.0, 0.0);

gl.Vertex(-2.0, 0.0, 0.0);

gl.Vertex(2.0, 0.0, 0.0);

// Ось Y — зелёная

gl.Color(0.0, 1.0, 0.0);

gl.Vertex(0.0, -2.0, 0.0);

gl.Vertex(0.0, 2.0, 0.0);

// Ось Z — синяя

gl.Color(0.0, 0.0, 1.0);

gl.Vertex(0.0, 0.0, -2.0);

gl.Vertex(0.0, 0.0, 2.0);

gl.End();

// Куб: грани от -1 до 1 по каждой оси

gl.Color(1.0, 1.0, 1.0); // белый

gl.Begin(OpenGL.GL\_LINES);

float[] x = { -1, 1 };

float[] y = { -1, 1 };

float[] z = { -1, 1 };

// 12 рёбер куба

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

{

// x-ребра

gl.Vertex(x[0], y[i], z[j]);

gl.Vertex(x[1], y[i], z[j]);

// y-ребра

gl.Vertex(x[i], y[0], z[j]);

gl.Vertex(x[i], y[1], z[j]);

// z-ребра

gl.Vertex(x[i], y[j], z[0]);

gl.Vertex(x[i], y[j], z[1]);

}

gl.End();

}

private void SaveOpenGLToFile(string filePath)

{

var gl = openGLControl1.OpenGL;

int width = openGLControl1.Width;

int height = openGLControl1.Height;

byte[] pixels = new byte[width \* height \* 4]; // RGBA

// Считываем пиксели с буфера кадра

gl.ReadPixels(0, 0, width, height, OpenGL.GL\_RGBA, OpenGL.GL\_UNSIGNED\_BYTE, pixels);

// Создаём Bitmap и копируем данные пикселей

using (Bitmap bmp = new Bitmap(width, height, PixelFormat.Format32bppArgb))

{

BitmapData data = bmp.LockBits(new Rectangle(0, 0, width, height), ImageLockMode.WriteOnly, bmp.PixelFormat);

System.Runtime.InteropServices.Marshal.Copy(pixels, 0, data.Scan0, pixels.Length);

bmp.UnlockBits(data);

// OpenGL читает пиксели снизу вверх, переворачиваем

bmp.RotateFlip(RotateFlipType.RotateNoneFlipY);

// Сохраняем в файл

bmp.Save(filePath, ImageFormat.Png);

}

}

private void btnSave\_Click(object sender, EventArgs e)

{

using (SaveFileDialog sfd = new SaveFileDialog())

{

sfd.Filter = "PNG Image|\*.png";

if (sfd.ShowDialog() == DialogResult.OK)

{

SaveOpenGLToFile(sfd.FileName);

MessageBox.Show("Изображение сохранено!");

}

}

}

}

}