**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 1**

Тема: Построение изображений 2D-кривых

Студент: Айрапетова Евгения Ашотовна

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата: 14.12.2021

Оценка:

Москва, 2021

1. Постановка задачи

Написать и отладить программу, строящую изображение заданной замечательной кривой.

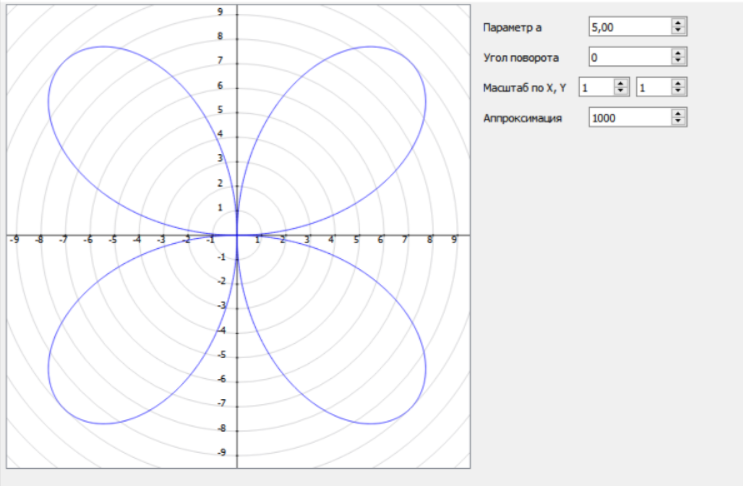
Вариант: 1 (ρ = a ∗ sin(2 ∗ φ))

ρ,φ- полярные координаты. a,b, k,A,B, - константы, значения которых выбираются пользователем (вводятся в окне программы). a,b>0 Обеспечить автоматическое масштабирование и центрирование кривой при изменении размеров окна.

1. Описание программы

Для решения задачи в файлах graphicsitem.h (.cpp) созданы 2 класса Grid, в котором реализованы построения осей координат, вспомогательных линий и Chart - построение заданной кривой. Оси координат отрисовываются от центра окна, которое рассчитывается от высоты и ширины окна. Далее ставится разметка значений на осях. При изменении масштаба, эти значения будут изменяться. Также отрисовываются вспомогательные окружности, для улучшения отображения графика в целом.

1. Результаты выполнения тестов



1. Листинг программы

main.cpp:

// Айрапетова вариант 1

#include "mainwindow.h"

#include <QApplication>

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

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

graphicsitem.h:

#ifndef GRAPHICSITEM\_H

#define GRAPHICSITEM\_H

#include <QAbstractGraphicsShapeItem>

#include <QtWidgets>

#include <math.h>

class Grid : public QGraphicsItem {

public:

Grid(QGraphicsItem\* parent = nullptr);

QRectF boundingRect() const;

void paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget);

void setScaleX(qreal scaleX);

void setScaleY(qreal scaleY);

void setAngle(qreal angle);

private:

qreal m\_space;

qreal m\_scaleX;

qreal m\_scaleY;

qreal m\_angle;

};

class Chart : public QGraphicsItem {

public:

Chart(QGraphicsItem\* parent = nullptr);

QRectF boundingRect() const;

void paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget);

void setScaleX(qreal scaleX);

void setScaleY(qreal scaleY);

void setA(qreal a);

void setApprox(qreal approx);

void setAngle(qreal angle);

protected:

qreal f\_phi(qreal phi) const;

qreal scaleX(qreal x) const;

qreal scaleY(qreal y) const;

private:

qreal m\_a;

qreal m\_b;

qreal m\_approx;

qreal m\_space;

qreal m\_scaleX;

qreal m\_scaleY;

qreal m\_angle;

};

#endif // GRAPHICSITEM\_H

graphicsitem.cpp:

#include "graphicsitem.h"

Grid::Grid(QGraphicsItem \*parent):

QGraphicsItem(parent),

m\_space{30},

m\_scaleX{1},m\_scaleY{1},

m\_angle{0}

{ }

QRectF Grid::boundingRect() const {

return scene()->views().first()->rect();

}

void Grid::paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) {

Q\_UNUSED(option); Q\_UNUSED(widget);

painter->setRenderHint(QPainter::Antialiasing);

qreal w = boundingRect().width();

qreal h = boundingRect().height();

qreal start = -m\_space\*50;

