**Описание к блок-схеме приложения rip3p**

Rip3p предназначена для ухода с непереносимой платформы Borland. Название было сохранено, но дополненно. Цифра 3 указывает на работу в трех измерениях, а буква p указывает на возможнось написания плагинов, компилируемых отдельно и без необходимости менять основную программу.

Приложение написано с использованием среды разработки Qt5. Эта среда подразумевает только фиксированную точку входа, но с изменяемыми параметрами. В отличии от предыдущей программы, каждый файл является именем класса, описанного в нем, переведенный в нижний регистр.

1. main.cpp - точка входа в графическое приложение. В нем описаны пользовательские типы, учавствующие в обмене между потоками и указана версия OpenGL. Так же в нем запускается основной класс GWindow(2) и инициализируется потокобезопасное хранилище данных(4 и 5):

**Код:**

#include "typelist.h"

#include "memory.h"

#include "nomain/rdata.h"

#include "gwindow.h"

#include <QApplication>

#include <QSurfaceFormat>

#include <QOpenGLContext>

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

{

QApplication a(argc, argv);

// Настройка openGL

QSurfaceFormat format;

format.setDepthBufferSize(24);

if (QOpenGLContext::openGLModuleType() == QOpenGLContext::LibGL) {

format.setVersion(3, 3);

format.setProfile(QSurfaceFormat::CompatibilityProfile);

}else {

format.setVersion(3, 0);

}

QSurfaceFormat::setDefaultFormat(format);

qRegisterMetaType<Clowd>("Clowd");

qRegisterMetaType<MathVector>("MathVector");

qRegisterMetaType<IntVector>("IntVector");

qRegisterMetaType<IntVector>("IntVector&");

qRegisterMetaType<QByteArray>("QByteArray&");

qRegisterMetaType<Clowd>("Clowd&");

QCoreApplication::setApplicationName("Rip3P");

GWindow w;

// Запуск основного класса

w.show();

return a.exec();

}

2. gwindow.cpp - основное окно программы. В нем реализован функционал MDI приложений и подгрузка и запуск плагинов. Так же в нем инициализируется класс управления проектом MControl(3) и передается ему управление mdi окнами.

**Код:**

#include <iostream>

#include "gwindow.h"

#include "ui\_gwindow.h"

#include <QFileDialog>

#include <QMessageBox>

#include <QVBoxLayout>

#include <QTimer>

#include <QTabWidget>

#include "nomain/extparam.h"

#include "pcap.h"

GWindow::GWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::GWindow),

control(new MControl)

{

ui->setupUi(this);

//определение сетевых устройств

QStringList devices;

pcap\_if\_t \*devlist;

pcap\_if\_t \*dev;

char errbuf[PCAP\_ERRBUF\_SIZE];

if(pcap\_findalldevs(&devlist,errbuf) != -1){

for(dev=devlist;dev;dev=dev->next)

devices << QString::fromLocal8Bit(dev->name);

}

QList<QAction \*> naList;

bool checkedAction = false;

QActionGroup \*naGroup = new QActionGroup(this);

foreach(QString dev, devices){

QAction \*na = new QAction(dev);

na->setCheckable(true);

if(Memory::get("device","").toString()==dev){

checkedAction = true;

na->setChecked(true);

}

else

na->setChecked(false);

connect(na,&QAction::triggered,[=](bool type){

Q\_UNUSED(type);

control->setDev(dev);

});

naList.append(na);

naGroup->addAction(na);

}

ui->devInterface->addActions(naList);

if(!checkedAction && naList.size() > 0){

naList[0]->setChecked(true);

}

//Настройка основных виджетов

pb = new QProgressBar();

pb->setMaximum(100);

pb->setMinimum(0);

pb->setTextVisible(true);

logLabel = new QLabel();

infoLabel = new QLabel();

ui->statusBar->addWidget(logLabel,1);

ui->statusBar->addWidget(infoLabel,1);

ui->statusBar->addWidget(pb,1);

pb->hide();

// ui->statusBar->setLayoutDirection(Qt::LayoutDirection);

this->setDockNestingEnabled(false);

this->setDockOptions(QMainWindow::AnimatedDocks | QMainWindow::AllowTabbedDocks | QMainWindow::ForceTabbedDocks);

ui->dockVisual->hide();

control->setMDI(ui->mdiArea);

connect(control,&MControl::setProgress,this,&GWindow::progress);

connect(control,&MControl::setLog,this,&GWindow::log);

connect(control,&MControl::loadPlugin,this,&GWindow::loadPlugin);

connect(control,&MControl::hidePlugin,this,&GWindow::hidePlugin);

control->load();

debug = control->debug;

addDock("debug","Панель отладки");

QActionGroup\* group = new QActionGroup( this );

ui->mmGpolarization->setCheckable(true);

ui->mmVpolarization->setCheckable(true);

ui->mmGpolarization->setActionGroup(group);

ui->mmVpolarization->setActionGroup(group);

ui->mmGpolarization->setChecked(Memory::get("Gpolarization",true).toBool());

ui->mmVpolarization->setChecked(Memory::get("Vpolarization",false).toBool());

fs = new formSettings;

connect(fs,&formSettings::save,control,&MControl::saveConfig);

connect(fs,&formSettings::sendParam,control,&MControl::sendParam);

connect(fs,&formSettings::sendMsg,control,&MControl::sendMsg);

fs->setleSubBufNum(Memory::get("leSubBufNum",4).toInt());

fs->setleFreq(Memory::get("leFreq",1777).toInt());

fs->setleBurstLen(Memory::get("leBurstLen",1).toInt());

fs->setleBurstLen(Memory::get("leBurstLen",1).toInt());

fs->setleGeterodin(Memory::get("leGeterodin",8000).toInt());

fs->setleAmp(Memory::get("leAmp",100).toInt());

fs->setleFreqRange(Memory::get("leFreqRange",100).toInt());

fs->setleGateDelay(Memory::get("leGateDelay",0).toInt());

fs->setleGateDuration(Memory::get("leGateDuration",0.01).toDouble());

fs->setcbPulseMod(Memory::get("cbPulseMod",true).toBool());

fs->setcbUWB(Memory::get("cbUWB",false).toBool());

fs->setcbLFM(Memory::get("cbLFM",false).toBool());

fs->setlePeriod(Memory::get("lePeriod",0.1).toDouble());

fs->setleDuration(Memory::get("leDuration",0.0033).toDouble());

fs->setcbGate(Memory::get("cbGate",false).toBool());

fs->setcbCont(Memory::get("cbCont",false).toBool());

fs->setcbCoherentAccum(Memory::get("cbCoherentAccum",false).toBool());

fs->setcbDDSReset(Memory::get("cbDDSReset",true).toBool());

fs->setcbLOGM(Memory::get("cbLOGM",false).toBool());

fs->setrbDdsRstBurst(Memory::get("rbDdsRstBurst",false).toBool());

fs->setrbDdsRstPulse(Memory::get("rbDdsRstPulse",true).toBool());

fs->setrbTxPolXX(Memory::get("rbTxPolXX",true).toBool());

fs->setrbTxPolXY(Memory::get("rbTxPolXY",false).toBool());

fs->setrbTxPolYX(Memory::get("rbTxPolYX",false).toBool());

fs->setrbTxPolYY(Memory::get("rbTxPolYY",false).toBool());

fs->setrbRxPolXX(Memory::get("rbRxPolXX",false).toBool());

fs->setrbRxPolXY(Memory::get("rbRxPolXY",false).toBool());

fs->setrbRxPolYX(Memory::get("rbRxPolYX",false).toBool());

fs->setrbRxPolYY(Memory::get("rbRxPolYY",false).toBool());

fs->setleTxAtt(Memory::get("leTxAtt",1).toInt());

fs->setrlsIP(Memory::get("rlsIP","192.168.114.100").toString());

fs->setrlsPort(Memory::get("rlsPort",30583).toInt());

fs->setcbMGEN(Memory::get("cbMGEN",false).toBool());

fs->setseMLEN(Memory::get("seMLEN",3).toInt());

fs->setleRxAtt(Memory::get("leRxAtt",1).toInt());

fs->setrbRxAnt0(Memory::get("rbRxAnt0",false).toBool());

fs->setrbRxAnt1(Memory::get("rbRxAnt1",false).toBool());

//Добавление дополнительных виджетов (окон)

dw["fs"] = new QDockWidget("Настройка сканирования",this);

dw["fs"]->setFloating(true);

dw["fs"]->setAllowedAreas(Qt::LeftDockWidgetArea | Qt::RightDockWidgetArea);

//this->addDockWidget(Qt::RightDockWidgetArea,dw["fs"]);

dw["fs"]->setWidget(fs);

this->setDockNestingEnabled(false);

dw["fs"]->hide();

this->setTabPosition(Qt::RightDockWidgetArea,QTabWidget::East);

this->setTabPosition(Qt::LeftDockWidgetArea,QTabWidget::West);

// setTabPosition(Qt::RightDockWidgetArea,QTabWidget::TabPosition::East);

//setTabPosition(Qt::LeftDockWidgetArea,QTabWidget::TabPosition::West);

ui->Barier->setValue(Memory::get("Barier",0).toInt());

ui->PhMin->setValue(Memory::get("PhMin",0).toInt());

ui->ArgMax->setValue(Memory::get("ArgMax",1024).toInt());

ui->ArgMin->setValue(Memory::get("ArgMin",0).toInt());

this->showMaximized();

sf = new SaveFile();

connect(sf,&SaveFile::sync,control,&MControl::saveConfig);

sf->hide();

cf = new ControlForm;

connect(cf,&ControlForm::sync,control,&MControl::saveConfig);

connect(cf,&ControlForm::sendMsg,control,&MControl::sendMsg);

dw["cf"] = new QDockWidget("Запрос",this);

dw["cf"]->setFloating(true);

dw["cf"]->setAllowedAreas(Qt::LeftDockWidgetArea | Qt::RightDockWidgetArea);

dw["cf"]->setWidget(cf);

this->setDockNestingEnabled(false);

dw["cf"]->hide();

pl = new PolarisationGet;

connect(pl,&PolarisationGet::sync,control,&MControl::saveConfig);

dw["pl"] = new QDockWidget("Настройка поляризации",this);

dw["pl"]->setFloating(true);

dw["pl"]->setAllowedAreas(Qt::LeftDockWidgetArea | Qt::RightDockWidgetArea);

//this->addDockWidget(Qt::RightDockWidgetArea,dw["fs"]);

dw["pl"]->setWidget(pl);

this->setDockNestingEnabled(false);

dw["pl"]->hide();

}

GWindow::~*GWindow*()

{

delete ui;

delete control;

}

/\*

\* Загрузка сторонних плагинов

\*/

void GWindow::loadPlugin(Plugs &plugin){

if(plugins.contains(plugin.alias)){

plugActions[plugin.alias]->setVisible(true);

return;

}

QPluginLoader loader(plugin.path);

if(loader.load()){

plugins[plugin.alias] = plugin;

PlugWidgetInterfaces \*pluginI = qobject\_cast<PlugWidgetInterfaces \*>(loader.instance());

plugObject[plugin.alias] = pluginI;

dw[plugin.alias] = new QDockWidget(plugin.name,this);

dw[plugin.alias]->setFloating(true);

dw[plugin.alias]->setAllowedAreas(Qt::LeftDockWidgetArea | Qt::RightDockWidgetArea);

dw[plugin.alias]->setWidget(pluginI);

this->setDockNestingEnabled(false);

dw[plugin.alias]->hide();

QAction \*pAction = ui->menuPlugins->addAction(plugin.name);

plugActions[plugin.alias] = pAction;

connect(pAction,&QAction::triggered,dw[plugin.alias],&QDockWidget::show);

plugActions[plugin.alias]->setVisible(true);

connect(pluginI,SIGNAL(processStep(int)),control,SLOT(shared(int)));

connect(pluginI,SIGNAL(sync()),control,SLOT(saveConfig()));

connect(control,SIGNAL(sync()),pluginI,SLOT(syncSlot()));

pluginI->*setMemory*(Memory::dLink);

pluginI->*init*();

}

}

/\*

\* Скрытие неиспользуемых плагинов

\*/

void GWindow::hidePlugin(Plugs &plugin){

if(plugins.contains(plugin.alias))

plugActions[plugin.alias]->setVisible(false);

else {

loadPlugin(plugin);

// plugActions[plugin.alias]->setVisible(false);

}

}

/\*

\* Отображение отладочной информации в панели отладки

\*/

void GWindow::log(QString msg){

logLabel->setText(msg);

}

/\*

\* Конец загрузки файла или сканирования

\*/

void GWindow::progressTimerNum(){

pb->setValue(0);

pb->hide();

}

/\*

\* Метод, отвечающий за синхронизацию настроек и значений в виджете

\*/

void GWindow::sync(){

ui->PhMin->setValue(Memory::get("PhMin",0).toInt());

ui->ArgMax->setValue(Memory::get("ArgMax",1024).toInt());

ui->ArgMin->setValue(Memory::get("ArgMin",0).toInt());

control->debug->log("ArgMax: "+QString::number(ui->ArgMax->value()));

control->debug->log("ArgMin: "+QString::number(ui->ArgMin->value()));

}

/\*

\* Метод, отображающий изменения в прогресс-баре

\*/

void GWindow::progress(int num){

if(!pb->isVisible())

pb->show();

pb->setValue(num);

if(num==100){

QTimer::singleShot(500, this, SLOT(progressTimerNum()));

