泊松抠图程序设计书

功能说明与算法流程图等

1 程序简述

本程序实现了基本的全局泊松抠图，并提供了配套的 GUI：可以使用鼠标绘制抠图的未知区域和填充前景区域，自动生成蒙版图，从而进行泊松抠图。抠图结束，自动弹出窗口显示抠图结果及其透明通道图。

2 开发环境

操作系统	Windows 10 x64 家庭版
开发框架	Qt Community 5.7.0
集成开发环境	Qt Creator 4.0.2
编译环境	Microsoft Visual C++ 2015 64bit
第三方图像库	OpenCV 3.10 64bit (VC14)

3 功能说明

3.1 文件结构

PoissonMatting

•PoissonMatting.pro	工程文件
•main.cpp	主程序文件
•mainwindow.h/cpp	主窗口程序头文件/源文件
•mattingcanvas.h/cpp	抠图画布程序头文件/源文件
•poissonmatting.h/cpp	泊松抠图程序头文件/源文件
•poisson[1-6].png	泊松抠图效果图共6张
•mainwindow.ui	主界面文件
•poisson.ts	界面文本翻译源文件
•poisson.qm	界面文本翻译编译文件
•README.md	程序简述

Release

•PoissonMatting.exe	编译程序
•poisson.qm	界面文本翻译编译文件
•其他	相关依赖DLL及Qt依赖插件

注：发布目录 Release 是 PoissonMatting 工程目录的子目录。

3.2 工程文件：PoissonMatting.pro

3.2.1 Qt 配置项

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = PoissonMatting
TEMPLATE = app
```

使用 Qt 核心库和图形界面库，Qt 5 以上版本还需要使用控件库。

工程构建目标为 PoissonMatting，工程模板为应用程序。

3.2.2 目录配置

```
INCLUDEPATH += "D:/opencv/build/include"

LIBS += -LD:/opencv/build/x64/vc14/lib
Debug:LIBS += -lopencv_world310d
Release:LIBS += -lopencv_world310
```

分别配置了 OpenCV 的包含文件目录和库文件目录。

同时为“调试”和“发布”配置不同的链接库。

3.2.3 头/源文件配置

```
SOURCES += main.cpp\
        mainwindow.cpp \
    poissonmatting.cpp \
    mattingcanvas.cpp

HEADERS  += mainwindow.h \
    poissonmatting.h \
    mattingcanvas.h
```

3.2.4 界面文件

```
FORMS    += mainwindow.ui
```

3.2.5 其他文件

```
DISTFILES += \
    README.md \
    poisson1.png \
    poisson2.png \
    poisson3.png \
    poisson4.png \
    poisson5.png \
    poisson6.png
```

3.2.6 翻译源文件

```
TRANSLATIONS += poisson.ts
```

3.3 主程序：MAIN.CPP

3.3.1 主函数

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QTranslator qtTranslator;
    qtTranslator.load("poisson.qm");
    a.installTranslator(&qtTranslator);
    MainWindow w;
    w.show();
    return a.exec();
}
```

3.3.1.1 Qt 应用程序

```
QApplication a(argc, argv);
```

3.3.1.2 Qt 应用翻译

```
QTranslator qtTranslator;
qtTranslator.load("poisson.qm");
a.installTranslator(&qtTranslator);
```

3.3.1.3 显示主窗口

```
MainWindow w;
w.show();
```

3.3.1.4 应用程序执行

```
return a.exec();
```

3.4 主窗口程序：MAINWINDOW.H/CPP

3.4.1 头文件

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_actionOpen_triggered();
```

```
    void on_actionDraw_Unknown_triggered(bool checked);
    void on_actionMark_Foreground_triggered(bool checked);
    void on_actionClear_triggered();
    void on_actionExit_triggered();
    void on_actionMatte_triggered();
    void on_actionClear_Mark_triggered();
    void on_action_Add_Pen_Size_triggered();
    void on_action_Sub_Pen_Size_triggered();
    void on_actionAbout_triggered();

private:
    Ui::MainWindow *ui;
    QString filename;
    PoissonMatting pmat;
    MattingCanvas *canvas;
    QScrollArea *scrollArea;

};
```

公开成员函数有：构造函数和析构函数。

Qt 的私有槽函数，类似于事件处理函数，有对以下操作的触发处理：

- 打开图片
- 绘制未知区域
- 标记前景区域
- 清空前景标记
- 清空所有标记
- 退出
- 抠图
- 增大画笔大小
- 减小画笔大小
- 关于我

私有成员包括一些组件的变量和信息记录需要的变量。

3.4.2 源文件

3.4.2.1 构造函数

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
```

```
    canvas = new MattingCanvas(this);
    scrollArea = new QScrollArea(this);
    scrollArea->setWidget(canvas);
    ui->gridLayout->addWidget(scrollArea, 0, 0, 1, 1);
}
```

初始化界面。

创建自定义的抠图画布组件和滚动区域组件，并将画布组件添加到滚动区域中，以展示大图。

最后，将滚动区域组件添加到界面的网格布局中。

3.4.2.2 析构函数

```
MainWindow::~MainWindow()
{
    delete ui;
}
```

释放界面。

3.4.2.3 打开图片槽函数

```
void MainWindow::on_actionOpen_triggered()
{
    filename = QFileDialog::getOpenFileName(this, tr("Open Image"), QString(), tr("Image Files (*.png *.jpg *.jpeg *.bmp)"));
    pmat.open(filename);
    canvas->setImage(pmat.getImageMat());
}
```

