25秋季学期计算机程序设计大作业

C语言数字图像处理

本课题负责人：刘明乐

一、前置知识

数字图像，顾名思义，就是数字化的图像。其由称为像素的最小单位构成，每个像素都有其值，这个值决定了像素的颜色。

从颜色的分类上来说，数字图像可以分为灰度图像和彩色图像。灰度图像的像素值称为灰度值，**其取值范围有两种表示方式：以8位二进制数表示的0-255，或者除以255归一化之后得到的double类型的0-1之间的小数，0为黑色，255或1为白色**；彩色图像同理，但彩色图像的像素值是一个向量，向量与实际颜色的映射关系称为通道，目前彩色图像中最常见的是RGB通道，即红绿蓝三个通道，这也是本课题中需要用到的通道。每个通道的取值范围与灰度图像相同。即**本课题中彩色图像像素值为一个三维向量，每个分量的取值范围为0-255（或0-1）。**

数字图像显然应该是二维的，即数字图像的本质是一个二维数组。其坐标系与数学中的坐标系相同，只是旋转了90度。原点在左上角，x轴指向下方，这个方向也称为图像的高度，y轴指向右方，这个方向也称为图像的宽度。即坐标表示如下：

$$\begin{matrix} (0,0) & (0,1) & \cdots & (0,w-1) \\ (1,0) & (1,1) & \cdots & (1,w-1) \\ \vdots & \vdots & \ddots & \vdots \\ (h-1,0) & (h-1,1) & \cdots & (h-1,w-1) \end{matrix}$$

其中，h为图像的高度，w为图像的宽度。已经自动按照计科的习惯从零开始数数（笑）。

二、环境配置

本课题需要用到两个图像读入和写出的头文件`stbi_image.h`和`stbi_image_write.h`。这两个头文件已经在课题的压缩包中，只需要将其随便找一个位置解压就可以使用，使用方法如下：

```
//以解压到桌面为例,记得改成自己解压的位置
#define STB_IMAGE_IMPLEMENTATION
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "C:/desktop/stb_image.h"
#include "C:/desktop/stb_image_write.h"
```

即你的程序应该以以上四行作为开头。
接下来介绍三个用于读入、写出的函数：

```
//读入图像用的函数stbi_load
/*
该函数成功运行之后，会将图像的宽度、高度、和“实际通道数”
分别存储进width、height、channels中。
```

```
本课题中不需要用到channels变量，故不做解释
最后一个参数，1代表灰度图，3代表RGB通道彩色图，不需要其他数字
其返回值如你所见为unsigned char*类型，指向存储图像像素值的数组的首地址。
*/
int width, height, channels;
unsigned char *image = stbi_load("a.jpg", &width, &height, &channels, 1);

/*
写出图像的的函数stbi_write_jpg
第一个参数代表输出的文件地址，第二个参数应该传入图像的宽度，第三个参数为高度。
第四个参数和stbi_load函数的最后一个参数一样，
若填入1，则会输出灰度图像，
若填入3，则会输出RGB通道的彩色图像。
第五个参数为图像的像素值数组，必须为unsigned char*类型。
最后一个参数为输出图像的压缩质量，100代表完全不压缩，所以固定填入100即可。
*/
stbi_write_jpg("ex1.jpg", width, height, 1, output, 100);

/*
释放内存的函数stbi_image_free
就如同malloc函数分配的内存需要使用free函数释放一样，stbi_load函数申请到的图像内存也需要用
stbi_image_free函数释放。
*/
stbi_image_free(image);
```

必须要说明的是：`stbi_load`函数申请到的内存是一维数组，是把二维图像拉成一维之后的结果。数组的长度是$height \times width$，具体的排列方式如下：

$$(0,0)\,(0,1)\cdots(0,w-1)\,(1,0)\,(1,1)\cdots(1,w-1)\cdots(h-1,0)\,(h-1,1)\cdots(h-1,w-1)$$

