**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №3**

**по дисциплине «Компьютерная графика»**

Тема: **Формирования различных поверхностей с использованием ее пространственного разворота и ортогонального проецирования на плоскость при ее визуализации (выводе на экран дисплея)**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 8362 |  | Ларионова Е.Е.  Матвеев Н.Д. |
| Преподаватель |  | Матвеева И. В. |

Санкт-Петербург

2021

**ЗАДАНИЕ**

Сформировать билинейную поверхность на основе произвольного задания ее четырех угловых точек. Обеспечить ее поворот относительно осей X и Y.

**РЕАЛИЗАЦИЯ ПРОГРАММЫ**

При запуске программы открываются 2 окна «MainWindow» и «Form». В «MainWindow» задаются координаты точек. В «Form» служит для отрисовки билинейной поверхности (Рисунок 1 и Рисунок 2).

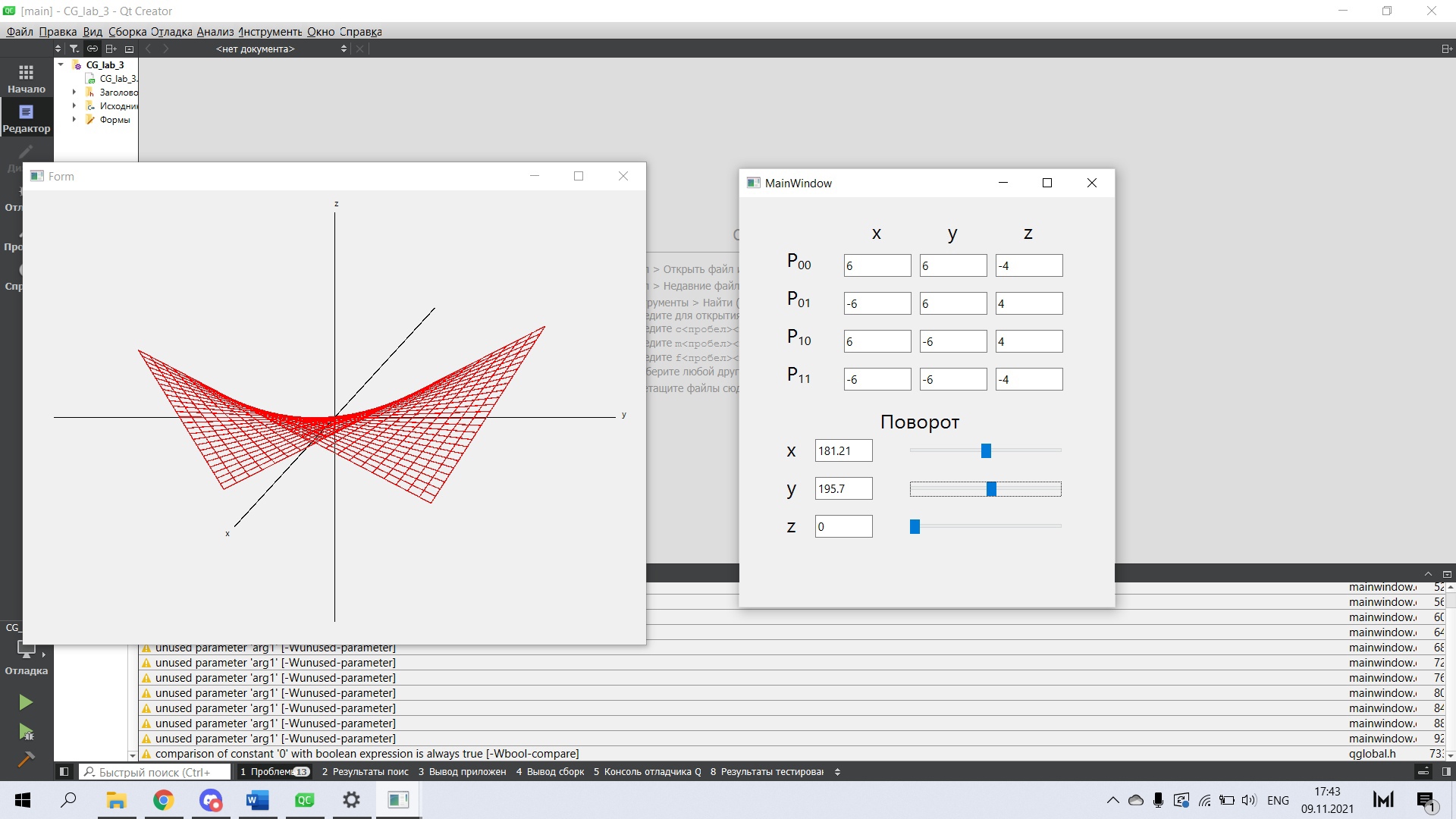


Рисунок 1 – Окно «MainWindow»

