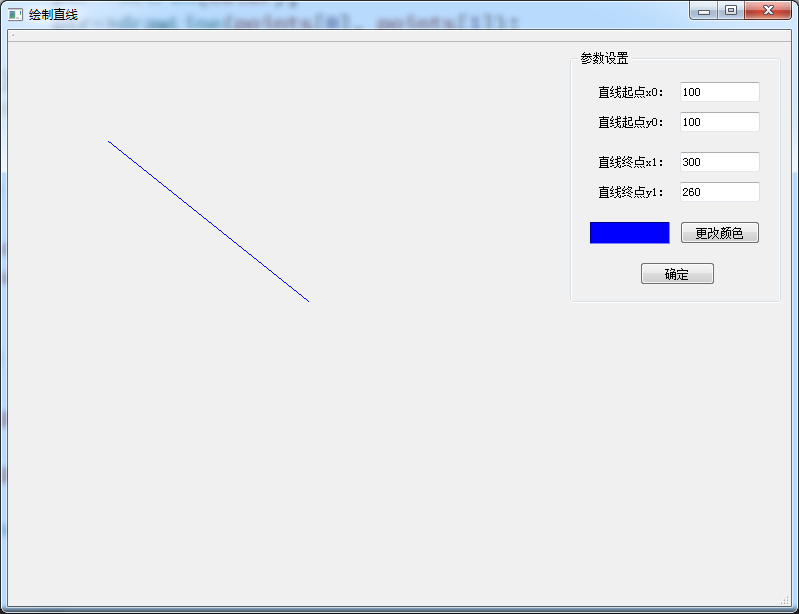
## 实验1 直线的绘制

一、要求：绘制一条直线，界面如图



二、步骤：

1. 启动Qt，选择New Project

1. 选择”Application” - “Qt Widgets Application” - “Choose”
2. 输入项目名称”cgExperiment01DrawLine”，选择路径，设为默认项目路径。注意：Qt不支持中文路径。点击“下一步”。
3. 选择库函数等配套组件kits为Desktop Qt 5.11.1 MinGW 32bit，点击“下一步”。
4. 选择主窗口MainWindow模式，保留“创建界面”，点击“下一步”。
5. 点击“完成”。
6. 双击“mainwindow.ui”，打开主界面，修改主界面属性
7. 将主窗口尺寸拖动到合适大小
8. 修改属性windowtitle为“绘制直线”。
9. 向主界面中拖入一个Group Box控件，对象名称ObjectName为“groupBox\_set”，标题Title设为“参数设置”，尺寸可以拖动到合适大小
10. 向“groupBox\_set”中拖入一个Label控件“label\_x0”，标题Title设为“直线起点x0: ”，尺寸为：宽度80，高度20，右对齐
11. 向“groupBox\_set”中拖入一个Line Edit控件“lineEdit\_x0”，尺寸为：宽度80，高度20，初始值设为0
12. 按住Ctrl键，用鼠标选中label\_x0和lineEdit\_x0，然后复制、粘贴，产生一对复本，将其拖曳到合适位置，修改其属性，分别为label\_y0和lineEdit\_y0
13. 用复制粘贴的办法产生直线终点控件label\_x1，lineEdit\_x1，label\_y1，和lineEdit\_y1
14. 向“groupBox\_color”中拖入一个Frame控件“frame\_color”，尺寸设置为合适大小，frameShape设为“Panel”，frameShadow设为“Sunken”，自动填充背景autoFillBackgroud选项选中，调色板palette设置背景为蓝色
15. 向“groupBox\_color”中拖入一个PushBotton控件“pushButton\_color”，标题Title设为“更改颜色”，尺寸设置为合适大小
16. 向“groupBox\_set”中拖入一个Push Botton控件“pushButton\_ok”，标题Title设为“确定”，尺寸自定义
17. 打开mainwindow.h头文件，添加

#include <QPainter>

#include <QRect>

1. 在mainwindow.h头文件中添加

public:

void ***paintEvent***(QPaintEvent\*);

1. 在mainwindow.cpp文件中添加

void MainWindow::***paintEvent***(QPaintEvent\*)

{

int w = ui->groupBox\_set->width();

int h = ui->groupBox\_set->height();

int x = width()-w-10;

int y = ui->groupBox\_set->y();

ui->groupBox\_set->setGeometry(QRect(x, y, w, h));

}

1. 在mainwindow.h头文件中添加

#include <QPoint>

#include <QVector>

#include <QColor>

#include <QColorDialog>

1. 在mainwindow.h头文件中添加

private:

QVector<QPoint> points;

QColor color;

1. 打开mainwindow.cpp文件，在构造函数中增加如下代码：

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

points.append(QPoint(ui->lineEdit\_x0->text().toInt(),

ui->lineEdit\_y0->text().toInt()+

ui->mainToolBar->height()));

points.append(QPoint(ui->lineEdit\_x1->text().toInt(),

ui->lineEdit\_y1->text().toInt()+

ui->mainToolBar->height()));

color = ui->frame\_color->palette().background().color();

}

1. 打开ui，为按钮“更改颜色”、“确定”添加槽函数，右键单击按钮即可添加（在头文件里）

private slots:

void **on\_pushButton\_color\_clicked**();

void **on\_pushButton\_ok\_clicked**();

1. 打开mainwindow.cpp文件，完成槽函数代码

void MainWindow::**on\_pushButton\_color\_clicked**()

{

color = QColorDialog::getColor();

ui->frame\_color->setPalette(QPalette(color));

}

void MainWindow::**on\_pushButton\_ok\_clicked**()

{

points[0] = QPoint(ui->lineEdit\_x0->text().toInt(),

ui->lineEdit\_y0->text().toInt()+

ui->mainToolBar->height());

points[1] = QPoint(ui->lineEdit\_x1->text().toInt(),

ui->lineEdit\_y1->text().toInt()+

ui->mainToolBar->height());

update();

}

1. 在***paintEvent***函数中增加绘制代码

QPainter\* ptr = new QPainter(this);

if(points.size()>=2)

{

ptr->save();

ptr->setPen(color);

ptr->drawLine(points[0], points[1]);

ptr->restore();

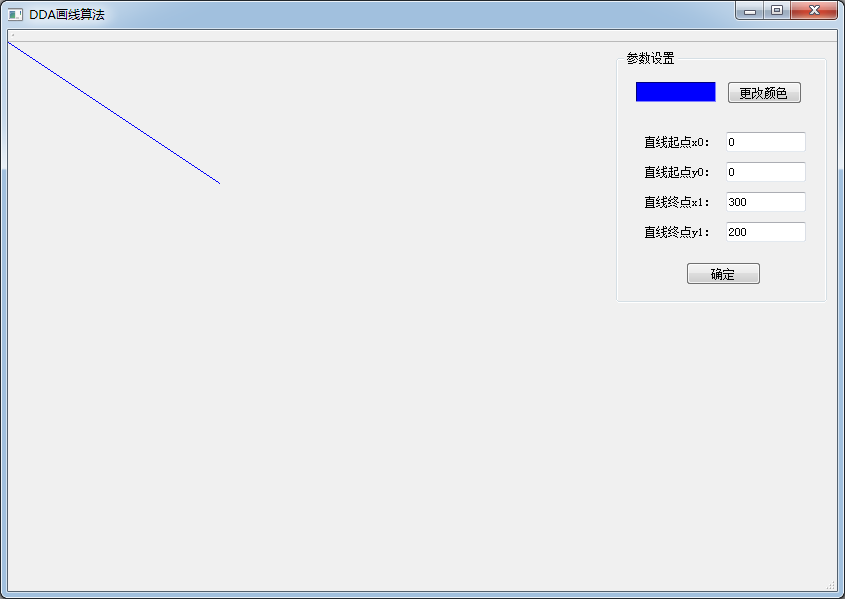
}

delete ptr;

1. 运行程序

## 实验2 直线的DDA生成算法

一、要求：用DDA算法绘制一条直线，界面同实验1。



1. 算法：
2. 可以将上一个实验的mainwindow.ui文件复制到本项目中来，覆盖本项目的ui文件，这样可以简化界面设计
3. 创建一个直线类line，添加如下代码

#ifndef LINE\_H

#define LINE\_H

#include <QPoint>

#include <QColor>

#include <QPainter>

class line

{

private:

QPoint p0;

QPoint p1;

QColor color;

public:

line(QPoint p0, QPoint p1, QColor color);

void **showLineInDDA**(QPainter\* ptr);

};

#endif // LINE\_H

#include "line.h"

line::**line**(QPoint p0, QPoint p1, QColor color)

:p0(p0), p1(p1), color(color)

{

}

void line::**showLineInDDA**(QPainter\* ptr)

{

int x, y;

double m, tx, ty;

QPoint temp;

ptr->save();

ptr->setPen(color);

if(p0.y() == p1.y())//水平边

{

if(p0.x()>p1.x())

{

temp = p0;

p0 = p1;

p1 = temp;

}

for(x=p0.x(); x<p1.x(); x++)

ptr->drawPoint(x, p0.y());

}

else if(p0.x() == p1.y())//垂直边

{

if(p0.y()>p1.y())

{

temp = p0;

p0 = p1;

p1 = temp;

}

for(y=p0.y(); y<p1.y(); y++)

ptr->drawPoint(p0.x(), y);

}

else

{

m = (double)(p1.y()-p0.y())/(double)(p1.x()-p0.x());

if(m>-1 && m<1)//斜率在-45°到45°之间

{

if(p0.x()>p1.x())

{

temp = p0;

p0 = p1;

p1 = temp;

}

ty = p0.y();

for(x=p0.x(); x<p1.y(); x++)

{

y = (int)(ty+0.5);

ptr->drawPoint(x, y);

ty += m;

}

}

else//斜率超过45°

{

if(p0.y()>p1.y())

{

temp = p0;

p0 = p1;

p1 = temp;

}

tx = p0.x();

for(y=p0.y(); y<p1.y(); y++)

{

x = (int)(tx+0.5);

ptr->drawPoint(x, y);

tx += 1/m;

}

}

}

ptr->restore();

}

1. 在主窗口程序添加如下代码

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPoint>

#include <QRect>

#include <QVector>

#include <QColor>

#include <QColorDialog>

#include <QPainter>

#include "line.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

void ***paintEvent***(QPaintEvent\*);

private slots:

void **on\_pushButton\_color\_clicked**();

void **on\_pushButton\_ok\_clicked**();

private:

Ui::MainWindow \*ui;

QVector<QPoint> points;

QColor color;

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

points.append(QPoint(ui->lineEdit\_x0->text().toInt(),

ui->lineEdit\_y0->text().toInt()+

ui->mainToolBar->height()));

points.append(QPoint(ui->lineEdit\_x1->text().toInt(),

ui->lineEdit\_y1->text().toInt()+

ui->mainToolBar->height()));

color = ui->frame\_color->palette().background().color();

}

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

{

delete ui;

}

void MainWindow::***paintEvent***(QPaintEvent\*)

{

int w = ui->groupBox\_set->width();

int h = ui->groupBox\_set->height();

int x = width()-w-10;

int y = ui->groupBox\_set->y();

ui->groupBox\_set->setGeometry(QRect(x, y, w, h));

QPainter\* ptr = new QPainter(this);

ptr->save();

ptr->setPen(color);

line l(points[0], points[1], color);

l.showLineInDDA(ptr);

ptr->restore();

delete ptr;

}

void MainWindow::**on\_pushButton\_color\_clicked**()

{

color = QColorDialog::getColor();

ui->frame\_color->setPalette(QPalette(color));

}

void MainWindow::**on\_pushButton\_ok\_clicked**()

{

points[0].setX(ui->lineEdit\_x0->text().toInt());

points[0].setY(ui->lineEdit\_y0->text().toInt()+

ui->mainToolBar->height());

points[1].setX(ui->lineEdit\_x1->text().toInt());

points[1].setY(ui->lineEdit\_y1->text().toInt()+

ui->mainToolBar->height());

update();

}

## 实验3 直线中点生成算法

一、要求：用中点算法绘制一条直线，界面同实验2。

二、算法：

1. 创建项目名称为cgExperiment03MidpointDrawLine

2. 将cgExperiment02DdaDrawLine项目中的mainwindow.ui复制到本项目中来，可以简化界面设计

3. 创建直线类line

#ifndef LINE\_H

#define LINE\_H

#include <QPoint>

#include <QColor>

#include <QPainter>

class line

{

private:

QPoint p0;

QPoint p1;

QColor color;

public:

line(QPoint p0, QPoint p1, QColor color);

void **showLineInMidpoint**(QPainter\* ptr);

};

#endif // LINE\_H

#include "line.h"

line::**line**(QPoint p0, QPoint p1, QColor color)

:p0(p0), p1(p1), color(color)

{

}

void line::**showLineInMidpoint**(QPainter\* ptr)

{

QPoint temp;

double m;//斜率

int d, dx, dy, dNE, dE, x, y;

if(p0.y() == p1.y())//水平线

{

if(p0.x()>p1.x())

{

temp = p0;

p0 = p1;

p1 = temp;

}

ptr->save();

ptr->setPen(color);

for(x=p0.x(); x<p1.x(); x++)

ptr->drawPoint(x, p0.y());

ptr->restore();

return;

}

if(p0.x() == p1.x())//垂直线

{

if(p0.y()>p1.y())

{

temp = p0;

p0 = p1;

p1 = temp;

}

ptr->save();

ptr->setPen(color);

for(y=p0.y(); y<p1.y(); y++)

ptr->drawPoint(p0.x(), y);

ptr->restore();

return;

}

ptr->save();

ptr->setPen(color);

m = (double)(p1.y()-p0.y())/(double)(p1.x()-p0.x());

if(m>0 && m<=1)//斜率在0°到45°之间,每次x自增1，y根据dp判别式增加

{

if(p0.x()>p1.x())

{

temp = p0;

p0 = p1;

p1 = temp;

}

dx = p1.x() - p0.x();//delt x

dy = p1.y() - p0.y();//delt y

d = dx - 2\*dy; //d0的判别式

dNE = 2\*dx - 2\*dy; //dp<=0时的d(p+1)递推式

dE = -2\*dy; //dp>0时的d(p+1)递推式

x = p0.x();

y = p0.y();

while(x<p1.x())

{

ptr->drawPoint(x, y);

if(d<=0)

{

d+=dNE;

y++;

}

else

d+=dE;

x++;

}

}

else if(m>=-1 && m<0)//斜率在-45°到0°之间,每次x自增1

{

if(p0.x()>p1.x())

{

temp = p0;

p0 = p1;

p1 = temp;

}

dx = p1.x() - p0.x();//delt x

dy = p1.y() - p0.y();//delt y

d = -1\*dx - 2\*dy; //d0的判别式

dNE = -2\*dx - 2\*dy; //dp>=0时的d(p+1)递推式

dE = -2\*dy; //dp<0时的d(p+1)递推式

x = p0.x();

y = p0.y();

while(x<p1.x())

{

ptr->drawPoint(x, y);

if(d>=0)

{

d+=dNE;

y--;

}

else

d+=dE;

x++;

}

}

else if(m>1)//斜率超过45°，每次y自增1，x根据判别式增加

{

if(p0.y()>p1.y())

{

temp = p0;

p0 = p1;

p1 = temp;

}

dx = p1.x() - p0.x();

dy = p1.y() - p0.y();

d = dy - 2\*dx;

dNE = 2\*dy - 2\*dx;

dE = -2\*dx;

x = p0.x();

y = p0.y();

while(y<p1.y())

{

ptr->drawPoint(x, y);

if(d<=0)

{

d+=dNE;

x++;

}

else

d+=dE;

y++;

}

}

else//斜率小于-45°，每次y自减1，x根据判别式增加

{

if(p0.y()>p1.y())

{

temp = p0;

p0 = p1;

p1 = temp;

}

dx = p1.x() - p0.x();//delt x

dy = p1.y() - p0.y();//delt y

d = -1\*dy - 2\*dx; //d0的判别式

dNE = -2\*dy - 2\*dx; //dp>=0时的d(p+1)递推式

dE = -2\*dx; //dp<0时的d(p+1)递推式

x = p0.x();

y = p0.y();

while(y<p1.y())

{

ptr->drawPoint(x, y);

if(d>=0)

{

d+=dNE;

x--;

}

else

d+=dE;

y++;

}

}

ptr->restore();

}

1. 在主窗口程序中添加如下代码

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPoint>

#include <QRect>

#include <QVector>

#include <QColor>

#include <QColorDialog>

#include <QPainter>

#include "line.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

void ***paintEvent***(QPaintEvent\*);

private slots:

void **on\_pushButton\_color\_clicked**();

void **on\_pushButton\_ok\_clicked**();

private:

Ui::MainWindow \*ui;

QVector<QPoint> points;

QColor color;

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

points.append(QPoint(ui->lineEdit\_x0->text().toInt(),

ui->lineEdit\_y0->text().toInt()+

ui->mainToolBar->height()));

points.append(QPoint(ui->lineEdit\_x1->text().toInt(),

ui->lineEdit\_y1->text().toInt()+

ui->mainToolBar->height()));

color = ui->frame\_color->palette().background().color();

}

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

{

delete ui;

}

void MainWindow::**on\_pushButton\_color\_clicked**()

{

color = QColorDialog::getColor();

ui->frame\_color->setPalette(QPalette(color));

}

void MainWindow::**on\_pushButton\_ok\_clicked**()

{

points[0].setX(ui->lineEdit\_x0->text().toInt());

points[0].setY(ui->lineEdit\_y0->text().toInt()+

ui->mainToolBar->height());

points[1].setX(ui->lineEdit\_x1->text().toInt());

points[1].setY(ui->lineEdit\_y1->text().toInt()+

ui->mainToolBar->height());

update();

}

void MainWindow::***paintEvent***(QPaintEvent\*)

{

int w = ui->groupBox\_set->width();

int h = ui->groupBox\_set->height();

int x = width()-w-10;

int y = ui->groupBox\_set->y();

ui->groupBox\_set->setGeometry(QRect(x, y, w, h));

QPainter\* ptr = new QPainter(this);

ptr->save();

ptr->setPen(color);

line l(points[0], points[1], color);

l.showLineInMidpoint(ptr);

ptr->restore();

delete ptr;

}

## 实验4 直线Bresenham生成算法

一、要求：用Bresenham算法绘制一条直线，界面同实验3

二、算法：

1. 创建项目cgExperiment04BresenhamDrawLine

2. 将前面实验3的ui文件复制到本项目

3. 创建直线类line,也可以复制前面的line类文件，加以修改

#ifndef LINE\_H

#define LINE\_H

#include <QPoint>

#include <QColor>

#include <QPainter>

class line

{

private:

QPoint p0;

QPoint p1;

QColor color;

public:

line(QPoint p0, QPoint p1, QColor color);

void **showLineInBresenham**(QPainter\* ptr);

};

#endif // LINE\_H

#include "line.h"

line::**line**(QPoint p0, QPoint p1, QColor color)

:p0(p0), p1(p1), color(color)

{

}

void line::**showLineInBresenham**(QPainter\* ptr)

{

QPoint temp;

double m;

double e = -0.5;

int x, y;

if(p0.y() == p1.y())//水平线

{

if(p0.x()>p1.x())

{

temp = p0;

p0 = p1;

p1 = temp;

}

ptr->save();

ptr->setPen(color);

for(x=p0.x(); x<p1.x(); x++)

ptr->drawPoint(x, p0.y());

ptr->restore();

return;

}

if(p0.x() == p1.x())//垂直线

{

if(p0.y()>p1.y())

{

temp = p0;

p0 = p1;

p1 = temp;

}

ptr->save();

ptr->setPen(color);

for(y=p0.y(); y<p1.y(); y++)

ptr->drawPoint(p0.x(), y);

ptr->restore();

return;

}

m = (double)(p1.y()-p0.y())/(double)(p1.x()-p0.x());

if(m>0 && m<=1)//斜率在-45°到45°之间

{

if(p0.x()>p1.x())

{

temp = p0;

p0 = p1;

p1 = temp;

}

e=-0.5;

x = p0.x();

y = p0.y();

while(x<p1.x())

{

ptr->drawPoint(x, y);

e=e+m;

if(e>0)

{

y++;

e=e-1;

}

x++;

}

}

else if(m>=-1 && m<0)

{

if(p0.x()>p1.x())

{

temp = p0;

p0 = p1;

p1 = temp;

}

e=0.5;

x = p0.x();

y = p0.y();

while(x<p1.x())

{

ptr->drawPoint(x, y);

e=e+m;

if(e<0)

{

y--;

e=e+1;

}

x++;

}

}

else if(m>1)//斜率超过45°，每次y自增1，x根据判别式增加

{

if(p0.y()>p1.y())

{

temp = p0;

p0 = p1;

p1 = temp;

}

e=-0.5;

x = p0.x();

y = p0.y();

while(y<p1.y())

{

ptr->drawPoint(x, y);

e=e+1/m;

if(e>0)

{

x=x+1;

e=e-1;

}

y++;

}

}

else

{

if(p0.y()>p1.y())

{

temp = p0;

p0 = p1;

p1 = temp;

}

e=0.5;

x = p0.x();

y = p0.y();

while(y<p1.y())

{

ptr->drawPoint(x, y);

e=e+1/m;

if(e<0)

{

x--;

e=e+1;

}

y++;

}

}

ptr->restore();

}

4. 修改主窗口文件代码

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPoint>

#include <QRect>

#include <QVector>

#include <QColor>

#include <QColorDialog>

#include <QPainter>

#include "line.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

void ***paintEvent***(QPaintEvent\*);

private slots:

void **on\_pushButton\_color\_clicked**();

void **on\_pushButton\_ok\_clicked**();

private:

Ui::MainWindow \*ui;

QVector<QPoint> points;

QColor color;

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

points.append(QPoint(ui->lineEdit\_x0->text().toInt(),

ui->lineEdit\_y0->text().toInt()+

ui->mainToolBar->height()));

points.append(QPoint(ui->lineEdit\_x1->text().toInt(),

ui->lineEdit\_y1->text().toInt()+

ui->mainToolBar->height()));

color = ui->frame\_color->palette().background().color();

}

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

{

delete ui;

}

void MainWindow::**on\_pushButton\_color\_clicked**()

{

color = QColorDialog::getColor();

ui->frame\_color->setPalette(QPalette(color));

}

void MainWindow::**on\_pushButton\_ok\_clicked**()

{

points[0].setX(ui->lineEdit\_x0->text().toInt());

points[0].setY(ui->lineEdit\_y0->text().toInt()+

ui->mainToolBar->height());

points[1].setX(ui->lineEdit\_x1->text().toInt());

points[1].setY(ui->lineEdit\_y1->text().toInt()+

ui->mainToolBar->height());

update();

}

void MainWindow::***paintEvent***(QPaintEvent\*)

{

int w = ui->groupBox\_set->width();

int h = ui->groupBox\_set->height();

int x = width()-w-10;

int y = ui->groupBox\_set->y();

ui->groupBox\_set->setGeometry(QRect(x, y, w, h));

QPainter\* ptr = new QPainter(this);

ptr->save();

ptr->setPen(color);

line l(points[0], points[1], color);

l.showLineInBresenham(ptr);

ptr->restore();

delete ptr;

}

## 实验5 中点画圆算法

1. 创建一个主窗口应用程序cgExperiment05DrawCircle
2. 建立如图所示ui界面



1. 创建一个圆类circle

#ifndef CIRCLE\_H

#define CIRCLE\_H

#include <QPoint>

#include <QColor>

#include <QPainter>

class circle

{

private:

QPoint p0;

int r;

QColor color;

public:

circle(QPoint p0, int r, QColor color);

void **showCircle**(QPainter\* ptr);

private:

void **draw8th**(QPoint p, QPainter \*ptr);

};

#endif // CIRCLE\_H

#include "circle.h"

circle::**circle**(QPoint p0, int r, QColor color)

:p0(p0)

,r(r)

,color(color)

{

}

void circle::**showCircle**(QPainter\* ptr)

{

if(r<1)

return;

int x, y;

int e=1-r;

x=0;

y=r;

while(x<y)

{

draw8th(QPoint(x, y), ptr);

if(e<=0)

e=e+2\*x+3;

else

{

e=e+2\*x-2\*y+5;

y--;

}

x++;

}

}

void circle::**draw8th**(QPoint p, QPainter \*ptr)

{

ptr->save();

ptr->setPen(color);

ptr->drawPoint(p.x() +p0.x(), p.y()+p0.y());

ptr->drawPoint(p.y() +p0.x(), p.x()+p0.y());

ptr->drawPoint(-p.y()+p0.x(), p.x()+p0.y());

ptr->drawPoint(-p.x()+p0.x(), p.y()+p0.y());

ptr->drawPoint(p.y() +p0.x(),-p.x()+p0.y());

ptr->drawPoint(p.x() +p0.x(),-p.y()+p0.y());

ptr->drawPoint(-p.x()+p0.x(),-p.y()+p0.y());

ptr->drawPoint(-p.y()+p0.x(),-p.x()+p0.y());

ptr->restore();

}

1. 修改主窗口类MainWindow

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPoint>

#include <QColor>

#include <QPainter>

#include <QColorDialog>

#include "circle.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

void ***paintEvent***(QPaintEvent\*);

private slots:

void **on\_pushButton\_color\_clicked**();

void **on\_pushButton\_ok\_clicked**();

private:

Ui::MainWindow \*ui;

QPoint p0;

int r;

QColor color;

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

p0.setX(ui->lineEdit\_cx->text().toInt());

p0.setY(ui->lineEdit\_cy->text().toInt());

r = ui->lineEdit\_r->text().toInt();

color = ui->frame\_color->palette().background().color();

}

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

{

delete ui;

}

void MainWindow::**on\_pushButton\_color\_clicked**()

{

color = QColorDialog::getColor();

ui->frame\_color->setPalette(QPalette(color));

}

void MainWindow::**on\_pushButton\_ok\_clicked**()

{

p0.setX(ui->lineEdit\_cx->text().toInt());

p0.setY(ui->lineEdit\_cy->text().toInt());

r = ui->lineEdit\_r->text().toInt();

update();

}

void MainWindow::***paintEvent***(QPaintEvent\*)

{

int w = ui->groupBox\_set->width();

int h = ui->groupBox\_set->height();

int x = width()-w-10;

int y = ui->groupBox\_set->y();

ui->groupBox\_set->setGeometry(QRect(x, y, w, h));

QPainter\* ptr = new QPainter(this);

ptr->save();

ptr->setPen(color);

circle c(p0, r, color);

c.showCircle(ptr);

ptr->restore();

delete ptr;

}

## 实验6 中点画椭圆算法

1. 创建一个主窗口应用程序cgExperiment06DrawEllipse

2. 建立ui界面

3. 创建椭圆类ellipse

#ifndef ELLIPSE\_H

#define ELLIPSE\_H

#include <QPoint>

#include <QColor>

#include <QPainter>

class ellipse

{

private:

QPoint p0;

int a, b;

QColor color;

public:

ellipse(QPoint p0, int a, int b, QColor color);

void **showEllipse**(QPainter\* ptr);

private:

void **Draw4thEllipse**(int x, int y, QPainter \*ptr);

};

#endif // ELLIPSE\_H

#include "ellipse.h"

ellipse::**ellipse**(QPoint p0, int a, int b, QColor color)

:p0(p0), a(a), b(b), color(color)

{

}

void ellipse::**showEllipse**(QPainter\* ptr)

{

if(a==0 || b==0)

return;

double d;

int x,y;

x = 0; y = b;

d = 4\*b\*b - 4\*a\*a\*b + a\*a;

while((a\*a\*(2\*y-1))>=2\*(b\*b\*(x+1)))

{

Draw4thEllipse(x, y, ptr);

if(d<0)

d=d+4\*b\*b\*(2\*x+3);

else{

d=d+4\*b\*b\*(2\*x+3)-8\*a\*a\*(y-1);

y--;

}

x++;

}

x = a; y = 0;

d = 4\*a\*a - 4\*a\*b\*b + b\*b;

while((b\*b\*(2\*x-1))>2\*(a\*a\*(y-1)))

{

Draw4thEllipse(x, y, ptr);

if(d<0)

d=d+4\*a\*a\*(2\*y+3);

else{

d=d+4\*a\*a\*(2\*y+3)-8\*b\*b\*(x-1);

x--;

}

y++;

}

}

void ellipse::**Draw4thEllipse**(int x, int y, QPainter \*ptr)

{

ptr->save();

ptr->setPen(color);

ptr->drawPoint(x + p0.x(), y + p0.y());

ptr->drawPoint(-x+ p0.x(), y + p0.y());

ptr->drawPoint(x + p0.x(), -y+ p0.y());

ptr->drawPoint(-x+ p0.x(), -y+ p0.y());

ptr->restore();

}

1. 修改主窗口类

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPoint>

#include <QColor>

#include <QColorDialog>

#include <QPainter>

#include <QRect>

#include "ellipse.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

void ***paintEvent***(QPaintEvent\*);

private slots:

void **on\_pushButton\_color\_clicked**();

void **on\_pushButton\_ok\_clicked**();

private:

Ui::MainWindow \*ui;

QPoint p0;

int a, b;

QColor color;

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

p0.setX(ui->lineEdit\_cx->text().toInt());

p0.setY(ui->lineEdit\_cy->text().toInt()+ui->mainToolBar->height());

a = ui->lineEdit\_a->text().toInt();

b = ui->lineEdit\_b->text().toInt();

color = ui->frame\_color->palette().background().color();

}

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

{

delete ui;

}

void MainWindow::***paintEvent***(QPaintEvent\*)

{

int w = ui->groupBox\_set->width();

int h = ui->groupBox\_set->height();

int x = width()-w-10;

int y = ui->groupBox\_set->y();

ui->groupBox\_set->setGeometry(QRect(x, y, w, h));

QPainter\* ptr = new QPainter(this);

ellipse e(p0, a, b, color);

e.showEllipse(ptr);

delete ptr;

}

void MainWindow::**on\_pushButton\_color\_clicked**()

{

color = QColorDialog::getColor();

ui->frame\_color->setPalette(QPalette(color));

}

void MainWindow::**on\_pushButton\_ok\_clicked**()

{

p0.setX(ui->lineEdit\_cx->text().toInt());

p0.setY(ui->lineEdit\_cy->text().toInt()+ui->mainToolBar->height());

a = ui->lineEdit\_a->text().toInt();

b = ui->lineEdit\_b->text().toInt();

color = ui->frame\_color->palette().background().color();

update();

}

## 实验7 多边形有序边表算法

1. 创建项目cgExperiment07EdgeTablePolygon
2. 用户界面如图



1. 创建edge类

#ifndef EDGE\_H

#define EDGE\_H

class edge

{

private:

double x;// 边的起始x坐标或边与扫描线的交点坐标

double dx;// 边的斜率的倒数

int ymax;// 边的最大y坐标

public:

edge();

edge(double x, double dx, int ymax);

void **setX**(double x);

void **setDx**(double dx);

void **setYmax**(int ymax);

double **getX**() const;

double **getDx**() const;

int **getYmax**() const;

