《计算机图形学》 系统技术报告

161220135-吴德亚

一 综述

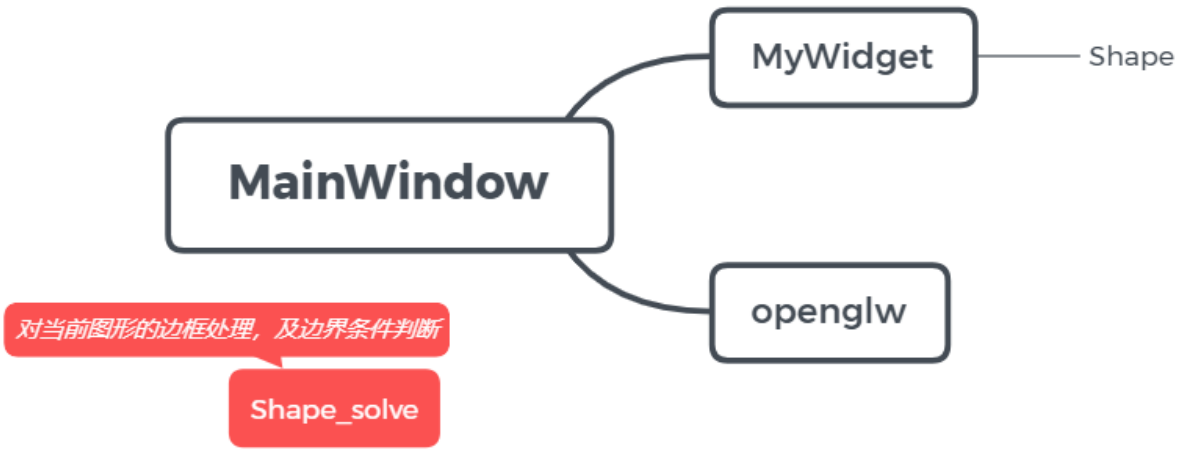
功能：

本次实验的开发平台为Qt, 编程语言为C++, 系统是win10, 完成了实验的基本要求，包括（直线，曲线，多边形，画笔）等的绘制，选中图形进行平移，旋转，拖放，放缩，填充，删除等，保存当前图画为图像，读取 .off 文件并显示三维图像。运用了课程中用到的图像生成算法。

主体思想：

将所有的图形先绘制在QImage上，然后通过paint函数显示在QWidget 界面上，所以都是在像素级别上进行操作。好处在于：可以方便地获取某一点的像素值（填充需要），并且方便保存。

二 整体框架，类之间的关系



MainWindow : 为主窗口，提供对用户的使用接口和画图展示，继承 **MyWidget**（二维图像）和 **openglw** (三维图像)，给出与两个子类相关联的槽函数。

MyWidget : 继承图形类 **Shape**, 管理图形，主要进行二维图形的绘制，操作。与**MainWindow**直接通信。

openglw : 主要进行三维图形读取与显示

Shape : 存储了与图形有关的所有信息。

三 类的数据成员与函数成员

1.MainWindow

```
public:
    explicit MainWindow(QWidget *parent=nullptr);
    ~MainWindow();
 signals:                // 通过connect 向 MyWidget类传递参数或发送信号
    void post_shape(int type);
    void post_color(QColor color);
    void post_back();
    void post_clear();
    void post_color2(QColor color);
    void save_file();   // 保存文件的signal

public slots:          // 用户点击界面中的图标触发槽函数
    void on_actionCircle_triggered();
    void on_actionRectangle_triggered();
    void on_actionLine_2_triggered();
    void on_actionColor_triggered();
    void on_actionback_triggered();
    void on_actionclear_triggered();
    void on_actionPen_triggered();
    void on_actionfilling_color_triggered();
    void on_actionsave_as_triggered();
    void on_actionCurve_triggered();
    void dot_line_triggered();
private slots:       // 2D, 3D触发
     void on_action2D_2_triggered();
     void on_action3D_2_triggered();
     void on_actionPolygon_triggered();   // 绘制多边形
private:
    Mywidget *mywidget;      // 2D 画图对象的指针
    openglw *glwidget;       // 3D 画图对象的指针
    QTimer clk;               // 定时器
    Ui::MainWindow*ui;
};
```

