**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 3**

Тема: Основы построения фотореалистичных изображений

Студент: Шиляева Н. С.

Группа: 80-304

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Используя результаты Л.Р.№2, аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света.

Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

Вариант тела: Прямой эллиптический цилиндр.

1. Решение задачи

Язык программирования: C#.

Выбранный язык удобен для решения задачи, так как он обладает удобными библиотеками для построения фигур и вывода их на экран.

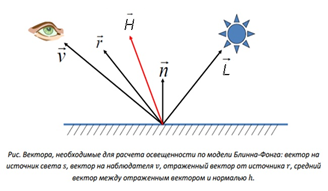
Пространство имен System.Drawing обеспечивает доступ к базовым функциональным возможностям графического интерфейса.

Пространство имен System.Collections.Generic содержит интерфейсы и классы, определяющие универсальные коллекции, которые позволяют обеспечить повышенную производительность и безопасность типов по сравнению с строго типизированными коллекциями.

С помощью правильного многоугольника строится прямой эллиптический цилиндр по вычисленным координатам. Затем добавляем источник света и создаем массивы распределения света.

Модель Блинна-Фонга.

Модель Блинна-Фонга – это унификация модели Фонга, исключающая расчет отраженного луча, что упрощает вычисления.



 где вектор *H* является «медианой» угла между векторами *V* и *L*.

Вектор *H* вычисляется по формуле: 

Модель Кука-Торренса.

Так как эта модель используется для расчета отраженного света, то рассеянный свет мы будем вычислять по классической формуле Ламберта, в которой освещенность точки зависит только от угла между нормалью к поверхности в данной точки, и положением источника света. Вычисляется как скалярное произведение нормали и нормализованного положения источника света:

Теперь рассмотрим модель Кука-Торренса. Количество отраженного света зависит от трех факторов:

1. Коэффициент Френеля (F);

2. Геометрическая составляющая, учитывающая самозатенение (G);

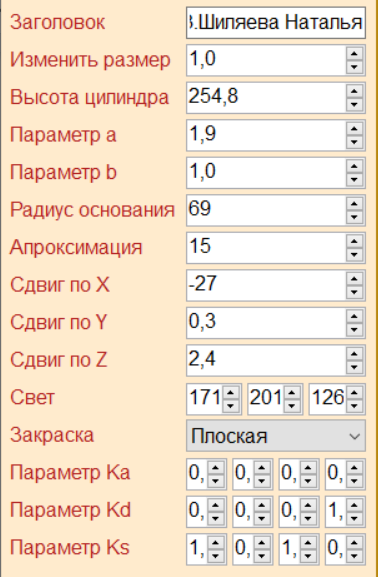
3. Компонент, учитывающий шероховатость поверхности (D);

Общая формула для вычисления отраженного света такова:

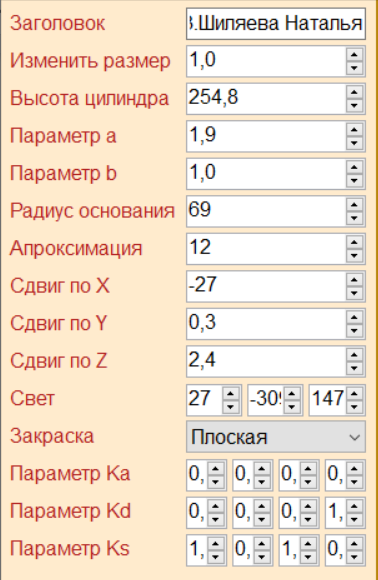
1. Набор тестов

Вводим все необходимые параметры прямого эллиптического цилиндра:

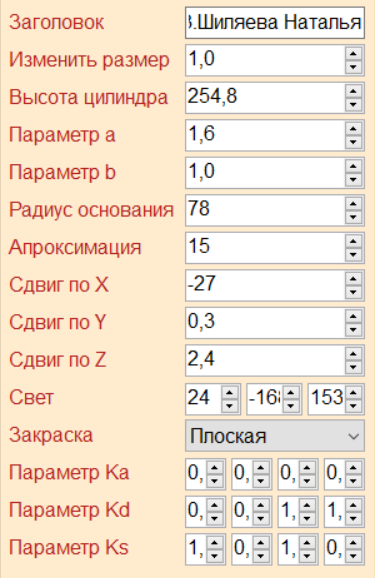
Тест 1:



Тест 2:



