Министерство науки и высшего образования Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа №2

по дисциплине

«Компьютерная графика»

Факультет прикладной математики и информатики

Группа ПМ-63

Студенты Майер В.А.

Сыпко Н.Е.

Преподаватели Задорожный А.Г.

Вариант Polygon

Новосибирск 2019

1. Цель работы

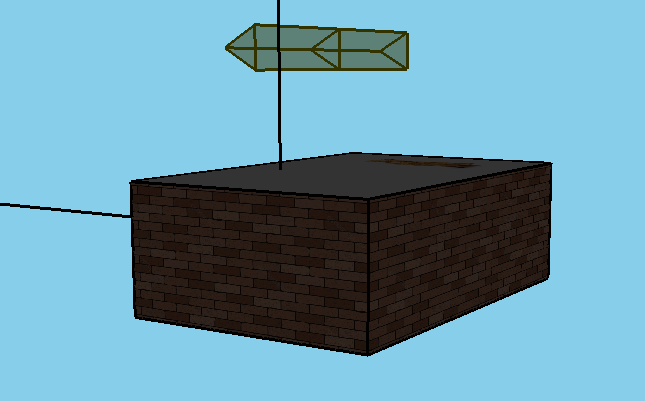
Ознакомиться с методом тиражирования сечений (основным способом задания полигональных моделей) и средствами трехмерной визуализации(системы координат, источники света, свойства материалов).

1. Текст задания
2. Построить 3д фигуру
3. Включить режимы бефера глубины, двойной буферизации, освещения материалов
4. Предоставить возможность показа каркаса объекта, нормалей, текстур
5. Предоставить возможность включения режима сглаживания нормалей
6. Предоставить возможность переключения между ортографической и перспективной камерой
7. Предоставить возможность передвижение по сцене
8. Использование различных источников освещения
9. Построение ломанного 3д объекта
10. Руководство пользователя

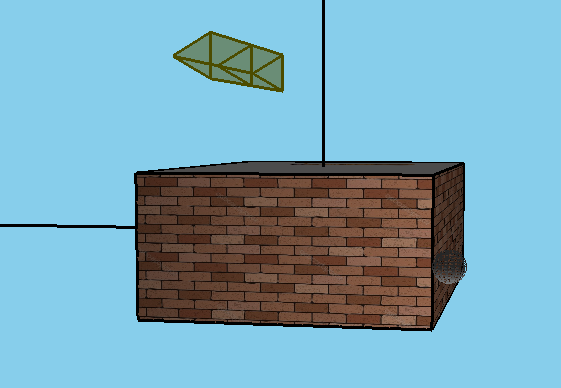
|  |  |
| --- | --- |
| Кнопка | Назначение |
| Вращение мыши | Вращение камеры |
| WASD | Передвижение по сцене(камера закреплена к фигуре, наплавление движения зависит от направления взгляда камеры) |
| QE | Вращение фигуры |
| V | Переключение между перспективой и ортографической камерой |
| Z | Добавление точечного источника света, который вращается |
| X | Добавление прожектора |
| Num 7 4 | Изменение размеров фигуры по оси X |
| Num 8 5 | Изменение размеров фигуры по оси Y |
| Num 9 6 | Изменение размеров фигуры по оси Z |
| +- | Изменение расстоянии камеры до фигуры |
| Left, Up, Right, Down | Передвижение центрального блока сломанного объекта |
| ,. | Изменение количества углов у сломанного объекта |
| I | Отключение сторон |
| O | Отключение контура |
| P | Включение нормалей |
| [ | Использование сглаживающий нормалей |
| Shift Ctrl | Изменение фигуры по вертикальной оси |
| Esc | Выход из программы |