弹出打开文件对话框，用户选择图片文件路径。

调用泊松抠图类实例 pmat 成员函数 open，打开该图片。

抠图画布设置通过抠图类获得的图像。

3.4.2.4 绘制未知区域槽函数

```
void MainWindow::on_actionDraw_Unknown_triggered(bool checked)
{
    ui->actionMark_Foreground->setChecked(false);
    canvas->setFilling(false);
    canvas->clearFilling();
    canvas->setDrawing(checked);
}
```

1. 标记前景区域的按钮选中状态取消；
2. 画布填充状态取消；
3. 画布清空前景填充；
4. 设置画布绘制状态。

3.4.2.5 填充前景区域槽函数

```
void MainWindow::on_actionMark_Foreground_triggered(bool checked)
```

```
{
    ui->actionDraw_Unknown->setChecked(false);
    canvas->setDrawing(false);
    canvas->setFilling(checked);
}
```

1. 绘制未知区域的按钮选中状态取消；
2. 画布绘制状态取消；
3. 设置画布填充状态。

3.4.2.6 清空所有标记槽函数

```
void MainWindow::on_actionClear_triggered()
{
    ui->actionMark_Foreground->setChecked(false);
    ui->actionDraw_Unknown->setChecked(false);
    canvas->setDrawing(false);
    canvas->setFilling(false);
    canvas->clearAll();
}
```

1. 绘制前景和未知区域的按钮选中状态取消；
2. 画布绘制状态取消；
3. 画布填充状态取消；
4. 画布清空所有标记。

3.4.2.7 退出槽函数

```
void MainWindow::on_actionExit_triggered()
{
    close();
}
```

关闭窗口。

3.4.2.8 抠图槽函数

```
void MainWindow::on_actionMatte_triggered()
{
    cv::Mat foreground, alpha;
    pmat.matting(canvas->getTrimapMat(), foreground, alpha);
    cv::imshow("trimap", canvas->getTrimapMat());
    cv::imshow("fore", foreground);
    cv::imshow("alpha", alpha);
}
```

1. 局部变量为抠图结果前景、透明通道矩阵；
2. 调用泊松抠图类实例 pmat 成员函数 matting 开始扣图得到返回的 foreground 和 alpha；
3. OpenCV 显示抠图蒙版、结果前景图和透明通道图。

3.4.2.9 清空前景标记槽函数

```
void MainWindow::on_actionClear_Mark_triggered()
{
    canvas->clearFilling();
}
```

画布清空填充的前景标记。

3.4.2.10 画笔大小增减槽函数

```
void MainWindow::on_action_Add_Pen_Size_triggered()
{
    canvas->setPenSize(canvas->getPenSize() + 5);
}

void MainWindow::on_action_Sub_Pen_Size_triggered()
{
    canvas->setPenSize(canvas->getPenSize() - 5);
}
```

调用画布成员函数获取并设置画笔大小，增减以 5 个单位为步长。

3.4.2.11 关于我槽函数

```
void MainWindow::on_actionAbout_triggered()
{
    QMessageBox::about(this, tr("About me"), tr("This is Poisson Matting Application for my final work of Digital Image Processing Course.\n\nHongxu Xu\n201411212027\n\nComputer Science Major\nCollege of Information Science and Technology\nBeijing Normal University"));
}
```

显示关于我的信息。

3.5 抠图画布程序：MATTINGCANVAS.H/CPP

3.5.1 头文件

```
class MattingCanvas : public QWidget
{
    Q_OBJECT
public:
    static QImage cvMatToQImage(const cv::Mat &inMat);
    static cv::Mat QImageToCvMat(const QImage &inImage, bool inCloneImageData = true);

    explicit MattingCanvas(QWidget *parent = 0);
    QSize sizeHint() const { return img.size(); }
    QSize minimumSizeHint() const { return img.size(); }

    cv::Mat getTrimapMat() const;
    void setImage(cv::Mat &imgMat);
    void setDrawing(bool drawing);
    void setFilling(bool filling);
```

```
    void clearFilling();
    void clearAll();
    void setPenSize(int s);
    int getPenSize() const { return penSize; }
protected:
    cv::Mat *imgMat = nullptr;
    cv::Mat fillMat;
    QImage img;
    QImage trimap;
    QImage fillMap;
    QPen pen = QPen(QColor(128, 128, 128));
    bool isDrawing = false;
    bool isFilling = false;
    bool isPressed = false;
    bool refill = true;
    int x, y;
    int mouseX, mouseY;
    int penSize = 10;

    // QWidget interface
protected:
    void paintEvent(QPaintEvent *event);
    void mousePressEvent(QMouseEvent *event);
    void mouseReleaseEvent(QMouseEvent *event);
    void mouseMoveEvent(QMouseEvent *event);
};
```

