Министерство образования и науки Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение науки и высшего образования**

«Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра информационных систем и программной инженерии

**Лабораторная работа № 14**

**по дисциплине**

**«Программирование компьютерной графики»**

**Тема: «Описание 3D объектов»**

Выполнил:

ст. гр. ПРИ-117

Хлызова В.Г.

Принял:

Жигалов И.Е.

Владимир, 2021

ЦЕЛЬ РАБОТЫ

Изучение методов описания графических изображений, состоящих из трехмерных объектов, приобретение навыков использования соответствующих алгоритмов при составлении графических программ.

ЗАДАНИЕ

1. Ознакомиться по методическим указаниям и литературе с теоретическим материалом.

2. Выполнить действия, приведенные в разделе 14.2. При разработке программы имя проекта, создаваемого в MS Visual Studio, должно содержать фамилию студента и группу (например, Ivanov\_Ivan\_ISG\_105\_lab\_1).

3. При выполнении 14.2 реализовать программу создания сцены, содержащей ландшафт минимум из 5 подобъектов (выполнены и экспортированы из 3d studio max). Все объекты должны быть протекстурированы методом наложения текстурной развертки. Помимо ландшафта загрузить и разместить в сцене еще 3 протекстурированные модели. Тематика сцены определяется вариантом задания.

4. Представить визуализацию (скриншоты) моделей в 3d studio max, а также полученную визуализацию в реализованной программе.

ХОД РАБОТЫ

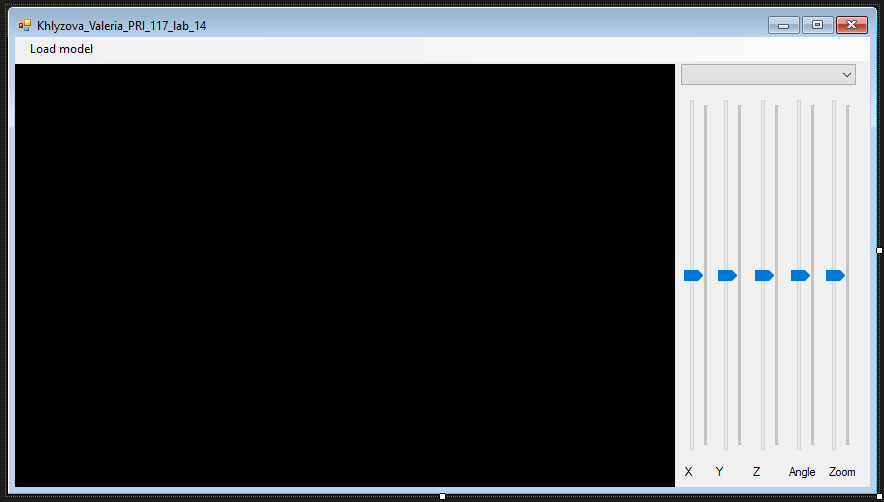


Рисунок 1 - Конструктор формы.

Обработчик формы:

using System;

using System.Windows.Forms;

using Tao.DevIl;

using Tao.FreeGlut;

using Tao.OpenGl;

namespace Khlyzova\_Valeria\_PRI\_117\_lab\_14

{

public partial class Form1 : Form

{

double a = 0, b = 0, c = -5, d = 0, zoom = 1;

int os\_x = 1, os\_y = 0, os\_z = 0;

anModelLoader Model = null;

public Form1()

{

InitializeComponent();

AnT.InitializeContexts();

}

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

{

Glut.glutInit();

Glut.glutInitDisplayMode(Glut.GLUT\_RGB | Glut.GLUT\_DOUBLE | Glut.GLUT\_DEPTH);

Il.ilInit();

Il.ilEnable(Il.IL\_CONV\_PAL);

Gl.glClearColor(0, 0, 0, 1);

Gl.glViewport(0, 0, AnT.Width, AnT.Height);

Gl.glMatrixMode(Gl.GL\_PROJECTION);

Gl.glLoadIdentity();

Glu.gluPerspective(45, (float)AnT.Width / (float)AnT.Height, 0.1, 200);

Gl.glMatrixMode(Gl.GL\_MODELVIEW);

Gl.glLoadIdentity();

Gl.glEnable(Gl.GL\_DEPTH\_TEST);

Gl.glEnable(Gl.GL\_LIGHTING);

Gl.glEnable(Gl.GL\_LIGHT0);

