# 实验二 《二维、三维图形变换》

1. **三角形的绘制**

第1步：在菜单中添加actiondrawtri

第2步：在cpp文件中添加以下代码：

connect(ui.actiondrawtri, SIGNAL(triggered()), this, SLOT(DrawTri()));

第3步：定义外形类Gsharp

Gsharp.h

#ifndef \_GSHARP\_H

#define \_GSHARP\_H

struct Rectangle

{

int xmin, xmax;

int ymin, ymax;

};

class GPoint

{

public:

GPoint::GPoint()

{

}

GPoint::GPoint(int a, int b)

{

x = a;

y = b;

}

GPoint::~GPoint()

{

}

int x;

int y;

int getX()const { return x; } //读x坐标

int getY()const { return y; } //读y坐标

GPoint& operator=(const GPoint& d)

{

if (this != &d)

{

x = d.x;

y = d.y;

}

return \*this;

}

};

#endif // !\_GSHARP\_H

第4步：定义三角形的三个顶点

GPoint tripoint[3];

第5步：初始化三角形的三个顶点

tripoint[0].x = 0;

tripoint[0].y = 0;

tripoint[1].x = 200;

tripoint[1].y = 300;

tripoint[2].x = 150;

tripoint[2].y = 250;

第6步：头文件中定义状态控制变量

int state;

在构造函数中初始化

testa::testa(QWidget \*parent)

: QMainWindow(parent)

{

ui.setupUi(this);

connect(ui.actiondrawtri, SIGNAL(triggered()), this, SLOT(DrawTri()));

state = 0；

}

第7步：头文件中添加绘图指针

QPainter \*paint;

第8步：添加绘图类

GraphAlgorithm.h

#ifndef \_GRAPHALGORITHM\_H

#define \_GRAPHALGORITHM\_H

#include <QPainter>

void BresenhamF(QImage \*image, int x1, int y1, int x2, int y2, QRgb value);

#endif

GraphAlgorithm.cpp

#include "GraphAlgorithm.h"

void BresenhamF(QImage \*image, int x1, int y1, int x2, int y2, QRgb value)

{

if (x2 < x1)

{

int temp;

//交换起始点

temp = x2, x2 = x1, x1 = temp;

temp = y2, y2 = y1, y1 = temp;

}

if (y1 == y2)

{

for (int i = x1; i <= x2; i++)

{

image->setPixel(i, y1, value);

}

return;

}

if (x1 == x2)

{

for (int i = y1; i <= y2; i++)

{

image->setPixel(x1, i, value);

}

return;

}

int x, y, dx, dy;

float k, e;

dx = x2 - x1;

dy = y2 - y1;

if (abs(dx) > abs(dy))

{

e = 2 \* dy - dx;

x = x1; y = y1;

image->setPixel(x, y, value);

for (int i = 1; i <= dx; i++)

{

x++;

if (e > 0)

{

e = e - 2 \* dx;

y++;

}

image->setPixel(x, y, value);

e = e + 2 \* dy;

}

}

else

{

e = 2 \* dx - dy;

x = x1; y = y1;

image->setPixel(x, y, value);

for (int i = 1; i <= dy; i++)

{

y++;

if (e > 0)

{

e = e - 2 \* dy;

x++;

}

image->setPixel(x, y, value);

e = e + 2 \* dx;

}

}

}

第9步：添加绘制三角形函数

GraphAlgorithm.h

void DrawTriangle(QImage \*image, GPoint apoint, GPoint bpoint, GPoint cpoint, QRgb value);

GraphAlgorithm.cpp

void DrawTriangle(QImage \*image, GPoint apoint, GPoint bpoint, GPoint cpoint, QRgb value)

{

BresenhamF(image, apoint.x, apoint.y, bpoint.x, bpoint.y, value);

BresenhamF(image, cpoint.x, cpoint.y, bpoint.x, bpoint.y, value);

BresenhamF(image, apoint.x, apoint.y, cpoint.x, cpoint.y, value);

}

在头文件中引用

#include "GraphAlgorithm.h"

第10步：添加绘图事件

在头文件中添加

void paintEvent(QPaintEvent\*);

在源文件中添加

void testa::paintEvent(QPaintEvent\*)

//The paintevent function is automatically called by the system, so we don't need to call it manually.