}

}

/\*

\* Метод, с помощью которого добавляются виджеты в основное окно

\*/

void GWindow::addDock(QString name, QString title){

dw[name] = new QDockWidget(title,this);

dw[name]->setFloating(true);

dw[name]->setAllowedAreas(Qt::LeftDockWidgetArea | Qt::RightDockWidgetArea);

//this->addDockWidget(Qt::RightDockWidgetArea,dw[name]);

if(name=="debug"){

dw[name]->setWidget(debug);

}

this->setDockNestingEnabled(false);

dw[name]->hide();

}

/\*

\* Событие нажатия кнопки "Замостить"

\*/

void GWindow::on\_mmTile\_triggered()

{

this->ui->mdiArea->tileSubWindows();

}

/\*

\* Событие нажатия кнопки "3D модель"

\*/

void GWindow::on\_mm3Dmodel\_triggered()

{

if(Memory::get("Vpolarization",true).toBool())

control->showGr3D("vertical");

else

control->showGr3D("gorizontal");

}

/\*

\* Событие нажатия кнопки "Горизонтальная поляризация"

\*/

void GWindow::on\_mmGpolarization\_triggered(bool checked)

{

control->debug->log("Gpolarization:"+QString::number(checked));

control->setAction("Gpolarization",checked);

control->setAction("Vpolarization",!checked);

control->debug->log("Gpolarization:"+QString::number(Memory::get("Gpolarization",false).toBool()));

}

/\*

\* Событие нажатия кнопки "Вертикальная поляризация"

\*/

void GWindow::on\_mmVpolarization\_triggered(bool checked)

{

control->debug->log("Vpolarization:"+QString::number(checked));

control->setAction("Gpolarization",!checked);

control->setAction("Vpolarization",checked);

control->debug->log("Vpolarization:"+QString::number(Memory::get("Vpolarization",false).toBool()));

}

/\*

\* Событие нажатия кнопки "Панель отладки"

\* Визуализируется виджет отладки

\*/

void GWindow::on\_mmDebug\_triggered()

{

dw["debug"]->show();

this->setDockNestingEnabled(false);

}

/\*

\* Событие нажатия кнопки "Панель отладки"

\* Открывается окно управления плагинами

\*/

void GWindow::on\_mmPlugs\_triggered()

{

control->winOpen("plugin");

}

/\*

\* Событие нажатия кнопки "Настройки сигнала"

\* Открывается окно-виджет настроек РЛМ

\*/

void GWindow::on\_mmSignal\_triggered()

{

dw["fs"]->show();

this->setDockNestingEnabled(false);

}

/\*

\* Событие нажатия кнопки "Загрузить" из меню "Файл"

\* Открывается окно выбора файла.

\*/

void GWindow::on\_mmLoad\_triggered()

{

QString fileName = QFileDialog::getOpenFileName(this, tr("Файл 3d объекта"), Memory::get("lastFileDir","").toString(),

tr("Бинарные файлы (\*.dat);;Бинарные файлы (\*.3d);;Бинарные файлы (\*.3df);;Все файлы (\*.\*)"));

if (fileName != "") { //Если файл выбран, то определяем расширение

control->log("Загрузка файла");

QFile file(fileName);

Memory::set("lastFileDir",fileName);

control->saveConfig();

if (!file.*open*(QIODevice::ReadOnly)) {

QMessageBox::critical(this, tr("Error"), tr("Could not open file"));

return;

}

QStringList fNameSpl = fileName.split(".");

QString ext = fNameSpl[fNameSpl.size()-1];

control->debug->log("ext: "+ext);

QByteArray data = file.readAll();

if(ext=="3d"){ //если 3d, тогда вызываем окно выбора размера кадра

ExtParam \*ex = new ExtParam;

ex->setData(fileName,data);

ex->show();

// После определения размеров вызываем Worker::loadFinished

connect(ex,&ExtParam::load,control,&MControl::loadData);

connect(ex,&ExtParam::sync,this,&GWindow::sync);

connect(ex,&ExtParam::log,control,&MControl::log);

return;

}

// Если dat, вызываем Worker::loadSrc или Worker::loadFinishedF

emit log(QString::number(data.size()));

control->loadData(fileName,data);

}

}

/\*

\* Событие изменения максимальной дальности

\* Сохраняем новое значение и распространяем событие

\*/

void GWindow::on\_ArgMax\_valueChanged(int value)

{

Memory::set("ArgMax",value);

control->saveConfig();

}

/\*

\* Событие изменения минимальное дальности

\* Сохраняем новое значение и распространяем событие

\*/

void GWindow::on\_ArgMin\_valueChanged(int value)

{

Memory::set("ArgMin",value);

control->saveConfig();

}

/\*

\* Событие изменения угла

\* Сохраняем новое значение и распространяем событие

\*/

void GWindow::on\_PhMin\_valueChanged(int value)

{

Memory::set("PhMin",value);

control->saveConfig();

}

/\*

\* Событие изменения Порога отображения

\* Сохраняем новое значение и распространяем событие

\*/

void GWindow::on\_Barier\_valueChanged(int value)

{

Memory::set("Barier",value);

control->saveConfig();

}

/\*

\* Событие нажания кнопки "Растр"

\* Открывает окно растрового графика

\*/

void GWindow::on\_mmRastr\_triggered()

{

if(Memory::get("Vpolarization",true).toBool())

control->showPlotRastr("vertical");

else

control->showPlotRastr("gorizontal");

}

/\*

\* Событие нажания кнопки "АФ Растр"

\* Открывает окно графика АФ Растра

\*/

void GWindow::on\_mmAFRastr\_triggered()

{

if(Memory::get("Vpolarization",true).toBool())

control->showPlotPolarization("vertical");

else

control->showPlotPolarization("gorizontal");

}

/\*

\* Событие нажания кнопки "Сохранить"

\* Открывает окно сохрания резутьтата в файл

\*/

void GWindow::on\_mmSave\_triggered()

{

sf->show();

}

/\*

\* Событие нажания кнопки "Непрерывный запрос"

\* Открывает окно запуска режима работы "Непрерывный запрос"

\*/

void GWindow::on\_mmControlForm\_triggered()

{

dw["cf"]->show();

}

/\*

\* Событие нажания кнопки "Осцилограмма"

\* Открывает окно графика Осцилограмма

\*/

void GWindow::on\_mmOsc\_triggered()

{

if(Memory::get("Vpolarization",true).toBool())

control->showPlotOsc("vertical");

else

control->showPlotOsc("gorizontal");

}

/\*

\* Событие нажания кнопки "Настройка поляризации"

\* Открывает окно настроек поляризации

\*/

void GWindow::on\_mmPolarization\_triggered()

{

dw["pl"]->show();

}

3. mcontrol.cpp - класс управления. Это центральный узел проекта. Он инициирует и распределяет сигналы и слоты между классами, участвующими в работе. Так же в нем заложен функционал управления настройками - сохранения в файл, в случае внесения изменений и загрузка при инициализации. Параметры после инициализации переносятся в потокобезопасное хранилище данных(4 и 5) и при сохранении берутся от туда. Так же класс при подаче команды от GWindow(2) создает mdi окна и присоединяет его сигналы к основным массивам сигналов. При запуске он создает только DebugDialog (16) и Worker(6). Остальные, только по команде от MControl(3)

**Код:**

#include "mcontrol.h"

#include "memory.h"

#include <QDockWidget>

#include <iostream>

MControl::MControl(QObject \*parent) : QObject(parent), debug(new DebugDialog),

isgr3dVmdi(false), isgr3dGmdi(false), isgrPlotVmdi(false), isgrPlotGmdi(false), isgrRPlotVmdi(false), isgrRPlotGmdi(false)

{

// получение ранее сохраненных настроек

settings = new QSettings("rip3p.ini",QSettings::IniFormat);

Memory::resultData["Gorizontal"].clear();

Memory::resultData["Vertical"].clear();

QStringList keys = settings->childKeys();

foreach (QString key, keys) {

Memory::set(key,settings->value(key));

}

// Запуск основного рабочего класса и перенаправление его в отдельный поток

worker = new Worker;

worker->moveToThread(&workerThread);

connect(&workerThread, &QThread::finished, worker, &QObject::deleteLater);

connect(this,&MControl::loadSrc,worker,&Worker::loadSrc);

connect(this,&MControl::loadFinished,worker,&Worker::loadFinished);

connect(this,&MControl::loadFinishedF,worker,&Worker::loadFinishedF);

connect(this,&MControl::sync,worker,&Worker::sync);

connect(this,&MControl::sendParamSignals,worker,&Worker::sendParam);

connect(this,&MControl::sendMsgSignal,worker,&Worker::sendMsgSlot);

connect(worker,&Worker::save,this,&MControl::saveConfig);

connect(worker,&Worker::log,this,&MControl::log);

connect(worker,&Worker::progress,this,&MControl::progress);

connect(worker,&Worker::shared,this,&MControl::shared);

connect(worker,&Worker::resultGorizontal,this,&MControl::resultXX);

connect(worker,&Worker::resultVertical,this,&MControl::resultYY);

workerThread.start();

hasGData = false;

hasVData = false;

saveTimer = new QTimer();

saveTimer->setInterval(300);

connect(saveTimer,&QTimer::timeout,this,&MControl::saveConfigTimer);

}

MControl::~*MControl*(){

workerThread.quit();

workerThread.wait();

}

/\*

\* Прокси-метод для вызова метода Worker::sendParam

\*/

void MControl::sendParam(){

emit sendParamSignals();

}

/\*

\* Метод-сигнал изменения сетевого устройства

\*/

void MControl::setDev(QString dev){

Memory::set("device",dev);

saveConfigTimer();

}

/\*

\* Прокси-метод, для вызова Worker::sendMsgSlot

\*/

void MControl::sendMsg(unsigned short BufferSize, unsigned char \*Buffer, unsigned short CmdNum){

emit sendMsgSignal(BufferSize,Buffer,CmdNum);

}

/\*

\* Создание окна управления плагинами

\*/

void MControl::load(){

plugin = new PlugWin;

connect(plugin,&PlugWin::sync,this,&MControl::saveConfig);

connect(plugin,&PlugWin::loadPlugin,this,&MControl::loadPlugin);

connect(plugin,&PlugWin::hidePlugin,this,&MControl::hidePlugin);

plugin->load();

}

/\*

\* Синхронизация изменения настроек программы (срабатывает от таймера)

\*/

void MControl::saveConfigTimer(){

saveTimer->stop();

QMap<QString, QVariant>::const\_iterator i = Memory::dLink->list.constBegin();

while (i != Memory::dLink->list.constEnd()) {

settings->setValue(i.key(),i.value());

++i;

}

settings->sync();

emit sync();

if(isgr3dVmdi){

Memory::scL["Vertical"]->sync();

}

if(isgr3dGmdi){

Memory::scL["Gorizontal"]->sync();

}

}

/\*

\* Запуск таймера синхронизации. Если таймер работает, прекращаем работу и после этого запускаем

\*/

void MControl::saveConfig(){

if(saveTimer->isActive()){

saveTimer->stop();

}

saveTimer->start();

//Memory::scL

}

/\*

\* Метод вызывается, когда файл полностью отработан. (горизонтальная поляризация)

\* Получает финальные данные и отдает из всем графикам

\*/

void MControl::resultXX(Clowd &dataA,Clowd &dataH){

int sizeA = dataA.size()\*sizeof(float);

int sizeH = dataH.size()\*sizeof(float);

Memory::setData("dGorizontalAr",dataA.data(),sizeA);

Memory::setData("vGorizontalAr",dataA.data(),sizeA);

Memory::setData("dGorizontalPh",dataH.data(),sizeH);

Memory::setData("vGorizontalPh",dataH.data(),sizeH);

hasGData = true;

if(isgr3dGmdi){

Memory::scL["Gorizontal"]->setBuf(dataA,dataH);

gr3dG->pgl->stopTimer();

}

if(isgrPlotGmdi){

grPlotG->setBuf(dataA,dataH);

grPlotG->plot();

}

if(isgrRPlotGmdi){

grRPlotG->setBuf(dataA,dataH);

grRPlotG->plot();

}

if(isgrOPlotGmdi){

grOPlotG->shared(Memory::get("Size",1).toInt());

}

}

/\*

\* Метод вызывается, когда файл полностью отработан. (вертикальная поляризация)

\* Получает финальные данные и отдает из всем графикам

\*/

void MControl::resultYY(Clowd &dataA, Clowd &dataH){

int sizeA = dataA.size()\*sizeof(float);

int sizeH = dataH.size()\*sizeof(float);

Memory::setData("dVerticalAr",dataA.data(),sizeA);

Memory::setData("vVerticalAr",dataA.data(),sizeA);

Memory::setData("dVerticalPh",dataH.data(),sizeH);

Memory::setData("vVerticalPh",dataH.data(),sizeH);

hasVData = true;

if(isgr3dVmdi){

Memory::scL["Vertical"]->setBuf(dataA,dataH);

gr3dV->pgl->stopTimer();

}

if(isgrPlotVmdi){

grPlotV->setBuf(dataA,dataH);

grPlotV->plot();

}

if(isgrRPlotVmdi){

grRPlotV->setBuf(dataA,dataH);

grRPlotV->plot();

}

if(isgrOPlotVmdi){

grOPlotV->shared(Memory::get("Size",1).toInt());

}

}

/\*

\* Обработка команды shared

\* Передает её всем активным графикам

\*/

void MControl::shared(int shp){

if(isgr3dVmdi){

Memory::scL["Vertical"]->shared(shp);

gr3dV->pgl->updateTimer();