两个静态公开函数，用于在 OpenCV 的矩阵和 Qt 的 QImage 之间进行转换。

sizeHint 和 minimumSizeHint 用于提供自定义组件的大小，此处直接返回图片大小。

其他公开成员函数均为获取或设置相关参数的函数，具体功能见下文的源文件说明。

保护成员变量大多为所需的类内变量。

保护成员函数为覆盖实现的基类事件，分别用于处理以下事件：

- 重绘
- 鼠标按下
- 鼠标释放
- 鼠标移动

3.5.2 源文件

3.5.2.1 cv::Mat 和 QImage 转换函数

```
QImage MattingCanvas::cvMatToQImage(const cv::Mat &inMat)
{
    switch ( inMat.type() )
    {
```

```
    // 8-bit, 4 channel
    case CV_8UC4:
    {
        QImage image( inMat.data,
                      inMat.cols, inMat.rows,
                      static_cast<int>(inMat.step),
                      QImage::Format_ARGB32 );

        return image;
    }
    // 8-bit, 3 channel
    case CV_8UC3:
    {
        QImage image( inMat.data,
                      inMat.cols, inMat.rows,
                      static_cast<int>(inMat.step),
                      QImage::Format_RGB888 );

        return image.rgbSwapped();
    }
    // 8-bit, 1 channel
    case CV_8UC1:
    {
        static QVector<QRgb>  sColorTable( 256 );

        // only create our color table the first time
        if ( sColorTable.isEmpty() )
        {
            for ( int i = 0; i < 256; ++i )
            {
                sColorTable[i] = qRgb( i, i, i );
            }
        }

        QImage image( inMat.data,
                      inMat.cols, inMat.rows,
                      static_cast<int>(inMat.step),
                      QImage::Format_Indexed8 );

        image.setColorTable( sColorTable );

        return image;
    }
    default:
        qWarning() << "cvMatToQImage() - cv::Mat image type not handled in switch:" << inMat.type();
        break;
    }
```

```
    return QImage();
}

cv::Mat MattingCanvas::QImageToCvMat(const QImage &inImage, bool inCloneImageData )
{
    switch ( inImage.format() )
    {
    // 8-bit, 4 channel
    case QImage::Format_ARGB32:
    case QImage::Format_ARGB32_Premultiplied:
    {
        cv::Mat  mat( inImage.height(), inImage.width(),
                      CV_8UC4,
                      const_cast<uchar*>(inImage.bits()),
                      static_cast<size_t>(inImage.bytesPerLine())
                      );

        return (inCloneImageData ? mat.clone() : mat);
    }

    // 8-bit, 3 channel
    case QImage::Format_RGB32:
    case QImage::Format_RGB888:
    {
        if ( !inCloneImageData )
        {
            qWarning() << "ASM::QImageToCvMat() - Conversion requires cloning because we use a temporary
QImage";
        }

        QImage   swapped;

        if ( inImage.format() == QImage::Format_RGB32 )
            swapped = inImage.convertToFormat( QImage::Format_RGB888 );

        swapped = inImage.rgbSwapped();

        return cv::Mat( swapped.height(), swapped.width(),
                        CV_8UC3,
                        const_cast<uchar*>(swapped.bits()),
                        static_cast<size_t>(swapped.bytesPerLine())
                        ).clone();
    }
    // 8-bit, 1 channel
    case QImage::Format_Indexed8:
    {
        cv::Mat  mat( inImage.height(), inImage.width(),
```

```
                    CV_8UC1,
                    const_cast<uchar*>(inImage.bits()),
                    static_cast<size_t>(inImage.bytesPerLine())
                    );

        return (inCloneImageData ? mat.clone() : mat);
    }
    default:
        qWarning() << "QImageToCvMat() - QImage format not handled in switch:" << inImage.format();
        break;
    }

    return cv::Mat();
}
```

这属于一个工具函数，针对不同颜色属性图片数据分别转换。支持的颜色属性并不全，仅实现了常见的灰度图片（8 位单通道）和 8 位三通道（RGB、BGR）图片等的转换。

3.5.2.2 构造函数

```
MattingCanvas::MattingCanvas(QWidget *parent) : QWidget(parent)
{
    pen.setWidth(penSize);
    setMouseTracking(true);
}
```

画笔初始化宽度，设置该组件追踪鼠标移动（否则鼠标移动事件只在鼠标按下后触发）。

3.5.2.3 返回蒙版图片

```
cv::Mat MattingCanvas::getTrimapMat() const
{
    cv::Mat temp;
    cv::cvtColor(fillMat, temp, CV_RGB2GRAY);
    return temp;
}
```

蒙版图片转换为灰度图后返回。

3.5.2.4 设置图片

```
void MattingCanvas::setImage(cv::Mat &imgMat)
{
    this->imgMat = &imgMat;
    this->img = cvMatToQImage(imgMat);
    this->setFixedSize(img.size());
    trimap = QImage(img.size(), QImage::Format_ARGB32);
    trimap.fill(Qt::transparent);
    refill = true;
    repaint();
}
```

设置 cv::Mat 类型图片，并转换为 QImage 用于 Qt 组件重绘事件中绘制。

3.5.2.5 设置绘画状态

```
void MattingCanvas::setDrawing(bool drawing)
{
    isDrawing = drawing;
}

void MattingCanvas::setFilling(bool filling)
{
    isFilling = filling;
}
```

3.5.2.6 清空标记

```
void MattingCanvas::clearFilling()
{
    fillMap = QImage();
    refill = true;
    repaint();
}

void MattingCanvas::clearAll()
{
    trimap.fill(Qt::transparent);
    clearFilling();
}
```