//рисуем вспомогательные окружности

for(auto chunk = start ; chunk <= w+qAbs(start) ; chunk+=m\_space) {

painter->save();

painter->setPen(QPen(QBrush(QColor(Qt::gray)), 0.3, Qt::PenStyle::SolidLine));

painter->drawEllipse(QPointF((int)(w/2), (int)(h/2)),chunk,chunk);

painter->restore();

}

painter->save();

//меняем начало координат

painter->translate((int)(w/2), (int)(h/2));

painter->rotate(m\_angle);

//Рисуем оси Оx и Оy

painter->drawLine(QLineF(0,0, 0,-h));

painter->drawLine(QLineF(0,0, 0, h));

painter->drawLine(QLineF(0,0, w, 0));

painter->drawLine(QLineF(0,0, -w, 0));

//Рисуем разметку для осей Оx и Оy

qreal scX = m\_scaleX;

for(auto chunk = m\_space ; chunk <= w ; chunk+=m\_space, scX+=m\_scaleX) {

painter->drawLine(chunk, -1, static\_cast<qreal>(chunk),1);

painter->drawText(chunk-7, 10, QString::number(scX, 'g', 3));

painter->drawLine(-chunk, -1,(-1)\* static\_cast<qreal>(chunk),1);

painter->drawText(-chunk-7, 10, QString::number(-scX, 'g', 3));

}

qreal scY = m\_scaleY;

for(auto chunk = -m\_space ; chunk >= -w ; chunk-=m\_space, scY+=m\_scaleY) {

painter->drawLine(-1, chunk,1, static\_cast<qreal>(chunk));

painter->drawText(-23, chunk, QString::number(scY, 'g', 3));

painter->drawLine(-1, -chunk,1,(-1)\* static\_cast<qreal>(chunk));

painter->drawText(-23, -chunk, QString::number(-scY, 'g', 3));

}

painter->restore();

}

void Grid::setScaleX(qreal scaleX){

m\_scaleX = scaleX;

}

void Grid::setScaleY(qreal scaleY){

m\_scaleY = scaleY;

}

void Grid::setAngle(qreal angle){

m\_angle = (-1)\*angle;

}

Chart::Chart(QGraphicsItem\* parent):

QGraphicsItem(parent),

m\_a{5},m\_b{200},

m\_approx{1000},

m\_space{30},

m\_scaleX{0.5},m\_scaleY{0.5},

m\_angle{0}

{}

QRectF Chart::boundingRect() const {

return scene()->views().first()->rect();

}

void Chart::paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) {

Q\_UNUSED(option); Q\_UNUSED(widget);

qreal w = boundingRect().width();

qreal h = boundingRect().height();

painter->setRenderHint(QPainter::Antialiasing);

painter->save();

//меняем точку начала координат

painter->translate((int)(w/2), (int)(h/2));

painter->rotate(m\_angle);

//ширина и цвет линии графика

painter->setPen(QPen(QColor(Qt::blue), 1));

//размечаем точки для графика

QPainterPath path(QPointF(0.0, 0.0));

qreal step = M\_PI/m\_approx; //параметр апроксимации

for(qreal phi = 0.0 ; phi <= 2\*M\_PI ; phi+=step) {

qreal ro = f\_phi(phi);

path.lineTo(scaleX(ro\*qCos(phi)), scaleY(ro\*qSin(phi)));

}

qreal ro = f\_phi(2\*M\_PI);

path.lineTo(scaleX(ro\*qCos(2\*M\_PI)), scaleY(ro\*qSin(2\*M\_PI)));

painter->drawPath(path);

painter->restore();

}

qreal Chart::f\_phi(qreal phi) const {

return m\_a\*qSin(2\*phi);

}

qreal Chart::scaleX(qreal x) const {

return m\_space \* x / m\_scaleX;

}

qreal Chart::scaleY(qreal y) const {

return m\_space \* y / m\_scaleY;

}

void Chart::setScaleX(qreal scaleX){

m\_scaleX = scaleX;

}

void Chart::setScaleY(qreal scaleY){

m\_scaleY = scaleY;

}

void Chart::setA(qreal a){

m\_a = a;

}

void Chart::setApprox(qreal approx){

m\_approx = approx;

}

void Chart::setAngle(qreal angle){

m\_angle = (-1)\*angle;