};

#endif // EDGE\_H

#include "edge.h"

edge::**edge**()

{

}

edge::**edge**(double x, double dx, int ymax)

:x(x), dx(dx), ymax(ymax)

{

}

void edge::**setX**(double x)

{

this->x = x;

}

void edge::**setDx**(double dx)

{

this->dx = dx;

}

void edge::**setYmax**(int ymax)

{

this->ymax = ymax;

}

double edge::**getX**() const

{

return x;

}

double edge::**getDx**() const

{

return dx;

}

int edge::**getYmax**() const

{

return ymax;

}

1. 创建polygon类

#ifndef POLYGON\_H

#define POLYGON\_H

#include <QPoint>

#include <QVector>

#include <QList>

#include <QColor>

#include <QPainter>

#include <algorithm>

using namespace std;

#include "edge.h"

class polygon

{

private:

QVector<QPoint> points;

QColor color;

int **getMaxX**();

int **getMinX**();

int **getMaxY**();

int **getMinY**();

public:

polygon(QVector<QPoint>& points, QColor color);

void **showPolygonInEdgeTable**(QPainter\* ptr);//有序边表算法

};

#endif // POLYGON\_H

#include "polygon.h"

polygon::**polygon**(QVector<QPoint>& points, QColor color)

:points(points)

,color(color)

{

}

void polygon::**showPolygonInEdgeTable**(QPainter\* ptr)

{

if(points.size()<3)

return;

int i, j;

int x0, x1, y, tx;

int scanLines;//扫描线数量

int min;//最低扫描线号

int max;//最高扫描线号

QVector<QList<edge>> ET;//边表

QList<edge> AET;//活化边表

QVector<double> arr;//扫描线与各边交点表

QPoint p0;//边的起点

QPoint p1;//边的终点

QPoint temp;//保存位置变量

edge\* pNode;//边结点指针

QVector<QList<edge>>::iterator iterET;//边表矢量迭代器

QList<edge>::iterator iterEdge;//边链表迭代器

min = getMinY();

max = getMaxY();

scanLines = max - min;

//建立边表ET

ET.resize(scanLines);

//逐边进行处理，将每一条边的信息插入到ET中

for(i=0; i<points.size(); i++)

{

if(i < points.size()-1)

{

p0 = points[i];

p1 = points[i+1];

}

else

{

p0 = points[i];

p1 = points[0];

}

if(p0.y()>p1.y())//将p0设为边的起点坐标，y坐标较小

{

temp = p0;

p0 = p1;

p1 = temp;

}

if(p0.y() != p1.y())//非水平边

{

pNode = new edge;

pNode->setX(static\_cast<double>(p0.x()));

pNode->setDx(static\_cast<double>(p1.x()-p0.x())/(p1.y()-p0.y()));

pNode->setYmax(p1.y()-1);//下闭上开

ET[p0.y()-min].append(\*pNode);

}

}//所有边都已插入到ET中

for (i=0; i<scanLines; i++)// 开始扫描，各边依次加入到AET中

{

y = i + min;//当前扫描线y坐标

if(!ET[i].isEmpty())//有边加入AET

{

for(iterEdge=ET[i].begin(); iterEdge!=ET[i].end(); iterEdge++)

AET.append(\*iterEdge);

}

ET[i].clear();//边结点已取出加入AET，无需保留

//处理活化边表AET

if(!AET.isEmpty())//首先删除扫描线以下的边

{

for(iterEdge=AET.begin(); iterEdge!=AET.end(); iterEdge++)

{

if(iterEdge->getYmax()<y){

AET.erase(iterEdge);

iterEdge=AET.begin();

}

}

}

if(!AET.isEmpty())//活化边表非空，求出交点，排序，画线

{

for(iterEdge=AET.begin(); iterEdge!=AET.end(); iterEdge++)

{

arr.append(iterEdge->getX());//取出所有交点

iterEdge->setX(iterEdge->getX() + iterEdge->getDx());

}

sort(arr.begin(), arr.end());//交点排序

for(j=0; j<arr.size(); j++)

{

if(j%2 == 0)

{

tx = static\_cast<int>(arr[j]);// 左边交点向右取整

if(arr[j]-tx)

x0 = tx + 1;

else

x0 = tx;

x1 = static\_cast<int>(arr[j + 1]);// 右边交点向左取整

}

ptr->save();

ptr->setPen(color);

ptr->drawLine(QPoint(x0, y),QPoint(x1, y));

ptr->restore();

}

arr.clear();

}

}//所有扫描线处理完毕

ET.clear();

}

int polygon::**getMaxX**()

{

int max = 0;

QVector<QPoint>::iterator iter;

for (iter=points.begin(); iter!=points.end(); iter++)

if (iter->x() > max)

max = iter->x();

return max;

}

int polygon::**getMinX**()

{

int min = getMaxX();

QVector<QPoint>::iterator iter;

for (iter=points.begin(); iter!=points.end(); iter++)

if (iter->x() < min)

min = iter->x();

return min;

}

int polygon::**getMaxY**()

{

int max = 0;

QVector<QPoint>::iterator iter;

for (iter=points.begin(); iter!=points.end(); iter++)

if (iter->y() > max)

max = iter->y();

return max;

}

int polygon::**getMinY**()

{

int min = getMaxY();

QVector<QPoint>::iterator iterPos;

for (iterPos=points.begin(); iterPos!=points.end(); iterPos++)

if (iterPos->y() < min)

min = iterPos->y();

return min;

}

1. 修改主窗口类

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPoint>

#include <QVector>

#include <QColor>

#include <QColorDialog>

#include <QPainter>

#include <QRect>

#include "polygon.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

void ***paintEvent***(QPaintEvent\*);

private slots:

void **on\_pushButton\_color\_clicked**();

void **on\_pushButton\_ok\_clicked**();

private:

Ui::MainWindow \*ui;

int n;

QPoint p[8];

QColor color;

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

n = ui->lineEdit\_n->text().toInt();

p[0].setX(ui->lineEdit\_x0->text().toInt());

p[0].setY(ui->lineEdit\_y0->text().toInt());

p[1].setX(ui->lineEdit\_x1->text().toInt());

p[1].setY(ui->lineEdit\_y1->text().toInt());

p[2].setX(ui->lineEdit\_x2->text().toInt());

p[2].setY(ui->lineEdit\_y2->text().toInt());

p[3].setX(ui->lineEdit\_x3->text().toInt());

p[3].setY(ui->lineEdit\_y3->text().toInt());

p[4].setX(ui->lineEdit\_x4->text().toInt());

p[4].setY(ui->lineEdit\_y4->text().toInt());

p[5].setX(ui->lineEdit\_x5->text().toInt());

p[5].setY(ui->lineEdit\_y5->text().toInt());

p[6].setX(ui->lineEdit\_x6->text().toInt());

p[6].setY(ui->lineEdit\_y6->text().toInt());

p[7].setX(ui->lineEdit\_x7->text().toInt());

p[7].setY(ui->lineEdit\_y7->text().toInt());

color = ui->frame\_color->palette().background().color();

}

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

{

delete ui;

}

void MainWindow::***paintEvent***(QPaintEvent\*)

{

int w = ui->groupBox\_set->width();

int h = ui->groupBox\_set->height();

int x = width()-w-10;

int y = ui->groupBox\_set->y();

ui->groupBox\_set->setGeometry(QRect(x, y, w, h));

QPainter\* ptr = new QPainter(this);

QVector<QPoint> points;

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

points.append(p[i]);

polygon poly(points, color);

poly.showPolygonInEdgeTable(ptr);

delete ptr;

}

void MainWindow::**on\_pushButton\_color\_clicked**()

{

color = QColorDialog::getColor();

ui->frame\_color->setPalette(QPalette(color));

}

void MainWindow::**on\_pushButton\_ok\_clicked**()

{

n = ui->lineEdit\_n->text().toInt();

p[0].setX(ui->lineEdit\_x0->text().toInt());

p[0].setY(ui->lineEdit\_y0->text().toInt());

p[1].setX(ui->lineEdit\_x1->text().toInt());

p[1].setY(ui->lineEdit\_y1->text().toInt());

p[2].setX(ui->lineEdit\_x2->text().toInt());

p[2].setY(ui->lineEdit\_y2->text().toInt());

p[3].setX(ui->lineEdit\_x3->text().toInt());

p[3].setY(ui->lineEdit\_y3->text().toInt());

p[4].setX(ui->lineEdit\_x4->text().toInt());

p[4].setY(ui->lineEdit\_y4->text().toInt());

p[5].setX(ui->lineEdit\_x5->text().toInt());

p[5].setY(ui->lineEdit\_y5->text().toInt());

p[6].setX(ui->lineEdit\_x6->text().toInt());

p[6].setY(ui->lineEdit\_y6->text().toInt());

p[7].setX(ui->lineEdit\_x7->text().toInt());

p[7].setY(ui->lineEdit\_y7->text().toInt());

update();

}

## 实验8 边标志多边形填充算法

1. 创建项目cgExperiment07EdgeTablePolygon
2. 用户界面如图



1. 创建polygon类

#ifndef POLYGON\_H

#define POLYGON\_H

#include <QPoint>

#include <QVector>

#include <QColor>

#include <QPainter>

class polygon

{

private:

QVector<QPoint> points;

QColor color;

int **getMaxX**();

int **getMinX**();

int **getMaxY**();

int **getMinY**();

public:

polygon(QVector<QPoint>& points, QColor color);

void **showPolygonInEdgeFlag**(QPainter\* ptr);//边标志算法

};

#endif // POLYGON\_H

#include "polygon.h"

polygon::**polygon**(QVector<QPoint>& points, QColor color)

:points(points)

,color(color)

{

}

void polygon::**showPolygonInEdgeFlag**(QPainter\* ptr)

{

if(points.size()<3)

return;

ptr->save();

ptr->setPen(color);

//获取多边形最小包围盒

int minx, maxx, miny, maxy;

minx = getMinX();

maxx = getMaxX();

miny = getMinY();

maxy = getMaxY();

//设置一个标志矩阵

int m = maxy - miny;

int n = maxx - minx;

bool MF[m][n];

int i,j;

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

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

MF[i][j] = false;

//对于多边形每一条边，从这条边向右直到包围盒右边界进行扫描

QPoint p0, p1, temp;

double dx, tx;

int x;

int k;

int vertics = points.size();

for(k=0; k<vertics; k++)

{

//获取一条边p0p1

if(k == vertics-1)

{

p0 = points[k];

p1 = points[0];

}

else

{

p0 = points[k];

p1 = points[k+1];

}

if(p0.y()>p1.y())//将p0设为边的起点坐标，y坐标较小

{

temp = p0;

p0 = p1;

p1 = temp;

}

//对于一条边，从左至右对标志位求反

if(p0.y() != p1.y())//非水平边

{

dx =static\_cast<double>(p1.x()-p0.x())/(p1.y()-p0.y());

//对于一条边，y每次递增1,x每次递增dx

x = p0.x();

tx = x;

for(i=p0.y(); i<p1.y(); i++)

{

x = static\_cast<int>(tx);

for(j=x; j<maxx; j++)

MF[i-miny][j-minx] = !MF[i-miny][j-minx];

tx = tx+dx;

}

}

}

//对整个包围盒扫描，为true，用前景色绘制

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

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

if(MF[i][j])

ptr->drawPoint(j+minx, i+miny);

ptr->restore();

}

int polygon::**getMaxX**()

{

int max = 0;

QVector<QPoint>::iterator iter;

for (iter=points.begin(); iter!=points.end(); iter++)

if (iter->x() > max)

max = iter->x();

return max;

}

int polygon::**getMinX**()

{

int min = getMaxX();

QVector<QPoint>::iterator iter;

for (iter=points.begin(); iter!=points.end(); iter++)

if (iter->x() < min)

min = iter->x();

return min;

}

int polygon::**getMaxY**()

{

int max = 0;

QVector<QPoint>::iterator iter;

for (iter=points.begin(); iter!=points.end(); iter++)

if (iter->y() > max)

max = iter->y();

return max;

}

int polygon::**getMinY**()

{

int min = getMaxY();

QVector<QPoint>::iterator iterPos;

for (iterPos=points.begin(); iterPos!=points.end(); iterPos++)

if (iterPos->y() < min)

min = iterPos->y();

return min;

}

1. 修改主窗口类

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPoint>

#include <QVector>

#include <QColor>

#include <QColorDialog>

#include <QPainter>

#include <QRect>

#include "polygon.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

void ***paintEvent***(QPaintEvent\*);

private slots:

void **on\_pushButton\_color\_clicked**();

void **on\_pushButton\_ok\_clicked**();

private:

Ui::MainWindow \*ui;

int n;

QPoint p[8];

QColor color;

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

n = ui->lineEdit\_n->text().toInt();

p[0].setX(ui->lineEdit\_x0->text().toInt());

p[0].setY(ui->lineEdit\_y0->text().toInt());

p[1].setX(ui->lineEdit\_x1->text().toInt());

p[1].setY(ui->lineEdit\_y1->text().toInt());

p[2].setX(ui->lineEdit\_x2->text().toInt());

p[2].setY(ui->lineEdit\_y2->text().toInt());

p[3].setX(ui->lineEdit\_x3->text().toInt());

p[3].setY(ui->lineEdit\_y3->text().toInt());

p[4].setX(ui->lineEdit\_x4->text().toInt());

p[4].setY(ui->lineEdit\_y4->text().toInt());

p[5].setX(ui->lineEdit\_x5->text().toInt());

p[5].setY(ui->lineEdit\_y5->text().toInt());

p[6].setX(ui->lineEdit\_x6->text().toInt());

p[6].setY(ui->lineEdit\_y6->text().toInt());

p[7].setX(ui->lineEdit\_x7->text().toInt());

p[7].setY(ui->lineEdit\_y7->text().toInt());

color = ui->frame\_color->palette().background().color();

}

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

{

delete ui;

}

void MainWindow::**on\_pushButton\_color\_clicked**()

{

color = QColorDialog::getColor();

ui->frame\_color->setPalette(QPalette(color));

}

void MainWindow::**on\_pushButton\_ok\_clicked**()

{

n = ui->lineEdit\_n->text().toInt();

p[0].setX(ui->lineEdit\_x0->text().toInt());

p[0].setY(ui->lineEdit\_y0->text().toInt());

p[1].setX(ui->lineEdit\_x1->text().toInt());

p[1].setY(ui->lineEdit\_y1->text().toInt());

p[2].setX(ui->lineEdit\_x2->text().toInt());

p[2].setY(ui->lineEdit\_y2->text().toInt());

p[3].setX(ui->lineEdit\_x3->text().toInt());

p[3].setY(ui->lineEdit\_y3->text().toInt());

p[4].setX(ui->lineEdit\_x4->text().toInt());

p[4].setY(ui->lineEdit\_y4->text().toInt());

p[5].setX(ui->lineEdit\_x5->text().toInt());

p[5].setY(ui->lineEdit\_y5->text().toInt());

p[6].setX(ui->lineEdit\_x6->text().toInt());

p[6].setY(ui->lineEdit\_y6->text().toInt());

p[7].setX(ui->lineEdit\_x7->text().toInt());

p[7].setY(ui->lineEdit\_y7->text().toInt());

update();

}

void MainWindow::***paintEvent***(QPaintEvent\*)

{

int w = ui->groupBox\_set->width();

int h = ui->groupBox\_set->height();

int x = width()-w-10;

int y = ui->groupBox\_set->y();

ui->groupBox\_set->setGeometry(QRect(x, y, w, h));

QPainter\* ptr = new QPainter(this);

QVector<QPoint> points;

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

points.append(p[i]);

polygon poly(points, color);

poly.showPolygonInEdgeFlag(ptr);

delete ptr;

}

## 实验9 种子填充算法

1. 创建项目cgExperiment09SeedPolygon
2. Ui界面



1. 创建polygon类

#ifndef POLYGON\_H

#define POLYGON\_H

#include <QPoint>

#include <QVector>

#include <QColor>

#include <QStack>

#include <QPainter>

class polygon

{

private:

QVector<QPoint> points;

QPoint seed;

QColor color;

int **getMaxX**();

int **getMinX**();

int **getMaxY**();

int **getMinY**();

bool **isInPolygon**(QPoint p);

public:

polygon(QVector<QPoint>& points, QPoint seed, QColor color);

void **showPolygonInSeed**(QPainter\* ptr);//边标志算法

};

#endif // POLYGON\_H

#include "polygon.h"

#include <QPolygon>

#include <QDebug>

polygon::**polygon**(QVector<QPoint>& points, QPoint seed, QColor color)

:points(points)

,seed(seed)

,color(color)

{

}

void polygon::**showPolygonInSeed**(QPainter\* ptr)

{

if(!isInPolygon(seed) || points.size()<3)

return;

//获取多边形最小包围盒

int minx, maxx, miny, maxy;

minx = getMinX()-1;

maxx = getMaxX()+1;

miny = getMinY()-1;

maxy = getMaxY()+1;

if(seed.x()<=minx || seed.x()>=maxx || seed.y()<=miny || seed.y()>=maxy)

return;

//设置一个标志矩阵

int m = maxy - miny;

int n = maxx - minx;

bool MF[m][n];

int i,j;

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

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

MF[i][j] = false;

//绘制边界，并将边界像素标志置为true

QPoint p0, p1;

int temp;

int vertics = points.size();

int x, y;

double tx, ty, dx, dy;

int k;

int x0, x1, y0, y1;

ptr->save();

ptr->setPen(color);

for(k=0; k<vertics; k++)//逐边绘制

{

//获取一条边p0p1

if(k == vertics-1)

{

p0 = points[k];

p1 = points[0];

}

else

{

p0 = points[k];

p1 = points[k+1];

}

x0 = p0.x();

y0 = p0.y();

x1 = p1.x();

y1 = p1.y();

if(y0 == y1)//水平边

{

y = y0;

if(x0>x1)

{

temp = x0;

x0 = x1;

x1 = temp;

}

for(x=x0; x<x1; x++)

{

ptr->drawPoint(x, y);

MF[y-miny][x-minx] = true;

}

}

else if(x0 == x1)//垂直边

{

x = x0;

if(y0>y1)

{

temp = y0;

y0 = y1;

y1 = temp;

}

for(y=y0; y<y1; y++)

{

ptr->drawPoint(x, y);

MF[y-miny][x-minx] = true;

}

}

else//非水平和垂直边

{

dy = (double)(y1-y0)/(double)(x1-x0);

dx = 1/dy;

if(dy>-1 && dy<1)

{

if(x0>x1)

{

temp = x0;

x0 = x1;

x1 = temp;

temp = y0;

y0 = y1;

y1 = temp;

}

ty = y0;

for(x=x0; x<=x1; x++)

{

y = (int)(ty+0.5);

ptr->drawPoint(x, y);

MF[y-miny][x-minx] = true;

ty += dy;

}

}

else

{

if(y0>y1)

{

temp = x0;

x0 = x1;

x1 = temp;

temp = y0;

y0 = y1;

y1 = temp;

}

tx = x0;

for(y=y0; y<=y1; y++)

{

x = (int)(tx+0.5);

ptr->drawPoint(x, y);

MF[y-miny][x-minx] = true;

tx += dx;

}

}

}

}

//种子点压栈，压栈像素标志置为true

QStack<QPoint> stack;

stack.push(seed);

MF[seed.y()-miny][seed.x()-minx] = true;

//当栈不为空时，退栈，绘制像素点

//对当前退栈元素的左上右下点进行检查，若标志为false，压栈，并将压栈像素标志置为true

QPoint ps;

while(!stack.isEmpty())

{

ps = stack.pop();

ptr->drawPoint(ps);

p0.setX(ps.x()-1);//左

p0.setY(ps.y());

if(!MF[p0.y()-miny][p0.x()-minx])

{

stack.push(p0);

MF[p0.y()-miny][p0.x()-minx] = true;

}

p0.setX(ps.x()+1);//右

p0.setY(ps.y());

if(!MF[p0.y()-miny][p0.x()-minx])

{

stack.push(p0);

MF[p0.y()-miny][p0.x()-minx] = true;

}

p0.setX(ps.x());//下

p0.setY(ps.y()-1);

if(!MF[p0.y()-miny][p0.x()-minx])

{

stack.push(p0);

MF[p0.y()-miny][p0.x()-minx] = true;

}

p0.setX(ps.x());//上

p0.setY(ps.y()+1);

if(!MF[p0.y()-miny][p0.x()-minx])

{

stack.push(p0);

MF[p0.y()-miny][p0.x()-minx] = true;

}

}

ptr->restore();

}

int polygon::**getMaxX**()

{

int max = 0;

QVector<QPoint>::iterator iter;

for (iter=points.begin(); iter!=points.end(); iter++)

if (iter->x() > max)

max = iter->x();

return max;

}

int polygon::**getMinX**()

{

int min = getMaxX();

QVector<QPoint>::iterator iter;

for (iter=points.begin(); iter!=points.end(); iter++)

if (iter->x() < min)

min = iter->x();

return min;

}

int polygon::**getMaxY**()

{

int max = 0;

QVector<QPoint>::iterator iter;

for (iter=points.begin(); iter!=points.end(); iter++)

if (iter->y() > max)

max = iter->y();

return max;

}

int polygon::**getMinY**()

{

int min = getMaxY();

QVector<QPoint>::iterator iterPos;

for (iterPos=points.begin(); iterPos!=points.end(); iterPos++)

if (iterPos->y() < min)

min = iterPos->y();

return min;

}

bool polygon::**isInPolygon**(QPoint p)

{

int i, j;

bool c = false;

for (i=0, j=points.size()-1; i<points.size(); j=i++)

{

if(((points[i].y()>seed.y()) != (points[j].y()>seed.y())) &&

(seed.x()<(points[j].x()-points[i].x()) \*

(seed.y()-points[i].y()) / (points[j].y()-points[i].y()) + points[i].x()))

c = !c;

}

return c;

}

1. 修改主窗口类

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPoint>

#include <QVector>

#include <QColor>

#include <QColorDialog>

#include <QPainter>

#include <QRect>

#include "polygon.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

void ***paintEvent***(QPaintEvent\*);

private slots:

void **on\_pushButton\_color\_clicked**();

void **on\_pushButton\_ok\_clicked**();

private:

Ui::MainWindow \*ui;

QColor color;

int n;

QPoint p[8];

QPoint seed;

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

color = ui->frame\_color->palette().background().color();

n = ui->lineEdit\_n->text().toInt();

p[0].setX(ui->lineEdit\_x0->text().toInt());

p[0].setY(ui->lineEdit\_y0->text().toInt());

p[1].setX(ui->lineEdit\_x1->text().toInt());

p[1].setY(ui->lineEdit\_y1->text().toInt());

p[2].setX(ui->lineEdit\_x2->text().toInt());

p[2].setY(ui->lineEdit\_y2->text().toInt());

p[3].setX(ui->lineEdit\_x3->text().toInt());

p[3].setY(ui->lineEdit\_y3->text().toInt());

p[4].setX(ui->lineEdit\_x4->text().toInt());

p[4].setY(ui->lineEdit\_y4->text().toInt());

p[5].setX(ui->lineEdit\_x5->text().toInt());

p[5].setY(ui->lineEdit\_y5->text().toInt());

p[6].setX(ui->lineEdit\_x6->text().toInt());

p[6].setY(ui->lineEdit\_y6->text().toInt());

p[7].setX(ui->lineEdit\_x7->text().toInt());

p[7].setY(ui->lineEdit\_y7->text().toInt());

seed.setX(ui->lineEdit\_sx->text().toInt());

seed.setY(ui->lineEdit\_sy->text().toInt());

}

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

{

delete ui;

}

void MainWindow::**on\_pushButton\_color\_clicked**()

{

color = QColorDialog::getColor();

ui->frame\_color->setPalette(QPalette(color));

}

void MainWindow::**on\_pushButton\_ok\_clicked**()

{

n = ui->lineEdit\_n->text().toInt();

p[0].setX(ui->lineEdit\_x0->text().toInt());

p[0].setY(ui->lineEdit\_y0->text().toInt());

p[1].setX(ui->lineEdit\_x1->text().toInt());

p[1].setY(ui->lineEdit\_y1->text().toInt());

p[2].setX(ui->lineEdit\_x2->text().toInt());

p[2].setY(ui->lineEdit\_y2->text().toInt());

p[3].setX(ui->lineEdit\_x3->text().toInt());

p[3].setY(ui->lineEdit\_y3->text().toInt());

p[4].setX(ui->lineEdit\_x4->text().toInt());

p[4].setY(ui->lineEdit\_y4->text().toInt());

p[5].setX(ui->lineEdit\_x5->text().toInt());

p[5].setY(ui->lineEdit\_y5->text().toInt());

p[6].setX(ui->lineEdit\_x6->text().toInt());

p[6].setY(ui->lineEdit\_y6->text().toInt());

p[7].setX(ui->lineEdit\_x7->text().toInt());

p[7].setY(ui->lineEdit\_y7->text().toInt());

seed.setX(ui->lineEdit\_sx->text().toInt());

seed.setY(ui->lineEdit\_sy->text().toInt());

update();

}

void MainWindow::***paintEvent***(QPaintEvent\*)

{

int w = ui->groupBox\_set->width();

int h = ui->groupBox\_set->height();

int x = width()-w-10;

int y = ui->groupBox\_set->y();

ui->groupBox\_set->setGeometry(QRect(x, y, w, h));

QPainter\* ptr = new QPainter(this);

QVector<QPoint> points;

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

points.append(p[i]);

polygon poly(points, seed, color);

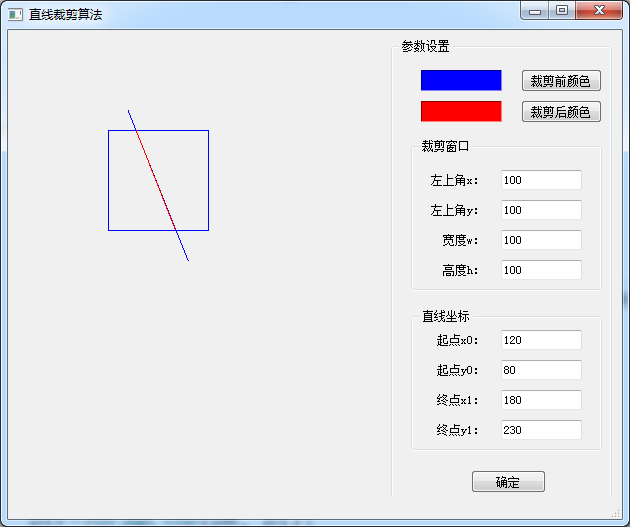
poly.showPolygonInSeed(ptr);

delete ptr;

}

## 实验10 直线的裁剪

1. 项目名称cgExperiment10ClipLine



1. 代码实现

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QColor>

#include <QPoint>

#include <QRect>

#include <QPainter>

#include <QColorDialog>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

void ***paintEvent***(QPaintEvent \*);

bool **lineClip**(QPoint& p0, QPoint& p1, QRect rt);

char **getCSCode**(int x, int y, QRect rt);

private slots:

void **on\_pushButton\_oldColor\_clicked**();

void **on\_pushButton\_newColor\_clicked**();

void **on\_pushButton\_ok\_clicked**();

private:

Ui::MainWindow \*ui;

QColor oldColor, newColor;

int x, y, w, h;

int x0, y0, x1, y1;

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

oldColor = ui->frame\_oldColor->palette().background().color();

newColor = ui->frame\_newColor->palette().background().color();

x = ui->lineEdit\_leftx->text().toInt();

y = ui->lineEdit\_lefty->text().toInt();

w = ui->lineEdit\_w->text().toInt();

h = ui->lineEdit\_h->text().toInt();

x0 = ui->lineEdit\_x0->text().toInt();

y0 = ui->lineEdit\_y0->text().toInt();

x1 = ui->lineEdit\_x1->text().toInt();

y1 = ui->lineEdit\_y1->text().toInt();

}

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

{

delete ui;

}

void MainWindow::***paintEvent***(QPaintEvent\*)

{

int ww = ui->groupBox\_set->width();

int hh = ui->groupBox\_set->height();

int xx = width()-ww-10;

int yy = ui->groupBox\_set->y();

ui->groupBox\_set->setGeometry(QRect(xx, yy, ww, hh));

QPainter \*ptr = new QPainter(this);

QPoint p0(x0, y0);

QPoint p1(x1, y1);

QRect rect(x, y, w, h);

ptr->save();

ptr->setPen(oldColor);

ptr->drawLine(p0, p1);

ptr->drawRect(rect);

ptr->restore();

if(lineClip(p0, p1, rect))

{

ptr->save();

ptr->setPen(newColor);

ptr->drawLine(p0, p1);

ptr->restore();

}

delete ptr;

}

bool MainWindow::**lineClip**(QPoint& p0, QPoint& p1, QRect rt)

{

bool accept, done;

char c0, c1, code;

int x0, y0, x1, y1, x, y;

double m;

accept = false;//线段可见标志

done = false;//裁剪完成标志

x0 = p0.x();

y0 = p0.y();

x1 = p1.x();

y1 = p1.y();

c0 = getCSCode(x0, y0, rt);//返回线段起点的编码

c1 = getCSCode(x1, y1, rt);//返回线段终点的编码

while (!done)

{

if (!c0 && !c1){//线段完全可见

p0 = QPoint(x0,y0);

p1 = QPoint(x1,y1);

accept = true;

done = true;

}

else if (c0&c1){//线段完全不可见

p0 = QPoint(0,0);

p1 = QPoint(0,0);

accept = false;

done = true;

}

else{//处理非完全可见又非显然不可见的情况

if (c0){//p0不可见

code = c0;

}

else{////p0可见，则p1一定不可见

code = c1;

}

if (code & 0x01){//线段与窗口的左边有交

x = rt.left();

m = (double)(y1 - y0) / (double)(x1 - x0);

y = y0 + (int)((x - x0)\*m);

}

else if (code & 0x08){//线段与窗口的下边有交

y = rt.bottom();

m = (double)(x1 - x0) / (double)(y1 - y0);

x = x0 + (int)((y - y0)\*m);

}

else if (code & 0x02){//线段与窗口的右边有交

x = rt.right();

m = (double)(y1 - y0) / (double)(x1 - x0);

y = y0 + (int)((x - x0)\*m);

}

else if (code & 0x04){//线段与窗口的上边有交

y = rt.top();

m = (double)(x1 - x0) / (double)(y1 - y0);

x = x0 + (int)((y - y0)\*m);

}

if (code == c0){

x0 = x; y0 = y;

c0 = getCSCode(x0, y0, rt);

}

else{

x1 = x; y1 = y;

c1 = getCSCode(x1, y1, rt);

}

}

}

return accept;

}

char MainWindow::**getCSCode**(int x, int y, QRect rt)