3.5.2.7 设置画笔大小

```
void MattingCanvas::setPenSize(int s)
{
    pen.setWidth(s);
    penSize = s;
}
```

3.5.2.8 重绘事件

```
void MattingCanvas::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    painter.drawImage(QPoint(0, 0), img);
    painter.drawImage(QPoint(0, 0), trimap);
    painter.setCompositionMode(QPainter::CompositionMode_SoftLight);
    painter.drawImage(QPoint(0, 0), fillMap);
    painter.setCompositionMode(QPainter::CompositionMode_Overlay);
    painter.drawRect(QRect(mouseX - penSize / 2, mouseY - penSize / 2, penSize, penSize));
    painter.end();
}
```

画原图，画用户绘制的未知区域（trimap），画用户填充的前景区域（fillMap），画鼠标位置。

3.5.2.9 鼠标事件

```
void MattingCanvas::mousePressEvent(QMouseEvent *event)
{
    isPressed = true;
    x = event->x();
    y = event->y();
}
```

鼠标按下：设置按下状态，记录临时坐标。

```
void MattingCanvas::mouseReleaseEvent(QMouseEvent *event)
{
    isPressed = false;
    x = event->x();
    y = event->y();
    if (isFilling) {
        if (refill) {
            refill = false;
            fillMat = QImageToCvMat(trimap);
            cv::cvtColor(fillMat, fillMat, CV_RGBA2RGB);
        }
        cv::floodFill(fillMat, cv::Point(x, y), cv::Scalar(255, 255, 255));
        fillMap = cvMatToQImage(fillMat);
        repaint();
    }
}
```

鼠标释放：取消按下状态，记录临时坐标。

如果是填充状态，进行填充：如果需要重新填充，则清空填充图像矩阵；然后对鼠标释放坐标处进行填充（cv::floodFill）。

```
void MattingCanvas::mouseMoveEvent(QMouseEvent *event)
{
    mouseX = event->x();
    mouseY = event->y();
    if (isDrawing && isPressed) {
        if (abs(x - event->x()) + abs(y - event->y()) > 5) {
            QPainter painter(&trimap);
            painter.setPen(pen);
            painter.drawLine(QPoint(x, y), QPoint(event->x(), event->y()));
            painter.end();
            refill = true;
            x = event->x();
            y = event->y();
        }
    }
    repaint();
}
```

鼠标移动：记录鼠标坐标。

如果鼠标按下且处于绘制未知区域状态，则绘制未知区域：设置画笔，画线（从鼠标按下位置，到当前位置的直线段），设置需要重新填充，更新临时坐标。

3.6 泊松抠图程序：POISSONMATTING.H/CPP

3.6.1 头文件

```
class PoissonMatting
{
public:
    PoissonMatting();
    void open(QString filename);
    cv::Mat &getImageMat();
    void matting(cv::InputArray _trimap, cv::OutputArray _foreground, cv::OutputArray _alpha);
protected:
    cv::Mat img;
    static std::vector<cv::Point> findBoundaryPixels(const cv::Mat_<uchar> &trimap, int a, int b);
    void _matting(cv::Mat _image, cv::Mat _trimap, cv::Mat &_foreground, cv::Mat &_alpha);
};
```

公开成员函数有 matting 抠图，是保护成员函数_matting 的包装。

其他函数参见下文源程序说明。

3.6.2 源文件

3.6.2.1 一些帮助函数

```
template <class T>
T sqr(T t) {
    return t * t;
}
```

任意类型的平方。

```
double dist_sqr(cv::Point p1, cv::Point p2) {
    return sqr(p1.x - p2.x) + sqr(p1.y - p2.y);
}
```

两点距离的平方。

```
int color_dis(cv::Vec3b p1, cv::Vec3b p2) {
    int t1 = fmax(fmax(p1[0], p1[1]), p1[2]);
    int t2 = fmax(fmax(p2[0], p2[1]), p2[2]);
    return t1 - t2;
}
```

颜色亮度差。

```
template <class T>
int inX(const T &image, int x) {
    if (x < 0) x = 0;
```

```
    if (x >= image.cols) x = image.cols - 1;
    return x;
}
```

边界处理 X 坐标。

```
template <class T>
int inY(const T &image, int y) {
    if (y < 0) y = 0;
    if (y >= image.rows) y = image.rows - 1;
    return y;
}
```

边界处理 Y 坐标。

```
double intensity(cv::Vec3b v) {
    return fmax(fmax(v[0], v[1]), v[2]);
}
```

计算颜色亮度。

3.6.2.2 打开图片、获取图片

```
void PoissonMatting::open(QString filename)
{
    img = cv::imread(std::string(filename.toLocal8Bit()));
}

cv::Mat &PoissonMatting::getImageMat()
{
    return img;
}
```