对于彩色图像则是

$$(0,0,r)\,(0,0,g)\,(0,0,b)\,(0,1,r)\cdots$$

并且，在使用`stbi_write_jpg`输出图像的时候，也必须传入`unsigned char*`类型的一维数组

基于以上原因，强烈建议各位全程都在一维数组的视角下处理图像，脑子里知道图像是二维的即可。

三、任务目标

本课题不允许使用任何奇怪的库函数以至于数行就写完的情况发生，违者对应任务点不得分！！！！

Task1 图像的平移旋转和放缩（10分）

最基本、最简单、最直观的图像处理操作集合，完全不需要对原图像的像素值进行任何处理，只需要处理像素的坐标。

线性代数高手都知道，由一个坐标`(x,y)`绕原点旋转角度θ从而得到新坐标`(x',y')`的公式如下所示：

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

对于数字图像而言，坐标`(x,y)`都是整数（因为图像的坐标都是离散的），但由于旋转的角度θ为任意角度，新坐标可不一定是整数。这给我们旋转图像带来了一定的麻烦。

因此，我们需要使用逆向思维，从新图像的一个整数坐标出发，反向旋转θ角度，查看它在原图像中落在什么位置，由于反向旋转后坐标变得不是整数，它会落在**某四个紧挨着的像素围成的方形区域内。**例如，假设有一坐标为（4.5,4.5），那他显然落在（4,4）（4,5）（5,4）（5,5）四个像素之间，不是吗？而对于我们而言，原图像是已知的，我们可以通过这周围的四个像素来决定这个新图像的整数坐标上应该是什么像素值。

有两种决定方法（专业名称叫插值法）：一、最近邻插值法：看落点离四个像素哪个最近，就拿哪个像素值做新图像整数坐标的像素值。二、双线性插值法：设（x,y）落在A（i，j）B（i，j+1）C（i+1，j）D（i+1，j+1）之间，f为像素到像素值的函数，则有:

$$f(E) = (x - i)(f(C) - f(A)) + f(C)$$
$$f(F) = (x - i)(f(D) - f(B)) + f(B)$$
$$f(x, y) = (y - j)(f(F) - f(E)) + f(E)$$

用这个f(x,y)作为新图像那个整数坐标的像素值即可。什么？不是整数？四舍五入即可。

本任务中的旋转不需要考虑超出边界的问题（即反向旋转后若发现其在原图像坐标之外，直接像素值置零），旋转后的图像大小应与原图像保持一致。

放缩也是同理，对于原图像的坐标`(x,y)`放缩后的坐标，如`(1.5x,0.5y)`可不一定是整数。仍然采用上述介绍的反向思维，利用两种插值法进行图像的放缩工作。理所当然的，放缩的图像大小应与原图像**“不一致”**

Task1任务目标为：利用上述介绍的方法，1、完成图像`alphabet1.jpg`的绕“任意”像素点旋转任意角度的处理；2、完成对任意图片的放缩处理；3、对比两种插值法给人视觉效果的不同。

Task2 图像的直方图均衡化（10分）

来介绍灰度图像的直方图，简单来说就是**统计图像中各个像素值出现频率的柱状图**，横坐标为灰度值，纵坐标为频率。频率由灰度值出现的次数d_k(k为灰度值),除以图像的像素总数$height \times width$得到，记灰度值k出现的频率为p_k。把p_0到p_{255}总共256个柱子并列在一起就得到了图像的直方图。

若一个图像的整体亮度偏暗，则图像的直方图也会整体偏向左侧，整体偏亮则会偏向右侧。直方图均衡化是为了将这种图像的灰度均衡地拉伸至0-255全体范围内，以增强图像的明暗对比，平衡图像的整体灰度。

要进行直方图均衡化，需要用到**累计直方图**。这是计算灰度值小于等于k的累计频率的统计图，灰度值k对应的累计频率记为s_k。有$s_k = \sum_{i=0}^{k} p_i$。接下来利用均衡化公式$F_k = (int)((L-1)s_k + 0.5)$得到原图像灰度值k到新图像灰度值$F_k$的映射，L为灰度值可能取值的个数，在本课题中取256。