}

if(isgr3dGmdi){

Memory::scL["Gorizontal"]->shared(shp);

gr3dG->pgl->updateTimer();

}

if(isgrPlotGmdi){

grPlotG->shared(shp);

}

if(isgrPlotVmdi){

grPlotV->shared(shp);

}

if(isgrRPlotGmdi){

grRPlotG->shared(shp);

}

if(isgrRPlotVmdi){

grRPlotV->shared(shp);

}

if(isgrOPlotGmdi){

grOPlotG->shared(shp);

}

if(isgrOPlotVmdi){

grOPlotV->shared(shp);

}

}

/\*

\* Метод по расширению файла определяет вызываемых обработчик

\*/

void MControl::loadData(QString fileName, QByteArray &data){

QStringList ex = fileName.split(".");

QString extension = ex[ex.length()-1];

if(extension=="dat"){

emit loadSrc(data);

}

else if(extension=="3d"){

emit loadFinished(data);

}

else if(extension=="3df"){

emit loadFinishedF(data);

}

}

void MControl::setMDI(QMdiArea \*lnk){

area = lnk;

}

/\*

\* Открытие 3D графика

\*/

void MControl::showGr3D(QString sType){

if(sType=="vertical"){

if(!isgr3dVmdi){

isgr3dVmdi = true;

gr3dV = new Gr3D();

gr3dV->setType("Vertical");

connect(gr3dV,&Gr3D::sync,this,&MControl::saveConfig);

gr3dVmdi = area->addSubWindow(gr3dV);

gr3dVmdi->setWindowTitle("3D Вертикальная поляризация");

gr3dVmdi->resize(400,400);

connect(gr3dVmdi,SIGNAL(destroyed()),this,SLOT(isgr3dVmdiHide()));

if(hasVData){

// Если данные уже есть, то отображаем

Clowd bufA, bufP;

int size = Memory::get("Size",1024).toInt()\*BLOCKLANGTH;

bufA.resize(size);

bufP.resize(size);

Memory::getData("vVerticalAr",bufA.data(),size\*sizeof(float));

Memory::getData("vVerticalPh",bufP.data(),size\*sizeof(float));

Memory::scL["Vertical"]->setBuf(bufA,bufP);

}

}

gr3dVmdi->show();

}

else{

if(!isgr3dGmdi){

isgr3dGmdi = true;

gr3dG = new Gr3D();

gr3dG->setType("Gorizontal");

connect(gr3dG,&Gr3D::sync,this,&MControl::saveConfig);

gr3dGmdi = area->addSubWindow(gr3dG);

gr3dGmdi->setWindowTitle("3D Горизонтальная поляризация");

gr3dGmdi->resize(400,400);

connect(gr3dGmdi,SIGNAL(destroyed()),this,SLOT(isgr3dGmdiHide()));

if(hasGData){

// Если данные уже есть, то отображаем

Clowd bufA, bufP;

int size = Memory::get("Size",1024).toInt()\*BLOCKLANGTH;

bufA.resize(size);

bufP.resize(size);

Memory::getData("vGorizontalAr",bufA.data(),size\*sizeof(float));

Memory::getData("vGorizontalPh",bufP.data(),size\*sizeof(float));

Memory::scL["Gorizontal"]->setBuf(bufA,bufP);

}

}

gr3dGmdi->show();

}

}

void MControl::showPlotRastr(QString sType){

if(sType=="vertical"){

if(!isgrRPlotVmdi){

isgrRPlotVmdi = true;

grRPlotV = new PlotRaster();

grRPlotV->setType("Vertical");

grRPlotV->syncSlot();

connect(grRPlotV,&PlotRaster::sync,this,&MControl::saveConfig);

connect(this,&MControl::sync,grRPlotV,&PlotRaster::syncSlot);

grRPlotVmdi = area->addSubWindow(grRPlotV);

grRPlotVmdi->setWindowTitle("Вертикальная поляризация");

grRPlotVmdi->resize(400,400);

connect(grRPlotVmdi,SIGNAL(destroyed()),this,SLOT(isgrRPlotVmdiHide()));

if(hasVData){

// Если данные уже есть, то отображаем

Clowd bufA, bufP;

int size = Memory::get("Size",1024).toInt()\*BLOCKLANGTH;

bufA.resize(size);

bufP.resize(size);

Memory::getData("vVerticalAr",bufA.data(),size\*sizeof(float));

Memory::getData("vVerticalPh",bufP.data(),size\*sizeof(float));

grRPlotV->setBuf(bufA,bufP);

grRPlotV->plot();

}

}

grRPlotVmdi->show();

}else {

if(!isgrRPlotGmdi){

isgrRPlotGmdi = true;

grRPlotG = new PlotRaster();

grRPlotG->setType("Gorizontal");

grRPlotG->syncSlot();

connect(grRPlotG,&PlotRaster::sync,this,&MControl::saveConfig);

connect(this,&MControl::sync,grRPlotG,&PlotRaster::syncSlot);

grRPlotGmdi = area->addSubWindow(grRPlotG);

grRPlotGmdi->setWindowTitle("Горизонтальная поляризация");

grRPlotGmdi->resize(400,400);

connect(grRPlotGmdi,SIGNAL(destroyed()),this,SLOT(isgrRPlotGmdiHide()));

if(hasGData){

// Если данные уже есть, то отображаем

Clowd bufA, bufP;

int size = Memory::get("Size",1024).toInt()\*BLOCKLANGTH;

bufA.resize(size);

bufP.resize(size);

Memory::getData("vGorizontalAr",bufA.data(),size\*sizeof(float));

Memory::getData("vGorizontalPh",bufP.data(),size\*sizeof(float));

grRPlotG->setBuf(bufA,bufP);

grRPlotG->plot();

}

}

grRPlotGmdi->show();

}

}

void MControl::showPlotOsc(QString sType){

if(sType=="vertical"){

if(!isgrOPlotVmdi){

isgrOPlotVmdi = true;

grOPlotV = new PlotOsc();

grOPlotV->setType("Vertical");

grOPlotV->syncSlot();

connect(grOPlotV,&PlotOsc::sync,this,&MControl::saveConfig);

connect(this,&MControl::sync,grOPlotV,&PlotOsc::syncSlot);

grOPlotVmdi = area->addSubWindow(grOPlotV);

grOPlotVmdi->setWindowTitle("Вертикальная осцилограмма");

grOPlotVmdi->resize(400,400);

connect(grOPlotVmdi,SIGNAL(destroyed()),this,SLOT(isgrOPlotVmdiHide()));

if(hasVData){

// Если данные уже есть, то отображаем

int size = Memory::get("Size",1024).toInt();

grOPlotV->shared(size);

grOPlotV->plot();

}

}

grOPlotVmdi->show();

}else {

if(!isgrOPlotGmdi){

isgrOPlotGmdi = true;

grOPlotG = new PlotOsc();

grOPlotG->setType("Gorizontal");

grOPlotG->syncSlot();

connect(grOPlotG,&PlotOsc::sync,this,&MControl::saveConfig);

connect(this,&MControl::sync,grOPlotG,&PlotOsc::syncSlot);

grOPlotGmdi = area->addSubWindow(grOPlotG);

grOPlotGmdi->setWindowTitle("Горизонтальная осцилограмма");

grOPlotGmdi->resize(400,400);

connect(grOPlotGmdi,SIGNAL(destroyed()),this,SLOT(isgrOPlotGmdiHide()));

if(hasGData){

// Если данные уже есть, то отображаем

int size = Memory::get("Size",1024).toInt();

grOPlotG->shared(size);

grOPlotG->plot();

}

}

grOPlotGmdi->show();

}

}

void MControl::showPlotPolarization(QString sType){

if(sType=="vertical"){

if(!isgrPlotVmdi){

isgrPlotVmdi = true;

grPlotV = new PlotPolarization();

grPlotV->setType("Vertical");

grPlotV->syncSlot();

connect(grPlotV,&PlotPolarization::sync,this,&MControl::saveConfig);

connect(this,&MControl::sync,grPlotV,&PlotPolarization::syncSlot);

grPlotVmdi = area->addSubWindow(grPlotV);

grPlotVmdi->setWindowTitle("Вертикальная поляризация");

grPlotVmdi->resize(400,400);

connect(grPlotVmdi,SIGNAL(destroyed()),this,SLOT(isgrPlotVmdiHide()));

if(hasVData){

// Если данные уже есть, то отображаем

Clowd bufA, bufP;

int size = Memory::get("Size",1024).toInt()\*BLOCKLANGTH;

bufA.resize(size);

bufP.resize(size);

Memory::getData("vVerticalAr",bufA.data(),size\*sizeof(float));

Memory::getData("vVerticalPh",bufP.data(),size\*sizeof(float));

grPlotV->setBuf(bufA,bufP);

grPlotV->plot();

}

}

grPlotVmdi->show();

}

else {

if(!isgrPlotGmdi){

isgrPlotGmdi = true;

grPlotG = new PlotPolarization();

grPlotG->setType("Gorizontal");

grPlotG->syncSlot();

connect(grPlotG,&PlotPolarization::sync,this,&MControl::saveConfig);

connect(this,&MControl::sync,grPlotG,&PlotPolarization::syncSlot);

grPlotGmdi = area->addSubWindow(grPlotG);

grPlotGmdi->setWindowTitle("Горизонтальная поляризация");

grPlotGmdi->resize(400,400);

connect(grPlotGmdi,SIGNAL(destroyed()),this,SLOT(isgrPlotGmdiHide()));

if(hasGData){

// Если данные уже есть, то отображаем

Clowd bufA, bufP;

int size = Memory::get("Size",1024).toInt()\*BLOCKLANGTH;

bufA.resize(size);

bufP.resize(size);

Memory::getData("vGorizontalAr",bufA.data(),size\*sizeof(float));

Memory::getData("vGorizontalPh",bufP.data(),size\*sizeof(float));

grPlotG->setBuf(bufA,bufP);

grPlotG->plot();

}

}

grPlotGmdi->show();

}

}

void MControl::isgrPlotVmdiHide(){

isgrPlotVmdi = false;

}

void MControl::isgrPlotGmdiHide(){

isgrPlotGmdi = false;

}

void MControl::isgr3dVmdiHide(){

isgr3dVmdi = false;

}

void MControl::isgr3dGmdiHide(){

isgr3dGmdi = false;

}

void MControl::isgrRPlotVmdiHide(){

isgrRPlotVmdi = false;

}

void MControl::isgrRPlotGmdiHide(){

isgrRPlotGmdi = false;

}

void MControl::isgrOPlotVmdiHide(){

isgrOPlotVmdi = false;

}

void MControl::isgrOPlotGmdiHide(){

isgrOPlotGmdi = false;

}

void MControl::winOpen(QString winName){

if(winName=="plugin")

plugin->show();

}

/\*

\* Метод, для передачи настроек в систему сохранения

\*/

void MControl::setAction(QString name, QVariant value){

Memory::set(name,value);

settings->setValue(name,value);

settings->sync();

debug->log(name+":"+QString::number(value.toBool()));

debug->log(name+":"+QString::number(settings->value(name,false).toBool()));

}

void MControl::log(QString line){

debug->log(line);

emit setLog(line);

}

/\*

\* Прокси-метод передачи прогресс-бару

\*/

void MControl::progress(int vl){

emit setProgress(vl);

if(isgr3dVmdi)

gr3dV->updates();

if(isgr3dGmdi)

gr3dG->updates();

}

4. Memory.cpp - статический класс хранения данных. Функционирует как статический интерфейс к классу Rdata(5). Реализован в связи с большим объемом обращений к конфигурационным элементам. Сигнал-слотовая модель слишком объемна для подобного, а статический класс полностью избавляет от избыточности.

**Код:**

#include "memory.h"

Memory::Memory(){}

#ifndef MEMORY\_INIT

#define MEMORY\_INIT

QMap<QString,QVariant> Memory::info = QMap<QString,QVariant>();

QMap<QString,ScObject\*> Memory::scL = QMap<QString,ScObject\*>();

QMap<QString,Clowd> Memory::resultData = QMap<QString,Clowd>();

QMap<QString,fdata\_t\*> Memory::dataLink = QMap<QString,fdata\_t\*>();

Rdata \*Memory::dLink = new Rdata();

#endif // MEMORY\_INIT

void Memory::set(QString name, QVariant value){

dLink->setVariant(name,value);

}

QVariant Memory::get(QString name, QVariant def){

return dLink->getVariant(name,def);

}

void Memory::clearData(){

// Memory::dLink->clear();

}

void Memory::setData(QString name, void \*from, int length, unsigned int start){

Memory::dLink->set(name, from, length, start);

}

void Memory::getData(QString name, void \*to, int length, unsigned int start){

Memory::dLink->get(name, to, length, start);

}

5. rdata.cpp - потокобезопасный класс хранения данных. Потокобезопасность реализована на мьютексах. Отдельные блокировщики для больших массивов данных (хранилище сырых данных, хранилище обработанных данных и массив для вывода на графики) и для большого количества переменных простых типов (float. int, string, bool).