OpenCV 的 imread 必须读取 GBK 编码的中文路径，故使用 toLocal8Bit。

3.6.2.3 抠图包装函数

```
void PoissonMatting::matting(cv::InputArray _trimap, cv::OutputArray _foreground, cv::OutputArray _alpha)
{
    cv::Mat image = img;
    cv::Mat trimap = _trimap.getMat();

    if (image.empty())
        CV_Error(CV_StsBadArg, "Image is empty");
    if (image.type() != CV_8UC3)
        CV_Error(CV_StsBadArg, "Image must have CV_8UC3 type");

    if (trimap.empty())
        CV_Error(CV_StsBadArg, "Trimap is empty");
    if (trimap.type() != CV_8UC1)
        CV_Error(CV_StsBadArg, "Trimap must have CV_8UC1 type");

    if (image.size() != trimap.size())
```

```
        CV_Error(CV_StsBadArg, "Trimap and image must have the same size.");

    cv::Mat &foreground = _foreground.getMatRef();
    cv::Mat &alpha = _alpha.getMatRef();

    _matting(image, trimap, foreground, alpha);
}
```

对提供的参数进行合法性判断，不合法则报错。

调用内部的保护成员函数，进行抠图。

3.6.2.4 获取边界像素

```
std::vector<cv::Point> PoissonMatting::findBoundaryPixels(const cv::Mat_<uchar> &trimap, int a, int b)
{
    std::vector<cv::Point> result;

    for (int x = 1; x < trimap.cols - 1; ++x) {
        for (int y = 1; y < trimap.rows - 1; ++y) {
            if (trimap(y, x) == a) {
                if (trimap(y - 1, x) == b ||
                        trimap(y + 1, x) == b ||
                        trimap(y, x - 1) == b ||
                        trimap(y, x + 1) == b) {
                    result.push_back(cv::Point(x, y));
                }
            }
        }
    }

    return result;
}
```

遍历蒙版图，寻找像素值为 a，相邻像素值为 b 的边界像素。

3.6.2.5 抠图函数

```
void PoissonMatting::_matting(cv::Mat _image, cv::Mat _trimap, cv::Mat &_foreground, cv::Mat &_alpha)
{
    const cv::Mat_<cv::Vec3b> &image = static_cast<const cv::Mat_<cv::Vec3b> &>(_image);
    cv::Mat_<uchar> &trimap = static_cast<cv::Mat_<uchar> &>(_trimap);

    _foreground.create(image.size(), CV_8UC3);
    _alpha.create(image.size(), CV_8UC1);

    cv::Mat_<cv::Vec3b> &foreground = static_cast<cv::Mat_<cv::Vec3b>&>(_foreground);
    cv::Mat_<uchar> &alpha = static_cast<cv::Mat_<uchar>&>(_alpha);

    cv::Mat_<double> FminusB = cv::Mat_<double>::zeros(trimap.rows, trimap.cols);
```