1. **Тесты**

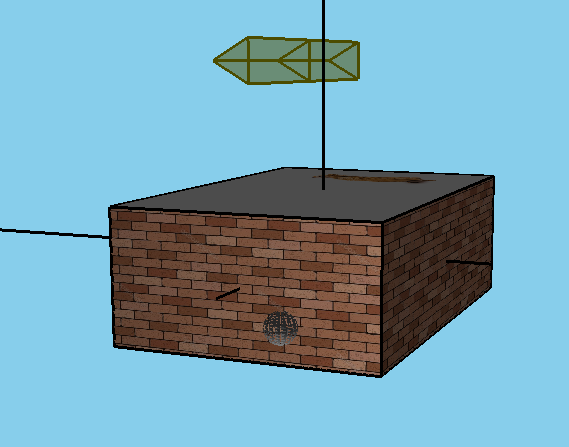
Создание объектов



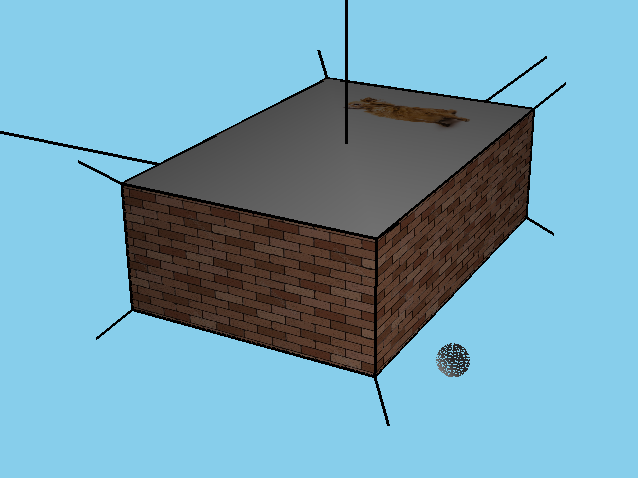
Включение источника света



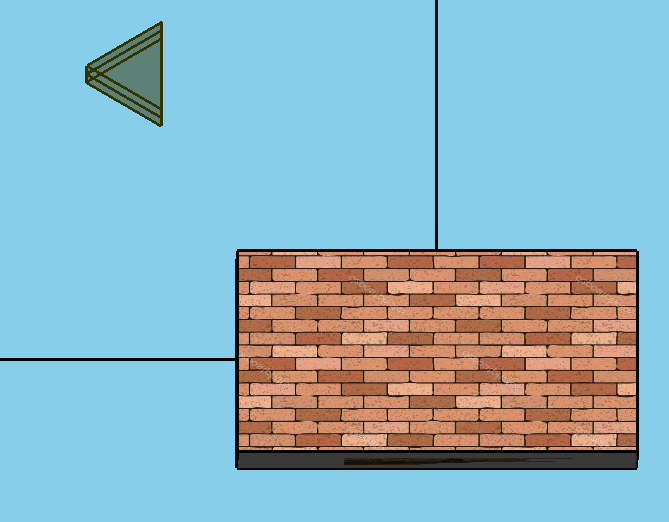
Включение нормалей



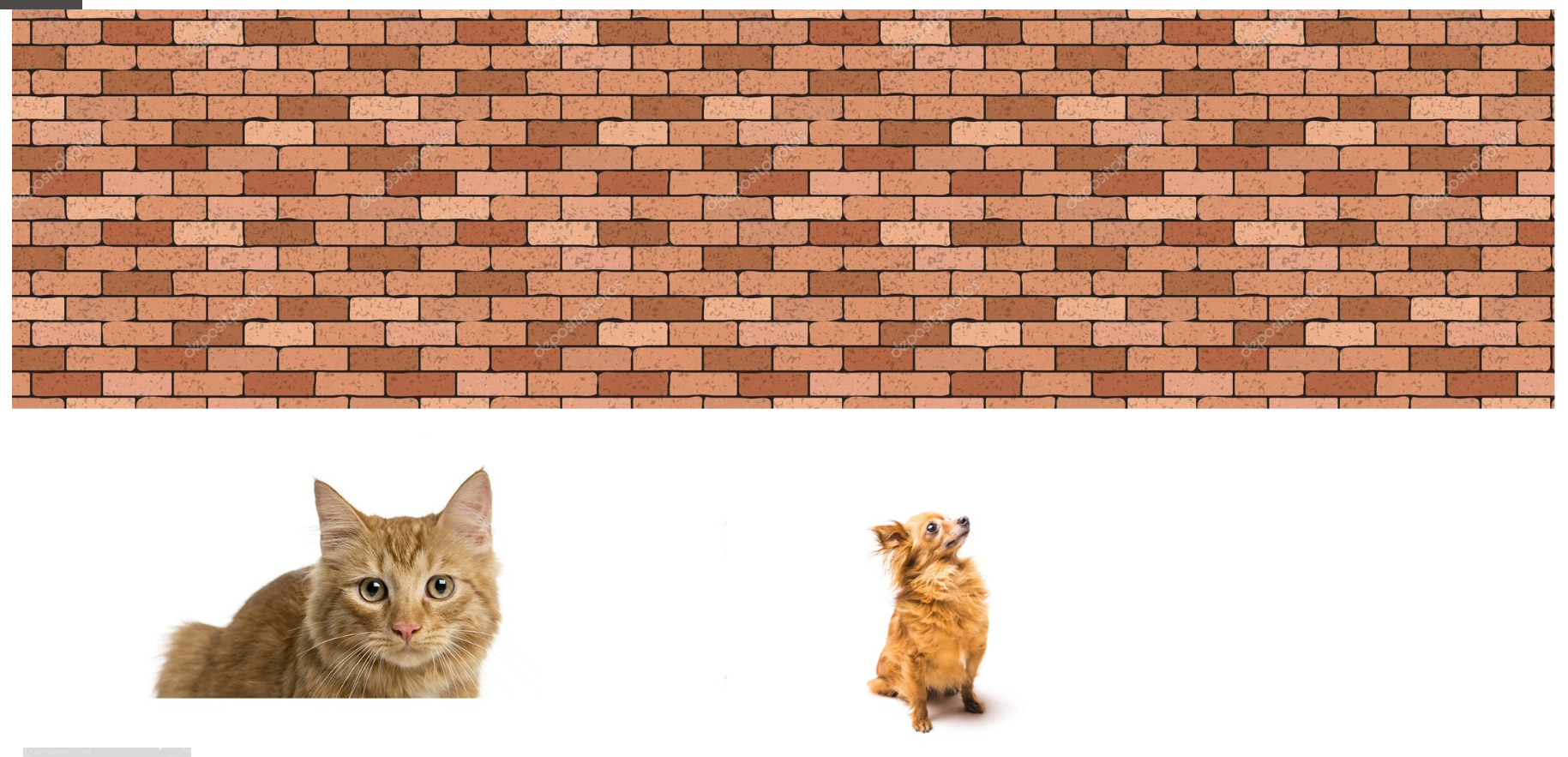
Использование сглаживающих нормалей



Использование ортографического режима

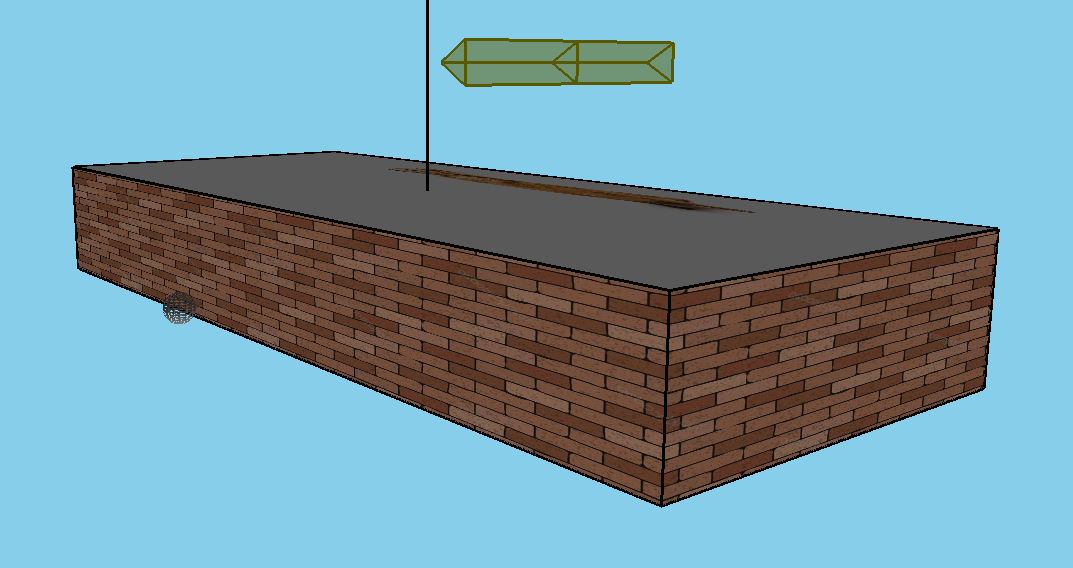


Развертка текстуры

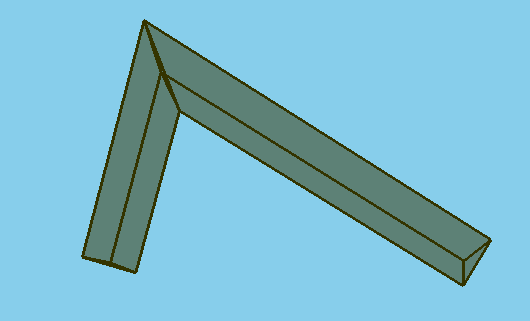


Идет отдельное разбиение на части, чтобы при изменение длины сторон объектов длина кирпичей оставалась одинаковая по кругу, поэтому текстура стены дробится в зависимости от соотношения сторон