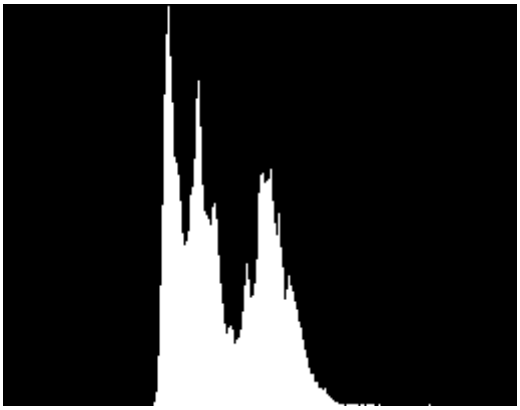
助教已经写好了一个可以画出直方图的函数，各位可以直接拿去用：

```
void histogram(int hist[]){
    int H = 200;
    unsigned char *out = calloc(256 * H, 1);
    memset(out, 0, 256*H);

    int max = hist[0];
    for(int i = 1; i < 256; ++i){
        if(hist[i] > max) max = hist[i];
    }

    for (int x = 0; x < 256; x++) {
```

```
            int bar_h = (int)((double)hist[x] / max * (H - 1));
            for (int y = 0; y < bar_h; y++) {
                out[(H - 1 - y) * 256 + x] = 255;
            }
        }

        stbi_write_jpg("hist.jpg", 256, H, 1, out, 100);
        free(out);
    }
```

传入的数组hist长度必须为256，否则助教不保证会发生什么乱七八糟的越界错误

函数运行画出来的直方图长成这样：



虽然既没有图例也没有横坐标也没有纵坐标，但是结合前面的文字说明相信已经足够让各位看懂这是个什么玩意了。

Task2任务目标为：对于图片`pout.jpg`和`lena.jpg`进行直方图均衡化，画出均衡化之前和均衡化之后的直方图，并对比均衡化前后的视觉效果

Task3 图像的空间域滤波去噪（15分）

先进行一个卷积操作的介绍：假设现在有一张3x3的小图片$\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}$,还有一个用于卷积的算子$\begin{pmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{pmatrix}$，卷积的操作如下：

将算子的中心分别对准图片的每一个像素，将算子和图片重合的部分分别进行相乘，将所有相乘的结果进行求和，用该结果代替算子中心对准的那个像素

例如将上述算子的中心对准图片的中心，5的位置，则分别有-1和1、-1和4、-1和7、0和2、0和5、0和8、1和3、1和6以及1和9重合，卷积的结果为
$(-1)\times 1+(-1)\times 4+(-1)\times 7+1\times 3+1\times 6+1\times 9=6$。用这个结果去代替原本的5即可。

算子的中心对准图像的边缘地带时，可能会出现算子超出图片边界的情况，这种情况一般有两种处理方式：一种是直接填零，另一种是像某些rpg游戏一样，超出地图的边界时会突然从地图的另一边出现，即上述图片中1的上方是7，1的左侧是3，依此类推。

在填零的情况下，上述例子的卷积结果为$\begin{pmatrix}7&4&-7\\15&6&-15\\13&4&-13\end{pmatrix}$

有些算子无法直观地使用矩阵形式表达，比如中值算子，将3x3的区域中心对准一个像素，然后取3x3区域里的中位数以代替对准的那个像素值。

用算子对图片进行卷积的过程，就叫做滤波。本任务点不需要对图片做预处理，直接对图像的灰度值空间进行操作，故称为图像的空间域滤波。最常见、最简单的空间域滤波方式为均值滤波和中值滤波，即将一个区域内的均值或中位数去代替区域中心对准的图片像素值。**本任务点需要进行四个滤波**：一、超限邻域均值滤波：先决定好邻域的大小，如3x3，5x5（一般为奇数边长），然后计算邻域内像素值的均值，若邻域中心对准的像素值和该均值差的绝对值超过了一个阈值T，则将该像素视为**噪声**，用均值代替该像素值。二、超限邻域中值滤波：同上，只是采用邻域内像素值的中位数。另外两个就是普通的均值滤波和中值滤波（不设阈值）。

