Министерство образования Республики Беларусь

Министерство образования и науки Российской Федерации

МУВПО “Белорусско-Российский университет”

Кафедра “ПОИТ”

Отчет по

Лабораторной работе №11

Модель отражения света Гуро

Выполнил: ст. гр. АСОИ-181

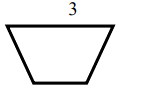
Остапенко А. К.

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 объекта № 3 с использованием алгоритма Гуро.



public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

bool IsRotating = true;

double zoom = 1;

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

int lengthX = (int)(180 \* zoom);

int lengthY = (int)(220 \* zoom);

int height = (int)(60 \* zoom);

int cubeLength = (int)(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.Orange},

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

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

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

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

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

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

};

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

Task.Factory.StartNew(() =>

{

do

{

bmp = new Bitmap(bmp.Width,bmp.Height);

DrawPolyhedronByGuro(bmp, polyhedron, new Vector(0, -1, 0));

pictureBox1.Image = bmp;

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;

}

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

rotatingAngle = Math.PI / 18;

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;

}

} while (IsRotating);

});

}

private void DrawPolyhedronByGuro(Bitmap bmp, Polyhedron polyhedron, Vector lightPoint)

{

polyhedron.ResetBodyMatrix(polyhedron.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 < polyhedron.Faces.Length; polygon++)

{

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

double[] coef1 = polyhedron.Faces[polygon].CalculateCoefficients(polyhedron.Center);

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

{

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

}

#region

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

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

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

#region

double[] IforPoints = new double[pointsBuf.Length];

for (int i = 0; i < IforPoints.Length - 1; i++)

{

var polygons = polyhedron.Faces.ToList().FindAll(x => x.Points.Contains(pointsBuf[i])); // грани, которым пренадлежит точка pointsBuf[i]

Vector normal = new Vector();

for (int k = 0; k < polygons.Count; k++)

{

int index = polyhedron.Faces.ToList().IndexOf(polygons[k]);

normal.X += polyhedron.BodyMatrix[0, index];

normal.Y += polyhedron.BodyMatrix[1, index];

normal.Z += polyhedron.BodyMatrix[2, index];

}

normal.X /= polygons.Count;

normal.Y /= polygons.Count;

normal.Z /= polygons.Count;

IforPoints[i] = GetCos(normal, lightPoint);

}

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

#endregion

List<Tuple<Point3D, Point3D, double, double>> CAP = new List<Tuple<Point3D, Point3D, double, double>>();

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

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

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 Tuple<Point3D, Point3D, double, double>(pointsBuf[i], pointsBuf[i - 1], IforPoints[i], IforPoints[i - 1]));

}

}

int xmin = Math.Min(CalculateX(CAP[0].Item1, CAP[0].Item2, y), CalculateX(CAP[1].Item1, CAP[1].Item2, y));

int xmax = Math.Max(CalculateX(CAP[0].Item1, CAP[0].Item2, y), CalculateX(CAP[1].Item1, CAP[1].Item2, y));

double Ya;

double Ia;

double Yb;

double Yc;

double Yd;

double Ib;

double Ic;

double Id;

Tuple<Point3D, Point3D, double, double> left = CAP[1];

Tuple<Point3D, Point3D, double, double> right = CAP[0];

if (CalculateX(CAP[0].Item1, CAP[0].Item2, y) < CalculateX(CAP[1].Item1, CAP[1].Item2, y))

{

left = CAP[0];

right = CAP[1];

}

if (right.Item1.Z > right.Item2.Z)

{

Ya = right.Item1.Z;

Ia = right.Item3;

Yb = right.Item2.Z;

Ib = right.Item4;

}

else

{

Ya = right.Item2.Z;

Ia = right.Item4;

Yb = right.Item1.Z;

Ib = right.Item3;

}

if (left.Item1.Z > left.Item2.Z)

{

Yd = left.Item1.Z;

Id = left.Item3;

Yc = left.Item2.Z;

Ic = left.Item4;

}

else

{

Yd = left.Item2.Z;

Id = left.Item4;

Yc = left.Item1.Z;

Ic = left.Item3;

}

double I1 = Id + (Ic - Id) \* (y - Yd) / (Yc - Yd);

double I2 = Ia + (Ib - Ia) \* (y - Ya) / (Yb - Ya);

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

{

double I = I1 + (I2 - I1) \* (x - xmin) / (xmax - xmin);

int R = (int)(polyhedron.Faces[polygon].Color.R \* I);

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

int G = (int)(polyhedron.Faces[polygon].Color.G \* I);

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

int B = (int)(polyhedron.Faces[polygon].Color.B \* I);

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

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

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);

}

}

}

#endregion

}

}

private static int CalculateX(Point3D p1, Point3D p2, int y)

{

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

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

return res;

}

private double GetCos(Vector v1, Vector v2)

{

return Vector.ScalarMultiplying(v1, v2) / (v1.Length \* v2.Length);

}

private void pictureBox1\_Click(object sender, EventArgs e)

{

}

