图形学第一次作业报告

计45 孟垂正 2013010952

一. 作业内容

1. Bresenham算法绘制线段
2. 区域加权采样法反走样

二. 实现方法

1. Bresenham算法绘制线段

Bresenham算法绘制原理是通过计算每个像素点的误差项符号来决定下一个像素取右边点还是右上点.
以线段$|dx| \geq |dy|$的情况为例, 记e为当前位置的y与所在像素方格中点纵坐标y_{mid}之差. 显然e初始值为-0.5. 记$k = \frac{dy}{dx}$, x在自左至右的过程中增加1, 则e相应增加k. $k < 0$表明右侧像素距离直线更近, $k \geq 0$表明右上方像素距离直线更近, 此时应将e相应减1.
为了转化为整数运算以加快速度, 将e的值乘以$2dx$倍. 此时e的初始值设置为:

```
e = -dx;
```

而e每次的增量变为$2dy$, 选择右上方像素后e的减少量为$-2dx$.

```
for (int i = 0; i <= dx; ++i)
{
    draw_pixel(x++, y, bgr);
    e += 2 * absdy; // Use abs in order to keep with the situation where dy < 0
    if (e >= 0)
    {
        y += inc;
        e -= (2 * dx);
    }
```

```
}
```

2. 区域加权采样法反走样

加权区域采样方法使相交区域对亮度的贡献值依赖于该区域与亮度中心之间的距离. 滤波器函数取高斯滤波器.

$$\omega(x,y)=\frac{1}{\sqrt{2\pi\sigma}}e^{\frac{x^2+y^2}{2\sigma^2}}$$

为了减少计算量, 采用加权表方法进行离散计算. 加权表可以用:

$$\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$$

即将每个像素分为3×3个子像素, 使用Bresenham算法计算线段经过了哪些像素, 根据加权表计算这些像素对原像素亮度的贡献并着色.
实现算法时, 采用的加权表的规模为3×3, 问题转化为在一张横纵像素数均为待显示画面像素数3倍的画布上进行直线绘制并在原图上着色的问题. 考虑到仅仅绘制一像素宽的线段将使得图像变暗, 故实际绘制了并行的3条线段, 总宽度为3像素.
为了避免对全图的无谓扫描, 绘制过程中每绘制完一个大像素中的3个小像素, 即用加权表计算大像素的颜色并进行着色. 因此使用了一个3×9的矩阵作为buffer(极端情况下宽度3像素的线段会涉及到3个大像素).

```
Mat buffer = Mat::zeros(anti_para, anti_para * 3, CV_8UC3); // 3 means upper, middle
and bottom
```

绘制时根据Bresenham画线算法在buffer中进行标记并记录buffer对应的大像素位置.

```
if (x % anti_para == 0) { // Get the position of meta pixel
    act_x = x / anti_para;
    act_y = y / anti_para;
}
```

到达buffer边界后计算整个buffer颜色RGB值的加权平均, 根据此结果为大像素着色.

```
if (x % anti_para == anti_para - 1) {
    for (int k = 0; k < 2; ++k) {
        int startx = 0;
        int starty = k * anti_para;
        if (iflegal(act_x, act_y + k)) {
            int bsum = 0, gsum = 0, rsum = 0;
```

```
            for (int pick_x = 0; pick_x < anti_para; ++pick_x) {
                for (int pick_y = 0; pick_y < anti_para; ++pick_y) {
                    bsum += (weight_table[pick_x * anti_para + pick_y] *
buffer.at<Vec3b>(pick_x, pick_y + starty)[0]);
                    gsum += (weight_table[pick_x * anti_para + pick_y] *
buffer.at<Vec3b>(pick_x, pick_y + starty)[1]);
                    rsum += (weight_table[pick_x * anti_para + pick_y] *
buffer.at<Vec3b>(pick_x, pick_y + starty)[2]);
                }
            }
            img.at<Vec3b>(act_x, act_y + k)[0] = (int)((double)bsum / weight_sum);
            img.at<Vec3b>(act_x, act_y + k)[1] = (int)((double)gsum / weight_sum);
            img.at<Vec3b>(act_x, act_y + k)[2] = (int)((double)rsum / weight_sum);
        }
    }
    buffer = Mat::zeros(anti_para, anti_para * 3, CV_8UC3);
}
```

与扫描全图相比, 此方法存在一定误差, 但计算速度大大加快, 可以做到实时绘制.

三. Demo操作说明

- X0, Y0, X1, Y1用于设置起点终点坐标, 点击Draw Line按钮绘制.
- 也可以使用鼠标直接在黑色区域内拖动绘制.
- RGB设置线段颜色值, 默认为白色.
- Clear清除区域中所有线段.
- 勾选Antialias打开反走样功能.

四. 补充

- 本程序基于Qt 5.5.1 (GCC 4.9.1 20140922 (Red Hat 4.9.1-10), 64 bit), 使用了Qt的窗口控件等GUI, 但绘图部分使用Bresenham算法和Opencv完成, 未使用Qt图形库.
- 实验过程中发现Opencv的highgui和Qt的GUI存在不兼容情况, 考虑使用Qt显示Opencv得到的图片. 而Opencv的Mat需要经过转换才能使用Qt的Label显示, 具体方法如下:

```
QImage Opencv_Drawer::cvmat2qtimage()
{
    QImage qtimage(img.data, img.cols, img.rows, img.step, QImage::Format_RGB888);
    return qtimage.rgbSwapped();
}
...
```

```
void MainWindow::draw_qpix()
{
    QImage img = opencvdrawer.cvmat2qtimage();
    ui->label->setPixmap(QPixmap::fromImage(img));
    ui->label->resize(img.width(), img.height());
}
```