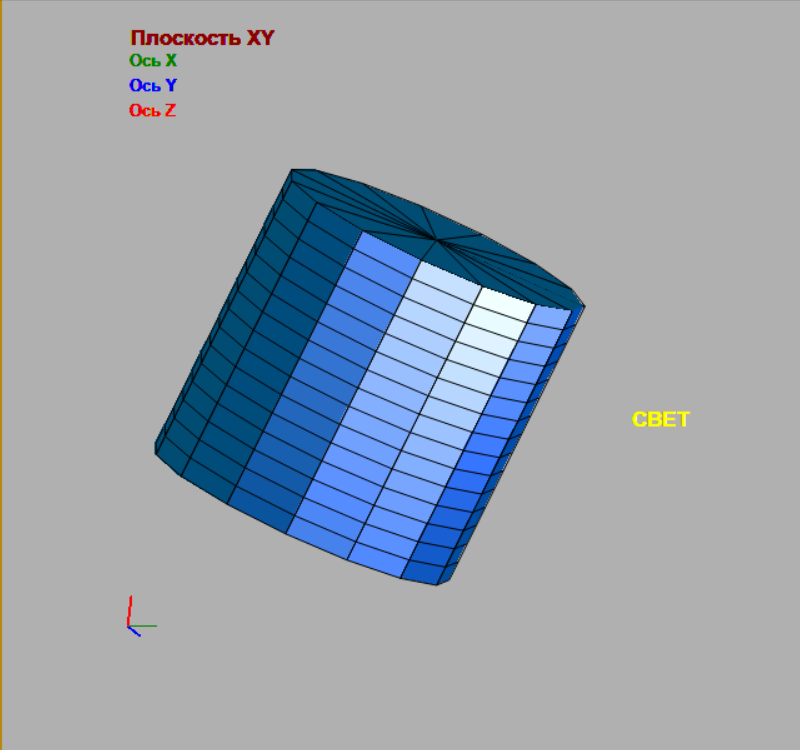
Тест 3:



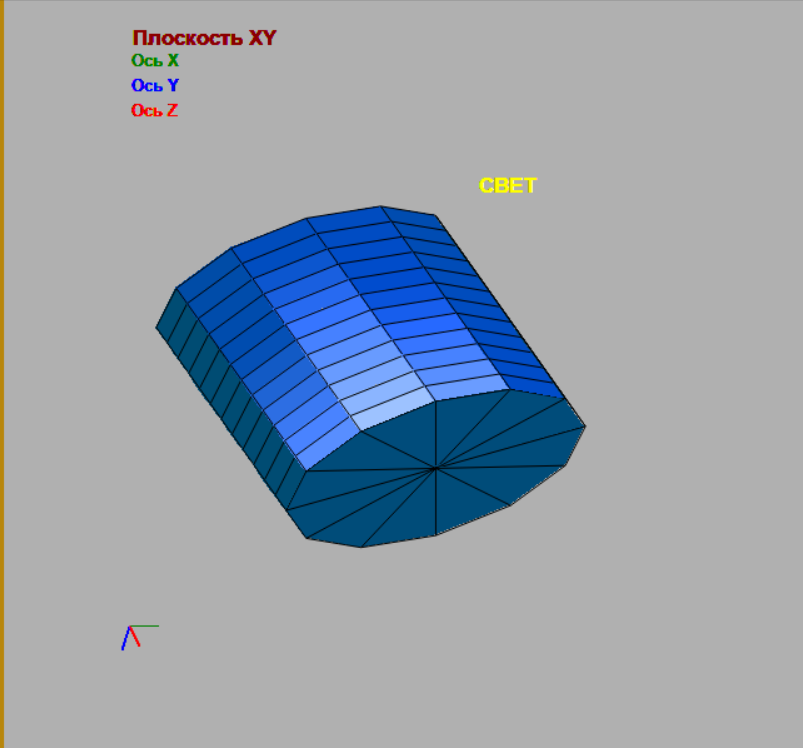
1. Результаты выполнения тестов

Наглядно видим выполнение программы по тестам, описанным ранее.