{

QImage image(800, 600, QImage::Format\_RGB32);

paint = new QPainter;

paint->begin(this);

paint->drawImage(0, 0, image);

paint->end();

}

第11步：添加绘图响应菜单事件

在头文件中添加

public slots:

void DrawTri();//绘制三角形

在源文件中添加

void testa::DrawTri()

{

state = 1;

repaint();

}

第12步：实现绘图事件

void testa::paintEvent(QPaintEvent\*)

//The paintevent function is automatically called by the system, so we don't need to call it manually.

{

QImage image(800, 600, QImage::Format\_RGB32);

paint = new QPainter;

paint->begin(this);

switch (state)

{

case 1:

DrawTriangle(&image, tripoint[0], tripoint[1], tripoint[2], qRgb(0, 255, 255));

break;

default:

break;

}

paint->drawImage(0, 0, image);

paint->end();

}

第13步：为三角形赋值

tripoint[0].x = 0;

tripoint[0].y = 0;

tripoint[1].x = 200;

tripoint[1].y = 300;

tripoint[2].x = 150;

tripoint[2].y = 250;

第14步：执行程序

1. **三角形的平移**

第1步：在菜单中添加actionTranslation

第2步：在cpp文件中添加以下代码：

connect(ui. actionTranslation, SIGNAL(triggered()), this, SLOT(DrawTri()));

第3步： 添加绘图响应菜单事件

在头文件中添加

public slots:

void Translation();//平移

在源文件中添加

void testa::Translation()

{

state = 2;

repaint();

}

第4步：添加矩阵类

**Gmatrix.h**

#ifndef \_GMATRIX\_H

#define \_GMATRIX\_H

#include <iostream>

#include <cmath>

using namespace std;

class GMatrix

{

public:

GMatrix(int r = 2, int c = 2);

GMatrix(GMatrix &m);

~GMatrix();

void GetValue(void);

void SetTranslationThree(void);

void SetRotateThree(void);

void SetScaleThree(void);

void SetoneThree(int a, int b);

int GetoneThreea();

int GetoneThreeb();

GMatrix operator =(GMatrix &m);

GMatrix operator +(GMatrix &m);

GMatrix operator -(GMatrix &m);

GMatrix operator \*(GMatrix &m);

double operator ()(int x, int y);

GMatrix rev(void);

friend ostream & operator <<(ostream & out, GMatrix &m);

private:

int row;

int col;

double \*\*p;

};

#endif // !

**Gmatrix.cpp**

#include "GMatrix.h"

GMatrix::GMatrix(int r, int c)

{

int i;

row = r;

col = c;

p = new double\*[row];

for (i = 0; i < row; i++)

\*(p + i) = new double[col];

}

GMatrix::GMatrix(GMatrix &m)

{

int i, j;

row = m.row;

col = m.col;

p = new double\*[row];

for (i = 0; i < row; i++)

\*(p + i) = new double[col];

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

\*(\*(p + i) + j) = \*(\*(m.p + i) + j);

}

GMatrix::~GMatrix()

{

int i;

for (i = 0; i < row; i++)

delete[] \* (p + i);

delete[]p;

}

void GMatrix::GetValue(void)

{

int i, j;

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

cin >> \*(\*(p + i) + j);

}

void GMatrix::SetTranslationThree(void)

{

p[0][0] = 1;

p[0][1] = 0;

p[0][2] = 0;

p[1][0] = 0;

p[1][1] = 1;

p[1][2] = 0;

p[2][0] = 50;

p[2][1] = 200;

p[2][2] = 1;

}

void GMatrix::SetScaleThree(void)

{

p[0][0] = 0.5;

p[0][1] = 0;

p[0][2] = 0;

p[1][0] = 0;

p[1][1] = 0.5;

p[1][2] = 0;

p[2][0] = 0;

p[2][1] = 0;

p[2][2] = 1;

}

void GMatrix::SetRotateThree(void)

{

p[0][0] = cos(3.1415926/12);

p[0][1] = sin(3.1415926 / 12);

p[0][2] = 0;

p[1][0] = -sin(3.1415926 / 12);

p[1][1] = cos(3.1415926 / 12);

p[1][2] = 0;

p[2][0] = 0;

p[2][1] = 0;

p[2][2] = 1;

}

void GMatrix::SetoneThree(int a, int b)

{

p[0][0] = a;

p[0][1] = b;

p[0][2] = 1;