{

char code=0;

if(x<rt.left())//编码为\*\*\*1,最后一位置1

code=code|0x01;

else//编码为\*\*\*0，最后一位置0

code=code&0xfe;

if(x>rt.right())//编码为\*\*1\*

code=code|0x02;

else//编码为\*\*0\*

code=code&0xfd;

if(y<rt.top())//编码为\*1\*\*

code=code|0x04;

else//编码为\*0\*\*

code=code&0xfb;

if(y>rt.bottom())//编码为1\*\*\*

code=code|0x08;

else//编码为0\*\*\*

code=code&0xf7;

return code;

}

void MainWindow::**on\_pushButton\_oldColor\_clicked**()

{

oldColor = QColorDialog::getColor();

ui->frame\_oldColor->setPalette(QPalette(oldColor));

}

void MainWindow::**on\_pushButton\_newColor\_clicked**()

{

newColor = QColorDialog::getColor();

ui->frame\_newColor->setPalette(QPalette(newColor));

}

void MainWindow::**on\_pushButton\_ok\_clicked**()

{

x = ui->lineEdit\_leftx->text().toInt();

y = ui->lineEdit\_lefty->text().toInt();

w = ui->lineEdit\_w->text().toInt();

h = ui->lineEdit\_h->text().toInt();

x0 = ui->lineEdit\_x0->text().toInt();

y0 = ui->lineEdit\_y0->text().toInt();

x1 = ui->lineEdit\_x1->text().toInt();

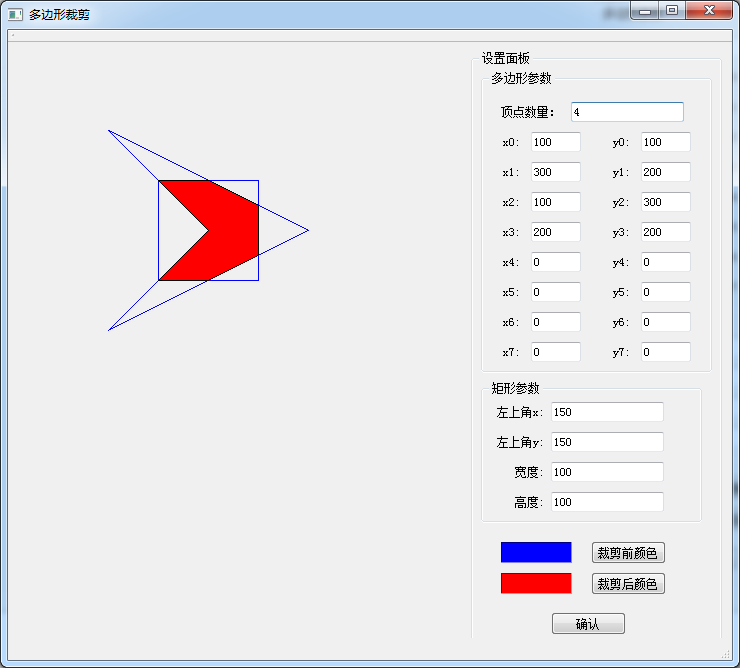
y1 = ui->lineEdit\_y1->text().toInt();

update();

}

## 实验11 多边形的裁剪算法

1. 项目名称：cgExperiment11ClipPolygon
2. ui



#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPainter>

#include <QPoint>

#include <QRect>

#include <QColor>

#include <QColorDialog>

#include <QVector>

#include <QPolygon>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

void ***paintEvent***(QPaintEvent\*);

private slots:

void **on\_pushButton\_oldColor\_clicked**();

void **on\_pushButton\_newColor\_clicked**();

void **on\_pushButton\_ok\_clicked**();

private:

Ui::MainWindow \*ui;

int n;

QPoint p[8];

QRect rect;

QColor oldColor, newColor;

QPolygon **polyClip**(QPolygon poly1, QRect &rect1);

QPolygon **edgeClip**(QPolygon poly1, QPoint p0, QPoint p1);

bool **isInsideEdge**(QPoint p, QPoint p0, QPoint p1);

QPoint **intersect**(QPoint s, QPoint p, QPoint p0, QPoint p1);

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

n = ui->lineEdit\_vertexNum->text().toInt();

p[0].setX(ui->lineEdit\_x0->text().toInt());

p[0].setY(ui->lineEdit\_y0->text().toInt());

p[1].setX(ui->lineEdit\_x1->text().toInt());

p[1].setY(ui->lineEdit\_y1->text().toInt());

p[2].setX(ui->lineEdit\_x2->text().toInt());

p[2].setY(ui->lineEdit\_y2->text().toInt());

p[3].setX(ui->lineEdit\_x3->text().toInt());

p[3].setY(ui->lineEdit\_y3->text().toInt());

p[4].setX(ui->lineEdit\_x4->text().toInt());

p[4].setY(ui->lineEdit\_y4->text().toInt());

p[5].setX(ui->lineEdit\_x5->text().toInt());

p[5].setY(ui->lineEdit\_y5->text().toInt());

p[6].setX(ui->lineEdit\_x6->text().toInt());

p[6].setY(ui->lineEdit\_y6->text().toInt());

p[7].setX(ui->lineEdit\_x7->text().toInt());

p[7].setY(ui->lineEdit\_y7->text().toInt());

rect.setLeft(ui->lineEdit\_topX->text().toInt());

rect.setTop(ui->lineEdit\_topY->text().toInt());

rect.setWidth(ui->lineEdit\_rectWidth->text().toInt());

rect.setHeight(ui->lineEdit\_rectHeight->text().toInt());

oldColor = ui->frame\_oldColor->palette().background().color();

newColor = ui->frame\_newColor->palette().background().color();

}

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

{

delete ui;

}

void MainWindow::**on\_pushButton\_oldColor\_clicked**()

{

oldColor = QColorDialog::getColor();

ui->frame\_oldColor->setPalette(QPalette(oldColor));

}

void MainWindow::**on\_pushButton\_newColor\_clicked**()

{

newColor = QColorDialog::getColor();

ui->frame\_oldColor->setPalette(QPalette(newColor));

}

void MainWindow::**on\_pushButton\_ok\_clicked**()

{

n = ui->lineEdit\_vertexNum->text().toInt();

p[0].setX(ui->lineEdit\_x0->text().toInt());

p[0].setY(ui->lineEdit\_y0->text().toInt());

p[1].setX(ui->lineEdit\_x1->text().toInt());

p[1].setY(ui->lineEdit\_y1->text().toInt());

p[2].setX(ui->lineEdit\_x2->text().toInt());

p[2].setY(ui->lineEdit\_y2->text().toInt());

p[3].setX(ui->lineEdit\_x3->text().toInt());

p[3].setY(ui->lineEdit\_y3->text().toInt());

p[4].setX(ui->lineEdit\_x4->text().toInt());

p[4].setY(ui->lineEdit\_y4->text().toInt());

p[5].setX(ui->lineEdit\_x5->text().toInt());

p[5].setY(ui->lineEdit\_y5->text().toInt());

p[6].setX(ui->lineEdit\_x6->text().toInt());

p[6].setY(ui->lineEdit\_y6->text().toInt());

p[7].setX(ui->lineEdit\_x7->text().toInt());

p[7].setY(ui->lineEdit\_y7->text().toInt());

rect.setLeft(ui->lineEdit\_topX->text().toInt());

rect.setTop(ui->lineEdit\_topY->text().toInt());

rect.setWidth(ui->lineEdit\_rectWidth->text().toInt());

rect.setHeight(ui->lineEdit\_rectHeight->text().toInt());

update();

}

void MainWindow::***paintEvent***(QPaintEvent\*)

{

int w = ui->groupBox\_set->width();

int h = ui->groupBox\_set->height();

int x = width()-w-10;

int y = ui->groupBox\_set->y();

ui->groupBox\_set->setGeometry(QRect(x, y, w, h));

QVector<QPoint> points;

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

points.append(p[i]);

if(points.size()<3)

return;

QPainter\* ptr = new QPainter(this);

QPolygon poly(points);

ptr->save();

ptr->setPen(oldColor);

ptr->drawRect(rect);

ptr->drawPolygon(poly);

ptr->restore();

QPolygon poly1 = polyClip(poly, rect);

if(poly1.size()<3)

return;

ptr->save();

ptr->setBrush(newColor);

ptr->drawPolygon(poly1);

ptr->restore();

delete ptr;

}

QPolygon MainWindow::**polyClip**(QPolygon poly1, QRect &rect1)

{

int x0, y0, x1, y1;

QPolygon py0, py1, py2, py3;

//裁剪左边

x0 = rect1.x();

y0 = rect1.y()+rect1.height();

x1 = rect1.x();

y1 = rect1.y();

py0 = edgeClip(poly1, QPoint(x0,y0), QPoint(x1,y1));

//裁剪下边

x0 = rect1.x()+rect1.width();

y0 = rect1.y()+rect1.height();

x1 = rect1.x();

y1 = rect1.y()+rect1.height();

py1 = edgeClip(py0, QPoint(x0,y0), QPoint(x1,y1));

//裁剪右边

x0 = rect1.x()+rect1.width();

y0 = rect1.y();

x1 = rect1.x()+rect1.width();

y1 = rect1.y()+rect1.height();

py2 = edgeClip(py1, QPoint(x0,y0), QPoint(x1,y1));

//裁剪上边

x0 = rect1.x();

y0 = rect1.y();

x1 = rect1.x()+rect1.width();

y1 = rect1.y();

py3 = edgeClip(py2, QPoint(x0,y0), QPoint(x1,y1));

return py3;

}

QPolygon MainWindow::**edgeClip**(QPolygon poly1, QPoint p0, QPoint p1)

{

QVector<QPoint> v;

int i;

int m = poly1.length();

QPoint s, p;

QPoint pt;

s = poly1.point(m-1);

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

{

p = poly1.point(i);

if(isInsideEdge(p,p0,p1))//点p在边(p0,p1)的内侧

{

if(isInsideEdge(s,p0,p1))//情况1

v.append(p);

else//情况4

{

pt = intersect(s,p,p0,p1);

v.append(pt);

v.append(p);

}

}

else if(isInsideEdge(s,p0,p1)){//情况2

pt = intersect(s,p,p0,p1);

v.append(pt);

}//情况3没有输出

s=p;

}

return QPolygon(v);

}

//判断点p位于矩形边(p0,p1)的内侧还是外侧

bool MainWindow::**isInsideEdge**(QPoint p, QPoint p0, QPoint p1)

{

if(p0.x()>p1.x())//裁剪边为窗口的下边

{

if(p.y()<=p0.y())

return true;

}

else if(p0.x()<p1.x())//裁剪边为窗口的上边

{

if(p.y()>=p0.y())

return true;

}

else if(p0.y()<p1.y())//裁减边为窗口的右边

{

if(p.x()<=p0.x())

return true;

}

else if(p0.y()>p1.y())//裁减边为窗口的左边

{

if(p.x()>=p0.x())

return true;

}

return false;

}

QPoint MainWindow::**intersect**(QPoint s, QPoint p, QPoint p0, QPoint p1)

{

double m;

int x,y;

if(p0.y()==p1.y())//水平裁剪边

{

m=(double)(p.x()-s.x())/(double)(p.y()-s.y());

x=s.x()+(int)((p0.y()-s.y())\*m);

y=p0.y();

}

else//竖直裁剪边

{

m=(double)(p.y()-s.y())/(double)(p.x()-s.x());

x = p0.x();

y=s.y()+(int)((p0.x()-s.x())\*m);

}

return QPoint(x,y);

}

## 实验12 Weiler-Athenton多边形裁剪算法

### 一、方法

假设被裁剪多边形和裁剪窗口的顶点序列都按顺时针方向排列。当两个多边形相交时，交点必然成对出现，其中一个是从被裁剪多边形进入裁剪窗口的交点，称为“入点”，另一个是从被裁剪多边形离开裁剪窗口的交点，称为“出点”。

　　算法从被裁剪多边形的一个入点开始，碰到入点，沿着被裁剪多边形按顺时针方向搜集顶点序列；

　　而当遇到出点时，则沿着裁剪窗口按顺时针方向搜集顶点序列。

　　按上述规则，如此交替地沿着两个多边形的边线行进，直到回到起始点。这时，收集到的全部顶点序列就是裁剪所得的一个多边形。

　　由于可能存在分裂的多边形，因此算法要考虑：将搜集过的入点的入点记号删去，以免重复跟踪。将所有的入点搜集完毕后算法结束。

### 二、算法

1、顺时针输入被裁剪多边形顶点序列Ⅰ放入数组1中。

2、顺时针输入裁剪窗口顶点序列Ⅱ放入数组2中。

3、求出被裁剪多边形和裁剪窗口相交的所有交点，并给每个交点打上“入”、“出”标记。然后将交点按顺序插入序列Ⅰ得到新的顶点序列Ⅲ，并放入数组3中；同样也将交点按顺序插入序列Ⅱ得到新的顶点序列Ⅳ，放入数组4中；

4、初始化输出数组Q，令数组Q为空。接着从数组3中寻找“入”点。如果“入”点没找到，程序结束。

5、如果找到“入”点，则将“入”点放入S中暂存。

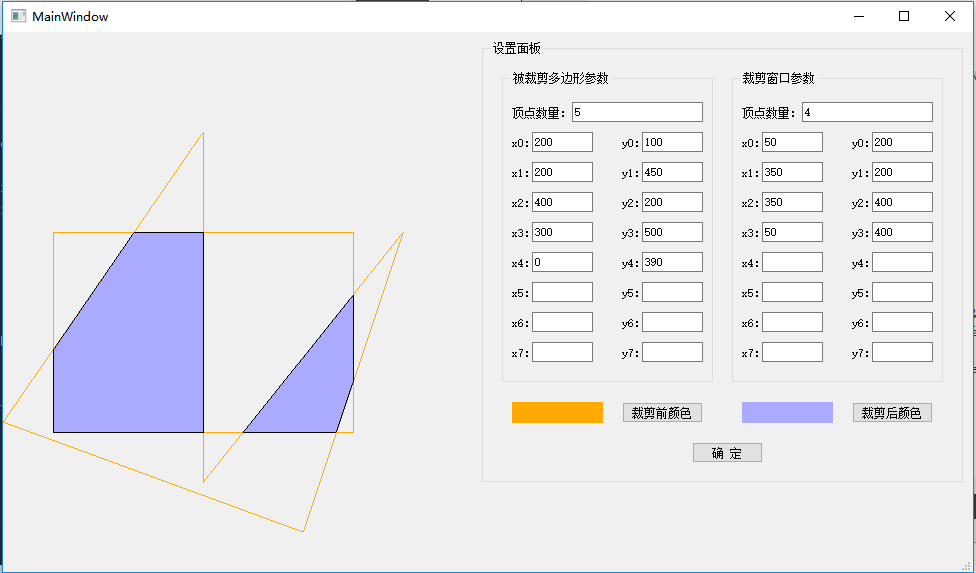
6、将“入”点录入到输出数组Q中。并从数组3中将该“入”点的“入”点标记删去。

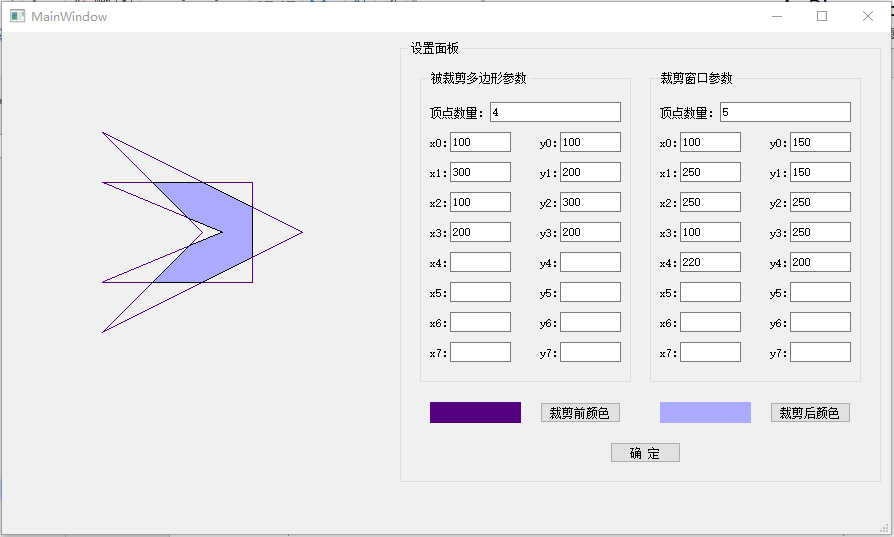
7、沿数组3顺序取顶点：如果顶点不是“出点”，则将顶点录入到输出数组Q中，流程转第7步。否则，流程转第8步。

8、沿数组4顺序取顶点：如果顶点不是“入点”，则将顶点录入到输出数组Q中，流程转第8步。否则，流程转第9步。

9、如果顶点不等于起始点S，流程转第6步，继续跟踪数组3。否则，将数组Q输出；流程转第4步，寻找可能存在的分裂多边形。算法在第4步：满足“入”点没找到的条件时，算法结束。

裁剪效果展示如下：





### 三、数据结构

#### （一）主窗口类

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

void ***paintEvent***(QPaintEvent\*);

private slots:

void **on\_pushButton\_oldColor\_clicked**();

void **on\_pushButton\_newColor\_clicked**();

void **on\_pushButton\_ok\_clicked**();

private:

Ui::MainWindow \*ui;

int pn;//被裁剪的多边形顶点个数

int vn;//裁剪窗口的顶点个数

QPoint pp[8];//被裁剪的多边形的顶点

QPoint vp[8];//裁剪窗口的顶点

QList<QLine>pLine; //被裁剪的多边形的边

QList<QLine>vLine; //裁剪窗口的边

QList<InPoint>ppoint; //被裁剪的多边形插入交点之后的序列

QList<InPoint>vpoint; //裁剪窗口的插入交点之后的序列

QColor oldColor,newColor;

//根据pLine、vLine的边数据，计算出所有交点，保存在ppoint，vpoint

void **SetPLane**();

//求直线AB与线段CD的交点，若有交点返回true

//t代表直线AB与线段CD的交点在线段AB上的位置，运用定比分点公式

bool **IsInsect**(const QPoint A,const QPoint B,const QPoint C,const QPoint D,QPoint &Insect,double &t);

//将搜索出的多边形顶点依次存入PointInter

//Num记录每个多边形的顶点个数，Num序列的大小代表多边形的个数

void **BoolIntersection**(QList<QPoint> &PointInter, QList<int> &Num);

};

class InPoint

{

public:

InPoint();

InPoint(QPoint point,int jud);

void **setPoint**(QPoint p);

void **setJudge**(int jud);

QPoint **getPoint**();

int **getJudge**();

private:

QPoint point;//点的位置

int judge; //1（入点）、2（出点）为交点，0为端点

};

#### （二）交点类

class InPoint

{

public:

InPoint();

InPoint(QPoint point,int jud);

void **setPoint**(QPoint p);

void **setJudge**(int jud);

QPoint **getPoint**();

int **getJudge**();

private:

QPoint point; //点的位置

int judge; //1（入点）、2（出点）为交点，0为端点

};

### 四、核心代码详注

#### （一）算法主体：paintEvent

void MainWindow::***paintEvent***(QPaintEvent\*){

//设置自动调整窗口大小

int w = ui->groupBox\_set->width();

int h = ui->groupBox\_set->height();

int x = width()-w-10;

int y = ui->groupBox\_set->y();

ui->groupBox\_set->setGeometry(QRect(x, y, w, h));

//将输入的点从存放到vector中

QVector<QPoint>points\_p,points\_v;

for(int i=0;i<pn;++i)points\_p.append(pp[i]);

if(points\_p.size()<3)

return;

for(int i=0;i<vn;++i)points\_v.append(vp[i]);

if(points\_v.size()<3)

return;

//先分别画出窗口以及被裁减的多边形

QPainter \*ptr=new QPainter(this);

QPolygon polyp(points\_p);

QPolygon polyv(points\_v);

ptr->save();

ptr->setPen(oldColor);

ptr->drawPolygon(polyp);

ptr->drawPolygon(polyv);

ptr->restore();

//清理之前的绘图数据

pLine.clear();

vLine.clear();

QPoint A,B;

//对于被裁减多边形，从起点到终点的所有边顺时针插入多边形边链表

for(int i=0;i<pn-1;++i){

A.setX(pp[i].x());A.setY(pp[i].y());

B.setX(pp[i+1].x());B.setY(pp[i+1].y());

pLine.push\_back(QLine(A,B));

A=B;

}

B.setX(pp[0].x());B.setY(pp[0].y());

pLine.push\_back(QLine(A,B));

//对于裁剪窗口，从起点到终点的所有边顺时针插入多边形边链表

for(int i=0;i<vn-1;++i){

A.setX(vp[i].x());A.setY(vp[i].y());

B.setX(vp[i+1].x());B.setY(vp[i+1].y());

vLine.push\_back(QLine(A,B));

A=B;

}

B.setX(vp[0].x());B.setY(vp[0].y());

vLine.push\_back(QLine(A,B));

//根据线段表pLine、vLine得出交点表ppoint,vpoint

SetPLane();

QList<QPoint>PointInter;//裁剪之后的多边形的坐标

QList<int>Num;//每个裁剪后多边形的坐标个数，Num的size表示交集面的个数

//根据插入好入点出点还有端点的序列ppoint、vpoint，找裁剪后的多边形

BoolIntersection(PointInter, Num);

ptr->save();

ptr->setBrush(newColor);

QList<QPoint>::iterator it;

QList<int>::iterator itN;

QVector<QPoint>points\_ans;//保留每个裁剪后多边形的顶点

it = PointInter.begin();

for (itN = Num.begin(); itN != Num.end(); itN++){

points\_ans.clear();

for (int i = 0; i < \*itN; i++){

points\_ans.push\_back(\*it);

it++;

}

ptr->drawPolygon(points\_ans);

}

ptr->restore();

delete ptr;

}

#### （二）求交点，插入原序列：SetPLane

//根据pLine、vLine的边数据，计算出所有交点，保存在ppoint，vpoint

void MainWindow::**SetPLane**(){

//先清空之前的绘图数据

ppoint.clear();

vpoint.clear();

QList<QLine>::iterator itp,itv;

QVector<double>t;//用于保存每条边与其他边相交时交点在线段上的位置

QPoint tp;

double tt;//对于线段A->B，tt<0代表交点在A左侧，tt==0代表在A，0<tt<1代表在AB中间，tt==1代表在B，tt>1代表在B右侧

//对于多边形的每一条边，找到窗口的每一条边（看作线段）与那条边（看作直线）的交点

for(itp=pLine.begin();itp!=pLine.end();itp++){

for(itv=vLine.begin();itv!=vLine.end();itv++){

if(IsInsect(itp->p1(),itp->p2(),itv->p1(),itv->p2(),tp,tt)){

t.push\_back(tt);

}

}

sort(t.begin(),t.end());//把交点从小到大排序

InPoint tInp;

int nt=0;

int nt\_in=0;

for (int i = 0; i < t.size(); i++)

{

if(i!=0&&t[i]==t[i-1])//避免相交在两条线段的同一个点导致出点入点判断失误的。

continue;

nt\_in++;// 判断交点是出去还是进入 被2整除出 余1进

if (t[i] <= 1) nt++; //统计小于1的个数 用于判断B点是否在内部 被2整除不在 余1在

if (t[i] > 0 && t[i] <= 1)

{

tInp.setPoint(itp->p1() +(itp->p2() - itp->p1()) \* t[i]);

if(nt\_in % 2 == 0) tInp.setJudge(2); //交点，出点

else tInp.setJudge(1); // 交点，入点

ppoint.push\_back(tInp);

}

}

tInp.setPoint(itp->p2());

if(nt % 2 == 0) tInp.setJudge(0);//端点

else tInp.setJudge(1);//交点，入点

ppoint.push\_back(tInp);

t.clear();

}

//对于VIEW的每一条边，找到POLYGON的每一条边与那条边的交点

for (itv = vLine.begin(); itv != vLine.end(); itv++){

for (itp = pLine.begin(); itp != pLine.end(); itp++){

if (IsInsect(itv->p1(), itv->p2(), itp->p1(), itp->p2(), tp, tt)){

t.push\_back(tt);

}

}

sort(t.begin(), t.end());

InPoint tInp;

int nt = 0;

int nt\_in = 0;

for (int i = 0; i < t.size(); i++)

{

// if(i!=0&&t[i]==t[i-1])//避免相交在两条线段的同一个点导致出点入点判断失误的

// continue;

nt\_in++;

if (t[i] <= 1) nt++; //统计小于1的个数（也就是B点之前交点的个数）用于判断B点是否在内部 被2整除不在 余1在

if (t[i] > 0 && t[i] <= 1)

{

tInp.setPoint(itv->p1() + (itv->p2() - itv->p1()) \* t[i]);

if (nt\_in % 2 == 0) tInp.setJudge(2); // 交点，出点

else tInp.setJudge(1); // 交点，入点

vpoint.push\_back(tInp);

}

}

tInp.setPoint(itv->p2());

if (nt % 2 == 0) tInp.setJudge(0);//端点在外部

else tInp.setJudge(1);//端点在内部，当作入点标记

vpoint.push\_back(tInp);

t.clear();

}

}

#### （三）搜索裁剪后区域：BoolIntersection

//将搜索出的多边形顶点依次存入PointInter，Num记录每个多边形的顶点个数，Num序列的大小代表多边形的个数

void MainWindow::**BoolIntersection**(QList<QPoint> &PointInter, QList<int> &Num){

bool jumpP=true;//true代表跳转到被裁剪多边形，false代表跳转到窗口多边形

int state=0;//0代表在遍历被裁剪多边形

QList<InPoint>::iterator itp,ittemp\_back,ittemp;

//先遍历ppoint

itp=ppoint.begin();

//P代表被裁剪多边形，V代表窗口多边形

while(true)

{

if(itp->getJudge()==1){//如果是入点

int Npoint=0;

PointInter.push\_back(QPoint(itp->getPoint().x(),itp->getPoint().y()));

Npoint++;

int a=itp->getPoint().x();

int b=itp->getPoint().y();

ittemp\_back=itp;

itp++;

ittemp=itp;

//如果之前的跳转是跳转到P，那么现在跳转过来之后下次就跳转到V；反之亦然

if(!jumpP)jumpP=true;

else jumpP=false;

//找到一个入点，以这个入点为起点找这个裁剪区域的其他点

while(a!=ittemp->getPoint().x()||b!=ittemp->getPoint().y())

{

if (ittemp->getJudge() == 1)//入点

{

PointInter.push\_back(QPoint(ittemp->getPoint().x(),ittemp->getPoint().y()));

ittemp->setJudge(3); // 标记已经pick的点

Npoint++;

ittemp\_back = ittemp;

ittemp++;

}//end入点的情况

else if (ittemp->getJudge() == 2)//出点，跳转

{

PointInter.push\_back(QPoint(ittemp->getPoint().x(),ittemp->getPoint().y()));

ittemp->setJudge(3); // 标记已经pick的点

Npoint++;

ittemp\_back = ittemp;

if (!jumpP) {//应该跳转到V

ittemp = vpoint.begin();

jumpP = true;//表明下一次应该跳转到P

}

else//应该跳转到P

{

ittemp = ppoint.begin();

jumpP = false;//下一次跳转到V

}

//在跳转到的数组中找到对应的点

while (ittemp->getPoint().x() != ittemp\_back->getPoint().x()||ittemp->getPoint().y()!=ittemp\_back->getPoint().y())

{

ittemp++;

}

ittemp->setJudge(3); // 标记已经pick的点

ittemp++;

if (ittemp->getJudge() == 0)

break; // 交点为一个的情况

}//end出点的情况

if (!jumpP)

{

if (ittemp == ppoint.end())

ittemp = ppoint.begin();

}

else

{

if (ittemp == vpoint.end())

ittemp = vpoint.begin();

}

}

Num.push\_back(Npoint);

Npoint = 0; //重新计数

}//找完一个裁剪后的区域了

//继续找下一个点，作为裁剪区域的起点

itp++;

if(state==0){//当前在遍历P数组

if(itp==ppoint.end()){//如果P数组遍历到结尾，那么下次从V开始，并且跳转标记为P

itp=vpoint.begin();

jumpP=true;

state=1;

}

}

else{//当前在遍历V数组，V遍历结束，说明整个都结束了

if(itp==vpoint.end())

break;

}

}//搜索完全部的顶点序列了

}

#### （四）辅助求交点：IsInsect

//求向量a、b的数量积

double dot(const QPoint &a,const QPoint &b){

return a.x()\*b.x()+a.y()\*b.y();

}

//求直线AB与线段CD的交点，若有交点返回true，Insect代表交点，t代表直线AB与线段CD的交点在线段AB上的位置，运用定比分点公式

bool MainWindow::IsInsect(const QPoint A,const QPoint B,const QPoint C,const QPoint D,QPoint &Insect,double &t)

{

QPoint b=B-A;//向量

QPoint d=D-C;

QPoint c=C-A;

QPoint d\_V(d.y(),-d.x());//d的垂线

if(dot(b,d\_V)==0)return false;//平行

else{

t=dot(c,d\_V)/dot(b,d\_V);

QPoint b\_V(b.y(),-b.x());

double u=dot(c,b\_V)/dot(b,d\_V);

if(u<0||u>1)return false;//线段不相交

else{//线段相交

Insect=A+b\*t;

return true;

}

}

}

### 五、结论

1、裁剪窗口可以是矩形、任意凸多边形、任意凹多边形。

2、相较Sutherland-Hodgman对于每条边都要调用一次裁剪算法来说，Weiler-Athenton算法裁剪思想新颖，方法简洁，裁剪一次完成，与裁剪窗口的边数无关；但是裁剪的算法复杂很多，主要难在如何正确找出交点、判断入点出点端点、将点正确插入原序列。

### 六、分析总结

在实验的过程中，相较于实现算法核心，我个人认为更难的是找交点、判断出入点等细节上的处理问题。计算几何涉及到的情况复杂多变，代码的细节不处理完善的话，就会有很多特殊情况导致程序运行出错。比如我在实验过程中遇到的：绘图时出现直线与另外两条线段都相交在端点处，导致多边形交点的出入点判断错误的情况；多边形端点与出点重合，标记设置错误的情况；直线与两条线段都相交在同一点的情况……因此，虽然Weiler-Athenton算法能实现相对于Sutherland-Hodgman更强大的裁剪功能，但是其算法的细节处理比它要复杂的多。

因此在以后的代码中，除了要理清算法的核心思路，在实现的时候也要尤其注意处理好算法的实现细节，保证程序的健壮性、鲁棒性。

## 实验13 视窗变换

1. 创建一个mainwindow应用程序，保留ui界面，修改窗口标题为“视窗变换”
2. 添加四个label，修改文本显示为窗口宽度，窗口高度，视口宽度，视口高度，对象名分别为label\_ww，label\_wh，label\_vw，label\_vh
3. 添加四个spinBox，对象名修改为spinBox\_ww，spinBox\_wh，spinBox\_vw，spinBox\_vh
4. 打开主窗口头文件，在主窗口类头文件mainwindow.h中添加如下包含文件