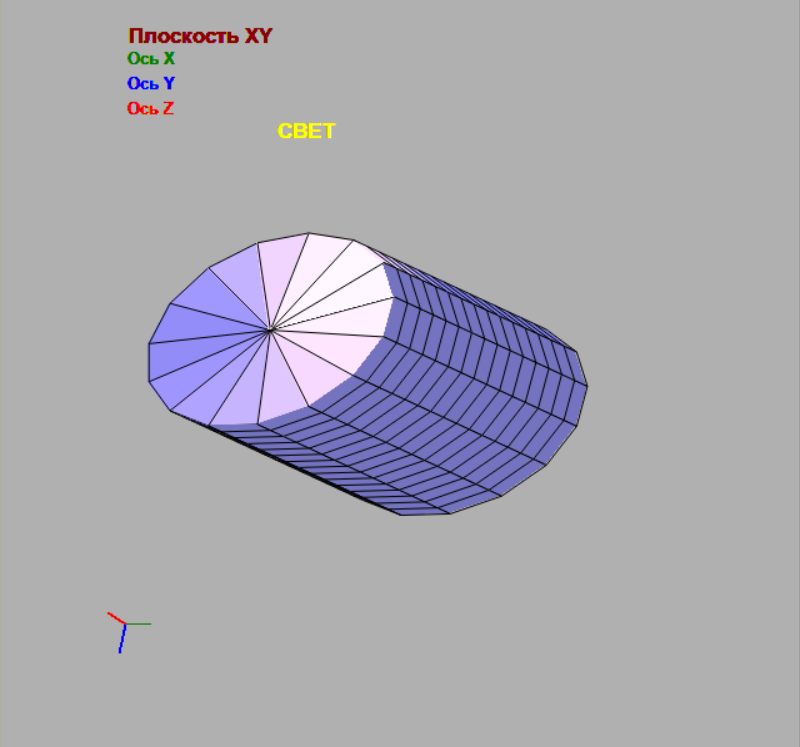
Тест 1:



Тест 2:



Тест 3:



1. Листинг программы

*Файл main.cs*

using System;

using System.Linq;

using System.Drawing;

using System.Drawing.Imaging;

using System.Windows.Forms;

using System.ComponentModel;

using System.Collections.Generic;

using CGLabPlatform;

namespace ConsoleApplication1

{

public abstract class CGLabEmpty : GFXApplicationTemplate<CGLabEmpty>

{

[DisplayTextBoxProperty("Лабораторная №3.Шиляева Наталья", "Заголовок")]

public virtual string LabelTxt

{

get { return \_LabelTxt; }

set { \_LabelTxt = value; }

}

private string \_LabelTxt;

[DisplayNumericProperty(1, 0.1, "Изменить размер", 1)]

public virtual double Size { get; set; }

[DisplayNumericProperty(250, 0.1, "Высота цилиндра", 1)]

public abstract double height { get; set; }

[DisplayNumericProperty(2, 0.1, "Параметр a", 1)]

public virtual double a { get; set; }

[DisplayNumericProperty(1, 0.1, "Параметр b", 1)]

public virtual double b { get; set; }

[DisplayNumericProperty(75, 1, "Радиус основания", 1)]

public abstract double radius { get; set; }

[DisplayNumericProperty(6, 1, "Апроксимация", 3)]

public abstract int resolution { get; set; }

[DisplayNumericProperty(0, 1, "Сдвиг по X")]

public abstract double ShiftX { get; set; }

[DisplayNumericProperty(0, 0.1, "Сдвиг по Y")]

public virtual double ShiftY { get; set; }

[DisplayNumericProperty(0, 0.1, "Сдвиг по Z")]

public virtual double ShiftZ { get; set; }

[DisplayNumericProperty(new double[] { 0, 0, 0 }, 1, "Свет")]

public virtual DVector3 Lighting { get; set; }

[DisplayEnumListProperty(Draw.Flat, "Закраска")]

public Draw draw

{

get { return Get<Draw>(); }

set { if (!Set<Draw>(value)) return; }

}

public enum Draw

{

//[Description("Не задан")] ND,

[Description("Плоская")] Flat,

[Description("Гуро")] Gouro,

}

[DisplayNumericProperty(new double[4] { 0d, 0.4d, 0.45d, 1.0 }, 0.05, "Параметр Ka", 0, 1)]

public virtual DVector4 ka { get; set; }

[DisplayNumericProperty(new double[] { 0d, 0d, 0.5d, 1.0 }, 0.05, "Параметр Kd", 0, 1)]

public virtual DVector4 kd { get; set; }

[DisplayNumericProperty(new double[] { 1d, 0.7d, 1d, 1.0 }, 0.05, "Параметр Ks", 0, 1)]

public virtual DVector4 ks { get; set; }

[STAThread]

static void Main()