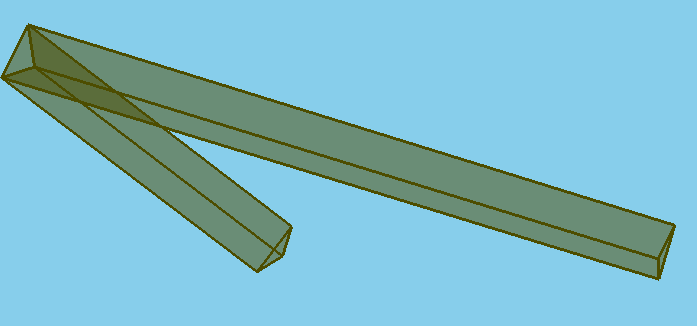
Пример: c растяжением по одной координате мы получаем растяжение текстуры и по другой, чтобы сохранять симметричность.



Построение ломанной фигуры



Случай когда центральное сечение вышло за пределы границ(после опускания кнопки передвижения граница и центральное сечение поменяются местами)



1. **Текст программы**

using System;

using System.Threading;

using System.IO;

using System.Text;

using System.Collections.Generic;

using System.Drawing;

using System.Drawing.Imaging;

using OpenTK;

using OpenTK.Graphics;

using OpenTK.Graphics.OpenGL;

using OpenTK.Input;

namespace Example

{

enum TypeObject

{

//Sphere,

Box

}

class Object

{

//Текустура

private TypeObject type;

public Vector3 worldCoordinat;

public Vector2 angele;

private float lennorm = 7;

public Vector3 width { set; get; }

private List<Vector3> points;

private List<Vector2> texturecoor;

private List<Vector3> smooth;

public bool poly, skel, nor, smothf;

public Object(TypeObject t, Vector3 wid = new Vector3(), Vector3 coor = new Vector3())

{

type = t;

angele = new Vector2();

worldCoordinat = coor;

width = wid;

points = new List<Vector3> { new Vector3(worldCoordinat.X - width.X, worldCoordinat.Y - width.Y, worldCoordinat.Z + width.Z),

new Vector3(worldCoordinat.X + width.X, worldCoordinat.Y - width.Y, worldCoordinat.Z + width.Z),

new Vector3(worldCoordinat.X + width.X, worldCoordinat.Y + width.Y, worldCoordinat.Z + width.Z),

new Vector3(worldCoordinat.X - width.X, worldCoordinat.Y + width.Y, worldCoordinat.Z + width.Z),

new Vector3(worldCoordinat.X - width.X, worldCoordinat.Y - width.Y, worldCoordinat.Z - width.Z),

new Vector3(worldCoordinat.X + width.X, worldCoordinat.Y - width.Y, worldCoordinat.Z - width.Z),

new Vector3(worldCoordinat.X + width.X, worldCoordinat.Y + width.Y, worldCoordinat.Z - width.Z),

new Vector3(worldCoordinat.X - width.X, worldCoordinat.Y + width.Y, worldCoordinat.Z - width.Z)};

float tp = width.X / (width.X + width.Z);

texturecoor = new List<Vector2> { new Vector2(0, 0),

new Vector2(tp, 0),

new Vector2(1, 0),

new Vector2(0, 0.5f),

new Vector2(tp, 0.5f),

new Vector2(0.5f, 1),

new Vector2(0.5f, 0.5f),

new Vector2(0f, 1f),

new Vector2(0.5f, 1f),

new Vector2(1f, 1f)};

smooth = new List<Vector3>

{

FindSmothNormal(new List<Vector3> { points[0], points[1], points[4], points[3] }),

FindSmothNormal(new List<Vector3> { points[1], points[0], points[2], points[5] }),

FindSmothNormal(new List<Vector3> { points[2], points[1], points[3], points[6] }),

FindSmothNormal(new List<Vector3> { points[3], points[0], points[7], points[2] }),

FindSmothNormal(new List<Vector3> { points[4], points[0], points[5], points[7] }),

FindSmothNormal(new List<Vector3> { points[5], points[1], points[6], points[4] }),

FindSmothNormal(new List<Vector3> { points[6], points[2], points[7], points[5] }),

FindSmothNormal(new List<Vector3> { points[7], points[0], points[4], points[6] })};

poly = skel = true;

smothf = nor = false;

}

public void Move(Vector3 p)

{

worldCoordinat += p;

}

private Vector3 Turn(Vector3 b, float angle)

{

Vector3 c = b;

float x = b.X - worldCoordinat.X;

float z = b.Z - worldCoordinat.Z; // game.Width;

//localCoor.Y = Convert.ToSingle(x \* Math.Sin(angle.Y) + y \* Math.Cos(angle.Y) + z \* Math.Sin( - angle.Y) + y \* Math.Cos(-angle.));

c.X = Convert.ToSingle(x \* Math.Cos(angle) - z \* Math.Sin(angle));

c.Z = Convert.ToSingle(x \* Math.Sin(angle) + z \* Math.Cos(angle));

c.X += worldCoordinat.X;

c.Z += worldCoordinat.Z;

return c;

}

public void Print(int textureID)