}

int GMatrix::GetoneThreea()

{

return p[0][0];

}

int GMatrix::GetoneThreeb()

{

return p[0][1];

}

GMatrix GMatrix::operator =(GMatrix &m)

{

int i, j;

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

\*(\*(p + i) + j) = \*(\*(m.p + i) + j);

return(\*this);

}

GMatrix GMatrix:: operator +(GMatrix &m)

{

int i, j;

GMatrix a(row, col);

if (row != m.row || col != m.col)

throw 0;

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

\*(\*(a.p + i) + j) = \*(\*(p + i) + j) + \*(\*(m.p + i) + j);

return(a);

}

GMatrix GMatrix::operator -(GMatrix &m)

{

int i, j;

GMatrix a(row, col);

if (row != m.row || col != m.col)

throw 0;

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

\*(\*(a.p + i) + j) = \*(\*(p + i) + j) - \*(\*(m.p + i) + j);

return(a);

}

GMatrix GMatrix::operator \*(GMatrix &m)

{

if (col != m.row)

throw 0;

int i, j, k;

double sum;

GMatrix a(row, m.col);

for (i = 0; i < row; i++)

for (j = 0; j < m.col; j++)

{

for (k = 0, sum = 0; k < col; k++)

sum = sum + \*(\*(p + i) + k) \* \*(\*(m.p + k) + j);

\*(\*(a.p + i) + j) = sum;

}

return(a);

}

double GMatrix::operator ()(int x, int y)

{

if (x > row || y > col)

throw 0.0;

return(\*(\*(p + x - 1) + y - 1));

}

GMatrix GMatrix::rev(void)

{

int i, j;

GMatrix a(col, row);

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

\*(\*(a.p + j) + i) = \*(\*(p + i) + j);

return(a);

}

ostream & operator <<(ostream & out, GMatrix &m)

{

int i, j;

for (i = 0; i < m.row; i++)

{

for (j = 0; j < m.col; j++)

out << \*(\*(m.p + i) + j) << " ";

out << endl;

}

return(out);

}

第5步：添加平移函数

GraphAlgorithm.h

#include "GMatrix.h"

void Trans(QImage \*image, GPoint apoint, GPoint bpoint, GPoint cpoint, QRgb value);

GraphAlgorithm.cpp

void Trans(QImage \*image, GPoint apoint, GPoint bpoint, GPoint cpoint, QRgb value)

{

GMatrix ThreeThreeMatrix(3, 3), ap(1, 3), bp(1, 3), cp(1, 3);

GMatrix moveap(1, 3), movebp(1, 3), movecp(1, 3);

ThreeThreeMatrix.SetTranslationThree();

ap.SetoneThree(apoint.getX(), apoint.getY());

bp.SetoneThree(bpoint.getX(), bpoint.getY());

cp.SetoneThree(cpoint.getX(), cpoint.getY());

moveap = ap\*ThreeThreeMatrix;

movebp = bp\*ThreeThreeMatrix;

movecp = cp\*ThreeThreeMatrix;

GPoint Apoint(moveap.GetoneThreea(), moveap.GetoneThreeb());

GPoint Bpoint(movebp.GetoneThreea(), movebp.GetoneThreeb());

GPoint Cpoint(movecp.GetoneThreea(), movecp.GetoneThreeb());

BresenhamF(image, Apoint.x, Apoint.y, Bpoint.x, Bpoint.y, value);

BresenhamF(image, Cpoint.x, Cpoint.y, Bpoint.x, Bpoint.y, value);

BresenhamF(image, Apoint.x, Apoint.y, Cpoint.x, Cpoint.y, value);

}

第6步：添加绘图事件

在源文件中添加

void testa::paintEvent(QPaintEvent\*)

//The paintevent function is automatically called by the system, so we don't need to call it manually.

{

QImage image(800, 600, QImage::Format\_RGB32);

paint = new QPainter;

paint->begin(this);

switch (state)

{

case 1:

DrawTriangle(&image, tripoint[0], tripoint[1], tripoint[2], qRgb(0, 255, 255));

break;

case 2:

Trans(&image, tripoint[0], tripoint[1], tripoint[2], qRgb(0, 255, 255));

break;

default:

break;

}

paint->drawImage(0, 0, image);

paint->end();

}

1. **课后作业：**

1）实现三角形的旋转

2）实现三角形的缩放