Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Петрозаводский государственный университет» Физико-технический институт Кафедра информационно-измерительных систем и физической электроники

ПРАКТИЧЕСКАЯ РАБОТА №1

ОТЧЁТ

по предмету «Обработка цифровых сигналов»

Частотные методы улучшения и восстановления изображений

Автор работы: студент группы 21512 Н. Ю. Новохатько \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Научный руководитель: доцент А.В. Бульба

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Петрозаводск 2020

**Результаты выполненной работы:**

*Назначение программы:*

Демонстрация обработки изображения при помощи идельного и гауссового фильтров низких частот.

*Код программы:*

*Листинг main.cpp*

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

return a.exec();

}

*Листинг mainwindow.h*

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <shumform1.h>

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

explicit **MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private:

Ui::MainWindow \*ui;

shumForm1 \*SF1;

QImage Ishodnoe;

QImage RIshodnoe;

QImage GIshodnoe;

QImage BIshodnoe;

float D0;

int iwidth;

int iheight;

void **CreateIshodnoe**();

enum **Colors**

{

COLOR\_R,

COLOR\_G,

COLOR\_B,

};

/\* Функция принимает три параметра COLOR\_R, COLOR\_G или COLOR\_B.

\* Заполняет переменные RIshodnoe,GIshodnoe,BIshodnoe

\* В зависимости от параметра

\*/

void **CreateGRB**(Colors color);

void **CreateImages**();

public slots:

void **IdealSlot**();

void **GaussSlot**();

void **showRGBVariablesForIdeal**();

void **showRGBVariablesForGauss**();

void **saveText**();

signals:

void **RGB\_ideal\_signal**(QImage clone, QImage cloneR, QImage cloneB, QImage cloneG, float D0, int width, int height);

void **RGB\_gauss\_signal**(QImage clone, QImage cloneR, QImage cloneB, QImage cloneG, float D0, int width, int height);

void **forRGB\_ideal\_signal**();

void **forRGB\_gauss\_signal**();

void **forSave\_images**(QString saveway);

};

#endif // MAINWINDOW\_H

*Листинг shumform1.h*

#ifndef SHUMFORM1\_H

#define SHUMFORM1\_H

#include <QWidget>

#include <QImage>

#include <complex>

#include <shumform2.h>

using namespace std;

namespace **Ui** {

class **shumForm1**;

}

class **shumForm1** : public QWidget

{

Q\_OBJECT

public:

explicit **shumForm1**(QWidget \*parent = nullptr);

~***shumForm1***();

private:

Ui::shumForm1 \*ui;

shumForm2 \*SF2;

enum **Images**

{

IMAGE\_CLONE,

IMAGE\_SPEKTR,

IMAGE\_CLONE1,

IMAGE\_R,

IMAGE\_G,

IMAGE\_B,

IMAGE\_ST1,

IMAGE\_ST2,

IMAGE\_ST3

};

enum **maskType**

{

Ideal,

Gauss

};

int imageWidth;

int imageHeight;

QVector <QImage> iImage;

void **SeeImages**(QImage image, Images im);

QImage **CentrImage**(QImage image);

QVector<QVector<float> > **CentrImage**(QImage image, Images im);

QVector<QVector<complex<float> > > **PryamoeFurieStroki**(QVector<QVector<float> > image);

QVector<QVector<complex<float> > > **PryamoeFurieStolbtsyi**(QVector<QVector<complex<float> > > image);

QVector <QVector <float> > **toVectorF**(QVector<QVector<complex<float> > > image);

QVector <QVector <float> > **toVectorFlog**(QVector<QVector<float> > image);

QVector <QVector <float> > **toVectorFshkala**(QVector<QVector<float> > image);

QImage **toMonoImage**(QVector<QVector<float> > image, Images im);

QImage **toStereoImage**(QVector<QVector<float> > R\_image, QVector<QVector<float> > G\_image, QVector<QVector<float> > B\_image);

QVector <QVector <float> > **Mask**(QVector <QVector <float> > image, float D0, maskType Type);

QImage **toMonoMask**(QVector<QVector<float> > image, Images im);

QImage **toStereoMask**(QVector<QVector<float> > R\_image, QVector<QVector<float> > G\_image, QVector<QVector<float> > B\_image);

QVector<QVector<float> > **idealFilter**(QVector<QVector<float> > spektr, QVector<QVector<float> > mask);

QVector<QVector<complex<float> > > **idealFilterComplex** (QVector<QVector<complex<float> > > spektr, QVector<QVector<float> > mask);

QVector<QVector<complex<float> > > **ObratnoyeFurieStroki**(QVector<QVector<complex<float> > > image);

QVector<QVector<complex<float> > > **ObratnoyeFurieStolbtsyi**(QVector<QVector<complex<float> > > image);

QVector<QVector<float> > **RealVector**(QVector<QVector<complex<float> > > image);

QVector<QVector<float> > **CentrVector**(QVector<QVector<float > > image);

QVector<QVector<float> > **Warning\_fix**(QVector<QVector<float> > image);

float **D\_u\_v**(float u, float v, float iwidth, float iheight);

float **H\_u\_v\_ideal**(float D0, float u, float v, float iwidth, float iheight);

float **H\_u\_v\_gauss**(float D0, float u, float v, float iwidth, float iheight);

float **one\_with\_N**(int n);

public slots:

void **process\_of\_ideal\_RGBsignal**(QImage Ishodnoe, QImage RIshodnoe, QImage GIshodnoe, QImage BIshodnoe, float D0, int width, int height);

void **process\_of\_gauss\_RGBsignal**(QImage Ishodnoe, QImage RIshodnoe, QImage GIshodnoe, QImage BIshodnoe, float D0, int width, int height);

void **saveImages**(QString saveway);

signals:

void **for\_form2\_signal**(QImage R\_SPEKTR, QImage G\_SPEKTR, QImage B\_SPEKTR, QImage R\_FILTER, QImage G\_FILTER, QImage B\_FILTER, QImage R\_RG, QImage G\_RG, QImage B\_RG);

void **forSave\_images**(QString saveway);

};

#endif // SHUMFORM1\_H

*Листинг shumform2.h*

#ifndef SHUMFORM2\_H

#define SHUMFORM2\_H

#include <QWidget>

namespace **Ui** {

class **shumForm2**;

}

class **shumForm2** : public QWidget

{

Q\_OBJECT

public:

explicit **shumForm2**(QWidget \*parent = nullptr);

~***shumForm2***();

private:

Ui::shumForm2 \*ui;

enum **Images**

{

IMAGE\_CLONE,

IMAGE\_SPEKTR,

IMAGE\_CLONE1,

IMAGE\_R,

IMAGE\_G,

IMAGE\_B,

IMAGE\_ST1,

IMAGE\_ST2,

IMAGE\_ST3

};

QVector <QImage> iImage;

void **SeeImages**(QImage image, Images im);

public slots:

void **process\_for\_seen\_images**(QImage R\_SPEKTR, QImage G\_SPEKTR, QImage B\_SPEKTR,

QImage R\_FILTER, QImage G\_FILTER, QImage B\_FILTER,

QImage R\_RG, QImage G\_RG, QImage B\_RG);

void **saveImages**(QString saveway);

};

#endif // SHUMFORM2\_H

*Листинг mainwindow.cpp*

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <shumform1.h>

#include <shumform2.h>

#include <QDebug>

#include <QString>

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

//Присвоение адреса объекта дочернего окна ссылке

SF1 = new shumForm1();

/\*Коннекты\*/

connect(ui->pushButton, SIGNAL(clicked() ), this, SLOT(IdealSlot() ) );

connect(ui->pushButton\_2, SIGNAL(clicked() ), this, SLOT(GaussSlot() ) );

connect(this, SIGNAL(forRGB\_ideal\_signal() ), this, SLOT(showRGBVariablesForIdeal() ) );

connect(this, SIGNAL(RGB\_ideal\_signal(QImage, QImage, QImage, QImage, float, int, int) ), SF1, SLOT(process\_of\_ideal\_RGBsignal(QImage, QImage, QImage, QImage, float, int, int) ) );

connect(this, SIGNAL(forRGB\_gauss\_signal() ), this, SLOT(showRGBVariablesForGauss() ) );

connect(this, SIGNAL(RGB\_gauss\_signal(QImage, QImage, QImage, QImage, float, int, int) ), SF1, SLOT(process\_of\_gauss\_RGBsignal(QImage, QImage, QImage, QImage, float, int, int) ) );

connect(ui->pushButton\_3, SIGNAL( clicked() ), this, SLOT( saveText() ) );

connect(this, SIGNAL( forSave\_images(QString) ), SF1, SLOT( saveImages(QString) ) );

/\*--\*/

}

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

{

delete ui;

}

void MainWindow::**saveText**()

{

QString saveway = ui->lineEdit\_3->text();

emit forSave\_images(saveway);

}

/\*Создание объекта "Исходное изображение"\*/

void MainWindow::**CreateIshodnoe**()

{

qDebug() << "This is CreateIshodnoe();";

QString imageWay = ui->lineEdit->text(); qDebug() << "imageWay = " << imageWay;

QString mask = ui->lineEdit\_2->text(); qDebug() << "mask = " << mask;

/\*Проверка пуста ли картинка, и её загрузка. Образец для вставки: D:\Obrazets\1test.bmp \*/

if (!imageWay.isEmpty())

{

QImage image(imageWay);

if (image.isNull())

{

qDebug() << "Error load image";

}

else

{

qDebug() << "Sucsessfull load image";

Ishodnoe = image;

iwidth = image.width();

iheight = image.height();

}

}

/\*Проверка пуста ли картинка, и её загрузка. Образец для вставки: D:\Obrazets\1test.bmp \*/

if (!mask.isEmpty())

{

qDebug() << "Sucsessfull load mask";

D0 = mask.toFloat();

}

else

{

qDebug() << "Error load mask";

}

}

/\* Разбиение изображения на каналы\*/

void MainWindow::**CreateGRB**(Colors color)

{

int iwidth = Ishodnoe.width(); qDebug() << "[ширина] width = " << iwidth;

int iheight = Ishodnoe.height(); qDebug() << "[высота] iheight = " << iheight;

//Создание объекта точка

QPoint p;

QColor iRgb, iclonee;

int ired, igreen, iblue;

QImage clone\_oneside = Ishodnoe;

QImage clone\_oneside\_clone = Ishodnoe;

switch (color)

{

case COLOR\_R:

qDebug() << "chosed R";

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

{

for(int j = 0; j < iwidth; j++)

{

p = QPoint(j, i);

iRgb = clone\_oneside.pixelColor(p);

ired = iRgb.red(); //qDebug() << "[] ired = " << ired;

iclonee = QColor(ired, 0, 0, 0);

clone\_oneside\_clone.setPixelColor(p, iclonee);

}

}

RIshodnoe = clone\_oneside\_clone;

qDebug() << "Created RIshodnoe";

break;

case COLOR\_G:

qDebug() << "chosed G";

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

{

for(int j = 0; j < iwidth; j++)

{

p = QPoint(j, i);

iRgb = clone\_oneside.pixelColor(p);

igreen = iRgb.green();

iclonee = QColor(0, igreen, 0, 0);

clone\_oneside\_clone.setPixelColor(p, iclonee);

}

}

GIshodnoe = clone\_oneside\_clone;

qDebug() << "Created GIshodnoe";

break;

case COLOR\_B:

qDebug() << "chosed B";

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

{

for(int j = 0; j < iwidth; j++)