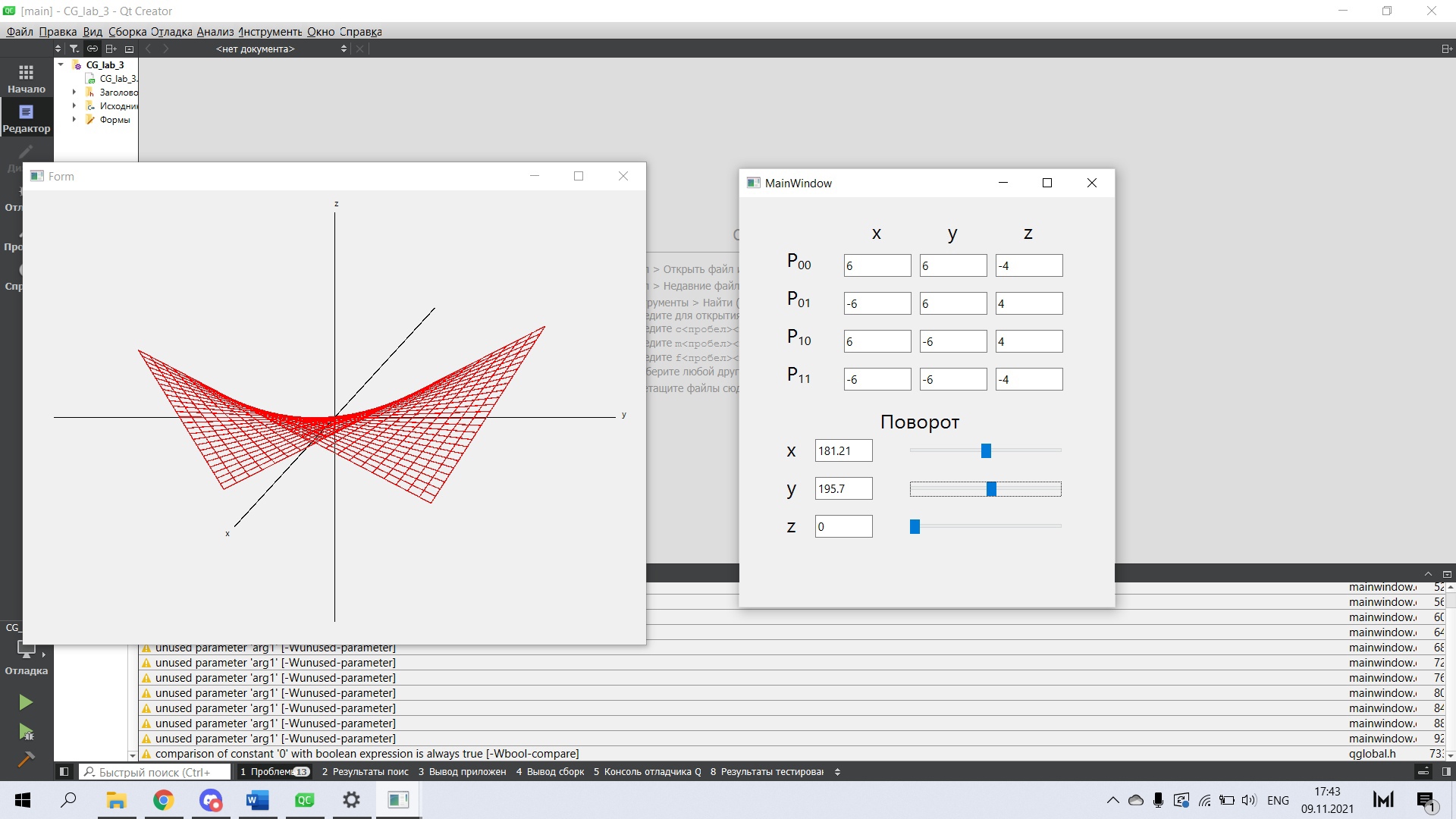


Рисунок 1 – Окно «Form»

