

深圳大学实验报告

课程名称： 数字图像处理

实验项目名称： 图像特效显示实验

学院： 计算机与软件学院

专业： 计算机科学与技术

指导教师： 吴惠思 教授

报告人： 林浩晟 学号： 2022280310

实验时间： 2025 年 5 月 19 日

实验报告提交时间： 2025 年 5 月 19 日

教务部制

一、实验目的：

1. 掌握图像空间增强原理
2. 掌握图像边缘原理及实现方法
3. 掌握图像统计滤波原理方法
4. 掌握图像中值(统计)滤波实现方法

二、实验原理：

实验要求：

1. 熟悉 C++语言编程
2. 熟练使用 C++语言实现图像文件的读取操作
3. 熟练使用 C++语言实现图像显示方法

实验内容：

- 1、图像(3x3)模板处理函数
- 2、实现图像(Laplace)边缘检测滤波函数
- 3、实现图像中值(统计)滤波函数

①图像(3x3 模板处理)函数

```
void ImageMaskProcessing(char *oImage, char *nImage, int wImage, int hImage,
                        int *Mask, int MaskWH, int MaskCoff)
{
    int Coff;    int i, j, m, n, k;
    k = (MaskWH-1)/2;
    for (i=k; i<hImage-k; i++) {
        for (j=k; j<wImage-k; j++) {
            Coff = 0;
            for (m=-k; m<=k; m++) {
                for (n=-k; n<=k; n++) {
                    Coff+=(BYTE)oImage[(i+m)*wImage+(j+n)]*Mask[(m+k)*MaskWH+(n+k)];
                }
            }
            Coff /= MaskCoff;
            if (Coff < 0) Coff *= -1;
            if (Coff > 255) Coff = 255;
            nImage[i*wImage+j] = (unsigned char) (Coff);
        }
    }
}
```

②图像(Laplace)边缘检测滤波函数

```
void LaplaceEdgeProcessing(char *oImage, char *nImage,
                        int wImage, int hImage)
{
    int Mask[9] = {0, 1, 0,           //Laplace 边缘检测模板
```

```

        1, -4, 1,
        0, 1, 0};

    ImageMaskProcessing(oImage, nImage, wImage, hImage, Mask, 3, 1);
}

```

③图像中值(统计)滤波函数

```

void MiddleFilterProcessing(char *oImage, char *nImage, int wImage, int
hImage)
{
    int i, j, m, n, k, l;
    int vSort[32], Middle, MiddlePos;
    for (i=1; i<hImage-1; i++) {
        for (j=1; j<wImage-1; j++) {l = 0;
            for (m=-1; m<=1; m++) {
                for (n=-1; n<=1; n++) {
                    vSort[l++] = (BYTE) oImage[(i+m)*wImage+(j+n)];}
                for (k=0; k<5; k++) {
                    Middle = vSort[k]; MiddlePos = k;
                    for (l=k; l<9; l++) {
                        if (Middle > vSort[l]) {Middle = vSort[l]; MiddlePos =
l;}}
                    vSort[MiddlePos] = vSort[k]; vSort[k] = Middle;}
                nImage[i*wImage+j] = (unsigned char) Middle;
            }
        }
    }
}

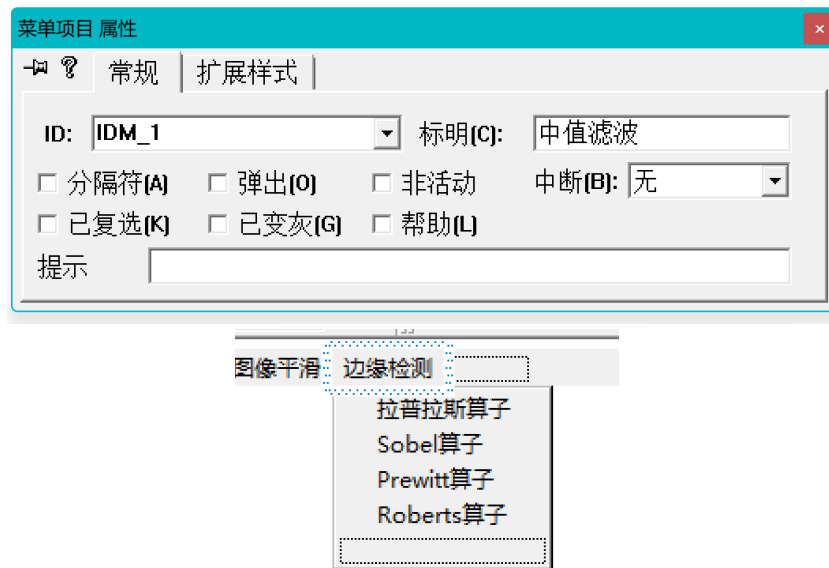
```