{

p = QPoint(j, i);

iRgb = clone\_oneside.pixelColor(p);

iblue = iRgb.blue();

iclonee = QColor(0, 0, iblue, 0);

clone\_oneside\_clone.setPixelColor(p, iclonee);

}

}

BIshodnoe = clone\_oneside\_clone;

qDebug() << "Created BIshodnoe";

break;

}

}

void MainWindow::**CreateImages**()

{

CreateIshodnoe();

CreateGRB(COLOR\_R);

CreateGRB(COLOR\_G);

CreateGRB(COLOR\_B);

}

void MainWindow::**IdealSlot**()

{

qDebug() << "This is IdeaSlot();";

CreateImages();

SF1->show();

emit forRGB\_ideal\_signal();

}

void MainWindow::**GaussSlot**()

{

qDebug() << "This is GaussSlot();";

CreateImages();

SF1->show();

emit forRGB\_gauss\_signal();

}

void MainWindow::**showRGBVariablesForIdeal**()

{

qDebug() << "This is showRGBVariablesForIdeal;";

QImage Ishodnoe1 = this->Ishodnoe;

QImage RIshodnoe1 = this->RIshodnoe;

QImage GIshodnoe1 = this->GIshodnoe;

QImage BIshodnoe1 = this->BIshodnoe;

float D0 = this->D0;

int width = this->iwidth;

int height = this->iheight;

emit RGB\_ideal\_signal(Ishodnoe1, RIshodnoe1, GIshodnoe1, BIshodnoe1, D0, width, height);

}

void MainWindow::**showRGBVariablesForGauss**()

{

qDebug() << "This is showRGBVariablesForGauss;";

QImage Ishodnoe1 = this->Ishodnoe;

QImage RIshodnoe1 = this->RIshodnoe;

QImage GIshodnoe1 = this->GIshodnoe;

QImage BIshodnoe1 = this->BIshodnoe;

float D0 = this->D0;

int width = this->iwidth;

int height = this->iheight;

emit RGB\_gauss\_signal(Ishodnoe1, RIshodnoe1, GIshodnoe1, BIshodnoe1, D0, width, height);

}

*Листинг shumform1.cpp*

#include "shumform1.h"

#include "ui\_shumform1.h"

#include <QDebug>

#include <QVector>

#include <QtMath>

#include <QImage>

#include <QCheckBox>

#include <complex>

#include <math.h>

#include <cmath>

#define PI 3.14159265

//Т.к. функция atan выдаёт значение угла в радианах, требуется конвертация в градусы.

// 1(радиана) = 57.2958(градусов)

#define grad 57.3

shumForm1::**shumForm1**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::shumForm1)

{

ui->setupUi(this);

//Присвоение адреса объекта дочернего окна ссылке

SF2 = new shumForm2();

connect(this, SIGNAL(for\_form2\_signal(QImage, QImage, QImage, QImage, QImage, QImage, QImage, QImage, QImage) ), SF2, SLOT(process\_for\_seen\_images(QImage, QImage, QImage, QImage, QImage, QImage, QImage, QImage, QImage) ) );

// connect(this, SIGNAL( clicked() ), this, SLOT( saveText() ) );

connect(this, SIGNAL( forSave\_images(QString) ), SF2, SLOT( saveImages(QString) ) );

}

shumForm1::~***shumForm1***()

{

delete ui;

}

void shumForm1::**SeeImages**(QImage image, shumForm1::Images im)

{

QSize labelSize;

QImage imageSmall;

switch (im)

{

case IMAGE\_CLONE:

qDebug() << "image CLONE";

labelSize = ui->label->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_SPEKTR:

qDebug() << "image SPEKTR";

labelSize = ui->label\_2->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_2->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_CLONE1:

qDebug() << "image CLONE1";

labelSize = ui->label\_3->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_3->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_R:

qDebug() << "image R";

labelSize = ui->label\_4->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_4->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_G:

qDebug() << "image G";

labelSize = ui->label\_5->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_5->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_B:

qDebug() << "image B";

labelSize = ui->label\_6->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_6->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_ST1:

qDebug() << "image ST1";

labelSize = ui->label\_7->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_7->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_ST2:

qDebug() << "image ST2";

labelSize = ui->label\_8->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_8->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_ST3:

qDebug() << "image ST3";

labelSize = ui->label\_9->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_9->setPixmap(QPixmap::fromImage(imageSmall));

break;

}

}

QImage shumForm1::**CentrImage**(QImage image)

{

//Создание объекта точка

QPoint p;

QColor iRgb, iredd, igreenn, ibluee;

float ired, igreen, iblue;

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

int step;

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

{

for(int j = 0; j < iwidth; j++)

{

p = QPoint(j, i);

iRgb = image.pixelColor(p);

step = one\_with\_N(i + j);

ired = float( iRgb.red() ); //qDebug() << "[] ired = " << ired;

ired = ired \* step;

igreen = iRgb.green();

igreen = igreen \* step;

iblue = iRgb.blue();

iblue = iblue \* step;

iRgb = QColor( int(ired), int(igreen), int(iblue), 0);

image.setPixelColor(p, iRgb);

}

}

return image;

}

QVector<QVector<float> > shumForm1::**CentrImage**(QImage image, shumForm1::Images im)

{

//Создание объекта точка

QPoint p;

QColor iRgb;

float icolor;

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

/\*Двумерный массив яркостей точек\*/

QVector < QVector <float> > complex\_image;

/\*Временный массив для строки массива яркостей точек \*/

QVector <float> complex\_str;

/\*Переменная для хранения центрирующего коэффициента \*/

int step;

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

{

for(int j = 0; j < iwidth; j++)

{

p = QPoint(j, i);

iRgb = image.pixelColor(p);

step = one\_with\_N(i + j);

switch (im)

{

case IMAGE\_R:

//qDebug() << "complex R";

icolor = float( iRgb.red() );

break;

case IMAGE\_G:

//qDebug() << "complex G";

icolor = float( iRgb.green() );

break;

case IMAGE\_B:

//qDebug() << "complex B";

icolor = float( iRgb.blue() );

break;

} qDebug() << "! icolor = " << icolor;

icolor = icolor \* step; qDebug() << "step = " << step << ";icolor = " << icolor;

complex\_str.append(icolor);

}

complex\_image.append(complex\_str);

complex\_str.clear();

}

return complex\_image;

}

/\*Образец для вставки: D:\Obrazets\1test.bmp \*/

/\*Образец для вставки: D:\Obrazets\1testcut.bmp \*/

QVector<QVector<complex<float> > > shumForm1::**PryamoeFurieStroki**(QVector<QVector<float> > image)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

/\*Двумерный массив из комплексных чисел\*/

QVector < QVector < complex <float> > > complex\_image;

/\*Временный массив для строки массива комплексных чисел \*/

QVector < complex <float> > complex\_str;

QVector <float> image\_str;

/\*Временная переменная для элменента массива из комплексных чиел \*/

complex <float> complex\_el\_sum, complex\_el;

/\*Переделать для двумерного массива комплексных чисел\*/

//complex <float> \* complex\_image = new complex <float> [iwidth][iheight];

float icolor, real\_part, imag\_part;

/\*Меняем номер столбца\*/

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

{

for(int j = 0; j < iwidth; j++)

{

/\*j - в данном случае частота, k - отсчёт\*/

for (int k = 0; k < iwidth; k++)

{

image\_str = image.at(i);

icolor = image\_str.at(k); //qDebug() << "----"; qDebug() << "i=" << i <<", j=" << j << ", k=" << k << "; iwidth= " << iwidth << ";[]icolor = " << icolor;

/\*Создание комплексного значения F(u, y)\*/

real\_part = icolor \* qCos( (2 \* PI \* j \* k) / iwidth ); //qDebug() << "real\_part = " << real\_part;

imag\_part = icolor \* qSin( (2 \* PI \* j \* k) / iwidth ); //qDebug() << "imag\_part = " << imag\_part; qDebug() << "----";

complex\_el = complex <float> ( real\_part, imag\_part );

complex\_el\_sum += complex\_el;

}

/\* добавление данных в строку массива \*/

complex\_str.append( complex\_el\_sum ); qDebug() << complex\_el\_sum.real() << " + " << complex\_el\_sum.imag() << " \* i";

complex\_el\_sum = 0 + 0 \* i;

complex\_el = 0 + 0 \* i;

}

complex\_image.append( complex\_str );

complex\_str.clear();

}

qDebug() << "return complex\_image!";

return complex\_image;

}

QVector<QVector<complex<float> > > shumForm1::**PryamoeFurieStolbtsyi**(QVector<QVector<complex<float> > > image)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

int isize;

/\*Двумерный массив из комплексных чисел\*/

QVector < QVector < complex <float> > > complex\_image;

/\*Временный массив для строки массива комплексных чисел \*/

QVector < complex <float> > complex\_str, image\_str;

/\*Временная переменная для элменента массива из комплексных чиел \*/

complex <float> image\_el, complex\_el\_sum, complex\_el;

float image\_real\_part, image\_imag\_part;

float complex\_real\_part, complex\_imag\_part;

float icos, isin;

/\*Проходим по всем элементам строки\*/

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

{

/\*Меняем номер столбца\*/

for (int j = 0; j < iheight; j++)

{

/\*Суммирование элементов массива по столбцам\*/

for (int k = 0; k < iheight; k++)

{

/\*Считывание элемента из image\*/

image\_str = image.at(k);

image\_el = image\_str.at(i);

/\*\_\_\*/

image\_real\_part = image\_el.real();

image\_imag\_part = image\_el.imag();

icos = qCos( (2 \* PI \* j \* k) / iheight );

isin = qSin( (2 \* PI \* j \* k) / iheight );

complex\_real\_part = image\_real\_part \* icos - image\_imag\_part \* isin; //qDebug() << "----"; qDebug() << "complex\_real\_part = " << complex\_real\_part;

complex\_imag\_part = image\_real\_part \* isin + image\_imag\_part \* icos; //qDebug() << "complex\_imag\_part = " << complex\_imag\_part; qDebug() << "----";

/\*Запись элемента в complex\_image\*/

complex\_el = complex <float> ( complex\_real\_part, complex\_imag\_part );

/\*Суммирование всех элементов\*/

complex\_el\_sum += complex\_el;

}

/\*Проверка, создан ли текущий строковый вектор\*/

isize = complex\_image.size(); //qDebug() << complex\_el\_sum.real() << " + " << complex\_el\_sum.imag() << " \* i";

if(j < isize)

{

complex\_str = complex\_image.at(j);

/\*complex\_str.insert(i, complex\_el\_sum); \*/ complex\_str.append(complex\_el\_sum);

//complex\_image.remove(j);

//complex\_image.insert(j, complex\_str);

complex\_image.replace(j, complex\_str);

}

else

{

complex\_str.append(complex\_el\_sum);

complex\_image.append(complex\_str);

}

/\*\_\_\*/

complex\_el\_sum = complex <float> ( 0, 0 );

complex\_str.clear();

}

}

return complex\_image;

}

QVector<QVector<float> > shumForm1::**toVectorF**(QVector<QVector<complex<float> > > image)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

QVector < complex <float> > image\_str;

/\*Временная переменная для элменента массива из комплексных чиел \*/

complex <float> image\_el;

QVector <QVector <float> > complex\_image;

QVector <float> complex\_image\_str;

float real, imag, el;

/\*Меняем номер столбца\*/

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

{

for(int j = 0; j < iwidth; j++)