**ПРИЛОЖЕНИЕ 1 – КОД ПРОГРАММЫ**

Файл main.cpp

#include "mainwindow.h"

#include <application.h>

int main(int argc, char \*argv[])

{

Application a(argc, *argv*);

return a.exec();

}

Файл mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~***MainWindow***()

{

delete ui;

}

control\_state MainWindow::**get\_state**()

{

control\_state tmp;

tmp.x\_rot = ui->lineEdit\_x\_rot->text().toDouble();

tmp.y\_rot = ui->lineEdit\_y\_rot->text().toDouble();

tmp.z\_rot = ui->lineEdit\_z\_rot->text().toDouble();

tmp.p[0].x = ui->lineEdit\_x\_1->text().toDouble();

tmp.p[0].y = ui->lineEdit\_y\_1->text().toDouble();

tmp.p[0].z = ui->lineEdit\_z\_1->text().toDouble();

tmp.p[1].x = ui->lineEdit\_x\_2->text().toDouble();

tmp.p[1].y = ui->lineEdit\_y\_2->text().toDouble();

tmp.p[1].z = ui->lineEdit\_z\_2->text().toDouble();

tmp.p[2].x = ui->lineEdit\_x\_3->text().toDouble();

tmp.p[2].y = ui->lineEdit\_y\_3->text().toDouble();

tmp.p[2].z = ui->lineEdit\_z\_3->text().toDouble();

tmp.p[3].x = ui->lineEdit\_x\_4->text().toDouble();

tmp.p[3].y = ui->lineEdit\_y\_4->text().toDouble();

tmp.p[3].z = ui->lineEdit\_z\_4->text().toDouble();

return tmp;

}

void MainWindow::**send\_state**()

{

emit send\_control(get\_state());

}

//дикая пачка служебных слотов

void MainWindow::**on\_lineEdit\_x\_1\_textChanged**(const QString &arg1)

{

send\_state();

}

void MainWindow::**on\_lineEdit\_y\_1\_textChanged**(const QString &arg1)

{

send\_state();

}

void MainWindow::**on\_lineEdit\_z\_1\_textChanged**(const QString &arg1)

{

send\_state();

}

void MainWindow::**on\_lineEdit\_x\_2\_textChanged**(const QString &arg1)

{

send\_state();

}

void MainWindow::**on\_lineEdit\_y\_2\_textChanged**(const QString &arg1)

{

send\_state();

}

void MainWindow::**on\_lineEdit\_z\_2\_textChanged**(const QString &arg1)

{

send\_state();

}

void MainWindow::**on\_lineEdit\_x\_3\_textChanged**(const QString &arg1)

{

send\_state();

}

void MainWindow::**on\_lineEdit\_y\_3\_textChanged**(const QString &arg1)

{

send\_state();

}

void MainWindow::**on\_lineEdit\_z\_3\_textChanged**(const QString &arg1)

{

send\_state();

}

void MainWindow::**on\_lineEdit\_x\_4\_textChanged**(const QString &arg1)

{

send\_state();

}

void MainWindow::**on\_lineEdit\_y\_4\_textChanged**(const QString &arg1)

{

send\_state();

}

void MainWindow::**on\_lineEdit\_z\_4\_textChanged**(const QString &arg1)

{

send\_state();

}

void MainWindow::**on\_lineEdit\_x\_rot\_textChanged**(const QString &arg1)

{

int tmp = static\_cast<int>(arg1.toDouble()\*100)%36001;

ui->horizontalSlider\_x->setValue(tmp);

ui->lineEdit\_x\_rot->setText(QString::number(0.01\*tmp));

send\_state();

}

void MainWindow::**on\_horizontalSlider\_x\_valueChanged**(int value)

{

ui->lineEdit\_x\_rot->setText(QString::number(0.01\*value));

send\_state();

}

void MainWindow::**on\_lineEdit\_y\_rot\_textChanged**(const QString &arg1)

{

int tmp = static\_cast<int>(arg1.toDouble()\*100)%36001;

ui->horizontalSlider\_y->setValue(tmp);

ui->lineEdit\_y\_rot->setText(QString::number(0.01\*tmp));

send\_state();