三、实验用品：

计算机、visual C++6.0

五、实验现象及数据处理：

1. 在 Menu 中添加所有的选项，如下图所示；



2. 随后先声明函数，方便调用；同时定义 NewImage 变量，用于保存处理过的图像，方便显示处理过的图像。

```
void ShowImage(char *, int, int, int, int);
BOOL ReadBmpImage(LPSTR, char *);
void ShowBmpImage(char *, int, int, int, int);
void OpenImageFileDialog(char *);
void LaplaceEdgeProcessing(char *oImage, char *nImage,
                           int wImage, int hImage);
void ImageMaskProcessing(char *oImage, char *nImage, int wImage, int hImage,
                          int *Mask, int MaskWH, int MaskCoff);
void MiddleFilterProcessing(char *oImage, char *nImage, int wImage, int hImage);
void SobelEdgeProcessing(char *oImage, char *nImage, int wImage, int hImage);
void PrewittEdgeProcessing(char *oImage, char *nImage, int wImage, int hImage);
void RobertsEdgeProcessing(char *oImage, char *nImage, int wImage, int hImage);
HDC hWinDC,
int ImageWidth, ImageHeight;
char ImgDlgFileName[MAX_PATH];
char ImgDlgFileDir[MAX_PATH];
char OrgImage[1024*1024];
char NewImage[1024*1024];
#define IMAGEWIDTH 256
#define IMAGEHEIGHT 256
#define XPOS 100
```

3. 在主函数中，调用对应的函数，并且显示处理过的图像。

```

        newBmpImage(imgviz_filename, orgImage);
        ShowBmpImage(orgImage, ImageWidth, ImageHeight, XPOS, YPOS);
        break;
    case IDM_1:
        MiddleFilterProcessing(orgImage, NewImage, IMAGEWIDTH, IMAGEHEIGHT);
        ShowImage(NewImage, IMAGEWIDTH, IMAGEHEIGHT, XPOS, YPOS+300);
        break;
    case IDM_2:
        LaplaceEdgeProcessing(orgImage, NewImage, IMAGEWIDTH, IMAGEHEIGHT);
        ShowImage(NewImage, IMAGEWIDTH, IMAGEHEIGHT, XPOS, YPOS+300);
        break;
    case IDM_3:
        SobelEdgeProcessing(orgImage, NewImage, IMAGEWIDTH, IMAGEHEIGHT);
        ShowImage(NewImage, IMAGEWIDTH, IMAGEHEIGHT, XPOS, YPOS+300);
        break;
    case IDM_4:
        PrewittEdgeProcessing(orgImage, NewImage, IMAGEWIDTH, IMAGEHEIGHT);
        ShowImage(NewImage, IMAGEWIDTH, IMAGEHEIGHT, XPOS, YPOS+300);
        break;
    case IDM_5:
        RobertsEdgeProcessing(orgImage, NewImage, IMAGEWIDTH, IMAGEHEIGHT);
        ShowImage(NewImage, IMAGEWIDTH, IMAGEHEIGHT, XPOS, YPOS+300);
        break;
    case IDM_ABOUT:
        DialogBox(hInst, (LPCSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About);
        break;
    case IDM_EXIT:
        DestroyWindow(hWnd);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
}