Gl.glBlendFunc(Gl.GL\_SRC\_ALPHA, Gl.GL\_ONE\_MINUS\_SRC\_ALPHA);

Gl.glEnable(Gl.GL\_BLEND);

Gl.glEnable(Gl.GL\_LINE\_SMOOTH);

Gl.glLineWidth(1.0f);

comboBox1.SelectedIndex = 0;

openFileDialog1.Filter = "ase files (\*.ase)|\*.ase|All files (\*.\*)|\*.\*";

}

private void Draw()

{

Gl.glClear(Gl.GL\_COLOR\_BUFFER\_BIT | Gl.GL\_DEPTH\_BUFFER\_BIT);

Gl.glLoadIdentity();

Gl.glColor3i(255, 0, 0);

Gl.glPushMatrix();

Gl.glTranslated(a, b, c);

Gl.glRotated(d, os\_x, os\_y, os\_z);

Gl.glScaled(zoom, zoom, zoom);

if (Model != null)

Model.DrawModel();

Gl.glPopMatrix();

Gl.glFlush();

AnT.Invalidate();

}

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

switch (comboBox1.SelectedIndex)

{

case 0:

{

os\_x = 1;

os\_y = 0;

os\_z = 0;

break;

}

case 1:

{

os\_x = 0;

os\_y = 1;

os\_z = 0;

break;

}

case 2:

{

os\_x = 0;

os\_y = 0;

os\_z = 1;

break;

}

}

Draw();

}

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

{

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

Model = new anModelLoader();

Model.LoadModel(openFileDialog1.FileName);

RenderTimer.Start();

}

}

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

{

Draw();

}

private void trackBar1\_Scroll(object sender, EventArgs e)

{

a = (double)trackBar1.Value / 1000.0;

label1.Text = a.ToString();

Draw();

}

private void trackBar2\_Scroll(object sender, EventArgs e)

{

b = (double)trackBar2.Value / 1000.0;

label2.Text = b.ToString();

Draw();

}

private void trackBar3\_Scroll(object sender, EventArgs e)

{

c = (double)trackBar3.Value / 1000.0;

label3.Text = c.ToString();

Draw();

}

private void trackBar4\_Scroll(object sender, EventArgs e)

{

d = (double)trackBar4.Value;

label4.Text = d.ToString();

Draw();

}

private void trackBar5\_Scroll(object sender, EventArgs e)

{

zoom = (double)trackBar5.Value / 1000.0;

label5.Text = zoom.ToString();

Draw();

}

}

}

Класс LIMB:

namespace Khlyzova\_Valeria\_PRI\_117\_lab\_14

{

class LIMB

{

public LIMB(int a, int b)

{

if (temp[0] == 0)

temp[0] = 1;

VandF[0] = a;

VandF[1] = b;

memcompl();

}

public int Itog;

public float[,] vert;

public int[,] face;

public float[,] t\_vert;

public int[,] t\_face;

private int MaterialNom = -1;

public int[] VandF = new int[4];

private int[] temp = new int[2];

private bool ModelHasTexture = false;

public bool NeedTexture()

{

return ModelHasTexture;

}

public void SetMaterialNom(int new\_nom)

{

MaterialNom = new\_nom;

if (MaterialNom > -1)

ModelHasTexture = true;

}

public void createTextureVertexMem(int a)

{

VandF[2] = a;

t\_vert = new float[3, VandF[2]];

}

public void createTextureFaceMem(int b)

{

VandF[3] = b;

t\_face = new int[3, VandF[3]];

}

private void memcompl()

{

vert = new float[3, VandF[0]];

face = new int[3, VandF[1]];

}

public int GetTextureNom()

{

return MaterialNom;

}

}

}

Класс anModelLoader:

using Tao.OpenGl;

namespace Khlyzova\_Valeria\_PRI\_117\_lab\_14

{

class anModelLoader

{

public string FName = "";

private bool isLoad = false;

private int count\_limbs;

private int mat\_nom = 0;

private int thisList = 0;

private int GlobalStringFrom = 0;

LIMB[] limbs = null;

TexturesForObjects[] text\_objects = null;

Model\_Prop coord = new Model\_Prop();

public void SetMinimum(float x, float y, float z)

{

coord.minimum[0] = x;

coord.minimum[1] = y;

coord.minimum[2] = z;

}

public void SetMaximum(float x, float y, float z)