{

image\_str = image.at(i);

image\_el = image\_str.at(j); // qDebug() << "----"; //qDebug() << "i=" << i <<", j=" << j << ", k=" << j << "; iwidth= " << iwidth << ";[]icolor = " << icolor;

//Получаем точку спектра

real = image\_el.real();

imag = image\_el.imag();

el = qSqrt ( qPow(real, 2) + qPow(imag, 2) );

complex\_image\_str.append(el);

}

complex\_image.append(complex\_image\_str);

complex\_image\_str.clear();

}

return complex\_image;

}

QVector<QVector<float> > shumForm1::**toVectorFlog**(QVector<QVector<float> > image)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

QVector<QVector<float> > complex\_image;

QVector <float> image\_str, complex\_str;

/\*Временная переменная для элменента массива из комплексных чиел \*/

float image\_el, complex\_el, image\_el0;

float max, min;

/\*Меняем номер столбца\*/

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

{

//image\_str = image.at(i);

//image\_el0 = image\_str.at(0);

for(int j = 0; j < iwidth; j++)

{

image\_str = image.at(i);

image\_el = image\_str.at(j); //qDebug() << "----"; //qDebug() << "i=" << i <<", j=" << j << ", k=" << j << "; iwidth= " << iwidth << ";[]icolor = " << icolor;

complex\_el = log2( image\_el / (iheight \* iwidth) ); //qDebug() << "!!! - " << complex\_el;

complex\_str.append(complex\_el);

}

complex\_image.append(complex\_str);

complex\_str.clear();

}

return complex\_image;

}

QVector<QVector<float> > shumForm1::**toVectorFshkala**(QVector<QVector<float> > image)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

QVector<QVector<float> > complex\_image;

QVector <float> image\_str, complex\_str;

float image\_el, complex\_el;

float min, max;

image\_str = image.at(0);

min = image\_str.at(0);

max = min;

/\*Меняем номер столбца\*/

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

{

for(int j = 0; j < iwidth; j++)

{

image\_str = image.at(i);

image\_el = image\_str.at(j);

if (image\_el > max)

max = image\_el;

if (image\_el < min)

min = image\_el;

}

}

/\*Меняем номер столбца\*/

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

{

for(int j = 0; j < iwidth; j++)

{

image\_str = image.at(i);

image\_el = image\_str.at(j);

complex\_el = (image\_el - min) / (max - min) \* 255;

complex\_str.append(complex\_el); qDebug() << "[toVectorFshkala] complex\_el = " << complex\_el;

}

complex\_image.append(complex\_str);

complex\_str.clear();

}

return complex\_image;

}

QImage shumForm1::**toMonoImage**(QVector<QVector<float> > image, Images im)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

QImage complex\_image(iwidth, iheight, QImage::Format\_RGB32);

QVector <float> image\_str, complex\_str;

float image\_el, complex\_el;

QPoint p;

QColor icolor;

int ired, igreen, iblue;

/\*Меняем номер столбца\*/

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

{

for(int j = 0; j < iwidth; j++)

{

image\_str = image.at(i);

image\_el = image\_str.at(j);

p = QPoint(j, i);

switch (im)

{

case IMAGE\_R:

ired = int(image\_el); //qDebug() << "[] ired = " << ired;

icolor = QColor(ired, 0, 0, 0);

break;

case IMAGE\_G:

igreen = int(image\_el);

icolor = QColor(0, igreen, 0, 0);

break;

case IMAGE\_B:

iblue = int(image\_el);

icolor = QColor(0, 0, iblue, 0);

break;

}

complex\_image.setPixelColor(p, icolor);

}

}

return complex\_image;

}

QImage shumForm1::**toStereoImage**(QVector<QVector<float> > R\_image, QVector<QVector<float> > G\_image, QVector<QVector<float> > B\_image)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

QImage complex\_image(iwidth, iheight, QImage::Format\_RGB32);

QVector <float> R\_image\_str, G\_image\_str, B\_image\_str, complex\_str;

float R\_image\_el, G\_image\_el, B\_image\_el, complex\_el;

QPoint p;

QColor icolor;

int ired, igreen, iblue;

/\*Меняем номер столбца\*/

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

{

for(int j = 0; j < iwidth; j++)

{

R\_image\_str = R\_image.at(i);

R\_image\_el = R\_image\_str.at(j);

G\_image\_str = G\_image.at(i);

G\_image\_el = G\_image\_str.at(j);

B\_image\_str = B\_image.at(i);

B\_image\_el = B\_image\_str.at(j);

p = QPoint(j, i);

ired = int(R\_image\_el);

igreen = int(G\_image\_el);

iblue = int(B\_image\_el);

icolor = QColor(ired, igreen, iblue, 0); qDebug() << "ired=" << ired << ",igreen=" << igreen << ",iblue="<<iblue;

complex\_image.setPixelColor(p, icolor);

}

}

return complex\_image;

}

QVector<QVector<float> > shumForm1::**Mask**(QVector<QVector<float> > image, float D0, maskType Type)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

QVector<QVector< float > > mask;

QVector <float> mask\_str;

/\*Временная переменная для элеменента массива \*/

float H\_uv;

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

{

for(int j = 0; j < iwidth; j++)

{

switch (Type)

{

case Ideal:

H\_uv = H\_u\_v\_ideal(D0, j, i, iwidth, iheight);

break;

case Gauss:

H\_uv = H\_u\_v\_gauss(D0, j, i, iwidth, iheight);

break;

}

mask\_str.append(H\_uv);

}

mask.append(mask\_str);

mask\_str.clear();

}

return mask;

}

QImage shumForm1::**toMonoMask**(QVector<QVector<float> > image, shumForm1::Images im)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

QImage complex\_image(iwidth, iheight, QImage::Format\_RGB32);

QVector <float> image\_str, complex\_str;

float image\_el, complex\_el;

QPoint p;

QColor icolor;

int ired, igreen, iblue;

/\*Меняем номер столбца\*/

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

{

for(int j = 0; j < iwidth; j++)

{

image\_str = image.at(i);

image\_el = image\_str.at(j);

p = QPoint(j, i);

switch (im)

{

case IMAGE\_R:

ired = 255 \* int(image\_el); //qDebug() << "[] ired = " << ired;

icolor = QColor(ired, 0, 0, 0);

break;

case IMAGE\_G:

igreen = 255 \* int(image\_el);

icolor = QColor(0, igreen, 0, 0);

break;

case IMAGE\_B:

iblue = 255 \* int(image\_el);

icolor = QColor(0, 0, iblue, 0);

break;

}

complex\_image.setPixelColor(p, icolor);

}

}

return complex\_image;

}

QImage shumForm1::**toStereoMask**(QVector<QVector<float> > R\_image, QVector<QVector<float> > G\_image, QVector<QVector<float> > B\_image)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

QImage complex\_image(iwidth, iheight, QImage::Format\_RGB32);

QVector <float> R\_image\_str, G\_image\_str, B\_image\_str, complex\_str;

float R\_image\_el, G\_image\_el, B\_image\_el, complex\_el;

QPoint p;

QColor icolor;

int ired, igreen, iblue;

/\*Меняем номер столбца\*/

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

{

for(int j = 0; j < iwidth; j++)

{

R\_image\_str = R\_image.at(i);

R\_image\_el = R\_image\_str.at(j);

G\_image\_str = G\_image.at(i);

G\_image\_el = G\_image\_str.at(j);

B\_image\_str = B\_image.at(i);

B\_image\_el = B\_image\_str.at(j);

p = QPoint(j, i);

ired = 255 \* int(R\_image\_el);

igreen = 255 \* int(G\_image\_el);

iblue = 255 \* int(B\_image\_el);

icolor = QColor(ired, igreen, iblue, 0);

complex\_image.setPixelColor(p, icolor);

}

}

return complex\_image;

}

QVector<QVector<float> > shumForm1::**idealFilter**(QVector<QVector<float> > spektr, QVector<QVector<float> > mask)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

/\*Добавить выбор фильтра\*/

//float D0 = 5;

QVector<QVector<float> > filtered\_spektr;

QVector <float> spektr\_str, mask\_str, filtered\_spekr\_str;

/\*Временная переменная для элменента массива из комплексных чиел \*/

float F\_uv, H\_uv, G\_uv;

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

{

for(int j = 0; j < iwidth; j++)

{

spektr\_str = spektr.at(i);

F\_uv = spektr\_str.at(j);

mask\_str = mask.at(i);

H\_uv = mask\_str.at(j);

/\*Сделать конвертацию\*/

G\_uv = H\_uv \* F\_uv; //qDebug() << "G\_uv = " << G\_uv.real() << " + " << G\_uv.imag() << "\*i";

filtered\_spekr\_str.append(G\_uv);

}

filtered\_spektr.append(filtered\_spekr\_str);

filtered\_spekr\_str.clear();

}

return filtered\_spektr;

}

QVector<QVector<complex<float> > > shumForm1::**idealFilterComplex**(QVector<QVector<complex<float> > > spektr, QVector<QVector<float> > mask)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

/\*Добавить выбор фильтра\*/

//float D0 = 5;

QVector<QVector<complex<float> > > filtered\_spektr;

QVector < complex<float> > spektr\_str, filtered\_spekr\_str;

QVector <float> mask\_str;

complex <float> spektr\_el;

float mask\_el;

/\*Временная переменная для элменента массива из комплексных чиел \*/

//float H\_uv;

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

{

for(int j = 0; j < iwidth; j++)

{

spektr\_str = spektr.at(i);

spektr\_el = spektr\_str.at(j);

mask\_str = mask.at(i);

mask\_el = mask\_str.at(j);

if (mask\_el == 1)

{

filtered\_spekr\_str.append(spektr\_el);

}

else

{

filtered\_spekr\_str.append( complex <float> (0, 0) );

}

}

filtered\_spektr.append(filtered\_spekr\_str);

filtered\_spekr\_str.clear();

}

return filtered\_spektr;

}

QVector<QVector<complex<float> > > shumForm1::**ObratnoyeFurieStroki**(QVector<QVector<complex<float> > > image)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

/\*Двумерный массив из комплексных чисел\*/

QVector < QVector < complex <float> > > complex\_image;

/\*Временный массив для строки массива комплексных чисел \*/

QVector < complex <float> > complex\_str, image\_str;

/\*Временная переменная для элменента массива из комплексных чиел \*/

complex <float> image\_el, complex\_el\_sum, complex\_el;

float icolor, real\_part, imag\_part, image\_real\_part, image\_imag\_part, complex\_real\_part, complex\_imag\_part, icos, isin;

/\*Меняем номер столбца\*/

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

{

for(int j = 0; j < iwidth; j++)

{

/\*j - в данном случае частота, k - отсчёт\*/

for (int k = 0; k < iwidth; k++)

{

//image\_str = image.at(i);

//icolor = image\_str.at(k); //qDebug() << "----"; qDebug() << "i=" << i <<", j=" << j << ", k=" << k << "; iwidth= " << iwidth << ";[]icolor = " << icolor;

/\*Создание комплексного значения F(u, y)\*/

//real\_part = icolor \* qCos( (2 \* PI \* j \* k) / iwidth ); //qDebug() << "real\_part = " << real\_part;

//imag\_part = icolor \* qSin( (2 \* PI \* j \* k) / iwidth ); //qDebug() << "imag\_part = " << imag\_part; qDebug() << "----";

//complex\_el = complex <float> ( real\_part, imag\_part );