{

RunApplication();

}

public double angleX = 0;

public double angleY = 0;

public double angleZ = 0;

public List<DVector4> axis = new List<DVector4>();//лист для осей

public int size = 6;

public DVector4 int\_bg = DVector4.One;//new DVector4(0.03, 0.07, 0.2, 1);

public DVector4 int\_main = DVector4.One;//new DVector4(0.2, 0.2, 0.6, 1);

public DVector4 int\_l = DVector4.One;//new DVector4(0.2, 0.2, 0.6, 1);

public DVector4 Light2 = new DVector4();

public void Solv(double h, int res)//Функция построения фигуры

{

List<Polygon> plist = new List<Polygon>();//лист полигонов (в пространстве)

List<DVector2> list1 = new List<DVector2>();//лист ключевых точек фигуры 1

List<DVector2> listH = new List<DVector2>();//лист ключевых точек фигуры 2

//сначала строим основание

double fi = 0;

list1.Clear();

listH.Clear();

plist.Clear();

double ht = (h / 2) / res;

list1.Add(new DVector2(0, 0)); //центр нижнего основания

listH.Add(new DVector2(0, 0)); //центр верхнего основания

for (int j = 0; j <= res - 1; j++)

{

list1.Add(new DVector2(a \* radius \* Math.Cos(fi), b \* radius \* Math.Sin(fi)));

listH.Add(new DVector2(a \* radius \* Math.Cos(fi), b \* radius \* Math.Sin(fi)));

fi = fi + 2 \* Math.PI / res;

}

fi = 0;

list1.Add(list1[1]);

listH.Add(listH[1]);

var obj3d = new ModelBuilder();

var z = -height / 2;

for (int i = 2; i < listH.Count; i++)

{

var p1 = list1[i - 1];

var p2 = list1[i];

obj3d.AddPolygon(new DVector3(p2, z), new DVector3(p1, z), new DVector3(list1[0], z));

obj3d.AddPolygon(new DVector3(p1, z + height), new DVector3(p2, z + height), new DVector3(list1[0], z + height));

double dt = height / res;

for (int j = 1; j <= res; j++)

{

var z1a = z + dt \* (j - 1); //a нижняя координата, b - верхняя координата

var z1b = z + dt \* j; //1 и 2 - соседние точки на основании

var z2a = z + dt \* (j - 1);

var z2b = z + dt \* j;

if (z2a == z2b)

obj3d.AddPolygon(new DVector3(p1, z1a), new DVector3(p2, z2a), new DVector3(p1, z1b));

else

{

obj3d.AddPolygon(new DVector3(p1, z1a), new DVector3(p2, z2a), new DVector3(p2, z2b), new DVector3(p1, z1b));

}

}

}

obj3d.Compile(out vertices, out polygons);

}

Polygon[] polygons;

Vertex[] vertices;

public DVector4 Light = new DVector4();

public void Move(double wd, double ht)

{

DMatrix4 zoom = new DMatrix4(Size, 0, 0, 0, 0, Size, 0, 0, 0, 0, Size, 0, 0, 0, 0, 1);

DMatrix4 shift = new DMatrix4(1, 0, 0, wd + ShiftX, 0, 1, 0, ht + ShiftY, 0, 0, 1, ShiftZ, 0, 0, 0, 1);

DMatrix4 rotx = new DMatrix4(1, 0, 0, 0, 0, Math.Cos(angleX), -Math.Sin(angleX), 0, 0, Math.Sin(angleX), Math.Cos(angleX), 0, 0, 0, 0, 1);

DMatrix4 roty = new DMatrix4(Math.Cos(angleY), 0, Math.Sin(angleY), 0, 0, 1, 0, 0, -Math.Sin(angleY), 0, Math.Cos(angleY), 0, 0, 0, 0, 1);

DMatrix4 rotz = new DMatrix4(Math.Cos(angleZ), -Math.Sin(angleZ), 0, 0, Math.Sin(angleZ), Math.Cos(angleZ), 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);

DMatrix4 point\_transform = shift \* zoom \* rotx \* roty \* rotz;

DMatrix4 light\_transform = new DMatrix4(1, 0, 0, ShiftX, 0, 1, 0, ShiftY, 0, 0, 1, ShiftZ, 0, 0, 0, 1) \* zoom \* rotx \* roty \* rotz;

DMatrix4 normal\_transform = DMatrix3.NormalVecTransf(point\_transform);

foreach (var vertex in vertices)