{

coord.maximum[0] = x;

coord.maximum[1] = y;

coord.maximum[2] = z;

}

public void SetAbsCoords(float x, float y, float z)

{

coord.pos\_abs[0] = x;

coord.pos\_abs[1] = y;

coord.pos\_abs[2] = z;

}

public int RotateModel(int os, float target, float step)

{

if ((coord.rotating\_angles[os] - target) > 0)

{

coord.rotating\_angles[os] -= step;

if (coord.rotating\_angles[os] < target)

{

coord.rotating\_angles[os] = target;

return -1;

}

}

else

{

coord.rotating\_angles[os] += step;

if (coord.rotating\_angles[os] > target)

{

coord.rotating\_angles[os] = target;

return -1;

}

}

return 0;

}

public int MoveModel(int os, float target, float step)

{

if (step == 0)

return -1;

float real\_target = target;

if ((coord.pos\_abs[os] - real\_target) > 0)

{

if (coord.pos\_abs[os] - step >= coord.minimum[os])

{

coord.pos\_abs[os] -= step;

if (coord.pos\_abs[os] < real\_target)

{

coord.pos\_abs[os] = real\_target;

return -1;

}

return 0;

}

else

{

coord.pos\_abs[os] = coord.minimum[os];

return -1;

}

}

if ((coord.pos\_abs[os] - real\_target) < 0)

{

if (coord.pos\_abs[os] + step <= coord.maximum[os])

{

coord.pos\_abs[os] += step;

if (coord.pos\_abs[os] > real\_target)

{

coord.pos\_abs[os] = real\_target;

return -1;

}

return 0;

}

else

{

coord.pos\_abs[os] = coord.maximum[os];

return -1;

}

}

if ((coord.pos\_abs[os] - real\_target) == 0)

return -1;

return 0;

}

public int LoadModel(string FileName)

{

limbs = new LIMB[256];

int limb\_ = -1;

FName = FileName;

StreamReader sw = File.OpenText(FileName);

string a\_buff = "";

string b\_buff = "";

string c\_buff = "";

int ver = 0, fac = 0;

while ((a\_buff = sw.ReadLine()) != null)

{

b\_buff = GetFirstWord(a\_buff, 0);

if (b\_buff[0] == '\*')

{

switch (b\_buff)

{

case "\*MATERIAL\_COUNT":

{

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

int mat = System.Convert.ToInt32(c\_buff);

text\_objects = new TexturesForObjects[mat];

continue;

}

case "\*MATERIAL\_REF":

{

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

int mat\_ref = System.Convert.ToInt32(c\_buff);

limbs[limb\_].SetMaterialNom(mat\_ref);

continue;

}

case "\*MATERIAL":

{

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

mat\_nom = System.Convert.ToInt32(c\_buff);

continue;

}

case "\*GEOMOBJECT":

{

limb\_++;

continue;

}

case "\*MESH\_NUMVERTEX":

{

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

ver = System.Convert.ToInt32(c\_buff);

continue;

}

case "\*BITMAP":

{

c\_buff = "";

for (int ax = GlobalStringFrom + 2; ax < a\_buff.Length - 1; ax++)

c\_buff += a\_buff[ax];

text\_objects[mat\_nom] = new TexturesForObjects();

text\_objects[mat\_nom].LoadTextureForModel(c\_buff);

continue;

}

case "\*MESH\_NUMTVERTEX":

{

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

if (limbs[limb\_] != null)

{

limbs[limb\_].createTextureVertexMem(System.Convert.ToInt32(c\_buff));

}

continue;

}

case "\*MESH\_NUMTVFACES":

{

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

if (limbs[limb\_] != null)

{

limbs[limb\_].createTextureFaceMem(System.Convert.ToInt32(c\_buff));

}

continue;

}

case "\*MESH\_NUMFACES":

{

c\_buff = GetFirstWord(a\_buff, GlobalStringFrom);

fac = System.Convert.ToInt32(c\_buff);

if (limb\_ > -1 && ver > -1 && fac > -1)

{

limbs[limb\_] = new LIMB(ver, fac);

}

else

{

return -1;

}

continue;

}

case "\*MESH\_VERTEX":