//complex\_el\_sum += complex\_el;

/\*Считывание элемента из image\*/

image\_str = image.at(i);

image\_el = image\_str.at(k);

/\*\_\_\*/

image\_real\_part = image\_el.real();

image\_imag\_part = image\_el.imag();

icos = qCos( (2 \* PI \* j \* k) / iwidth );

isin = qSin( (2 \* PI \* j \* k) / iwidth );

complex\_real\_part = image\_real\_part \* icos + image\_imag\_part \* isin; //qDebug() << "----"; qDebug() << "complex\_real\_part = " << complex\_real\_part;

complex\_imag\_part = image\_real\_part \* isin - image\_imag\_part \* icos; //qDebug() << "complex\_imag\_part = " << complex\_imag\_part; qDebug() << "----";

/\*Запись элемента в complex\_image\*/

complex\_el = complex <float> ( complex\_real\_part, complex\_imag\_part );

/\*Суммирование всех элементов\*/

complex\_el\_sum += complex\_el;

}

/\* добавление данных в строку массива \*/

complex\_str.append( complex\_el\_sum ); //qDebug() << complex\_el\_sum.real() << " + " << complex\_el\_sum.imag() << " \* i";

complex\_el\_sum = 0 + 0 \* i;

complex\_el = 0 + 0 \* i;

}

complex\_image.append( complex\_str );

complex\_str.clear();

}

qDebug() << "ObratnoyeFurieStroki!";

return complex\_image;

}

QVector<QVector<complex<float> > > shumForm1::**ObratnoyeFurieStolbtsyi**(QVector<QVector<complex<float> > > image)

{

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

int isize;

QVector<QVector<complex<float> > > complex\_image;

QVector<complex<float> > image\_str, complex\_image\_str;

complex <float> image\_el, complex\_image\_el, complex\_image\_el\_current;

float image\_real\_part, image\_imag\_part;

float complex\_real\_part, complex\_imag\_part;

float icos, isin;

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

{

/\*Меняем номер столбца\*/

for(int j = 0; j < iheight; j++)

{

/\*k - в данном случае частота, j - отсчёт\*/

for (int k = 0; k < iheight; k++)

{

image\_str = image.at(k);

image\_el = image\_str.at(i);

image\_real\_part = image\_el.real();

image\_imag\_part = image\_el.imag();

icos = qCos( ( 2 \* PI \* j \* k) / iheight );

isin = qSin( ( 2 \* PI \* j \* k) / iheight );

complex\_real\_part = image\_real\_part \* icos + image\_imag\_part \* isin; //qDebug() << "----"; qDebug() << "complex\_real\_part1 = " << complex\_real\_part;

complex\_imag\_part = image\_real\_part \* isin - image\_imag\_part \* icos; //qDebug() << "complex\_imag\_part1 = " << complex\_imag\_part; qDebug() << "----";

/\*Запись элемента в complex\_image\*/

complex\_image\_el\_current = complex <float> ( complex\_real\_part, complex\_imag\_part );

/\*Суммирование всех элементов\*/

complex\_image\_el += complex\_image\_el\_current;

}

/\*Проверка, создан ли текущий строковый вектор\*/

isize = complex\_image.size(); //qDebug() << "OBR real=" << complex\_image\_el.real() << ", imag=" << complex\_image\_el.imag();

if(j < isize)

{

complex\_image\_str = complex\_image.at(j);

/\*complex\_image\_str.insert(i, complex\_image\_el);\*/complex\_image\_str.append(complex\_image\_el);

complex\_image.replace(j, complex\_image\_str);

}

else

{

complex\_image\_str.append(complex\_image\_el);

complex\_image.append(complex\_image\_str);

}

/\*\_\_\*/

complex\_image\_el = complex <float> ( 0, 0 );

complex\_image\_str.clear();

}

}

qDebug() << "ObratnoyeFurieStolbtsyi!";

return complex\_image;

}

QVector<QVector<float> > shumForm1::**RealVector**(QVector<QVector<complex<float> > > image)

{

QVector<QVector<float> > complex\_image;

QVector <complex <float> > image\_str;

complex <float> image\_el;

QVector<float > complex\_image\_str;

float real, imag, complex\_image\_el;

float icolor;

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

int step;

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

{

for(int j = 0; j < iwidth; j++)

{

image\_str = image.at(i);

image\_el = image\_str.at(j);

real = image\_el.real();

imag = image\_el.imag();

complex\_image\_el = (real + imag) / (iheight \* iwidth);

complex\_image\_str.append(complex\_image\_el);

}

complex\_image.append(complex\_image\_str);

complex\_image\_str.clear();

}

return complex\_image;

}

QVector<QVector<float> > shumForm1::**CentrVector**(QVector<QVector<float> > image)

{

QVector<QVector<float> > complex\_image;

QVector<float > image\_str, complex\_image\_str;

float image\_el, complex\_image\_el, complex\_image\_el\_current;

float icolor;

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

int step;

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

{

for(int j = 0; j < iwidth; j++)

{

image\_str = image.at(i);

image\_el = image\_str.at(j);

step = one\_with\_N(i + j);

icolor = image\_el \* step; qDebug() << "[CentrVector icolor] = " << icolor;

complex\_image\_str.append(icolor);

}

complex\_image.append(complex\_image\_str);

complex\_image\_str.clear();

}

return complex\_image;

}

QVector<QVector<float> > shumForm1::**Warning\_fix**(QVector<QVector<float> > image)

{

QVector<QVector<float> > complex\_image;

QVector<float > image\_str, complex\_image\_str;

float image\_el, complex\_image\_el, complex\_image\_el\_current;

int iwidth = this->imageWidth;

int iheight = this->imageHeight;

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

{

for(int j = iwidth - 1; j >= 0; j--)

{

image\_str = image.at(i);

image\_el = image\_str.at(j);

complex\_image\_str.append(image\_el);

}

complex\_image.append(complex\_image\_str);

complex\_image\_str.clear();

}

return complex\_image;

}

float shumForm1::**D\_u\_v**(float u, float v, float iwidth, float iheight)

{

float D = qSqrt( qPow(u - iwidth/2, 2) + qPow(v - iheight/2, 2) );

return D;

}

float shumForm1::**H\_u\_v\_ideal**(float D0, float u, float v, float iwidth, float iheight)

{

float D = D\_u\_v(u,v,iwidth,iheight); //qDebug() << "[!] D = " << D;

float H;

if ( D <= D0)

H = 1;

else

H = 0;

return H;

}

float shumForm1::**H\_u\_v\_gauss**(float D0, float u, float v, float iwidth, float iheight)

{

float D = D\_u\_v(u,v,iwidth,iheight); //qDebug() << "[!] D = " << D;

float H;

H = 1 - qExp( - qPow(D, 2) / qPow(D0, 2) );

return H;

}

float shumForm1::**one\_with\_N**(int n)

{

float m;

n = n % 2;

/\*Если остаток существует, то число нечетное\*/

if (n)

m = -1;

else

m = 1;

return m;

}

//D:\Obrazets\teatbwcut.bmp

/\*Образец для вставки: D:\Obrazets\1test.bmp \*/

/\*Образец для вставки: D:\Obrazets\1testcut.bmp \*/

void shumForm1::**process\_of\_ideal\_RGBsignal**(QImage Ishodnoe, QImage RIshodnoe, QImage GIshodnoe, QImage BIshodnoe, float D0, int width, int height)

{

imageWidth = width;

imageHeight = height;

qDebug() << "This is process\_of\_ideal\_RGBsignal; D0=" << D0;

/\*Прямое\*/

QVector <QVector <float> > r = CentrImage(RIshodnoe, IMAGE\_R); //() << r; //SeeImages(Ishodnoe, IMAGE\_CLONE);

QVector <QVector <complex <float> > > r\_complex = PryamoeFurieStroki(r);

QVector <QVector <complex <float> > > r\_complex\_1 = PryamoeFurieStolbtsyi(r\_complex);

QVector <QVector <float> > g = CentrImage(GIshodnoe, IMAGE\_G); //qDebug() << g; //SeeImages(GIshodnoe, IMAGE\_CLONE);

QVector <QVector <complex <float> > > g\_complex = PryamoeFurieStroki(g);

QVector <QVector <complex <float> > > g\_complex\_1 = PryamoeFurieStolbtsyi(g\_complex);

QVector <QVector <float> > b = CentrImage(BIshodnoe, IMAGE\_B); //qDebug() << r; //SeeImages(Ishodnoe, IMAGE\_CLONE);

QVector <QVector <complex <float> > > b\_complex = PryamoeFurieStroki(b);

QVector <QVector <complex <float> > > b\_complex\_1 = PryamoeFurieStolbtsyi(b\_complex);

/\*Фильтрация\*/

/\*D0 = 100;\*/

QVector<QVector<float> > r\_mask = Mask(r, D0, Ideal);

QVector<QVector<float> > g\_mask = Mask(g, D0, Ideal);

QVector<QVector<float> > b\_mask = Mask(b, D0, Ideal);

QVector <QVector <complex <float> > > r\_G = idealFilterComplex(r\_complex\_1, r\_mask);

QVector <QVector <complex <float> > > g\_G = idealFilterComplex(g\_complex\_1, g\_mask);

QVector <QVector <complex <float> > > b\_G = idealFilterComplex(b\_complex\_1, b\_mask);

/\*Обратное\*/

QVector <QVector <complex <float> > > r\_obr = ObratnoyeFurieStolbtsyi(r\_G);

QVector <QVector <complex <float> > > r\_obr\_1 = ObratnoyeFurieStroki(r\_obr);

QVector <QVector <float> > r\_obr\_2 = RealVector(r\_obr\_1);

QVector <QVector <float> > r\_obr\_3 = CentrVector(r\_obr\_2);

QVector <QVector <complex <float> > > g\_obr = ObratnoyeFurieStolbtsyi(g\_G);

QVector <QVector <complex <float> > > g\_obr\_1 = ObratnoyeFurieStroki(g\_obr);

QVector <QVector <float> > g\_obr\_2 = RealVector(g\_obr\_1);

QVector <QVector <float> > g\_obr\_3 = CentrVector(g\_obr\_2);

QVector <QVector <complex <float> > > b\_obr = ObratnoyeFurieStolbtsyi(b\_G);

QVector <QVector <complex <float> > > b\_obr\_1 = ObratnoyeFurieStroki(b\_obr);

QVector <QVector <float> > b\_obr\_2 = RealVector(b\_obr\_1);

QVector <QVector <float> > b\_obr\_3 = CentrVector(b\_obr\_2);

/\*Преобразование к виду, удобному для показа\*/

/\*Прямое\*/

QVector <QVector <float> > r\_after = toVectorF(r\_complex\_1); //qDebug() << r\_after;

QVector <QVector<float> > r\_after\_log = toVectorFlog(r\_after); //qDebug() << r\_after\_log;

QVector<QVector<float> > r\_shkala = toVectorFshkala(r\_after\_log); //qDebug() << r\_shkala;

QVector<QVector<float> > r\_G\_sh = idealFilter(r\_shkala, r\_mask);

QVector <QVector <float> > g\_after = toVectorF(g\_complex\_1); //qDebug() << g\_after;

QVector <QVector<float> > g\_after\_log = toVectorFlog(g\_after); //qDebug() << g\_after\_log;

QVector<QVector<float> > g\_shkala = toVectorFshkala(g\_after\_log); //qDebug() << g\_shkala;