{

points[0] = new Vector3(worldCoordinat.X - width.X, worldCoordinat.Y - width.Y, worldCoordinat.Z + width.Z);

points[1] = new Vector3(worldCoordinat.X + width.X, worldCoordinat.Y - width.Y, worldCoordinat.Z + width.Z);

points[2] = new Vector3(worldCoordinat.X + width.X, worldCoordinat.Y + width.Y, worldCoordinat.Z + width.Z);

points[3] = new Vector3(worldCoordinat.X - width.X, worldCoordinat.Y + width.Y, worldCoordinat.Z + width.Z);

points[4] = new Vector3(worldCoordinat.X - width.X, worldCoordinat.Y - width.Y, worldCoordinat.Z - width.Z);

points[5] = new Vector3(worldCoordinat.X + width.X, worldCoordinat.Y - width.Y, worldCoordinat.Z - width.Z);

points[6] = new Vector3(worldCoordinat.X + width.X, worldCoordinat.Y + width.Y, worldCoordinat.Z - width.Z);

points[7] = new Vector3(worldCoordinat.X - width.X, worldCoordinat.Y + width.Y, worldCoordinat.Z - width.Z);

for (int i = 0; i < points.Count; i++)

points[i] = Turn(points[i], angele.X);

float tp = width.X / (width.X + width.Z);

texturecoor[0] = new Vector2(0, 0);

texturecoor[1] = new Vector2(tp, 0);

texturecoor[2] = new Vector2(1, 0);

texturecoor[3] = new Vector2(0, 0.5f);

texturecoor[4] = new Vector2(tp, 0.5f);

texturecoor[5] = new Vector2(1, 0.5f);

texturecoor[6] = new Vector2(0.5f, 0.5f);

texturecoor[7] = new Vector2(0f, 1f);

texturecoor[8] = new Vector2(0.5f, 1f);

texturecoor[9] = new Vector2(1f, 1f);

smooth[0] = FindSmothNormal(new List<Vector3> { points[0], points[1], points[4], points[3] });

smooth[1] = FindSmothNormal(new List<Vector3> { points[1], points[0], points[2], points[5] });

smooth[2] = FindSmothNormal(new List<Vector3> { points[2], points[1], points[3], points[6] });

smooth[3] = FindSmothNormal(new List<Vector3> { points[3], points[0], points[7], points[2] });

smooth[4] = FindSmothNormal(new List<Vector3> { points[4], points[0], points[5], points[7] });

smooth[5] = FindSmothNormal(new List<Vector3> { points[5], points[1], points[6], points[4] });

smooth[6] = FindSmothNormal(new List<Vector3> { points[6], points[2], points[7], points[5] });

smooth[7] = FindSmothNormal(new List<Vector3> { points[7], points[0], points[4], points[6] });

PrintPolygon(new List<Vector3> { points[4], points[7], points[6], points[5] }, new List<Vector2> { texturecoor[1], texturecoor[4], texturecoor[3], texturecoor[0]},new List<Vector3> {smooth[4], smooth[7], smooth[6], smooth[5]});

PrintPolygon(new List<Vector3> { points[3], points[0], points[1], points[2] }, new List<Vector2> { texturecoor[1], texturecoor[4], texturecoor[3], texturecoor[0]},new List<Vector3> {smooth[3], smooth[0], smooth[1], smooth[2]});

PrintPolygon(new List<Vector3> { points[1], points[5], points[6], points[2] }, new List<Vector2> { texturecoor[1], texturecoor[2], texturecoor[5], texturecoor[4]},new List<Vector3> {smooth[1], smooth[5], smooth[6], smooth[2]});

PrintPolygon(new List<Vector3> { points[4], points[0], points[3], points[7] }, new List<Vector2> { texturecoor[1], texturecoor[2], texturecoor[5], texturecoor[4]},new List<Vector3> {smooth[4], smooth[0], smooth[3], smooth[7]});

PrintPolygon(new List<Vector3> { points[0], points[4], points[5], points[1] }, new List<Vector2> { texturecoor[3], texturecoor[6], texturecoor[8], texturecoor[7]},new List<Vector3> {smooth[0], smooth[4], smooth[5], smooth[1]});

PrintPolygon(new List<Vector3> { points[2], points[6], points[7], points[3] }, new List<Vector2> { texturecoor[6], texturecoor[5], texturecoor[9], texturecoor[8]},new List<Vector3> {smooth[2], smooth[6], smooth[7], smooth[3]});

if (smothf && nor)

PrintSmothNormal(points, smooth);

}

private void PrintPolygon(List<Vector3> a, List<Vector2> te, List<Vector3> normal)

{

GL.Color3(Color.White);

if (poly)

{

GL.Enable(EnableCap.Texture2D);

GL.Begin(PrimitiveType.Polygon);

if(!smothf)

GL.Normal3(Oper.Normal(a[1], a[0], a[2]).Normalized());

for (int i = 0; i < a.Count; i++)

{

if (smothf)

GL.Normal3(normal[i]);

GL.TexCoord2(te[i]);

GL.Vertex3(a[i]);

}

GL.End();

GL.Disable(EnableCap.Texture2D);

}

GL.Color3(Color.Black);

if (skel)

{

GL.Begin(PrimitiveType.LineLoop);

for (int i = 0; i < a.Count; i++)

GL.Vertex3(a[i]);

GL.End();

}

if (nor && !smothf)

{

Vector3 s = new Vector3(0);

for (int i = 0; i < a.Count; i++)

s += a[i];

s /= a.Count;

GL.Begin(PrimitiveType.Lines);

GL.Vertex3(s);

GL.Vertex3(Oper.Normal(a[1], a[0], a[2]).Normalized() \* lennorm + s);

GL.End();

}

}

private void PrintSmothNormal(List<Vector3> a, List<Vector3> norm)

{

for (int i = 0; i < a.Count; i++)

{

GL.Begin(PrimitiveType.Lines);

GL.Vertex3(a[i]);

GL.Vertex3(norm[i] \* lennorm + a[i]);

GL.End();

}

}

private Vector3 FindSmothNormal(List<Vector3> a)

{

return (Oper.Normal(a[0], a[1], a[2]).Normalized() + Oper.Normal(a[0], a[2], a[3]).Normalized() + Oper.Normal(a[0], a[3], a[1]).Normalized()).Normalized();

}

public bool CheckCollison(ObjectCut cut)

{

return false;

}

}

public bool CheckCollison(ObjectCut cut)

{

return false;

}

}

class ObjectCut

{

public Vector3 beginpoint;

public Vector3 endpoint;

public Vector3 breakpoint;

public float r;

private double angleB;

private double angleE;

public int n;

private List<Vector3> points;

private List<Vector3> cross;

private Vector3 bufvec;

public ObjectCut()