{

if (limb\_ == -1)

return -2;

if (limbs[limb\_] == null)

return -3;

string a1 = "", a2 = "", a3 = "", a4 = "";

a1 = GetFirstWord(a\_buff, GlobalStringFrom);

a2 = GetFirstWord(a\_buff, GlobalStringFrom);

a3 = GetFirstWord(a\_buff, GlobalStringFrom);

a4 = GetFirstWord(a\_buff, GlobalStringFrom);

int NomVertex = System.Convert.ToInt32(a1);

a2 = a2.Replace('.', ',');

a3 = a3.Replace('.', ',');

a4 = a4.Replace('.', ',');

limbs[limb\_].vert[0, NomVertex] = (float)System.Convert.ToDouble(a2); // x

limbs[limb\_].vert[1, NomVertex] = (float)System.Convert.ToDouble(a3); // y

limbs[limb\_].vert[2, NomVertex] = (float)System.Convert.ToDouble(a4); // z

continue;

}

case "\*MESH\_FACE":

{

if (limb\_ == -1)

return -2;

if (limbs[limb\_] == null)

return -3;

string a1 = "", a2 = "", a3 = "", a4 = "", a5 = "", a6 = "", a7 = "";

a1 = GetFirstWord(a\_buff, GlobalStringFrom);

a2 = GetFirstWord(a\_buff, GlobalStringFrom);

a3 = GetFirstWord(a\_buff, GlobalStringFrom);

a4 = GetFirstWord(a\_buff, GlobalStringFrom);

a5 = GetFirstWord(a\_buff, GlobalStringFrom);

a6 = GetFirstWord(a\_buff, GlobalStringFrom);

a7 = GetFirstWord(a\_buff, GlobalStringFrom);

int NomFace = System.Convert.ToInt32(a1.Replace(':', '\0'));

limbs[limb\_].face[0, NomFace] = System.Convert.ToInt32(a3);

limbs[limb\_].face[1, NomFace] = System.Convert.ToInt32(a5);

limbs[limb\_].face[2, NomFace] = System.Convert.ToInt32(a7);

continue;

}

case "\*MESH\_TVERT":

{

if (limb\_ == -1)

return -2;

if (limbs[limb\_] == null)

return -3;

string a1 = "", a2 = "", a3 = "", a4 = "";

a1 = GetFirstWord(a\_buff, GlobalStringFrom);

a2 = GetFirstWord(a\_buff, GlobalStringFrom);

a3 = GetFirstWord(a\_buff, GlobalStringFrom);

a4 = GetFirstWord(a\_buff, GlobalStringFrom);

int NomVertex = System.Convert.ToInt32(a1);

a2 = a2.Replace('.', ',');

a3 = a3.Replace('.', ',');

a4 = a4.Replace('.', ',');

limbs[limb\_].t\_vert[0, NomVertex] = (float)System.Convert.ToDouble(a2);

limbs[limb\_].t\_vert[1, NomVertex] = (float)System.Convert.ToDouble(a3);

limbs[limb\_].t\_vert[2, NomVertex] = (float)System.Convert.ToDouble(a4);

continue;

}

case "\*MESH\_TFACE":

{

if (limb\_ == -1)

return -2;

if (limbs[limb\_] == null)

return -3;

string a1 = "", a2 = "", a3 = "", a4 = "";

a1 = GetFirstWord(a\_buff, GlobalStringFrom);

a2 = GetFirstWord(a\_buff, GlobalStringFrom);

a3 = GetFirstWord(a\_buff, GlobalStringFrom);

a4 = GetFirstWord(a\_buff, GlobalStringFrom);

int NomFace = System.Convert.ToInt32(a1);

limbs[limb\_].t\_face[0, NomFace] = System.Convert.ToInt32(a2);

limbs[limb\_].t\_face[1, NomFace] = System.Convert.ToInt32(a3);

limbs[limb\_].t\_face[2, NomFace] = System.Convert.ToInt32(a4);

continue;

}

}

}

}

count\_limbs = limb\_;

int nom\_l = Gl.glGenLists(1);

thisList = nom\_l;

Gl.glNewList(nom\_l, Gl.GL\_COMPILE);

CreateList();

Gl.glEndList();

isLoad = true;

return 0;

}

private void CreateList()

{

Gl.glPushMatrix();

for (int l = 0; l <= count\_limbs; l++)

{

if (limbs[l].NeedTexture())

if (text\_objects[limbs[l].GetTextureNom()] != null)

{

Gl.glEnable(Gl.GL\_TEXTURE\_2D);

uint nn = text\_objects[limbs[l].GetTextureNom()].GetTextureObj();