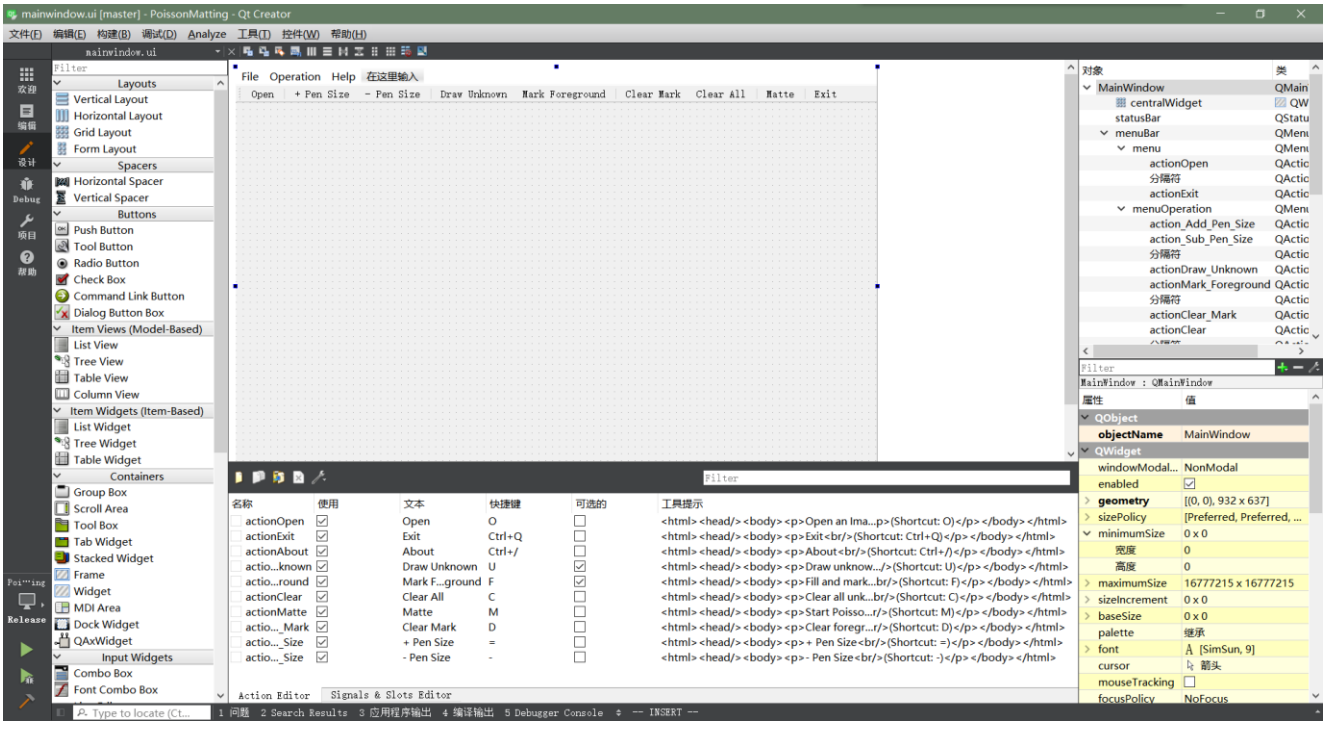
```
    for (int times = 0; times < 5; ++times) {

        // Output the Progress
        qDebug() << times;

        std::vector<cv::Point> foregroundBoundary = findBoundaryPixels(trimap, 255, 128);
        std::vector<cv::Point> backgroundBoundary = findBoundaryPixels(trimap, 0, 128);

        cv::Mat_<uchar> trimap_blur;

        // Smooth Trimap by gaussian filter to denoise
        cv::GaussianBlur(trimap, trimap_blur, cv::Size(9, 9), 0);

        // Build the F-B Map
        for (int x = 0; x < trimap.cols; ++x) {
            for (int y = 0; y < trimap.rows; ++y) {
                cv::Point current;
                current.x = x;
                current.y = y;
                if (trimap_blur(y, x) == 255) {
                    FminusB(y, x) = color_dis(image(y, x), cv::Vec3b(0, 0, 0));
                } else if (trimap_blur(y, x) == 0) {
                    FminusB(y, x) = color_dis(cv::Vec3b(0, 0, 0), image(y, x));
                } else {
                    // is in Unknown Area
                    // Find Nearest Foreground and Background Point
                    cv::Point nearestForegroundPoint, nearestBackgroundPoint;
                    double nearestForegroundDistance = 1e9, nearestBackgroundDistance = 1e9;
                    for(cv::Point &p : foregroundBoundary) {
                        double t = dist_sqr(p, current);
                        if (t < nearestForegroundDistance) {
                            nearestForegroundDistance = t;
                            nearestForegroundPoint = p;
                        }
                    }
                    for(cv::Point &p : backgroundBoundary) {
                        double t = dist_sqr(p, current);
                        if (t < nearestBackgroundDistance) {
                            nearestBackgroundDistance = t;
                            nearestBackgroundPoint = p;
                        }
                    }
                    // Calculate F - B
                    FminusB(y, x) = color_dis(image(nearestForegroundPoint.y, nearestForegroundPoint.x),
                                              image(nearestBackgroundPoint.y, nearestBackgroundPoint.x));
                    if (FminusB(y, x) == 0)
                        FminusB(y, x) = 1e-9;
```

```
            }
        }
    }

    // Smooth (F - B) image by Gaussian filter
    cv::GaussianBlur(FminusB, FminusB, cv::Size(9, 9), 0);

    // Solve the Poisson Equation By The Gauss-Seidel Method (Iterative Method)
    for (int times2 = 0; times2 < 300; ++times2) {
        for (int x = 0; x < trimap.cols; ++x) {
            for (int y = 0; y < trimap.rows; ++y) {
                if (trimap(y, x) == 128) {
                    // is in Unknown Area
#define I(x, y) (intensity(image(inY(image, y), inX(image, x))))
#define FmB(y, x) (FminusB(inY(FminusB, y), inX(FminusB, x)))
                    // Calculate the divergence
                    double dvgX = ((I(x + 1, y) + I(x - 1, y) - 2 * I(x, y)) * FmB(y, x)

                            - (I(x + 1, y) - I(x, y)) * (FmB(y, x + 1) - FmB(y, x)))
                            / (FmB(y, x) * FmB(y, x));
                    double dvgY = ((I(x, y + 1) + I(x, y - 1) - 2 * I(x, y)) * FmB(y, x)
                            - (I(x, y + 1) - I(x, y)) * (FmB(y + 1, x) - FmB(y, x)))
                            / (FmB(y, x) * FmB(y, x));
                    double dvg = dvgX + dvgY;
#undef FmB
#undef I
                    // Calculate the New Alpha (Gauss-Seidel Method)
                    double newAlpha = (((double)alpha(y, x + 1)
                                    + alpha(y, x - 1)
                                    + alpha(y + 1, x)
                                    + alpha(y - 1, x)
                                    - dvg * 255.0) / 4.0);
                    // Update the Trimap
                    if (newAlpha > 253) {
                        // fore
                        trimap(y, x) = 255;
                    } else if (newAlpha < 3) {
                        // back
                        trimap(y, x) = 0;
                    }
                    // Avoid overflow
                    if (newAlpha < 0) {
                        newAlpha = 0;
                    }
                    if (newAlpha > 255) {
                        newAlpha = 255;
                    }
```

```
                    // Assign new alpha
                    alpha(y, x) = newAlpha;
                } else if (trimap(y, x) == 255) {
                    // is Foreground
                    alpha(y, x) = 255;
                } else if (trimap(y, x) == 0) {
                    // is Background
                    alpha(y, x) = 0;
                }
            }
        }
    }

    // Generate Foreground Image (Red Background)
    for (int x = 0; x < alpha.cols; ++x) {
        for (int y = 0; y < alpha.rows; ++y) {
            foreground(y, x) = ((double) alpha(y, x) / 255) * image(y, x) + ((255.0 - alpha(y, x)) / 255 *
cv::Vec3b(0, 0, 255));
        }
    }
}
```

