> 本节为**opencv数字图像处理**（7）：**频率域滤波**的第四小节，**频率域滤波的基础公式、步骤与C++实现**，频率域滤波的前三小节（频率域滤波的基础概念、取样和取样函数的傅立叶变换以及二维取样定理与二维傅里叶变换），更偏向于数学概念，围绕傅立叶变换对展开，包括一维、二维、连续、离散傅立叶变换对机器性质以及此基础上得到的采样定理。**从本节开始，会更偏向于如何使用频率域滤波来处理图像**，本小节的主要内容包括：频率域滤波的基础公式与步骤，以及用C++代码实现的完整过程。

1. 频率域滤波基础

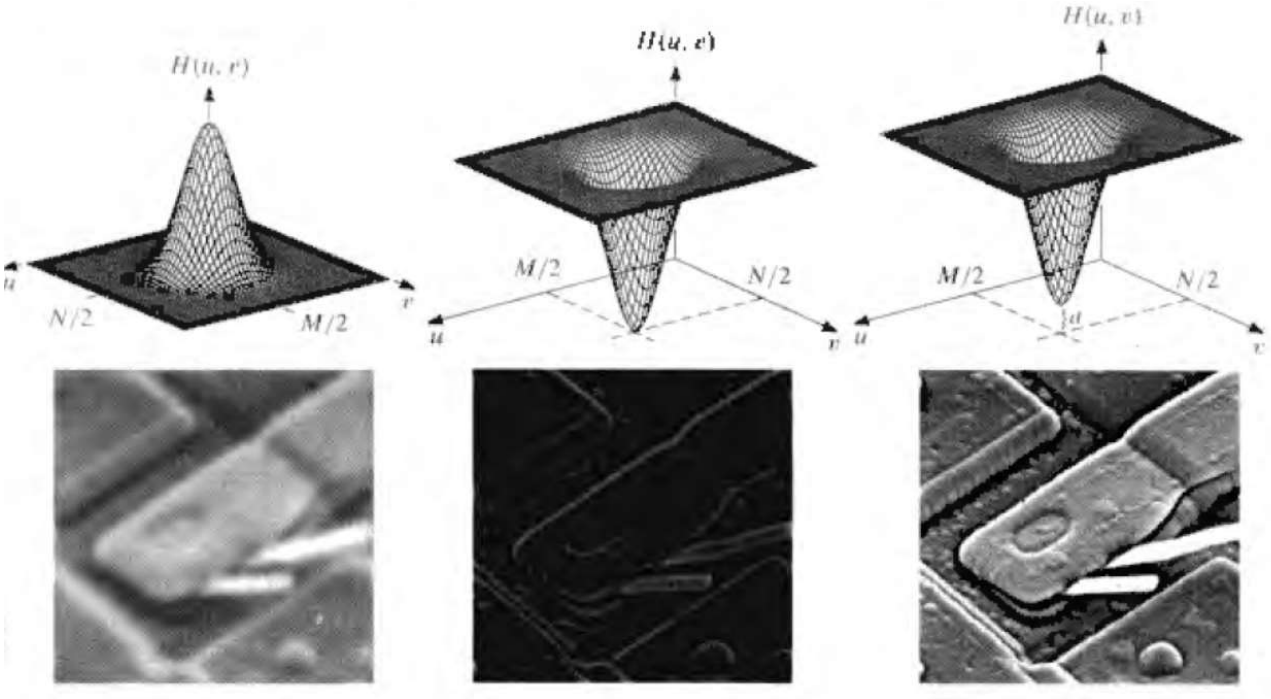
频域率滤波由修改一幅图像的傅里叶变换然后计算其反变换得到处理后的结果组成，给定一幅大小为$M \times N$的数字图像$f(x,y)$，则基本滤波公式如下：

$$g(x,y) = \zeta^{-1}[H(u,v)F(u,v)]$$

其中，ζ^{-1}时IDFT，$F(u,v)$是输入图像$f(x,y)$的DFT，$H(u,v)$是滤波函数（滤波器/滤波传递函数），$g(x,y)$是滤波后输出的图像，并且函数F 、 H 、 g都是大小与输入图像相同的$M \times N$阵列【使用中心堆成的函数可以显著简化$H(u,v)$的技术条件，这要求$F(u,v)$也被中心化，可以在变换前使用$(-1)^{x+y}$乘以输入图像来实现】。通常情况下，H是实对称函数而f是是函数，则上式中的IDFT理论上生成实数量，但实际中该反变换通常包含由舍入误差和其他计算错误引起的寄生复数像，因此，通常取IDFT的实部来形成函数g。