{

r = 5;

angleB = 0;

angleE = 0;

n = 3;

beginpoint = new Vector3(30, 30, 30);

endpoint = new Vector3(30, 30, 90);

breakpoint = new Vector3(30, 30, 60);

points = new List<Vector3>();

cross = new List<Vector3>();

}

public void Print()

{

angleB = Math.Atan((beginpoint.Y - breakpoint.Y) / (breakpoint.Z - beginpoint.Z));

angleE = Math.Atan((endpoint.Y - breakpoint.Y) / (endpoint.Z - breakpoint.Z));

float \_cosB = (float)Math.Cos(angleB);

float \_sinB = (float)Math.Sin(angleB);

float \_cosE = (float)Math.Cos(-angleE);

float \_sinE = (float)Math.Sin(-angleE);

float \_cosM = (\_cosB + \_cosE) / 2.0f;

float \_sinM = (\_sinB + \_sinE) / 2.0f;

if (n != points.Count)

{

points.Clear();

cross.Clear();

for (int i = n - 1, i1 = 0; i >= 0; i--, i1++)

{

bufvec = new Vector3(beginpoint.X + r \* (float)Math.Cos(2 \* Math.PI \* i / n), beginpoint.Y + r \* (float)Math.Sin(2 \* Math.PI \* i / n), beginpoint.Z) - beginpoint;

points.Add(new Vector3(bufvec.X, bufvec.Y \* \_cosB + bufvec.Z \* \_sinB, bufvec.Y \* \_sinB - bufvec.Z \* \_cosB) + beginpoint);

}

for (int i = 0, i1 = n; i < n; i++, i1++)

{

bufvec = new Vector3(endpoint.X + r \* (float)Math.Cos(2 \* Math.PI \* i / n), endpoint.Y + r \* (float)Math.Sin(2 \* Math.PI \* i / n), endpoint.Z) - endpoint;

points.Add(new Vector3(bufvec.X, bufvec.Y \* \_cosE + bufvec.Z \* \_sinE, bufvec.Y \* \_sinE - bufvec.Z \* \_cosE) + endpoint);

}

for (int i = 0, i1 = n; i < n; i++, i1++)

{

bufvec = new Vector3(breakpoint.X + r \* (float)Math.Cos(2 \* Math.PI \* i / n), breakpoint.Y + r \* (float)Math.Sin(2 \* Math.PI \* i / n), breakpoint.Z) - breakpoint;

cross.Add(new Vector3(bufvec.X, bufvec.Y / \_cosB, bufvec.Z) + breakpoint);

}

for (int i = 0, i1 = n; i < n; i++, i1++)

{

bufvec = new Vector3(breakpoint.X + r \* (float)Math.Cos(2 \* Math.PI \* i / n), breakpoint.Y + r \* (float)Math.Sin(2 \* Math.PI \* i / n), breakpoint.Z) - breakpoint;

cross.Add(new Vector3(bufvec.X, bufvec.Y / \_cosE, bufvec.Z) + breakpoint);

}

for (int i = 0, i1 = n - 1, i2 = 2 \* n - 1; i < n; i++, i1--, i2--)

{

points.Add(Cross(points[i], cross[i1], cross[i2], points[i2]));

}

}

else

{

for (int i = n - 1, i1 = 0; i >= 0; i--, i1++)

{

bufvec = new Vector3(beginpoint.X + r \* (float)Math.Cos(2 \* Math.PI \* i / n), beginpoint.Y + r \* (float)Math.Sin(2 \* Math.PI \* i / n), beginpoint.Z) - beginpoint;

points[i1] = new Vector3(bufvec.X, bufvec.Y \* \_cosB + bufvec.Z \* \_sinB, bufvec.Y \* \_sinB - bufvec.Z \* \_cosB) + beginpoint;

}

for (int i = 0, i1 = n; i < n; i++, i1++)

{

bufvec = new Vector3(endpoint.X + r \* (float)Math.Cos(2 \* Math.PI \* i / n), endpoint.Y + r \* (float)Math.Sin(2 \* Math.PI \* i / n), endpoint.Z) - endpoint;

points[i1] = new Vector3(bufvec.X, bufvec.Y \* \_cosE + bufvec.Z \* \_sinE, bufvec.Y \* \_sinE - bufvec.Z \* \_cosE) + endpoint;

}

for (int i = 0, i1 = 2 \* n; i < n; i++, i1++)

points[i1] = new Vector3(breakpoint.X + r \* (float)Math.Cos(2 \* Math.PI \* i / n), breakpoint.Y + r \* (float)Math.Sin(2 \* Math.PI \* i / n), breakpoint.Z);

}

GL.Color4(1.0f, 1.0f, 0.0f, 0.5);

GL.BlendFunc(BlendingFactor.SrcAlpha, BlendingFactor.OneMinusSrcAlpha);

GL.Disable(EnableCap.DepthTest);

GL.DepthMask(true);

GL.DepthFunc(DepthFunction.Lequal);

GL.Enable(EnableCap.Blend);

GL.Enable(EnableCap.CullFace);

GL.Begin(PrimitiveType.Polygon);

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

GL.Vertex3(points[i]);

GL.End();

GL.Begin(PrimitiveType.Polygon);

for (int i = n; i < 2 \* n; i++)

GL.Vertex3(points[i]);

GL.End();

for (int i = 0, i1 = 2 \* n - 1, i2 = 2 \* n; i < n; i++, i1--, i2++)

{

GL.Begin(PrimitiveType.Polygon);

GL.Vertex3(points[i]);

GL.Vertex3(points[i2]);

GL.Vertex3(points[(i2 + 1) % n + 2 \* n]);

GL.Vertex3(points[(i + 1) % n]);

GL.End();

GL.Begin(PrimitiveType.Polygon);

GL.Vertex3(points[i2]);

GL.Vertex3(points[i1]);

GL.Vertex3(points[(i1 - 1) % n + n]);

GL.Vertex3(points[(i2 + 1) % n + 2 \* n]);

GL.End();

}

GL.Disable(EnableCap.CullFace);

GL.Enable(EnableCap.DepthTest);

GL.Disable(EnableCap.Blend);

for (int i = 0, i1 = 2 \* n - 1, i2 = 2 \* n; i < n; i++, i1--, i2++)