{

vertex.Point = point\_transform \* vertex.\_Point;

vertex.Normal = normal\_transform \* vertex.\_Normal;

}

foreach (var polygon in polygons)

{

polygon.Normal = normal\_transform \* polygon.\_Normal;

}

Light = light\_transform \* new DVector4(Lighting, 1);

}

public void Invisible\_Sides()

{

foreach (var vert in vertices)

vert.IsVisble = false;

foreach (var polygon in polygons)

{

DVector4 v1 = new DVector4(polygon.Vertex[0].Point.X - polygon.Vertex[2].Point.X, polygon.Vertex[0].Point.Y - polygon.Vertex[2].Point.Y, polygon.Vertex[0].Point.Z - polygon.Vertex[2].Point.Z, 1);

DVector4 v2 = new DVector4(polygon.Vertex[1].Point.X - polygon.Vertex[0].Point.X, polygon.Vertex[1].Point.Y - polygon.Vertex[0].Point.Y, polygon.Vertex[1].Point.Z - polygon.Vertex[0].Point.Z, 1);

polygon.\_Normal = v1 \* v2;

polygon.Normal = v1 \* v2;

if (polygon.Normal.GetLength() != 0)

{

polygon.Normal.Normalize();

}

if (polygon.\_Normal.GetLength() != 0)

{

polygon.\_Normal.Normalize();

}

polygon.Invisible = polygon.Normal.Z < 0;

if (!polygon.Invisible)

foreach (var vert in polygon.Vertex)

vert.IsVisble = true;

}

foreach (var polygon in polygons)

{

foreach (var vertex in vertices)

{

vertex.Normal = DVector4.Zero;

for (int j = 0; j < vertex.Polygon.Count(); j++)

{

vertex.Normal += vertex.Polygon[j].Normal;

}

vertex.Normal /= vertex.Polygon.Length;

vertex.Normal.Normalize();

}

}

}

public void Axis\_()

{

axis.Clear();

DMatrix4 rtx = new DMatrix4(1, 0, 0, 0, 0, Math.Cos(angleX), -Math.Sin(angleX), 0, 0, Math.Sin(angleX), Math.Cos(angleX), 0, 0, 0, 0, 1);

DMatrix4 rty = new DMatrix4(Math.Cos(angleY), 0, Math.Sin(angleY), 0, 0, 1, 0, 0, -Math.Sin(angleY), 0, Math.Cos(angleY), 0, 0, 0, 0, 1);

DMatrix4 rtz = new DMatrix4(Math.Cos(angleZ), -Math.Sin(angleZ), 0, 0, Math.Sin(angleZ), Math.Cos(angleZ), 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);

DMatrix4 dotx = new DMatrix4(25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1);

DMatrix4 doty = new DMatrix4(0, 0, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1);

DMatrix4 dotz = new DMatrix4(0, 0, 0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 1, 0, 0, 1);

DMatrix4 res1 = new DMatrix4();

DMatrix4 res2 = new DMatrix4();

DMatrix4 res3 = new DMatrix4();

res1 = rtx \* dotx;

res1 = rty \* res1;

res1 = rtz \* res1;

axis.Add(new DVector4(0, 0, 0, 1)); //первая точка начала координат

axis.Add(new DVector4(res1.M11, res1.M21, res1.M31, 1));

res2 = rtx \* doty;

res2 = rty \* res2;

res2 = rtz \* res2;

axis.Add(new DVector4(res2.M11, res2.M21, res2.M31, 1));

res3 = rtx \* dotz;

res3 = rty \* res3;

res3 = rtz \* res3;

axis.Add(new DVector4(res3.M11, res3.M21, res3.M31, 1));

}

public void drawvec(Color col, GDIDeviceUpdateArgs e, Polygon p, DVector3 v)

{

DVector4 Center\_d4 = DVector4.Zero;

foreach (var vert in p.Vertex)

Center\_d4 += vert.Point;

DVector3 Center = (DVector3)Center\_d4 / p.Vertex.Length;

var p1 = (DVector2)Center;

var p2 = p1 + (DVector2)v \* 40;

var p0 = (p1 - 1).ToFloatArray();

e.Surface.DrawLine(col.ToArgb(), p1, p2);

e.Graphics.DrawRectangle(new Pen(col), p0[0], p0[1], 3, 3);

}

public void drawvec2(Color col, GDIDeviceUpdateArgs e, Vertex p, DVector3 v)