考虑一个最简单的滤波器，它在变换的中心处是0，其他处为1。当建立乘积$H(u,v)F(u,v)$时，滤波器将抑制$F(u,v)$的直流项而通过所有其他项。因为直流项决定图像的平均灰度，所以，将其置为0会把图像的平均灰度减小为0，图像变得更暗，因为均值为0灰度那么必存在负灰度，通常为显示目的将负灰度修剪为0。

图像经傅里叶变换至频率域，其中的低频与图像中缓慢变换的灰度分量有关，而高频由灰度的尖锐过渡造成。基于此，我们认为衰减高频而通过低频的低通滤波器$H(u,v)$将模糊一幅图像，具有相反特性的高通滤波器将增强尖锐的细节同时造成图像对比度的降低。如下图所示，第一行是从左到右为低通滤波器，高通滤波器和调整后的高通滤波器（对滤波器加上一个小常数，不影响尖锐性的同时，防止直流项的消除以保留色调），第二行是效果图。

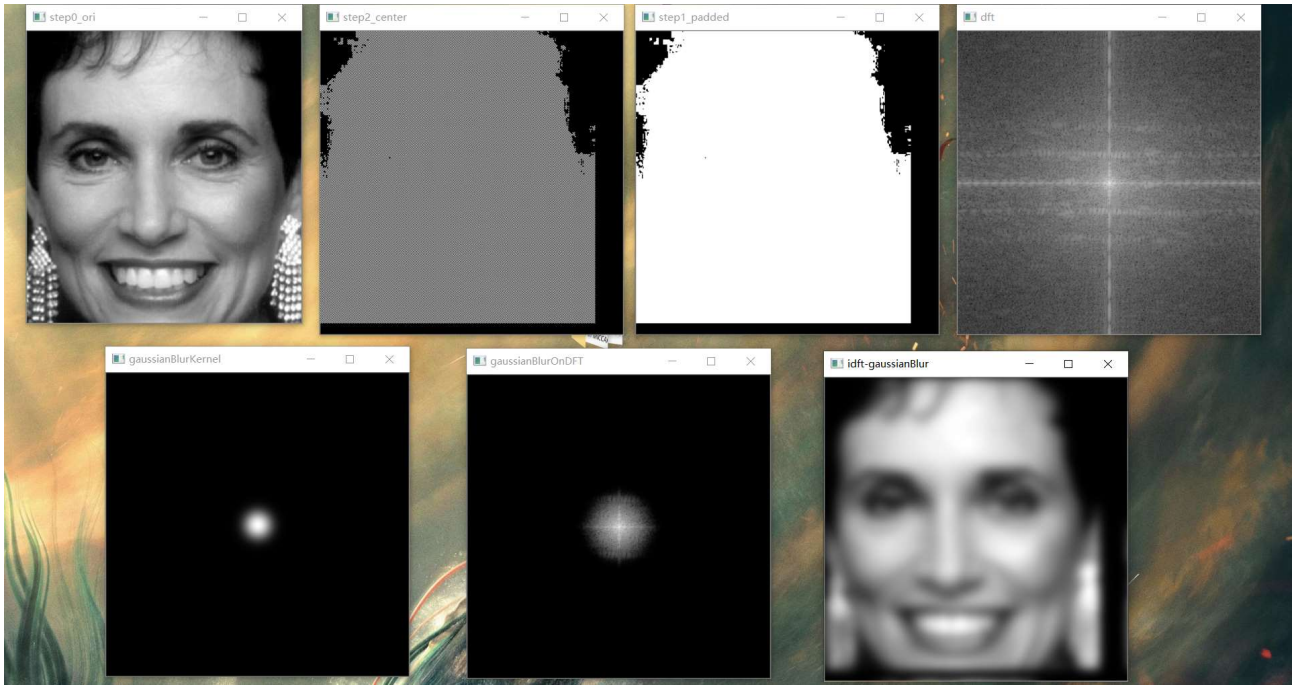


2. 频率域滤波步骤

频率域滤波分为以下七步：

- 给定一幅大小为$M \times N$的输入图像$f(x,y)$，设置填充参数$P=2M, Q=2N$；
- 对$f(x,y)$添加必要数量的0，形成大小为$P \times Q$的填充后的图像$f_p(x,y)$；
- 用$(-1)^{x+y}$乘以$f_p(x,y)$移到其变换的中心；
- 计算上一步骤得到的图像的DFT，得到$F(u,v)$；
- 生成一个实对称的滤波函数$H(u,v)$，其大小为$P \times Q$，中心在$(P/2, Q/2)$处，用阵列相乘形成乘积$G(u,v)=H(u,v)F(u,v)$，即$G(i,k)=H(i,k)F(i,k)$；
- 得到处理后的图像：$g_p(x,y)=\{real[\zeta^{-1}[G(u,v)]]\}(-1)^{x+y}$，其中，忽略计算不准确导致的寄生复分量，选择实部，下标p指出处理的是填充的阵列
- 通过从$g(x,y)$的左上象限提取$M \times N$区域，得到最终处理结果$g(x,y)$

示意图和代码如下所示：



C++实现：

```
#if 1
#include "opencv2/opencv.hpp"

int main()
{
 cv::Mat input = cv::imread("1.JPG", cv::IMREAD_GRAYSCALE);
 cv::imshow("step0_ori", input);
 int w = cv::getOptimalDFTSize(input.cols);
 int h = cv::getOptimalDFTSize(input.rows);
 cv::Mat padded;
 cv::copyMakeBorder(input, padded, 0, h - input.rows, 0, w - input.cols,
  cv::BORDER_CONSTANT, cv::Scalar::all(0));
 padded.convertTo(padded, CV_32FC1);
 cv::imshow("step1_padded", padded);
 for (int i = 0; i < padded.rows; i++)
 {
  float* ptr = padded.ptr<float>(i);
  for (int j = 0; j < padded.cols; j++)
   ptr[j] *= pow(-1, i + j);
 }
 cv::imshow("step2_center", padded);
 cv::Mat plane[] = { padded,cv::Mat::zeros(padded.size(),CV_32F) };
 cv::Mat complexImg;
 cv::merge(plane, 2, complexImg);