同时给出与 **Mywidget** 类connect 的函数（用户对图形行为的操作是通过界面的按钮触发的，而按钮发送的信号可以被**MyWidget, openglw** 类中的槽函数**SLOT** 接收。

```
    connect(this,SIGNAL(post_shape(int)),mywidget,SLOT(set_shape(int)));
    // 设置当前的图形形状
    connect(this,SIGNAL(post_color(QColor)),mywidget,SLOT(set_color(QColor)));
```

```
    // 设置当前的颜色
    connect(this,SIGNAL(post_back()),mywidget,SLOT(set_back()));
    // 回退操作
    connect(this,SIGNAL(post_clear()),mywidget,SLOT(set_clear()));
    // 清屏操作
    connect(this,SIGNAL(post_color2(QColor)),mywidget,SLOT(set_filling_color(QColor)));
    // 边框颜色选择器
    connect(this,SIGNAL(save_file()),mywidget,SLOT(set_save_file()));
    // 文件保存选项
    connect(spinbox,SIGNAL(valueChanged(int)),mywidget,SLOT(set_size(int)));
    // 边框尺寸选择
    connect(dot_line,SIGNAL(triggered()),this,SLOT(dot_line_triggered()));
    // 虚线框的绘制
    connect(my_cut,SIGNAL(triggered()),mywidget,SLOT(set_cutting()));
    // 裁剪选择
    connect(p_edit,SIGNAL(textChanged(QString)),mywidget,SLOT(set_degree(QString)));
    // 旋转角度输入
    connect(spining,SIGNAL(triggered()),mywidget,SLOT(set_spining()));
    // 旋转选择
    connect(&clk,SIGNAL(timeout()),glwidget,SLOT(update_3d()));
    // 显示3D图形
```

其中，给出的用户接口可以通过代码添加，也可以通过ui 文件添加。（在本实验中，大部分是ui文件添加的，在 QSpinBox, QLineEdit 方面则是代码手动添加）。

在本类中，比较重要的函数是：与openglw模块的交互。当需要实现3D图像旋转时，通过控制时钟来达到图像动态变化的目的。

```
    QFileDialog filelog;    // 打开文件对话框
    QString str = filelog.getOpenFileName(this,tr("open file"),".",tr("file(*.off)"));
    if(str == "")
    {
        QMessageBox::warning(this,tr("Warning"),tr("文件不能为空!"));
    }
    else
    {
        glwidget = new openglw(str);  // 创建新对象
        glwidget->show();             // 显示3D 图像
        clk.start(20);          // 设置20ms的timeout,当时间超过20ms时，更新3D图像
        connect(&clk,SIGNAL(timeout()),glwidget,SLOT(update_3d()));
    }
```

2. MyWidget :

给出该类的成员函数和数据成员：

```
    QColor get_from_point(int x,int y);     // 得到某一点的像素值（用于填充算法）
    void handle_paintevent();               // 处理 paintevent
    void addItem(QMouseEvent *m);          // 新建图形时添加
    void set_currentItem(QMouseEvent *m); // 为currentItem 赋值
    void shape_draging(QMouseEvent *m,Shape *current);  // 当前图形的拖拽
    void shape_resizing(QMouseEvent *m,Shape *current);  // 当前图形的放缩
```

```
    void mousePressEvent(QMouseEvent *m);       // 重载鼠标按下事件
    void mouseMoveEvent(QMouseEvent *m);        // 重载鼠标移动事件
    void mouseReleaseEvent(QMouseEvent *m);     // 重载鼠标松开事件
    void paintEvent(QPaintEvent *event);        // 重载绘制事件
    void shape_clear();                         // 清屏
    void shape_back();                          // 图形删除
    void get_fill_item(int x,int y,QColor before_color,QColor new_color); // 填充
    void print_all_fillings(Shape *fill_shape);
    void shape_spining();                        // 图形旋转
    void shape_cutting();                        // 图形裁剪
    void set_flags(QPoint a,bool* p,Shape *rect_dot);
    bool judge_safe(bool *start,bool *end);
    bool judge_inside(bool *start);
    void get_cut_point(vector<QPoint>&cut_point,QPoint a,QPoint b,
                       const Shape *rect_dot);
```

