中山大學
SUN YAT-SEN UNIVERSITY

《计算机视觉》

实验文档

（实验二）

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	16 软件工程（数字媒体）
学 生 姓 名 ：	黎汛言
学 号 ：	16340109
时 间 ：	2018 年 10 月 16 日

实验二 ：用CImg重写、封装给定的Canny代码，并测试

一. 实验目的

1. 学习如何利用Canny算子进行边缘检测。

二. 实验内容

1. 利用CImg库改写Code0，封装成C++类；
2. 新增函数，将相邻的边缘连成长的线条，删除长度小于20的边缘。

三. 实验环境

Windows 10 64位

四. 实验过程

1. 设计Canny类，封装Canny边缘检测的步骤

Canny边缘检测需要的成员变量：

```
class Canny {
    private:
        char *infilename = NULL;        /* Name of the input image                    */
        char *dirfilename = NULL;       /* Name of the output gradient direction image  */
        FILE *fpdir = NULL;             /* File to write the gradient image to.       */
        char outfilename[128];          /* Name of the output "edge" image            */
        char composedfname[128];        /* Name of the output "direction" image       */
        int w, h;                       /* The dimensions of the image.               */
        float *dir_radians = NULL;      /* Gradient direction image.                  */
        CImg<unsigned char> src,        /* The input image                            */
                            image,      /* The grey scale image                       */
                            edge,       /* The output edge image                      */
                            nms;        /* Points that are local maximal magnitude.   */
        CImg<short int> smoothedim,     /* The image after gaussian smoothing.        */
                        delta_x,        /* The first devivative image, x-direction.   */
                        delta_y,        /* The first derivative image, y-direction.   */
                        magnitude;      /* The magnitude of the gadient image.        */
        float sigma,                    /* Standard deviation of the gaussian kernel. */
              tlow,                     /* Fraction of the high threshold in hysteresis.*/
              thigh;                    /* High hysteresis threshold control. The actual
                                           threshold is the (100 * thigh) percentage
                                           point in the histogram of the magnitude of
                                           the gradient image that passes non-maximal
                                           suppression.                               */
```

将Canny边缘检测的步骤封装成Canny类的成员函数。其中，to_grey_scale()、optimize()、connect()是在原代码基础上增加的函数：

```
        void to_grey_scale();                       /* Convert the image to grey scale.                */
        void gaussian_smooth();                     /* Perform gaussian smoothing on the
                                                       image using the input standard deviation.       */
        void derrivative_x_y();                     /* Compute the first derivative in the
                                                       x and y directions.                             */
        void radian_direction();                    /* Compute the direction of the gradient.(optional) */
        void magnitude_x_y();                       /* Compute the magnitude of the gradient.          */
        void non_max_supp();                        /* Perform non-maximal suppression.                */
        void apply_hysteresis();                    /* Use hysteresis to mark the edge pixels.         */
        void optimize();                            /* Delete edges that are shorter than 20 px.       */
        void output();                              /* Write out the edge image to a file.             */

        void make_gaussian_kernel(float**, int*);   /* Create a one dimensional gaussian kernel.       */
        double angle_radians(double, double);       /* Computes the angle of a vector                  */
        void follow_edges(int, int, int);           /* A recursive routine that traces edgs along
                                                       all paths whose magnitude values remain above
                                                       some specifyable lower threshhold.              */

        bool connect(int, int, int);                /* A recursive routine that finds whether an edge
                                                       point should be preserved.                      */
```

Canny类的public函数如下，供主程序调用：

```
    public:
        Canny(char*, float, float, float, char*);   /* The constructor. */
        void edge_detection();                      /* Start the edge detection process. */
};
```

2. 编写Canny类的构造函数，对成员变量作初始化

```
Canny::Canny(char *in, float s, float tl, float th, char *dir) {
    if(VERBOSE) cout << "Reading the image " << in << "." << endl;
    infilename = in;
    src.load_bmp(infilename);
    sigma = s;
    tlow = tl;
    thigh = th;
    dirfilename = dir;
    w = src._width;
    h = src._height;
    image.assign(w, h, 1, 1, 0);
    smoothedim.assign(w, h, 1, 1, 0);
    delta_x.assign(w, h, 1, 1, 0);
    delta_y.assign(w, h, 1, 1, 0);
    magnitude.assign(w, h, 1, 1, 0);
    nms.assign(w, h, 1, 1, 0);
    edge.assign(w, h, 1, 1, 0);
    if (dirfilename != NULL) {
        sprintf(composedfname, "%s_s_%3.2f_l_%3.2f_h_%3.2f.fim", infilename,
        sigma, tlow, thigh);
        dirfilename = composedfname;
    }
}
```