QVector<QVector<float> > g\_G\_sh = idealFilter(g\_shkala, g\_mask);

QVector <QVector <float> > b\_after = toVectorF(b\_complex\_1); //qDebug() << b\_after;

QVector <QVector<float> > b\_after\_log = toVectorFlog(b\_after); //qDebug() << b\_after\_log;

QVector<QVector<float> > b\_shkala = toVectorFshkala(b\_after\_log); //qDebug() << b\_shkala;

QVector<QVector<float> > b\_G\_sh = idealFilter(b\_shkala, b\_mask);

/\*Обратное\*/

QVector<QVector<float> > r\_obr\_shkala = toVectorFshkala(r\_obr\_3); //qDebug() << r\_obr\_shkala;

QVector<QVector<float> > r\_obr\_shkala\_1 = Warning\_fix(r\_obr\_shkala);

QVector<QVector<float> > g\_obr\_shkala = toVectorFshkala(g\_obr\_3); //qDebug() << r\_obr\_shkala;

QVector<QVector<float> > g\_obr\_shkala\_1 = Warning\_fix(g\_obr\_shkala);

QVector<QVector<float> > b\_obr\_shkala = toVectorFshkala(b\_obr\_3); //qDebug() << r\_obr\_shkala;

QVector<QVector<float> > b\_obr\_shkala\_1 = Warning\_fix(b\_obr\_shkala);

/\*Преобразование некоторых изображений из массивов в картинки\*/

QImage RGB\_SPEKTR = toStereoImage(r\_shkala, g\_shkala, b\_shkala);

QImage R\_SPEKTR = toMonoImage(r\_shkala, IMAGE\_R);

QImage G\_SPEKTR = toMonoImage(g\_shkala, IMAGE\_G);

QImage B\_SPEKTR = toMonoImage(b\_shkala, IMAGE\_B);

QImage RGB\_FILTER = toStereoMask(r\_mask, g\_mask, b\_mask);

QImage R\_FILTER = toMonoMask(r\_mask, IMAGE\_R);

QImage G\_FILTER = toMonoMask(g\_mask, IMAGE\_G);

QImage B\_FILTER = toMonoMask(b\_mask, IMAGE\_B);

QImage RGB\_G = toStereoImage(r\_G\_sh, g\_G\_sh, b\_G\_sh);

QImage R\_RG = toMonoImage(r\_G\_sh, IMAGE\_R);

QImage G\_GG = toMonoImage(g\_G\_sh, IMAGE\_G);

QImage B\_BG = toMonoImage(b\_G\_sh, IMAGE\_B);

QImage RGB\_AFTER\_FILTER = toStereoImage(r\_obr\_shkala\_1, g\_obr\_shkala\_1, b\_obr\_shkala\_1);

QImage R\_AFTER\_FILTER = toMonoImage(r\_obr\_shkala\_1, IMAGE\_R);

QImage G\_AFTER\_FILTER = toMonoImage(g\_obr\_shkala\_1, IMAGE\_G);

QImage B\_AFTER\_FILTER = toMonoImage(b\_obr\_shkala\_1, IMAGE\_B);

/\*Показать изображения\*/

SeeImages(Ishodnoe, IMAGE\_CLONE);

SeeImages(RIshodnoe, IMAGE\_R);

SeeImages(GIshodnoe, IMAGE\_G);

SeeImages(BIshodnoe, IMAGE\_B);

SeeImages(RGB\_SPEKTR, IMAGE\_SPEKTR);

//SeeImages(R\_SPEKTR, IMAGE\_ST1);

//SeeImages(G\_SPEKTR, IMAGE\_ST2);

//SeeImages(B\_SPEKTR, IMAGE\_ST3);

SeeImages(RGB\_FILTER, IMAGE\_CLONE1);

//SeeImages(R\_FILTER, IMAGE\_ST1);

//SeeImages(G\_FILTER, IMAGE\_ST2);

//SeeImages(B\_FILTER, IMAGE\_ST3);

SeeImages(RGB\_G, IMAGE\_ST1);

//SeeImages(R\_RG, IMAGE\_ST1);

//SeeImages(G\_GG, IMAGE\_ST2);

//SeeImages(B\_BG, IMAGE\_ST3);

SeeImages(RGB\_AFTER\_FILTER, IMAGE\_ST2);

//SeeImages(R\_AFTER\_FILTER, IMAGE\_ST1);

//SeeImages(G\_AFTER\_FILTER, IMAGE\_ST2);

//SeeImages(B\_AFTER\_FILTER, IMAGE\_ST3);

/\*Сохранить изображения\*/

iImage.append(Ishodnoe);

iImage.append(RIshodnoe);

iImage.append(GIshodnoe);

iImage.append(BIshodnoe);

iImage.append(RGB\_SPEKTR);

iImage.append(RGB\_FILTER);

iImage.append(RGB\_G);

iImage.append(RGB\_AFTER\_FILTER);

emit for\_form2\_signal(R\_SPEKTR, G\_SPEKTR, B\_SPEKTR, R\_FILTER, G\_FILTER, B\_FILTER, R\_AFTER\_FILTER, G\_AFTER\_FILTER, B\_AFTER\_FILTER);

SF2->show();

}

void shumForm1::**process\_of\_gauss\_RGBsignal**(QImage Ishodnoe, QImage RIshodnoe, QImage GIshodnoe, QImage BIshodnoe, float D0, int width, int height)

{

imageWidth = width;

imageHeight = height;

qDebug() << "This is process\_of\_gauss\_RGBsignal;";

/\*Прямое\*/

QVector <QVector <float> > r = CentrImage(RIshodnoe, IMAGE\_R); //() << r; //SeeImages(Ishodnoe, IMAGE\_CLONE);

QVector <QVector <complex <float> > > r\_complex = PryamoeFurieStroki(r);

QVector <QVector <complex <float> > > r\_complex\_1 = PryamoeFurieStolbtsyi(r\_complex);

QVector <QVector <float> > g = CentrImage(GIshodnoe, IMAGE\_G); //qDebug() << g; //SeeImages(GIshodnoe, IMAGE\_CLONE);

QVector <QVector <complex <float> > > g\_complex = PryamoeFurieStroki(g);

QVector <QVector <complex <float> > > g\_complex\_1 = PryamoeFurieStolbtsyi(g\_complex);

QVector <QVector <float> > b = CentrImage(BIshodnoe, IMAGE\_B); //qDebug() << r; //SeeImages(Ishodnoe, IMAGE\_CLONE);

QVector <QVector <complex <float> > > b\_complex = PryamoeFurieStroki(b);

QVector <QVector <complex <float> > > b\_complex\_1 = PryamoeFurieStolbtsyi(b\_complex);

/\*Фильтрация\*/

/\*D0 = 100;\*/

QVector<QVector<float> > r\_mask = Mask(r, D0, Gauss);

QVector<QVector<float> > g\_mask = Mask(g, D0, Gauss);

QVector<QVector<float> > b\_mask = Mask(b, D0, Gauss);

QVector <QVector <complex <float> > > r\_G = idealFilterComplex(r\_complex\_1, r\_mask);

QVector <QVector <complex <float> > > g\_G = idealFilterComplex(g\_complex\_1, g\_mask);

QVector <QVector <complex <float> > > b\_G = idealFilterComplex(b\_complex\_1, b\_mask);

/\*Обратное\*/

QVector <QVector <complex <float> > > r\_obr = ObratnoyeFurieStolbtsyi(r\_G);

QVector <QVector <complex <float> > > r\_obr\_1 = ObratnoyeFurieStroki(r\_obr);

QVector <QVector <float> > r\_obr\_2 = RealVector(r\_obr\_1);

QVector <QVector <float> > r\_obr\_3 = CentrVector(r\_obr\_2);

QVector <QVector <complex <float> > > g\_obr = ObratnoyeFurieStolbtsyi(g\_G);

QVector <QVector <complex <float> > > g\_obr\_1 = ObratnoyeFurieStroki(g\_obr);

QVector <QVector <float> > g\_obr\_2 = RealVector(g\_obr\_1);

QVector <QVector <float> > g\_obr\_3 = CentrVector(g\_obr\_2);

QVector <QVector <complex <float> > > b\_obr = ObratnoyeFurieStolbtsyi(b\_G);

QVector <QVector <complex <float> > > b\_obr\_1 = ObratnoyeFurieStroki(b\_obr);

QVector <QVector <float> > b\_obr\_2 = RealVector(b\_obr\_1);

QVector <QVector <float> > b\_obr\_3 = CentrVector(b\_obr\_2);

/\*Преобразование к виду, удобному для показа\*/

/\*Прямое\*/

QVector <QVector <float> > r\_after = toVectorF(r\_complex\_1); //qDebug() << r\_after;

QVector <QVector<float> > r\_after\_log = toVectorFlog(r\_after); //qDebug() << r\_after\_log;

QVector<QVector<float> > r\_shkala = toVectorFshkala(r\_after\_log); //qDebug() << r\_shkala;

QVector<QVector<float> > r\_G\_sh = idealFilter(r\_shkala, r\_mask);

QVector <QVector <float> > g\_after = toVectorF(g\_complex\_1); //qDebug() << g\_after;

QVector <QVector<float> > g\_after\_log = toVectorFlog(g\_after); //qDebug() << g\_after\_log;

QVector<QVector<float> > g\_shkala = toVectorFshkala(g\_after\_log); //qDebug() << g\_shkala;

QVector<QVector<float> > g\_G\_sh = idealFilter(g\_shkala, g\_mask);

QVector <QVector <float> > b\_after = toVectorF(b\_complex\_1); //qDebug() << b\_after;

QVector <QVector<float> > b\_after\_log = toVectorFlog(b\_after); //qDebug() << b\_after\_log;

QVector<QVector<float> > b\_shkala = toVectorFshkala(b\_after\_log); //qDebug() << b\_shkala;

QVector<QVector<float> > b\_G\_sh = idealFilter(b\_shkala, b\_mask);

/\*Обратное\*/

QVector<QVector<float> > r\_obr\_shkala = toVectorFshkala(r\_obr\_3); //qDebug() << r\_obr\_shkala;

QVector<QVector<float> > r\_obr\_shkala\_1 = Warning\_fix(r\_obr\_shkala);

QVector<QVector<float> > g\_obr\_shkala = toVectorFshkala(g\_obr\_3); //qDebug() << r\_obr\_shkala;

QVector<QVector<float> > g\_obr\_shkala\_1 = Warning\_fix(g\_obr\_shkala);

QVector<QVector<float> > b\_obr\_shkala = toVectorFshkala(b\_obr\_3); //qDebug() << r\_obr\_shkala;

QVector<QVector<float> > b\_obr\_shkala\_1 = Warning\_fix(b\_obr\_shkala);

/\*Преобразование некоторых изображений из массивов в картинки\*/

QImage RGB\_SPEKTR = toStereoImage(r\_shkala, g\_shkala, b\_shkala);

QImage R\_SPEKTR = toMonoImage(r\_shkala, IMAGE\_R);

QImage G\_SPEKTR = toMonoImage(g\_shkala, IMAGE\_G);

QImage B\_SPEKTR = toMonoImage(b\_shkala, IMAGE\_B);

QImage RGB\_FILTER = toStereoMask(r\_mask, g\_mask, b\_mask);

QImage R\_FILTER = toMonoMask(r\_mask, IMAGE\_R);

QImage G\_FILTER = toMonoMask(g\_mask, IMAGE\_G);

QImage B\_FILTER = toMonoMask(b\_mask, IMAGE\_B);

