// StripsDlg.cpp : implementation file

//

#include "pch.h"

#include "framework.h"

#include "Strips.h"

#include "StripsDlg.h"

#include "afxdialogex.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

using namespace ClipperLib;

// CAboutDlg dialog used for App About

class CAboutDlg : public CDialogEx

{

public:

CAboutDlg();

// Dialog Data

#ifdef AFX\_DESIGN\_TIME

enum { IDD = IDD\_ABOUTBOX };

#endif

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV support

// Implementation

protected:

DECLARE\_MESSAGE\_MAP()

};

CAboutDlg::CAboutDlg() : CDialogEx(IDD\_ABOUTBOX)

{

}

void CAboutDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

}

BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)

END\_MESSAGE\_MAP()

// CStripsDlg dialog

CStripsDlg::CStripsDlg(CWnd\* pParent /\*=nullptr\*/)

: CDialogEx(IDD\_STRIPS\_DIALOG, pParent)

{

m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

}

void CStripsDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

DDX\_Control(pDX, IDC\_SLIDER1, XPosition);

DDX\_Control(pDX, IDC\_SLIDER2, YPosition);

DDX\_Control(pDX, IDC\_SLIDER3, AngleRotation);

DDX\_Control(pDX, IDC\_EDIT1, HausdorffDistance);

DDX\_Control(pDX, IDC\_EDIT2, NumOfCoveredBlocks);

DDX\_Control(pDX, IDC\_EDIT3, CurrXPos);

DDX\_Control(pDX, IDC\_EDIT4, CurrYPos);

DDX\_Control(pDX, IDC\_EDIT5, CurrAnglePos);

}

BEGIN\_MESSAGE\_MAP(CStripsDlg, CDialogEx)

ON\_WM\_SYSCOMMAND()

ON\_WM\_PAINT()

ON\_WM\_QUERYDRAGICON()

ON\_BN\_CLICKED(IDC\_BUTTON1, &CStripsDlg::add\_net)

ON\_BN\_CLICKED(IDC\_BUTTON2, &CStripsDlg::draw\_everything)

ON\_BN\_CLICKED(IDC\_BUTTON3, &CStripsDlg::add\_oval)

ON\_BN\_CLICKED(IDC\_BUTTON4, &CStripsDlg::add\_intersec\_oval\_and\_net)

ON\_WM\_HSCROLL()

ON\_BN\_CLICKED(IDC\_BUTTON5, &CStripsDlg::find\_haus\_dist)

ON\_BN\_CLICKED(IDC\_BUTTON6, &CStripsDlg::do\_check\_all\_positions)

END\_MESSAGE\_MAP()

// CStripsDlg message handlers

Clipper cp;

std::vector<Path> CurrInts(1); // внутренний (для общей функции пересечения) промежуточный результат

// для функции пересечения (чтобы каждый раз не объявлять новый)

std::vector<Path> CurrentIntersection(1); // внешний промежуточный результат (для конкретной функции).

//Можно было бы и без него, но так код читабельнее

CRect RedrawArea;

int scale = 20;

// чтобы не потерять точность

int scale\_helper = 1000;

int window\_center\_x = 0;

int window\_center\_y = 0;

CString TextForCtrl = \_T("");

BOOL CStripsDlg::OnInitDialog()

{

CDialogEx::OnInitDialog();

// Add "About..." menu item to system menu.

// IDM\_ABOUTBOX must be in the system command range.

ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);

ASSERT(IDM\_ABOUTBOX < 0xF000);

CMenu\* pSysMenu = GetSystemMenu(FALSE);

if (pSysMenu != nullptr)

{

BOOL bNameValid;

CString strAboutMenu;

bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);

ASSERT(bNameValid);

if (!strAboutMenu.IsEmpty())

{

pSysMenu->AppendMenu(MF\_SEPARATOR);

pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);

}

GetClientRect(&RedrawArea);

window\_center\_x = RedrawArea.Width() / 2;

window\_center\_y = RedrawArea.Height() / 2;

RedrawArea.SetRect(window\_center\_x - 10 \* scale, window\_center\_y - 10 \* scale, window\_center\_x + 10 \* scale, window\_center\_y + 10 \* scale);

set\_drawing\_param(window\_center\_x, window\_center\_y, scale, scale\_helper);

}

// Set the icon for this dialog. The framework does this automatically

// when the application's main window is not a dialog

SetIcon(m\_hIcon, TRUE); // Set big icon

SetIcon(m\_hIcon, FALSE); // Set small icon

// TODO: Add extra initialization here

return TRUE; // return TRUE unless you set the focus to a control

}

void CStripsDlg::OnSysCommand(UINT nID, LPARAM lParam)

{

if ((nID & 0xFFF0) == IDM\_ABOUTBOX)

{

CAboutDlg dlgAbout;

dlgAbout.DoModal();

}

else

{

CDialogEx::OnSysCommand(nID, lParam);

}

}

// If you add a minimize button to your dialog, you will need the code below

// to draw the icon. For MFC applications using the document/view model,

// this is automatically done for you by the framework.

std::vector<Paths> Net(1);

std::vector<Path> OvalKassini(1); // изначальный в начале координат. На его основании поворачиваем и двигаем

std::vector<Path> RotatedAndMovedOvalKassini(1);

std::vector<Paths> Intersections(1);

std::vector<Paths> DotsOfCurrBlock(1);

// чтобы рисовать объекты типа Clipperlib::vector<Path> их надо превартить в структуры типа POINT

POINT \*\* StructureForDrawPaths;

POINT \* StructureForDrawPath;

CPen blackpen(PS\_SOLID, 1, RGB(0, 0, 0));

CPen redpen(PS\_SOLID, 1, RGB(255, 0, 0));

CPen bluepen(PS\_SOLID, 1, RGB(0, 0, 255));

int HausBlock = 0; // какой блок соответствует Хаусдорфовому расстоянию

int HausDot = 0; // какая точка в этом блоке соответствует Хаусдорфовому расстоянию

int XOvalHausPrev = 0; // точка на овале Кассини, соответствующая расстоянию Хаусдорфа

int YOvalHausPrev = 0;

int XOvalHaus = 0; // точка на овале Кассини, соответствующая расстоянию Хаусдорфа

int YOvalHaus = 0;

void CStripsDlg::OnPaint()

{

if (IsIconic())

{

CPaintDC dc(this); // device context for painting

SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);

// Center icon in client rectangle

int cxIcon = GetSystemMetrics(SM\_CXICON);

int cyIcon = GetSystemMetrics(SM\_CYICON);

CRect rect;

GetClientRect(&rect);

int x = (rect.Width() - cxIcon + 1) / 2;

int y = (rect.Height() - cyIcon + 1) / 2;

// Draw the icon

dc.DrawIcon(x, y, m\_hIcon);

}

else

{

CPaintDC dc(this); // device context for painting

// рисуем сетку

dc.SelectObject(blackpen);

if (Net[0].size() > 0)

{

StructureForDrawPaths = make\_structure\_for\_draw(StructureForDrawPaths, Net);

for (int i = 0; i < Net[0].size(); i++) {

dc.Polygon(StructureForDrawPaths[i], Net[0][i].size());

}

}

// рисуем овал Кассини

dc.SelectObject(redpen);

if (RotatedAndMovedOvalKassini[0].size() > 0)

{

StructureForDrawPath = make\_structure\_for\_draw(StructureForDrawPath, RotatedAndMovedOvalKassini);

dc.Polyline(StructureForDrawPath, RotatedAndMovedOvalKassini[0].size());

}

// рисуем пересечение

dc.SelectObject(bluepen);

if (Intersections[0].size() > 0)

{

StructureForDrawPaths = make\_structure\_for\_draw(StructureForDrawPaths, Intersections);

for (int i = 0; i < Intersections[0].size(); i++) {

dc.Polygon(StructureForDrawPaths[i], Intersections[0][i].size());