#include <QPainter>

#include <QPoint>

#include <QRect>

#include <QVector>

#include <QMouseEvent>

1. 继续添加私有数据成员，成员函数以及槽函数

private:

QPoint wldOrg;//世界坐标系原点

QPoint scrOrg;//屏幕坐标系原点

QRect wldRect;//世界坐标系矩形区域

QRect scrRect;//屏幕坐标系矩形区域

QRect winRect;//窗口

QRect viewRect;//视口

QVector<QRect> wldFigs;//世界坐标系中图形

QVector<QRect> scrFigs;//屏幕坐标系中图形

bool isInWindow;//鼠标是否在窗口内标志

bool isInView;//鼠标是否在视口内标志

QPoint oldPos;//鼠标按下位置

public:

void **clip**();

void **drawRect**(QPainter\* p, QRect rect, QColor clr);

void ***paintEvent***(QPaintEvent\*);

void ***mousePressEvent***(QMouseEvent \*event);

void ***mouseMoveEvent***(QMouseEvent \*event);

void ***mouseReleaseEvent***(QMouseEvent \*event);

private slots:

void **on\_spinBox\_ww\_valueChanged**(int arg1);

void **on\_spinBox\_wh\_valueChanged**(int arg1);

void **on\_spinBox\_vw\_valueChanged**(int arg1);

void **on\_spinBox\_vh\_valueChanged**(int arg1);

1. 在mainwindow.cpp中实现代码如下：

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

isInWindow = false;

isInView = false;

wldOrg = QPoint(50, 120); //记录世界坐标系原点

scrOrg = QPoint(width()/2+50, 120);//记录屏幕坐标系原点

wldRect = QRect(0, 0, width()\*3/8, height()\*5/8);

scrRect = wldRect;

wldRect.translate(wldOrg);

scrRect.translate(scrOrg);

winRect = QRect(wldRect.width()\*1/8,

wldRect.width()\*7/32,

wldRect.width()\*6/8,

wldRect.width()\*18/32);//窗口矩形

viewRect = winRect;//视口矩形

winRect.translate(wldOrg);

viewRect.translate(scrOrg);

//初始化世界坐标系中的图形

wldFigs.append(QRect(wldRect.width()/20,

wldRect.height()\*3/10,

wldRect.width()\*17/20,

wldRect.height()\*4/15));

wldFigs.append(QRect(wldRect.width()\*11/26,

wldRect.height()\*25/70,

wldRect.width()\*5/13,

wldRect.height()\*27/70));

wldFigs.append(QRect(wldRect.width()/10,

wldRect.height()\*2/3,

wldRect.width()\*17/30,

wldRect.height()\*7/30));

wldFigs.append(QRect(wldRect.width()\*3/10,

wldRect.height()\*1/10,

wldRect.width()\*9/20,

wldRect.height()\*11/40));

int i;

for(i=0; i<wldFigs.length(); i++)

wldFigs[i].translate(wldOrg);

//初始化屏幕坐标系中的图形

scrFigs.resize(4);

ui->spinBox\_ww->setMaximum(wldRect.width()-50);

ui->spinBox\_wh->setMaximum(wldRect.height()-50);

ui->spinBox\_vw->setMaximum(scrRect.width()-50);

ui->spinBox\_vh->setMaximum(scrRect.height()-50);

ui->spinBox\_ww->setValue(winRect.width());

ui->spinBox\_wh->setValue(winRect.height());

ui->spinBox\_vw->setValue(viewRect.width());

ui->spinBox\_vh->setValue(viewRect.height());

}

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

{

delete ui;

}

void MainWindow::**clip**()

{

QRect wr;

double sx = static\_cast<double>(viewRect.width())/winRect.width();

double sy = static\_cast<double>(viewRect.height())/winRect.height();

int dx = viewRect.x()-winRect.x();

int dy = viewRect.y()-winRect.y();

int xMin;

int yMin;

int uMin;

int vMin;

for(int i=0;i<wldFigs.length();i++)

{

wr = wldFigs[i].intersected(winRect);

//当前矩形与窗口求交运算，得到裁剪后的矩形

xMin = wr.x()-winRect.x();

//当前矩形左上角相对于窗口左上角x坐标

yMin = wr.y()-winRect.y();

//当前矩形左上角相对于窗口左上角y坐标

uMin = sx\*xMin;//当前矩形左上角相对于视口左上角x坐标

vMin = sy\*yMin;//当前矩形左上角相对于视口左上角y坐标

wr.translate(-xMin, -yMin);//平移至窗口左上角

wr.setWidth(sx\*wr.width());//沿x方向的放缩变换

wr.setHeight(sy\*wr.height());//沿y方向的放缩变换

wr.translate(dx,dy);//平道至视口左上角

wr.translate(uMin, vMin);//平移至视口对应位置

scrFigs[i] = wr;

}

}

void MainWindow::**drawRect**(QPainter\* ptr, QRect rect, QColor clr)

{

ptr->save();

ptr->setPen(clr);

ptr->drawRect(rect);

ptr->restore();

}

void MainWindow::***paintEvent***(QPaintEvent\*)

{

QPainter \*ptr = new QPainter(this);

clip();//对窗口中的图形进行裁剪，并变换到视口中去

//生成视口中的图形数据

//wldFigs -> scrFigs

//显示标题

double x,y,w,h;

x = width()/2-50.0;

y = 80.0;

w = 100.0;

h = 20;

ptr->save();

ptr->setPen(Qt::red);

ptr->drawText(QRectF(x, y, w, h), Qt::AlignCenter,tr("窗口到视口的变换"));

ptr->restore();

//处理世界坐标系

ptr->save();

ptr->setPen(Qt::blue);

x = wldRect.x()+wldRect.width()/2-20.0;

y = wldRect.y()+10.0;

w = 40.0;

ptr->drawText(QRectF(x, y, w, h), Qt::AlignCenter, tr("窗口"));

ptr->setPen(Qt::green);

x = wldRect.x()+wldRect.width()/2-30.0;

y = wldRect.y()+wldRect.height()-30.0;

w = 60.0;

ptr->drawText(QRectF(x, y, w, h), Qt::AlignCenter, tr("世界坐标系"));

ptr->restore();

//处理屏幕坐标系

ptr->save();

ptr->setPen(Qt::blue);

x = scrRect.x()+scrRect.width()/2-20.0;

y = scrRect.y()+10.0;

w = 40.0;

ptr->drawText(QRectF(x, y, w, h), Qt::AlignCenter, tr("视口"));

ptr->setPen(Qt::green);

x = scrRect.x()+scrRect.width()/2-30.0;

y = scrRect.y()+scrRect.height()-30.0;

w = 60.0;

ptr->drawText(QRectF(x, y, w, h), Qt::AlignCenter, tr("屏幕坐标系"));

ptr->restore();

//绘制世界坐标系图形

drawRect(ptr, wldRect, Qt::green);

drawRect(ptr, wldFigs[0],Qt::black);

drawRect(ptr, wldFigs[1],Qt::black);

drawRect(ptr, wldFigs[2],Qt::black);

drawRect(ptr, wldFigs[3],Qt::black);

drawRect(ptr, winRect, Qt::blue);

//绘制屏幕坐标系图形

drawRect(ptr, scrRect, Qt::green);

drawRect(ptr, scrFigs[0], Qt::black);

drawRect(ptr, scrFigs[1], Qt::black);

drawRect(ptr, scrFigs[2], Qt::black);

drawRect(ptr, scrFigs[3], Qt::black);

drawRect(ptr, viewRect, Qt::blue);

delete ptr;

}

void MainWindow::**on\_spinBox\_ww\_valueChanged**(int arg1)

{

winRect.setWidth(arg1);

clip();

update();

}

void MainWindow::**on\_spinBox\_wh\_valueChanged**(int arg1)

{

winRect.setHeight(arg1);

clip();

update();

}

void MainWindow::**on\_spinBox\_vw\_valueChanged**(int arg1)

{

viewRect.setWidth(arg1);

clip();

update();

}

void MainWindow::**on\_spinBox\_vh\_valueChanged**(int arg1)

{

viewRect.setHeight(arg1);

clip();

update();

}

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

{

QCursor csr;

if(event->button()==Qt::LeftButton)

{

csr.setShape(Qt::ClosedHandCursor);

QApplication::setOverrideCursor(csr);

oldPos = event->pos();

if(winRect.contains(event->pos()))

isInWindow=true;

else

isInWindow=false;

if(viewRect.contains(event->pos()))

isInView=true;

else

isInView=false;

}

update();

}

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

{

QPoint newPos = event->pos();

if(isInWindow)

winRect.translate(newPos.x()-oldPos.x(),

newPos.y()-oldPos.y());

if(isInView)

viewRect.translate(newPos.x()-oldPos.x(),

newPos.y()-oldPos.y());

oldPos = newPos;

update();

}

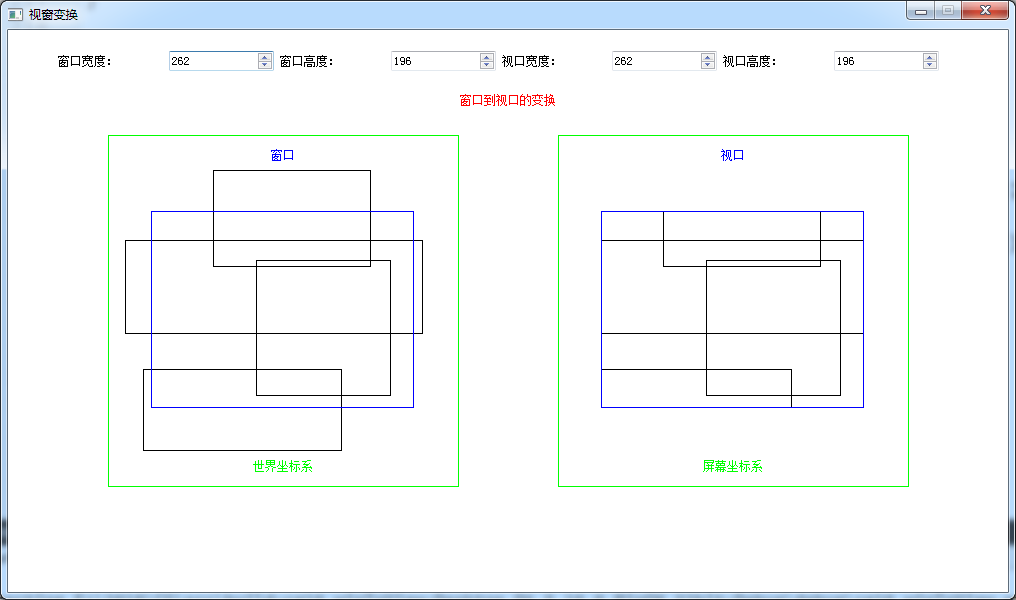
void MainWindow::***mouseReleaseEvent***(QMouseEvent \*event)

{

Q\_UNUSED(event);

QApplication::restoreOverrideCursor();

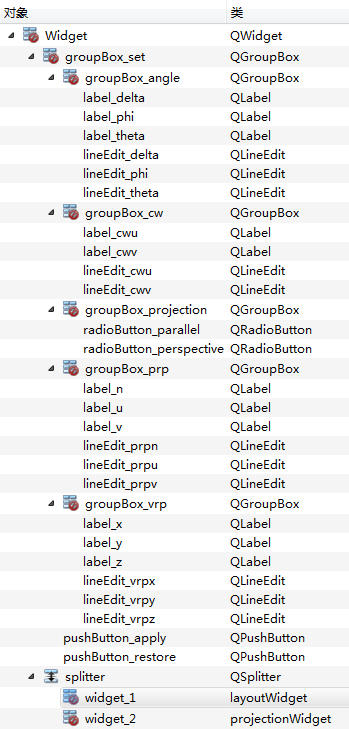
}



## 实验14 3D房屋绘制

1. 创建一个Widget项目，名称为cg10\_house3d，保留ui界面
2. 设置ui的起始尺寸为1000×650
3. 在ui界面上创建一个控制面板，控制面板的参数设定groupBox宽度约为313，并在右边拖入两个Widget控件，名称分别为widget\_1和widget\_2。控制面板及相应对象的名称如下图：

![C:\Users\dm\AppData\Roaming\Tencent\Users\46432248\QQ\WinTemp\RichOle\854Q`H7LYT0](EB9K)FXE]M.png](data:image/png;base64,)



1. 对widget\_1和widget\_2实施垂直分裂布局，布局对象的名称为splitter，选中splitter对象，用鼠标将其向右下拖动，使其充满整个主窗口Widget
2. 添加两个类layoutWidget和projectionWidget，其基类均为QWidget
3. 将widget\_1和widget\_2提升为layoutWidget和projectionWidget
4. 打开layoutWidget类的构造函数，添加如下代码

setPalette(QPalette(QColor(254,254,254)));

setAutoFillBackground(true);

1. 打开projectionWidget类的构造函数，添加如下代码

setPalette(QPalette(QColor(251,251,251)));

setAutoFillBackground(true);

1. 为主窗口类Widget添加函数void ***paintEvent***(QPaintEvent\*)

void Widget::***paintEvent***(QPaintEvent\*)

{

QPainter\* ptr = new QPainter(this);

ui->groupBox\_set->setFixedSize(313, height()-20);

ui->splitter->setFixedSize(width()-318, height()-20);

}

1. 运行程序查看程序界面是否正常
2. 添加3D图形类。新建一个类object3d

//object3d.h

#include <QPainter>

#include <QPoint>

#include <QRectF>

#include <QString>

const double PI = 3.1415926;

class object3d

{

private:

int nProjectionType;// 投影类型, 0:透视投影 1:平行投影

double VRP[3]; //视平面参考点(相对于世界坐标系)

double Theta; //视图参考坐标系与z轴的偏角

double Phi; //视图参考坐标系与y轴的偏角

double Delta; //视图参考坐标系与x轴的偏角

double PRP[3]; //投影参考点(相对于视图参考坐标系)

double CW[2]; //窗口中心点(相对于视图参考坐标系)

double wcHouse[14][3];// 世界坐标系中的房屋坐标，原始数据

double vcHouse[14][3]; //视图参考坐标系中的房屋坐标

int scHouse[14][2]; //屏幕坐标系中的房屋坐标

double lPRP[3]; //布局图所用的投影参考点

double lwcWC[4][3]; //布局图中的世界坐标系(相对于世界坐标系)

int lscWC[4][2]; //布局图中的世界坐标系(相对于屏幕坐标系)

double lwcVRC[4][3]; //布局图中的视图参考坐标系(相对于世界坐标系)

double lvcVRC[4][3];

int lscVRC[4][2]; //布局图中的视图参考坐标系(相对于世界坐标系)

double lwcPRP[2][3]; //布局图中的投影参考点(相对于世界坐标系)

double lvcPRP[2][3];

int lscPRP[2][2]; //布局图中的投影参考点(相对于世界坐标系)

double MT[4][4]; //三维变换矩阵

double MP[4][4]; //投影变换矩阵

public:

object3d();

void **setProjectionType**(int n);

void **setVRP**(double vrp0, double vrp1, double vrp2);

void **setAngle**(double th, double ph, double dt);

void **setPRP**(double prp0, double prp1, double prp2);

void **setCW**(double cw0, double cw1);

void **ShowLayout**(QPainter\* pt);// 显示布局图，供CLayoutView调用

void **ShowProjection**(QPainter\* pt);// 显示投影图，供CProjectionView调用

void **Transform**(double MS[][3], double MO[][3], int nRow, double MT[4][4]);//三维变换

void **Project**(double MS[][3], int MO[][2], int nRow, double MT[4][4]);// 投影变换

void **ToScreen**(int MS[][2], int nRow);// 屏幕变换

void **MatricMultiply**(double M1[4][4], double M2[4][4], double M3[4][4]);// 矩阵乘法

void **GenerateMatricTransform**(double dx, double dy, double dz,

double theta, double phi, double delta, double MT[4][4]);// 构造三维变换矩阵

void **GenerateMatricProject**(double PRP[3], double MP[4][4], int nType);// 投影矩阵

void **DrawHouse**(QPainter\* pt);// 绘制房子

};

//object3d.cpp

#include "object3d.h"

object3d::**object3d**()

:nProjectionType(0)

{

int i, j;

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

//初始化房屋数据

wcHouse[0][0] = 20.0; wcHouse[0][1] = 0.0; wcHouse[0][2] = 0.0;

wcHouse[1][0] = 20.0; wcHouse[1][1] = 0.0; wcHouse[1][2] = 8.0;

wcHouse[2][0] = 20.0; wcHouse[2][1] = 6.0; wcHouse[2][2] = 8.0;

wcHouse[3][0] = 20.0; wcHouse[3][1] = 9.0; wcHouse[3][2] = 4.0;

wcHouse[4][0] = 20.0; wcHouse[4][1] = 6.0; wcHouse[4][2] = 0.0;

wcHouse[5][0] = 28.0; wcHouse[5][1] = 4.0; wcHouse[5][2] = 8.0;

wcHouse[6][0] = 28.0; wcHouse[6][1] = 0.0; wcHouse[6][2] = 8.0;

for (i = 7; i<14; i++)

{

wcHouse[i][0] = wcHouse[i - 7][0] + 20;

wcHouse[i][1] = wcHouse[i - 7][1];

wcHouse[i][2] = wcHouse[i - 7][2];

wcHouse[i][3] = 1.0;

}

wcHouse[12][0] = wcHouse[5][0] + 4;

wcHouse[13][0] = wcHouse[6][0] + 4;

//调整房屋大小

int nScale = 5; //缩放比例

for (i = 0; i<14; i++)

{

wcHouse[i][0] = nScale \* wcHouse[i][0];

wcHouse[i][1] = nScale \* wcHouse[i][1];

wcHouse[i][2] = nScale \* wcHouse[i][2];

}

//初始化世界坐标系

for (i = 0; i<4; i++)

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

lwcWC[i][j] = 0.0;

lwcWC[0][0] = 300;

lwcWC[1][1] = 100;

lwcWC[2][2] = 100;

//初始化视图参考坐标系

for (i = 0; i<4; i++)

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

lwcVRC[i][j] = 0.0;

lwcVRC[0][0] = 30;

lwcVRC[1][1] = 30;

lwcVRC[2][2] = 30;

lPRP[0] = 3000.0;

lPRP[1] = 1000.0;

lPRP[2] = 6000.0;

//初始化投影参数

VRP[0] = 0;

VRP[1] = 0;

VRP[2] = 0;

Theta = 0;

Phi = 0;

Delta = 0;

PRP[0] = 250.0;

PRP[1] = 100.0;

PRP[2] = 250.0;

CW[0] = 0.0;

CW[1] = 0.0;

}

void object3d::**setProjectionType**(int n)

{

nProjectionType = n;

}

void object3d::**setVRP**(double vrp0, double vrp1, double vrp2)

{

VRP[0] = vrp0;

VRP[1] = vrp1;

VRP[2] = vrp2;

}

void object3d::**setAngle**(double th, double ph, double dt)

{

Theta = th;

Phi = ph;

Delta = dt;

}

void object3d::**setPRP**(double prp0, double prp1, double prp2)

{

PRP[0] = prp0;

PRP[1] = prp1;

PRP[2] = prp2;

}

void object3d::**setCW**(double cw0, double cw1)

{

CW[0] = cw0;

CW[1] = cw1;

}

//显示布局图，供layoutWidget调用

void object3d::**ShowLayout**(QPainter\* pt)

{

int i, j;

//计算投影参考点的世界坐标

for (i = 0; i<2; i++)

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

{

lwcPRP[i][j] = VRP[j] + PRP[j];

lwcPRP[i][j] = VRP[j] + PRP[j];

}

lwcPRP[1][1] = 0.0;

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

//三维变换

GenerateMatricTransform(VRP[0], VRP[1], VRP[2], Theta, Phi, Delta, MT);

//生成三维变换矩阵

Transform(lwcVRC, lvcVRC, 4, MT); //视图参考坐标系

Transform(lwcPRP, lvcPRP, 2, MT); //投影参考点

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

//投影变换

GenerateMatricProject(lPRP, MP, 0);//生成布局图所用的投影矩阵

Project(wcHouse, scHouse, 14, MP); //房屋

Project(lwcWC, lscWC, 4, MP); //世界坐标系

Project(lvcVRC, lscVRC, 4, MP); //视图参考坐标系

Project(lvcPRP, lscPRP, 2, MP); //投影参考点

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

//屏幕变换

ToScreen(scHouse, 14); //房屋

ToScreen(lscWC, 4); //世界坐标系

ToScreen(lscVRC, 4); //视图参考坐标系

ToScreen(lscPRP, 2); //投影参考点

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

//显示图形

DrawHouse(pt);//画房屋

//画世界坐标系

for (i = 0; i<3; i++)

pt->drawLine(QPoint(lscWC[3][0], lscWC[3][1]), QPoint(lscWC[i][0], lscWC[i][1]));

pt->drawText(QRectF(lscWC[0][0], lscWC[0][1], 5, 20), Qt::AlignCenter, "x");

pt->drawText(QRectF(lscWC[1][0], lscWC[1][1], 5, 20), Qt::AlignCenter, "y");

pt->drawText(QRectF(lscWC[2][0], lscWC[2][1], 5, 20), Qt::AlignCenter, "z");

//画视图参考坐标系

pt->save();

pt->setPen(QColor(255,0,0));

for (i = 0; i<3; i++)

pt->drawLine(QPoint(lscVRC[i][0], lscVRC[i][1]), QPoint(lscVRC[3][0], lscVRC[3][1]));

pt->drawText(QRectF(lscVRC[0][0], lscVRC[0][1], 5, 20), Qt::AlignCenter, "u");

pt->drawText(QRectF(lscVRC[1][0], lscVRC[1][1], 5, 20), Qt::AlignCenter, "v");

pt->drawText(QRectF(lscVRC[2][0], lscVRC[2][1], 5, 20), Qt::AlignCenter, "n");

pt->restore();

//画投影参考点

pt->save();

pt->setPen(QColor(0, 0, 255));

pt->drawEllipse(lscPRP[0][0] - 3, lscPRP[0][1] - 3, 6, 6);

pt->drawLine(QPoint(lscPRP[0][0], lscPRP[0][1]), QPoint(lscPRP[1][0], lscPRP[1][1]));

pt->drawText(QRectF(lscPRP[0][0], lscPRP[0][1], 10, 20), Qt::AlignCenter, "PRP");

pt->restore();

}

//显示投影图，供projectionWidget调用

void object3d::**ShowProjection**(QPainter\* pt)

{

//三维变换

GenerateMatricTransform(-VRP[0], -VRP[1], -VRP[2], -Theta, -Phi, -Delta, MT);

Transform(wcHouse, vcHouse, 14, MT);

//投影变换

GenerateMatricProject(PRP, MP, nProjectionType);

Project(vcHouse, scHouse, 14, MP);

//屏幕变换

ToScreen(scHouse, 14);

//显示图形

DrawHouse(pt);

}

//三维变换

void object3d::**Transform**(double MS[][3], double MO[][3], int nRow, double MT[4][4])

{

int i, j;

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

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

MO[i][j] = MT[j][0] \* MS[i][0] + MT[j][1] \* MS[i][1] +MT[j][2] \* MS[i][2] + MT[j][3] \* 1.0;

}

//投影变换

void object3d::**Project**(double MS[][3], int MO[][2], int nRow, double MT[4][4])

{

int i;

double xObs, yObs, zObs, wObs;

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

{

xObs = MS[i][0] \* MP[0][0] + MS[i][2] \* MP[0][2];

yObs = MS[i][1] \* MP[1][1] + MS[i][2] \* MP[1][2];

zObs = 0.0;

wObs = MS[i][2] \* MP[3][2] + 1.0;

MO[i][0] = (int)(xObs / wObs);

MO[i][1] = (int)(yObs / wObs);

}

}

// 屏幕变换

void object3d::**ToScreen**(int MS[][2], int nRow)

{

int i;

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

MS[i][1] = -MS[i][1];

}

// 矩阵乘法

void object3d::**MatricMultiply**(double M1[4][4], double M2[4][4], double M3[4][4])

{

for (int i = 0; i<4; i++)

for (int j = 0; j<4; j++)

M3[i][j] = M1[i][0] \* M2[0][j] + M1[i][1] \* M2[1][j] + M1[i][2] \* M2[2][j] + M1[i][3] \* M2[3][j];

}

// 构造三维变换矩阵

void object3d::**GenerateMatricTransform**(double dx, double dy, double dz, double theta, double phi, double delta, double MT[4][4])

{

//将角度转换成弧度

double \_theta = theta\*PI / 180;

double \_phi = phi\*PI / 180;

double \_delta = delta\*PI / 180;

double T[4][4], Rz[4][4], Ry[4][4], Rx[4][4], temp[4][4];

//初始化矩阵

int i, j;

for (i = 0; i<4; i++)

for (j = 0; j<4; j++)

{

T[i][j] = 0.0;

Ry[i][j] = 0.0;

Rx[i][j] = 0.0;

Rz[i][j] = 0.0;

temp[i][j] = 0.0;

MT[i][j] = 0.0;

}

//平移变换

T[0][0] = 1.0;

T[0][3] = dx;

T[1][1] = 1.0;

T[1][3] = dy;

T[2][2] = 1.0;

T[2][3] = dz;

T[3][3] = 1.0;

//绕z轴旋转

Rz[0][0] = cos(\_theta);

Rz[0][1] = -sin(\_theta);

Rz[1][0] = sin(\_theta);

Rz[1][1] = cos(\_theta);

Rz[2][2] = 1.0;

Rz[3][3] = 1.0;

//绕y轴旋转

Ry[0][0] = cos(\_phi);

Ry[0][2] = sin(\_phi);

Ry[1][1] = 1.0;

Ry[2][0] = -sin(\_phi);

Ry[2][2] = cos(\_phi);

Ry[3][3] = 1.0;

//绕x轴旋转

Rx[0][0] = 1.0;

Rx[1][1] = cos(\_delta);

Rx[1][2] = -sin(\_delta);

Rx[2][1] = sin(\_delta);

Rx[2][2] = cos(\_delta);

Rx[3][3] = 1.0;

//生成三维变换矩阵

//MT = Rx \* Ry \* Rz \* T

MatricMultiply(T, Rz, MT);

MatricMultiply(MT, Ry, temp);

MatricMultiply(temp, Rx, MT);

}

//投影矩阵

void object3d::**GenerateMatricProject**(double PRP[3], double MP[4][4], int nType)

{

//透视投影

if (nType == 0)

{

MP[0][0] = 1.0;

MP[0][1] = 0.0;

MP[0][2] = -PRP[0] / PRP[2];

MP[0][3] = 0.0;

MP[1][0] = 0.0;

MP[1][1] = 1.0;

MP[1][2] = -PRP[1] / PRP[2];

MP[1][3] = 0.0;

MP[2][0] = 0.0;

MP[2][1] = 0.0;

MP[2][2] = 0.0;

MP[2][3] = 0.0;

MP[3][0] = 0.0;

MP[3][1] = 0.0;

MP[3][2] = -1.0 / PRP[2];

MP[3][3] = 1.0;

}

//平行投影

else if (nType == 1)

{

MP[0][0] = 1.0;

MP[0][1] = 0.0;

MP[0][2] = -(PRP[0] - CW[0]) / PRP[2];

MP[0][3] = 0.0;

MP[1][0] = 0.0;

MP[1][1] = 1.0;

MP[1][2] = -(PRP[1] - CW[1]) / PRP[2];

MP[1][3] = 0.0;

MP[2][0] = 0.0;

MP[2][1] = 0.0;

MP[2][2] = 0.0;

MP[2][3] = 0.0;

MP[3][0] = 0.0;

MP[3][1] = 0.0;

MP[3][2] = 0.0;

MP[3][3] = 1.0;

}

}

//绘制房子

void object3d::**DrawHouse**(QPainter\* pt)

{

int i;

//画左侧墙壁

for (i = 0; i<5; i++)

pt->drawLine(QPoint(scHouse[i][0], scHouse[i][1]), QPoint(scHouse[(i + 1) % 5][0], scHouse[(i + 1) % 5][1]));

//画右侧墙壁

for (i = 7; i<11; i++)

pt->drawLine(QPoint(scHouse[i][0], scHouse[i][1]), QPoint(scHouse[i + 1][0], scHouse[i + 1][1]));

pt->drawLine(QPoint(scHouse[i][0], scHouse[i][1]), QPoint(scHouse[7][0], scHouse[7][1]));

//画横梁

for (i = 0; i<6; i++)

pt->drawLine(QPoint(scHouse[i][0], scHouse[i][1]), QPoint(scHouse[i + 7][0], scHouse[i + 7][1]));

//画门

pt->drawLine(QPoint(scHouse[5][0], scHouse[5][1]), QPoint(scHouse[6][0], scHouse[6][1]));

pt->drawLine(QPoint(scHouse[12][0], scHouse[12][1]), QPoint(scHouse[13][0], scHouse[13][1]));

}

1. 在layoutWidget类和projectWidget类中分别添加3D图形类成员变量

public:

object3d o3d;

1. 在主窗口类的构造函数中添加如下代码，用以初始化控制面板各编辑框

ui->lineEdit\_vrpx->setText("0");

ui->lineEdit\_vrpy->setText("0");

ui->lineEdit\_vrpz->setText("0");

ui->lineEdit\_theta->setText("0");

ui->lineEdit\_phi->setText("0");

ui->lineEdit\_delta->setText("0");

ui->lineEdit\_prpu->setText("250");

ui->lineEdit\_prpv->setText("100");

ui->lineEdit\_prpn->setText("250");

ui->lineEdit\_cwu->setText("0");

ui->lineEdit\_cwv->setText("0");

1. 为按钮“应用”、“恢复”添加槽函数

private slots:

void **on\_pushButton\_apply\_clicked**();

void **on\_pushButton\_restore\_clicked**();

1. 实现上述槽函数

void Widget::**on\_pushButton\_apply\_clicked**()

{

if(ui->radioButton\_perspective->isChecked())

{

ui->widget\_1->o3d.setProjectionType(0);

ui->widget\_2->o3d.setProjectionType(0);

}

else

{

ui->widget\_1->o3d.setProjectionType(1);

ui->widget\_2->o3d.setProjectionType(1);