泊松抠图，代码含注释。涉及算法，具体不述。

3.7 主界面：MAINWINDOW.UI

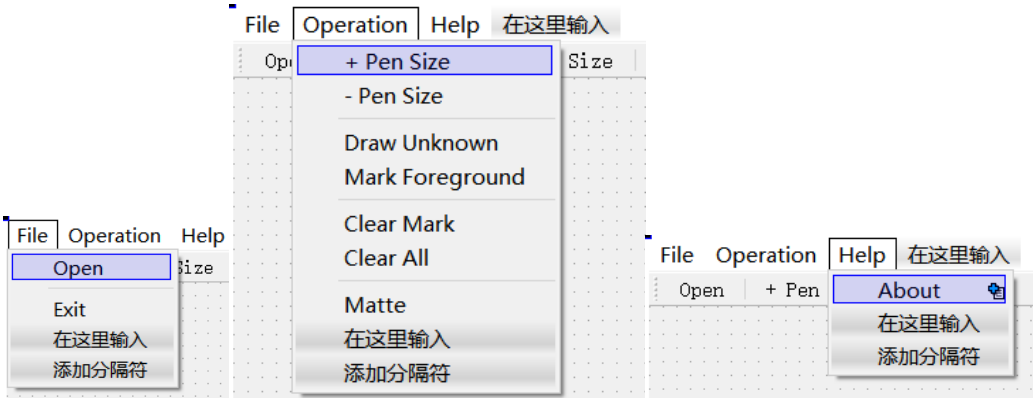


3.7.1 命令

名称	使用	文本	快捷键	可选的
actionOpen	☑	Open	O	☐
actionExit	☑	Exit	Ctrl+Q	☐
actionAbout	☑	About	Ctrl+/	☐
actionDraw_Unknown	☑	Draw Unknown	U	☑
actionMark_Foreground	☑	Mark F...ground	F	☑
actionClear	☑	Clear All	C	☐
actionMatte	☑	Matte	M	☐
actionClear_Mark	☑	Clear Mark	D	☐
action_Add_Pen_Size	☑	+ Pen Size	=	☐
action_Sub_Pen_Size	☑	- Pen Size	-	☐

3.7.2 工具条

3.7.3 菜单栏



3.7.4 说明

抠图画布在主窗口程序的构造函数中，通过代码创建并加入到主界面中。

3.8 算法流程

3.8.1 透明通道抠图

泊松抠图属于透明通道抠图：

$$\mathrm{I} = \alpha\mathrm{F} + (1-\alpha)\mathrm{B}$$

其中I是给定图像的，F、B为给定图像的前景和背景，α是透明度。即一幅图是由我们关心的前景图像，和无关的背景图像由一定的透明分布混合而成。由于光线的反射、折射、散射等，透明通道抠图结果具有更好的视觉效果。

计算图像梯度：

$$\nabla\mathrm{I} = (\mathrm{F}-\mathrm{B})\nabla\alpha + \alpha\nabla\mathrm{F} + (1-\alpha)\nabla\mathrm{B}$$

在较滑图像中，前景、背景的变化小，因此后两项可以忽略。变换可得：

公式 1

$$\nabla\alpha \approx \frac{1}{F-B}\nabla I$$

3.8.2 全局泊松抠图

3.8.2.1 原理

令：Ω_F为绝对前景区域，Ω_B为绝对背景区域，Ω为未知区域。

对于图像的每个像素$\mathrm{p}=(\mathrm{x},\mathrm{y})$，$\mathrm{I_p}$为其亮度，$\mathrm{F_p}$和$\mathrm{B_p}$是其前景和背景亮度。

令：$\mathrm{N_p}$为像素p的四个相邻像素，$\partial\Omega=\{\mathrm{p}\in\Omega_F\cup\Omega_B|N_p\cap\Omega\neq\emptyset\}$为Ω的外边界。

为实现对未知区域Ω的抠图，需求解下列问题：

$$\alpha^* = arg\min_{\alpha}\iint_{p\in\Omega}\left\|\nabla\alpha_p-\frac{1}{F_p-B_p}\nabla I_p\right\|^2 dp$$

狄利克雷边界条件为：

$$\alpha|_{\partial\Omega}=\hat{\alpha}|_{\partial\Omega}=\begin{cases}1, & \mathrm{p}\in\Omega_\mathrm{F}\\ 0, & \mathrm{p}\in\Omega_\mathrm{B}\end{cases}$$

即，令利用公式 1 计算的透明通道与其估计值的最小平方误差最小，取此时的α。

该问题可转换为具有相同边界条件的泊松方程求解的问题：

$$\Delta\alpha=\mathrm{div}\left(\frac{\nabla I}{F-B}\right)$$

其中，$\Delta=\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)$和div分别是拉普拉斯算子和散度算子。

求解泊松方程的解已经有很多研究出的方法，应用程序中采用 Gauss-Seidel 超松弛迭代法。另外，彩色图像的$(\mathrm{F}-\mathrm{B})$和I均以灰度值来计算。