这也体现了空间域滤波的主要用途，去噪。助教准备了`lena_saltpepper.jpg`、`lena_gaussian.jpg`和`lena_random.jpg`三个噪声图像，分别对应3%椒盐噪声、高斯噪声和3%随机噪声。

Task3任务目标为：对以上三个噪声图像进行上述四种滤波，横向比较不同滤波方式对不同噪声的效果。

Task4 彩色图像的K-means算法有损压缩（20分）

正如环境配置一段所述，彩色图像在RGB通道下是一个分散在三维空间（R，G，B三个轴）的一串向量集合。既然是分散在全空间内的散点，那么就适用人工智能算法中一类非常普遍且重要的算法——聚类算法：将分散的点按照一定的规则进行分类的算法。

聚类算法中，一个特别简便的算法为K-means算法，其步骤如下：

1、在全空间中随机选取K个点，作为K个聚类的中心。

2、将所有的向量分配给离自己最近的那个中心（聚类步）

3、每一类的向量分别取均值（即mean），总共得到K个新的中心（迭代步）（什么？均值不是整数？四舍五入呗？」

4、反复执行聚类步和迭代步，直到K个中心都不再发生变化（算法收敛）

算法结束之后，每一个向量都有自己所属的类。对于彩色图像，用每一个像素的所属类的中心值去替代像素值即可完成彩色图像的有损压缩。为了减轻大家的负担，助教不要求各位实际进行内存上的压缩和压缩率的计算了。

Task4任务目标为：对`cow.jpg`进行K值分别为2,4,8,16,32,64的K-means算法压缩，横向比较不同K值压缩结果图的视觉效果

感兴趣的同学还可以试试K-means的兄弟算法K-medians，K-medoids和K-centers等

Task5 图像的频率域滤波（20分）

任务3中我们介绍了空间域滤波，因为其为不对图像做预处理，直接在像素图的灰度值空间做处理的滤波，故称为空间域滤波。

本任务点需要使用各位下学期数学分析会学到的大型知识点——**傅立叶变换**对数字图像进行预处理，将图像从空间域变换至频率域，在频率域进行的滤波，称为频率域滤波。

不出所料的话，数学分析只会教给各位连续空间的傅立叶变换，故此，助教给出适用于图像处理的离散空间傅立叶变换。对于一个长度为N的数组，其傅里叶变换如下：

$$F[k]=\sum_{x=0}^{N-1} f(x)e^{-i2\pi xk/N} \qquad k=0,1,2,\ldots,N-1$$

i为虚数单位

该变换将一个以x为自变量的函数变换为以k为自变量，我们称之从空间域变换至频率域，**本任务点在完成的时候，f(x)，即原图像需要以double类型，取值范围为0-1的格式出现。因此用`stbi_load`函数读入图像后，请第一时间除以255进行归一化。**

$F[k]$是一个复数类型的函数，请自行设法实现C语言下的复数计算。由于图像是二维的，我们需要进行的是**二维的傅立叶变换，这也很简单，先对每一行都进行傅立叶变换，然后对每一列都进行傅立叶变换即可。**

一般而言，傅立叶变换之后，我们认为低频分量会分布在图像的边角处，但我们希望在进行频率域滤波的时候低频分量能够处于图像中心部分。假设有一频率域图像如下：

$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix}$。进行低频分量（主要是处于1号位置的分量）平移至图像中心的变换：

$\begin{bmatrix} 14 & 15 & 11 & 12 & 13 \\ 4 & 5 & 1 & 2 & 3 \\ 9 & 10 & 6 & 7 & 8 \end{bmatrix}$

没错，就跟介绍卷积操作的时候一样，平移时被挤出地图的家伙从地图的另一边出现了。不管图像的边长是奇数还是偶数，总之把左上角的那个平移到坐标为$(height/2, width/2)$的地方即可（坐标从0开始计数）。