**Код:**

#include "rdata.h"

#include <QMutexLocker>

Rdata::Rdata()

{

qm["dGorizontalAr"] = new QMutex();

qm["dGorizontalPh"] = new QMutex();

qm["dVerticalAr"] = new QMutex();

qm["dVerticalPh"] = new QMutex();

qm["vGorizontalAr"] = new QMutex();

qm["vGorizontalPh"] = new QMutex();

qm["vVerticalAr"] = new QMutex();

qm["vVerticalPh"] = new QMutex();

qm["srcData"] = new QMutex();

qb.insert("dGorizontalAr",QByteArray());

qb.insert("dGorizontalPh",QByteArray());

qb.insert("dVerticalAr",QByteArray());

qb.insert("dVerticalPh",QByteArray());

qb.insert("vGorizontalAr",QByteArray());

qb.insert("vGorizontalPh",QByteArray());

qb.insert("vVerticalAr",QByteArray());

qb.insert("vVerticalPh",QByteArray());

qb.insert("srcData",QByteArray());

qm["Var"] = new QMutex();

}

Rdata::~Rdata(){

qm["dGorizontalAr"]->unlock();

qm["dGorizontalPh"]->unlock();

qm["dVerticalAr"]->unlock();

qm["dVerticalPh"]->unlock();

qm["vGorizontalAr"]->unlock();

qm["vGorizontalPh"]->unlock();

qm["vVerticalAr"]->unlock();

qm["vVerticalPh"]->unlock();

}

QVariant Rdata::getVariant(QString name, QVariant defval){

QMutexLocker locker(qm["Var"]);

if(qv.contains(name))

return qv[name];

qv[name] = defval;

return defval;

}

void Rdata::setVariant(QString name, QVariant val){

qm["Var"]->lock();

qv[name] = val;

list[name] = val;

qm["Var"]->unlock();

}

void Rdata::set(QString name, void \*from, int length, unsigned int start){

char \*r;

char \*lsk = (char \*)from;

r = lsk+start;

qm[name]->lock();

if (qb[name].length() < length){

qb[name].clear();

qb[name].resize(length);

}

memcpy(qb[name].data(),r,length);

qm[name]->unlock();

}

void Rdata::get(QString name, void \*to, int length, unsigned int start){

void \*r;

char \*s = (char \*)to;

qm[name]->lock();

if(qb[name].length()==0){

qm[name]->unlock();

return;

}

r = qb[name].data()+start;

int position = length;

int subP = 0;

unsigned int lengthQb = qb[name].length();

if(lengthQb < (unsigned int)length+start){

position = qb[name].length() - start;

subP = length-position;

}

memcpy(to,r,position);

if(subP > 0)

memset(s+position,0,subP);

qm[name]->unlock();

}

6. worker.cpp - класс, производящий основные вычисления. Работает в отдельно потоке. Реализует основную математику программы. Создает в отдельных потоках сетевые воркеры (7 и 8).

**Код:**

7. tcpwork.cpp - класс - tcp клиент. Отправляет команды управления на РЛС и получает данные с него. Выделяет отдельные пачки сигналов и передает поблочно в Worker(6).

8. udpsock.cpp - класс - udp клиент. Отправляет команды управления на РЛС (по старому протоколу) и получает данные с него. Выделяет отдельные пачки сигналов и передает поблочно в Worker::Process(6).

**Код:**

#include "udpsock.h"

#include <QDataStream>

#include <QDateTime>

#include <QFile>

#include "memory.h"

#include <arpa/inet.h>

#include <netinet/ether.h>

#include <netinet/ip.h>

#include <netinet/tcp.h>

#include <netinet/udp.h>

#include <iostream>

using namespace std;

static const int PCAP\_TIMEOUT = 10000;

UDPSock::UDPSock(QObject \*parent) : QObject(parent){}

UDPSock::~*UDPSock*(){

close();

}

void UDPSock::*run*(){

fname = "";

position = 0;

updateInterface();

}

bool UDPSock::openConnection( const QString &dev, int snaplen, bool promisc )

{

handle = pcap\_open\_live( dev.toLocal8Bit().constData(),

snaplen,

(int)promisc,

1000,

errbuf );

return isValid();

}

bool UDPSock::isValid(){

//cout << "isValid: " << handle << endl;

return (0 != handle);

}

bool UDPSock::setFilter( const QString &filterexp )

{

int status = pcap\_compile(handle, &filter, filterexp.toLocal8Bit().constData(), 0, 0);

if (status != 0)

return false;

cout << "pcap\_compile: " << filterexp.toStdString() << ": " << status << endl;

status = pcap\_setfilter(handle, &filter);

cout << "pcap\_setfilter: " << status << endl;

if (status != 0)

return false;

return true;

}

bool UDPSock::close()

{

if (!isValid())

return false;

pcap\_close(handle);

handle = 0;

return true;

}

void UDPSock::listen(){

pcap\_pkthdr \*pktheader;

const u\_char \*pktdata;

int res = 0;

if(isValid()){

while(isValid()){

res = pcap\_next\_ex(handle, &pktheader, &pktdata);

if(res == 1){

//cout << pktheader->len << endl;

packReady(pktdata,pktheader->len);

}

}

}

else {

cout << "stop" << endl;

}

}

void UDPSock::call(uchar \*self, const pcap\_pkthdr \*header, const uchar \*packet){

Q\_UNUSED(self);

Q\_UNUSED(packet);

//UDPSock \*usock = (UDPSock \*)self;

cout << header->len << endl;

//usock->packReady(packet);

}

void UDPSock::updateInterface(){

QString devNew = Memory::get("device","").toString();

int rlsPortNew = Memory::get("rlsPort",30583).toInt();

QString rlsHostNew = Memory::get("rlsIP","127.0.0.1").toString();

bool re = false;

if(devNew!=dev){

dev = devNew;

re = true;

}

if(rlsPortNew!=rlsPort){

rlsPort = rlsPortNew;

re = true;

}

if(rlsHostNew!=rlsHost){

rlsHost = rlsHostNew;

re = true;

}

if(!re)

return;

if(isValid()){

close();

QTimer::singleShot(1500,[=](){

this->updateInterface();

});

return;

}

QString filter ="ip src host "+rlsHost+" and udp dst port "+QString::number(rlsPort);

if(dev.size() > 0){

cout << filter.toStdString() << endl;

if(openConnection( dev, 65535, true )){

setFilter( filter );

// setFilter( QString("udp") );

listen();

}

cout << isValid() << endl;

}

}

void UDPSock::packReady(const uchar \*packet, int len){

int pos = len - 1046;

const uchar \*payload = packet + pos;

const uchar \*DataPtr = payload;

QByteArray ba((const char \*)DataPtr,1046);

emit Process(ba);

}

9. formSettings.cpp - класс - окно с настройками для РЛМ.

**Код:**

#include "formsettings.h"

#include "ui\_formsettings.h"

#include "memory.h"

#include <QDesktopWidget>

formSettings::formSettings(QWidget \*parent) :

QDialog(parent),

ui(new Ui::formSettings)

{

ui->setupUi(this);

this->setWindowFlags(Qt::WindowStaysOnTopHint);

QDesktopWidget \*desktop = QApplication::desktop();

int screenWidth = desktop->width();

int screenHeight = desktop->height();

int width = this->geometry().width();

int height = this->geometry().height();

this->move((screenWidth-width)/2,(screenHeight-height)/2);

}

formSettings::~*formSettings*()

{

delete ui;

}

void formSettings::on\_leSubBufNum\_valueChanged(int arg1)

{

Memory::set("leSubBufNum",arg1);

emit leSubBufNum(arg1);

emit save();

}

void formSettings::on\_leBurstLen\_valueChanged(int arg1)

{

Memory::set("leBurstLen",arg1);

emit leBurstLen(arg1);

emit save();

}

void formSettings::on\_leGeterodin\_valueChanged(int arg1)

{

Memory::set("leGeterodin",arg1);

emit leGeterodin(arg1);

emit save();

}

void formSettings::on\_leFreq\_valueChanged(int arg1)

{

Memory::set("leFreq",arg1);

emit leFreq(arg1);

emit save();

}

void formSettings::on\_leAmp\_valueChanged(int arg1)

{

Memory::set("leAmp",arg1);

emit leAmp(arg1);

emit save();

}

void formSettings::on\_cbPulseMod\_toggled(bool checked)

{

Memory::set("cbPulseMod",checked);

emit cbPulseMod(checked);

emit save();

}

void formSettings::on\_cbUWB\_toggled(bool checked)

{

Memory::set("cbUWB",checked);

emit cbUWB(checked);

emit save();

}

void formSettings::on\_cbLFM\_toggled(bool checked)

{

Memory::set("cbLFM",checked);

emit cbLFM(checked);

emit save();

}

void formSettings::on\_lePeriod\_valueChanged(double arg1)

{

Memory::set("lePeriod",arg1);

emit lePeriod(arg1);

emit save();

}

void formSettings::on\_leDuration\_valueChanged(double arg1)

{

Memory::set("leDuration",arg1);

emit leDuration(arg1);

emit save();

}

void formSettings::on\_leFreqRange\_valueChanged(int arg1)

{

Memory::set("leFreqRange",arg1);

emit leFreqRange(arg1);

emit save();

}

void formSettings::on\_cbGate\_toggled(bool checked)

{

Memory::set("cbGate",checked);

emit cbGate(checked);

emit save();

}

void formSettings::on\_leGateDelay\_valueChanged(int arg1)

{

Memory::set("leGateDelay",arg1);

emit leGateDelay(arg1);

emit save();

}

void formSettings::on\_leGateDuration\_valueChanged(double arg1)

{

Memory::set("leGateDuration",arg1);

emit leGateDuration(arg1);

emit save();

}

void formSettings::on\_cbCont\_toggled(bool checked)

{

Memory::set("cbCont",checked);

emit cbCont(checked);

emit save();

}

void formSettings::on\_cbCoherentAccum\_toggled(bool checked)

{

Memory::set("cbCoherentAccum",checked);

emit cbCoherentAccum(checked);

emit save();

}

void formSettings::on\_cbDDSReset\_toggled(bool checked)

{

Memory::set("cbDDSReset",checked);

emit cbDDSReset(checked);

emit save();

}

void formSettings::on\_cbLOGM\_toggled(bool checked)

{

Memory::set("cbLOGM",checked);

emit cbLOGM(checked);

emit save();

}

void formSettings::on\_rbDdsRstBurst\_toggled(bool checked)

{

Memory::set("rbDdsRstBurst",checked);

emit rbDdsRstBurst(checked);

emit save();

}

void formSettings::on\_rbDdsRstPulse\_toggled(bool checked)

{

Memory::set("rbDdsRstPulse",checked);

emit rbDdsRstPulse(checked);

emit save();

}

void formSettings::setleSubBufNum(int value){

ui->leSubBufNum->setValue(value);

}

void formSettings::setleBurstLen(int value){

ui->leBurstLen->setValue(value);

}

void formSettings::setleGeterodin(int val){

ui->leGeterodin->setValue(val);

}

void formSettings::setleFreq(int val){

ui->leFreq->setValue(val);

}

void formSettings::setleAmp(int val){

ui->leAmp->setValue(val);

}

void formSettings::setcbPulseMod(bool val){

ui->cbPulseMod->setChecked(val);

}

void formSettings::setcbUWB(bool val){

ui->cbUWB->setChecked(val);

}

void formSettings::setcbLFM(bool val){

ui->cbLFM->setChecked(val);

}

void formSettings::setlePeriod(double val){

ui->lePeriod->setValue(val);

}

void formSettings::setleDuration(double val){

ui->leDuration->setValue(val);

}

void formSettings::setleFreqRange(int val){

ui->leFreqRange->setValue(val);

}

void formSettings::setcbGate(bool val){

ui->cbGate->setChecked(val);

}

void formSettings::setleGateDelay(int val){

ui->leGateDelay->setValue(val);

}

void formSettings::setleGateDuration(double val){

ui->leGateDuration->setValue(val);

}

void formSettings::setcbCont(bool val){

ui->cbCont->setChecked(val);

}

void formSettings::setcbCoherentAccum(bool val){

ui->cbCoherentAccum->setChecked(val);

}

void formSettings::setcbDDSReset(bool val){

ui->cbDDSReset->setChecked(val);

}

void formSettings::setcbLOGM(bool val){

ui->cbLOGM->setChecked(val);

}

void formSettings::setrbDdsRstBurst(bool val){

ui->rbDdsRstBurst->setChecked(val);

}

void formSettings::setrbDdsRstPulse(bool val){

ui->rbDdsRstPulse->setChecked(val);

}

void formSettings::setrbTxPolXX(bool val){

ui->rbTxPolXX->setChecked(val);

}

void formSettings::setrbTxPolXY(bool val){

ui->rbTxPolXY->setChecked(val);

}

void formSettings::setrbTxPolYX(bool val){

ui->rbTxPolYX->setChecked(val);

}

void formSettings::setrbTxPolYY(bool val){

ui->rbTxPolYY->setChecked(val);

}

void formSettings::setleTxAtt(int val){

ui->leTxAtt->setValue(val);

}

void formSettings::on\_rbTxPolXX\_toggled(bool checked)

{

Memory::set("rbTxPolXX",checked);

emit save();

}

void formSettings::on\_rbTxPolXY\_toggled(bool checked)

{

Memory::set("rbTxPolXY",checked);

emit save();

}

void formSettings::on\_rbTxPolYX\_toggled(bool checked)

{

Memory::set("rbTxPolYX",checked);

emit save();

}

void formSettings::on\_rbTxPolYY\_toggled(bool checked)

{

Memory::set("rbTxPolYY",checked);

emit save();

}

void formSettings::on\_leTxAtt\_valueChanged(int arg1)

{

Memory::set("leTxAtt",arg1);

emit save();

}

void formSettings::on\_BitBtn1\_clicked()

{

emit sendParam();