Gl.glBindTexture(Gl.GL\_TEXTURE\_2D, nn);

}

Gl.glEnable(Gl.GL\_NORMALIZE);

Gl.glBegin(Gl.GL\_TRIANGLES);

for (int i = 0; i < limbs[l].VandF[1]; i++)

{

float x1, x2, x3, y1, y2, y3, z1, z2, z3 = 0;

x1 = limbs[l].vert[0, limbs[l].face[0, i]];

x2 = limbs[l].vert[0, limbs[l].face[1, i]];

x3 = limbs[l].vert[0, limbs[l].face[2, i]];

y1 = limbs[l].vert[1, limbs[l].face[0, i]];

y2 = limbs[l].vert[1, limbs[l].face[1, i]];

y3 = limbs[l].vert[1, limbs[l].face[2, i]];

z1 = limbs[l].vert[2, limbs[l].face[0, i]];

z2 = limbs[l].vert[2, limbs[l].face[1, i]];

z3 = limbs[l].vert[2, limbs[l].face[2, i]];

float n1 = (y2 - y1) \* (z3 - z1) - (y3 - y1) \* (z2 - z1);

float n2 = (z2 - z1) \* (x3 - x1) - (z3 - z1) \* (x2 - x1);

float n3 = (x2 - x1) \* (y3 - y1) - (x3 - x1) \* (y2 - y1);

Gl.glNormal3f(n1, n2, n3);

if (limbs[l].NeedTexture() && (limbs[l].t\_vert != null) && (limbs[l].t\_face != null))

{

Gl.glTexCoord2f(limbs[l].t\_vert[0, limbs[l].t\_face[0, i]], limbs[l].t\_vert[1, limbs[l].t\_face[0, i]]);

Gl.glVertex3f(x1, y1, z1);

Gl.glTexCoord2f(limbs[l].t\_vert[0, limbs[l].t\_face[1, i]], limbs[l].t\_vert[1, limbs[l].t\_face[1, i]]);

Gl.glVertex3f(x2, y2, z2);

Gl.glTexCoord2f(limbs[l].t\_vert[0, limbs[l].t\_face[2, i]], limbs[l].t\_vert[1, limbs[l].t\_face[2, i]]);

Gl.glVertex3f(x3, y3, z3);

}

else

{

Gl.glVertex3f(x1, y1, z1);

Gl.glVertex3f(x2, y2, z2);

Gl.glVertex3f(x3, y3, z3);

}

}

Gl.glEnd();

Gl.glDisable(Gl.GL\_NORMALIZE);

Gl.glDisable(Gl.GL\_TEXTURE\_2D);

}

Gl.glPopMatrix();

}

private string GetFirstWord(string word, int from)

{

char a = word[from];

string res\_buff = "";

int L = word.Length;

if (word[from] == ' ' || word[from] == '\t')

{

int ax = 0;

for (ax = from; ax < L; ax++)

{

a = word[ax];

if (a != ' ' && a != '\t')

break;

}

if (ax == L)

return res\_buff;

else

from = ax;

}

int bx = 0;

for (bx = from; bx < L; bx++)

{

if (word[bx] == ' ' || word[bx] == '\t')

break;

res\_buff += word[bx];

}

if (bx == L)

bx--;

GlobalStringFrom = bx;

return res\_buff;

}

public void DrawModel()

{

if (!isLoad)

return;

Gl.glPushMatrix();

Gl.glScalef(0.05f, 0.05f, 0.05f);

//CreateList();

Gl.glCallList(thisList);

Gl.glPopMatrix();

}

}

}

Класс Model\_Prop:

namespace Khlyzova\_Valeria\_PRI\_117\_lab\_14

{

class Model\_Prop

{

public Model\_Prop()

{

pos\_abs[0] = 0;

pos\_abs[1] = 0;

pos\_abs[2] = 0;

maximum[0] = 0;

maximum[1] = 0;

maximum[2] = 0;

minimum[0] = 0;

minimum[1] = 0;

minimum[2] = 0;

rotating\_angles[0] = 0;

rotating\_angles[1] = 0;

rotating\_angles[2] = 0;

}

public float[] pos\_abs = new float[3];

public float[] maximum = new float[3];

public float[] minimum = new float[3];

public float[] rotating\_angles = new float[3];