槽函数： 从**MainWindow** 类中接收用户的操作，并为新生成的类传递参数

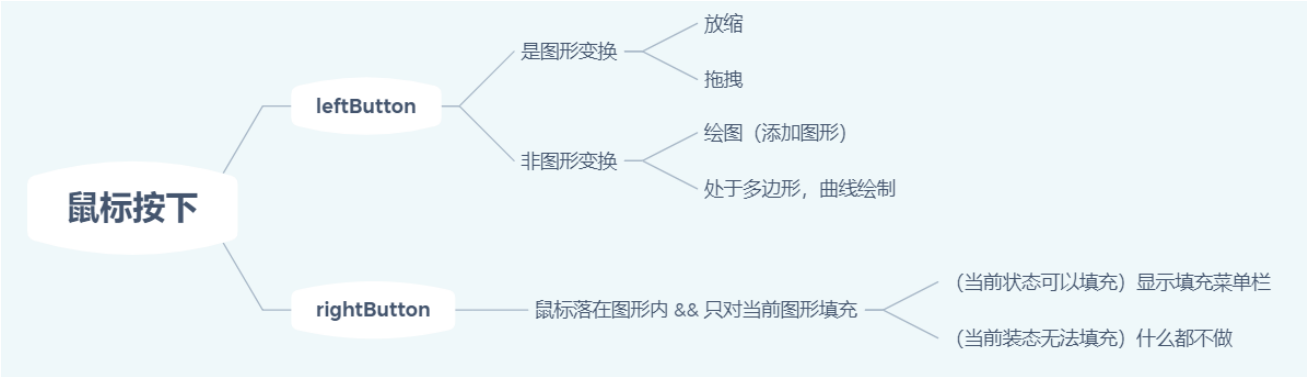
```
public slots:
    void set_save_file();
    void receive_fill_siganl();
    void receive_not_fill_signal();
    void set_degree(QString degree);
    void set_spining();          // 旋转槽函数
    void set_cutting();        // 裁剪槽函数
    void set_shape(int type);
    void set_color(QColor mycolor);
    void set_size(int tsize){ size=tsize;}
    void set_filling_color(QColor filling_color);
    void set_clear();
    void set_back();
```

数据成员： 临时存储当前绘制图形的参数并保存为CurrentItem（指向当前图形的指针）

```
    int current_type,size;
    bool pressed,saving_file,is_curve;  // 从槽函数中获取
    QColor color;
    QColor s_filling_color;
    double angle;
    Shape *currentItem;         // 当前指向的Shape 对象
    Shape *rect_dot;            // 指向虚线框
    vector<Shape *>store_shape; // 存储管理所有的Shape
    bool filling,draging,resizing; // 标志当前处于哪种操作状态
    Shape_solve *mysolve;       // 图形边界处理对象的指针
    QPoint mouse_addr;          // 鼠标位置
    QString degree_get;
    QImage myimg;              // 所有图形保存在QImage 对象上
    bool is_polygon;
```