{

DVector3 Center = new DVector3(p.Point.X, p.Point.Y, p.Point.Z);

var p1 = (DVector2)Center;

var p2 = p1 + (DVector2)v \* 40;

var p0 = (p1 - 1).ToFloatArray();

e.Surface.DrawLine(col.ToArgb(), p1, p2);

e.Graphics.DrawRectangle(new Pen(col), p0[0], p0[1], 3, 3);

}

public Vertex[] Vertecis;

public Polygon[] Polygons;

protected override void OnMainWindowLoad(object sender, EventArgs args)

{

base.RenderDevice.BufferBackCol = 0xB0;

base.ValueStorage.Font = new Font("Arial", 12f);

base.ValueStorage.ForeColor = Color.Firebrick;

base.ValueStorage.RowHeight = 30;

base.ValueStorage.BackColor = Color.BlanchedAlmond;

base.MainWindow.BackColor = Color.DarkGoldenrod;

base.ValueStorage.RightColWidth = 50;

base.VSPanelWidth = 300;

base.VSPanelLeft = true;

base.MainWindow.Size = new Size(960, 640);

base.RenderDevice.MouseMoveWithLeftBtnDown += (s, e) =>

{//смещение по осям

angleX += 0.01 \* e.MovDeltaX;

angleY += 0.01 \* e.MovDeltaY;

};

base.RenderDevice.MouseMoveWithRightBtnDown += (s, e) =>

{//вращение по осям OX и OY

ShiftX += 2 \* e.MovDeltaX;

ShiftY += 2 \* e.MovDeltaY;

};

base.RenderDevice.MouseMoveWithMiddleBtnDown += (s, e) =>

{//вращение по оси OZ

angleZ += 0.01 \* e.MovDeltaX;//двигаем по оси X (по столу), фигура вращается вокруг OZ

};

//Горячие клавиши из фреймворка

RenderDevice.HotkeyRegister(Keys.Up, (s, e) => ++ShiftY);

RenderDevice.HotkeyRegister(Keys.Down, (s, e) => --ShiftY);

RenderDevice.HotkeyRegister(Keys.Left, (s, e) => --ShiftX);

RenderDevice.HotkeyRegister(Keys.Right, (s, e) => ++ShiftX);

RenderDevice.HotkeyRegister(KeyMod.Shift, Keys.Up, (s, e) => ShiftY += 10);

RenderDevice.HotkeyRegister(KeyMod.Shift, Keys.Down, (s, e) => ShiftY -= 10);

RenderDevice.HotkeyRegister(KeyMod.Shift, Keys.Left, (s, e) => ShiftX -= 10);

RenderDevice.HotkeyRegister(KeyMod.Shift, Keys.Right, (s, e) => ShiftX += 10);

}

public DVector4 In(GDIDeviceUpdateArgs e, Polygon polygon, int v)

{

DVector4 l = new DVector4();

DVector3 Normal1 = new DVector3();

DVector3 CameraPos = new DVector3(0, 0, 10000);

DVector3 Eye = new DVector3();

switch (v)

{

case -1:

{

//p\_ = (DVector3)((p1 + p2 + p3) / 3);

DVector4 Center\_d4 = DVector4.Zero;

foreach (var vert in polygon.Vertex)

Center\_d4 += vert.Point;

DVector3 Center = (DVector3)Center\_d4 / polygon.Vertex.Length;

l = polygon.L = (Light - new DVector4(Center, 1)).Normalized();

Eye = polygon.E = (CameraPos - Center).Normalized();

Normal1 = (DVector3)polygon.Normal;

break;

}

case 0:

case 1:

case 2:

case 3:

{

l = polygon.Vertex[v].L = new DVector4(Light - polygon.Vertex[v].Point).Normalized();

Normal1 = (DVector3)polygon.Vertex[v].Normal;

Eye = ((DVector3)(new DVector4(CameraPos,1) - polygon.Vertex[v].Point)).Normalized();

} break;

default: { break; }

}

DVector4 Ia = new DVector4();

DVector4 Id = new DVector4();

DVector4 Is = new DVector4();

DVector3 Reflex = new DVector3();

DVector4 I = new DVector4();

Reflex = ((-(DVector3)l).Reflect(Normal1));

Reflex.Normalize();

switch (v)

{

case -1: polygon.R = Reflex; break;

}

Ia.X = ka.X \* int\_bg.X;