}

}

Класс TexturesForObjects:

using System;

using System.Collections.Generic;

using System.IO;

using Tao.DevIl;

using Tao.OpenGl;

namespace Khlyzova\_Valeria\_PRI\_117\_lab\_14

{

class TexturesForObjects

{

private string texture\_name = "";

private int imageId = 0;

private uint mGlTextureObject = 0;

Dictionary<string, string> planets = new Dictionary<string, string>

{

{"sun", "texture\\sun.jpg"},

{"mercury", "texture\\mercury.jpg"},

{"venus", "texture\\venus.jpg"},

{"mars", "texture\\mars.jpg"},

{"uranus", "texture\\uranus.jpg"},

{"saturn", "texture\\saturn.jpg"},

{"neptune", "texture\\neptune.jpg"},

{"jupiter", "texture\\jupiter.jpg"},

{"earth", "texture\\earth.jpg"},

{"roket", "texture\\roket\_1.jpg"}

};

public uint GetTextureObj()

{

return mGlTextureObject;

}

public void LoadTextureForModel(string FileName)

{

texture\_name = FileName;

Il.ilGenImages(1, out imageId);

Il.ilBindImage(imageId);

string fileName = "";

foreach (KeyValuePair<string, string> planet in planets)

{

if (texture\_name.Contains(planet.Key)) fileName = planet.Value;

}

if (texture\_name.Contains("roket"))

Il.ilEnable(Il.IL\_ORIGIN\_SET);

else

Il.ilEnable(Il.IL\_CONV\_PAL);

string url = Path.GetFullPath(fileName);

if (Il.ilLoadImage(url))

{

int width = Il.ilGetInteger(Il.IL\_IMAGE\_WIDTH);

int height = Il.ilGetInteger(Il.IL\_IMAGE\_HEIGHT);

int bitspp = Il.ilGetInteger(Il.IL\_IMAGE\_BITS\_PER\_PIXEL);

switch (bitspp)

{

case 24:

mGlTextureObject = MakeGlTexture(Gl.GL\_RGB, Il.ilGetData(), width, height);

break;

case 32:

mGlTextureObject = MakeGlTexture(Gl.GL\_RGBA, Il.ilGetData(), width, height);

break;

}

Il.ilDeleteImages(1, ref imageId);

}

}

private static uint MakeGlTexture(int Format, IntPtr pixels, int w, int h)

{

uint texObject;

Gl.glGenTextures(1, out texObject);

Gl.glPixelStorei(Gl.GL\_UNPACK\_ALIGNMENT, 1);

Gl.glBindTexture(Gl.GL\_TEXTURE\_2D, texObject);

Gl.glTexParameteri(Gl.GL\_TEXTURE\_2D, Gl.GL\_TEXTURE\_WRAP\_S, Gl.GL\_REPEAT);

Gl.glTexParameteri(Gl.GL\_TEXTURE\_2D, Gl.GL\_TEXTURE\_WRAP\_T, Gl.GL\_REPEAT);

Gl.glTexParameteri(Gl.GL\_TEXTURE\_2D, Gl.GL\_TEXTURE\_MAG\_FILTER, Gl.GL\_LINEAR);

Gl.glTexParameteri(Gl.GL\_TEXTURE\_2D, Gl.GL\_TEXTURE\_MIN\_FILTER, Gl.GL\_LINEAR);

Gl.glTexEnvf(Gl.GL\_TEXTURE\_ENV, Gl.GL\_TEXTURE\_ENV\_MODE, Gl.GL\_REPLACE);

switch (Format)

{

case Gl.GL\_RGB:

Gl.glTexImage2D(Gl.GL\_TEXTURE\_2D, 0, Gl.GL\_RGB, w, h, 0, Gl.GL\_RGB, Gl.GL\_UNSIGNED\_BYTE, pixels);

break;

case Gl.GL\_RGBA:

Gl.glTexImage2D(Gl.GL\_TEXTURE\_2D, 0, Gl.GL\_RGBA, w, h, 0, Gl.GL\_RGBA, Gl.GL\_UNSIGNED\_BYTE, pixels);

break;

}

return texObject;