}

ui->widget\_1->o3d.setVRP(ui->lineEdit\_vrpx->text().toInt(),

ui->lineEdit\_vrpy->text().toInt(),

ui->lineEdit\_vrpz->text().toInt());

ui->widget\_2->o3d.setVRP(ui->lineEdit\_vrpx->text().toInt(),

ui->lineEdit\_vrpy->text().toInt(),

ui->lineEdit\_vrpz->text().toInt());

ui->widget\_1->o3d.setAngle(ui->lineEdit\_theta->text().toInt(),

ui->lineEdit\_phi->text().toInt(),

ui->lineEdit\_delta->text().toInt());

ui->widget\_2->o3d.setAngle(ui->lineEdit\_theta->text().toInt(),

ui->lineEdit\_phi->text().toInt(),

ui->lineEdit\_delta->text().toInt());

ui->widget\_1->o3d.setPRP(ui->lineEdit\_prpu->text().toInt(),

ui->lineEdit\_prpv->text().toInt(),

ui->lineEdit\_prpn->text().toInt());

ui->widget\_2->o3d.setPRP(ui->lineEdit\_prpu->text().toInt(),

ui->lineEdit\_prpv->text().toInt(),

ui->lineEdit\_prpn->text().toInt());

ui->widget\_1->o3d.setCW(ui->lineEdit\_cwu->text().toInt(),

ui->lineEdit\_cwv->text().toInt());

ui->widget\_2->o3d.setCW(ui->lineEdit\_cwu->text().toInt(),

ui->lineEdit\_cwv->text().toInt());

ui->widget\_1->update();

ui->widget\_2->update();

}

void Widget::**on\_pushButton\_restore\_clicked**()

{

ui->lineEdit\_vrpx->setText("0");

ui->lineEdit\_vrpy->setText("0");

ui->lineEdit\_vrpz->setText("0");

ui->lineEdit\_theta->setText("0");

ui->lineEdit\_phi->setText("0");

ui->lineEdit\_delta->setText("0");

ui->lineEdit\_prpu->setText("250");

ui->lineEdit\_prpv->setText("100");

ui->lineEdit\_prpn->setText("250");

ui->lineEdit\_cwu->setText("0");

ui->lineEdit\_cwv->setText("0");

ui->widget\_1->o3d.setProjectionType(0);

ui->widget\_2->o3d.setProjectionType(0);

ui->widget\_1->o3d.setVRP(ui->lineEdit\_vrpx->text().toInt(),

ui->lineEdit\_vrpy->text().toInt(),

ui->lineEdit\_vrpz->text().toInt());

ui->widget\_2->o3d.setVRP(ui->lineEdit\_vrpx->text().toInt(),

ui->lineEdit\_vrpy->text().toInt(),

ui->lineEdit\_vrpz->text().toInt());

ui->widget\_1->o3d.setAngle(ui->lineEdit\_theta->text().toInt(),

ui->lineEdit\_phi->text().toInt(),

ui->lineEdit\_delta->text().toInt());

ui->widget\_2->o3d.setAngle(ui->lineEdit\_theta->text().toInt(),

ui->lineEdit\_phi->text().toInt(),

ui->lineEdit\_delta->text().toInt());

ui->widget\_1->o3d.setPRP(ui->lineEdit\_prpu->text().toInt(),

ui->lineEdit\_prpv->text().toInt(),

ui->lineEdit\_prpn->text().toInt());

ui->widget\_2->o3d.setPRP(ui->lineEdit\_prpu->text().toInt(),

ui->lineEdit\_prpv->text().toInt(),

ui->lineEdit\_prpn->text().toInt());

ui->widget\_1->o3d.setCW(ui->lineEdit\_cwu->text().toInt(),

ui->lineEdit\_cwv->text().toInt());

ui->widget\_2->o3d.setCW(ui->lineEdit\_cwu->text().toInt(),

ui->lineEdit\_cwv->text().toInt());

ui->widget\_1->update();

ui->widget\_2->update();

}

1. 分别为layoutWidget和projectionWidget类添加重绘事件处理函数

void ***paintEvent***(QPaintEvent\*);

1. 实现layoutWidget和projectionWidget类的重绘事件处理函数

void layoutWidget::***paintEvent***(QPaintEvent\*)

{

QPainter\* ptr = new QPainter(this);

ptr->translate(52, 102);

o3d.ShowLayout(ptr);

}

void projectionWidget::***paintEvent***(QPaintEvent\*)

{

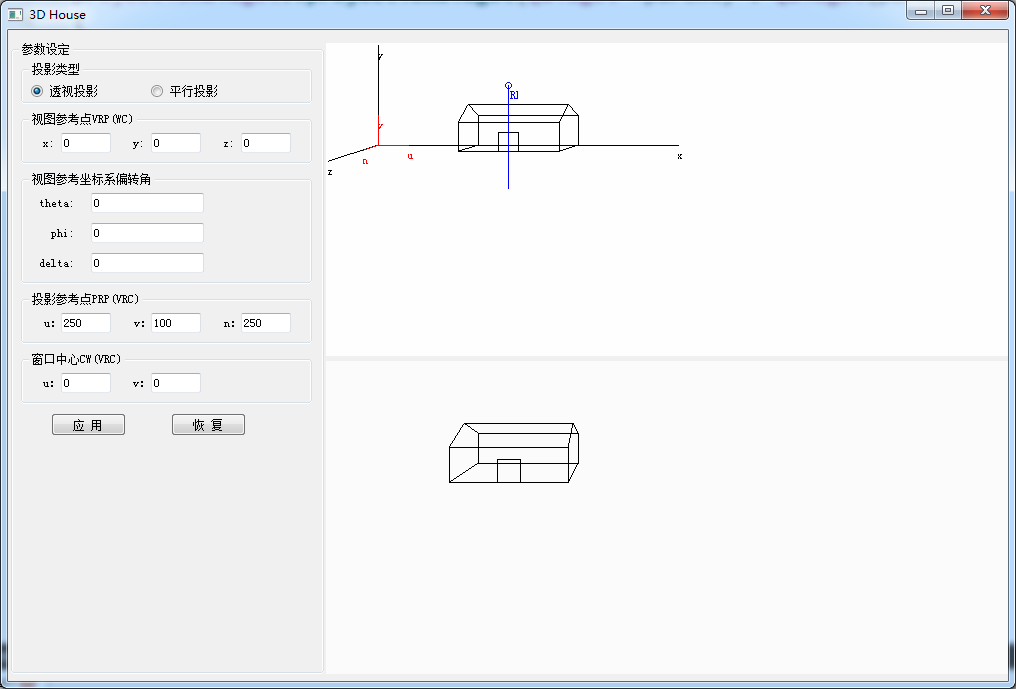
QPainter\* ptr = new QPainter(this);

ptr->translate(52, 102);

o3d.ShowProjection(ptr);

}

1. 程序运行结果如下



## 实验15 金字塔

1. 建立工程cg11\_pyramid，Widget应用程序，保留ui界面
2. 在cg11\_pyramid.pro文件加入代码

QT += opengl

LIBS = -lopengl32 -lglu32 -lglut

1. 打开widget.h文件，添加代码

#include <QGLWidget>

#include <QtMath>

const double PI = 3.14159265;

1. 将Widget基类设置为QGLWidge

class Widget : public QGLWidget

1. 为Widget添加函数

protected:

void ***paintGL***();

1. 修改Widget构造函数

Widget::**Widget**(QWidget \*parent) :

QGLWidget(parent),

ui(new Ui::Widget)

{

ui->setupUi(this);

setFormat(QGLFormat(QGL::DoubleBuffer | QGL::DepthBuffer));

}

1. 实现***paintGL***()代码

void Widget::***paintGL***()

{

glClearColor(0.5, 0.5, 0.5, 0.0);//设置背景色

glClearDepth(1.0f);//初始化深度

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

glEnable(GL\_DEPTH\_TEST);//开启深度测试

glShadeModel(GL\_SMOOTH);//平滑阴影模式

int w = width();

int h = height();

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);//选择投影矩阵

glLoadIdentity();

gluPerspective(60,GLfloat(w)/h, 0.1, 100.0);

GLfloat fov = 60;//眼睛上下睁开的角度

GLfloat zNear = 0.1;//前裁剪面

GLfloat zFar = 100.0;//后裁剪面

GLfloat rFov = fov \* PI / 180.0;

glFrustum( -zNear \* qTan( rFov / 2.0 ) \* GLfloat(w)/h,

zNear \* qTan( rFov / 2.0 ) \* GLfloat(w)/h,

-zNear \* qTan( rFov / 2.0 ),

zNear \* qTan( rFov / 2.0 ),

zNear, zFar );

glMatrixMode(GL\_MODELVIEW);//选择模式矩阵

glLoadIdentity();//重置模式矩阵

//设置光照(定义一个位于左上方的白色定位光源)

GLfloat lmodel\_ambient[] = { 1.0f,1.0f,1.0f,1.0f };//定义环境光

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, lmodel\_ambient);//设置环境光

GLfloat light0\_ambient[] = { 1.0,1.0,1.0,1.0 };//定义光源环境光

GLfloat light0\_diffuse[] = { 1.0,1.0,1.0,1.0 };//定义光源散射光

GLfloat light0\_specular[] = { 1.0,1.0,1.0,1.0 };//定义光源反射光

GLfloat light0\_position[] = { 15.0,15.0,15.0,10.0 };//定义光源位置

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light0\_ambient);//设置光源环境光

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light0\_diffuse);//设置光源散射光

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light0\_specular);//设置光源反射光

glLightfv(GL\_LIGHT0, GL\_POSITION, light0\_position);//设置光源位置

glEnable(GL\_LIGHT0);//启用光源

glEnable(GL\_LIGHTING);//启用光照效果

//定义黄铜材质

GLfloat brass\_ambient[] = { 0.329412f,0.223529f,0.027451f,1.0f };//定义材质环境光

GLfloat brass\_diffuse[] = { 0.780392f,0.568627f,0.113725f,1.0f };//定义材质散射光

GLfloat brass\_specular[] = { 0.992157f,0.941176f,0.807843f,1.0f };//定义材质反射光

GLfloat brass\_sinines[] = { 100.0f };//定义材质镜面反射强度

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, brass\_ambient);//设置材质环境光

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, brass\_diffuse);//设置材质散射光

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, brass\_specular);//设置材质反射光

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SHININESS, brass\_sinines);//设置材质镜面反射强度

//设置坐标系统

glTranslatef(0.0f, -0.1f, -2.0f);//将坐标系统下移0.1，后移2.0

glRotatef(45, 1.0, 0.0, 0.0);//将坐标系统绕x轴逆时针旋转45度

glRotatef(30, 0.0, -1.0, 0.0);//将坐标系统绕y轴顺时针旋转30度

//绘制金字塔

GLfloat pyramid[][3] = {

{ 0.0,1.0,0.0 },

{ 0.5,0.0,0.5 },

{ -0.5,0.0,0.5 },

{ -0.5,0.0,-0.5 },

{ 0.5,0.0,-0.5 }

};

glBegin(GL\_TRIANGLE\_FAN);//使用扇形三角形模式

glNormal3f(0.0, 0.447214f, 0.894427f);//前平面法向量

glVertex3fv(pyramid[0]);//前平面

glVertex3fv(pyramid[1]);

glVertex3fv(pyramid[2]);

glNormal3f(-0.894427f, 0.447214f, 0.0);//左平面法向量

glVertex3fv(pyramid[3]);//左平面

glNormal3f(0.0, 0.447214f, -0.894427f);//后平面法向量

glVertex3fv(pyramid[4]);//后平面

glNormal3f(0.894427f, 0.447214f, 0.0);//右平面法向量

glVertex3fv(pyramid[1]);//右平面

glEnd();//结束绘制

//定义塑料材质

GLfloat plastic\_ambient[] = { 0.0f,0.0f,0.00f,1.0f };//定义材质环境光

GLfloat plastic\_diffuse[] = { 0.7f,0.85f,0.7f,1.0f };//定义材质散射光

GLfloat plastic\_specular[] = { 0.75f,0.75f,0.75f,1.0f };//定义材质反射光

GLfloat plastic\_sinines[] = { 10.0f };//定义材质镜面反射强度

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, plastic\_ambient);//设置材质环境光

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, plastic\_diffuse);//设置材质散射光

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, plastic\_specular);//设置材质反射光

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SHININESS, plastic\_sinines);//设置材质镜面反射强度

//定义纹理图案(类似于国际象棋的黑白棋盘)

#define textureWidth 64//定义纹理图案宽度

#define textureHeight 64//定义纹理图案高度

GLubyte texture[textureWidth][textureHeight][3];//定义纹理图案数组，3分量

GLubyte c;

for (int i = 0; i<textureWidth; i++)//生成纹理图案数组

{

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

{

c = (((i & 8) == 0) ^ ((j & 8) == 0)) \* 255;

texture[i][j][0] = c;

texture[i][j][1] = c;

texture[i][j][2] = c;

}

}

glTexImage2D(//定义纹理

GL\_TEXTURE\_2D,//二维纹理

0,//纹理级别，单重纹理

3,//3分量模式，包含R、G、B信息

textureWidth,//纹理图案宽度

textureHeight,//纹理图案高度

0,//纹理边界宽度

GL\_RGB,//纹理图案数据种类

GL\_UNSIGNED\_BYTE,//纹理图案数据存储类型

&texture[0][0][0]//纹理图案数据起始地址

);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);//设置纹理在s方向上重复

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);//设置纹理在t方向上重复

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);//设置纹理放大方式

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);//设置纹理缩小方式

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);//设置调节映射方式

glEnable(GL\_TEXTURE\_2D);//启用纹理映射

GLfloat texpoints[2][2][2] =//定义纹理坐标(s,t)

{

{ { 0.0,0.0 },{ 1.0,0.0 } },

{ { 0.0,1.0 },{ 1.0,1.0 } }

};

glMap2f(//定义纹理坐标求值器

GL\_MAP2\_TEXTURE\_COORD\_2,//求值器生成值种类，选用纹理坐标

0, 1, //u的取值范围

2,//u的跨度 = u方向上两个值之间的间隔 = texpoints数组第一个维度

2, //u的阶数 = texpoints数组第二个维度

0, 1, //v的取值范围

4, //v的跨度 = v方向上两个值之间的间隔 = texpoints数组第一个维度\*第二个纬度

2, //v的阶数 = texpoints数组第三个维度

&texpoints[0][0][0] //控制点矩阵的指针

);

glEnable(GL\_MAP2\_TEXTURE\_COORD\_2);//激活纹理坐标求值器

//定义曲面控制点

#define ustride 3//u的跨度

#define uorder 6//u的阶数

#define vstride (ustride \* uorder)//v的跨度

#define vorder 5//v的阶数

GLdouble points[vorder][uorder][ustride] =//定义顶点坐标(x,y,z)

{

{

{-2.00f,0.4f,-1.00f },

{ -1.20f,0.3f,-1.00f },

{ -0.40f,0.4f,-1.00f },

{ 0.40f,-0.3f,-1.00f },

{ 1.20f,0.0f,-1.00f },

{ 2.00f,0.5f,-1.00f }

},

{

{ -2.00f,0.1f,-0.60f },

{ -1.20f,0.0f,-0.60f },

{ -0.40f,0.5f,-0.60f },

{ 0.40f,0.0f,-0.60f },

{ 1.20f,0.0f,-0.60f },

{ 2.00f,0.0f,-0.60f }

},

{

{ -2.00f,0.3f,-0.20f },

{ -1.20f,0.3f,-0.20f },

{ -0.40f,0.0f,-0.20f },

{ 0.40f,0.0f,-0.20f },

{ 1.20f,0.0f,-0.20f },

{ 2.00f,0.5f,-0.20f }

},

{

{ -2.00f,0.1f,0.50f },

{ -1.20f,0.2f,0.20f },

{ -0.40f,0.0f,0.50f },

{ 0.40f,0.0f,0.20f },

{ 1.20f,0.6f,0.60f },

{ 2.00f,0.0f,0.40f }

},

{

{ -2.00f,-0.5f,1.00f },

{ -1.20f,0.6f,1.00f },

{ -0.40f,0.0f,1.00f },

{ 0.40f,0.0f,1.00f },

{ 1.20f,0.1f,1.00f },

{ 2.00f,0.5f,1.00f }

}

};

glMap2d(//定义顶点坐标求值器

GL\_MAP2\_VERTEX\_3,

0, 1, ustride, uorder,

0, 1, vstride, vorder,

&points[0][0][0]

);

glEnable(GL\_MAP2\_VERTEX\_3);//激活顶点坐标求值器

//绘制曲面

glPushMatrix();//将当前模视矩阵压栈，防止当前变换影响原有的变换模式

glLoadIdentity();//重置模视矩阵

glTranslatef(0.0f, -0.1f, -2.2f);//将坐标系统下移0.1，后移2.2

glRotatef(45, 1.0, 0.0, 0.0);//将坐标系统绕x轴逆时针旋转45度

glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0);//定义二维均布网格

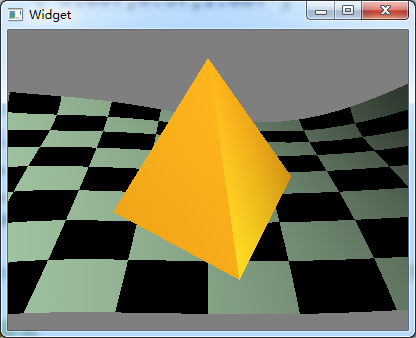
glEvalMesh2(GL\_FILL, 0, 20, 0, 20);//生成二维多边形网格

glDisable(GL\_TEXTURE\_2D);//禁用纹理映射

glPopMatrix();//将模视矩阵出栈

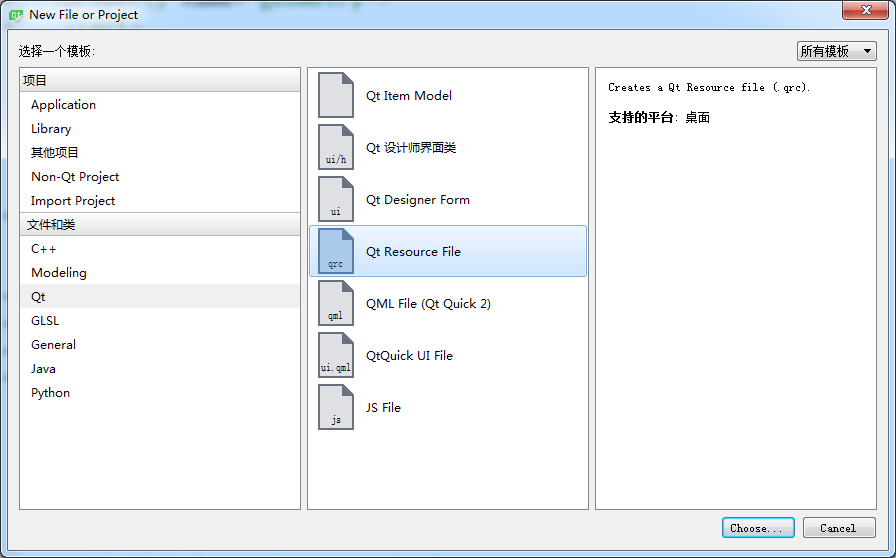
glFlush();

}

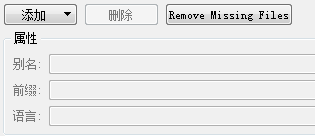


## 实验16 交互技术应用

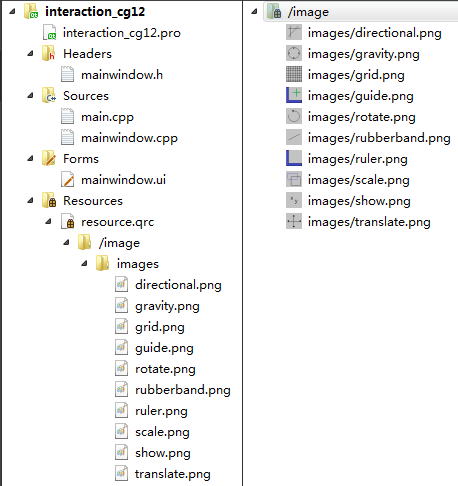
1. 创建一个项目cg12\_interaction，类型为mainwindow，保留ui界面
2. 添加资源。选择文件-新建-Qt资源



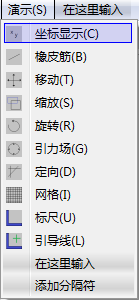
1. 资源的名称这里命名为Resources，自动产生资源文件名Resources.qrc
2. 在当前目录cg12\_interaction中创建一个文件夹images，并将实现制作好的位图文件directional.png，gravity.png，grid.png，guide.png，rotate.png，rubberband.png，ruler.png，scale.png，show.png，translate.png复制到该目录下
3. 在资源属性面板上单击“添加”——“添加前缀”——“image”



1. 在单击“添加文件”，将images中的文件全部添加进来



1. 回到ui设计界面，将主窗体宽度高度设置为800×600
2. 添加菜单资源。主菜单为：演示(&S)。由于版本的缘故，汉字无法输入，可以采用粘贴的办法输入各菜单项，注意输入完后回车确认，还有，各个括号为半角字符。各菜单项依次为：坐标显示(&C)，橡皮筋(&B)，移动(&T)，缩放(&S)，旋转(&R)，引力场(&G)，定向(&D)，网格(&I)，标尺(&U)，引导线(&L)



1. 修改各菜单项属性，为其指定相应的位图和快捷键。双击各个action editor项如action\_C等即可



1. 双击action\_C，在弹出的编辑动作对话框中，添加图标资源show.png，添加快捷键Ctrl+C



1. 其余菜单项动作编辑类似



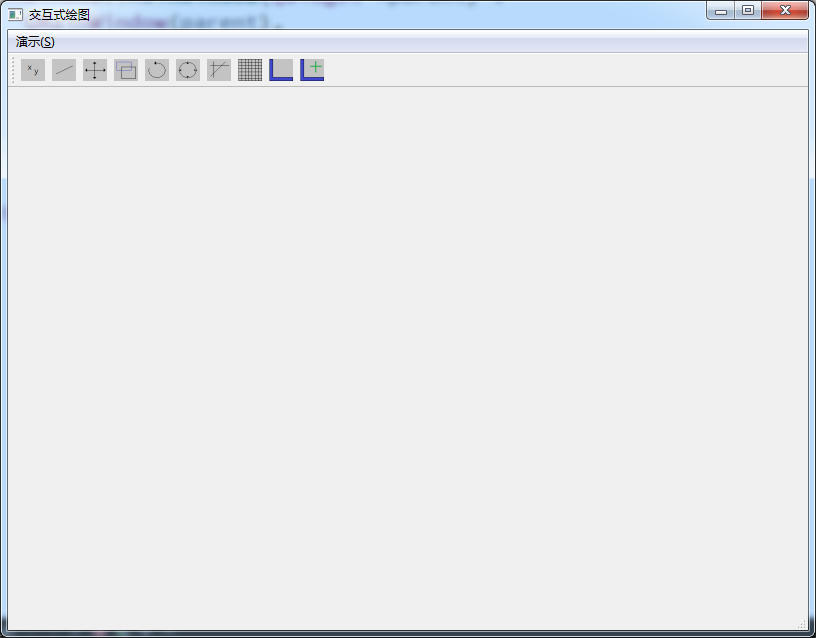
1. 鼠标选中底部菜单属性动作项，按住鼠标将其拖放当工具条上，形成工具条资源

C:\Users\dm\AppData\Roaming\Tencent\Users\46432248\QQ\WinTemp\RichOle\P_2]YZ(]NG]CC`(C(M{G9BQ.png

1. 打开主窗口构造函数，添加一行代码

setWindowTitle(tr("交互式绘图"));

1. 运行程序，查看效果



1. 往中心窗口区拖入两个Widget控件，分别为widget1和widget2，其初始大小手工拖动大致设置一下即可
2. 建立两个类figureWidget和textWidget，基类均为QWidget，将widget1提升为textWidget，widget2提升为figureWidget
3. 对widget1和widget2进行水平分裂布局，布局对象为splitter
4. 在textWidget的构造函数中添加窗口初始化代码

textWidget::**textWidget**(QWidget \*parent) : QWidget(parent)

{

setPalette(QPalette(QColor(251,251,251)));

setAutoFillBackground(true);

}

1. 在figureWidget的构造函数中添加窗口初始化代码

figureWidget::**figureWidget**(QWidget \*parent) : QWidget(parent)

{

setPalette(QPalette(QColor(254,254,254)));

setAutoFillBackground(true);

}

1. 在主窗口类mainwindow.h中添加代码

#include <QPainter>

public:

void ***paintEvent***(QPaintEvent\*);

1. 在主窗口mainwindow.cpp中添加代码以自适应窗口尺寸的变化

void MainWindow::***paintEvent***(QPaintEvent\*)

{

QPainter\* pt = new QPainter(this);

ui->splitter->setFixedSize(width(), height());

//ui->widget1->setFixedSize(width()/3, height());

//ui->widget2->setFixedSize(width()\*2/3, height());

}

1. 新建直线类Line

//line.h

#ifndef LINE\_H

#define LINE\_H

#include <QObject>

#include <QPoint>

#include <QPolygon>

#include <QPainter>

#include <QtMath>

#include <QVector>

const double PI = 3.1415926;

class Line

{

private:

bool initStart;//起点初始化标志

bool initEnd;//终点初始化标志

QPoint start;//线段起点

QPoint end;//线段终点

int gi;//引力场强度

QPolygon gravity;//引力场区域

public:

Line();

Line(QPoint start, QPoint end);

void **setStart**(QPoint start);//设置起点

void **setEnd**(QPoint end);//设置终点

QPoint **getStart**();//获取起点

QPoint **getEnd**();//获取终点

QPoint **getClosestPoint**(QPoint point);//返回线段上离指定点最近的点

bool **ptInGravity**(QPoint point);//判断指定点是否在线段的引力场内

void **setGravityIntensity**(int intensity);//设置引力强度：引力场的范围大小

void **drawGravity**(QPainter\* pt);//绘制引力场

void **draw**(QPainter\* pt);//绘制线段

protected:

void **initGravity**();//初始化引力场

};

#endif // LINE\_H

//line.cpp

#include "line.h"

Line::**Line**()

: initStart(false)

, initEnd(false)

, gi(3)

{

}

Line::**Line**(QPoint start, QPoint end)

: start(start)

, end(end)

, initStart(false)

, initEnd(false)

, gi(3)

{

initGravity();

}

void Line::**setStart**(QPoint start)

{

this->start = start;

initStart = true;

initGravity();

}

void Line::**setEnd**(QPoint end)

{

this->end = end;

initEnd = true;

initGravity();

}

QPoint Line::**getStart**()

{

return start;

}

QPoint Line::**getEnd**()

{

return end;

}

//绘线

void Line::**draw**(QPainter\* pt)

{

pt->drawLine(start, end);

}

//初始化引力场，使用哑铃模型

void Line::**initGravity**()

{

if (!initStart || !initEnd)

return;//只有当起点、终点都设置后才能初始化

QVector<QPoint> vertex;

int dx, dy;

double k, t;

if (end.x()==start.x()){//垂直线

vertex.append(QPoint(start.x() - gi, start.y() - gi));

vertex.append(QPoint(start.x() + gi, start.y() - gi));

vertex.append(QPoint(end.x() + gi, end.y() + gi));

vertex.append(QPoint(end.x() - gi, end.y() + gi));

}

else

{

k = static\_cast<double>(end.y()-start.y())/(end.x()-start.x());//直线斜率

t = qSqrt(1/(1 + k\*k));

dx = static\_cast<int>(t\*k\*gi);

dy = static\_cast<int>(t\*gi);

vertex.append(QPoint(start.x() + dx, start.y() - dy));

vertex.append(QPoint(start.x() - dx, start.y() + dy));

vertex.append(QPoint(end.x() - dx, end.y() + dy));

vertex.append(QPoint(end.x() + dx, end.y() - dy));

}

gravity = QPolygon(vertex);

}

//绘制引力场

void Line::**drawGravity**(QPainter\* pt)

{

initGravity();

pt->save();

pt->setPen(Qt::blue);

double k = (double)(end.y() - start.y()) /

(double)(end.x() - start.x());

pt->drawLine(start, end);

pt->drawArc(start.x() - 2\*gi, start.y() - 2\*gi,

4\*gi, 4\*gi, 480 - 16\*180\*qAtan(k)/PI, 4800);

pt->drawArc(end.x() - 2\*gi, end.y() - 2\*gi,

4\*gi, 4\*gi, -180\*16 + 30\*16 - 16\*180\*qAtan(k)/PI, 4800);

pt->drawLine(start+

QPoint(2\*gi\*qCos(PI/6-qAtan(k)), -2\*gi\*qSin(PI/6-qAtan(k))),

end+

QPoint(2\*gi\*qCos(PI\*5/6-qAtan(k)), -2\*gi\*qSin(PI\*5/6-qAtan(k))));

pt->drawLine(start+

QPoint(2\*gi\*qCos(-PI/6-qAtan(k)), -2\*gi\*qSin(-PI/6-qAtan(k))),

end+

QPoint(2\*gi\*qCos(-PI\*5/6-qAtan(k)), -2\*gi\*qSin(-PI\*5/6-qAtan(k))));

pt->restore();

}

//设置引力场强度

void Line::**setGravityIntensity**(int intensity)

{

gi = intensity;

initGravity();

}

//判断指定点是否在引力场区域内

bool Line::**ptInGravity**(QPoint point)

{

QRegion rgn1 = QRegion(QRect(start.x() - 2\*gi, start.y() - 2\*gi, 4\*gi, 4\*gi), QRegion::Ellipse);

QRegion rgn2 = QRegion(QRect(end.x() - 2\*gi, end.y() - 2\*gi, 4\*gi, 4\*gi), QRegion::Ellipse);

QRegion rgn3 = QRegion(gravity, Qt::WindingFill);

QRegion rgn = rgn1.united(rgn2.united(rgn3));

if(rgn.contains(point))

return true;

else

return false;

}

//返回线段上离指定点最近的点

QPoint Line::**getClosestPoint**(QPoint point)

{

QPoint pt;

int xt, yt;

double k;

//获取线段所在直线离指定点最近的点

if (start.x() == end.x())//垂直线

{

pt.setX(start.x());

pt.setY(point.y());

}

else

{

k = static\_cast<double>(end.y()-start.y())/(end.x()-start.x());//线段斜率

xt = static\_cast<int>((k\*(point.y()-start.y()+k\*start.x())+point.x())/(1+k\*k));

yt = static\_cast<int>((k\*(point.x()-start.x()+k\*point.y())+start.y())/(1+k\*k));

pt.setX(xt);

pt.setY(yt);

}

//将点约束到线段上

if (pt.x() < qMin(start.x(), end.x()))

pt.setX(qMin(start.x(), end.x()));

if (pt.x() > qMax(start.x(), end.x()))

pt.setX(qMax(start.x(), end.x()));

if (pt.y() < qMin(start.y(), end.y()))

pt.setY(qMin(start.y(), end.y()));

if (pt.y() > qMax(start.y(), end.y()))

pt.setY(qMax(start.y(), end.y()));

return pt;