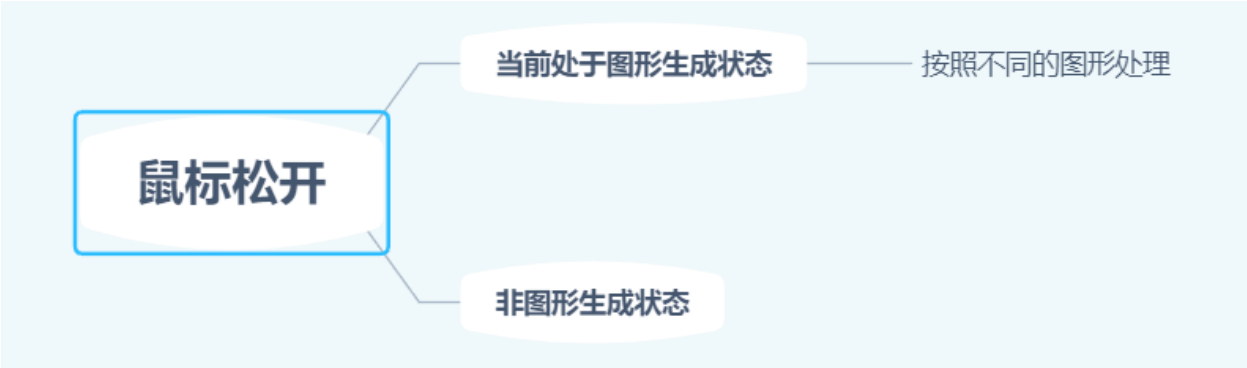
重要的函数解析：

1. 鼠标按下事件重载：*void Mywidget::mousePressEvent(QMouseEvent *m)*



2. 鼠标移动事件重载： *void Mywidget::mouseMoveEvent(QMouseEvent *m)*

3. 鼠标松开事件重载： *void Mywidget::mouseReleaseEvent(QMouseEvent *m)*



4. 图形拖拽算法：

计算当前鼠标相对之前移动的位置 diet， 然后给该图形的每个顶点进行相同的操作，再调用update()，重绘图形即可。

5. 图形缩放算法：

   以当前图形的几何中心为放缩中心，记录鼠标相对之前位置的x , y 放缩比例，然后对该图形的每个端点进行相同的操作，调用update()函数，重绘图形即可。

6. *paintEvent* 重载：

```
void Mywidget::paintEvent(QPaintEvent *event)
{
     myimg = QImage(this->width(),this->height(),QImage::Format_ARGB32);
     myimg.fill(Qt::white);
     handle_paintevent();      // 在这里处理绘图函数

    QPainter painter(this);
    painter.drawImage(QRect(0,0,this->width(),this->height()),myimg);
    // 将 myimg 图像显示在画布上
}
```

7.

shape_clear, shape_back 等函数都是从**store_shape**这个结构遍历图形，进行操作。

8. 图像的填充算法：（洪泛填充算法 Flood Fill Algorithm）

从一个像素点开始，将其附近的点填充成新的颜色，直到封闭区域内的所有像素点都被填充新颜色为止。在本次实验中，使用了栈结构。依次弹出点，并寻找它周围符合要求的四个点，填充完成后再压入栈中，为下一次操作准备。

```
void Mywidget::get_fill_item(int x,int y,QColor before_color,QColor new_color)
{
    stack <QPoint> fill_stack;
    QPoint p(x,y);          // 种子点
    fill_stack.push(p);
    myimg.setPixelColor(p.x(),p.y(),new_color);
    while(!fill_stack.empty())
    {
        p = fill_stack.top(); // 从栈顶获取一个点
        fill_stack.pop();
        QPoint will_filled[4]; // 使用四连通，获取它周围的四个点
         will_filled[0] = QPoint(p.x()-1,p.y());
         will_filled[1]=QPoint(p.x()+1,p.y());
         will_filled[2]=QPoint(p.x(),p.y()+1);
         will_filled[3]=QPoint(p.x(),p.y()-1);
         for(int i = 0;i < 4;i++)
         {     // 只有当前的点在边界内部
             if( will_filled[i].x() > 0 && will_filled[i].y() > 0
                     && will_filled[i].x()<this->width() && will_filled[i].y()<this->height())
             {
                 QColor p_color=get_from_point(will_filled[i].x(),will_filled[i].y());
                 if( p_color == before_color)  // 需要填充的点是否符合要求
                 {
                     fill_stack.push(will_filled[i]);
                     myimg.setPixelColor(will_filled[i].x(),will_filled[i].y(),new_color);
                     // 对myimg进行像素上的操作
                 }
             }
         }
    }
}
```

9. 图像旋转算法：找出几何中心，然后以它为中心，遍历该图形的所有顶点进行旋转，再重绘即可。

```
void set_point(QPoint &f,const QPoint &a,double centerx,double centery,double
angle)
{    //  带入公式
    f.rx() = centerx+(a.x()-centerx)*cos(angle)-(a.y()-centery)*sin(angle);
    f.ry() = centery+(a.x()-centerx)*sin(angle)+(a.y()-centery)*cos(angle);
}
```

10. 图像裁剪算法： *Cohen-Sutherland算法*

    将区域分成9块，每块进行编码。区域码位 = 0：端点不落在相应位置上；区域码位 = 1：端点落在相应位置上；通过得到startpoint, endponit的区域码，可以得出：