}

}

// рисуем ближайшую точку

dc.SelectObject(redpen);

if (Net[0].size() > 0) {

dc.Ellipse(window\_center\_x + (scale \* Net[0][HausBlock][HausDot].X / scale\_helper) - 3,

window\_center\_y - (scale \* Net[0][HausBlock][HausDot].Y / scale\_helper) - 3,

window\_center\_x + (scale \* Net[0][HausBlock][HausDot].X / scale\_helper) + 3,

window\_center\_y - (scale \* Net[0][HausBlock][HausDot].Y / scale\_helper) + 3);

// и само расстояние Хаусдорфа

dc.MoveTo(window\_center\_x + (scale \* Net[0][HausBlock][HausDot].X / scale\_helper),

window\_center\_y - (scale \* Net[0][HausBlock][HausDot].Y / scale\_helper));

dc.LineTo(window\_center\_x + (scale \* XOvalHaus / scale\_helper),

window\_center\_y - (scale \* YOvalHaus / scale\_helper));

}

}

}

// The system calls this function to obtain the cursor to display while the user drags

// the minimized window.

HCURSOR CStripsDlg::OnQueryDragIcon()

{

return static\_cast<HCURSOR>(m\_hIcon);

}

//int scale\_slider = 20; // так как слайдер принимает только целые занчения

// создаем сетку

void CStripsDlg::add\_net()

{

double w = 1;

double h = w;

XPosition.SetRange(-w\*2, w\*2, 1);

XPosition.SetPos(0);

YPosition.SetRange(-w\*2, w\*2, 1);

YPosition.SetPos(0);

AngleRotation.SetRange(0, 360, 1);

AngleRotation.SetPos(0);

// начинаем рисовать сетку из верхнего левого угла (двигаемся вправо и вниз). Начало координат в середине окна

double init\_x = -8 - 1;

double init\_y = 4 \* 2 + 1;

double step\_x = w;

double step\_y = h;

// координаты начала отрисовки текущего блока

double current\_x = init\_x;

double current\_y = init\_y;

while (current\_y > -1 \* init\_y)

{

add\_block(current\_x, current\_y, step\_x, step\_y);

current\_x += step\_x;

if (current\_x > -1 \* init\_x - 1)

{

current\_x = init\_x;

current\_y -= step\_y;

}

}

// TODO: Add your control notification handler code here

}

// создание отдельного блока для сетки

void CStripsDlg::add\_block(double init\_value\_x, double init\_value\_y, double step\_value\_x, double step\_value\_y)

{

std::vector<Path> Block(1);

//по часовой стрелке с левого верхнего угла

Block[0] << IntPoint(init\_value\_x \* scale\_helper, init\_value\_y \* scale\_helper);

Block[0] << IntPoint((init\_value\_x + step\_value\_x) \* scale\_helper, init\_value\_y \* scale\_helper);

Block[0] << IntPoint((init\_value\_x + step\_value\_x) \* scale\_helper, (init\_value\_y - step\_value\_y) \* scale\_helper);

Block[0] << IntPoint(init\_value\_x \* scale\_helper, (init\_value\_y - step\_value\_y) \* scale\_helper);

Net[0].push\_back(Block[0]);

}

double a = 5.5;

double c = 5;

double N = 16;

double y = 0;

double x\_current = 0;

double y\_current = 0;

// создаем овал Кассини

void CStripsDlg::add\_oval()

{

if (OvalKassini[0].size() == 0) {

for (double x = -N / 2; x <= N / 2; x += 0.05) {

y = sqrt(sqrt(pow(a, 4) + 4 \* c \* c \* x \* x) - x \* x - c \* c);

if (y >= 0) {

OvalKassini[0] << IntPoint(x \* scale\_helper, y \* scale\_helper);

}

}

// рисуем нижнюю часть овала Кассини

for (int i = OvalKassini[0].size() - 1; i >= 0; i--) {

OvalKassini[0] << IntPoint(OvalKassini[0][i].X, -1 \* OvalKassini[0][i].Y);