}

1. 新建矩形类RectEx

//RectEx.h

#include <QObject>

#include <QPoint>

#include <QRect>

#include <QtMath>

#include <QVector>

#include <QPainter>

//定义矩形状态

#define RS\_NORMAL 0 //普通

#define RS\_TRANSLATING 1 //移动

#define RS\_SCALING 2 //缩放

#define RS\_ROTATING 3 //旋转

//定义点相对于矩形的位置

#define CR\_SELF 0 //点在矩形内

#define CR\_NORTHWEST 1 //点在矩形左上

#define CR\_WEST 2 //点在矩形左中

#define CR\_SOUTHWEST 3 //点在矩形左下

#define CR\_SOUTH 4 //点在矩形下中

#define CR\_SOUTHEAST 5 //点在矩形右下

#define CR\_EAST 6 //点在矩形右中

#define CR\_NORTHEAST 7 //点在矩形右上

#define CR\_NORTH 8 //点在矩形上中

#define CR\_CENTER 9 //点在矩形中心

class RectEx

{

private:

QPoint oldPoint;

QRect rectangle;

int status;//矩形当前状态

QPoint centerPoint;//矩形的中心点

QVector<QPoint> points;//矩形的原始顶点，用于旋转

QVector<QPoint> pointstemp;//矩形的临时顶点，用于旋转

double deflection;//旋转偏角

QVector<QRect> ctrlRects;//矩形控制框

void **setCtrlRects**();//设置矩形的控制框

public:

RectEx();

QPoint **getTopLeft**();

QPoint **getBottonRight**();

void **setRect**(int x1, int y1, int x2, int y2);//设置矩形

void **setTop**(int top);//设置矩形上边

void **setBottom**(int bottom);//设置矩形下边

void **setLeft**(int left);//设置矩形左边

void **setRight**(int right);//设置矩形右边

void **setCenterPoint**(QPoint point);//设置矩形中心

void **setStatus**(int status);//设置矩形状态

int **getStatus**();//返回矩形状态

QPoint **topLeft**();//返回矩形的左上顶点

QPoint **bottomRight**();//返回矩形的右下顶点

void **offsetRect**(QSize size);//移动矩形

static double **getAngle**(QPoint origin, QPoint start, QPoint end);//根据点获取偏角

void **rotateRect**(QPoint p1, QPoint p2, bool change = false);//旋转矩形

int **getCurrentCtrlRect**(QPoint point);//返回指定点位于矩形的哪个控制框内

void **draw**(QPainter\* pt);//绘制矩形

};

#endif // RECTEX\_H

//RectEx.cpp

#include "rectex.h"

RectEx::**RectEx**()

{

status = RS\_NORMAL;//初始化矩形为一般状态

deflection = 0;//初始化旋转偏角为0

}

void RectEx::**setRect**(int x1, int y1, int x2, int y2)

{

rectangle = QRect(QPoint(x1,y1), QPoint(x2, y2));

centerPoint = rectangle.center();

//初始化points数组，用于旋转

if(!points.isEmpty())points.clear();

points.append(QPoint(x1,y1));

points.append(QPoint(x1,y2));

points.append(QPoint(x2,y2));

points.append(QPoint(x2,y1));

pointstemp = points;

}

//设置上边

void RectEx::**setTop**(int top)

{

rectangle.setTop(top);

}

//设置下边

void RectEx::**setBottom**(int bottom)

{

rectangle.setBottom(bottom);

}

//设置左边

void RectEx::**setLeft**(int left)

{

rectangle.setLeft(left);

}

//设置右边

void RectEx::**setRight**(int right)

{

rectangle.setRight(right);

}

//设置状态

void RectEx::**setStatus**(int status)

{

this->status = status;

//在缩放结束后需规格化矩形，以便控制框具有正确的位置

if (this->status == RS\_NORMAL)

rectangle.normalized();

}

//获取矩形状态

int RectEx::**getStatus**()

{

return status;

}

//获取左上点

QPoint RectEx::**topLeft**()

{

return rectangle.topLeft();

}

//获取右下点

QPoint RectEx::**bottomRight**()

{

return rectangle.bottomRight();

}

//设置旋转中心

void RectEx::**setCenterPoint**(QPoint point)

{

centerPoint = point;

//for rotating

deflection = 0;//重置旋转偏角为0

points = pointstemp;

}

//设置控制框

void RectEx::**setCtrlRects**()

{

/////////following shows the contrl rectangles

/\*

O---O---O

| |

O O O

| |

O---O---O

\*/

int cx = centerPoint.x();

int cy = centerPoint.y();

if (status != RS\_ROTATING)//非旋转情况

{

QRect rect(rectangle);

int l = rect.left();

int r = rect.right();

int t = rect.top();

int b = rect.bottom();

int h = (l + r) / 2;

int v = (t + b) / 2;

if(!ctrlRects.isEmpty())ctrlRects.clear();

ctrlRects.append(rectangle);

ctrlRects.append(QRect(QPoint(l - 5, t - 5), QPoint(l + 5, t + 5)));

ctrlRects.append(QRect(QPoint(l - 5, v - 5), QPoint(l + 5, v + 5)));

ctrlRects.append(QRect(QPoint(l - 5, b - 5), QPoint(l + 5, b + 5)));

ctrlRects.append(QRect(QPoint(h - 5, b - 5), QPoint(h + 5, b + 5)));

ctrlRects.append(QRect(QPoint(r - 5, b - 5), QPoint(r + 5, b + 5)));

ctrlRects.append(QRect(QPoint(r - 5, v - 5), QPoint(r + 5, v + 5)));

ctrlRects.append(QRect(QPoint(r - 5, t - 5), QPoint(r + 5, t + 5)));

ctrlRects.append(QRect(QPoint(h - 5, t - 5), QPoint(h + 5, t + 5)));

ctrlRects.append(QRect(QPoint(cx - 5, cy - 5), QPoint(cx + 5, cy + 5)));

}

else//旋转情况

{ if(ctrlRects.isEmpty()){

QRect rect(rectangle);

int l = rect.left();

int r = rect.right();

int t = rect.top();

int b = rect.bottom();

int h = (l + r) / 2;

int v = (t + b) / 2;

ctrlRects.append(rectangle);

ctrlRects.append(QRect(QPoint(l - 5, t - 5), QPoint(l + 5, t + 5)));

ctrlRects.append(QRect(QPoint(l - 5, v - 5), QPoint(l + 5, v + 5)));

ctrlRects.append(QRect(QPoint(l - 5, b - 5), QPoint(l + 5, b + 5)));

ctrlRects.append(QRect(QPoint(h - 5, b - 5), QPoint(h + 5, b + 5)));

ctrlRects.append(QRect(QPoint(r - 5, b - 5), QPoint(r + 5, b + 5)));

ctrlRects.append(QRect(QPoint(r - 5, v - 5), QPoint(r + 5, v + 5)));

ctrlRects.append(QRect(QPoint(r - 5, t - 5), QPoint(r + 5, t + 5)));

ctrlRects.append(QRect(QPoint(h - 5, t - 5), QPoint(h + 5, t + 5)));

ctrlRects.append(QRect(QPoint(cx - 5, cy - 5), QPoint(cx + 5, cy + 5)));

}

QPoint pt1, pt2;

int i;

for (i = 0; i<4; i++)//位于顶点的控制框

{

pt1 = pointstemp.at(i);

ctrlRects.replace(i\*2+1, QRect(QPoint(pt1.x() - 5, pt1.y() - 5), QPoint(pt1.x() + 5, pt1.y() + 5)));

}

for (i = 0; i<4; i++)//位于边中点的控制框

{

pt1 = pointstemp.at(i);

pt2 = pointstemp.at((i+1)%4);

ctrlRects.replace(i\*2+2, QRect(QPoint((pt1.x()+pt2.x())/2-5, (pt1.y()+pt2.y())/2-5),

QPoint((pt1.x()+pt2.x())/2+5, (pt1.y()+pt2.y())/2+5)));

}

ctrlRects.replace(9, QRect(QPoint(cx - 5, cy - 5), QPoint(cx + 5, cy + 5)));

}

}

QPoint RectEx::**getTopLeft**()

{

return rectangle.topLeft();

}

QPoint RectEx::**getBottonRight**()

{

return rectangle.bottomRight();

}

//返回指定点位于矩形的哪个控制框中

int RectEx::**getCurrentCtrlRect**(QPoint point)

{

setCtrlRects();//设置控制框

// if(m\_Status == RS\_TRANSLATING && m\_CtrlRects[0].PtInRect(point))

// return 0;

// if(m\_Status == RS\_SCALING || m\_Status == RS\_ROTATING)

for (int i = 9; i >= 0; i--)

if (ctrlRects.at(i).contains(point))

return i;

return -1;

}

//移动矩形

void RectEx::**offsetRect**(QSize size)

{

rectangle.translate(size.width(), size.height());

//

for(int i=0;i<ctrlRects.length();i++)

ctrlRects[i].translate(size.width(), size.height());

}

//获取旋转角度，origin为旋转参考点，start为起点，end为终点

double RectEx::**getAngle**(QPoint origin, QPoint start, QPoint end)

{

int aa = (start.y() - origin.y())\*(start.y() - origin.y()) + (start.x() - origin.x())\*(start.x() - origin.x());

int bb = (end.y() - origin.y())\*(end.y() - origin.y()) + (end.x() - origin.x())\*(end.x() - origin.x());

int cc = (end.y() - start.y())\*(end.y() - start.y()) + (end.x() - start.x())\*(end.x() - start.x());

double cosagl = (aa + bb - cc) / (2 \* qSqrt(aa)\*qSqrt(bb));//使用余弦定理计算角度

int flag = 1;//角度正负系数，在屏幕坐标系中，顺时针为正，逆时针为负

if (start.x() == origin.x())//起始边垂直

{

flag = (start.y()>origin.y()) ? 1 : -1;//起始边向量方向(origin->start)，下正上负

flag = (end.x()<start.x()) ? flag : -flag;//终点在起始边向量的右边为正左边为负

}

else//起始边不垂直

{

double k = (start.y() - origin.y()) / (double)(start.x() - origin.x());//起始边斜率

double t = (end.y() - start.y()) - k\*(end.x() - start.x());//终点相对于起始边的位置

flag = (start.x()>origin.x()) ? 1 : -1;//起始边向量方向，右正左负

flag = t>0 ? flag : -flag;//终点在起始边向量的右边为正左边为负

}

return flag\*qAcos(cosagl);

}

/\*\*旋转矩形，p1为旋转起点，p2为旋转终点，

\* change为假表示当前是旋转的中间过程，为真则结束旋转，记录旋转角度

\* 原因是因为旋转叠代次数越多误差越大

\*\*/

void RectEx::**rotateRect**(QPoint p1, QPoint p2, bool change)

{

double agl = getAngle(centerPoint, p1, p2);//获取偏角

agl += deflection;//累加偏角，agl变成相对于最原始状态的偏角

double R[3][3];//构造旋转矩阵

R[0][0] = qCos(agl);

R[0][1] = -qSin(agl);

R[0][2] = centerPoint.x()\*(1 - qCos(agl)) + centerPoint.y()\*qSin(agl);

R[1][0] = qSin(agl);

R[1][1] = qCos(agl);

R[1][2] = centerPoint.y()\*(1 - qCos(agl)) - centerPoint.x()\*qSin(agl);

R[2][0] = 0;

R[2][1] = 0;

R[2][2] = 1;

//生成新坐标

for (int i = 0; i<4; i++)

pointstemp.replace(i, QPoint((int)(R[0][0] \* points[i].x() + R[0][1] \* points[i].y() + R[0][2]),

(int)(R[1][0] \* points[i].x() + R[1][1] \* points[i].y() + R[1][2])));

if (change){//结束旋转时，记录当前偏角

deflection = agl;

}

}

//绘制矩形

void RectEx::**draw**(QPainter\* pt)

{

int i;

switch (status)

{

case RS\_NORMAL:

case RS\_TRANSLATING:

pt->drawRect(rectangle);

break;

case RS\_SCALING:

setCtrlRects();

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

pt->drawRect(ctrlRects.at(i));

break;

case RS\_ROTATING:

setCtrlRects();

pt->drawPolygon(pointstemp);

for (i = 1; i<10; i++)

pt->drawEllipse(ctrlRects.at(i));

break;

}

}

1. 新建圆类Circle

//Circle.h

#ifndef CIRCLE\_H

#define CIRCLE\_H

#include <QObject>

#include <QPoint>

#include <QPainter>

class Circle

{

private:

QPoint center;//圆心

int radius;//圆半径

public:

Circle();

Circle(QPoint center, int radius);

void **setCircle**(QPoint center, int radius);//通过中心和半径设置圆

void **setCenter**(QPoint center);//设置圆心

void **setRadius**(int radius);//设置半径

QPoint **getCenter**();//获取圆心

int **getRadius**();//获取半径

void **draw**(QPainter\* pt);//绘制圆

bool **ptInCircle**(QPoint point);//判断点是否在圆内

};

#endif // CIRCLE\_H

//Circle.cpp

#include "circle.h"

Circle::**Circle**()

{

}

Circle::**Circle**(QPoint center, int radius)

{

this->center = center;

this->radius = radius;

}

//设置圆

void Circle::**setCircle**(QPoint center, int radius)

{

this->center = center;

this->radius = radius;

}

//设置圆心

void Circle::**setCenter**(QPoint center)

{

this->center = center;

}

//设置半径

void Circle::**setRadius**(int radius)

{

this->radius = radius;

}

//获取圆心

QPoint Circle::**getCenter**()

{

return center;

}

//获取半径

int Circle::**getRadius**()

{

return radius;

}

//绘制圆

void Circle::**draw**(QPainter\* pt)

{

pt->drawEllipse(center.x() - radius, center.y() - radius, 2\*radius, 2\*radius);

}

//判断指定的点是否在圆内

bool Circle::**ptInCircle**(QPoint point)

{

return (point.x() - center.x())\*(point.x() - center.x()) +

(point.y() - center.y())\*(point.y() - center.y()) <= radius\*radius;

}

1. 新建网格类Grid

//Grid.h

#ifndef GRID\_H

#define GRID\_H

#include <QObject>

#include <QPoint>

#include <QRect>

#include <QPainter>

#include <QColor>

#include <QPen>

class Grid

{

private:

QColor color;//网格线颜色

int xspacing;//网格水平间距

int yspacing;//网格垂直间距

QRect size;//网格范围，客户空间大小

public:

Grid();

Grid(int xspacing, int yspacing, QColor color, QRect size);

QRect **getSize**();

void **setSpacing**(int xspacing, int yspacing);//设置网格间距

int **getXSpacing**();//返回水平间距

int **getYSpacing**();//返回垂直间距

void **setColor**(QColor color);//设置网格线颜色

void **setSize**(QRect size);

void **draw**(QPainter\* pt);

QPoint **getClosestPoint**(QPoint point);//获取离指定点最近的网点

};

#endif // GRID\_H

//Grid.cpp

#include "grid.h"

Grid::**Grid**()

{

xspacing = 50;

yspacing = 50;

color = Qt::blue;

}

Grid::**Grid**(int xspacing, int yspacing, QColor color, QRect size)

{

this->xspacing = xspacing;

this->yspacing = yspacing;

this->color = color;

this->size = size;

}

//设置网格线颜色

void Grid::**setColor**(QColor color)

{

this->color = color;

}

//设置网格间距

void Grid::**setSpacing**(int xspacing, int yspacing)

{

this->xspacing = xspacing;

this->yspacing = yspacing;

}

void Grid::**setSize**(QRect size)

{

this->size = size;

}

//返回水平间距

int Grid::**getXSpacing**()

{

return xspacing;

}

//返回垂直间距

int Grid::**getYSpacing**()

{

return yspacing;

}

QRect Grid::**getSize**()

{

return size;

}

//绘制网格

void Grid::**draw**(QPainter\* pt)

{

pt->save();

pt->setPen(color);

//绘制水平线

for (int i = 0; i<size.height()/yspacing + 1; i++)

{

int y = i\*yspacing;

pt->drawLine(QPoint(size.left(),y), QPoint(size.right(),y));

}

//绘制垂直线

for (int j = 0; j<size.width()/xspacing + 1; j++)

{

int x = j\*xspacing;

pt->drawLine(QPoint(x, size.top()), QPoint(x, size.bottom()));

}

pt->restore();

}

//返回离指定点最近的网点(两条网线的交点)

QPoint Grid::**getClosestPoint**(QPoint point)

{

int m = size.width()/xspacing\*xspacing;

int n = size.height()/yspacing\*yspacing;

if (point.x() <= (size.left() - xspacing / 2))

point.setX(size.left());

if (point.x() >= (m + xspacing / 2))

point.setX(m);

if (point.y() <= (size.top() - yspacing / 2))

point.setY(size.top());

if (point.y() >= (n + yspacing / 2))

point.setY(n);

int x = (point.x() + xspacing / 2) / xspacing\*xspacing;

int y = (point.y() + yspacing / 2) / yspacing\*yspacing;

return QPoint(x, y);

}

1. 新建标尺类Ruler

//Ruler.h

#ifndef RULER\_H

#define RULER\_H

#include <QPainter>

#define CP\_OUTER 0

#define CP\_HORIZONTAL 1

#define CP\_VERTICAL 2

#define CP\_CORNER 3

#define CP\_CLIENT 4

class Ruler

{

private:

QRect vrect;

QRect hrect;

int width;//标尺宽度

int spacing;//标尺刻度间距

QSize size;//客户区大小

public:

Ruler();

int **GetWidth**();//返回标尺宽度

int **GetCursorPosition**(QPoint point);//返回指定点相对标尺的位置

void **Show**(QRect win, QPainter \*pt);//显示标尺

void **ShowVernier**(QRect win, QPainter \*pt, QPoint point);//显示游标

void **HideVernier**(QRect win, QPainter \*pt);//隐藏游标

};

#endif // RULER\_H

//Ruler.cpp

#include "ruler.h"

Ruler::**Ruler**()

{

width=30;

spacing=5;

}

int Ruler::**GetWidth**()

{

return width;

}

int Ruler::**GetCursorPosition**(QPoint point)

{

QRect rect1=hrect;

QRect rect2=vrect;

rect1.adjust(0,0,0,2);

rect2.adjust(0,0,2,0);

if(rect1.contains(point)&&rect2.contains(point))

return CP\_CORNER;

if(rect1.contains(point))

return CP\_HORIZONTAL;

if(rect2.contains(point))

return CP\_VERTICAL;

rect1.setTop(rect1.bottom());

rect1.setLeft(rect2.right());

rect1.setBottom(rect2.bottom());

if(rect1.contains(point))

return CP\_CLIENT;

return CP\_OUTER;

}

void Ruler::**Show**(QRect win,QPainter \*pt)

{

size.setWidth(win.width());

size.setHeight(win.height());

hrect=vrect=win;

hrect.setBottom(width);

vrect.setRight(width);

pt->save();

pt->setPen(QPen(QColor(Qt::black)));

pt->drawRect(hrect);

pt->drawRect(vrect);

for(int i=0;i<size.width();i+=spacing)

{

pt->drawLine(QPoint(i,hrect.bottom()),QPoint(i,hrect.bottom()-width/4));

if(i%(spacing\*10)==0){

pt->drawLine(QPoint(i,hrect.bottom()),QPoint(i,hrect.bottom()-width\*3/4));

pt->drawText(i+2,hrect.bottom()-width/2+2,QString::number(i));

}

}

for(int i = 0;i<size.height();i+=spacing)

{

pt->drawLine(QPoint(vrect.right(),i),QPoint(vrect.right()-width/4,i));

if(i%(spacing\*10)==0){

pt->drawLine(QPoint(vrect.right(),i),QPoint(vrect.right()-width\*3/4,i));

pt->drawText(0,i+2,QString::number(i));

}

}

}

void Ruler::**ShowVernier**(QRect win,QPainter \*pt, QPoint point)

{

Show(win,pt);

if(point.x()==0&&point.y()==0)return;

pt->drawLine(QPoint(point.x(),width-1),QPoint(point.x(),width/8));

pt->drawLine(QPoint(width-1,point.y()),QPoint(width/8,point.y()));

}

void Ruler::**HideVernier**(QRect win,QPainter \*pt)

{

ShowVernier(win,pt,QPoint(0,0));

}

1. 新建引导线类Guide

//Guide.h

#ifndef GUIDE\_H

#define GUIDE\_H

#include <QPainter>

#include <QVector>

#include "ruler.h"

#include "rectex.h"

class Guide

{

private:

int gravity;

bool drawline;

QLine line;

Ruler ruler;

public:

Guide();

void **ShowGuides**(QPainter \*pt, QRect win, QPoint point, bool vline, bool drawline);

void **SetGravity**(int gravity);

int **getGravity**();

bool **isInRuler**(QPoint point);

bool **isVRuler**(QPoint point);

void **setLine**(bool vline,QPoint point);

bool **isInClient**(QPoint point);

QLine **getLine**();

};

#endif // GUIDE\_H

// Guide.cpp

#include "guide.h"

Guide::**Guide**()

{

gravity=10;

}

void Guide::**ShowGuides**(QPainter \*pt, QRect win,QPoint point,bool vline,bool drawline)

{

ruler.HideVernier(win,pt);

pt->drawLine(line);

}

void Guide::**SetGravity**(int gravity)

{

this->gravity=gravity;

}

int Guide::**getGravity**()

{

return gravity;

}

bool Guide::**isInRuler**(QPoint point)

{

if((ruler.GetCursorPosition(point)==CP\_HORIZONTAL||ruler.GetCursorPosition(point)==CP\_VERTICAL)&&ruler.GetCursorPosition(point)!=CP\_CORNER)

return true;

return false;

}

bool Guide::**isVRuler**(QPoint point)

{

if(ruler.GetCursorPosition(point)==CP\_VERTICAL)

return true;

if(ruler.GetCursorPosition(point)==CP\_HORIZONTAL)

return false;

}

void Guide::**setLine**(bool vline, QPoint point)

{

if(vline)line=QLine(QPoint(ruler.GetWidth(),point.y()),point);

if(!vline)line=QLine(QPoint(point.x(),ruler.GetWidth()),point);

}

bool Guide::**isInClient**(QPoint point)

{

return ruler.GetCursorPosition(point)==CP\_CLIENT;

}

QLine Guide::**getLine**()

{

return line;

}

1. 完善textWidget类

//textWidget.h

#ifndef TEXTWIDGET\_H

#define TEXTWIDGET\_H

#include <QWidget>

#include <QPainter>

#include <QVector>

#include <QString>

#include <QRectF>

class textWidget : public QWidget

{

Q\_OBJECT

public:

explicit textWidget(QWidget \*parent = nullptr);

void ***paintEvent***(QPaintEvent\*);

void **setcurrentShow**(int cur);

void **startOperat**();

signals:

public slots:

private:

int currentShow;

bool isOperating;

QVector<QString> text;

};

#endif // TEXTWIDGET\_H

//textWidget.cpp

#include "textwidget.h"

textWidget::**textWidget**(QWidget \*parent) : QWidget(parent)

{

setPalette(QPalette(QColor(251,251,251)));

setAutoFillBackground(true);

text<<tr("\r\n坐标显示：\r\n\r\n为了使定位准确，可以在光标附近显示光标所在位置的坐标。实际应用中，也可以在状态栏等其它位置显示坐标。\r\n\r\n\r\n演示：\r\n\r\n在演示区当中按下鼠标左键并移动鼠标，指针附近便会出现坐标值。");

text<<tr("\r\n橡皮筋：\r\n\r\n橡皮筋是一种比喻，它是在用鼠标绘制直线、矩形、椭圆等形体时动态的显示中间过程，使得几何形体就象具有弹性一样。这样做，可以让用户直接看到形体的当前位置和形状。\r\n\r\n\r\n演示：\r\n\r\n本例子只演示了直线橡皮筋。在演示区中按下鼠标左键并移动，便能看见一条动态的直线。");

text<<tr("\r\n移动：\r\n\r\n移动是改变对象位置的一种操作。通过实时的显示对象的当前位置，可以让用户准确的对对象进行定位。\r\n\r\n\r\n演示：\r\n\r\n将鼠标放在矩形上，指针形状将变成“移动”样式，这时按下左键并移动鼠标，矩形便会跟着移动。到了合适的位置松开左键，矩形的位置便被固定在了该处。");

text<<tr("\r\n缩放：\r\n\r\n缩放是改变对象大小的一种操作。通过实时的显示对象的当前形状，可以让用户直观的看到对象形状的变换。\r\n\r\n\r\n演示：\r\n\r\n将鼠标放在矩形的八个控制框上，鼠标指针的形状会发生相应的变化，按下左键并移动鼠标，矩形的形状和大小便会随之变化。");

text<<tr("\r\n旋转：\r\n\r\n旋转也是经常用到的一种操作。它需要指定一个“参考点”作为旋转中心，该旋转中心的位置可以改变，一般默认为形体的几何中心。\r\n\r\n\r\n演示：\r\n\r\n如右图，在九个小圆圈中，中间的一个是旋转中心（鼠标指针放上去是“移动”形状），可以用鼠标改变旋转中心的位置。然后用鼠标拖动其它八个小圆圈，便能让矩形绕着“中心”旋转。");

text<<tr("\r\n引力场：\r\n\r\n引力场技术实际是通过模糊控制将某个范围内的点用一个特殊的点来代替，主要用于“细小”形体的选取，以及线与线间的链接等操作。\r\n\r\n\r\n演示：\r\n\r\n本例子演示的是一条线段周围的引力场，右图中的蓝色哑铃区域表示引力场，实际中是不显示出来的。按下鼠标左键并移动鼠标，演示区中会出现一条直线，当鼠标指针位于蓝色区域中时，直线的活动端便会自动吸附到线段上。这实际上演示了两条线段的连接技术。");

text<<tr("\r\n定向：\r\n\r\n通过定向技术，可以绘制特殊方向的直线，如水平方向、垂直方向、斜45度等，这为工程制图中的绘线等带来了方便。\r\n\r\n\r\n演示：\r\n\r\n本例子约束直线的方向为：水平、垂直和斜45度。实际中，一般是在按住Shift键时为定向操作，不按Shift键时可画任意方向的直线。");

text<<tr("\r\n网格：\r\n\r\n网格也是一种约束技术，它限制了形体仅能放置于特殊的点上，主要用于各种布局操作，如印刷线路板的绘制，棋盘上棋子的放置等。\r\n\r\n\r\n演示：\r\n\r\n使用鼠标可以移动图中圆圈的位置，但圆圈只能位于其中的网点上。");

text<<tr("\r\n标尺：\r\n\r\n标尺是绘图中的一种辅助工具，它可以让用户定量的了解形体的位置和尺寸。\r\n\r\n\r\n演示：\r\n\r\n在演示区中按下鼠标左键并移动鼠标，标尺上会出现相应的指针显示当前的位置。");

text<<tr("\r\n引导线：\r\n\r\n引导线也是一种辅助绘图工具，它一般与标尺配合使用，主要用于对形体进行精确定位。\r\n\r\n\r\n演示：\r\n\r\n在标尺上按下左键并往演示区移动鼠标便可以生成一条引导线，可以移动引导线改变其位置，如果要删除某条引导线只需将其拖到客户区以外即可。将演示区中的矩形拖动到引导线附近，矩形便被自动吸附到标尺上。");

}

void textWidget::***paintEvent***(QPaintEvent\*)

{

QPainter\* pt = new QPainter(this);

if(!isOperating)

pt->drawText(QRectF(rect()),Qt::TextWrapAnywhere,tr("\r\n交互技术演示\r\n\r\n本例程演示了实际中常用的一些交互技术，主要模仿了Flash和PhotoShop中的相似操作，用户可以参考这些应用软件，看看其实际效果。"));

else

pt->drawText(QRectF(rect()),Qt::TextWrapAnywhere,text[currentShow]);

}

void textWidget::**setcurrentShow**(int cur)

{

currentShow=cur;

update();

}

void textWidget::**startOperat**()

{

isOperating=true;

update();

}

1. 完善figureWidget类

//figureWidget.h

#ifndef FIGUREWIDGET\_H

#define FIGUREWIDGET\_H

#include <QWidget>

#include <QPainter>

#include <QMouseEvent>

#include <QString>

#include <QPoint>

#include <QPen>

#include <QColor>

#include <QRect>

#include "line.h"

#include "rectex.h"

#include "circle.h"

#include "grid.h"

#include "ruler.h"

#include "guide.h"

#define ID\_SHOW\_XY 0

#define ID\_SHOW\_RUBBERBAND 1

#define ID\_SHOW\_TRANSLATE 2

#define ID\_SHOW\_SCALE 3

#define ID\_SHOW\_ROTATE 4

#define ID\_SHOW\_GRAVITY 5

#define ID\_SHOW\_DIRECTIONAL 6

#define ID\_SHOW\_GRID 7

#define ID\_SHOW\_RULER 8

#define ID\_SHOW\_GUIDE 9

class figureWidget : public QWidget

{

Q\_OBJECT

public:

explicit figureWidget(QWidget \*parent = nullptr);

signals:

public slots:

private:

bool isInitialized;//是否初始化

int currentShow;//当前演示项目

bool isOperating;//是否开始演示

QPoint centerPoint;//演示区中心点

QPoint oldPoint;//用于记录以前的点

QPoint newPoint;

Line line;//直线

Line gravityLine;//引力直线

RectEx rectangle;//矩形

Circle circle;//圆

Grid grid;//网格

Ruler ruler;//标尺

bool showVernier;

Guide guide;

bool drawGuide;

bool vOrH;

int currentCtrlRect;//矩形的当前控制框

bool isInCircle;//光标是否在圆中

public:

void ***paintEvent***(QPaintEvent\*);

void **Initialize**();//初始化函数

void ***mouseMoveEvent***(QMouseEvent \*event);

void ***mousePressEvent***(QMouseEvent \*event);

void ***mouseReleaseEvent***(QMouseEvent \*event);

void **setcurrentShow**(int cur);

void **startOperat**();

void **stopOperat**();

void **ID\_SHOW\_XY\_Work**(QPoint point);

void **ID\_SHOW\_RUBBERBAND\_Work**(QPoint point);

void **ID\_SHOW\_TRANSLATE\_Work**(QPoint point);

void **ID\_SHOW\_SCALE\_Work**(QPoint point);

void **ID\_SHOW\_ROTATE\_Work**(QPoint point,bool rot);

void **ID\_SHOW\_GRAVITY\_Work**(QPoint point);

void **ID\_SHOW\_DIRECTIONAL\_Work**(QPoint point);

void **ID\_SHOW\_GRID\_Work**(QPoint point);

void **ID\_SHOW\_RULER\_Work**(QPoint point);

void **ID\_SHOW\_GUIDE\_Work**(QPoint point);

};

#endif // FIGUREWIDGET\_H

//figureWidget.cpp

#include "figurewidget.h"

figureWidget::**figureWidget**(QWidget \*parent) : QWidget(parent)

{

setPalette(QPalette(QColor(254,254,254)));

setAutoFillBackground(true);

setMouseTracking(true);

}

void figureWidget::***paintEvent***(QPaintEvent\*)