接下来介绍**频率域图的查看方法**：1、幅度谱——将空间域图进行傅立叶变换变成频率域图、将低频分量移至中心、对每个像素值（目前应该都是复数）取各自的模长，即幅度、对每个分量进行ln(1+x)的映射，然后将**最小值映射成零，最大值映射成一，其他数值线性映射即可**，最后输出图像。2、幅度谱逆变换——傅立叶变换至频率域、取幅度、傅立叶逆变换至空间域、取实部、**小于0的截断成0、大于1的截断成1**、输出。3、相位谱逆变换——变换至频率域、对每个像素取复数的辐角θ、将每个像素重新构建为$Ae^{i\theta}$，A为复数的幅度，是一个实数，具体多少各位自行决定，输出的图好看就行、然后将重构建的图逆变换至空间域、对每个像素取模长、**截断至0-1**、输出图像。

接下来介绍**频率域滤波步骤**：变换至频率域、低频分量移至中心、根据像素距离低频分量的距离进行滤波、低频分量移回原位、逆变换回空间域、对每个像素取模长、**线性映射至0-1**进行输出。本次要实现的滤波方式有三个：**理想低通滤波器**——记像素距离中心的距离为D，阈值为D_0，将$D \leq D_0$的部分保持原样，其余部分舍弃成零；**巴特沃斯低通滤波器**——对每个像素都乘上系数$\frac{1}{(1+D/D_0)^{2n}}$，n为滤波器的阶数；**高斯低通滤波器**——对每个像素都乘上系数$exp(-(D/D_0)^n)$，n为滤波器的阶数。

理想低通滤波器对应的**高通滤波器**，仅需要将原本舍弃的部分保持原样，保持原样的部分舍弃即可得到，其余两个高通滤波器通过把系数中的D/D_0变为D_0/D即可得到。

似乎忘了介绍**傅立叶逆变换**，补个公式在下面：

$$f[x] = \frac{1}{N}\sum_{k=0}^{N-1} F[k]e^{i2\pi xk/N} \qquad x = 0, 1, \ldots, N-1$$

Task5任务目标为：1、生成`Rect1.jpg` `Rect2.jpg`的幅度谱图、幅度谱逆变换图、相位谱逆变换图；2、对`Girl.jpg` `pout.jpg`进行上述介绍的$3 \times 2 = 6$种频率域滤波手段进行滤波，观察不同D_0和n下输出图像的不同。

Task6 图像的四叉树分裂合并算法（25分）

如果说上一个任务是算法复杂但实现代码量较少的话，该任务则刚好相反。

算法步骤如下：

1、若区域符合分裂条件，则分裂成均匀的4块（口分裂成一个田），不断地分裂直至所有的块都已经：边长到达最小设定值，不允许再分裂，或不再符合分裂条件，没必要再分裂。

2、若相邻的区域满足合并条件则进行合并，不断的合并直至再也没有能够合并的相邻区域。

完毕。

本任务点给出的目标图像是256x256大小，各位不需要考虑边长为奇数的情况，请把注意力集中在算法实现上就好。

Task6任务目标为对 `cameraman.jpg` 进行分裂处理，将分裂结果的每个块的右边缘和下边缘都置成白色（255）以输出分裂图。然后对分裂的图进行合并处理，右边缘和下边缘置成白色以输出合并图，请各位自行尝试不同的阈值以进行对比。

效果图大致如下：

建议最开始尝试的分裂条件为：以0-1范围表示的区域的极差大于某个阈值。合并条件为：两相邻区域的并集的极差小于某个阈值

有余力的同学可以自己创造自己的分裂合并条件以进行更多的尝试

Task EX 最优图像边缘检测算子——Canny算子青春版（20分）

该任务点为本次大作业的附加题，为选作项。

Canny算子并非简单的用矩阵和卷积操作就能实现的算子，助教对Canny算子进行了一定简化，简化后的算法步骤如下：

1、高斯平滑

生成一个高斯算子，这是一个$n \times n$的矩阵，n取奇数。记矩阵元离算子中心的距离为D，则算子中每个元素的值为$exp(\frac{-D^2}{2\sigma^2})$，**$n$和$\sigma$为这一步的超参数。**然后对算子中的所有元素求和，然后对算子中的所有元素都除以这个总和，进行归一化。