}

}

// смещаем и поворачиваем (оператор поворота: x' = x\*cos(alpha) - y\*sin(alpha); y'= x\*sin(alpha) + y\*cos(alpha))

RotatedAndMovedOvalKassini[0].clear();

for (int i = 0; i < OvalKassini[0].size(); i++) {

x\_current = (OvalKassini[0][i].X \* cos(AngleRotation.GetPos()\*(PI / 180)) - OvalKassini[0][i].Y \* sin(AngleRotation.GetPos()\*(PI / 180))) +

(XPosition.GetPos() \* scale\_helper / 4);

y\_current = (OvalKassini[0][i].X \* sin(AngleRotation.GetPos()\*(PI / 180)) + OvalKassini[0][i].Y \* cos(AngleRotation.GetPos()\*(PI / 180))) +

(YPosition.GetPos() \* scale\_helper / 4);

RotatedAndMovedOvalKassini[0] << IntPoint(x\_current, y\_current);

}

// TODO: Add your control notification handler code here

}

// площади пересечения элементов сетки и овала Кассини (size = Net[0].size())

vector<double> Areas;

int CoveredBlocksCounter = 0; // считаем, сколько блоков покрывают частично или полностью овал Кассини

// пересекаем овал Кассини и сетку

void CStripsDlg::add\_intersec\_oval\_and\_net()

{

Areas.clear();

CoveredBlocksCounter = 0;

Intersections[0].clear();

for (int i = 0; i < Net[0].size(); i++) { // i - отдельный блок сетки

CurrentIntersection = do\_intersectrion(Net, i, RotatedAndMovedOvalKassini); // пересекли

// проверяем результат пересечения

if (CurrentIntersection.size() > 0) {

CoveredBlocksCounter++;

Intersections[0].push\_back(CurrentIntersection[0]);

Areas.push\_back(Area(CurrentIntersection[0])/pow(scale\_helper, 2));

}

else {

Areas.push\_back(0);

}

}

TextForCtrl.Format(\_T("%d"), CoveredBlocksCounter);

NumOfCoveredBlocks.SetWindowTextW(TextForCtrl);

// TODO: Add your control notification handler code here

}

double HausDist = 0; // указываем расстояние, которое точно меньше минимального

double CurrHausDist = 0;

// считаем Хаусдорфово расстояние

void CStripsDlg::find\_haus\_dist()

{

////////////////////////////временно отложенный подход//////////////////////////////////////////

//// найдем номер блока, у которого минимальная площадь пересечения. Для такого болка будем считать Хаусдорфово расстояние

//int BlockMinAreaNum = 0;

//double MinArea = 1; // 1 - площадь квадрата

//for (int i = 0; i < Areas.size(); i++) {

// if ((Areas[i] > 0) && (Areas[i] < MinArea)) {

// MinArea = Areas[i];

// BlockMinAreaNum = i;

// }

//}

////////////////////////////////////////////////////////////////////////////////////////////////////

// для начала надо определить, лежит ли угол блока в овале Кассини. Для этого преобразуем точки в маааленький треугольник

// и пересечем с овалом Кассини

for (int i = 0; i < Areas.size(); i++) { // i - это блоки сетки

if ((Areas[i] != 0) && (Areas[i] != 1)) { // 0 - не пересекается, 1 (площадь клетки) - лежит внутри овала Кассини

DotsOfCurrBlock = make\_polys\_for\_dots(Net[0], i); // превратили точки в маааленькие треугольники

// проверяем, лежит ли маааленький треугольик в овале Кассини. Если лежит, то мы его игнорируем

// заменить на функции типа "сделать\_пересечение", вернуть размер CurrentIntersection

for (int j = 0; j < DotsOfCurrBlock[0].size(); j++) { // j - это точки частично покрывающего блока