 cv::dft(complexImg, complexImg);
 cv::split(complexImg, plane);
```

```
    cv::magnitude(plane[0], plane[1], plane[0]);
    plane[0] += cv::Scalar::all(1);
    cv::log(plane[0], plane[0]);
    cv::normalize(plane[0], plane[0], 1, 0, cv::NORM_MINMAX);
    cv::imshow("dft", plane[0]);

    cv::Mat gaussianBlur(padded.size(), CV_32FC2);
    float D0 = 2 * 10 * 10;
    for (int i = 0; i < padded.rows; i++)
    {
     float* p = gaussianBlur.ptr<float>(i);
     for (int j = 0; j < padded.cols; j++)
     {
      float d = pow(i - padded.rows / 2, 2) + pow(j - padded.cols / 2, 2);
      p[2 * j] = expf(-d / D0);
      p[2 * j + 1] = expf(-d / D0);
     }
    }
    cv::split(gaussianBlur, plane);
    cv::magnitude(plane[0], plane[1], plane[0]);
    plane[0] += cv::Scalar::all(1);
    cv::log(plane[0], plane[0]);
    cv::normalize(plane[0], plane[0], 1, 0, cv::NORM_MINMAX);
    cv::imshow("gaussianBlurKernel", plane[0]);

    multiply(complexImg, gaussianBlur, gaussianBlur);
    cv::split(gaussianBlur, plane);
    cv::magnitude(plane[0], plane[1], plane[0]);
    plane[0] += cv::Scalar::all(1);
    cv::log(plane[0], plane[0]);
    cv::normalize(plane[0], plane[0], 1, 0, cv::NORM_MINMAX);
    cv::imshow("gaussianBlurOnDFT", plane[0]);

    cv::idft(gaussianBlur, gaussianBlur);
    cv::split(gaussianBlur, plane);
    cv::magnitude(plane[0], plane[1], plane[0]);
    cv::normalize(plane[0], plane[0], 1, 0, cv::NORM_MINMAX);
    cv::imshow("idft-gaussianBlur", plane[0]);

    cv::waitKey();
    return 0;
}
#endif
```

欢迎扫描二维码关注微信公众号 深度学习与数学 [每天获取免费的大数据、AI等相关的学习资源、经典和最新的深度学习相关的论文研读，算法和其他互联网技能的学习，概率论、线性代数等高等数学知识的回顾]