}

void MainWindow::**on\_horizontalSlider\_y\_valueChanged**(int value)

{

ui->lineEdit\_y\_rot->setText(QString::number(0.01\*value));

send\_state();

}

void MainWindow::**on\_lineEdit\_z\_rot\_textChanged**(const QString &arg1)

{

int tmp = static\_cast<int>(arg1.toDouble()\*100)%36001;

ui->horizontalSlider\_z->setValue(tmp);

ui->lineEdit\_z\_rot->setText(QString::number(0.01\*tmp));

send\_state();

}

void MainWindow::**on\_horizontalSlider\_z\_valueChanged**(int value)

{

ui->lineEdit\_z\_rot->setText(QString::number(0.01\*value));

send\_state();

}

Файл drawwindow.cpp

#include "drawwindow.h"

#include "ui\_drawwindow.h"

DrawWindow::**DrawWindow**(QWidget \*parent) :

QWidget(*parent*),

ui(new Ui::DrawWindow)

{

ds = nullptr;

ui->setupUi(this);

}

DrawWindow::~***DrawWindow***()

{

delete ui;

}

void DrawWindow::**recive\_draw**(draw\_state \*rds)

{

ds=rds;

repaint();

}

void DrawWindow::***paintEvent*** (QPaintEvent \*event)

{

Q\_UNUSED(event);

QPainter painter(this);

QFont font;

support\_state s;

s.xred = 0.5;

s.cw = 0.5\*rect().width();

s.ch = 0.5\*rect().height();

qreal xshift = 0.08;

qreal xcw = s.cw \* xshift;

qreal xch = s.ch \* xshift;

qreal cmw = s.cw / 10;

qreal cmh = s.ch / 10;

int num = 5;

if (ds!=nullptr)

{

num = ds->max;

}

s.ew = (s.cw - cmw) / (num + 5);

s.eh = (s.ch - cmh) / (num + 5);

qreal c = s.cw>s.ch?s.ch:s.cw;

qreal cf = 0.03 \* c;

font.setPointSize(cf);

painter.setFont(font);

painter.setPen(QPen(Qt::black));

QLineF Ox(2\*s.cw-cmw-s.cw\*s.xred-xcw,0+cmh+s.ch\*s.xred-xch,0+cmw+s.cw\*s.xred+xcw,2\*s.ch-cmh-s.ch\*s.xred+xch);

QLineF Oy(0+cmw,s.ch,2\*s.cw-cmw,s.ch);

QLineF Oz(s.cw,2\*s.ch-cmh,s.cw,0+cmh);

s.ex = Ox.unitVector().p2()-Ox.unitVector().p1();

s.ey = Oy.unitVector().p2()-Oy.unitVector().p1();

s.ez = Oz.unitVector().p2()-Oz.unitVector().p1();

painter.drawLine(Ox);

painter.drawLine(Oy);

painter.drawLine(Oz);

painter.drawText(Ox.p2()+ 2\*cf\*s.ex,"x");

painter.drawText(Oy.p2()+ cf\*s.ey,"y");

painter.drawText(Oz.p2()+ cf\*s.ez,"z");

if (ds != nullptr)

{

painter.setPen(QPen(Qt::red));

for (size\_t i = 0; i < ds->num; i++)

{

for (size\_t j = 0; j < ds->num; j++)

{

if (i != ds->num-1)

{

painter.drawLine(transform(ds->d[i][j],s),transform(ds->d[i+1][j],s));

}

if (j != ds->num-1)

{

painter.drawLine(transform(ds->d[i][j],s),transform(ds->d[i][j+1],s));

}

}

}

}

}

QPointF DrawWindow::**transform**(PointF3D a, support\_state s)

{

QPointF tmp;

tmp.setX(s.cw + (a.y\*s.ey.rx()+ a.x\*s.ex.rx()\*s.xred )\*s.ew);

tmp.setY(s.ch + (a.z\*s.ez.ry()+ a.x\*s.ex.ry()\*s.xred )\*s.eh);

return tmp;

}

Файл application.cpp

#include "application.h"

#define STEP 0.04

Application::**Application**(int argc, char \*argv[])

: QApplication(*argc*,*argv*)