}

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QtWidgets>

#include <QScrollArea>

#include <QAbstractScrollArea>

#include "graphicsitem.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

bool eventFilter(QObject \*watched, QEvent \*event);

~MainWindow();

public slots:

void ChangeScale();

void ChangeA();

void ChangeApprox();

void Rotate();

private:

Grid\* grid;

Chart\* chart;

Ui::MainWindow \*ui;

QGraphicsScene scene;

QPoint curr\_cursor\_pos;

QPoint cursor\_pos;

bool left\_button\_pressed;

};

#endif // MAINWINDOW\_H

mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QDebug>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

left\_button\_pressed = false;

setWindowTitle("2d graphic.exe");

scene.setObjectName("scene"); //scene объявлен в MainWindow

ui->gv->setScene(&scene); //QGraphicsView на ui

grid = new Grid;

chart = new Chart;

scene.addItem(grid);

scene.addItem(chart);

// устанавливаем фильтр событий для перехвата событий колёсика

scene.installEventFilter(this);

}

void MainWindow::ChangeScale(){

grid->setScaleX(ui->ScaleX->value());

grid->setScaleY(ui->ScaleY->value());

chart->setScaleX(ui->ScaleX->value());

chart->setScaleY(ui->ScaleY->value());

scene.update();

}

void MainWindow::ChangeA(){

chart->setA(ui->param\_a->value());

scene.update();

}

void MainWindow::ChangeApprox(){

chart->setApprox(ui->Approx->value());

scene.update();

}

void MainWindow::Rotate(){

grid->setAngle(ui->angle->value());

chart->setAngle(ui->angle->value());

scene.update();

}

bool MainWindow::eventFilter(QObject \*watched, QEvent \*event) {

if(watched->objectName() == "scene") {

if(event->type() == QEvent::GraphicsSceneWheel) {

QGraphicsSceneWheelEvent\* wevent = static\_cast<QGraphicsSceneWheelEvent\*>(event);

//изменяем масштаб

ui->gv->scale(wevent->delta() < 0 ? 0.9 : 1.1, wevent->delta() < 0 ? 0.9 : 1.1);

//центрируем в точке позиции мыши на сцене

if(wevent->delta() < 0) ui->gv->centerOn(wevent->scenePos());

scene.update();

event->accept();

return true;

}

}

/\*if (event->type() == QEvent::GraphicsSceneMousePress) {

//QMouseEvent \* pe = (QMouseEvent\*) event;

QTouchEvent\* pe = static\_cast<QTouchEvent\*>(event);

qDebug() << pe->type();

left\_button\_pressed =true;

//cursor\_pos = pe->touchPoints()

QApplication::setOverrideCursor(Qt::ClosedHandCursor);

scene.update();

event->accept();

qDebug() << "Ok";

return true;

}

if (event->type() == QEvent::GraphicsSceneMouseRelease) {

//QMouseEvent \* pe = (QMouseEvent\*) event;

//QTouchEvent\* pe = static\_cast<QTouchEvent\*>(event);

left\_button\_pressed = true;

cursor\_pos = QPoint(0,0);

QApplication::restoreOverrideCursor();

return true;

}

else if (event->type() == QEvent::GraphicsSceneMouseMove) {

QTouchEvent\* pe = static\_cast<QTouchEvent\*>(event);

//QGraphicsSceneMouseEvent\* pe = static\_cast<QGraphicsSceneMouseEvent\*>(event);

// QMouseEvent \* pe = (QMouseEvent\*) event;

QPoint curr\_cursor\_pos = pe->scenePos();

QPoint d\_coord = curr\_cursor\_pos - cursor\_pos;

int x = ui->gv->horizontalScrollBar()->value() - d\_coord.x();

int y = ui->gv->verticalScrollBar()->value() - d\_coord.y();

if (x < 0)

x = 0;

if (y < 0)

y = 0;

ui->gv->horizontalScrollBar()->setValue(x);

ui->gv->verticalScrollBar()->setValue(y);

cursor\_pos = curr\_cursor\_pos - d\_coord;

scene.update();

event->accept();

return true;

}\*/

return false;

}

MainWindow::~MainWindow() {

delete ui;

}

ВЫВОД

Я научилась основам работы с графикой в Qt, а также написала программу для построения графика. Для этого я использовала различные преобразования координат и кривых.

ЛИТЕРАТУРА

Документация по Qt: https://doc.qt.io (дата обращения 05.12.2021).