3. 编写public函数edge_detection()，依次调用Canny边缘检测的各个步骤

生成灰度图、高斯模糊、求偏导：

```
void Canny::edge_detection() {
    if(VERBOSE) cout << "Starting Canny edge detection." << endl;

    /****************************************************************************
    * Convert the image to grey scale.
    ****************************************************************************/
    if(VERBOSE) cout << "Converting the image to grey scale." << endl;
    to_grey_scale();

    /****************************************************************************
    * Perform gaussian smoothing on the image using the input standard
    * deviation.
    ****************************************************************************/
    if(VERBOSE) cout << "Smoothing the image using a gaussian kernel." << endl;
    gaussian_smooth();

    /****************************************************************************
    * Compute the first derivative in the x and y directions.
    ****************************************************************************/
    if(VERBOSE) cout << "Computing the X and Y first derivatives." << endl;
    derrivative_x_y();
```

以下过程为原代码的一个可选过程，用于生成一个记录图像梯度方向的文件：

```
    /****************************************************************************
    * This option to write out the direction of the edge gradient was added
    * to make the information available for computing an edge quality figure
    * of merit.
    ****************************************************************************/
    if (dirfilename != NULL) {
        /****************************************************************************
        * Compute the direction up the gradient, in radians that are
        * specified counteclockwise from the positive x-axis.
        ****************************************************************************/
        radian_direction();

        /****************************************************************************
        * Write the gradient direction image out to a file.
        ****************************************************************************/
        if((fpdir = fopen(dirfilename, "wb")) == NULL) {
            fprintf(stderr, "Error opening the file %s for writing.\n", dirfilename);
            exit(1);
        }
        fwrite(dir_radians, sizeof(float), h * w, fpdir);
        fclose(fpdir);
        free(dir_radians);
    }
```

计算梯度大小、非最大化抑制、滞后边界跟踪、优化（去掉短边缘）并输出：

```
    /****************************************************************************
    * Compute the magnitude of the gradient.
    ****************************************************************************/
    if(VERBOSE) cout << "Computing the magnitude of the gradient." << endl;
    magnitude_x_y();

    /****************************************************************************
    * Perform non-maximal suppression.
    ****************************************************************************/
    if(VERBOSE) cout << "Doing the non-maximal suppression." << endl;
    non_max_supp();

    /****************************************************************************
    * Use hysteresis to mark the edge pixels.
    ****************************************************************************/
    if(VERBOSE) cout << "Doing hysteresis thresholding." << endl;
    apply_hysteresis();

    /****************************************************************************
    * Delete edges that are shorter than 20 px.
    ****************************************************************************/
    if(VERBOSE) cout << "Deleting short edges." << endl;
    optimize();

    /****************************************************************************
    * Write out the edge image to a file.
    ****************************************************************************/
    output();
}
```

4. 用CImg重写原代码的过程函数

用CImg对象取代原代码的指针，对图像数据进行处理。具体实现见代码文件Canny.cpp。下面以计算x方向偏导数的步骤为例。其中，smoothedim和delta_x均为CImg对象，分别记录原图像高斯模糊后的信息和x方向的偏导数信息。

```
void Canny::derrivative_x_y() {
    /****************************************************************************
    * Compute the x-derivative. Adjust the derivative at the borders to avoid
    * losing pixels.
    ****************************************************************************/
    if(VERBOSE) cout << "   Computing the X-direction derivative." << endl;
    cimg_forXY(smoothedim, x, y) {
        if (x == 0) {
            delta_x(x, y) = smoothedim(x + 1, y) - smoothedim(x, y);
        }
        else if (x == w - 1) {
            delta_x(x, y) = smoothedim(x, y) - smoothedim(x - 1, y);
        }
        else {
            delta_x(x, y) = smoothedim(x + 1, y) - smoothedim(x - 1, y);
        }
    }
```

5. 新增to_grey_scale()、optimize()和connect()函数

编写to_grey_scale()函数，使用最流行的加权平均法，将原图像转换成灰度图像：

```
/****************************************************************************
* PROCEDURE: to_grey_scale
* PURPOSE: Convert the image to grey scale.
****************************************************************************/
void Canny::to_grey_scale() {
    cimg_forXY(src, x, y) {
        int r = src(x, y, 0);
        int g = src(x, y, 1);
        int b = src(x, y, 2);
        int grey = (r * 30 + g * 59 + b * 11 + 50) / 100;
        image(x, y) = grey;
    }
}
```

