Министерство образования Республики Беларусь

Министерство образования и науки Российской Федерации

МУВПО “Белорусско-Российский университет”

Кафедра “ПОИТ”

Отчет по

Лабораторной работе №10

Модели отражения света

Выполнил: ст. гр. АСОИ-181

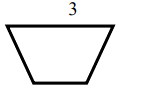
Остапенко А. К.

Преподаватель

Шилов А. В.

Могилёв 2020

Составить алгоритм и программу для изображения поверхности согласно интенсивности отраженного света при учете взаимного расположения поверхности, источника света и наблюдателя при пересечении объекта № 3 и куба.



public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

int zoom = 1;

Bitmap bmp = new Bitmap(pictureBox1.Width \* zoom, pictureBox1.Height \* zoom);

int lengthX = 180 \* zoom;

int lengthY = 220 \* zoom;

int height = 60 \* zoom;

int cubeLength = 150 \* zoom;

Point3D a = new Point3D(100 \* zoom, 0 \* zoom, 130 \* zoom);

Point3D a1 = new Point3D(a.X, a.Y, a.Z + height);

Point3D b = new Point3D(a.X + lengthX, a.Y, a.Z);

Point3D b1 = new Point3D(b.X, b.Y, b.Z + height);

Point3D b2 = new Point3D(b1.X, b1.Y + lengthY / 2, b1.Z);

Point3D d = new Point3D(a.X, a.Y + lengthY, a.Z);

Point3D d1 = new Point3D(d.X, d.Y, d.Z + height);

Point3D d2 = new Point3D(d1.X + lengthX / 3, d1.Y, d1.Z);

Point3D d3 = new Point3D(d1.X + lengthX / 3, d1.Y, d1.Z - height);

Point3D b3 = new Point3D(b1.X, b1.Y + lengthY / 2, b1.Z - height);

Polygon3D[] polygons = new Polygon3D[]

{

new Polygon3D(new Point3D[] { a1,b1,b2,d2,d1 }){Color = Color.Red},

new Polygon3D(new Point3D[] { a,b,b3,d3,d }){Color = Color.Red},

new Polygon3D(new Point3D[] { a,b,b1,a1 }){Color = Color.Red},

new Polygon3D(new Point3D[] { b,b1,b2,b3}){Color = Color.Red},

new Polygon3D(new Point3D[] { b3,b2,d2,d3}){Color = Color.Red},

new Polygon3D(new Point3D[] { d2,d3,d,d1}){Color = Color.Red},

new Polygon3D(new Point3D[] { d1,d,a,a1}){Color = Color.Red},

};

Polyhedron polyhedron = new Polyhedron(polygons, a, b, b3, d3, d, a1, b1, b2, d2, d1);

#region повороты

var center2D = new Point2D(polyhedron.Center.X, polyhedron.Center.Y);

double rotatingAngle = -Math.PI / 4;

foreach (var point in polyhedron.Vertexes)

{

var point2d = Grafic.Rotation(center2D, new Point2D(point.X, point.Y), rotatingAngle);

point.X = point2d.X;

point.Y = point2d.Y;

}

rotatingAngle = Math.PI / 4;

center2D = new Point2D(polyhedron.Center.Y, polyhedron.Center.Z);

foreach (var point in polyhedron.Vertexes)

{

var point2d = Grafic.Rotation(center2D, new Point2D(point.Y, point.Z), rotatingAngle);

point.Y = point2d.X;

point.Z = point2d.Y;

}

#endregion

List<Polygon3D> list = new List<Polygon3D>();

list.AddRange(polyhedron.Faces);

list.AddRange(GetCube(polyhedron.Center, cubeLength).Faces);

DrawPolyhedronByZBufferMetod(bmp, list.ToArray(), new Point3D(0, -1, 0), polyhedron.Center);

bmp.SetPixel(0, bmp.Height - 1, Color.Black);

pictureBox1.Image = bmp;

}

private Polyhedron GetCube(Point3D center, int length)

{

Point3D a = new Point3D(center.X - length / 2, center.Y - length / 2, center.Z - length / 2);

Point3D a1 = new Point3D(center.X - length / 2, center.Y - length / 2, center.Z + length / 2);

Point3D b = new Point3D(center.X + length / 2, center.Y - length / 2, center.Z - length / 2);

Point3D b1 = new Point3D(center.X + length / 2, center.Y - length / 2, center.Z + length / 2);

Point3D c = new Point3D(center.X + length / 2, center.Y + length / 2, center.Z - length / 2);

Point3D c1 = new Point3D(center.X + length / 2, center.Y + length / 2, center.Z + length / 2);

Point3D d = new Point3D(center.X - length / 2, center.Y + length / 2, center.Z - length / 2);

Point3D d1 = new Point3D(center.X - length / 2, center.Y + length / 2, center.Z + length / 2);

Polygon3D[] polygons = new Polygon3D[]

{

new Polygon3D(new Point3D[] { a,d,d1,a1 }){Color = Color.Green},

new Polygon3D(new Point3D[] { a1,b1,c1,d1 }){Color = Color.Green},

new Polygon3D(new Point3D[] { a,b,c,d }){Color = Color.Green},

new Polygon3D(new Point3D[] { a,b,b1,a1 }){Color = Color.Green},

new Polygon3D(new Point3D[] { b,c,c1,b1 }){Color = Color.Green},

new Polygon3D(new Point3D[] { c,d,d1,c1 }){Color = Color.Green},

};