QByteArray ba;

ba.resize(sizeof(int));

unsigned int leSubBufNum = (unsigned int)Memory::get("leSubBufNum",1).toInt();

memcpy(ba.data(),&leSubBufNum,sizeof(int));

QTimer::singleShot(1000,[=](){

emit sendMsg(4, (unsigned char \*)ba.data(), 0x0002);

});

}

void formSettings::setrlsIP(QString val){

ui->rlsIP->setText(val);

}

void formSettings::on\_rlsIP\_textEdited(const QString &arg1)

{

Memory::set("rlsIP",arg1);

emit save();

}

void formSettings::on\_rlsIP\_textChanged(const QString &arg1)

{

Memory::set("rlsIP",arg1);

emit save();

}

void formSettings::setrbRxAnt0(bool checked){

ui->rbRxAnt0->setChecked(checked);

}

void formSettings::setrbRxAnt1(bool checked){

ui->rbRxAnt1->setChecked(checked);

}

void formSettings::on\_rbRxAnt0\_toggled(bool checked)

{

Memory::set("rbRxAnt0",checked);

emit save();

}

void formSettings::on\_rbRxAnt1\_toggled(bool checked)

{

Memory::set("rbRxAnt1",checked);

emit save();

}

void formSettings::setcbMGEN(bool checked){

ui->cbMGEN->setChecked(checked);

}

void formSettings::on\_cbMGEN\_toggled(bool checked)

{

Memory::set("cbMGEN",checked);

emit save();

}

void formSettings::setseMLEN(int arg1){

ui->seMLEN->setValue(arg1);

}

void formSettings::on\_seMLEN\_valueChanged(int arg1)

{

Memory::set("seMLEN",arg1);

emit save();

}

void formSettings::setrlsPort(int port){

ui->rlsPort->setValue(port);

}

void formSettings::on\_rlsPort\_valueChanged(int arg1)

{

Memory::set("rlsPort",arg1);

emit save();

}

void formSettings::on\_leRxAtt\_valueChanged(int arg1)

{

Memory::set("leRxAtt",arg1);

emit save();

}

void formSettings::setleRxAtt(int val){

ui->leRxAtt->setValue(val);

}

void formSettings::on\_rbRxPolXX\_toggled(bool checked)

{

Memory::set("rbRxPolXX",checked);

emit save();

}

void formSettings::on\_rbRxPolXY\_toggled(bool checked)

{

Memory::set("rbRxPolXY",checked);

emit save();

}

void formSettings::on\_rbRxPolYX\_toggled(bool checked)

{

Memory::set("rbRxPolYX",checked);

emit save();

}

void formSettings::on\_rbRxPolYY\_toggled(bool checked)

{

Memory::set("rbRxPolYY",checked);

emit save();

}

void formSettings::setrbRxPolXX(bool checked){

ui->rbRxPolXX->setChecked(checked);

}

void formSettings::setrbRxPolXY(bool checked){

ui->rbRxPolXY->setChecked(checked);

}

void formSettings::setrbRxPolYX(bool checked){

ui->rbRxPolYX->setChecked(checked);

}

void formSettings::setrbRxPolYY(bool checked){

ui->rbRxPolYY->setChecked(checked);

}

10. gr3d.cpp - класс - окно для 3d графика.

Код:

#include "gr3d.h"

#include "ui\_gr3d.h"

#include "memory.h"

Gr3D::Gr3D(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::Gr3D)

{

ui->setupUi(this);

pgl = new PlotGl();

connect(pgl,&PlotGl::xRotationChanged,this,&Gr3D::xRotationChanged);

connect(pgl,&PlotGl::yRotationChanged,this,&Gr3D::yRotationChanged);

connect(pgl,&PlotGl::zRotationChanged,this,&Gr3D::zRotationChanged);

}

void Gr3D::setType(QString typeName){

pgl->setType(typeName);

tName = typeName;

ui->controlBlock->addWidget(pgl);

ui->SlideScale->setValue(Memory::get(tName+"Scale",0).toInt());

ui->SlideX->setValue(Memory::get(tName+"SLideX",0).toInt());

ui->SlideY->setValue(Memory::get(tName+"SLideY",0).toInt());

ui->SlideZ->setValue(Memory::get(tName+"SLideZ",0).toInt());

}

Gr3D::~*Gr3D*()

{

delete ui;

}

void Gr3D::updates(){

pgl->updateSc();

}

void Gr3D::on\_SlideScale\_valueChanged(int value)

{

pgl->setScale(ui->SlideScale->maximum()-value);

Memory::set(tName+"Scale",value);

emit sync();

}

void Gr3D::on\_SlideZ\_valueChanged(int value)

{

pgl->setZRotation(value);

Memory::set(tName+"SLideZ",value);

emit sync();

}

void Gr3D::on\_SlideX\_valueChanged(int value)

{

pgl->setXRotation(value);

Memory::set(tName+"SLideX",value);

emit sync();

}

void Gr3D::on\_SlideY\_valueChanged(int value)

{

pgl->setYRotation(value);

Memory::set(tName+"SLideY",value);

emit sync();

}

void Gr3D::xRotationChanged(int val){

ui->SlideX->setValue(val);

Memory::set(tName+"SLideX",val);

emit sync();

}

void Gr3D::yRotationChanged(int val){

ui->SlideY->setValue(val);

Memory::set(tName+"SLideY",val);

emit sync();

}

void Gr3D::zRotationChanged(int val){

ui->SlideZ->setValue(val);

Memory::set(tName+"SLideZ",val);

emit sync();

}

11. scobject.cpp - класс для хранения данных для передачи в виджет OpenGL графика. Инициализирует дополнительный класс ScWorker(13) в отдельном потоке.

Код:

#include "scobject.h"

#include "memory.h"

ScObject::ScObject(QObject \*parent)

:QObject(parent),

m\_count(0),

maxColor(0)

{

clearData();

loadDataImage();

Size = 0;

AngleMin = 0;

AngleMax = 0;

OffsetMin = 0;

OffsetMax = 0;

delY = 1.0f;

plotWork = false;

rePlot = false;

isShared = false;

sharedTimer = new QTimer;

sharedTimer->setInterval(40);

connect(sharedTimer,&QTimer::timeout,this,&ScObject::sharedNow);

worker = new ScWorker();

worker->moveToThread(&mathThread);

connect(&mathThread, &QThread::finished, worker, &QObject::deleteLater);

connect(this,&ScObject::syncSignal,worker,&ScWorker::syncSlot);

connect(this,&ScObject::dataSignal,worker,&ScWorker::dataSlot);

connect(this,&ScObject::sharedSignal,worker,&ScWorker::sharedSlot);

connect(this,&ScObject::plotSignal,worker,&ScWorker::plotSlot);

connect(worker,&ScWorker::plot,this,&ScObject::result);

mathThread.start();

}

ScObject::~*ScObject*(){

mathThread.quit();

mathThread.wait();

}

void ScObject::sharedNow(){

if(isShared){

isShared = false;

}

else return;

emit sharedSignal(sharedShip,tName);

}

void ScObject::shared(int shp){

isShared = true;

sharedShip = shp;

if(!sharedTimer->isActive()){

sharedTimer->start();

}

}

void ScObject::setBuf(Clowd &dataA,Clowd &dataH){

if(sharedTimer->isActive())

sharedTimer->stop();

histA = dataA;

histY = dataH;

emit dataSignal(histA,histY);

//emit plotSignal();

}

void ScObject::result(Clowd &data){

m\_data = data;

m\_count = m\_data.size();

emit plot();

}

void ScObject::sync(){

unsigned int nSize = (int)Memory::get("Size",0).toInt();

if(nSize != Size){

Size = nSize;

histA.clear();

histA.resize(Size\*BLOCKLANGTH);

histY.clear();

histY.resize(Size\*BLOCKLANGTH);

}

AngleMin = 180-Memory::get("PhMin",0).toInt();

AngleMax = Memory::get("PhMin",0).toInt()+180;

OffsetMin = Memory::get("ArgMin",0).toInt();

OffsetMax = Memory::get("ArgMax",1024).toInt();

maxColor = Memory::get("colorValue",0).toInt();

MaxBarier = Memory::get("Barier",100).toInt();

delY = (GLfloat)32.0f/(181-Memory::get("PhMin",0).toInt());

emit syncSignal();

reLine();

emit plotSignal();

}

void ScObject::setType(QString typeName){

tName = typeName;

}

void ScObject::setAngle(int min,int max){

AngleMin = min+180;

AngleMax = max+180;

delY = (GLfloat)32.0f/(181-max);

reLine();

}

void ScObject::setOffset(unsigned int min, unsigned int max){

OffsetMin = min;

OffsetMax = max;

reLine();

}

void ScObject::setSizeBlock(unsigned int size){

Size = size;

reLine();

}

void ScObject::reLine(){

if(Size == 0) return;

l\_count = 12\*2\*6;

l\_data.resize(24\*6);

sObjectLink = l\_data.data();

centerX = (GLfloat)Size/2;

centerY = AngleMin + (GLfloat) ( AngleMax - AngleMin ) /2;

centerZ = OffsetMin + (GLfloat) ( OffsetMax - OffsetMin ) /2;

//top

appendPoint(-1,AngleMin-1,OffsetMin-1);

appendPoint(-1,AngleMin-1,OffsetMax+1);

appendPoint(-1,AngleMin-1,OffsetMin-1);

appendPoint(-1,AngleMax+1,OffsetMin-1);

appendPoint(-1,AngleMax+1,OffsetMax+1);

appendPoint(-1,AngleMax+1,OffsetMin-1);

appendPoint(-1,AngleMax+1,OffsetMax+1);

appendPoint(-1,AngleMin-1,OffsetMax+1);

appendPoint(Size+1,AngleMin-1,OffsetMin-1);

appendPoint(Size+1,AngleMin-1,OffsetMax+1);

appendPoint(Size+1,AngleMin-1,OffsetMin-1);

appendPoint(Size+1,AngleMax+1,OffsetMin-1);

appendPoint(Size+1,AngleMax+1,OffsetMax+1);

appendPoint(Size+1,AngleMax+1,OffsetMin-1);

appendPoint(Size+1,AngleMax+1,OffsetMax+1);

appendPoint(Size+1,AngleMin-1,OffsetMax+1);

appendPoint(-1,AngleMax+1,OffsetMax+1);

appendPoint(Size+1,AngleMax+1,OffsetMax+1);

appendPoint(-1,AngleMin-1,OffsetMax+1);

appendPoint(Size+1,AngleMin-1,OffsetMax+1);

appendPoint(-1,AngleMax+1,OffsetMin-1);

appendPoint(Size+1,AngleMax+1,OffsetMin-1);

appendPoint(-1,AngleMin-1,OffsetMin-1);

appendPoint(Size+1,AngleMin-1,OffsetMin-1);

}

void ScObject::appendPoint(int x,int y,int z){

\*(sObjectLink++) = (centerX-x)/MAXBYTEFLOAT;

\*(sObjectLink++) = (centerY-y)/(MAXBYTEFLOAT\*delY);

\*(sObjectLink++) = -(centerZ-z)/MAXBYTEFLOAT;

\*(sObjectLink++) = 0.0f;

\*(sObjectLink++) = 1.0f;

\*(sObjectLink++) = 0.0f;

}

void ScObject::setMax(int color){

maxColor = color;

m\_data.clear();

m\_count = 0;

loadDataImage();

}

void ScObject::loadDataImage(){

/\*m\_data = c\_data;

int size = m\_data.size();

m\_count = size;\*/

}

void ScObject::editData(const Clowd &result){

c\_data = result;

loadDataImage();

}

void ScObject::clearData(){

c\_data.clear();

}

void ScObject::appendPointObj(int x,int y,int z, int color){

\*(m\_dataLink++) = (centerX-x)/MAXBYTEFLOAT;

\*(m\_dataLink++) = (centerY-y)/(MAXBYTEFLOAT\*delY);

\*(m\_dataLink++) = -(centerZ-z)/MAXBYTEFLOAT;

if(color == 0) {

\*(m\_dataLink++) = 0;

\*(m\_dataLink++) = 0;

\*(m\_dataLink++) = 0;

}

else {

\*(m\_dataLink++) = (float)color/MAXBYTEFLOAT;

\*(m\_dataLink++) = 0.0f;

\*(m\_dataLink++) = (float)(MAXBYTEFLOAT - color)/MAXBYTEFLOAT;

}

}

void ScObject::plotFlower(int x,int y,int z, unsigned int color){

double NormalColor = (double)(QUINT16\_SIZE);

int nColor = (int)MAXBYTE\*((double)color)/NormalColor;

if(nColor > MAXBYTE)

nColor = MAXBYTE;

//appendPoint(x,y,z,nColor);

appendPointObj(x-1,y,z-1,nColor);

appendPointObj(x+1,y,z-1,nColor);

appendPointObj(x,y+1,z,nColor);

appendPointObj(x-1,y,z+1,nColor);

appendPointObj(x+1,y,z+1,nColor);

appendPointObj(x,y+1,z,nColor);

appendPointObj(x+1,y,z-1,nColor);

appendPointObj(x+1,y,z+1,nColor);

appendPointObj(x,y+1,z,nColor);

appendPointObj(x-1,y,z-1,nColor);

appendPointObj(x-1,y,z+1,nColor);

appendPointObj(x,y+1,z,nColor);

appendPointObj(x-1,y,z-1,nColor);

appendPointObj(x+1,y,z-1,nColor);

appendPointObj(x,y-1,z,nColor);

appendPointObj(x-1,y,z+1,nColor);

appendPointObj(x+1,y,z+1,nColor);

appendPointObj(x,y-1,z,nColor);

appendPointObj(x+1,y,z-1,nColor);

appendPointObj(x+1,y,z+1,nColor);

appendPointObj(x,y-1,z,nColor);

appendPointObj(x-1,y,z-1,nColor);

appendPointObj(x-1,y,z+1,nColor);

appendPointObj(x,y-1,z,nColor);

}

12. plotgl.cpp - класс - OpenGL виджет. Реализует вывод на экрана 3d графика.