编写optimize()函数，以删除长度小于20的边缘。首先对每个边缘点调用connect()函数，标记需要保留的边缘点，然后删除未被标记的边缘点：

```
/****************************************************************************
* PROCEDURE: optimize
* PURPOSE: This routine finds edges that are shorter than 20px and delete them.
****************************************************************************/
void Canny::optimize() {
    cimg_forXY(edge, x, y) {
        if (edge(x, y) == EDGE) {
            connect(x, y, 1);
        }
    }
    cimg_forXY(edge, x, y) {
        if (edge(x, y) == EDGE) {
            // delete the edge points of short edges.
            edge(x, y) = NOEDGE;
        }
        else if (edge(x, y) == PRESERVED) {
            edge(x, y) = EDGE;
        }
    }
}
```

编写connect()函数。采用深度优先搜索的思路，对边缘点八邻域上的八个位置进行递归操作，以搜索相邻的边缘点，并记录搜索路径的长度。若搜索长度超过20，或者路径与已经被标记为保留的像素点相交，则说明该路径是有效的，因此将路径上所有的边缘点标记为保留。否则，将边缘点标记为普通边缘点，后续在optimize()函数中删除。

```
/*******************************************************************************
* PROCEDURE: connect
* PURPOSE: This is a recursive routine that finds whether an edge point
* is on a long edge and should be preserved.
*******************************************************************************/
bool Canny::connect(int x, int y, int length) {
    if (x < 0 || x >= w || y < 0 || y >= h) return false;
    if (edge(x, y) == NOEDGE || edge(x, y) == VISITED) return false;
    if (length >= 20 || edge(x, y) == PRESERVED) {
        edge(x, y) = PRESERVED;
        return true;
    }
    edge(x, y) = VISITED;
    int dirX[8] = {1,1,0,-1,-1,-1,0,1},
        dirY[8] = {0,1,1,1,0,-1,-1,-1};
    bool isPreserved = false;
    for (int i = 0; i < 8; i++) {
        isPreserved = isPreserved || connect(x + dirX[i], y + dirY[i], length + 1);
        if (isPreserved) {
            edge(x, y) = PRESERVED;
            return true;
        }
    }
    edge(x, y) = EDGE;
    return false;
}
```

6. 编写output()函数，输出最终的边缘图像：

```
void Canny::output() {
    sprintf(outfilename, "%s_s_%3.2f_l_%3.2f_h_%3.2f.bmp", infilename,
            sigma, tlow, thigh);
    if(VERBOSE) cout << "Writing the edge image in the file "
                     << outfilename << "." << endl;
    edge.save(outfilename);
}
```

五. 测试过程及结果

1. 保留原代码的main()函数，来读取命令行的参数。在函数的最后，用读取的参数构造Canny对象，并执行边缘检测操作

```
Canny cny(infilename, sigma, tlow, thigh, dirfilename);
cny.edge_detection();
```

2. 编译程序

```
g++ Canny.cpp main.cpp -lgdi32 -o main
```

3. 编写test.cmd执行程序，使用不同的参数批量测试数据

```
.\main.exe bigben.bmp 1 0.5 0.5
.\main.exe bigben.bmp 2 0.5 0.5
.\main.exe bigben.bmp 3 0.5 0.5
.\main.exe bigben.bmp 1 0.1 0.5
.\main.exe bigben.bmp 1 0.9 0.5
.\main.exe bigben.bmp 1 0.5 0.2
.\main.exe bigben.bmp 1 0.5 0.8

.\main.exe lena.bmp 1 0.5 0.8
.\main.exe lena.bmp 1 0.8 0.7
.\main.exe lena.bmp 1.5 0.5 0.8
.\main.exe lena.bmp 1.5 0.3 0.9
.\main.exe lena.bmp 2 0.1 0.8
.\main.exe lena.bmp 2 0.3 0.8

.\main.exe stpietro.bmp 0.5 0.8 0.3
.\main.exe stpietro.bmp 1 0.5 0.5
.\main.exe stpietro.bmp 1 0.8 0.3
.\main.exe stpietro.bmp 1.5 0.8 0.3
.\main.exe stpietro.bmp 2 0.4 0.6
.\main.exe stpietro.bmp 2 0.5 0.5

.\main.exe twows.bmp 0.5 0.5 0.5
.\main.exe twows.bmp 1 0.3 0.5
.\main.exe twows.bmp 1 0.5 0.5
.\main.exe twows.bmp 1.5 0.5 0.5
.\main.exe twows.bmp 2 0.3 0.8
.\main.exe twows.bmp 2 0.2 0.5
```

4. 四幅测试图像各取一组结果，展示如下

原图像　　灰度图像　　梯度值图像　　边缘图像

六. 结果分析

1. 参数分析；

边缘检测过程有三个重要的参数：高斯模糊的标准差σ，滞后边缘跟踪的高阈值因子thigh和低阈值因子tlow。

i. σ的影响

固定thigh = tlow = 0.5，改变σ的值，来研究σ对边缘检测的影响。可以看到，σ越大，高斯模糊后的图像越平滑。图像越平滑，噪声越小，但边缘也越不清晰，丢失了许多细节信息。因此随着σ增大，细节边缘丢失越多（如钟的细节边缘），轮廓越简单。