{

GL.Begin(PrimitiveType.LineLoop);

GL.Vertex3(points[i]);

GL.Vertex3(points[i2]);

GL.Vertex3(points[(i2 + 1) % n + 2 \* n]);

GL.Vertex3(points[(i + 1) % n]);

GL.End();

GL.Begin(PrimitiveType.LineLoop);

GL.Vertex3(points[i2]);

GL.Vertex3(points[i1]);

GL.Vertex3(points[(i1 - 1) % n + n]);

GL.Vertex3(points[(i2 + 1) % n + 2 \* n]);

GL.End();

}

}

private Vector3 Cross(Vector3 a, Vector3 b, Vector3 c, Vector3 d) //точки a и b концы первого отрезка c и d второго

{

if (b == c)

return b;

Vector3 T = new Vector3();

T.Z = -((a.Z \* b.Y - b.Z \* a.Y) \* (d.Z - c.Z) - (c.Z \* d.Y - d.Z \* c.Y) \* (b.Z - a.Z)) / ((a.Y - b.Y) \* (d.Z - c.Z) - (c.Y - d.Y) \* (b.Z - a.Z));

T.Y = ((c.Y - d.Y) \* (-T.Z) - (c.Z \* d.Y - d.Z \* c.Y)) / (d.Z - c.Z);

T.X = a.X;

return T;

}

}

class Light

{

Vector3 pos1;

public bool t0, t1, t2;

public Light()

{

GL.Enable(EnableCap.Lighting);

pos1 = new Vector3(50, 0, 0);

t1 = false;

t2 = false;

GL.Material(MaterialFace.Front, MaterialParameter.AmbientAndDiffuse, new Color4(0.3f, 0.3f, 0.3f, 0.5f));

GL.Enable(EnableCap.ColorMaterial);

}

public void Show0()

{

GL.Enable(EnableCap.DepthTest);

GL.Enable(EnableCap.ColorMaterial);

GL.Enable(EnableCap.Lighting);

GL.Enable(EnableCap.Light0);

GL.Light(LightName.Light0, LightParameter.Position, new float[4] { 0f, 1f, 0f, 1f });

GL.Light(LightName.Light0, LightParameter.Ambient, new float[4] { 0.2f, 0.2f, 0.2f, 1f });

GL.Light(LightName.Light0, LightParameter.Diffuse, new float[3] { 0.4f, 0.6f, 0.4f });

GL.Light(LightName.Light0, LightParameter.LinearAttenuation, 0.01f);

//GL.Light(LightName.Light0, LightParameter.SpotCutoff, new float[3] { 1, 0, 0});

//GL.Light(LightName.Light0, LightParameter.SpotExponent, new float[3] { 1, 0, 0});

//GL.Light(LightName.Light0, LightParameter.SpotDirection, new float[3] { 1, 0, 0});

//GL.Light(LightName.Light0, LightParameter.ConstantAttenuation, 0.3f);

//GL.Light(LightName.Light0, LightParameter.Diffuse, 0.3f);

//GL.Light(LightName.Light0, LightParameter.Ambient, 0.2f);

//GL.LightModel(LightModelParameter.LightModelTwoSide, 1);

//GL.LightModel(LightModelParameter.LightModelColorControl, 1);

//GL.LightModel(LightModelParameter.LightModelLocalViewer, 1);

}

public void Show1()

{

if (t1)

{

GL.Enable(EnableCap.Light1);

GL.Light(LightName.Light1, LightParameter.Position, new Color4(pos1.X, pos1.Y, pos1.Z, 1f));

GL.Light(LightName.Light1, LightParameter.Ambient, new Color4(0.1f, 0.1f, 0.1f, 1.0f));

GL.Light(LightName.Light1, LightParameter.Diffuse, new Color4(0.7f, 0.7f, 0.7f, 1f));

GL.Light(LightName.Light1, LightParameter.LinearAttenuation, 0.03f);

}

else

GL.Disable(EnableCap.Light1);

}

public void Show2()

{

if (t2)

{

GL.Enable(EnableCap.Light2);

GL.Light(LightName.Light2, LightParameter.SpotDirection, new float[4] { 0.0f, 0.0f, 1.0f, 0.0f });

GL.Light(LightName.Light2, LightParameter.SpotCutoff, 90);

GL.Light(LightName.Light2, LightParameter.SpotExponent, 2.0f);

GL.Light(LightName.Light2, LightParameter.Ambient, new Color4(0.001f, 0.001f, 0.001f, 0.001f));

GL.Light(LightName.Light2, LightParameter.Diffuse, new Color4(0.8f, 0.8f, 0.8f, 1f));

//GL.Light(LightName.Light2, LightParameter.LinearAttenuation, 0.03f);

}

else

{

GL.Disable(EnableCap.Light2);

}

}

public void Print1()

{

GL.Color4(0.5f, 0.5f, 0.5f, 1f);

float r = 2;

float startU = 0;

float startV = 0;

float endU = (float)Math.PI \* 2;

float endV = (float)Math.PI;

int UResolution = 16;

int VResolution = 16;

float stepU = (endU - startU) / UResolution; // step size between U-points on the grid

float stepV = (endV - startV) / VResolution; // step size between V-points on the grid

int i, j;

float u, v, un, vn;

for (i = 0; i < UResolution; i++)

{ // U-points

for (j = 0; j < VResolution; j++)

{ // V-points

u = i \* stepU + startU;

v = j \* stepV + startV;

un = (i + 1 == UResolution) ? endU : (i + 1) \* stepU + startU;

vn = (j + 1 == VResolution) ? endV : (j + 1) \* stepV + startV;

Vector3 p0 = new Vector3((float)(Math.Cos(u) \* Math.Sin(v) \* r),(float)( Math.Cos(v) \* r),(float)( Math.Sin(u) \* Math.Sin(v) \* r));

Vector3 p1 = new Vector3((float)(Math.Cos(u) \* Math.Sin(vn) \* r),(float)( Math.Cos(vn) \* r),(float)( Math.Sin(u) \* Math.Sin(vn) \* r));

Vector3 p2 = new Vector3((float)(Math.Cos(un) \* Math.Sin(v) \* r),(float)( Math.Cos(v) \* r),(float)( Math.Sin(un) \* Math.Sin(v) \* r));