{

QPainter \* pt = new QPainter(this);

pt->setPen(QColor(Qt::black));

QString pos;

switch(currentShow){

case ID\_SHOW\_XY :

pt->save();

pt->setPen(QPen(QColor(0,0,0)));

pos+=QString::number(newPoint.x())+","+QString::number(newPoint.y());

pt->drawText(newPoint.x()+10,newPoint.y()+10,pos);

pt->restore();

break;

case ID\_SHOW\_RUBBERBAND :

line.draw(pt);

break;

case ID\_SHOW\_TRANSLATE:

rectangle.draw(pt);

break;

case ID\_SHOW\_SCALE:

rectangle.draw(pt);

break;

case ID\_SHOW\_ROTATE:

rectangle.draw(pt);

break;

case ID\_SHOW\_GRAVITY:

gravityLine.draw(pt);

gravityLine.drawGravity(pt);

line.draw(pt);

circle.draw(pt);

break;

case ID\_SHOW\_DIRECTIONAL:

line.draw(pt);

break;

case ID\_SHOW\_GRID:

grid.draw(pt);

circle.draw(pt);

break;

case ID\_SHOW\_RULER:

ruler.Show(rect(),pt);

if(showVernier)ruler.ShowVernier(rect(),pt,newPoint);

else ruler.HideVernier(rect(),pt);

break;

case ID\_SHOW\_GUIDE:

rectangle.draw(pt);

guide.ShowGuides(pt,rect(),oldPoint,vOrH,drawGuide);

break;

default:

break;

}

}

void figureWidget::**Initialize**()

{

QRect rc=QRect(0,0,this->width(),this->height());

centerPoint=QPoint(this->width()/2,this->height()/2);

oldPoint=centerPoint;

//橡皮筋

if(currentShow==ID\_SHOW\_RUBBERBAND){

line.setStart(centerPoint);

line.setEnd(centerPoint);

}

//移动、比例尺、旋转

rectangle.setRect(centerPoint.x()-50,centerPoint.y()-40,centerPoint.x()+50,centerPoint.y()+40);

//引力场

if(currentShow==ID\_SHOW\_GRAVITY){

gravityLine.setStart(QPoint(50,200));

gravityLine.setEnd(QPoint(400,50));

gravityLine.setGravityIntensity(20);

line.setStart(centerPoint);

line.setEnd(centerPoint);

circle.setCircle(centerPoint,5);

}

//网格

if(currentShow==ID\_SHOW\_GRID){

QPoint point;

int radius=(grid.getXSpacing()>10)?10:grid.getXSpacing()/3;

grid.setSize(rc);

point=grid.getClosestPoint(centerPoint);

circle.setCircle(point ,radius);

}

if(currentShow==ID\_SHOW\_GUIDE){

rectangle.setRect(width()/3,height()\*3/5,width()\*5/6,height()\*4/5);

}

}

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

{

switch (currentShow) {

case ID\_SHOW\_XY:

setCursor(Qt::CrossCursor);

break;

case ID\_SHOW\_RUBBERBAND:

setCursor(Qt::ArrowCursor);

break;

case ID\_SHOW\_TRANSLATE:

if(rectangle.getCurrentCtrlRect(event->pos())==CR\_SELF)

setCursor(Qt::SizeAllCursor);

else

setCursor(Qt::ArrowCursor);

break;

case ID\_SHOW\_SCALE:

switch (currentCtrlRect)

{

case CR\_NORTHWEST:

case CR\_SOUTHEAST:

setCursor(Qt::SizeFDiagCursor);

break;

case CR\_NORTHEAST:

case CR\_SOUTHWEST:

setCursor(Qt::SizeBDiagCursor);

break;

case CR\_WEST:

case CR\_EAST:

setCursor(Qt::SizeHorCursor);

break;

case CR\_NORTH:

case CR\_SOUTH:

setCursor(Qt::SizeVerCursor);

break;

default:

setCursor(Qt::ArrowCursor);

}

break;

case ID\_SHOW\_ROTATE:

if(rectangle.getCurrentCtrlRect(event->pos())==CR\_CENTER)

setCursor(Qt::SizeAllCursor);

else if(rectangle.getCurrentCtrlRect(event->pos())==-1)

setCursor(Qt::ArrowCursor);

else

setCursor(Qt::ArrowCursor);

break;

case ID\_SHOW\_GRID:

if(circle.ptInCircle(event->pos()))

setCursor(Qt::SizeAllCursor);

else

setCursor(Qt::ArrowCursor);

break;

case ID\_SHOW\_RULER:

if(ruler.GetCursorPosition(event->pos())==CP\_CLIENT)

setCursor(Qt::SizeAllCursor);

else setCursor(Qt::ArrowCursor);

break;

case ID\_SHOW\_GUIDE:

if(rectangle.getCurrentCtrlRect(event->pos())==CR\_SELF)

setCursor(Qt::SizeAllCursor);

else

setCursor(Qt::ArrowCursor);

break;

default:

setCursor(Qt::ArrowCursor);

break;

}

if(!isOperating)

{

currentCtrlRect=rectangle.getCurrentCtrlRect(event->pos());

isInCircle=circle.ptInCircle(event->pos());

return;

}

switch (currentShow) {

case ID\_SHOW\_XY:

ID\_SHOW\_XY\_Work(event->pos());

break;

case ID\_SHOW\_RUBBERBAND:

ID\_SHOW\_RUBBERBAND\_Work(event->pos());

break;

case ID\_SHOW\_TRANSLATE:

ID\_SHOW\_TRANSLATE\_Work(event->pos());

break;

case ID\_SHOW\_SCALE:

ID\_SHOW\_SCALE\_Work(event->pos());

break;

case ID\_SHOW\_ROTATE:

ID\_SHOW\_ROTATE\_Work(event->pos(),false);

break;

case ID\_SHOW\_GRAVITY:

ID\_SHOW\_GRAVITY\_Work(event->pos());

break;

case ID\_SHOW\_DIRECTIONAL:

ID\_SHOW\_DIRECTIONAL\_Work(event->pos());

break;

case ID\_SHOW\_GRID:

ID\_SHOW\_GRID\_Work(event->pos());

break;

case ID\_SHOW\_RULER:

ID\_SHOW\_RULER\_Work(event->pos());

break;

case ID\_SHOW\_GUIDE:

ID\_SHOW\_GUIDE\_Work(event->pos());

break;

default:

break;

}

update();

}

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

{

switch (currentShow) {

case ID\_SHOW\_RUBBERBAND:

line.setStart(event->pos());

line.setEnd(event->pos());

break;

case ID\_SHOW\_ROTATE:

currentCtrlRect=rectangle.getCurrentCtrlRect(event->pos());

break;

case ID\_SHOW\_SCALE:

currentCtrlRect=rectangle.getCurrentCtrlRect(event->pos());

break;

case ID\_SHOW\_RULER:

showVernier=false;

update();

break;

case ID\_SHOW\_GUIDE:

drawGuide=guide.isInRuler(event->pos());

vOrH=guide.isVRuler(event->pos());

break;

default:

break;

}

if(!isInitialized){

Initialize();

isInitialized=true;

}

startOperat();

oldPoint=event->pos();

update();

}

void figureWidget::***mouseReleaseEvent***(QMouseEvent \*event)

{

stopOperat();

if(currentShow==ID\_SHOW\_ROTATE)

ID\_SHOW\_ROTATE\_Work(event->pos(),true);

if(currentShow==ID\_SHOW\_GUIDE)

if(drawGuide&&guide.isInClient(event->pos()))

guide.setLine(vOrH==true,event->pos());

}

void figureWidget::**setcurrentShow**(int cur)

{

currentShow=cur;

switch (cur) {

case ID\_SHOW\_TRANSLATE:

rectangle.setStatus(RS\_TRANSLATING);

break;

case ID\_SHOW\_SCALE:

rectangle.setStatus(RS\_SCALING);

break;

case ID\_SHOW\_ROTATE:

rectangle.setStatus(RS\_ROTATING);

break;

case ID\_SHOW\_GRID:

grid.setSize(rect());

break;

case ID\_SHOW\_GUIDE:

rectangle.setStatus(RS\_TRANSLATING);

break;

default:

break;

}

Initialize();

update();

}

void figureWidget::**startOperat**()

{

isOperating=true;

update();

}

void figureWidget::**stopOperat**()

{

isOperating=false;

update();

}

void figureWidget::**ID\_SHOW\_XY\_Work**(QPoint point)

{

newPoint=point;

}

void figureWidget::**ID\_SHOW\_RUBBERBAND\_Work**(QPoint point)

{

line.setEnd(point);

}

void figureWidget::**ID\_SHOW\_TRANSLATE\_Work**(QPoint point)

{

currentCtrlRect=rectangle.getCurrentCtrlRect(point);

if(currentCtrlRect==CR\_SELF)

rectangle.offsetRect(QSize(point.x()-oldPoint.x(),point.y()-oldPoint.y()));

oldPoint=point;

}

void figureWidget::**ID\_SHOW\_SCALE\_Work**(QPoint point)

{

switch (currentCtrlRect) {

case CR\_NORTHWEST:

rectangle.setTop(point.y());

rectangle.setLeft(point.x());

break;

case CR\_SOUTHEAST:

rectangle.setRight(point.x());

rectangle.setBottom(point.y());

break;

case CR\_NORTHEAST:

rectangle.setTop(point.y());

rectangle.setRight(point.x());

break;

case CR\_SOUTHWEST:

rectangle.setBottom(point.y());

rectangle.setLeft(point.x());

break;

case CR\_WEST:

rectangle.setLeft(point.x());

break;

case CR\_EAST:

rectangle.setRight(point.x());

break;

case CR\_NORTH:

rectangle.setTop(point.y());

break;

case CR\_SOUTH:

rectangle.setBottom(point.y());

break;

default:

break;

}

}

void figureWidget::**ID\_SHOW\_ROTATE\_Work**(QPoint point,bool rot)

{

switch (currentCtrlRect){

case CR\_NORTHWEST:

case CR\_SOUTHEAST:

case CR\_NORTHEAST:

case CR\_SOUTHWEST:

case CR\_WEST:

case CR\_EAST:

case CR\_NORTH:

case CR\_SOUTH:

rectangle.rotateRect(oldPoint,point,rot);//旋转矩形

break;

case CR\_CENTER:

rectangle.setCenterPoint(point);//改变旋转中心

break;

default:

break;

}

}

void figureWidget::**ID\_SHOW\_GRAVITY\_Work**(QPoint point)

{

line.setStart(QPoint(0,0));

if(gravityLine.ptInGravity(point))

line.setEnd(gravityLine.getClosestPoint(point));

else

line.setEnd(point);

circle.setCenter(line.getEnd());

}

void figureWidget::**ID\_SHOW\_DIRECTIONAL\_Work**(QPoint point)

{

QPoint start=line.getStart();

if(start.x()!=point.x())

if((point.y()-start.y())<tan(3.1415926/8)\*(point.x()-start.x()))

line.setEnd(QPoint(point.x(),start.y()));

else if((point.y()-start.y())>tan(3\*3.1415926/8)\*(point.x()-start.x()))

line.setEnd(QPoint(start.x(),point.y()));

else{

int temp= ((point.x()-start.x())>(point.y()-start.y()))?(point.x()-start.x()):(point.y()-start.y());

if(point.x()>start.x()&&point.y()>start.y())

line.setEnd(QPoint(start.x()+temp,start.y()+temp));

else if(point.x()<start.x()&&point.y()<start.y())

line.setEnd(QPoint(start.x()-temp,start.y()-temp));

else if(point.x()>start.x())

line.setEnd(QPoint(start.x()+temp,start.y()-temp));

else

line.setEnd(QPoint(start.x()-temp,start.y()+temp));

}

else

line.setEnd(QPoint(start.x(),point.y()));

}

void figureWidget::**ID\_SHOW\_GRID\_Work**(QPoint point)

{

if(grid.getClosestPoint(point)!=circle.getCenter())

circle.setCenter(grid.getClosestPoint(point));

}

void figureWidget::**ID\_SHOW\_RULER\_Work**(QPoint point)

{

if(ruler.GetCursorPosition(point)==CP\_CLIENT){

newPoint=point;

showVernier=true;

}

else

showVernier=false;

}

void figureWidget::**ID\_SHOW\_GUIDE\_Work**(QPoint point)

{

if(rectangle.getCurrentCtrlRect(point)==CR\_SELF)

rectangle.offsetRect(QSize(point.x()-oldPoint.x(),point.y()-oldPoint.y()));

if(drawGuide&&guide.isInClient(point))

guide.setLine(vOrH==true,point);

if(guide.getLine().dx()!=0&&guide.getLine().dy()==0){

if((rectangle.getTopLeft().y()-guide.getLine().p1().y())<guide.getGravity()&&(guide.getLine().p1().y()-rectangle.getTopLeft().y())<guide.getGravity())

rectangle.offsetRect(QSize(0,guide.getLine().p1().y()-rectangle.getTopLeft().y()));

if((rectangle.getBottonRight().y()-guide.getLine().p1().y())<guide.getGravity()&&(guide.getLine().p1().y()-rectangle.getBottonRight().y())<guide.getGravity())

rectangle.offsetRect(QSize(0,guide.getLine().p1().y()-rectangle.getBottonRight().y()));

}

if(guide.getLine().dx()==0&&guide.getLine().dy()!=0)

{

if((rectangle.getTopLeft().x()-guide.getLine().p1().x())<guide.getGravity()&&(guide.getLine().p1().x()-rectangle.getTopLeft().x())<guide.getGravity())

rectangle.offsetRect(QSize(guide.getLine().p1().x()-rectangle.getTopLeft().x(),0));

if((rectangle.getBottonRight().x()-guide.getLine().p1().x())<guide.getGravity()&&(guide.getLine().p1().x()-rectangle.getBottonRight().x())<guide.getGravity())

rectangle.offsetRect(QSize(guide.getLine().p1().x()-rectangle.getBottonRight().x(),0));

}

oldPoint=point;

}

1. 完善主窗口类，为各菜单项添加槽函数

//mainwindow.h

private slots:

void **on\_action\_C\_triggered**();

void **on\_action\_B\_triggered**();

void **on\_action\_T\_triggered**();

void **on\_action\_S\_triggered**();

void **on\_action\_R\_triggered**();

void **on\_action\_G\_triggered**();

void **on\_action\_D\_triggered**();

void **on\_action\_I\_triggered**();

void **on\_action\_U\_triggered**();

void **on\_action\_L\_triggered**();

//mainwindow.cpp

void MainWindow::***paintEvent***(QPaintEvent\*)

{

QPainter\* pt = new QPainter(this);

ui->splitter->setFixedSize(width(), height());

}

void MainWindow::**on\_action\_C\_triggered**()

{

ui->widget2->setcurrentShow(ID\_SHOW\_XY);

ui->widget1->setcurrentShow(ID\_SHOW\_XY);

ui->widget1->startOperat();

}

void MainWindow::**on\_action\_B\_triggered**()

{

ui->widget2->setcurrentShow(ID\_SHOW\_RUBBERBAND);

ui->widget1->setcurrentShow(ID\_SHOW\_RUBBERBAND);

ui->widget1->startOperat();

}

void MainWindow::**on\_action\_T\_triggered**()

{

ui->widget2->setcurrentShow(ID\_SHOW\_TRANSLATE);

ui->widget1->setcurrentShow(ID\_SHOW\_TRANSLATE);

ui->widget1->startOperat();

}

void MainWindow::**on\_action\_S\_triggered**()

{

ui->widget2->setcurrentShow(ID\_SHOW\_SCALE);

ui->widget1->setcurrentShow(ID\_SHOW\_SCALE);

ui->widget1->startOperat();

}

void MainWindow::**on\_action\_R\_triggered**()

{

ui->widget2->setcurrentShow(ID\_SHOW\_ROTATE);

ui->widget1->setcurrentShow(ID\_SHOW\_ROTATE);

ui->widget1->startOperat();

}

void MainWindow::**on\_action\_G\_triggered**()

{

ui->widget2->setcurrentShow(ID\_SHOW\_GRAVITY);

ui->widget1->setcurrentShow(ID\_SHOW\_GRAVITY);

ui->widget1->startOperat();

}

void MainWindow::**on\_action\_D\_triggered**()

{

ui->widget2->setcurrentShow(ID\_SHOW\_DIRECTIONAL);

ui->widget1->setcurrentShow(ID\_SHOW\_DIRECTIONAL);

ui->widget1->startOperat();

}

void MainWindow::**on\_action\_I\_triggered**()

{

ui->widget2->setcurrentShow(ID\_SHOW\_GRID);

ui->widget1->setcurrentShow(ID\_SHOW\_GRID);

ui->widget1->startOperat();

}

void MainWindow::**on\_action\_U\_triggered**()

{

ui->widget2->setcurrentShow(ID\_SHOW\_RULER);

ui->widget1->setcurrentShow(ID\_SHOW\_RULER);

ui->widget1->startOperat();

}

void MainWindow::**on\_action\_L\_triggered**()

{

ui->widget2->setcurrentShow(ID\_SHOW\_GUIDE);

ui->widget1->setcurrentShow(ID\_SHOW\_GUIDE);

ui->widget1->startOperat();

}

## 实验17光照模型实例

1. 创建一个MainWindow程序，保留ui设计界面
2. 修改MainWindow的属性windowTitle为“光照模型”
3. 将MainWindows的尺寸设置为650×450
4. 在ui中加入一个groupBox，对象名称为guoupBox\_set，标题设为“设置面板”，尺寸为170×410
5. 在设置面板groupBox\_set中加入一个checkBox，对象名称为checkBox\_lightEnable，文本标题为“启用光照”，位置为x=20，y=20，宽度70，高度20
6. 在设置面板groupBox\_set中加入一个groupBox，对象名称为groupBox\_light，文本标题为“光照模式”，位置x=10，y=50，宽度150，高度90
7. 在groupBox\_light中插入三个checkBox，对象名称为checkBox\_doubleFace、checkBox\_infinete、checkBox\_global，文本为双面光照模式、无穷远光照模式、全场环境光模式
8. 插入一个光源颜色guoupBox\_lightColor，位置x=10，y=150，宽度150，高度110
9. 在guoupBox\_lightColor中插入三个frame控件，尺寸50×20，名称为frameColor\_diffuse、frameColor\_specular、frameColor\_ambient，frameShape为Panel，frameShadow为Sunken，autoFillBackground为true
10. 打开MainWindow.cpp，在构造函数中添加代码，设置光源颜色初始值

ui->frameColor\_ambient->setPalette(QPalette(Qt::black));

ui->frameColor\_diffuse->setPalette(QPalette(Qt::black));

ui->frameColor\_specular->setPalette(QPalette(Qt::black));

1. 在guoupBox\_lightColor中插入三个按钮：漫反射、镜面反射、环境光，对象名称为：pushButton\_diffuse、pushButton\_specular、pushButton\_ambient，尺寸为60×20
2. 在设置面板groupBox\_set中加入一个groupBox，对象名称为groupBox\_ material，文本为“物体材质”，尺寸为150×50
3. 在groupBox\_ material中加入一个comboBox\_material，尺寸为70×20
4. 鼠标右击comboBox\_material，选择编辑项目，依次添加下拉项：默认值，红宝石，黄铜，青铜，白银，祖母绿
5. 在ui中加入一个widget\_draw，位置x=186，y=10，尺寸460×400
6. 新建一个widgetDraw类，基类为QWidget，autoFillBackground设为true
7. 将ui中的widget\_draw提升为widgetDraw
8. 打开widgetDraw类的构造函数，添加一行代码

setPalette(QPalette(Qt::white));

1. 打开MainWindow.h，加入

#include <QPainter>

1. 添加函数

void ***paintEvent***(QPaintEvent\*);

1. 在MainWindow.cpp中实现函数

void MainWindow::***paintEvent***(QPaintEvent\*)

{

QPainter\* ptr = new QPainter(this);

ui->widget\_draw->setFixedSize(width()-185, height()-20);

delete ptr;

}

1. 打开Lightdemo\_cg13.pro文件，添加代码

QT += opengl

LIBS += -lopengl32 -lglu32 -lglut

1. 在widgetDraw.h中添加

#include <QColor>

#include <QGLWidget>

#include <glut.h>

1. 修改widgetDraw类的基类及构造函数

class widgetDraw : public QGLWidget

widgetDraw::**widgetDraw**(QWidget \*parent) : QGLWidget(parent)

1. 打开widgetDraw.h文件，为widgetDraw类添加成员

private:

bool lightEnable;

bool doubleFaceEnable;

bool infiniteEnable;

bool globalAmbientEnable;

QColor diffuseColor;

QColor specularColor;

QColor ambientColor;

int comboIndex;

public:

void **setLightEnable**(bool flag);

void **setDoubleFaceEnable**(bool flag);

void **setInfiniteEnable**(bool flag);

void **setGlobalAmbientEnable**(bool flag);

void **setDiffuseColor**(QColor clr);

void **setSpecularColor**(QColor clr);

void **setAmbientColor**(QColor clr);

void **setComboIndex**(int n);

void ***paintGL***();

void **colorChange**(QColor clr, int mode);

void **render**(int index);

void **initLight**();

void **DrawScene**();

1. 打开widgetDraw.cpp文件，实现各函数

#include "widgetdraw.h"

widgetDraw::**widgetDraw**(QWidget \*parent) : QGLWidget(parent)

{

lightEnable = false;

doubleFaceEnable = false;

infiniteEnable = false;

globalAmbientEnable = false;

diffuseColor = QColor(Qt::black);

specularColor = QColor(Qt::white);

ambientColor = QColor(Qt::white);

comboIndex = -1;

setFormat(QGLFormat(QGL::DoubleBuffer | QGL::DepthBuffer));

}

void widgetDraw::**setLightEnable**(bool flag)

{

lightEnable = flag;

update();

}

void widgetDraw::**setDoubleFaceEnable**(bool flag)

{

doubleFaceEnable = flag;

update();

}

void widgetDraw::**setInfiniteEnable**(bool flag)

{

infiniteEnable = flag;

update();

}

void widgetDraw::**setGlobalAmbientEnable**(bool flag)

{

globalAmbientEnable = flag;

update();

}

void widgetDraw::**setDiffuseColor**(QColor clr)

{

diffuseColor = clr;

update();

}

void widgetDraw::**setSpecularColor**(QColor clr)

{

specularColor = clr;

update();

}

void widgetDraw::**setAmbientColor**(QColor clr)

{

ambientColor = clr;

update();

}

void widgetDraw::**setComboIndex**(int n)

{

comboIndex = n;

update();

}

void widgetDraw::***paintGL***()

{

glPolygonMode(GL\_FRONT, GL\_FILL); // 设置绘图模式

glPolygonMode(GL\_BACK, GL\_FILL);

glShadeModel(GL\_SMOOTH); // 平滑处理模式

glEnable(GL\_AUTO\_NORMAL);

glEnable(GL\_NORMALIZE); // 开启自动单位化

glFrontFace(GL\_CW);

glCullFace(GL\_BACK);

initLight(); //初始化光照设置

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

glClearColor(0.8f, 0.8f, 0.8f, 1.0); // 设置背景颜色

glViewport(0, 0, width(), height());

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(30.0, 1.0, 1.0, 128.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(0.0, 10.0, 10.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); // 设置观察者位置、方向。

glDrawBuffer(GL\_BACK);

glEnable(GL\_DEPTH\_TEST);

render(comboIndex);

if (lightEnable)

glEnable(GL\_LIGHTING);

else

glDisable(GL\_LIGHTING);

if (doubleFaceEnable)

glLightModelf(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_TRUE);

else

glLightModelf(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_FALSE);

if (infiniteEnable)

glLightModeli(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, GL\_FALSE);// 远视点

else

glLightModeli(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, GL\_TRUE); // 近视点

GLfloat model\_ambient[] = { 1.0, 1.0, 1.0, 1.0 };

if (globalAmbientEnable)

{

glPushAttrib(GL\_LIGHTING\_BIT);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, model\_ambient);

}

else

{

glPopAttrib();

colorChange(ambientColor, 1);

colorChange(diffuseColor, 2);

colorChange(specularColor, 3);

}

DrawScene();

}

void widgetDraw::**colorChange**(QColor clr, int mode)

{

GLfloat r, g, b;

r = clr.redF();

g = clr.greenF();

b = clr.blueF();

switch (mode)

{

case 1:

{

GLfloat tempAmbient[] = { r, g, b, 1.0f };

glLightfv(GL\_LIGHT0, GL\_AMBIENT, tempAmbient);

break;

}

case 2:

{

GLfloat tempDiffuse[] = { r, g, b, 1.0f };

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, tempDiffuse);

break;

}

case 3:

{

GLfloat tempSpecular[] = { r, g, b, 1.0f };

glLightfv(GL\_LIGHT0, GL\_SPECULAR, tempSpecular);

break;

}

default:

break;

}

}

void widgetDraw::**render**(int index)

{

GLfloat m\_MaterialAmbient[][4] = {

{ 0.2f,0.2f,0.2f,1.0f },

{ 0.25f,0.20725f,0.20725f,1.0f },

{ 0.33f,0.22f,0.028f,1.0f },

{ 0.05f,0.05f,0.0f,1.0f },

{ 0.25f,0.25f,0.25f,1.0f },

{ 0.02f,0.17f,0.02f,1.0f }

};

GLfloat m\_MaterialDiffuse[][4] = {

{ 0.8f,0.8f,0.8f,1.0f },

{ 1.0f,0.829f,0.829f,1.0f },

{ 0.78f,0.57f,0.11f,1.0f },

{ 0.5f,0.5f,0.4f,1.0f },

{ 0.4f,0.4f,0.4f,1.0f },

{ 0.07f,0.61f,0.07f,1.0f }

};

GLfloat m\_MaterialSpecular[][4] = {

{ 0.0f,0.0f,0.0f,1.0f },

{ 0.297f,0.297f,0.297f,1.0f },

{ 0.99f,0.94f,0.8f,1.0f },

{ 0.7f,0.7f,0.04f,1.0f },

{ 0.77f,0.77f,0.77f,1.0f },

{ 0.63f,0.73f,0.63f,1.0f }

};

// 高光系数，即镜面反射的指数

GLfloat m\_MaterialShininess[] = { 0.0f,0.01f,0.22f,0.08f,0.6f,0.6f };

glMaterialfv(GL\_FRONT, GL\_AMBIENT, m\_MaterialAmbient[index]);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, m\_MaterialDiffuse[index]);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, m\_MaterialSpecular[index]);

glMaterialf(GL\_FRONT, GL\_SHININESS, m\_MaterialShininess[index] \* 128.0f);

}

void widgetDraw::**initLight**()

{

GLfloat ambient[] = { 0.0f, 0.0f, 0.0f, 0.0f }; // 环境光颜色

GLfloat diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f }; // 漫反射颜色

GLfloat specular[] = { 1.0f, 1.0f, 1.0f, 1.0f }; // 镜面反射颜色

GLfloat light\_pos[] = { 0.0f, 3.0f, 3.0f, 0.0f };// 光源位置

glClearDepth(1.0f);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, specular);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_pos);

glEnable(GL\_LIGHT0); // 激活光源0

glEnable(GL\_LIGHTING); // 开启光照

}

void widgetDraw::**DrawScene**()

{

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

glColor3f(0.0, 1.0, 0.0);

GLfloat m\_BackDiffuse[] = { 0.8f,0.8f,0.8f,1.0f };

GLfloat m\_emission[] = { 1.0f, 0.2f, 0.2f, 1.0f };

glMaterialfv(GL\_BACK, GL\_DIFFUSE, m\_BackDiffuse);

glPushMatrix();

glFrontFace(GL\_CW);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(1.5, 0.0, -15.0);

glRotatef(40.0f, 1.0f, 0.0f, 0.0f);

GLUquadricObj \*objCylinder = gluNewQuadric();

gluCylinder(objCylinder, 1.6f,1.6f,1.4f,32,32);

glPopMatrix();

glFlush();

}

1. 打开MainWindow.h文件，添加数据成员

private:

QColor diffuseColor;

QColor specularColor;

QColor ambientColor;

1. 打开MainWindow.cpp文件，在MainWindow的构造函数中添加代码

ui->frameColor\_diffuse->setPalette(QPalette(Qt::black));

ui->frameColor\_specular->setPalette(QPalette(Qt::white));

ui->frameColor\_ambient->setPalette(QPalette(Qt::white));

1. 鼠标右击各控件，添加槽函数

private slots:

void **on\_checkBox\_lightEnable\_stateChanged**(int arg1);

void **on\_checkBox\_doubleFace\_stateChanged**(int arg1);

void **on\_checkBox\_infinite\_stateChanged**(int arg1);

void **on\_checkBox\_global\_stateChanged**(int arg1);

void **on\_pushButton\_diffuse\_clicked**();

void **on\_pushButton\_specular\_clicked**();

void **on\_pushButton\_ambient\_clicked**();

void **on\_comboBox\_material\_activated**(int index);

1. 打开MainWindow.cpp文件，实现槽函数

void MainWindow::**on\_checkBox\_lightEnable\_stateChanged**(int arg1)

{

if(arg1)

ui->widget\_draw->setLightEnable(true);

else

ui->widget\_draw->setLightEnable(false);

}

void MainWindow::**on\_checkBox\_doubleFace\_stateChanged**(int arg1)

{

if(arg1)

ui->widget\_draw->setDoubleFaceEnable(true);

else

ui->widget\_draw->setDoubleFaceEnable(false);

}

void MainWindow::**on\_checkBox\_infinite\_stateChanged**(int arg1)

{

if(arg1)

ui->widget\_draw->setInfiniteEnable(true);

else

ui->widget\_draw->setInfiniteEnable(false);

}

void MainWindow::**on\_checkBox\_global\_stateChanged**(int arg1)

{

if(arg1)

ui->widget\_draw->setGlobalAmbientEnable(true);

else

ui->widget\_draw->setGlobalAmbientEnable(false);

}

void MainWindow::**on\_pushButton\_diffuse\_clicked**()

{

diffuseColor = QColorDialog::getColor(static\_cast<int>(Qt::black));

ui->frameColor\_diffuse->setPalette(QPalette(diffuseColor));

ui->widget\_draw->setDiffuseColor(diffuseColor);

}

void MainWindow::**on\_pushButton\_specular\_clicked**()

{

specularColor = QColorDialog::getColor(static\_cast<int>(Qt::black));

ui->frameColor\_specular->setPalette(QPalette(specularColor));

ui->widget\_draw->setSpecularColor(specularColor);

}

void MainWindow::**on\_pushButton\_ambient\_clicked**()

{

ambientColor = QColorDialog::getColor(static\_cast<int>(Qt::black));

ui->frameColor\_ambient->setPalette(QPalette(ambientColor));

ui->widget\_draw->setAmbientColor(ambientColor);