CurrentIntersection = do\_intersectrion(DotsOfCurrBlock, j, RotatedAndMovedOvalKassini);

if (CurrentIntersection.size() <= 0) { // иначе игнорируем

CurrHausDist = count\_nearest\_distance(Net, i, j);

if (CurrHausDist > HausDist) {

HausDist = CurrHausDist; // узнали расстояние

HausBlock = i; // в каком оно блоке

HausDot = j; // и от какой точки

XOvalHaus = XOvalHausPrev;

YOvalHaus = YOvalHausPrev;

}

}

}

}

}

HausDist /= scale\_helper;

TextForCtrl.Format(\_T("%.2f"), HausDist);

HausdorffDistance.SetWindowTextW(TextForCtrl);

// TODO: Add your control notification handler code here

}

// ищем минмальное Хаусдорфово расстояние

void CStripsDlg::do\_check\_all\_positions()

{

int MinHausDist = 20 \* scale\_helper;

// зафиксируем, в каком случае достигается минимальное значение Хаусдорфова расстояния

int MinXPos = 0;

int MinYPos = 0;

int MinAnglePos = 0;

add\_net();

for (int i = XPosition.GetRangeMin(); i <= XPosition.GetRangeMax(); i++) {

XPosition.SetPos(i); // один фиг: он не заполнит edit control-ы, пока циклы не закончатся

for (int j = YPosition.GetRangeMin(); j <= YPosition.GetRangeMax(); j++) {

YPosition.SetPos(j);

for (int k = AngleRotation.GetRangeMin(); k <= 90; k += 10) {

AngleRotation.SetPos(k);

add\_oval();

add\_intersec\_oval\_and\_net();

find\_haus\_dist();

if (HausDist < MinHausDist) {

MinHausDist = HausDist;

MinXPos = XPosition.GetPos();

MinYPos = YPosition.GetPos();

MinAnglePos = AngleRotation.GetPos();

}

}

}

}

MinHausDist /= scale\_helper;

TextForCtrl.Format(\_T("%.2f"), HausDist);

HausdorffDistance.SetWindowTextW(TextForCtrl);

// установим подходящие условия

set\_slider(XPosition, MinXPos, 4, CurrXPos);

set\_slider(YPosition, MinYPos, 4, CurrYPos);

set\_slider(AngleRotation, MinAnglePos, 1, CurrAnglePos);

draw\_everything();

// TODO: Add your control notification handler code here

}

// ищем кратчайшее расстояния для каждой вершины покрывающего блока сетки

double MinDistance = 20 \* scale\_helper; // указываем расстяние, которое точно больше максимального

double CurrentDistance = 20 \* scale\_helper;

double CStripsDlg::count\_nearest\_distance(std::vector<Paths> net\_block, int num\_of\_block, int num\_of\_dot)

{

MinDistance = 20 \* scale\_helper;

CurrentDistance = 20 \* scale\_helper;

for (int i = 0; i < OvalKassini[0].size(); i++) { // i - точки оала Кассини

CurrentDistance = sqrt(pow((net\_block[0][num\_of\_block][num\_of\_dot].X - RotatedAndMovedOvalKassini[0][i].X), 2) +

pow((net\_block[0][num\_of\_block][num\_of\_dot].Y - RotatedAndMovedOvalKassini[0][i].Y), 2));

if (CurrentDistance < MinDistance) {

MinDistance = CurrentDistance;

XOvalHausPrev = RotatedAndMovedOvalKassini[0][i].X;

YOvalHausPrev = RotatedAndMovedOvalKassini[0][i].Y;

}

}

return MinDistance;

}

// общая функция пересечения

std::vector<Path> CStripsDlg::do\_intersectrion(std::vector<Paths> who\_clip, int num\_of\_path, std::vector<Path> who\_clipped)

{

cp.Clear();

CurrInts.clear();

cp.AddPath(who\_clip[0][num\_of\_path], ptSubject, true);

cp.AddPath(who\_clipped[0], ptClip, true);

cp.Execute(ctIntersection, CurrInts, pftNonZero, pftNonZero);

return CurrInts;

}

// реагируем на движение слайдера

void CStripsDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar\* pScrollBar)