使用生成的这个高斯算子对图像进行卷积操作，这一步就是高斯平滑。算子超出图像边界的地方置零即可。

2、使用Sobel算子求图像的梯度

Sobel算子分成x方向的算子和y方向的算子，形式如下：

$$Sobel_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad Sobel_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

用两个算子分别对图像做卷积操作，得到x和y方向上的梯度图。注意这里我们对于x轴和y轴的描述变回了普通坐标系的方向。

3、求梯度的大小和方向

对每个像素分别求梯度的大小和方向，记x方向的梯度为$\frac{\partial f}{\partial x}$，y方向的梯度为$\frac{\partial f}{\partial y}$。

则梯度的大小为$\sqrt{(\frac{\partial f}{\partial x})^2+(\frac{\partial f}{\partial y})^2}$，梯度的方向为$arctan(\frac{\partial f}{\partial y}/\frac{\partial f}{\partial x})$。建议梯度的方向使用C语言自带的`atan2`函数进行计算，会返回$[-\pi,\pi]$的返回值。需要把小于零的返回值都加上π，使所有的方向都在$[0,\pi]$中，然后转换成角度制以备后用。

4、非极大值抑制（Non-Max Suppression，NMS）

梯度方向在$[0°,22.5°]\cup[157.5°,180°]$的像素，其方向认为是左右方向，梯度方向在$[22.5°,67.5°]$的像素，方向认为是右上、左下方向，依此类推总共有四个方向。

将像素的梯度大小和其方向上的两个像素的梯度大小做比较，如左右方向的像素需要和其左侧和右侧的相邻像素比较，依此类推。除非该像素是参与比较的三个像素的最大值，可以保持其梯度大小，否则梯度大小直接归零。称为非极大值抑制。

图像边缘一圈的像素会缺乏一个甚至两个比较对象，因此可以不参与这一步。

这一步得到的有一部分梯度大小归零了的梯度大小图，称为NMS图。

5、双阈值标记

取NMS图中的最大值记为max，高阈值$high_threshold=\alpha\times max$，**$\alpha$为一(0,1)之间的实数，为超参数。**

低阈值$low_threshold=high_threshold\times\beta$，**$\beta$为一(0,1)之间的实数，为超参数。**

NMS图中，大于高阈值的标记为强边，在两个阈值之间的标记为弱边，低于低阈值的像素舍弃。

6、迟滞链接

先把结果图置为全黑，然后上一步标记为强边的像素置为白色。然后：

1、凡是在以强边像素为中心3x3范围内的弱边都标记为强边。

2、反复执行步骤1直到没有更多的弱边像素变成强边。即算法收敛。

所有强边置为白色，剩下所有弱边和本来就被舍弃了的边在这里都要被舍弃，即保持黑色。

这样结果图就是原图的轮廓图。

至此，简化版轮廓检测算子——Canny算子的介绍到此为止。

Task EX任务目标为，对`lena.jpg` `blood.jpg`两张图用Canny算子进行处理，调整算法中出现的各个超参数，比较不同超参数大小对输出轮廓的影响。

四、实验报告要求

按任务点分为六或七个部分（取决于你做没做附加题）

每个部分要求：

1、展示和任务点相关的算法的核心部分的代码（不允许全部复制粘贴，会扣报告分）并做文字说明实现思路

2、展示用代码处理之后的图像

3、如果任务点要求调整参数以观察参数对图像造成的影响，则需要用你的语言说明参数的变化对图像造成的影响（每个人对图像的主观审美是不一样的，所以这个部分可以大胆写，不设标准答案，但不能不写。）