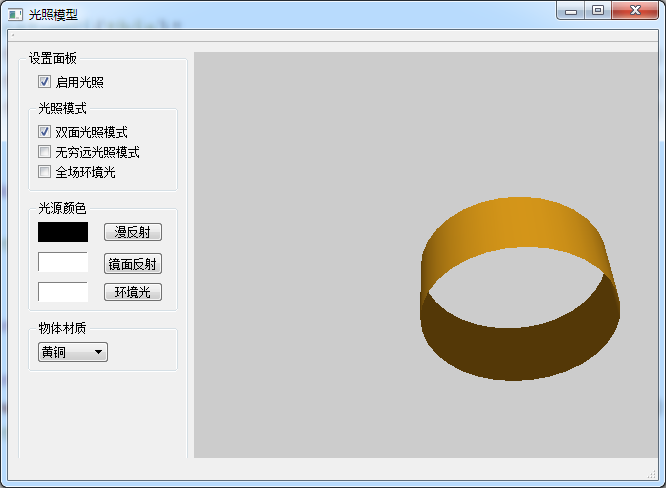
}

void MainWindow::**on\_comboBox\_material\_activated**(int index)

{

ui->widget\_draw->setComboIndex(index);

}

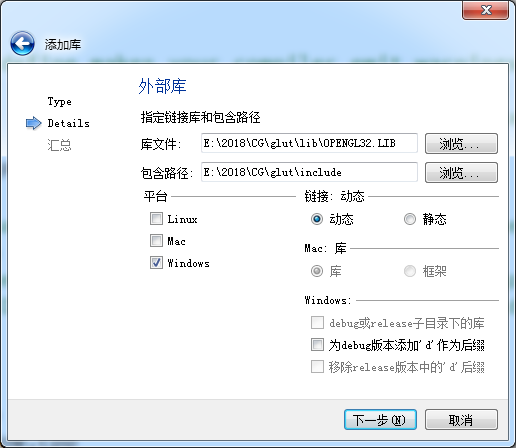


## 实验18 阴影Shade

1. 创建一个MainWindow程序，名为shade\_cg14
2. 添加opengl库，假设文件路径为E:\cg\glut。打开shad3\_cg14.pro文件，先添加一行代码

QT += opengl

1. 在shad3\_cg14.pro文件中点击鼠标右键，添加外部库，在E:\cg\glut\lib中选择opengl32.lib，这是opengl的基本库，首先将其添加进来；然后再依次添加glu32.lib和glut32.lib



1. 打开ui，进入设计模式，添加主菜单“设置”，添加菜单项“线框模式”，“表面模式”，修改菜单项对象名称为action\_lineFrame和action\_surface
2. 在shade\_cg14文件夹中，建立一个文件夹img，将两张图片文件放入其中，lineFrame.png和surface.png
3. 创建资源。点击菜单“新建文件或项目”，创建Qt资源，资源文件名称为image.qrc
4. 右击image.qrc，增加前缀img，导入图片文件lineFrame.png和surface.png
5. 回到设计模式，双击action\_lineFrame，添加lineFrame.png位图；同样添加surface.png
6. 将action\_lineFrame和action\_surface拖入工具条
7. 在设计模式下，拖入一个Widget控件，名称为widget\_draw
8. 创建一个新类widgetDraw，基类为QWidget
9. 将控件widget\_draw提升为widgetDraw类
10. 打开mainWindow.h文件，添加

void ***paintEvent***(QPaintEvent\*);

1. 打开mainWindow.cpp文件，添加

void MainWindow::***paintEvent***(QPaintEvent\*)

{

QPainter\* ptr = new QPainter(this);

ui->widget\_draw->setFixedSize(width(), height());

delete ptr;

}

1. 打开widgetDraw.h文件，添加

#include <QGLWidget>

#include <glut.h>

#include <QTime>

#include <QMouseEvent>

#include <QPainter>

typedef GLfloat Point3[3];

typedef GLfloat Point4[4];

typedef GLfloat Matrix[16];

1. 为widgetDraw.h添加私有数据成员

private:

GLUquadricObj\* pSphere;

float m\_angle;

bool isLine;

bool isTimer;

int id;

1. 添加成员函数

void **ground**();

void **setLight**();

void **GetShadeMatrix**(Point4 lightPos, Point4 planeEquation, Matrix desMatrix);

void **setLineFrame**();

void **setSurface**();

void ***paintGL***();

void ***timerEvent***(QTimerEvent\*);

void ***mousePressEvent***(QMouseEvent \*event);

void ***mouseReleaseEvent***(QMouseEvent \*event);

1. 实现各函数

widgetDraw::**widgetDraw**(QWidget \*parent) : QGLWidget(parent)

{

isLine = false;

m\_angle = 0.0;//初始化旋转角

id = startTimer(11);

isTimer = true;

}

void widgetDraw::**setLight**()

{

GLfloat mat\_specular[] = {0.0,0.0,0.0,1.0};

GLfloat mat\_shininess[] = {5.0};

GLfloat light\_position[] = {4.0,4.0,4.0,0.0};

GLfloat light\_color[] = {1.0,0.0,1.0,1.0};

GLfloat lmodel\_ambient[] = {0.2f,0.2f,0.2f,1.0};

glClearColor(0.0,0.0,1.0,1.0);

glShadeModel(GL\_SMOOTH);

glMaterialfv(GL\_FRONT,GL\_SPECULAR,mat\_specular);

glMaterialfv(GL\_FRONT,GL\_SHININESS,mat\_shininess);

glLightfv(GL\_LIGHT0,GL\_POSITION,light\_position);

glLightfv(GL\_LIGHT0,GL\_DIFFUSE,light\_color);

glLightfv(GL\_LIGHT0,GL\_SPECULAR,light\_color);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT,lmodel\_ambient);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_DEPTH\_TEST);

}

void widgetDraw::**ground**()

{

glShadeModel(GL\_SMOOTH);

GLfloat fsize = 200.0, fyPos = -11;

glBegin(GL\_QUADS);

glColor3f(0.0,1.0,0.0);

glVertex3f( fsize, fyPos, fsize);

glColor3f(0.0,1.0,1.0);

glVertex3f( fsize, fyPos, -fsize);

glColor3f(0.0,1.0,0.0);

glVertex3f(-fsize, fyPos, -fsize);

glColor3f(0.0,1.0,1.0);

glVertex3f(-fsize, fyPos, fsize);

glEnd();

}

void widgetDraw::**GetShadeMatrix**(Point4 lightPos, Point4 planeEquation, Matrix desMatrix)

{

GLfloat dot;

//点积，即数量积

dot = planeEquation[0] \* lightPos[0] +

planeEquation[1] \* lightPos[1] +

planeEquation[2] \* lightPos[2] +

planeEquation[3] \* lightPos[3];

desMatrix[0] = dot - lightPos[0] \* planeEquation[0];

desMatrix[4] = 0.0f - lightPos[0] \* planeEquation[1];

desMatrix[8] = 0.0f - lightPos[0] \* planeEquation[2];

desMatrix[12] = 0.0f - lightPos[0] \* planeEquation[3];

desMatrix[1] = 0.0f - lightPos[1] \* planeEquation[0];

desMatrix[5] = dot - lightPos[1] \* planeEquation[1];

desMatrix[9] = 0.0f - lightPos[1] \* planeEquation[2];

desMatrix[13] = 0.0f - lightPos[1] \* planeEquation[3];

desMatrix[2] = 0.0f - lightPos[2] \* planeEquation[0];

desMatrix[6] = 0.0f - lightPos[2] \* planeEquation[1];

desMatrix[10] = dot - lightPos[2] \* planeEquation[2];

desMatrix[14] = 0.0f - lightPos[2] \* planeEquation[3];

desMatrix[3] = 0.0f - lightPos[3] \* planeEquation[0];

desMatrix[7] = 0.0f - lightPos[3] \* planeEquation[1];

desMatrix[11] = 0.0f - lightPos[3] \* planeEquation[2];

desMatrix[15] = dot - lightPos[3] \* planeEquation[3];

}

void widgetDraw::**setLineFrame**()

{

isLine = true;

update();

if(!isTimer){

id = startTimer(11);

isTimer = true;

}

}

void widgetDraw::**setSurface**()

{

isLine = false;

update();

if(!isTimer){

id = startTimer(11);

isTimer = true;

}

}

void widgetDraw::***timerEvent***(QTimerEvent\*)

{

m\_angle += 1.0;

update();

}

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

{

if(event->button() == Qt::LeftButton){

if(!isTimer){

id = startTimer(11);

isTimer = true;

}

}

else if(event->button() == Qt::RightButton){

if(isTimer){

killTimer(id);

isTimer = false;

}

}

}

void widgetDraw::***mouseReleaseEvent***(QMouseEvent \*event)

{

Q\_UNUSED(event);

}

void widgetDraw::***paintGL***()

{

setLight(); //设置光照模型参数

pSphere = gluNewQuadric();

if (isLine)

gluQuadricDrawStyle(pSphere, GLU\_LINE);//设置线框的绘制方式

else

gluQuadricDrawStyle(pSphere, GLU\_FILL);//设置曲面的绘制方式

glViewport(0, 0, width(), height());

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(60.0, (float)width() / (float)height(), 1.0, 200.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0.0, 0.0, -20.0);

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

GLfloat light\_position[4];

Matrix desMatrix;

Point4 planeEquation = { 0.0,1.0,0.0,10.0 }; //平面方程为z+10=0.

glGetLightfv(GL\_LIGHT0, GL\_POSITION, light\_position); //获得光源位置

GetShadeMatrix(light\_position, planeEquation, desMatrix);

//绘制光源

glPushMatrix();

glColor3f(1.0, 0.0, 1.0);

glDisable(GL\_LIGHTING);

glDisable(GL\_DEPTH\_TEST);

glTranslatef(light\_position[0], light\_position[1], light\_position[2]);

gluSphere(pSphere, 0.2, 24, 18);

glEnable(GL\_LIGHTING);

glDisable(GL\_DEPTH\_TEST);

glPopMatrix();

//绘制地面

glPushMatrix();

glDisable(GL\_LIGHTING);

glDisable(GL\_DEPTH\_TEST);

ground();

glEnable(GL\_LIGHTING);

glDisable(GL\_DEPTH\_TEST);

glPopMatrix();

//绘制阴影

glPushMatrix();

glColor3f(0.5, 0.5, 0.5);

glDisable(GL\_LIGHTING);

glDisable(GL\_DEPTH\_TEST);

glMultMatrixf(desMatrix);

glRotatef(m\_angle, 0.0, 1.0, 0.0);

glRotatef(45.0, 1.0, 0.0, 0.0);

gluCylinder(pSphere, 0.8, 2.0, 4.0, 20, 10);

glEnable(GL\_LIGHTING);

glEnable(GL\_DEPTH\_TEST);

glPopMatrix();

//绘制原物体

glPushMatrix();

glRotatef(m\_angle, 0.0, 1.0, 0.0);

glRotatef(45.0, 1.0, 0.0, 0.0);

gluCylinder(pSphere, 0.8, 2.0, 4.0, 20, 10);

glPopMatrix();

}

1. 在mainWindow中为工具条按钮添加槽函数

void MainWindow::**on\_action\_lineFrame\_triggered**()

{

ui->widget\_draw->setLineFrame();

}

void MainWindow::**on\_action\_surface\_triggered**()

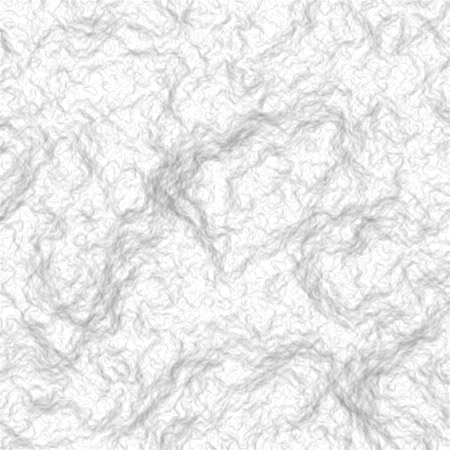
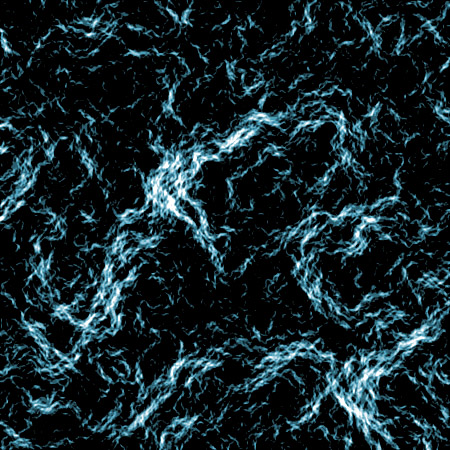
{

ui->widget\_draw->setSurface();

}

## 实验19 纹理实验

1. 创建一个Qt Widget Application项目texDemo\_cg15，基类选择QWidget，不保留ui界面
2. 在工程目录下新建文件夹images，在文件夹中添加图片1.bmp和2.bmp



1. 添加资源文件textures.qrc，并加入图片1.bmp和2.bmp
2. 在widget.h中添加：

包含库：

#include <QOpenGLWidget>

#include <QOpenGLFunctions>

#include <QOpenGLBuffer>

#include <QKeyEvent>

#include <QtMath>

#include <QOpenGLTexture>

#include <QOpenGLShaderProgram>

1. 在widget添加private成员：

QColor clearColor; //背景颜色

int xRot; //x轴旋转量

int yRot; //y轴旋转量

int zRot; //z轴旋转量

QOpenGLTexture \*texForDesk; //桌子纹理指针

QOpenGLTexture \*texForBall; //球体纹理指针

QOpenGLShaderProgram \*program; //着色器

QOpenGLBuffer buffer；//绘图缓冲

protected成员：

void ***initializeGL***() override;

void ***paintGL***() override;

void ***resizeGL***(int width, int height) override;

void ***keyPressEvent***(QKeyEvent \*event);

void **makeDeskAndBall**(); //建立桌子和球的模型

6. 在widget.cpp中添加函数实现：

Widget::**Widget**(QWidget \*parent)

: QOpenGLWidget(parent),

clearColor(Qt::yellow), //设置背景颜色为黄色（可替换）

xRot(70\*16), //初始的旋转角，从物体上方观察

yRot(0),

zRot(40\*16),

program(0)

{

}

Widget::~***Widget***()

{

makeCurrent();

buffer.destroy();

delete texForDesk;

delete texForBall;

delete program;

doneCurrent();

}

void Widget::***initializeGL***()

{

initializeOpenGLFunctions();

makeDeskAndBall();//生成物体模型

glEnable(GL\_DEPTH\_TEST); //开启深度测试

glEnable(GL\_CULL\_FACE); //开启消隐

#define PROGRAM\_VERTEX\_ATTRIBUTE 0

#define PROGRAM\_TEXCOORD\_ATTRIBUTE 1

//对着色器进行设置

QOpenGLShader \*vshader = new QOpenGLShader(QOpenGLShader::Vertex, this); //设置顶点着色器，对输入的顶点进行处理

const char \*vsrc =

"attribute highp vec4 vertex;\n"

"attribute mediump vec4 texCoord;\n"

"varying mediump vec4 texc;\n"

"uniform mediump mat4 matrix;\n"

"void main(void)\n"

"{\n"

" gl\_Position = matrix \* vertex;\n"

" texc = texCoord;\n"

"}\n";

vshader->compileSourceCode(vsrc);

QOpenGLShader \*fshader = new QOpenGLShader(QOpenGLShader::Fragment, this); //设置片元着色器，对纹理进行插值等处理

const char \*fsrc =

"uniform sampler2D texture;\n"

"varying mediump vec4 texc;\n"

"void main(void)\n"

"{\n"

" gl\_FragColor = texture2D(texture, texc.st);\n"

"}\n";

fshader->compileSourceCode(fsrc);

program = new QOpenGLShaderProgram;

program->addShader(vshader); //添加着色器

program->addShader(fshader); //添加着色器

program->bindAttributeLocation("vertex", PROGRAM\_VERTEX\_ATTRIBUTE); //绑定顶点坐标属性

program->bindAttributeLocation("texCoord", PROGRAM\_TEXCOORD\_ATTRIBUTE); //绑定纹理坐标属性

program->*link*();//连接到上面加入的两个着色器

program->bind();//将着色器程序绑定到活动的OpenGL环境，并成为当前着色器程序

program->setUniformValue("texture", 0); //将当前环境中的纹理设置为0

}

}

void Widget::***paintGL***()

{

glClearColor(clearColor.redF(), clearColor.greenF(), clearColor.blueF(), clearColor.alphaF());//设置背景颜色

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); //清空颜色缓存和深度缓存

QMatrix4x4 m; //变换矩阵

m.ortho(-0.5f, +0.5f, +0.5f, -0.5f, 4.0f, 15.0f); //设置视见体

m.translate(0.0f, 0.0f, -10.0f);

m.rotate(xRot / 16.0f, 1.0f, 0.0f, 0.0f);

m.rotate(yRot / 16.0f, 0.0f, 1.0f, 0.0f);

m.rotate(zRot / 16.0f, 0.0f, 0.0f, 1.0f);

program->setUniformValue("matrix", m);

program->enableAttributeArray(PROGRAM\_VERTEX\_ATTRIBUTE); //启用顶点坐标属性

program->enableAttributeArray(PROGRAM\_TEXCOORD\_ATTRIBUTE); //启用纹理坐标属性

program->setAttributeBuffer(PROGRAM\_VERTEX\_ATTRIBUTE, GL\_FLOAT, 0, 3, 5 \* sizeof(GLfloat)); //设置数组，以float为数据类型，5个为一组，前三个为顶点坐标

program->setAttributeBuffer(PROGRAM\_TEXCOORD\_ATTRIBUTE, GL\_FLOAT, 3 \* sizeof(GLfloat), 2, 5 \* sizeof(GLfloat));

//设置数组，以float为数据类型，5个为一组，后两个为纹理坐标

for (int i = 0; i < 1026 ; ++i) {

if(i<26)

texForDesk->bind();//绑定桌面的纹理

else

texForBall->bind();//绑定球面的纹理

glDrawArrays(GL\_TRIANGLE\_FAN, i \* 4, 4); //每四个点画一个平面

}

}

void Widget::***resizeGL***(int width, int height)

{

glViewport(0,0,width,height);

}

void Widget::***keyPressEvent***(QKeyEvent \*event) //实现桌子的旋转

{

if(event->key()==Qt::Key\_Right)

zRot-=10\*16; //旋转10°，16=1°

if(event->key()==Qt::Key\_Left)

zRot+=10\*16;

update();

}

void Widget::**makeDeskAndBall**()

{

static const int coords[26][4][3] = {//定义一个26个面的桌子，每个面的点按逆时针方向排序，让计算机知道哪一边是外面

{{5,5,8},{5,-5,8},{-5,-5,8},{-5,5,8}},

{{-5,5,7},{-5,-5,7},{5,-5,7},{5,5,7}},

{{5,5,8},{5,5,7},{5,-5,7},{5,-5,8}},

{{5,-5,8},{5,-5,7},{-5,-5,7},{-5,-5,8}},

{{-5,-5,8},{-5,-5,7 },{-5,5,7},{-5,5,8}},

{{-5,5,8},{-5,5,7},{5,5,7},{5,5,8}},

{{4,4,0},{3,4,0},{3,3,0},{4,3,0}},

{{4,4,7},{4,4,0},{4,3,0},{4,3,7}},

{{4,3,7},{4,3,0},{3,3,0},{3,3,7}},

{{3,3,7},{3,3,0},{3,4,0},{3,4,7}},

{{3,4,7},{3,4,0},{4,4,0},{4,4,7}},

{{-4,4,0},{-4,3,0},{-3,3,0},{-3,4,0}},

{{-4,4,0},{-4,4,7},{-4,3,7},{-4,3,0}},

{{-4,3,0},{-4,3,7},{-3,3,7},{-3,3,0}},

{{-3,3,0},{-3,3,7},{-3,4,7},{-3,4,0}},

{{-3,4,0},{-3,4,7},{-4,4,7},{-4,4,0}},

{{-3,-4,0},{-3,-3,0},{-4,-3,0},{-4,-4,0}},

{{-4,-4,7},{-4,-4,0},{-4,-3,0},{-4,-3,7}},

{{-4,-3,7},{-4,-3,0},{-3,-3,0},{-3,-3,7}},

{{-3,-3,7},{-3,-3,0},{-3,-4,0},{-3,-4,7}},

{{-3,-4,7},{-3,-4,0},{-4,-4,0},{-4,-4,7}},

{{4,-4,0},{4,-3,0},{3,-3,0},{3,-4,0}},

{{4,-4,0},{4,-4,7},{4,-3,7},{4,-3,0}},

{{4,-3,0},{4,-3,7},{3,-3,7},{3,-3,0}},

{{3,-3,0},{3,-3,7},{3,-4,7},{3,-4,0}},

{{3,-4,0},{3,-4,7},{4,-4,7},{4,-4,0}}

};

QVector<GLfloat> vertData;

//将点加入到数组中，之所以能这样加入点，是因为在前面已经设置过着色器、属性，并且还会在重绘函数中定义缓存数组的结构

for (int i = 0; i < 26; ++i) {

for (int j = 0; j < 4; ++j) {

// 定点

vertData.append(0.05 \* coords[i][j][0]); //乘0.05以调节物体大小

vertData.append(0.05 \* coords[i][j][1]);

vertData.append(0.05\* (coords[i][j][2]-6));

// 纹理坐标

vertData.append(j == 0 || j == 3);

vertData.append(j == 0 || j == 1);

}

}

float r,r1,k=0.08,h,h1,up=0.18; //k用于调整大小，up用于调整物体上下位置

for(int i=0;i<10;++i){ //将球体按照经度——维度模型分割成20\*50个平面加入到数组

r=qSin(M\_PI\_2\*i/10);

r1=qSin(M\_PI\_2\*(1+i)/10);

h=qCos(M\_PI\_2\*i/10);

h1=qCos(M\_PI\_2\*(1+i)/10);

for(int j=0;j<50;j++){

//北半球

//添加顶点坐标

vertData<<k\*r\*qCos(M\_PI\*(j+1)/25);

vertData<<k\*r\*qSin(M\_PI\*(j+1)/25);

vertData<<k\*h+up;

//添加纹理坐标，0~1.0

vertData<<(float)(j+1)/(float)51;

vertData<<(float)i/(float)21;

vertData<<k\*r1\*qCos(M\_PI\*(j+1)/25);

vertData<<k\*r1\*qSin(M\_PI\*(j+1)/25);

vertData<<k\*h1+up;

vertData<<(float)(j+1)/(float)51;

vertData<<(float)(i+1)/(float)21;

vertData<<k\*r1\*qCos(M\_PI\*j/25);

vertData<<k\*r1\*qSin(M\_PI\*j/25);

vertData<<k\*h1+up;

vertData<<(float)j/(float)51;

vertData<<(float)(i+1)/(float)21;

vertData<<k\*r\*qCos(M\_PI\*j/25);

vertData<<k\*r\*qSin(M\_PI\*j/25);

vertData<<k\*h+up;

vertData<<(float)j/(float)51;

vertData<<(float)i/(float)21;

//南半球

vertData<<k\*r\*qCos(M\_PI\*j/25);

vertData<<k\*r\*qSin(M\_PI\*j/25);

vertData<<-k\*h+up;

vertData<<(float)j/(float)51;

vertData<<(float)(21-i)/(float)21;

vertData<<k\*r1\*qCos(M\_PI\*j/25);

vertData<<k\*r1\*qSin(M\_PI\*j/25);

vertData<<-k\*h1+up;

vertData<<(float)j/(float)51;

vertData<<(float)(21-i-1)/(float)21;

vertData<<k\*r1\*qCos(M\_PI\*(j+1)/25);

vertData<<k\*r1\*qSin(M\_PI\*(j+1)/25);

vertData<<-k\*h1+up;

vertData<<(float)(j+1)/(float)51;

vertData<<(float)(21-i-1)/(float)21;

vertData<<k\*r\*qCos(M\_PI\*(j+1)/25);

vertData<<k\*r\*qSin(M\_PI\*(j+1)/25);

vertData<<-k\*h+up;

vertData<<(float)(j+1)/(float)51;

vertData<<(float)(21-i)/(float)21;

}

}

texForDesk = new QOpenGLTexture(QImage(QString(":/images/1.bmp")).mirrored());//生成桌子纹理对象

texForBall = new QOpenGLTexture(QImage(QString(":/images/2.bmp")).mirrored());//生成球纹理对象

buffer.create();

buffer.bind();//绑定到OpenGL环境

buffer.allocate(vertData.constData(), vertData.count() \* sizeof(GLfloat)); //将数组加入到缓存中

}



## 实验20 贝塞尔曲线

1. 创建一个Qt Widget Application项目bezierDemo\_cg16，基类选择QWidget，不保留ui界面
2. 在widget.h中添加：

包含库：

#include <QTimer>

#include <QPainter>

#include <QMouseEvent>

#define INTERAL 500//画线的点的数量，表示每一次用多少个点来画贝塞尔曲线

public成员：

void ***paintEvent***(QPaintEvent \*event);

void ***mousePressEvent***(QMouseEvent \*event);

void ***mouseMoveEvent***(QMouseEvent \*event);

void ***mouseReleaseEvent***(QMouseEvent \*event);

槽函数：

void **nextPoint**();//生成下一个点

private 成员：

QPoint point[4]; //控制点

QPoint auxPoint[6]; //辅助点

QPoint tem; //移动控制点时用的临时点

QVector<QPoint> bezier; //贝塞尔曲线

bool isMoving;

QTimer timer; //定时器

int order; //移动的是第几个控制点

int t; //插值公式的t\*INTERAL

1. 在widget.cpp里添加函数实现：

Widget::**Widget**(QWidget \*parent)

: QWidget(parent)

{

resize(800,600);

setPalette(QPalette(Qt::white));

setAutoFillBackground(true);

point[0]=QPoint(100,550); //控制点初始位置

point[1]=QPoint(230,200);

point[2]=QPoint(590,120);

point[3]=QPoint(700,500);

t=INTERAL-1;

isMoving=false;

nextPoint();

timer.start(8);

connect(&timer,SIGNAL(timeout()),this,SLOT(nextPoint()));

//连接定时器信号和槽函数

}

Widget::~***Widget***()

{

}

void Widget::***paintEvent***(QPaintEvent \*event)

{

QPainter \*pt=new QPainter(this);

QPen pen;

pen.setWidth(2); //设置线宽

pen.setColor(QColor(Qt::blue));

pt->setPen(pen);

pt->drawLine(point[0],point[1]);

pt->drawLine(point[1],point[2]);

pt->drawLine(point[2],point[3]);

pen.setColor(QColor(Qt::red));

pt->setPen(pen);

for(int i=0;i<bezier.length();++i){

pt->drawPoint(bezier[i]); //画点和周围8个点

pt->drawPoint(bezier[i]+QPoint(-1,-1));

pt->drawPoint(bezier[i]+QPoint(0,-1));

pt->drawPoint(bezier[i]+QPoint(1,-1));

pt->drawPoint(bezier[i]+QPoint(-1,0));

pt->drawPoint(bezier[i]+QPoint(1,0));

pt->drawPoint(bezier[i]+QPoint(-1,1));

pt->drawPoint(bezier[i]+QPoint(0,1));

pt->drawPoint(bezier[i]+QPoint(1,1));

}

pen.setColor(QColor(Qt::green));

pt->setPen(pen);

pt->drawLine(auxPoint[0],auxPoint[1]);

pt->drawLine(auxPoint[1],auxPoint[2]);

pen.setColor(QColor(Qt::yellow));

pt->setPen(pen);

pt->drawLine(auxPoint[3],auxPoint[4]);

pen.setColor(QColor(Qt::black));

pen.setWidth(1);

pt->setPen(pen);

for(int i=0;i<4;++i)

pt->drawEllipse(point[i].x()-5,point[i].y()-5,10,10);

for(int i=0;i<6;++i)

pt->drawEllipse(auxPoint[i].x()-5,auxPoint[i].y()-5,10,10);

if(isMoving) {//如果点被移动，在被移动的点和鼠标当前位置画一条虚线

pen.setStyle(Qt::DashLine); //将线性设置为虚线

pt->setPen(pen);

pt->drawEllipse(tem.x()-5,tem.y()-5,10,10);

pt->drawLine(tem,point[order]);

}

}

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

{

QPoint p=event->pos();

if((point[0].x()-p.x())\*(point[0].x()-p.x())+(point[0].y()-p.y())\*(point[0].y()-p.y())<25)

{

order=0;

isMoving=true;

return;

}

if((point[1].x()-p.x())\*(point[1].x()-p.x())+(point[1].y()-p.y())\*(point[1].y()-p.y())<25)

{

order=1;

isMoving=true;

return;

}

if((point[2].x()-p.x())\*(point[2].x()-p.x())+(point[2].y()-p.y())\*(point[2].y()-p.y())<25)

{

order=2;

isMoving=true;

return;

}

if((point[3].x()-p.x())\*(point[3].x()-p.x())+(point[3].y()-p.y())\*(point[3].y()-p.y())<25)

{

order=3;

isMoving=true;

return;

}

}

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

{

if(isMoving)tem=event->pos();

update();

}

void Widget::***mouseReleaseEvent***(QMouseEvent \*event)

{

if(isMoving){

point[order]=tem; //把控制点移动到当前位置

isMoving=false; //重新绘制贝塞尔曲线

t=0;

nextPoint();

}

}

void Widget::**nextPoint**()

{

if(t==0){

t=INTERAL-1;

bezier.clear();

}

//用插值公式生成负责点坐标

auxPoint[0]=QPoint((point[0].x()\*t+point[1].x()\*(INTERAL-t))/INTERAL,(point[0].y()\*t+point[1].y()\*(INTERAL-t))/INTERAL);

auxPoint[1]=QPoint((point[1].x()\*t+point[2].x()\*(INTERAL-t))/INTERAL,(point[1].y()\*t+point[2].y()\*(INTERAL-t))/INTERAL);

auxPoint[2]=QPoint((point[2].x()\*t+point[3].x()\*(INTERAL-t))/INTERAL,(point[2].y()\*t+point[3].y()\*(INTERAL-t))/INTERAL);

auxPoint[3]=QPoint((auxPoint[0].x()\*t+auxPoint[1].x()\*(INTERAL-t))/INTERAL,(auxPoint[0].y()\*t+auxPoint[1].y()\*(INTERAL-t))/INTERAL);

auxPoint[4]=QPoint((auxPoint[1].x()\*t+auxPoint[2].x()\*(INTERAL-t))/INTERAL,(auxPoint[1].y()\*t+auxPoint[2].y()\*(INTERAL-t))/INTERAL);

auxPoint[5]=QPoint((auxPoint[3].x()\*t+auxPoint[4].x()\*(INTERAL-t))/INTERAL,(auxPoint[3].y()\*t+auxPoint[4].y()\*(INTERAL-t))/INTERAL);

bezier<<auxPoint[5];

--t;

update();

}