{

m = new MainWindow;

m->show();

d = new DrawWindow;

d->show();

connect(m,SIGNAL(send\_control(control\_state)),

this,SLOT(get\_control(control\_state)));

connect(this,SIGNAL(send\_draw(draw\_state\*)),

d,SLOT(recive\_draw(draw\_state\*)));

}

void Application::**get\_control**(control\_state cs)

{

draw\_state \*ds = new draw\_state;

double x\_ang = 2\*M\_PI\*cs.x\_rot/360;

double y\_ang = 2\*M\_PI\*cs.y\_rot/360;

double z\_ang = 2\*M\_PI\*cs.z\_rot/360;

Mmatrix Tx(3,3),Ty(3,3),Tz(3,3);

//блок с нахождением максимальной координаты

ds->max = 0;

for (size\_t i = 0; i < 4; i++)

{

if (qAbs(cs.p[i].x > ds->max))

{

ds->max = static\_cast<int>(cs.p[i].x);

}

if (qAbs(cs.p[i].y > ds->max))

{

ds->max = static\_cast<int>(cs.p[i].y);

}

if (qAbs(cs.p[i].y > ds->max))

{

ds->max = static\_cast<int>(cs.p[i].y);

}

}

//Матрица поворота по x

Tx.data[0][0] = 1;

Tx.data[0][1] = 0;

Tx.data[0][2] = 0;

Tx.data[1][0] = 0;

Tx.data[1][1] = cos(x\_ang);

Tx.data[1][2] = -sin(x\_ang);

Tx.data[2][0] = 0;

Tx.data[2][1] = sin(x\_ang);

Tx.data[2][2] = cos(x\_ang);

//Матрица поворота по y

Ty.data[0][0] = cos(y\_ang);

Ty.data[0][1] = 0;

Ty.data[0][2] = sin(y\_ang);

Ty.data[1][0] = 0;

Ty.data[1][1] = 1;

Ty.data[1][2] = 0;

Ty.data[2][0] = -sin(y\_ang);

Ty.data[2][1] = 0;

Ty.data[2][2] = cos(y\_ang);

//Матрица поворота по z

Tz.data[0][0] = cos(z\_ang);

Tz.data[0][1] = -sin(z\_ang);

Tz.data[0][2] = 0;

Tz.data[1][0] = sin(z\_ang);

Tz.data[1][1] = cos(z\_ang);

Tz.data[1][2] = 0;

Tz.data[2][0] = 0;

Tz.data[2][1] = 0;

Tz.data[2][2] = 1;

for (size\_t i = 0; i < 4; i++)

{

Mmatrix tmp(1,3),res(1,3);

tmp.data[0][0] = cs.p[i].x;

tmp.data[0][1] = cs.p[i].y;

tmp.data[0][2] = cs.p[i].z;

res = ((tmp \* Tx) \*Ty) \* Tz;

cs.p[i].x = res.data[0][0];

cs.p[i].y = res.data[0][1];

cs.p[i].z = res.data[0][2];

}

ds->num = static\_cast <size\_t> (1.0/STEP) +1;

for (size\_t i = 0; i < ds->num; i++)

{

std::vector <PointF3D> tmp;

for (size\_t j = 0; j < ds->num; j++)

{

PointF3D tmp\_p;

tmp\_p.x = cs.p[0].x\*(1-i\*STEP)\*(1-j\*STEP) + cs.p[2].x\*(1-i\*STEP)\*(j\*STEP) + cs.p[1].x\*(i\*STEP)\*(1-j\*STEP) + cs.p[3].x\*(i\*STEP)\*(j\*STEP);

tmp\_p.y = cs.p[0].y\*(1-i\*STEP)\*(1-j\*STEP) + cs.p[2].y\*(1-i\*STEP)\*(j\*STEP) + cs.p[1].y\*(i\*STEP)\*(1-j\*STEP) + cs.p[3].y\*(i\*STEP)\*(j\*STEP);

tmp\_p.z = cs.p[0].z\*(1-i\*STEP)\*(1-j\*STEP) + cs.p[2].z\*(1-i\*STEP)\*(j\*STEP) + cs.p[1].z\*(i\*STEP)\*(1-j\*STEP) + cs.p[3].z\*(i\*STEP)\*(j\*STEP);

tmp.push\_back(tmp\_p);

}

ds->d.push\_back(tmp);

}

emit(send\_draw(*ds*));

}