QImage RGB\_G = toStereoImage(r\_G\_sh, g\_G\_sh, b\_G\_sh);

QImage R\_RG = toMonoImage(r\_G\_sh, IMAGE\_R);

QImage G\_GG = toMonoImage(g\_G\_sh, IMAGE\_G);

QImage B\_BG = toMonoImage(b\_G\_sh, IMAGE\_B);

QImage RGB\_AFTER\_FILTER = toStereoImage(r\_obr\_shkala\_1, g\_obr\_shkala\_1, b\_obr\_shkala\_1);

QImage R\_AFTER\_FILTER = toMonoImage(r\_obr\_shkala\_1, IMAGE\_R);

QImage G\_AFTER\_FILTER = toMonoImage(g\_obr\_shkala\_1, IMAGE\_G);

QImage B\_AFTER\_FILTER = toMonoImage(b\_obr\_shkala\_1, IMAGE\_B);

/\*Показать изображения\*/

SeeImages(Ishodnoe, IMAGE\_CLONE);

SeeImages(RIshodnoe, IMAGE\_R);

SeeImages(GIshodnoe, IMAGE\_G);

SeeImages(BIshodnoe, IMAGE\_B);

SeeImages(RGB\_SPEKTR, IMAGE\_SPEKTR);

//SeeImages(R\_SPEKTR, IMAGE\_ST1);

//SeeImages(G\_SPEKTR, IMAGE\_ST2);

//SeeImages(B\_SPEKTR, IMAGE\_ST3);

SeeImages(RGB\_FILTER, IMAGE\_CLONE1);

//SeeImages(R\_FILTER, IMAGE\_ST1);

//SeeImages(G\_FILTER, IMAGE\_ST2);

//SeeImages(B\_FILTER, IMAGE\_ST3);

SeeImages(RGB\_G, IMAGE\_ST1);

//SeeImages(R\_RG, IMAGE\_ST1);

//SeeImages(G\_GG, IMAGE\_ST2);

//SeeImages(B\_BG, IMAGE\_ST3);

SeeImages(RGB\_AFTER\_FILTER, IMAGE\_ST2);

//SeeImages(R\_AFTER\_FILTER, IMAGE\_ST1);

//SeeImages(G\_AFTER\_FILTER, IMAGE\_ST2);

//SeeImages(B\_AFTER\_FILTER, IMAGE\_ST3);

/\*Сохранить изображения\*/

iImage.append(Ishodnoe);

iImage.append(RIshodnoe);

iImage.append(GIshodnoe);

iImage.append(BIshodnoe);

iImage.append(RGB\_SPEKTR);

iImage.append(RGB\_FILTER);

iImage.append(RGB\_G);

iImage.append(RGB\_AFTER\_FILTER);

emit for\_form2\_signal(R\_SPEKTR, G\_SPEKTR, B\_SPEKTR, R\_FILTER, G\_FILTER, B\_FILTER, R\_AFTER\_FILTER, G\_AFTER\_FILTER, B\_AFTER\_FILTER);

SF2->show();

}

void shumForm1::**saveImages**(QString saveway)

{

/\*QString saveWay = ui->lineEdit->text();\*/ qDebug() << "[] saveWay = " << saveway;

QVector <bool> checkboxState;

//Создание массива значение чекбоксов

checkboxState.append(ui->checkBox->isChecked() );

checkboxState.append(ui->checkBox\_2->isChecked() );

checkboxState.append(ui->checkBox\_4->isChecked() );

checkboxState.append(ui->checkBox\_5->isChecked() );

checkboxState.append(ui->checkBox\_6->isChecked() );

checkboxState.append(ui->checkBox\_7->isChecked() );

checkboxState.append(ui->checkBox\_8->isChecked() );

checkboxState.append(ui->checkBox\_9->isChecked() );

/\*checkboxState.append(ui->checkBox\_3->isChecked() );\*/qDebug() << "[] Состояние чекбоксов = " << checkboxState;

bool currentCheckbox = false;

QString str = "AfterProgram-";

QString currentStr, status;

//Перебор массива и сохранение отмеченных картинок

for (int i = 0; i < 8; i++)

{

currentCheckbox = checkboxState.at(i);

if (currentCheckbox == true)

{

currentStr = saveway + str + QString::number(i) + ".bmp";

iImage.at(i).save( currentStr, "BMP" );

status = status + currentStr + "\n"; qDebug() << status;

}

}

ui->label\_15->setText(status);

emit forSave\_images(saveway);

qDebug() << "Проверка, кнопка сохранения нажата!";

}

*Листинг shumform2.cpp*

#include "shumform2.h"

#include "ui\_shumform2.h"

#include <QDebug>

shumForm2::**shumForm2**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::shumForm2)

{

ui->setupUi(this);

}

shumForm2::~***shumForm2***()

{

delete ui;

}

void shumForm2::**SeeImages**(QImage image, shumForm2::Images im)

{

QSize labelSize;

QImage imageSmall;

switch (im)

{

case IMAGE\_CLONE:

qDebug() << "image CLONE";

labelSize = ui->label->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_SPEKTR:

qDebug() << "image SPEKTR";

labelSize = ui->label\_2->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_2->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_CLONE1:

qDebug() << "image CLONE1";

labelSize = ui->label\_3->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_3->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_R:

qDebug() << "image R";

labelSize = ui->label\_4->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_4->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_G:

qDebug() << "image G";

labelSize = ui->label\_5->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_5->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_B:

qDebug() << "image B";

labelSize = ui->label\_6->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_6->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_ST1:

qDebug() << "image ST1";

labelSize = ui->label\_7->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_7->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_ST2:

qDebug() << "image ST2";

labelSize = ui->label\_8->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_8->setPixmap(QPixmap::fromImage(imageSmall));

break;

case IMAGE\_ST3:

qDebug() << "image ST3";

labelSize = ui->label\_9->size();

imageSmall = image.scaled(labelSize.width(), labelSize.height(), Qt::KeepAspectRatio);

ui->label\_9->setPixmap(QPixmap::fromImage(imageSmall));

break;

}

}

void shumForm2::**process\_for\_seen\_images**(QImage R\_SPEKTR, QImage G\_SPEKTR, QImage B\_SPEKTR, QImage R\_FILTER, QImage G\_FILTER, QImage B\_FILTER, QImage R\_RG, QImage G\_RG, QImage B\_RG)

{

qDebug() << "show images!!!";

SeeImages(R\_SPEKTR, IMAGE\_CLONE);

SeeImages(G\_SPEKTR, IMAGE\_SPEKTR);

SeeImages(B\_SPEKTR, IMAGE\_CLONE1);

SeeImages(R\_FILTER, IMAGE\_R);

SeeImages(G\_FILTER, IMAGE\_G);

SeeImages(B\_FILTER, IMAGE\_B);

SeeImages(R\_RG, IMAGE\_ST1);

SeeImages(G\_RG, IMAGE\_ST2);

SeeImages(B\_RG, IMAGE\_ST3);

/\*Сохранить изображения\*/

iImage.append(R\_SPEKTR);

iImage.append(G\_SPEKTR);

iImage.append(B\_SPEKTR);

iImage.append(R\_FILTER);

iImage.append(G\_FILTER);

iImage.append(B\_FILTER);

iImage.append(R\_RG);

iImage.append(G\_RG);

iImage.append(B\_RG);

}

void shumForm2::**saveImages**(QString saveway)

{

/\*QString saveWay = ui->lineEdit->text();\*/ qDebug() << "[] saveWay = " << saveway;

QVector <bool> checkboxState;

//Создание массива значение чекбоксов

checkboxState.append(ui->checkBox->isChecked() );

checkboxState.append(ui->checkBox\_2->isChecked() );

checkboxState.append(ui->checkBox\_4->isChecked() );

checkboxState.append(ui->checkBox\_5->isChecked() );

checkboxState.append(ui->checkBox\_6->isChecked() );

checkboxState.append(ui->checkBox\_7->isChecked() );

checkboxState.append(ui->checkBox\_8->isChecked() );

checkboxState.append(ui->checkBox\_9->isChecked() );

checkboxState.append(ui->checkBox\_3->isChecked() );qDebug() << "[] Состояние чекбоксов = " << checkboxState;

bool currentCheckbox = false;

QString str = "AfterProgram-";

QString currentStr, status;

//Перебор массива и сохранение отмеченных картинок

for (int i = 0; i < 9; i++)

{

currentCheckbox = checkboxState.at(i);

if (currentCheckbox == true)

{

currentStr = saveway + str + QString::number(i + 9) + ".bmp";

iImage.at(i).save( currentStr, "BMP" );

status = status + currentStr + "\n"; qDebug() << status;

}

}

ui->label\_15->setText(status);

qDebug() << "Проверка, кнопка сохранения нажата!";

}

*Листинг mainwindow.ui*

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>341</width>

<height>274</height>

</rect>

</property>

<property name="windowTitle">

<string>MainWindow</string>

</property>

<widget class="QWidget" name="centralWidget">

<widget class="QLineEdit" name="lineEdit">

<property name="geometry">

<rect>

<x>10</x>

<y>10</y>

<width>161</width>

<height>20</height>

</rect>

</property>

</widget>

<widget class="QLabel" name="label">

<property name="geometry">

<rect>

<x>190</x>

<y>0</y>

<width>201</width>

<height>31</height>

</rect>

</property>

<property name="text">

<string>Впишите путь к картинке в

формате bmp</string>

</property>

</widget>

<widget class="QPushButton" name="pushButton">

<property name="geometry">

<rect>

<x>10</x>

<y>70</y>

<width>191</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Идеальный фильтр низких частот</string>

</property>

</widget>

<widget class="QPushButton" name="pushButton\_2">

<property name="geometry">

<rect>

<x>10</x>

<y>110</y>

<width>191</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Гауссов фильтр высоких частот</string>

</property>

</widget>

<widget class="QLineEdit" name="lineEdit\_2">

<property name="geometry">

<rect>

<x>10</x>

<y>40</y>

<width>161</width>

<height>20</height>

</rect>

</property>

</widget>

<widget class="QLabel" name="label\_2">

<property name="geometry">

<rect>

<x>190</x>

<y>40</y>

<width>141</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Радиус маски (1-500)</string>

</property>

</widget>

<widget class="QLabel" name="label\_3">

<property name="geometry">

<rect>

<x>180</x>

<y>140</y>

<width>151</width>

<height>41</height>

</rect>

</property>

<property name="text">

<string>Путь для сохранения

картинок</string>

</property>

</widget>

<widget class="QPushButton" name="pushButton\_3">

<property name="geometry">

<rect>

<x>10</x>

<y>180</y>

<width>191</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QLineEdit" name="lineEdit\_3">

<property name="geometry">

<rect>

<x>10</x>

<y>150</y>

<width>151</width>

<height>20</height>

</rect>

</property>

</widget>

</widget>

<widget class="QMenuBar" name="menuBar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>341</width>

<height>21</height>

</rect>

</property>

</widget>

<widget class="QToolBar" name="mainToolBar">

<attribute name="toolBarArea">

<enum>TopToolBarArea</enum>

</attribute>

<attribute name="toolBarBreak">

<bool>false</bool>

</attribute>

</widget>

<widget class="QStatusBar" name="statusBar"/>

</widget>

<layoutdefault spacing="6" margin="11"/>

<resources/>

<connections/>

</ui>

*Листинг shumform1.ui*

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>shumForm1</class>

<widget class="QWidget" name="ShumForm1">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>952</width>