Код:

#include "plotgl.h"

#include <QMouseEvent>

#include <QCoreApplication>

#include <QPainter>

#include <math.h>

#include <QOpenGLPaintDevice>

#include <QGraphicsScene>

#include <QGraphicsView>

#include "memory.h"

PlotGl::PlotGl(QWidget \*parent)

: QOpenGLWidget(parent),

m\_xRot(0),

m\_yRot(0),

m\_zRot(0),

m\_program(0)

{

m\_scale = 2.0;

shiperTimer = new QTimer;

shiperTimer->setInterval(40);

connect(shiperTimer,&QTimer::timeout,this,&PlotGl::timeout);

isUpd = false;

}

void PlotGl::timeout(){

if(isUpd)

isUpd = false;

else return;

updateSc();

}

void PlotGl::updateTimer(){

isUpd = true;

if(!shiperTimer->isActive())

shiperTimer->start();

}

void PlotGl::stopTimer(){

shiperTimer->stop();

}

void PlotGl::setType(QString typeName){

if(Memory::scL.contains(typeName))

m\_Sc = Memory::scL[typeName];

else {

m\_Sc = new ScObject;

Memory::scL[typeName] = m\_Sc;

}

m\_Sc->setType(typeName);

m\_Sc->c\_data = Memory::resultData[typeName];

m\_Sc->loadDataImage();

m\_Sc->sync();

connect(m\_Sc,&ScObject::plot,this,&PlotGl::updateSc);

}

void PlotGl::setScale(int scale){

m\_scale = (float)scale/50;

update();

}

PlotGl::~*PlotGl*()

{

cleanup();

delete m\_Sc;

}

QSize PlotGl::*minimumSizeHint*() const

{

return QSize(50, 50);

}

QSize PlotGl::*sizeHint*() const

{

return QSize(400, 400);

}

static void qNormalizeAngle(int &angle)

{

while (angle < 0)

angle += 360 \* 16;

while (angle > 360 \* 16)

angle -= 360 \* 16;

}

void PlotGl::setXRotation(int angle)

{

qNormalizeAngle(angle);

if (angle != m\_xRot) {

m\_xRot = angle;

emit xRotationChanged(angle);

update();

}

}

void PlotGl::setYRotation(int angle)

{

qNormalizeAngle(angle);

if (angle != m\_yRot) {

m\_yRot = angle;

emit yRotationChanged(angle);

update();

}

}

void PlotGl::setZRotation(int angle)

{

qNormalizeAngle(angle);

if (angle != m\_zRot) {

m\_zRot = angle;

emit zRotationChanged(angle);

update();

}

}

void PlotGl::cleanup()

{

makeCurrent();

m\_ScVbo.destroy();

delete m\_program;

m\_program = 0;

doneCurrent();

}

static const char \*vertexShaderSource =

"attribute vec4 vertex;\n"

//"attribute vec3 normal;\n"

"attribute vec3 aVertexColor;\n"

"varying vec3 vert;\n"

"varying vec3 vertNormal;\n"

"uniform mat4 projMatrix;\n"

"uniform mat4 mvMatrix;\n"

"uniform mat3 normalMatrix;\n"

"varying highp vec4 vColor;\n"

"varying highp vec4 vPosition;\n"

"void main() {\n"

" vert = vertex.xyz;\n"

" vertNormal = normalMatrix \* vertex.xyz;\n"

" gl\_PointSize = 3.0;\n"

" vPosition = projMatrix \* mvMatrix \* vertex;\n"

" gl\_Position = vPosition;\n"

" vColor = vec4(aVertexColor.x,aVertexColor.y,aVertexColor.z,0.0);\n"

"}\n";

static const char \*fragmentShaderSource =

"varying highp vec3 vert;\n"

"varying highp vec3 vertNormal;\n"

"uniform highp vec3 lightPos;\n"

"varying highp vec4 vColor;\n"

"varying highp vec4 vPosition;\n"

"void main() {\n"

" highp vec3 L = normalize(lightPos - vert);\n"

" highp float NL = max(dot(normalize(vertNormal), L), 0.0);\n"

" highp vec4 col = clamp(vColor \* 0.85 + vColor \* 0.95 \* NL, 0.0, 1.0);\n"

" gl\_FragColor = col;\n"

"}\n";

QByteArray versionedShaderCode(const char \*src)

{

QByteArray versionedSrc;

versionedSrc.append(src);

return versionedSrc;

}

void PlotGl::*initializeGL*()

{

connect(context(), &QOpenGLContext::aboutToBeDestroyed, this, &PlotGl::cleanup);

initializeOpenGLFunctions();

glClearColor(0.1, 0.1, 0.1, 1);

m\_program = new QOpenGLShaderProgram;

m\_program->addShaderFromSourceCode(QOpenGLShader::Vertex, versionedShaderCode(vertexShaderSource));

m\_program->addShaderFromSourceCode(QOpenGLShader::Fragment, versionedShaderCode(fragmentShaderSource));

m\_program->bindAttributeLocation("vertex", 0);

m\_program->bindAttributeLocation("aVertexColor", 1);

m\_program->*link*();

m\_program->bind();

m\_projMatrixLoc = m\_program->uniformLocation("projMatrix");

m\_mvMatrixLoc = m\_program->uniformLocation("mvMatrix");

m\_normalMatrixLoc = m\_program->uniformLocation("normalMatrix");

m\_fragColorLoc = m\_program->uniformLocation("fragColor");

m\_lightPosLoc = m\_program->uniformLocation("lightPos");

m\_vao.create();

QOpenGLVertexArrayObject::Binder vaoBinder(&m\_vao);

m\_ScVbo.create();

m\_ScVbo.bind();

m\_ScVbo.allocate(m\_Sc->constData(), m\_Sc->count() \* sizeof(GLfloat));

vaoBinder.rebind();

setupVertexAttribs();

m\_vaoLast.create();

QOpenGLVertexArrayObject::Binder vaoBinderLast(&m\_vaoLast);

m\_ScVboLast.create();

m\_ScVboLast.bind();

m\_ScVboLast.allocate(m\_Sc->constDataLines(), m\_Sc->countLines() \* sizeof(GLfloat));

vaoBinderLast.rebind();

setupVertexAttribsLast();

m\_camera.setToIdentity();

m\_camera.translate(0, 0, -0.2);

m\_program->setUniformValue(m\_lightPosLoc, QVector3D(0, 0, 2));

m\_program->release();

}

void PlotGl::setupVertexAttribs()

{

m\_ScVbo.bind();

QOpenGLFunctions \*f = QOpenGLContext::currentContext()->functions();

f->glEnableVertexAttribArray(0);

f->glEnableVertexAttribArray(1);

f->glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 6 \* sizeof(GLfloat), 0);

f->glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 6 \* sizeof(GLfloat), reinterpret\_cast<void \*>(3 \* sizeof(GLfloat)));

m\_ScVbo.release();

}

void PlotGl::setupVertexAttribsLast()

{

m\_ScVboLast.bind();

QOpenGLFunctions \*f = QOpenGLContext::currentContext()->functions();

f->glEnableVertexAttribArray(0);

f->glEnableVertexAttribArray(1);

f->glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 6 \* sizeof(GLfloat), 0);

f->glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 6 \* sizeof(GLfloat), reinterpret\_cast<void \*>(3 \* sizeof(GLfloat)));

m\_ScVboLast.release();

}

void PlotGl::updateSc(){

m\_ScVbo.release();

m\_ScVbo.bind();

m\_ScVbo.allocate(m\_Sc->constData(), m\_Sc->count() \* sizeof(GLfloat));

setupVertexAttribs();

m\_ScVboLast.release();

m\_ScVboLast.bind();

m\_ScVboLast.allocate(m\_Sc->constDataLines(), m\_Sc->countLines() \* sizeof(GLfloat));

setupVertexAttribs();

m\_program->release();

update();

}

void PlotGl::*paintGL*()

{

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

glEnable(GL\_BLEND);

glEnable(GL\_DEPTH\_TEST);

m\_camera.setToIdentity();

m\_camera.translate(0, 0, - m\_scale);

m\_world.setToIdentity();

m\_world.rotate(180.0f - (m\_xRot / 16.0f), 1, 0, 0);

m\_world.rotate(m\_yRot / 16.0f, 0, 1, 0);

m\_world.rotate(m\_zRot / 16.0f, 0, 0, 1);

m\_program->bind();

m\_program->setUniformValue(m\_projMatrixLoc, m\_proj);

m\_program->setUniformValue(m\_mvMatrixLoc, m\_camera \* m\_world);

QMatrix3x3 normalMatrix = m\_world.normalMatrix();

m\_program->setUniformValue(m\_normalMatrixLoc, normalMatrix);

QOpenGLVertexArrayObject::Binder vaoBinder(&m\_vao);

//glDrawArrays(GL\_POINTS, 0, m\_Sc->vertexCount());

glDrawArrays(GL\_TRIANGLES, 0, m\_Sc->vertexCount());

vaoBinder.release();

QOpenGLVertexArrayObject::Binder vaoBinderLast(&m\_vaoLast);

// glDrawArrays(GL\_POINTS, 0, m\_Sc->vertexCount());

glDrawArrays(GL\_LINES, 0, m\_Sc->vertexCountLines());

//vaoBinderLast.release();

vaoBinderLast.release();

m\_program->release();

}

void PlotGl::initData(){

}

void PlotGl::*resizeGL*(int w, int h)

{

m\_proj.setToIdentity();

// m\_proj.perspective(100.0f, GLfloat(w) / h, 0.01f, 100.0f);

m\_proj.perspective(45.0f, GLfloat(w) / h, 0.01f, 100.0f);

}

void PlotGl::*mousePressEvent*(QMouseEvent \*event)

{

m\_lastPos = event->pos();

}

void PlotGl::*mouseMoveEvent*(QMouseEvent \*event)

{

int dx = event->x() - m\_lastPos.x();

int dy = event->y() - m\_lastPos.y();

if (event->buttons() & Qt::LeftButton) {

setXRotation(m\_xRot + 8 \* dy);

setYRotation(m\_yRot + 8 \* dx);

} else if (event->buttons() & Qt::RightButton) {

setXRotation(m\_xRot + 8 \* dy);

setZRotation(m\_zRot + 8 \* dx);

}

m\_lastPos = event->pos();

}

13. scworker.cpp - класс, преобразующий данные из Rdata(5), а именно массива для вывода на графики, в примитивы и объекты, родные OpenGL виджету. Для улучшение отклика программы реализован в отдельном потоке.

Код:

#include "scworker.h"

#include "memory.h"

#include "math.h"

ScWorker::ScWorker(QObject \*parent) : QObject(parent)

{

syncSlot();

GLfloat R=0.75; // радиус сферы

// начальные значения для икосаэдра

GLfloat a=4\*R/sqrt(10+2\*sqrt(5)); // сторона икосаэдра

GLfloat alpha=acos((1-a\*a/2/R/R)); // первый угол поворота по тэта

float pi = 3.14159265358979323846;

float k = pi/180;

// cos(alpha)=(1-a\*a/2/R/R)

// вычисляем точки икосаэдра

//0 точка

VertexArray[0][0]=0; // x

VertexArray[0][1]=0; // y

VertexArray[0][2]=R; // z

//1 точка

VertexArray[1][0]=R\*sin(alpha)\*sin(0);

VertexArray[1][1]=R\*sin(alpha)\*cos(0);

VertexArray[1][2]=R\*cos(alpha);

//2 точка

VertexArray[2][0]=R\*sin(alpha)\*sin(72\*k);

VertexArray[2][1]=R\*sin(alpha)\*cos(72\*k);

VertexArray[2][2]=R\*cos(alpha);

//3 точка

VertexArray[3][0]=R\*sin(alpha)\*sin(2\*72\*k);

VertexArray[3][1]=R\*sin(alpha)\*cos(2\*72\*k);

VertexArray[3][2]=R\*cos(alpha);

//4 точка

VertexArray[4][0]=R\*sin(alpha)\*sin(3\*72\*k);

VertexArray[4][1]=R\*sin(alpha)\*cos(3\*72\*k);

VertexArray[4][2]=R\*cos(alpha);

//5 точка

VertexArray[5][0]=R\*sin(alpha)\*sin(4\*72\*k);

VertexArray[5][1]=R\*sin(alpha)\*cos(4\*72\*k);

VertexArray[5][2]=R\*cos(alpha);

//6 точка

VertexArray[6][0]=R\*sin(pi-alpha)\*sin(-36\*k);

VertexArray[6][1]=R\*sin(pi-alpha)\*cos(-36\*k);

VertexArray[6][2]=R\*cos(pi-alpha);

//7 точка

VertexArray[7][0]=R\*sin(pi-alpha)\*sin(36\*k);

VertexArray[7][1]=R\*sin(pi-alpha)\*cos(36\*k);

VertexArray[7][2]=R\*cos(pi-alpha);

//8 точка

VertexArray[8][0]=R\*sin(pi-alpha)\*sin((36+72)\*k);