{

CSliderCtrl \*pSlider = reinterpret\_cast<CSliderCtrl\*>(pScrollBar);

if (pSlider == &XPosition || pSlider == &YPosition || pSlider == &AngleRotation) {

add\_oval();

}

draw\_everything();

// TODO: Add your message handler code here and/or call default

CDialogEx::OnHScroll(nSBCode, nPos, pScrollBar);

}

// рисуем все, что создали

void CStripsDlg::draw\_everything()

{

InvalidateRect(RedrawArea, 1);

OnPaint();

// TODO: Add your control notification handler code here

}

// настраиваем слайдер и поле начения для него

void CStripsDlg::set\_slider(CSliderCtrl& slider, int position, int divide\_position, CEdit& value)

{

slider.SetPos(position);

TextForCtrl.Format(\_T("%.2f"), position / divide\_position);

value.SetWindowTextW(TextForCtrl);

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "pch.h"

#include "IMMDraw.h"

CPen blackpan(PS\_SOLID, 1, RGB(0, 0, 0));

CPen bluepan(PS\_SOLID, 1, RGB(0, 0, 255));

CPen redpan(PS\_SOLID, 1, RGB(255, 0, 0));

int center\_window\_x = 0;

int center\_window\_y = 0;

int scale\_param = 1;

int scale\_helper\_param = 1;

void draw\_object(std::vector<Paths> object, CString tool, CString color, CString regime, CPaintDC dc)

{

if (color == "Black")

{

dc.SelectObject(blackpan);

}

//StructureForDraw = make\_structure\_for\_draw(StructureForDraw, object);

//for (int i = 0; i < object[0][i].size(); i++) {

// dc.Polygon(StructureForDraw[i], object[0][i].size());

//}

}

POINT \* make\_structure\_for\_draw(POINT \* structure, std::vector<Path> object)

{

structure = new POINT[object[0].size()];

for (int i = 0; i < object[0].size(); i++) {

structure[i].x = center\_window\_x + (scale\_param \* object[0][i].X) / scale\_helper\_param;

structure[i].y = center\_window\_y - (scale\_param \* object[0][i].Y) / scale\_helper\_param;

}

return structure;

}

POINT \*\* make\_structure\_for\_draw(POINT \*\* structure, std::vector<Paths> object)

{

delete[] structure;

structure = new POINT\*[object[0].size()];

for (int i = 0; i < object[0].size(); i++) {

structure[i] = new POINT[object[0][i].size()];

}

for (int i = 0; i < object[0].size(); i++) {

for (int j = 0; j < object[0][i].size(); j++) {

structure[i][j].x = center\_window\_x + (scale\_param \* object[0][i][j].X) / scale\_helper\_param;

structure[i][j].y = center\_window\_y - (scale\_param \* object[0][i][j].Y) / scale\_helper\_param;

}

}

return structure;

}

void set\_drawing\_param(int center\_x, int center\_y, int scale\_value, int scale\_helper\_value)

{

center\_window\_x = center\_x;

center\_window\_y = center\_y;

scale\_param = scale\_value;

scale\_helper\_param = scale\_helper\_value;

}

std::vector<Paths> make\_polys\_for\_dots(std::vector<Path> paths, int num\_of\_path)

{

std::vector<Paths> PolyFromDot(1);

std::vector<Path> NewDot(1);

double x = 0;

double y = 0;

for (int i = 0; i < paths[num\_of\_path].size(); i++) {

NewDot[0].clear();

for (int j = 0; j < 3; j++) {

x = paths[num\_of\_path][i].X + (cos((90 - (120 \* j))\*PI / 180)) \* (0.01 \* scale\_helper\_param); /////////////////

y = paths[num\_of\_path][i].Y - (sin((90 - (120 \* j))\*PI / 180)) \* (0.01 \* scale\_helper\_param); /////////////////

NewDot[0] << IntPoint(x, y);

}

PolyFromDot[0].push\_back(NewDot[0]);

}

return PolyFromDot;

}