<height>819</height>

</rect>

</property>

<property name="windowTitle">

<string>Form</string>

</property>

<widget class="QLabel" name="label">

<property name="geometry">

<rect>

<x>10</x>

<y>30</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_2">

<property name="geometry">

<rect>

<x>260</x>

<y>30</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_3">

<property name="geometry">

<rect>

<x>510</x>

<y>30</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_6">

<property name="geometry">

<rect>

<x>510</x>

<y>280</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_5">

<property name="geometry">

<rect>

<x>260</x>

<y>280</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_4">

<property name="geometry">

<rect>

<x>10</x>

<y>280</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_7">

<property name="geometry">

<rect>

<x>20</x>

<y>530</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_8">

<property name="geometry">

<rect>

<x>270</x>

<y>530</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_9">

<property name="geometry">

<rect>

<x>520</x>

<y>530</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_10">

<property name="geometry">

<rect>

<x>10</x>

<y>10</y>

<width>47</width>

<height>13</height>

</rect>

</property>

<property name="text">

<string>Исходное:</string>

</property>

</widget>

<widget class="QLabel" name="label\_11">

<property name="geometry">

<rect>

<x>270</x>

<y>10</y>

<width>47</width>

<height>13</height>

</rect>

</property>

<property name="text">

<string>Шум:</string>

</property>

</widget>

<widget class="QLabel" name="label\_12">

<property name="geometry">

<rect>

<x>510</x>

<y>10</y>

<width>121</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>После удаления шума:</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox">

<property name="geometry">

<rect>

<x>80</x>

<y>250</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_2">

<property name="geometry">

<rect>

<x>330</x>

<y>250</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_3">

<property name="geometry">

<rect>

<x>590</x>

<y>250</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_4">

<property name="geometry">

<rect>

<x>80</x>

<y>510</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_5">

<property name="geometry">

<rect>

<x>330</x>

<y>510</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_6">

<property name="geometry">

<rect>

<x>590</x>

<y>510</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_7">

<property name="geometry">

<rect>

<x>70</x>

<y>760</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_9">

<property name="geometry">

<rect>

<x>590</x>

<y>760</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_8">

<property name="geometry">

<rect>

<x>330</x>

<y>760</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QLabel" name="label\_13">

<property name="geometry">

<rect>

<x>750</x>

<y>30</y>

<width>161</width>

<height>41</height>

</rect>

</property>

<property name="text">

<string>Отметьте изображения и пропишите путь к сохранению резуьтатаПропишите путь к сохранению

результатов</string>

</property>

</widget>

<widget class="QLineEdit" name="lineEdit">

<property name="geometry">

<rect>

<x>750</x>

<y>80</y>

<width>171</width>

<height>20</height>

</rect>

</property>

</widget>

<widget class="QPushButton" name="pushButton">

<property name="geometry">

<rect>

<x>750</x>

<y>110</y>

<width>171</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QLabel" name="label\_14">

<property name="geometry">

<rect>

<x>756</x>

<y>170</y>

<width>161</width>

<height>21</height>

</rect>

</property>

<property name="text">

<string>Сохранённые картинки:</string>

</property>

</widget>

<widget class="QLabel" name="label\_15">

<property name="geometry">

<rect>

<x>760</x>

<y>200</y>

<width>161</width>

<height>591</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

</widget>

<resources/>

<connections/>

</ui>

*Листинг shumform2.ui*

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>shumForm2</class>

<widget class="QWidget" name="ShumForm2">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>952</width>

<height>819</height>

</rect>

</property>

<property name="windowTitle">

<string>Form</string>

</property>

<widget class="QLabel" name="label">

<property name="geometry">

<rect>

<x>10</x>

<y>30</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_2">

<property name="geometry">

<rect>

<x>260</x>

<y>30</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_3">

<property name="geometry">

<rect>

<x>510</x>

<y>30</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_6">

<property name="geometry">

<rect>

<x>510</x>

<y>280</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_5">

<property name="geometry">

<rect>

<x>260</x>

<y>280</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_4">

<property name="geometry">

<rect>

<x>10</x>

<y>280</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_7">

<property name="geometry">

<rect>

<x>20</x>

<y>530</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_8">

<property name="geometry">

<rect>

<x>270</x>

<y>530</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_9">

<property name="geometry">

<rect>

<x>520</x>

<y>530</y>

<width>231</width>

<height>211</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QLabel" name="label\_10">

<property name="geometry">

<rect>

<x>10</x>

<y>10</y>

<width>47</width>

<height>13</height>

</rect>

</property>

<property name="text">

<string>Исходное:</string>

</property>

</widget>

<widget class="QLabel" name="label\_11">

<property name="geometry">

<rect>

<x>270</x>

<y>10</y>

<width>47</width>

<height>13</height>

</rect>

</property>

<property name="text">

<string>Шум:</string>

</property>

</widget>

<widget class="QLabel" name="label\_12">

<property name="geometry">

<rect>

<x>510</x>

<y>10</y>

<width>121</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>После удаления шума:</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox">

<property name="geometry">

<rect>

<x>80</x>

<y>250</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_2">

<property name="geometry">

<rect>

<x>330</x>

<y>250</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_3">

<property name="geometry">

<rect>

<x>590</x>

<y>250</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_4">

<property name="geometry">

<rect>

<x>80</x>

<y>510</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_5">

<property name="geometry">

<rect>

<x>330</x>

<y>510</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_6">

<property name="geometry">

<rect>

<x>590</x>

<y>510</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_7">

<property name="geometry">

<rect>

<x>70</x>

<y>760</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_9">

<property name="geometry">

<rect>

<x>590</x>

<y>760</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QCheckBox" name="checkBox\_8">

<property name="geometry">

<rect>

<x>330</x>

<y>760</y>

<width>81</width>

<height>17</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QLabel" name="label\_13">

<property name="geometry">

<rect>

<x>750</x>

<y>30</y>

<width>161</width>

<height>41</height>

</rect>

</property>

<property name="text">

<string>Отметьте изображения и пропишите путь к сохранению резуьтатаПропишите путь к сохранению

результатов</string>

</property>

</widget>

<widget class="QLineEdit" name="lineEdit">

<property name="geometry">

<rect>

<x>750</x>

<y>80</y>

<width>171</width>

<height>20</height>

</rect>

</property>

</widget>

<widget class="QPushButton" name="pushButton">

<property name="geometry">

<rect>

<x>750</x>

<y>110</y>

<width>171</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Сохранить</string>