```

4. 图像中值(统计)滤波函数如下, 该函数对原图像的每一个像素进行中值滤波, 处理结果存入 nImage; 首先定义变量, 其中 i, j 表示遍历图像的每个像素位置, m, n 表示遍历该像素邻域的偏移量, k, l 用于排序等逻辑, vSort[32] 用于存放邻域像素值, Middle 表示当前最小值或最终中值, MiddlePos 表示当前最小值的索引 (用于交换); 外层双重循环用于遍历所有非边界像素, 同时从 1 到 hImage-2, 避免越界, 而 l = 0 是用于计数当前放入 vSort 的位置; 内层双重循环用于提取 3x3 邻域像素, 遍历中心像素 (i, j) 周围的 3x3 区域, 将其值 (8 位无符号整数) 依次放入 vSort 数组中, l 最终为 9; 再两层循环用于排序, 寻找中值; 最后将排序结果中找到的中值 Middle 赋给输出图像中对应位置;

```

void MiddleFilterProcessing(char *oImage, char *nImage, int wImage, int hImage)
{
    int i, j, m, n, k, l;
    int vSort[32], Middle, MiddlePos;
    for (i=1; i<hImage-1; i++) {
        for (j=1; j<wImage-1; j++) {l = 0;
            for (m=-1; m<=1; m++) {
                for (n=-1; n<=1; n++) {
                    vSort[l++] = (BYTE) oImage[(i+m)*wImage+(j+n)];}
                for (k=0; k<5; k++) {
                    Middle = vSort[k]; MiddlePos = k;
                    for (l=k; l<9; l++) {
                        if (Middle > vSort[l]) {Middle = vSort[l]; MiddlePos = l;}
                        vSort[MiddlePos] = vSort[k]; vSort[k] = Middle;}
                    nImage[i*wImage+j] = (unsigned char) Middle;
                }
            }
        }
    }
}

```

5. 图像(3x3 模板处理)函数如下, 用于图像卷积, 进行掩模处理; 首先定义变量, Coff 为卷积结果 (临时存放计算总和), k 为卷积核半径; 再遍历图像中间区域 i、j 是当前像素的坐标, 每个像素将用其周围 MaskWH × MaskWH 的邻域进行处理; 随后进行卷积操作, 遍历卷积核的每一个元素, 取图像当前点 (i, j) 的邻域像素值与掩模对应元素相乘, 然后累加结果, 最终得到卷积值 Coff; 最后进行卷积值归一化、取绝对值、限制范围等。

```

void ImageMaskProcessing(char *oImage, char *nImage, int wImage, int hImage,
                        int *Mask, int MaskWH, int MaskCoff)
{
    int Coff;    int i, j, m, n, k;
    k = (MaskWH-1)/2;
    for (i=k; i<hImage-k; i++) {
        for (j=k; j<wImage-k; j++) {
            Coff = 0;
            for (m=-k; m<=k; m++) {
                for (n=-k; n<=k; n++) {
                    Coff+=(BYTE)oImage[(i+m)*wImage+(j+n)]*Mask[(m+k)*MaskWH+(n+k)];
                }
            }
            Coff /= MaskCoff;
            if (Coff < 0) Coff *= -1;
            if (Coff > 255) Coff = 255;
            nImage[i*wImage+j] = (unsigned char) (Coff);
        }
    }
}