Vector3 p3 = new Vector3((float)(Math.Cos(un) \* Math.Sin(vn) \* r),(float)( Math.Cos(vn) \* r),(float)( Math.Sin(un) \* Math.Sin(vn) \* r));

GL.Begin(PrimitiveType.Polygon);

GL.Vertex3(p0 + pos1);

GL.Vertex3(p2 + pos1);

GL.Vertex3(p1 + pos1);

GL.End();

//GL.Begin(PrimitiveType.Polygon);

//GL.Vertex3(p3 + pos1);

//GL.Vertex3(p1 + pos1);

//GL.Vertex3(p2 + pos1);

//GL.End();

}

}

}

public void Turn1(Vector2 angle)

{

float x = pos1.X; // game.Height;

float z = pos1.Z; // game.Width;

//localCoor.Y = Convert.ToSingle(x \* Math.Sin(angle.Y) + y \* Math.Cos(angle.Y) + z \* Math.Sin( - angle.Y) + y \* Math.Cos(-angle.));

pos1.X = Convert.ToSingle(x \* Math.Cos(angle.X) - z \* Math.Sin(angle.X));// \* game.Height;

pos1.Z = Convert.ToSingle(x \* Math.Sin(angle.X) + z \* Math.Cos(angle.X));// \* game.Width;

}

};

class Camera

{

private Object target;

private Vector3 localCoor;

public float len { set; get; }

#region Standart

public Camera(Object o, GameWindow game, Vector3 lp = new Vector3())

{

GL.ClearColor(Color.SkyBlue);

GL.Enable(EnableCap.DepthTest);

persp(game);

target = o;

localCoor = lp;

Matrix4 modelview = Matrix4.LookAt(target.worldCoordinat.X + localCoor.X, target.worldCoordinat.Y + localCoor.Y, target.worldCoordinat.Z + localCoor.Z, target.worldCoordinat.X, target.worldCoordinat.Y, target.worldCoordinat.Z, 0, 1, 0);

GL.MatrixMode(MatrixMode.Modelview);

GL.LoadMatrix(ref modelview);

len = Convert.ToSingle(Oper.Norm3D(localCoor));

}

public void persp(GameWindow game)

{

Matrix4 p = Matrix4.CreatePerspectiveFieldOfView((float)(50 \* Math.PI / 180), 1f \* game.Width / game.Height, 1, 5000);

GL.MatrixMode(MatrixMode.Projection);

GL.LoadMatrix(ref p);

}

public void ortg(GameWindow game)

{

Matrix4 p = Matrix4.CreateOrthographic(game.Width / 10, game.Height / 10, 1, 5000);

GL.MatrixMode(MatrixMode.Projection);

GL.LoadMatrix(ref p);

}

#endregion

#region Paint

public void Paint()

{

localCoor = localCoor.Normalized() \* len;

Matrix4 modelview = Matrix4.LookAt(target.worldCoordinat.X + localCoor.X, target.worldCoordinat.Y + localCoor.Y, target.worldCoordinat.Z + localCoor.Z, target.worldCoordinat.X, target.worldCoordinat.Y, target.worldCoordinat.Z, 0, 1, 0);

GL.MatrixMode(MatrixMode.Modelview);

GL.LoadMatrix(ref modelview);

}

#endregion

#region Moving

public void Move(Vector2 p)

{

float cx = Convert.ToSingle(p.X / Oper.Norm2D(localCoor));

float cy = Convert.ToSingle(p.Y / Oper.Norm2D(localCoor));

target.worldCoordinat.X += -localCoor.X \* cx + localCoor.Z \* cy;

target.worldCoordinat.Z += -localCoor.Z \* cx - localCoor.X \* cy;

}

public void Turn(Vector2 angle)

{

float x = localCoor.X; // game.Height;

float z = localCoor.Z; // game.Width;

//localCoor.Y = Convert.ToSingle(x \* Math.Sin(angle.Y) + y \* Math.Cos(angle.Y) + z \* Math.Sin( - angle.Y) + y \* Math.Cos(-angle.));

localCoor.X = Convert.ToSingle(x \* Math.Cos(angle.X) - z \* Math.Sin(angle.X));// \* game.Height;

localCoor.Z = Convert.ToSingle(x \* Math.Sin(angle.X) + z \* Math.Cos(angle.X));// \* game.Width;

float len = Convert.ToSingle(Math.Sqrt(Math.Pow(x, 2) + Math.Pow(z, 2)));

float y = localCoor.Y;

float len1;

localCoor.Y = Convert.ToSingle(y \* Math.Cos(angle.Y) - len \* Math.Sin(angle.Y));

len1 = Convert.ToSingle(y \* Math.Sin(angle.Y) + len \* Math.Cos(angle.Y));

localCoor.X \*= len1 / len;

localCoor.Z \*= len1 / len;

}

#endregion

}

partial class MyApplication

{

//Иноформирует активное акно или нет

static bool activ = true;

//Загрузка текстуры по патчу file

public static int LoadTexture(string file)

{

Bitmap bitmap = new Bitmap(file);

int tex;

GL.Hint(HintTarget.PerspectiveCorrectionHint, HintMode.Nicest);

GL.GenTextures(1, out tex);

GL.BindTexture(TextureTarget.Texture2D, tex);

BitmapData data = bitmap.LockBits(new System.Drawing.Rectangle(0, 0, bitmap.Width, bitmap.Height),

ImageLockMode.ReadOnly, System.Drawing.Imaging.PixelFormat.Format32bppArgb);

GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba, data.Width, data.Height, 0,

OpenTK.Graphics.OpenGL.PixelFormat.Bgra, PixelType.UnsignedByte, data.Scan0);

bitmap.UnlockBits(data);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter, (int)TextureMinFilter.Linear);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter, (int)TextureMagFilter.Linear);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapS, (int)TextureWrapMode.Repeat);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapT, (int)TextureWrapMode.Repeat);

return tex;

}

[STAThread]

public static void Main()

{

using (var game = new GameWindow())

{

game.Title = "Lab2";

game.WindowState = WindowState.Maximized;

game.WindowBorder = WindowBorder.Fixed | WindowBorder.Hidden;

game.CursorVisible = false;

bool pof = true;

int textureID = LoadTexture("C:/Users/PM65M/OneDrive/Desktop/Graf2.jpg");

