Canny Edge Detection

STONE

STEP 1

利用高斯滤波器完成 src 图像的去噪。

使用 opencv 中的 GaussianBlur 函数完成。

STEP 2

利用 Sobel 算子计算梯度的幅值(magnitude)和方向(direction)。

使用 Opencv 中的 Sobel 函数完成两个方向梯度图像生成(magx, magy)，之后根据公式

|G| = |Gx| + |Gy|计算 magnitude，根据公式 d = Gy / Gx 计算 direction。

这部分我将所有操作封装在 calSlope 函数中，获得 magnitude 与 direction，代码如下：

```
void calSlope(Mat src, Mat& direction, Mat& magnitude){
    Mat magX = Mat(src.rows, src.cols, CV_32F);
    Mat magY = Mat(src.rows, src.cols, CV_32F);
    Sobel(src, magX, CV_32F, 1, 0, 3);
    Sobel(src, magY, CV_32F, 0, 1, 3);

    direction = Mat(src.rows, src.cols, CV_32F);
    divide(magY, magX, direction);

    Mat mag2X = Mat(src.rows, src.cols, CV_64F);
    Mat mag2Y = Mat(src.rows, src.cols, CV_64F);
    multiply(magX, magX, mag2X);
    multiply(magY, magY, mag2Y);
    sqrt(mag2X + mag2Y, magnitude);
}
```

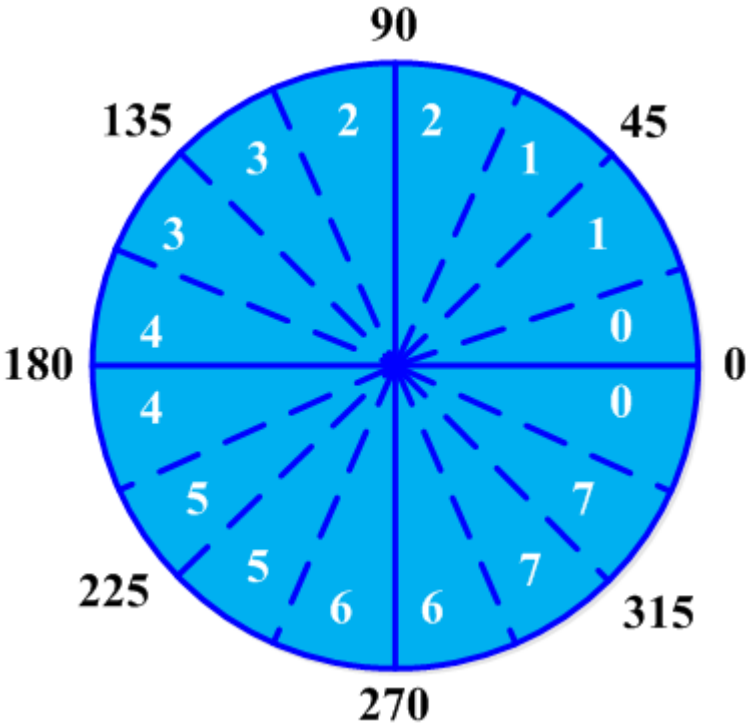
STEP 3

使用非极大值抑制算法（NMS）将局部梯度非最大的值去除来确定边缘。

此处算法未使用 Opencv 中的函数。

首先从对从上一步获取的 direction 进行分析，获取其角度值。

参考下图进行说明：



对获取的角度值根据算法分成 0&4, 1&5,2&6,3&7 四个区域，对应下示意图进行分析

$ & #
* O *
# & $

例如对于 0&4 这一个区域中的值，中心点 O 对应的梯度角方向直线经过的是*的位置，那么 O 点像素与*对应的两个位置点像素比较来获得结果，其他与之同理。

在区域的每一点上，将中心点像素与梯度角方向上的两个点的像素进行比较，如果比中心点像素比任意一个位置的像素小，那么说明该点非极大值，将其抑制（中心点像素设为 0）。