}

}

}

Результат выполнения

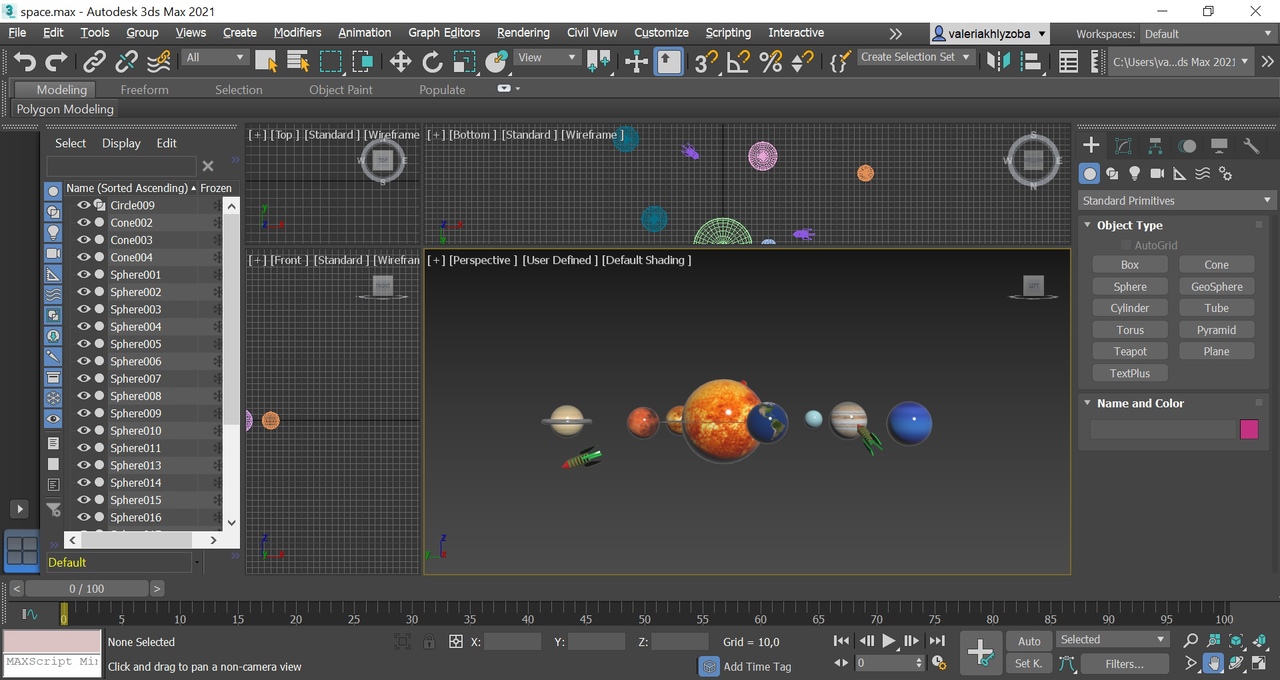


Рисунок 2 - Сцена в 3ds max.

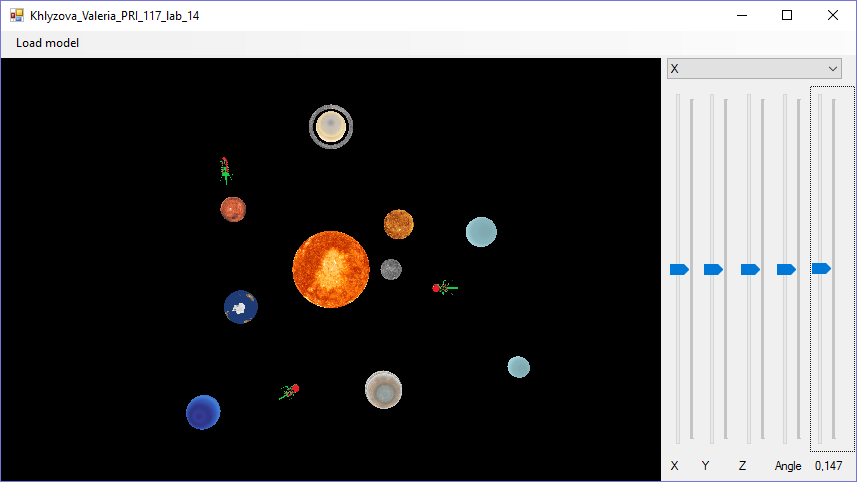


Рисунок 3 - Результат работы программы.