```

6. 图像(Laplace)边缘检测滤波函数如下，实现了拉普拉斯算子 (Laplace Operator) 的边缘检测。首先定义 Mask，一个 3x3 拉普拉斯模板；随后调用 ImageMaskProcessing 函数，这个函数会将 3x3 的拉普拉斯卷积核 应用到输入图像 oImage 上，并将处理后的结果存储到 nImage 中；其中 3 表示卷积核的宽度和高度是 3(即 3x3 卷积核)，1 是卷积核的归一化系数(MaskCoff)，对于拉普拉斯算子，由于其已经归一化，因此 MaskCoff 设置为 1。

```

void LaplaceEdgeProcessing(char *oImage, char *nImage,
                          int wImage, int hImage)
{
    int Mask[9] = {0, 1, 0,           //Laplace边缘检测模板
                  1,-4, 1,
                  0, 1, 0};

    ImageMaskProcessing(oImage, nImage, wImage, hImage, Mask, 3, 1);
}

```

7. 图像(Sobel)边缘检测滤波函数如下，使用两个方向的卷积核（分别计算水平方向和垂直方向的梯度），然后结合这两个梯度来得到最终的边缘图像；首先定义 Sobel 水平梯度卷积核 MaskX，Sobel 垂直梯度卷积核 MaskY；再分配两个临时图像缓冲区 GxImage 和 GyImage，分别用于存储水平方向 (Gx) 和垂直方向 (Gy) 的梯度结果，这两个缓冲区的大小与原始图像 oImage 一样，都是 wImage * hImage 字节；随后通过调用 ImageMaskProcessing 函数分别使用水平方向的 Sobel 卷积核和垂直方向的 Sobel 卷积核对原始图像进行卷积操作来计算相应的梯度值；再通过合并 Gx 和 Gy 的结果，得到最终的边缘强度，其中 g 是这两个梯度的近似梯度，计算方法是 $|Gx| + |Gy|$ ，并且需要限制 g 的范围在灰度值的范围中：0 到 255；最后在处理完成后，释放之前分配的临时内存缓冲区 GxImage 和 GyImage，避免内存泄漏。

```

void SobelEdgeProcessing(char *oImage, char *nImage, int wImage, int hImage)
{
    int MaskX[9] = {
        -1, 0, 1,
        -2, 0, 2,
        -1, 0, 1
    };
    int MaskY[9] = {
        -1, -2, -1,
        0, 0, 0,
        1, 2, 1
    };
    // 临时图像缓冲区用于存放 Gx 和 Gy 结果
    char *GxImage = (char *)malloc(wImage * hImage);
    char *GyImage = (char *)malloc(wImage * hImage);
    // 使用 ImageMaskProcessing 计算 Gx 和 Gy
    ImageMaskProcessing(oImage, GxImage, wImage, hImage, MaskX, 3, 1);
    ImageMaskProcessing(oImage, GyImage, wImage, hImage, MaskY, 3, 1);
    // 合并 Gx 和 Gy 结果
    for (int i = 0; i < hImage; i++) {
        for (int j = 0; j < wImage; j++) {
            int gx = (unsigned char)GxImage[i * wImage + j];
            int gy = (unsigned char)GyImage[i * wImage + j];
            int g = abs(gx) + abs(gy); // 近似梯度
            if (g > 255) g = 255;
            nImage[i * wImage + j] = (unsigned char)g;
        }
    }
    free(GxImage);
    free(GyImage);
}