Ia.Y = ka.Y \* int\_bg.Y;

Ia.Z = ka.Z \* int\_bg.Z;

if (l.DotProduct(new DVector4(Normal1, 1)) > 0) {

Id.X = kd.X \* int\_main.X \* l.DotProduct(new DVector4(Normal1, 1));

Id.Y = kd.Y \* int\_main.Y \* l.DotProduct(new DVector4(Normal1, 1));

Id.Z = kd.Z \* int\_main.Z \* l.DotProduct(new DVector4(Normal1, 1));

}

if (l.DotProduct(new DVector4(Normal1, 1)) > 0 && Reflex.Z >= 0)

{

Is.X = ks.X \* int\_l.X \* Eye.DotProduct(Reflex);

Is.Y = ks.Y \* int\_l.Y \* Eye.DotProduct(Reflex);

Is.Z = ks.Z \* int\_l.Z \* Eye.DotProduct(Reflex);

}

if (Ia.X < 0)

Ia.X = 0;

if (Ia.Y < 0)

Ia.Y = 0;

if (Ia.Z < 0)

Ia.Z = 0;

if (Ia.X > 1)

Ia.X = 1;

if (Ia.Y > 1)

Ia.Y = 1;

if (Ia.Z > 1)

Ia.Z = 1;

if (Id.X < 0)

Id.X = 0;

if (Id.Y < 0)

Id.Y = 0;

if (Id.Z < 0)

Id.Z = 0;

if (Id.X > 1)

Id.X = 1;

if (Id.Y > 1)

Id.Y = 1;

if (Id.Z > 1)

Id.Z = 1;

if (Is.X < 0)

Is.X = 0;

if (Is.Y < 0)

Is.Y = 0;

if (Is.Z < 0)

Is.Z = 0;

if (Is.X > 1)

Is.X = 1;

if (Is.Y > 1)

Is.Y = 1;

if (Is.Z > 1)

Is.Z = 1;

I = Ia + Id + Is;

if (I.X < 0)

I.X = 0;

if (I.Y < 0)

I.Y = 0;

if (I.Z < 0)

I.Z = 0;

if (I.X > 1)

I.X = 1;

if (I.Y > 1)

I.Y = 1;

if (I.Z > 1)

I.Z = 1;

return I;

}

public void Intensive(GDIDeviceUpdateArgs e, Polygon polygon)

{

if (draw == Draw.Flat)

{

polygon.I = In(e, polygon, -1);

}

else

{

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

polygon.Vertex[i].I = In(e, polygon, i);

}

}

protected override void OnDeviceUpdate(object s, GDIDeviceUpdateArgs e)

{

Solv(height, resolution);

Move(0, 0);

Invisible\_Sides();

Axis\_();

foreach (var polygon in polygons)

{

if (polygon.Invisible == false)

{

Intensive(e, polygon);

}

}

Move(e.Width / 2, e.Heigh / 2);

foreach (var polygon in polygons)

{

if (polygon.Invisible == false)

{

var vert = polygon.Vertex; //отрисовка полигонов

for (int i = 0; i < vert.Length - 2; ++i)

{

if (draw == Draw.Flat)

{

e.Surface.DrawTriangle(Color.FromArgb(255, (int)Math.Round(255 \* polygon.I.X), (int)Math.Round(255 \* polygon.I.Y), (int)Math.Round(255 \* polygon.I.Z)).ToArgb(), polygon.Vertex[0].Point.X, polygon.Vertex[0].Point.Y, polygon.Vertex[1].Point.X, polygon.Vertex[1].Point.Y, polygon.Vertex[2].Point.X, polygon.Vertex[2].Point.Y);

if (polygon.Vertex.Length == 4) e.Surface.DrawTriangle(Color.FromArgb(255, (int)Math.Round(255 \* polygon.I.X), (int)Math.Round(255 \* polygon.I.Y), (int)Math.Round(255 \* polygon.I.Z)).ToArgb(), polygon.Vertex[0].Point.X, polygon.Vertex[0].Point.Y, polygon.Vertex[2].Point.X, polygon.Vertex[2].Point.Y, polygon.Vertex[3].Point.X, polygon.Vertex[3].Point.Y);

// e

}

if (draw == Draw.Gouro)