VertexArray[8][1]=R\*sin(pi-alpha)\*cos((36+72)\*k);

VertexArray[8][2]=R\*cos(pi-alpha);

//9 точка

VertexArray[9][0]=R\*sin(pi-alpha)\*sin((36+2\*72)\*k);

VertexArray[9][1]=R\*sin(pi-alpha)\*cos((36+2\*72)\*k);

VertexArray[9][2]=R\*cos(pi-alpha);

//10 точка

VertexArray[10][0]=R\*sin(pi-alpha)\*sin((36+3\*72)\*k);

VertexArray[10][1]=R\*sin(pi-alpha)\*cos((36+3\*72)\*k);

VertexArray[10][2]=R\*cos(pi-alpha);

//11 точка

VertexArray[11][0]=0;

VertexArray[11][1]=0;

VertexArray[11][2]=-R;

}

void ScWorker::syncSlot(){

int nSize = Memory::get("Size",0).toInt();

if(nSize != Size){

Size = nSize;

histA.clear();

histA.resize(Size\*BLOCKLANGTH);

histY.clear();

histY.resize(Size\*BLOCKLANGTH);

}

AngleMin = 180-Memory::get("PhMin",0).toInt();

AngleMax = Memory::get("PhMin",0).toInt()+180;

OffsetMin = Memory::get("ArgMin",0).toInt();

OffsetMax = Memory::get("ArgMax",1024).toInt();

maxColor = Memory::get("colorValue",0).toInt();

MaxBarier = Memory::get("Barier",100).toInt();

delY = (GLfloat)32.0f/(181-Memory::get("PhMin",0).toInt());

centerX = (GLfloat)Size/2;

centerY = AngleMin + (GLfloat) ( AngleMax - AngleMin ) /2;

centerZ = OffsetMin + (GLfloat) ( OffsetMax - OffsetMin ) /2;

}

void ScWorker::sharedSlot(int ship, QString type){

if(histA.size()==0){

syncSlot();

}

QString nameA,nameP;

if(type=="Vertical"){

nameA = "vVerticalAr";

nameP = "vVerticalPh";

}

else{

nameA = "vGorizontalAr";

nameP = "vGorizontalPh";

}

int shipPosition = ship\*BLOCKLANGTH\*sizeof(float);

Memory::getData(nameA,histA.data(),shipPosition);

Memory::getData(nameP,histY.data(),shipPosition);

plotSlot();

}

void ScWorker::dataSlot(Clowd &dataA, Clowd &dataH){

histA = dataA;

histY = dataH;

plotSlot();

}

void ScWorker::plotSlot(){

if(histA.size() == 0)

return;

centerX = (GLfloat)Size/2;

centerY = AngleMin + (GLfloat) ( AngleMax - AngleMin ) /2;

centerZ = OffsetMin + (GLfloat) ( OffsetMax - OffsetMin ) /2;

int realSize = 0;

m\_data.clear();

//размер = длина \* ширина \* 24 точки по 6 координат в каждой

//m\_data.resize( (OffsetMax-OffsetMin) \* Size \* 144);

m\_data.resize( (OffsetMax-OffsetMin) \* Size \* 60\*6);

m\_dataLink = m\_data.data();

for( int x = 0; x < Size; x++){

//номер пачки

int packet = x\*BLOCKLANGTH;

for(int z = OffsetMin; z < OffsetMax; z++){

int y = (int)histY[packet+z];

if(y >= AngleMin && y <= AngleMax){

uint color = (uint)histA[packet+z];

if(color >= MaxBarier){

plotFlower((int)x,(int)y,(int)z,color);

realSize++;

}

}

}

}

// m\_data.resize(realSize\*24\*6);

m\_data.resize(realSize\*60\*6);

emit plot(m\_data);

}

void ScWorker::plotFlower(int x,int y,int z, uint color){

double NormalColor = (double)(QUINT16\_SIZE);

int nColor = (int)MAXBYTE\*((double)color)/NormalColor;

if(nColor > MAXBYTE)

nColor = MAXBYTE;

appendPointObj((float)x-VertexArray[0][0],(float)y-VertexArray[0][1],(float)z-VertexArray[0][2],nColor);

appendPointObj((float)x-VertexArray[2][0],(float)y-VertexArray[2][1],(float)z-VertexArray[2][2],nColor);

appendPointObj((float)x-VertexArray[1][0],(float)y-VertexArray[1][1],(float)z-VertexArray[1][2],nColor);

appendPointObj((float)x-VertexArray[0][0],(float)y-VertexArray[0][1],(float)z-VertexArray[0][2],nColor);

appendPointObj((float)x-VertexArray[3][0],(float)y-VertexArray[3][1],(float)z-VertexArray[3][2],nColor);

appendPointObj((float)x-VertexArray[2][0],(float)y-VertexArray[2][1],(float)z-VertexArray[2][2],nColor);

// 2 треугольник

appendPointObj((float)x-VertexArray[0][0],(float)y-VertexArray[0][1],(float)z-VertexArray[0][2],nColor);

appendPointObj((float)x-VertexArray[4][0],(float)y-VertexArray[4][1],(float)z-VertexArray[4][2],nColor);

appendPointObj((float)x-VertexArray[3][0],(float)y-VertexArray[3][1],(float)z-VertexArray[3][2],nColor);

// 3 треугольник

appendPointObj((float)x-VertexArray[0][0],(float)y-VertexArray[0][1],(float)z-VertexArray[0][2],nColor);

appendPointObj((float)x-VertexArray[5][0],(float)y-VertexArray[5][1],(float)z-VertexArray[5][2],nColor);

appendPointObj((float)x-VertexArray[4][0],(float)y-VertexArray[4][1],(float)z-VertexArray[4][2],nColor);

// 4 треугольник

appendPointObj((float)x-VertexArray[0][0],(float)y-VertexArray[0][1],(float)z-VertexArray[0][2],nColor);

appendPointObj((float)x-VertexArray[1][0],(float)y-VertexArray[1][1],(float)z-VertexArray[1][2],nColor);

appendPointObj((float)x-VertexArray[5][0],(float)y-VertexArray[5][1],(float)z-VertexArray[5][2],nColor);

// 5 треугольник

appendPointObj((float)x-VertexArray[6][0],(float)y-VertexArray[6][1],(float)z-VertexArray[6][2],nColor);

appendPointObj((float)x-VertexArray[1][0],(float)y-VertexArray[1][1],(float)z-VertexArray[1][2],nColor);

appendPointObj((float)x-VertexArray[7][0],(float)y-VertexArray[7][1],(float)z-VertexArray[7][2],nColor);

// 6 треугольник

appendPointObj((float)x-VertexArray[7][0],(float)y-VertexArray[7][1],(float)z-VertexArray[7][2],nColor);

appendPointObj((float)x-VertexArray[1][0],(float)y-VertexArray[1][1],(float)z-VertexArray[1][2],nColor);

appendPointObj((float)x-VertexArray[2][0],(float)y-VertexArray[2][1],(float)z-VertexArray[2][2],nColor);

// 7 треугольник

appendPointObj((float)x-VertexArray[7][0],(float)y-VertexArray[7][1],(float)z-VertexArray[7][2],nColor);

appendPointObj((float)x-VertexArray[2][0],(float)y-VertexArray[2][1],(float)z-VertexArray[2][2],nColor);

appendPointObj((float)x-VertexArray[8][0],(float)y-VertexArray[8][1],(float)z-VertexArray[8][2],nColor);

// 8 треугольник

appendPointObj((float)x-VertexArray[8][0],(float)y-VertexArray[8][1],(float)z-VertexArray[8][2],nColor);

appendPointObj((float)x-VertexArray[2][0],(float)y-VertexArray[2][1],(float)z-VertexArray[2][2],nColor);

appendPointObj((float)x-VertexArray[3][0],(float)y-VertexArray[3][1],(float)z-VertexArray[3][2],nColor);

// 9 треугольник

appendPointObj((float)x-VertexArray[8][0],(float)y-VertexArray[8][1],(float)z-VertexArray[8][2],nColor);

appendPointObj((float)x-VertexArray[3][0],(float)y-VertexArray[3][1],(float)z-VertexArray[3][2],nColor);

appendPointObj((float)x-VertexArray[9][0],(float)y-VertexArray[9][1],(float)z-VertexArray[9][2],nColor);

// 10 треугольник

appendPointObj((float)x-VertexArray[9][0],(float)y-VertexArray[9][1],(float)z-VertexArray[9][2],nColor);

appendPointObj((float)x-VertexArray[3][0],(float)y-VertexArray[3][1],(float)z-VertexArray[3][2],nColor);

appendPointObj((float)x-VertexArray[4][0],(float)y-VertexArray[4][1],(float)z-VertexArray[4][2],nColor);

// 11 треугольник

appendPointObj((float)x-VertexArray[9][0],(float)y-VertexArray[9][1],(float)z-VertexArray[9][2],nColor);

appendPointObj((float)x-VertexArray[4][0],(float)y-VertexArray[4][1],(float)z-VertexArray[4][2],nColor);

appendPointObj((float)x-VertexArray[10][0],(float)y-VertexArray[10][1],(float)z-VertexArray[10][2],nColor);

// 12 треугольник

appendPointObj((float)x-VertexArray[10][0],(float)y-VertexArray[10][1],(float)z-VertexArray[10][2],nColor);

appendPointObj((float)x-VertexArray[4][0],(float)y-VertexArray[4][1],(float)z-VertexArray[4][2],nColor);

appendPointObj((float)x-VertexArray[5][0],(float)y-VertexArray[5][1],(float)z-VertexArray[5][2],nColor);

// 13 треугольник

appendPointObj((float)x-VertexArray[10][0],(float)y-VertexArray[10][1],(float)z-VertexArray[10][2],nColor);

appendPointObj((float)x-VertexArray[5][0],(float)y-VertexArray[5][1],(float)z-VertexArray[5][2],nColor);

appendPointObj((float)x-VertexArray[6][0],(float)y-VertexArray[6][1],(float)z-VertexArray[6][2],nColor);

// 14 треугольник

appendPointObj((float)x-VertexArray[6][0],(float)y-VertexArray[6][1],(float)z-VertexArray[6][2],nColor);

appendPointObj((float)x-VertexArray[5][0],(float)y-VertexArray[5][1],(float)z-VertexArray[5][2],nColor);

appendPointObj((float)x-VertexArray[1][0],(float)y-VertexArray[1][1],(float)z-VertexArray[1][2],nColor);

// 15 треугольник

appendPointObj((float)x-VertexArray[7][0],(float)y-VertexArray[7][1],(float)z-VertexArray[7][2],nColor);

appendPointObj((float)x-VertexArray[11][0],(float)y-VertexArray[11][1],(float)z-VertexArray[11][2],nColor);

appendPointObj((float)x-VertexArray[6][0],(float)y-VertexArray[6][1],(float)z-VertexArray[6][2],nColor);

// 16 треугольник

appendPointObj((float)x-VertexArray[8][0],(float)y-VertexArray[8][1],(float)z-VertexArray[8][2],nColor);

appendPointObj((float)x-VertexArray[11][0],(float)y-VertexArray[11][1],(float)z-VertexArray[11][2],nColor);

appendPointObj((float)x-VertexArray[7][0],(float)y-VertexArray[7][1],(float)z-VertexArray[7][2],nColor);

// 17 треугольник

appendPointObj((float)x-VertexArray[9][0],(float)y-VertexArray[9][1],(float)z-VertexArray[9][2],nColor);

appendPointObj((float)x-VertexArray[11][0],(float)y-VertexArray[11][1],(float)z-VertexArray[11][2],nColor);

appendPointObj((float)x-VertexArray[8][0],(float)y-VertexArray[8][1],(float)z-VertexArray[8][2],nColor);

// 18 треугольник

appendPointObj((float)x-VertexArray[10][0],(float)y-VertexArray[10][1],(float)z-VertexArray[10][2],nColor);

appendPointObj((float)x-VertexArray[11][0],(float)y-VertexArray[11][1],(float)z-VertexArray[11][2],nColor);

appendPointObj((float)x-VertexArray[9][0],(float)y-VertexArray[9][1],(float)z-VertexArray[9][2],nColor);

// 19 треугольник

appendPointObj((float)x-VertexArray[6][0],(float)y-VertexArray[6][1],(float)z-VertexArray[6][2],nColor);

appendPointObj((float)x-VertexArray[11][0],(float)y-VertexArray[11][1],(float)z-VertexArray[11][2],nColor);

appendPointObj((float)x-VertexArray[10][0],(float)y-VertexArray[10][1],(float)z-VertexArray[10][2],nColor);

}

void ScWorker::appendPointObj(float x,float y,float z, int color){

\*(m\_dataLink++) = (centerX-x)/MAXBYTEFLOAT;

\*(m\_dataLink++) = (centerY-y)/(MAXBYTEFLOAT\*delY);

\*(m\_dataLink++) = -(centerZ-z)/MAXBYTEFLOAT;

if(color == 0) {

\*(m\_dataLink++) = 0;

\*(m\_dataLink++) = 0;

\*(m\_dataLink++) = 0;

}

else {

\*(m\_dataLink++) = (float)color/MAXBYTEFLOAT;

\*(m\_dataLink++) = 0.0f;

\*(m\_dataLink++) = (float)(MAXBYTEFLOAT - color)/MAXBYTEFLOAT;

}

}

14. plotpolarization.cpp - класс, реализующий вывод растрового изображения Амплитудно-фазовой характеристики.