```

8. 图像(Prewitt)边缘检测滤波函数如下, Prewitt 算子代码与 Sobel 算子代码没有很大的差异, 只需要更改 Prewitt 水平梯度卷积核 MaskX, Prewitt 垂直梯度卷积核 MaskY 即可, 因此这里不过多赘述。

```

void PrewittEdgeProcessing(char *oImage, char *nImage, int wImage, int hImage)
{
    int MaskX[9] = {
        -1, 0, 1,
        -1, 0, 1,
        -1, 0, 1
    };
    int MaskY[9] = {
        -1, -1, -1,
        0, 0, 0,
        1, 1, 1
    };

    // 分别存储 Gx 和 Gy 的卷积结果
    char *GxImage = (char *)malloc(wImage * hImage);
    char *GyImage = (char *)malloc(wImage * hImage);
    // 通过 ImageMaskProcessing 分别计算 Gx 和 Gy
    ImageMaskProcessing(oImage, GxImage, wImage, hImage, MaskX, 3, 1);
    ImageMaskProcessing(oImage, GyImage, wImage, hImage, MaskY, 3, 1);
    // 合并 Gx 和 Gy 得到梯度幅值
    for (int i = 0; i < hImage; i++) {
        for (int j = 0; j < wImage; j++) {
            int gx = (unsigned char)GxImage[i * wImage + j];
            int gy = (unsigned char)GyImage[i * wImage + j];
            int g = abs(gx) + abs(gy); // 简化梯度幅值计算
            if (g > 255) g = 255;
            nImage[i * wImage + j] = (unsigned char)g;
        }
    }
    free(GxImage);
    free(GyImage);
}

```

9. 图像(Roberts)边缘检测滤波函数如下；首先遍历图像的每一个像素，其中范围是 $hImage - 1$ 和 $wImage - 1$ ，这是因为 Roberts 算子使用 2×2 的区域计算，所以不能遍历到最底和最右的边界像素，否则越界；再实现 $gx = I(x, y) - I(x+1, y+1)$ ，对应 Roberts 的 G_x 卷积核，取当前像素与右下角像素之差，同时实现 $gy = I(x+1, y) - I(x, y+1)$ ，对应 Roberts 的 G_y 卷积核，取当前像素下方与右边像素之差；随后计算梯度幅值，取两个方向梯度的绝对值并相加，作为边缘强度的近似值；最后防止灰度值不合法，将计算得到的边缘强度写入输出图像对应位置。

```
void RobertsEdgeProcessing(char *oImage, char *nImage, int wImage, int hImage)
{
    int gx, gy, g;

    for (int i = 0; i < hImage - 1; i++) {
        for (int j = 0; j < wImage - 1; j++) {
            gx = (unsigned char)oImage[i * wImage + j] -
                (unsigned char)oImage[(i + 1) * wImage + (j + 1)];

            gy = (unsigned char)oImage[(i + 1) * wImage + j] -
                (unsigned char)oImage[i * wImage + (j + 1)];

            g = abs(gx) + abs(gy); // 梯度强度

            if (g > 255) g = 255;
            if (g < 0) g = 0;

            nImage[i * wImage + j] = (unsigned char)g;
        }
    }
}
```

10. 运行函数得到如下结果

中值滤波



拉普拉斯算子

拉普拉斯算子
Sobel算子
Prewitt算子
Roberts算子





Hello World!

Sobel 算子

示RAW图像 显示BMP图像 Help 图像平滑 边缘检测

拉普拉斯算子
Sobel算子
Prewitt算子
Roberts算子







Hello World!

Prewitt 算子

JKAW图像 显示BMP图像 Help 图像平滑 边缘检测

拉普拉斯算子
Sobel算子
Prewitt算子
Roberts算子



Hello World!



Roberts 算子

拉普拉斯算子
Sobel算子
Prewitt算子
Roberts算子

Hello World!



六、实验结论：

实验成功完成！成功实现了中值滤波、拉普拉斯算子、Sobel 算子、Prewitt 算子、Roberts 算子。

实验感想：

通过本次实验，我对图像处理中的空间增强、边缘检测和统计滤波有了更深刻的理解。在实现图像模板处理函数时，我体会到了卷积操作对图像特征提取的重要性，通过不同模板可以实现多种效果。Laplace 边缘检测让我看到了边缘提取的神奇效果，它能清晰地勾勒出图像的轮廓。中值滤波的实现过程让我明白了排序在图像去噪中的关键作用，它能有效去除椒盐噪声。

指导教师批阅意见：
成绩评定：
指导教师签字：
年 月 日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。