{

e.Surface.DrawTriangle(Color.FromArgb(255, (int)Math.Round(255 \* polygon.Vertex[0].I.X), (int)Math.Round(255 \* polygon.Vertex[0].I.Y), (int)Math.Round(255 \* polygon.Vertex[0].I.Z)).ToArgb(), polygon.Vertex[0].Point.X, polygon.Vertex[0].Point.Y, Color.FromArgb(255, (int)Math.Round(255 \* polygon.Vertex[1].I.X), (int)Math.Round(255 \* polygon.Vertex[1].I.Y), (int)Math.Round(255 \* polygon.Vertex[1].I.Z)).ToArgb(), polygon.Vertex[1].Point.X, polygon.Vertex[1].Point.Y, Color.FromArgb(255, (int)Math.Round(255 \* polygon.Vertex[2].I.X), (int)Math.Round(255 \* polygon.Vertex[2].I.Y), (int)Math.Round(255 \* polygon.Vertex[2].I.Z)).ToArgb(), polygon.Vertex[2].Point.X, polygon.Vertex[2].Point.Y);

if (polygon.Vertex.Length == 4) e.Surface.DrawTriangle(Color.FromArgb(255, (int)Math.Round(255 \* polygon.Vertex[0].I.X), (int)Math.Round(255 \* polygon.Vertex[0].I.Y), (int)Math.Round(255 \* polygon.Vertex[0].I.Z)).ToArgb(), polygon.Vertex[0].Point.X, polygon.Vertex[0].Point.Y, Color.FromArgb(255, (int)Math.Round(255 \* polygon.Vertex[2].I.X), (int)Math.Round(255 \* polygon.Vertex[2].I.Y), (int)Math.Round(255 \* polygon.Vertex[2].I.Z)).ToArgb(), polygon.Vertex[2].Point.X, polygon.Vertex[2].Point.Y, Color.FromArgb(255, (int)Math.Round(255 \* polygon.Vertex[3].I.X), (int)Math.Round(255 \* polygon.Vertex[3].I.Y), (int)Math.Round(255 \* polygon.Vertex[3].I.Z)).ToArgb(), polygon.Vertex[3].Point.X, polygon.Vertex[3].Point.Y);

}

if (!polygon.Invisible)

{

}

}

var rescnt = vertices.Select(v => new { pol = v.Polygon.Count() }).GroupBy(v => v.pol).ToDictionary(v => v.Key, l => l.Sum(i => 1));

var \_p1 = vert.Last().Point; //отрисовка контуров

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

{

var \_p2 = vert[i].Point;

e.Surface.DrawLine(Color.Black.ToArgb(), \_p1.X, \_p1.Y, \_p2.X, \_p2.Y);

\_p1 = \_p2;

}

e.Graphics.DrawString("СВЕТ", new Font("Arial", 13f, FontStyle.Bold), Brushes.Yellow,

new PointF((float)(Light.X + e.Width / 2), (float)(Light.Y + e.Heigh / 2)));

e.Graphics.DrawLine(Pens.Green, axis[0].X + 100, axis[0].Y + 500, axis[1].X + 100, axis[1].Y + 500);

e.Graphics.DrawLine(Pens.Blue, axis[0].X + 100, axis[0].Y + 500, axis[2].X + 100, axis[2].Y + 500);

e.Graphics.DrawLine(Pens.Red, axis[0].X + 100, axis[0].Y + 500, axis[3].X + 100, axis[3].Y + 500);

e.Graphics.DrawString("Плоскость XY", new Font("Arial", 13f, FontStyle.Bold), Brushes.DarkRed, new PointF(100, 20));

e.Graphics.DrawString("Ось X", new Font("Arial", 10f, FontStyle.Bold), Brushes.Green, new PointF(100, 40));

e.Graphics.DrawString("Ось Y", new Font("Arial", 10f, FontStyle.Bold), Brushes.Blue, new PointF(100, 60));

e.Graphics.DrawString("Ось Z", new Font("Arial", 10f, FontStyle.Bold), Brushes.Red, new PointF(100, 80));

}

}

}

}

}

1. Вывод

В ходе данной лабораторной работы получены основные навыки работы с трехмерной графикой C#, освоены методы реализации простой модели закраски для одного источника света и отрисовки плоскостей.

Список литературы

1. Шилдт, Герберт. Ш57 С# 4.0: полное руководство. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2011. — 1056 с.: ил. — Парал. тит. англ. (дата обращения: 17.10.2020).
2. Прайс. C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов, 3-е издание. -М.: Питер, 2018 - 640 с. (дата обращения: 18.10.2020).