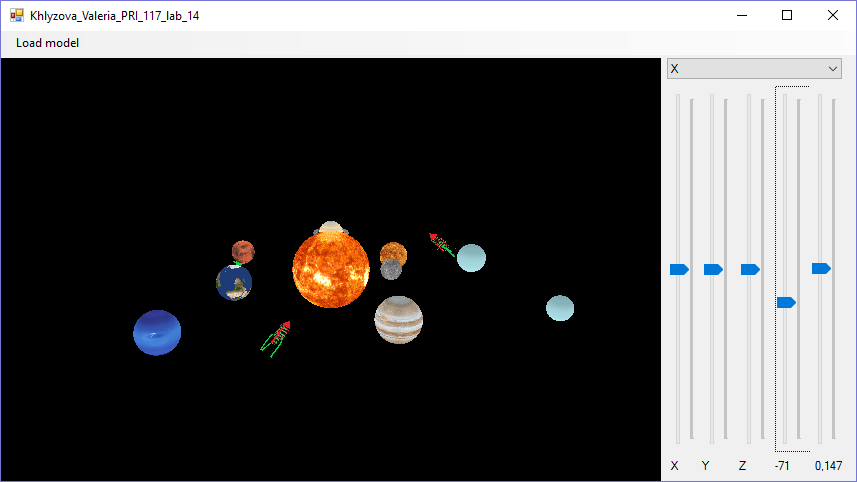


Рисунок 4 - Результат работы программы.

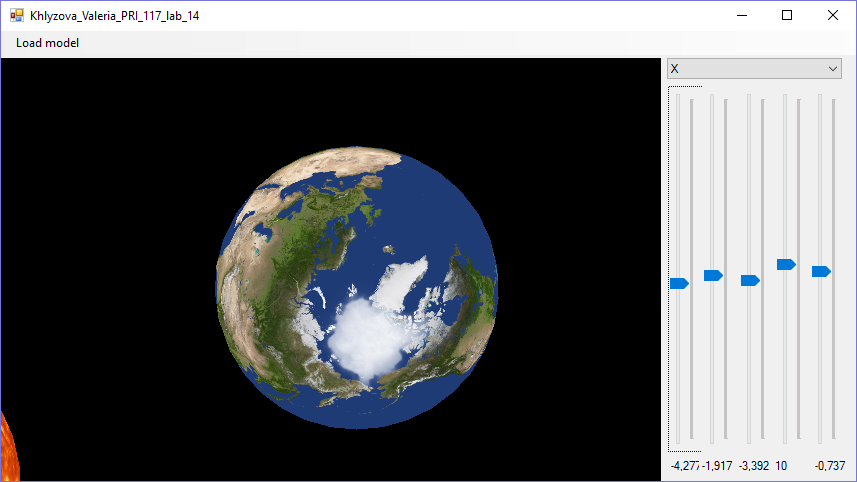


Рисунок 5 - Результат работы программы.

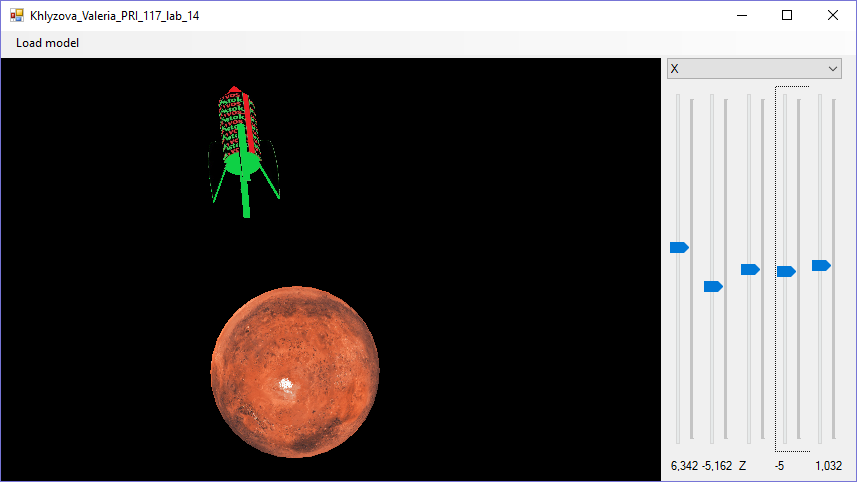


Рисунок 6 - Результат работы программы.

ВЫВОД

В ходе выполнения лабораторной работы были изучены методы описания графических изображений, состоящих из трехмерных объектов, приобретены навыки использования соответствующих алгоритмов при составлении графических программ.