Object box = new Object(TypeObject.Box, new Vector3(20, 10 ,30));

Camera camera = new Camera(box, game, new Vector3(100));

ObjectCut cut = new ObjectCut();

Light light = new Light();

game.Load += (sender, e) =>

{

game.VSync = VSyncMode.On;

GL.LineWidth(3);

};

game.Resize += (sender, e) =>

{

GL.Viewport(0, 0, game.Width, game.Height);

};

game.MouseMove += (sender, e) =>

{

camera.Turn(new Vector2((e.X - game.Width / 2f) / game.Width \* 2f, (game.Height / 2f - e.Y + 2) / game.Height \* 2f));

};

game.KeyDown += (sender, e) =>

{

if (e.Key == Key.G)

{

GL.MatrixMode(MatrixMode.Projection);

GL.Rotate(30, 0, 0, 1);

}

if (e.Key == Key.BracketLeft)

box.smothf = !box.smothf;

if (e.Key == Key.Z)

light.t1 = !light.t1;

if (e.Key == Key.X)

light.t2 = !light.t2;

if (e.Key == Key.Plus)

camera.len += 5;

if (e.Key == Key.Minus)

camera.len -= 5;

if(e.Key == Key.V)

if(pof)

{

camera.ortg(game);

pof = false;

}

else

{

camera.persp(game);

pof = true;

}

if (e.Key == Key.Keypad7)

box.width += new Vector3(1f, 0f, 0f);

if (e.Key == Key.Keypad4)

box.width += new Vector3(-1f, 0f, 0f);

if (e.Key == Key.Keypad8)

box.width += new Vector3(0f, 1f, 0f);

if (e.Key == Key.Keypad5)

box.width += new Vector3(0f, -1f, 0f);

if (e.Key == Key.Keypad9)

box.width += new Vector3(0f, 0f, 1f);

if (e.Key == Key.Keypad6)

box.width += new Vector3(0f, 0f, -1f);

if (e.Key == Key.Q)

box.angele += new Vector2(3.14f / 180 \* 10, 0);

if (e.Key == Key.E)

box.angele -= new Vector2(3.14f / 180 \* 10, 0);

if (e.Key == Key.ShiftLeft)

box.Move(new Vector3(0, 10, 0));

//GL.Translate(0, 10, 0);

if (e.Key == Key.LControl)

box.Move(new Vector3(0, -10, 0));

//GL.Translate(0, -10, 0);

if (e.Key == Key.I)

box.poly = !box.poly;

if (e.Key == Key.O)

box.skel= !box.skel;

if (e.Key == Key.P)

box.nor = !box.nor;

if (e.Key == Key.Comma)

if(cut.n > 3)

cut.n--;

if (e.Key == Key.Period)

cut.n++;

if(e.Key == Key.Right)

cut.breakpoint.Z += 1;

if (e.Key == Key.Left)

cut.breakpoint.Z -= 1;

if (e.Key == Key.Up)

cut.breakpoint.Y += 1;

if (e.Key == Key.Down)

cut.breakpoint.Y -= 1;

if (e.Key == Key.B)

{

GL.MatrixMode(MatrixMode.Modelview);

GL.Rotate(30, 0, 0, 1);

}

if (e.Key == Key.Escape)

game.Close();

};

game.KeyUp += (sender, e) =>

{

if (e.Key == Key.Right)

if (cut.breakpoint.Z > cut.endpoint.Z)

{

Vector3 c = cut.endpoint;

cut.endpoint = cut.breakpoint;

cut.breakpoint = c;

}

if (e.Key == Key.Left)

if (cut.breakpoint.Z < cut.beginpoint.Z)

{

Vector3 c = cut.beginpoint;

cut.beginpoint = cut.breakpoint;

cut.breakpoint = c;

}

};

game.UpdateFrame += (sender, e) =>

{

//GL.MatrixMode(MatrixMode.Modelview);

//GL.Rotate(1, 0, 1, 0);

var state = Keyboard.GetState();

if (state[Key.W])

//box.Move(new Vector3(-10, 0, 0));

camera.Move(new Vector2(3, 0));

//GL.Translate(-10, 0, 0);

if (state[Key.A])

//box.Move(new Vector3(0, 0, 10));

camera.Move(new Vector2(0, -3));

//GL.Translate(0, 0, 10);

if (state[Key.S])

//box.Move(new Vector3(10, 0, 0));

camera.Move(new Vector2(-3, 0));

//GL.Translate(10, 0, 0);

if (state[Key.D])

//box.Move(new Vector3(0, 0, -10));

camera.Move(new Vector2(0, 3));

//GL.Translate(0, 0, -10);

OpenTK.Input.Mouse.SetPosition(game.Width / 2f, (game.Bounds.Bottom - game.Bounds.Top) / 2f + 3);

};

game.RenderFrame += (sender, e) =>

{

// render graphics

GL.Clear(ClearBufferMask.DepthBufferBit | ClearBufferMask.ColorBufferBit | ClearBufferMask.StencilBufferBit);

camera.Paint();

light.Show1();

if (light.t1)

{

light.Print1();

light.Turn1(new Vector2(3.14f / 75, 0));

}

light.Show2();

box.Print(textureID);

cut.Print();

GL.Color3(Color.Black);

GL.Begin(PrimitiveType.Lines);

GL.Vertex3(0, 0, 0);

GL.Vertex3(100, 0, 0);

GL.Vertex3(0, 0, 0);

GL.Vertex3(0, 100, 0);

GL.Vertex3(0, 0, 0);

GL.Vertex3(0, 0, 100);

GL.End();

game.SwapBuffers();

};

//60 кадров в сек

game.Run(60.0);

}

}

}

class Oper

{

//Норма по осям X, Z

public static double Norm2D(Vector3 p)

{

return Math.Sqrt(Math.Pow(p.X, 2) + Math.Pow(p.Z, 2));

}

public static double Norm3D(Vector3 p)

{

return Math.Sqrt(Math.Pow(p.X, 2) + Math.Pow(p.Y, 2) + Math.Pow(p.Z, 2));

}

public static Vector3 Normal(Vector3 a, Vector3 b, Vector3 c)

{

b = b - a;

c = a - c;

return new Vector3(b.Y \* c.Z - b.Z \* c.Y, b.Z \* c.X - b.X \* c.Z, b.X \* c.Y - b.Y \* c.X);

}

}

}