Polyhedron polyhedron = new Polyhedron(polygons, a, b, c, d, a1, b1, c1, d1);

var center2D = new Point2D(polyhedron.Center.X, polyhedron.Center.Y);

double rotatingAngle = -Math.PI / 4;

foreach (var point in polyhedron.Vertexes)

{

var point2d = Grafic.Rotation(center2D, new Point2D(point.X, point.Y), rotatingAngle);

point.X = point2d.X;

point.Y = point2d.Y;

}

rotatingAngle = Math.PI / 4;

center2D = new Point2D(polyhedron.Center.Y, polyhedron.Center.Z);

foreach (var point in polyhedron.Vertexes)

{

var point2d = Grafic.Rotation(center2D, new Point2D(point.Y, point.Z), rotatingAngle);

point.Y = point2d.X;

point.Z = point2d.Y;

}

return polyhedron;

}

public static void DrawPolyhedronByZBufferMetod(Bitmap bmp, Polygon3D[] polygons, Point3D LightPoint, Point3D center)

{

double[,] ZBuffer = new double[bmp.Width, bmp.Height];

for (int y = 0; y < ZBuffer.GetLength(1) - 1; y++)

for (int x = 0; x < ZBuffer.GetLength(0) - 1; x++)

{

ZBuffer[x, y] = double.NegativeInfinity;

}

for (int polygon = 0; polygon < polygons.Length; polygon++)

{

Point3D[] pointsBuf = new Point3D[polygons[polygon].Points.Length + 1];

polygons[polygon].Points.CopyTo(pointsBuf, 0);

pointsBuf[pointsBuf.Length - 1] = pointsBuf[0];

List<Line> CAP = new List<Line>();

int ymin = (int)pointsBuf.Min(x => x.Z);

int ymax = (int)pointsBuf.Max(x => x.Z);

double[] coef = polygons[polygon].CalculateCoefficients(new Point3D(0, 0, 0));

double[] coef1 = polygons[polygon].CalculateCoefficients(center);

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

{

coef1[i] \*= Math.Sign(coef1[3]);

}

double L = GetCos(new Point3D(coef1[0], coef1[1], coef1[2]), LightPoint);

int R = (int)(polygons[polygon].Color.R \* L);

R = R > 0 && R < 255 ? R : R > 0 ? 255 : 0;

int G = (int)(polygons[polygon].Color.G \* L);

G = G > 0 && G < 255 ? G : G > 0 ? 255 : 0;

int B = (int)(polygons[polygon].Color.B \* L);

B = B > 0 && B < 255 ? B : B > 0 ? 255 : 0;

Color color = Color.FromArgb(R, G, B);

for (int y = ymin + 1; y < ymax; y++)

{

CAP.Clear();

for (int i = 1; i < pointsBuf.Length; i++)

{

int a = (int)Math.Min(pointsBuf[i].Z, pointsBuf[i - 1].Z);

int b = (int)Math.Max(pointsBuf[i].Z, pointsBuf[i - 1].Z);

if (a < y && b >= y && a != b)

{

CAP.Add(new Line(new Point3D(pointsBuf[i].X, pointsBuf[i].Z, pointsBuf[i].Y), new Point3D(pointsBuf[i - 1].X, pointsBuf[i - 1].Z, pointsBuf[i - 1].Y)));

}

}

int xmin = Math.Min(CalculateX(CAP[0].Point1, CAP[0].Point2, y), CalculateX(CAP[1].Point1, CAP[1].Point2, y));

int xmax = Math.Max(CalculateX(CAP[0].Point1, CAP[0].Point2, y), CalculateX(CAP[1].Point1, CAP[1].Point2, y));

for (int x = xmin; x < xmax; x++)

{

double z = 0;

z = (-(coef[0] \* x + coef[2] \* y + coef[3]) / coef[1]);

if (x > 0 && y > 0 && x < bmp.Width && y < bmp.Height)

if (z > ZBuffer[x, bmp.Height - y])

{

ZBuffer[x, bmp.Height - y] = z;

bmp.SetPixel(x, bmp.Height - y, color);

}

}

}

}

}

private static int CalculateX(Point2D p1, Point2D p2, double y)

{

double k = (p2.X - p1.X) / (p2.Y - p1.Y);

int res = (int)(p1.X + (y - p1.Y) \* k);

return res;

}

private static double GetCos(Point3D p1, Point3D p2)

{

double p1Length = Math.Pow((Math.Pow(p1.X, 2) + Math.Pow(p1.Y, 2) + Math.Pow(p1.Z, 2)), 0.5);

double p2Length = Math.Pow((Math.Pow(p2.X, 2) + Math.Pow(p2.Y, 2) + Math.Pow(p2.Z, 2)), 0.5);

double cos = (p1.X \* p2.X + p1.Y \* p2.Y + p1.Z \* p2.Z) / (p1Length \* p2Length);

return cos;

}

}