我将这一部分的实现封装在 nonMaximumSuppression 函数中，获得的结果保存在 magnitude 中，代码如下：

```
void nonMaximumSuppression(Mat &magnitudeImage, Mat &directionImage) {
    Mat checkImage = Mat(magnitudeImage.rows, magnitudeImage.cols, CV_8U);

    MatIterator_<float>itMag = magnitudeImage.begin<float>();
    MatIterator_<float>itEnd = magnitudeImage.end<float>();
    MatIterator_<float>itDirection = directionImage.begin<float>();
    MatIterator_<unsigned char>itRet = checkImage.begin<unsigned char>();

    for (; itMag != itEnd; ++itDirection, ++itRet, ++itMag) {
        const Point pos = itRet.pos();
        float currentDirection = atan(*itDirection) * (180 / PI);
        *itDirection = currentDirection;
        if ((currentDirection > 22.5 && currentDirection <= 67.5) ||
            (currentDirection > -157.5 && currentDirection <= -112.5)) {
            if (pos.y > 0 && pos.x > 0 && *itMag <= magnitudeImage.at<float>(pos.y - 1, pos.x - 1)) {
                magnitudeImage.at<float>(pos.y, pos.x) = 0;
            }
            if (pos.y < magnitudeImage.rows - 1 && pos.x < magnitudeImage.cols - 1 && *itMag <= magnitudeImage.at<float>(pos.y + 1, pos.x + 1)) {
                magnitudeImage.at<float>(pos.y, pos.x) = 0;
            }
        }
        else if ((currentDirection > 67.5 && currentDirection <= 112.5) ||
                 (currentDirection > -112.5 && currentDirection <= -67.5)){
            if (pos.y > 0 && *itMag <= magnitudeImage.at<float>(pos.y - 1, pos.x)) {
                magnitudeImage.at<float>(pos.y, pos.x) = 0;
            }
            if (pos.y<magnitudeImage.rows - 1 && *itMag <= magnitudeImage.at<float>(pos.y + 1, pos.x)) {
                magnitudeImage.at<float>(pos.y, pos.x) = 0;
            }

        }
        else if ((currentDirection > 112.5 && currentDirection <= 157.5) ||
                 (currentDirection > -67.5 && currentDirection <= -22.5)){
            if (pos.y>0 && pos.x<magnitudeImage.cols - 1 && *itMag <= magnitudeImage.at<float>(pos.y - 1, pos.x + 1)) {
                magnitudeImage.at<float>(pos.y, pos.x) = 0;;
            }
            if (pos.y < magnitudeImage.rows - 1 && pos.x>0 && *itMag <= magnitudeImage.at<float>(pos.y + 1, pos.x - 1)) {
                magnitudeImage.at<float>(pos.y, pos.x) = 0;
            }
        }
        else {
            if (pos.x > 0 && *itMag <= magnitudeImage.at<float>(pos.y, pos.x - 1)) {
                magnitudeImage.at<float>(pos.y, pos.x) = 0;
            }
            if (pos.x < magnitudeImage.cols - 1 && *itMag <= magnitudeImage.at<float>(pos.y, pos.x + 1)) {
                magnitudeImage.at<float>(pos.y, pos.x) = 0;
            }
        }
    }
}
```

STEP 4

使用 hysteresis 双阈值算法获取边缘。

此处算法未使用 Opencv 中的函数。

先使用一个高阈值（UP）来进行检测，这样会获得间断的边缘，然后对于边缘点位置作为中心点进行八格临点位置中进行递归寻找，直到找到能连接闭合(某边缘点像素为 255)且该中心点像素高于低阈值（LOW）【UP≈2*LOW】时完成，这样对全图进行扫描完成边缘获取。

我将这一部分代码封装在 hysteresis 函数中，其中该函数会调用 findEdge 函数，该函数递归调用自身完成边缘连接，hysteresis 函数将生成的边缘保存在 magnitude 中，代码如下：

```
void findEdges(int x, int y, Mat magnitude, int up, int low, Mat &edges) {
    edges.at<float>(x, y) = 255;

    for (int i = -1; i < 2; i++) {
        for (int j = -1; j < 2; j++) {
            if ((i != 0) && (j != 0) && (x + i >= 0) && (y + j >= 0) && (x + i < magnitude.rows) && (y + j < magnitude.cols)){
                if ((magnitude.at<float>(x + i, y + j) > low) && (edges.at<float>(x + i, y + j) != 255)) {
                    findEdges(x + i, y + j, magnitude, up, low, edges);
                }
            }
        }
    }
}

void hysteresis(Mat magnitude, int up, int low, Mat& edgesImage) {
    int width = magnitude.rows;
    int height = magnitude.cols;

    edgesImage = Mat(magnitude.rows, magnitude.cols, magnitude.type());
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            if (magnitude.at<float>(x, y) >= up){
                findEdges(x, y, magnitude, up, low, edgesImage);
            }
        }
    }
}
```

Final

```
void cannyEdgeDetection(Mat &src, Mat &edgesImage, int upperThresh, int lowerThresh) {
    Mat image = src.clone();;
    //step1:使用高斯滤波器进行噪声去除
    GaussianBlur(src, image, Size(3, 3), 1.4);
    //step2:使用Sobel算子计算边缘梯度值和梯度方向
    Mat direction;
    Mat magnitude;
    calSlope(image, direction, magnitude);
    //step3:进行非最大梯度点值抑制
    nonMaximumSuppression(magnitude, direction);
    //step4:使用hysteresis双阈值算法进行边缘生成
    hysteresis(magnitude, upperThresh, lowerThresh, edgesImage);
    //其中step3,step4均为不借助opencv已有函数完成，具体实现算法描述详见文档

}
```

使用本算法进行边缘获取，原图与在 UP=90，LOW=45 的情况下生成的边缘图片如下：