3.8.2.2 流程

全局泊松抠图是一个迭代优化过程：

1. $(\mathrm{F}-\mathrm{B})$初始化

   对图像的每个像素$\mathrm{p}\in\Omega$，其$\mathrm{F_p}$和$\mathrm{B_p}$分别利用Ω_F和Ω_B中距离最近的绝对前景像素和绝对背景像素来估计。然后，建立$(\mathrm{F}-\mathrm{B})$图像，并利用高斯滤波器降噪。

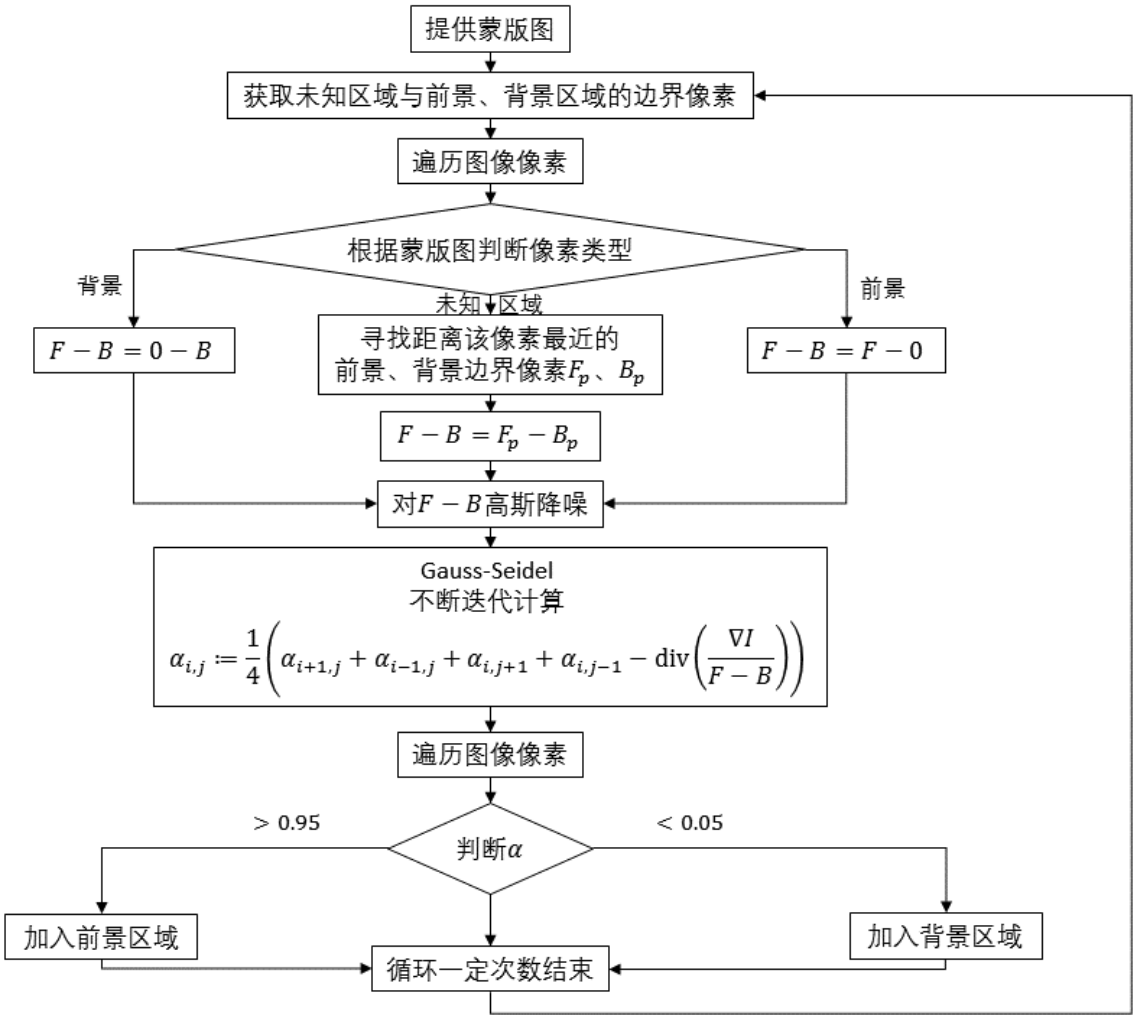
2. α重建

   利用$(\mathrm{F}-\mathrm{B})$和∇I求解泊松方程以重建α。

3. F, B优化再提取

   根据重建的α，将$\alpha>0.95$的像素点加入绝对前景区域，$\alpha<0.05$的像素点加入绝对背景区域。更新$(\mathrm{F}-\mathrm{B})$并应用高斯滤波。

3.8.2.3 算法流程图


提供蒙版图
获取未知区域与前景、背景区域的边界像素
遍历图像像素
根据蒙版图判断像素类型
背景
未知区域
前景
$F-B=0-B$
寻找距离该像素最近的
前景、背景边界像素F_p、B_p
$F-B=F-0$
$F-B=F_p-B_p$
对$F-B$高斯降噪
Gauss-Seidel
不断迭代计算
$\alpha_{i,j}:=\frac{1}{4}\left(\alpha_{i+1,j}+\alpha_{i-1,j}+\alpha_{i,j+1}+\alpha_{i,j-1}-\mathrm{div}\left(\frac{\nabla I}{F-B}\right)\right)$
遍历图像像素
>0.95
判断α
<0.05
加入前景区域
加入背景区域
循环一定次数结束