σ = 1　　σ = 2　　σ = 3

σ = 1　　σ = 2　　σ = 3

ii. thigh的影响

thigh是梯度值低于高阈值的像素点占所有像素的比例，决定了高阈值的大小。固定σ = 1，tlow = 0.5，改变thigh的值。可以看出，随着thigh的增大，高阈值越大，因此梯度值较小（灰度值变化不明显）的边缘点越来越少，例如图像右下方的树叶边缘。

thigh = 0.1　　thigh = 0.5　　thigh = 0.9

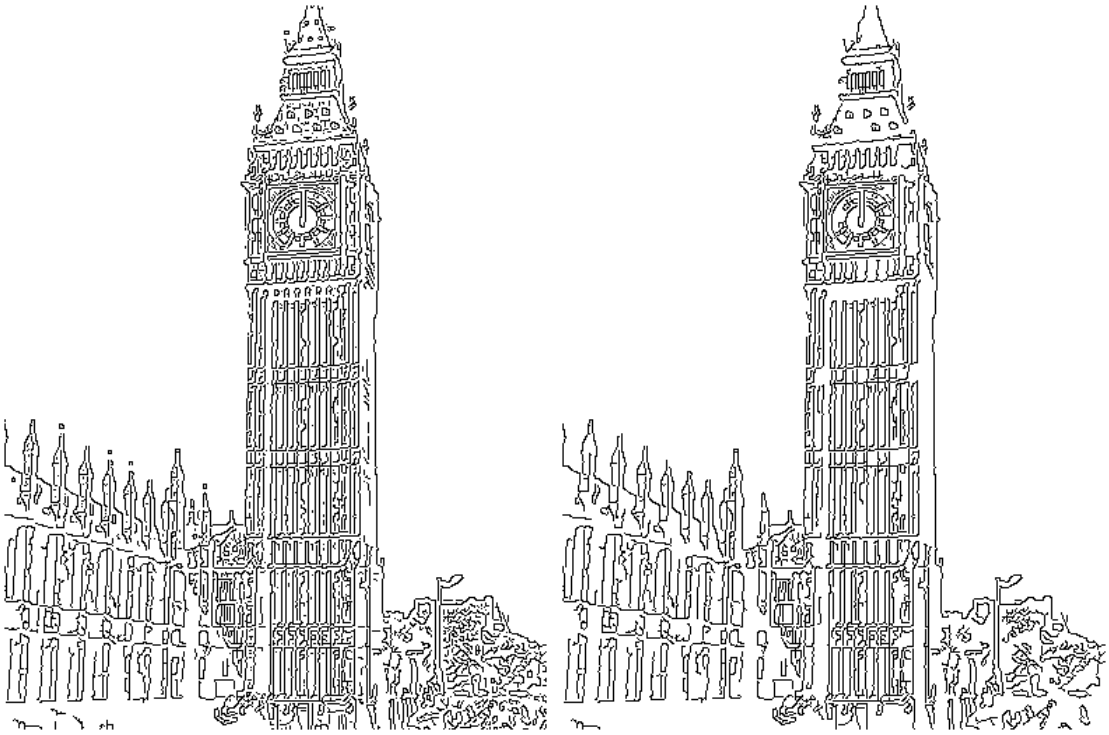
iii. tlow的影响

tlow是梯度值低于低阈值的像素点占所有梯度值低于高阈值像素的比例，与高阈值一起决定了低阈值的大小。固定σ = 1，thigh = 0.5，改变tlow的值。可以看出，随着tlow的增大，低阈值越大，因此不与真实边缘相连的边缘点（假轮廓）越来越少，如天空中的线条。但当低阈值变得很大时，真实但梯度值较小的边缘点也会被当作假边缘被删除，如图像左下方的边缘。

tlow = 0.2　　tlow = 0.5　　tlow = 0.8

2. 删除短边缘前后对比；

通过optimize()和connect()函数，可以删除长度小于20的边缘，有利于减少假边缘的出现，减轻边缘图的颗粒化现象。下面为四幅测试图像的前后对比图。



σ = 1，thigh = 0.5，tlow = 0.5

σ = 1，thigh = 0.8，tlow = 0.7

σ = 1，thigh = 0.8，tlow = 0.3

σ = 1，thigh = 0.5，tlow = 0.5

从上面的结果能够看出，增加的函数能够显著减少短边缘的数量，令边缘图更加清晰和干净。但是，这样的处理方法也有它的不足之处，在去噪的同时，会丢失一些真实的边缘信息，例如第二组图的眼睛边缘和第四组图的字母边缘。因此，该算法还存在进一步优化的空间。