    （1）如果code1和code2均为0，则说明直线全部位于窗口内部 （2）如果code1和code2经过按位与运算后的结果code1&code2不等于0，说明两点同时在窗口的上方、下方、左方或右方，那么线段全部位于窗口的外部。 （3）如果上述两种条件均不成立，则进行求解：直线与四条边界的交点，该交点即为直线的新端点。

    给出代码的判断逻辑：

```
                if(!judge_safe(start,end))     // 需要进行裁剪
                {
                    vector<QPoint>cut_point;
                    get_cut_point(cut_point,a,b,rect_dot);
                    if(judge_inside(start) && judge_inside(end)) // 线段正好在框内
                    {
                            // 该线段不需要修改
                    }
                    else if(judge_inside(start))     // 起始点点在框内
```

```
                {
                    (*it)->endpoint = cut_point[0];
                }
                else if(judge_inside(end))      // 终止点在框内
                {
                    (*it)->startpoint = cut_point[0];
                }
                else                // 端点都不在框内
                {
                    (*it)->startpoint = cut_point[0];
                    (*it)->endpoint = cut_point[1];
                }
                it++;          // 遍历
            }
```

3. Shape :

成员函数： 主要包括了画笔，直线，矩形，圆，多边形，曲线的生成算法。

```
    void my_draw_point(int x,int y,QPainter *my_painter);  // 点
    void my_draw_pen(QPainter *my_painter);    // 画笔
    void my_draw_curve(QPainter *my_painter);  // 曲线
    void my_draw_line(QPainter *my_painter);   // 直线
    void drawing_line(QPoint start,QPoint end,QPainter *my_painter);
    void my_draw_ellipse(QPainter *my_painter);  // 圆
    void my_draw_rect(QPainter *my_painter);      // 矩形
    void my_draw_dot_line(QPainter *my_painter);  // 虚线框
    void my_draw_polygon(QPainter *my_painter);   // 多边形
    void set_curve_point();    //  曲线顶点设置
    void set_shape_point();     // 矩形顶点设置
```

成员数据：包括了对象的所有属性

```
    QPoint startpoint,endpoint;  // 起始点，终止点
    QColor shape_color;          // 图形颜色
    int shape_size;              // 图形尺寸
    int init_type;               // 当前图形形状
    vector<QPoint>my_pen;        // 画笔存储
    QColor shape_fill_color;     // 填充颜色
    bool enable_fill;            // 是否填充
    vector<QPoint>my_curve;      // 图形顶点存储
```

绘图函数解析：

1. 直线生成算法：**DDA 算法**，如果斜率绝对值大于1，每次y增加或减小一个单位，x增加或减小diet x，如果斜率绝对值小于1，每次y 增加或减小diet y, x增加或减小一个单位。

```
    int startx = start.x(),starty = start.y(),endx=end.x(),endy=end.y();
    int dietx =endx-startx,diety = endy-starty;
    int maxsteps = abs(dietx) > abs(diety)?abs(dietx):abs(diety); // 循环次数
    double x=startx,y=starty;
    double xi=((double)dietx)/maxsteps,yi=((double)diety)/maxsteps;
    for(int label = 1;label<=maxsteps;label++)
    {
        x += xi;
        y +=yi;
        my_painter->drawPoint(QPoint(x,y));  // 画点
    }
```

2. 中点椭圆画法：由于椭圆的对称性，只需要考虑第一象限即可。**关键**：需要根据椭圆切线的斜率绝对值是否大于1考虑，即如果|k| >1, 则需要每次y减少一个单位,x每次增加diet x,，如果|k| < 1， 则需要x每次增加一个单位，y每次增加diet y 即可。

```
    ..............................  // 给出1/8的实现
    double a1_p1 = ry*ry+rx*rx/4-rx*rx*ry;
    my_draw_point(center_x,center_y+y,my_painter);
    my_draw_point(center_x,center_y-y,my_painter);
    while(ry*ry*x < rx*rx*y)
    {
        if(a1_p1 < 0)
        {
            x = x+1;
            a1_p1 += ry*ry + 2*ry*ry*x;
        }
        else
        {
            x = x+1;
            y=y-1;
            a1_p1 += 2*ry*ry*x-2*rx*rx*y+ry*ry;
        }
        my_draw_point(center_x+x,center_y+y,my_painter);
        my_draw_point(center_x+x,center_y-y,my_painter);
        my_draw_point(center_x-x,center_y+y,my_painter);
        my_draw_point(center_x-x,center_y-y,my_painter);
    }
    ........................
```