Код:

#include "plotpolarization.h"

#include "ui\_plotpolarization.h"

#include "memory.h"

PlotPolarization::PlotPolarization(QWidget \*parent) :

QWidget(parent),

ui(new Ui::PlotPolarization)

{

ui->setupUi(this);

customPlot = ui->plot;

customPlot->setInteractions(QCP::iRangeZoom); //QCP::iRangeDrag| this will also allow rescaling the color scale by dragging/zooming

customPlot->axisRect()->setupFullAxesBox(true);

customPlot->xAxis->setLabel("Градусы");

customPlot->yAxis->setLabel("Отсчеты");

colorMap = new QCPColorMap(customPlot->xAxis, customPlot->yAxis);

customPlot->addPlottable(colorMap);

syncSlot();

worker = new PPworker();

worker->moveToThread(&mathThread);

connect(&mathThread, &QThread::finished, worker, &QObject::deleteLater);

connect(this,&PlotPolarization::syncSignal,worker,&PPworker::syncSlot);

connect(this,&PlotPolarization::dataSignal,worker,&PPworker::dataSlot);

connect(this,&PlotPolarization::sharedSignal,worker,&PPworker::sharedSlot);

connect(this,&PlotPolarization::plotSignal,worker,&PPworker::plotSlot);

connect(worker,&PPworker::plot,this,&PlotPolarization::result);

mathThread.start();

sharedTimer = new QTimer();

sharedTimer->setInterval(40);

connect(sharedTimer,&QTimer::timeout,this,&PlotPolarization::sharedNow);

isShared = false;

}

void PlotPolarization::sharedNow(){

if(isShared){

isShared = false;

sharedTimer->stop();

}

else return;

emit sharedSignal(sharedShip,Type);

}

void PlotPolarization::shared(int shp){

isShared = true;

sharedShip = shp;

if(!sharedTimer->isActive()){

sharedTimer->start();

}

}

PlotPolarization::~*PlotPolarization*()

{

delete ui;

mathThread.quit();

mathThread.wait();

}

void PlotPolarization::syncSlot(){

ArgMin = Memory::get("ArgMin",0).toInt();

ArgMax = Memory::get("ArgMax",1024).toInt();

PhMin = 0;

PhMax = Memory::get("PhMin",180).toInt();

leRasterPeriod = Memory::get("leRasterPeriod"+Type,10).toInt();

Lay = Memory::get("PlotPLay"+Type,0).toInt();

Barier = Memory::get("Barier",0).toInt();

int size = Memory::get("Size",400).toInt();

if(size!=Size){

ui->horizontalSlider->setMaximum(size);

ui->lay->setMaximum(size);

Size = size;

}

emit syncSignal(Type);

emit plotSignal();

// plot();

}

void PlotPolarization::setType(QString type){

Type = type;

}

void PlotPolarization::setBuf(Clowd &Ar, Clowd &Ph){

if(sharedTimer->isActive()){

sharedTimer->stop();

}

Amp = Ar;

Phase = Ph;

emit dataSignal(Ar, Ph);

}

void PlotPolarization::result(QCPColorMapData \*data){

colorMap->setData(data);

colorMap->setGradient(QCPColorGradient::gpThermal);

colorMap->rescaleDataRange();

customPlot->rescaleAxes();

customPlot->replot();

}

void PlotPolarization::plot(){

emit plotSignal();

}

void PlotPolarization::on\_rasterPeriod\_valueChanged(int arg1)

{

leRasterPeriod = arg1;

Memory::set("leRasterPeriod"+Type,arg1);

emit sync();

emit syncSignal(Type);

}

void PlotPolarization::on\_lay\_valueChanged(int arg1)

{

Lay = arg1;

Memory::set("PlotPLay"+Type,arg1);

emit sync();

emit syncSignal(Type);

}

void PlotPolarization::on\_horizontalSlider\_valueChanged(int value)

{

Lay = value;

Memory::set("PlotPLay"+Type,value);

emit sync();

emit syncSignal(Type);

}

15. ppworker.cpp - класс, преобразующий данные из Rdata(5), а именно массива для вывода на графики, в массив, который требуется для отображения битовой карты PlotPolarization(14).

Код:

#include "ppworker.h"

#include "memory.h"

PPworker::PPworker(QObject \*parent) : QObject(parent)

{

}

void PPworker::syncSlot(QString type){

int nSize = Memory::get("Size",0).toInt();

if(nSize != Size){

Size = nSize;

histA.clear();

histA.resize(Size\*BLOCKLANGTH);

histY.clear();

histY.resize(Size\*BLOCKLANGTH);

}

AngleMin = 180-Memory::get("PhMin",0).toInt();

AngleMax = Memory::get("PhMin",0).toInt()+180;

OffsetMin = Memory::get("ArgMin",0).toInt();

OffsetMax = Memory::get("ArgMax",1024).toInt();

maxColor = Memory::get("colorValue",0).toInt();

MaxBarier = Memory::get("Barier",100).toInt();

leRasterPeriod = Memory::get("leRasterPeriod"+type,10).toInt();

Lay = Memory::get("PlotPLay"+type,0).toInt();

}

void PPworker::sharedSlot(int ship, QString type){

if(histA.size()==0){

syncSlot(type);

}

QString nameA,nameP;

if(type=="Vertical"){

nameA = "vVerticalAr";

nameP = "vVerticalPh";

}

else{

nameA = "vGorizontalAr";

nameP = "vGorizontalPh";

}

int shipPosition = ship\*BLOCKLANGTH\*sizeof(float);

Memory::getData(nameA,histA.data(),shipPosition);

Memory::getData(nameP,histY.data(),shipPosition);

plotSlot();

}

void PPworker::dataSlot(Clowd &dataA, Clowd &dataH){

histA = dataA;

histY = dataH;

plotSlot();

}

void PPworker::plotSlot(){

if(histA.size()==0){

return;

}

int nx =AngleMax - AngleMin;

int ny =OffsetMax - OffsetMin;

QCPColorMapData \*data = new QCPColorMapData(nx,ny,QCPRange(AngleMin, AngleMax), QCPRange(OffsetMin, OffsetMax));

double x, y;

float Max = 0.0;

if(histY.size() >= Size\*BLOCKLANGTH){

for (int xIndex=0; xIndex<nx; ++xIndex)

{

for (int yIndex=0; yIndex<ny; ++yIndex)

{

data->cellToCoord(xIndex, yIndex, &x, &y);

int rY = (int)(x+180);

int rZ = (int)round(y);

float A = 0;

int P = 0;

int Step = 0;

for(P = Lay, Step = 0; Step <= leRasterPeriod && P < Size; ++Step, ++P){

if((int)histY[P\*BLOCKLANGTH+rZ] == rY){

double AmpR = (double)histA[P\*BLOCKLANGTH+rZ];

if(AmpR >= MaxBarier)

A+= (AmpR - MaxBarier);

}

}

if(Max < A) Max = A;

data->setCell(xIndex, yIndex, A);

}

}

}

emit plot(data);

}

16. debugdialog.cpp форма для вывода отладочной информации.

Код:

#include "debugdialog.h"

#include "ui\_debugdialog.h"

DebugDialog::DebugDialog(QWidget \*parent) :

QWidget(parent),

ui(new Ui::DebugDialog)

{

ui->setupUi(this);

}

DebugDialog::~*DebugDialog*()

{

delete ui;

}

void DebugDialog::log(QString msg){

ui->list->addItem(msg);

}

17. plugwin.cpp - класс для загрузки и включения/отключения отдельно написанных и скомпилированных плагинов.

Код:

#include "plugwin.h"

#include "ui\_plugwin.h"

#include <QFileDialog>

#include <QFileInfo>

#include <QCheckBox>

#include <QStyle>

#include <QDesktopWidget>

#include "memory.h"

PlugWin::PlugWin(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::PlugWin)

{

ui->setupUi(this);

this->setWindowFlags(Qt::WindowStaysOnTopHint);

this->setGeometry(

QStyle::alignedRect(

Qt::LeftToRight,

Qt::AlignCenter,

this->size(),

qApp->desktop()->availableGeometry()

)

);

ui->tableWidget->setColumnCount(3);

QHeaderView \*header = new QHeaderView(Qt::Horizontal,ui->tableWidget);

header->setSectionResizeMode(QHeaderView::Stretch);

ui->tableWidget->verticalHeader()->hide();

ui->tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->tableWidget->horizontalHeader()->setSectionResizeMode(0,QHeaderView::Interactive);

ui->tableWidget->horizontalHeader()->resizeSection(1, 30);

ui->tableWidget->horizontalHeader()->hide();

}

PlugWin::~*PlugWin*()

{

delete ui;

}

void PlugWin::on\_addPlug\_clicked()

{

QString fileName = QFileDialog::getOpenFileName(this, tr("Добавить плагин"), Memory::get("lastPlugDir","").toString(),

tr("Файлы плагинов (\*.so)"));

if (fileName != "") {

Memory::set("lastPlugDir",fileName);

QPluginLoader loader(fileName);

Plugs p;

p.enable = true;

p.name = loader.metaData().value("MetaData").toObject().value("Descr").toString();

p.alias = loader.metaData().value("MetaData").toObject().value("Name").toString();

p.path = fileName;

// emit loadPlugin(p.name,alias,loader);

QStringList plugs = Memory::get("plugins","").toStringList();

plugs << p.path;

Memory::set("plugins",plugs);

emit sync();

tableAppend(p);

}

}

void PlugWin::load(){

QStringList plugs = Memory::get("plugins","").toStringList();

foreach (QString plugPath, plugs) {

QFileInfo check\_file(plugPath);

if(check\_file.exists()){

QPluginLoader loader(plugPath);

Plugs p;

p.enable = true;

p.name = loader.metaData().value("MetaData").toObject().value("Descr").toString();

p.alias = loader.metaData().value("MetaData").toObject().value("Name").toString();

p.path = plugPath;

tableAppend(p);

}

}

}

void PlugWin::tableAppend(Plugs &p){

QFileInfo check\_file(p.path);

if(check\_file.exists()){

plugList.insert(p.name,p);

int rows = ui->tableWidget->rowCount();

ui->tableWidget->insertRow(rows);

QTableWidgetItem \*item0 = new QTableWidgetItem();

QTableWidgetItem \*item1 = new QTableWidgetItem();

QTableWidgetItem \*item2 = new QTableWidgetItem();

item0->setTextAlignment(Qt::AlignHCenter);

/\*if (p.enable)

item0->setCheckState(Qt::Checked);

else

item0->setCheckState(Qt::Unchecked);\*/

item1->setText(p.name);

item2->setText(p.path);

ui->tableWidget->setItem(rows, 0, item0);

ui->tableWidget->setItem(rows, 1, item1);

ui->tableWidget->setItem(rows, 2, item2);

QWidget \*pWidget = new QWidget();

QCheckBox \*pCheckBox = new QCheckBox();

QHBoxLayout \*pLayout = new QHBoxLayout(pWidget);

pCheckBox->setChecked(p.enable);

// connect(pCheckBox,&QCheckBox::toggled,this,&PlugWin::checked);

//PlugWin \*tw = this;

connect(pCheckBox,&QCheckBox::toggled,[=](bool checked){this->checkedItem(item0,checked);});

pLayout->addWidget(pCheckBox);

pLayout->setAlignment(Qt::AlignCenter);

pLayout->setContentsMargins(0,0,0,0);

pWidget->setLayout(pLayout);

ui->tableWidget->setCellWidget(0,0,pWidget);

checkedItem(item0,p.enable);

/\*QCheckBox \*chb = new QCheckBox("");

chb->setChecked(p.enable);

chb->setObjectName(p.name);

ui->tableWidget->setCellWidget(rows,0,chb);\*/

}

}

void PlugWin::checked(bool check){

bool status = check;

if(status){}

}

void PlugWin::checkedItem(QTableWidgetItem \*itm, bool check){

int row = itm->row();

QString name = ui->tableWidget->item(row,1)->text();

bool status = check;

if(plugList[name].enable!=status) {

plugList[name].enable = status;

}

if(status){

emit loadPlugin(plugList[name]);

}

else

emit hidePlugin(plugList[name]);

//bool status = itm->checkState();

}

18. pluginterface.h содержит интерфейс, реализация которого позволяет написать совместимый с программой плагин.

Код:

#ifndef PLUGINTERFACE\_H

#define PLUGINTERFACE\_H

#include <QObject>

#include <QWidget>

#include "nomain/rdata.h"

class SecondaryProcessingInterfaces: public QObject

{

public:

virtual ~*SecondaryProcessingInterfaces*() {}

signals:

void processComplete();

};

#define SecondaryProcessingInterfaces\_iid "ru.nixson.physicks.Rip3p.SecondaryProcessingInterfaces"

Q\_DECLARE\_INTERFACE(SecondaryProcessingInterfaces, SecondaryProcessingInterfaces\_iid);

class PlugWidgetInterfaces: public QWidget

{

public:

virtual ~*PlugWidgetInterfaces*() {}

virtual void *setMemory*(Rdata \*mLink) = 0;

virtual void *init*() = 0;

signals:

void processStep(int);

void processStop(int);

void processStart();

void sync();

public slots:

void syncSlot();

};

#define PlugWidgetInterfaces\_iid "ru.nixson.physicks.Rip3p.PlugWidgetInterfaces"

Q\_DECLARE\_INTERFACE(PlugWidgetInterfaces, PlugWidgetInterfaces\_iid);

#endif // PLUGINTERFACE\_H

19 и 20. Пример реализации стороннего плагина.