</property>

</widget>

<widget class="QLabel" name="label\_14">

<property name="geometry">

<rect>

<x>756</x>

<y>170</y>

<width>161</width>

<height>21</height>

</rect>

</property>

<property name="text">

<string>Сохранённые картинки:</string>

</property>

</widget>

<widget class="QLabel" name="label\_15">

<property name="geometry">

<rect>

<x>760</x>

<y>200</y>

<width>161</width>

<height>591</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

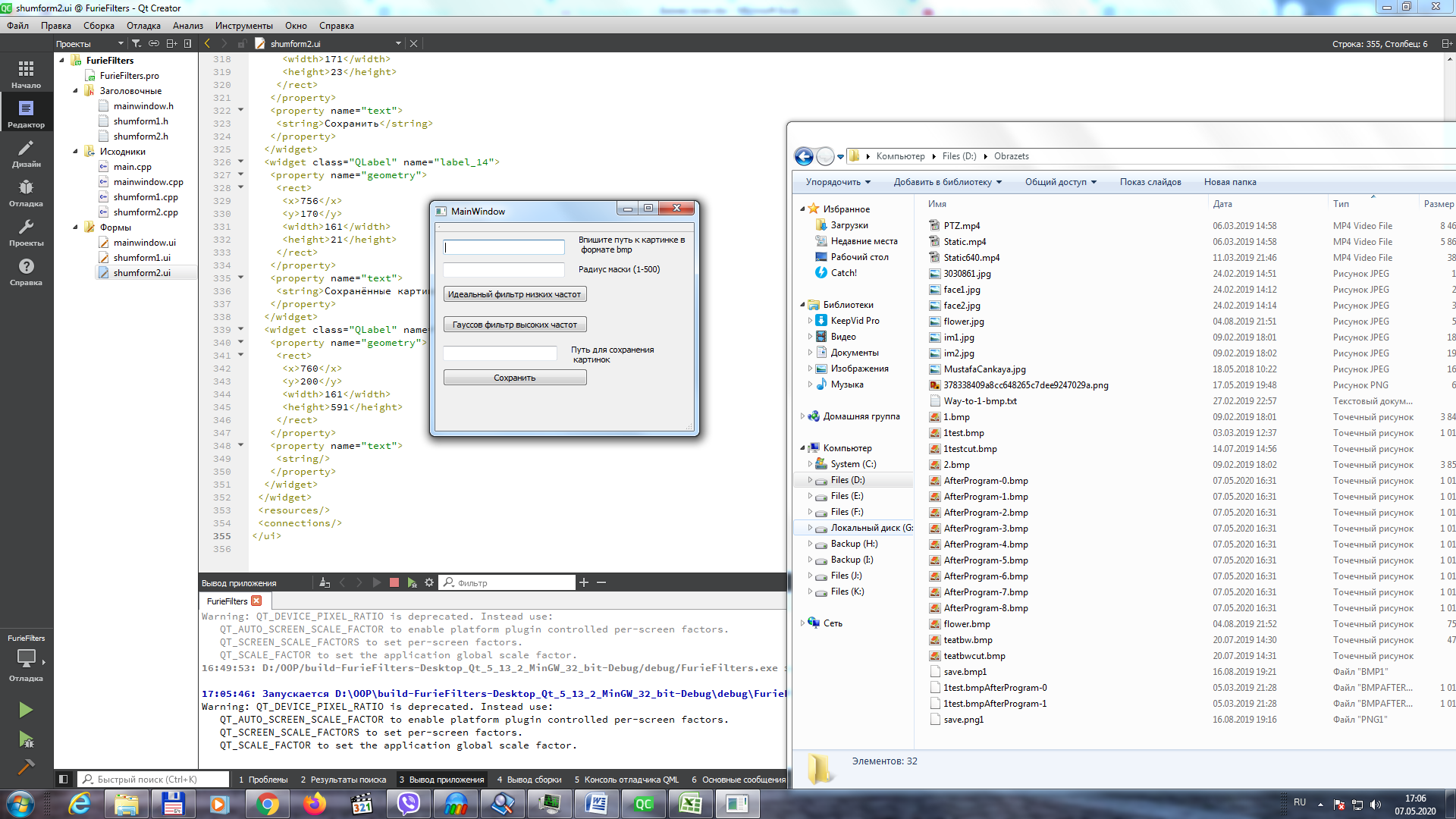
</widget>

<resources/>

<connections/>

</ui>

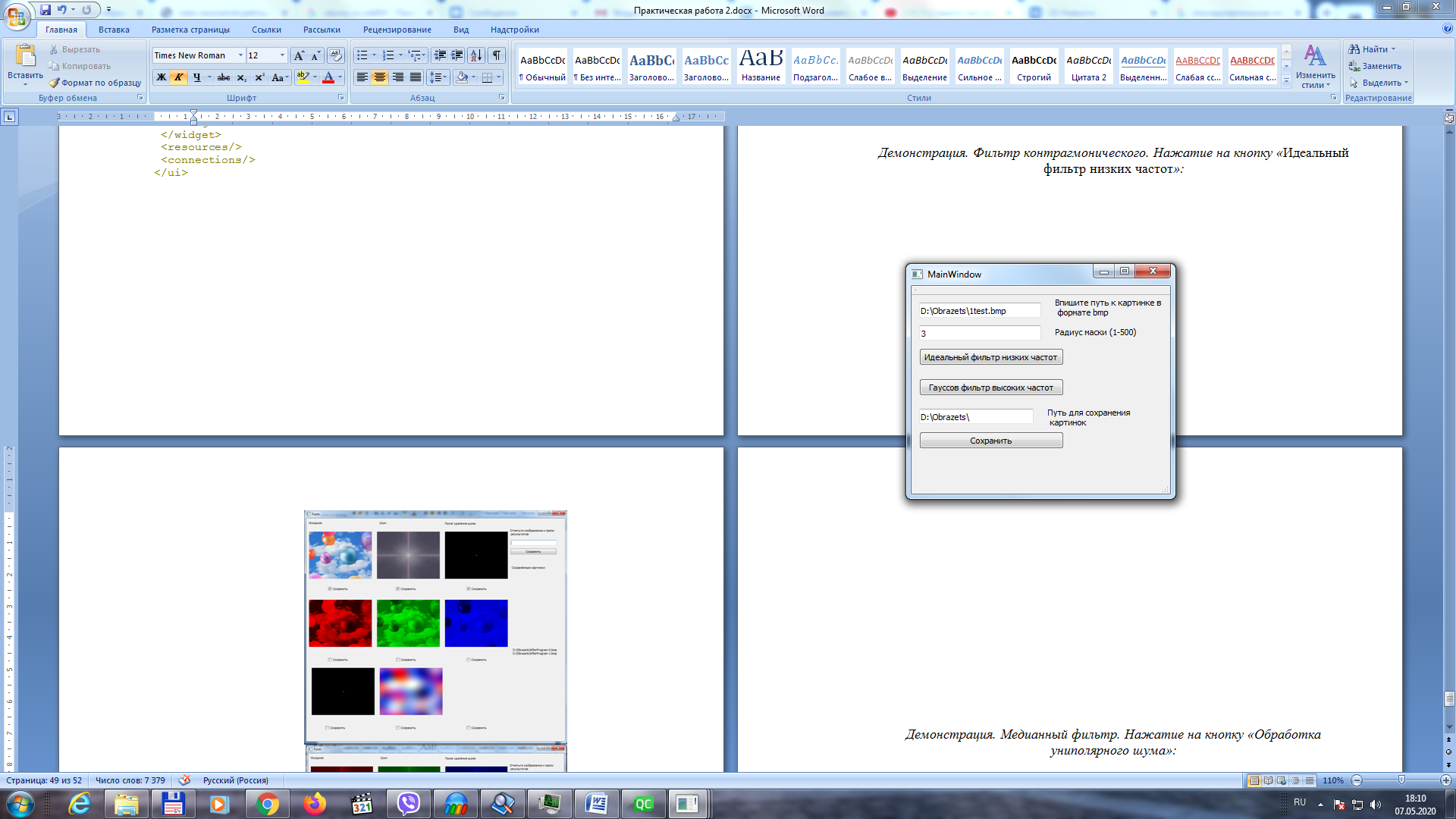
*Результат выполнения:*

**

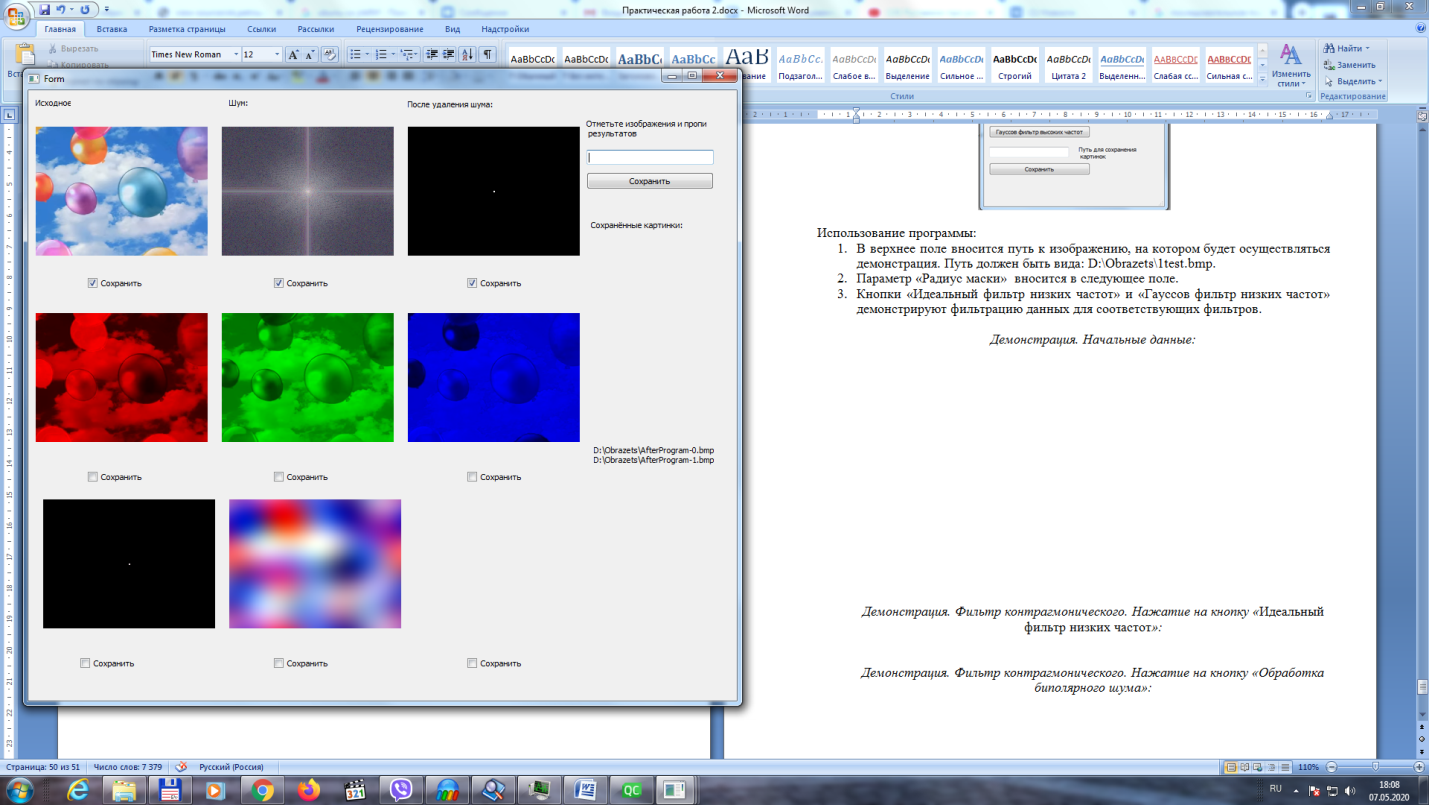
Использование программы:

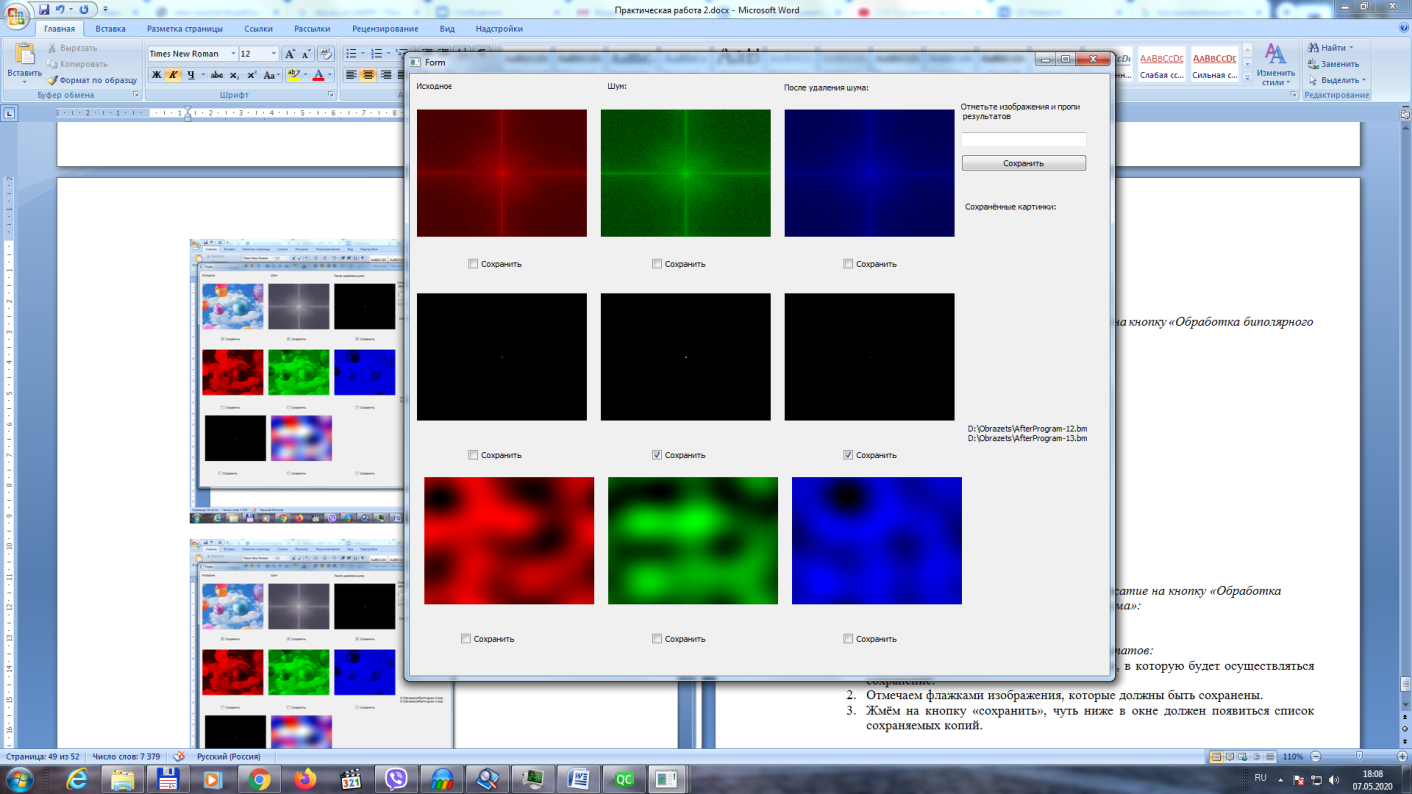
1. В верхнее поле вносится путь к изображению, на котором будет осуществляться демонстрация. Путь должен быть вида: D:\Obrazets\1test.bmp.
2. Параметр «Радиус маски» вносится в следующее поле.
3. Кнопки «Идеальный фильтр низких частот» и «Гауссов фильтр высоких частот» демонстрируют фильтрацию данных для соответствующих фильтров.
4. Нажатие на кнопку «Сохранить» сохраняет отмеченные галочкой сообщения по пути, внесённом в соответствующее поле (чуть выше кнопки).

*Демонстрация. Начальные данные:*



*Демонстрация. Фильтр контрагмонического. Нажатие на кнопку «Идеальный фильтр низких частот»:*

**

**

*Демонстрация. Фильтр контрагмонического. Нажатие на кнопку «Гауссов фильтр высоких частот»:*