3. 曲线生成算法： 贝塞尔曲线（4阶）

```
    for(int i = 1;i < n;i++)   // 带入公式运算即可
    {
        double u = i/n;
        draw_x = start_x*(pow(1-u,3)) + 3*mid_1_x*u*pow(1-u,2)
        + 3*mid_2_x*pow(u,2)*(1-u) + end_x*pow(u,3);
        draw_y = start_y*(pow(1-u,3)) + 3*mid_1_y*u*pow(1-u,2)
        +3*mid_2_y*pow(u,2)*(1-u) + end_y*pow(u,3);
        my_draw_point(draw_x,draw_y,my_painter);
    }
```

4. 多边形实现： 可以基于直线生成算法，循环遍历所有的顶点，一次连接即可，并不困难。

4. Shape_solve :

用在缩放，拖拽等需要边界判断的地方。

```
    bool Inside_Last_Shape(QPoint f,QPoint start,QPoint end,int size);
// 判断鼠标点击的点是否在当前图形内部
    bool On_Dot_Shape(QPoint f,int size); // 判断鼠标点击点是否在图形边缘的顶点上
    void paint_edge(Shape *current,QPainter *painter,bool polygon);  // 绘制虚线框
    void getdot_xy(int &x_max,int &x_min,int &y_max,int &y_min);
    // 得到当前图形的边界点
    private:
    int s_max_x,s_max_y,s_min_x,s_min_y;
    int dotsize;
```

虚线框的实现：

```
        for(int i=  0;i < curve_size;i++)   // 遍历所有点，找出该图形的边界
        {
            if(my_curve[i].x() > s_max_x)
            {
                s_max_x = my_curve[i].x();
            }
            ……
        }
```

包括所有图形的边框显示，虚线框都是用的这种算法。

5. openglw :

```
    void read_from_file(QString &fname);  // 读取文件
    void initializeGL();                  // opengl 初始化
    void resizeGL(int w,int h);
    void paintGL();                      // 绘制3D图形

public slots:
    void update_3d();
private:
    vector<Vertex*>points;   // 存储所有顶点
    int vertexs,faces,num;
    vector<vector<int>> face_points; // 存储所有构成面的点
    QString filename;
    int f_angle;            // 度数
```

重要函数的解析：

由于之前进行过文件读取，故此处不再赘述文件读取操作，只进行3D图形的显示。

1. **paintGL:**

```
void openglw::paintGL()
```

```
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotatef(f_angle,1.0f,1.0f,1.0f);        // 给定旋转中心

    for(int i = 0;i < faces;i++)
    {
        glBegin(GL_TRIANGLES);
        for(int j = 0;j < num;j++)
        {
            switch(j%num)
            {
            case 0: glColor3f(1.0f, 0.0f, 0.0f); ;break;
            case 1: glColor3f(0.0f, 1.0f, 0.0f); break;
            case 2: glColor3f(0.0f, 0.0f, 1.0f); break;
            }
            Vertex *p = points[face_points[i][j]];
            glVertex3f(p->x,p->y,p->z);    // 绘制点
        }
        glEnd();
    }
    f_angle  = (f_angle + 1)%360;     // 动态旋转的度数控制
    glFlush();
}
```

四 结语:

1. 首先，需要感谢图形学，让我完成了近2000行的代码量（在之前的所有编程作业中貌似只有PA可以比拟）
2. 其次，在此次代码编写过程中，阅读了很多次的开发手册，让我意识到了网上的博客，总结等不一定全面甚至正确，毕竟只是参考作用。
3. 锻炼了我的编程能力，学以致用，管理近2000行的代码，逻辑思维能力也得到了锻炼。
4. 最后，向给过我帮助的所有同学，文献作者们致谢！

五 参考文献:

贝塞尔曲线：https://blog.csdn.net/Jurbo/article/details/75069054

3D 显示 http://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_examples.html

QT + Opengl 实现框架https://blog.csdn.net/chaojiwudixiaofeixia/article/details/77917697

QT 官方文档 https://doc.qt.io/

QT 基础 https://chorior.github.io/2017/08/28/Qt5-basis/