湖南科技大学课程教案
（章节、专题首页）

授课教师：王志喜　　职称：副教授　　单位：计算机学院

课程名称	计算机图形图像技术
章节、专题	图像的空间转换
教学目标及基本要求	掌握对图像进行空间转换的基本方法，熟悉相应的OpenCV函数，获得对图像进行空间转的程序设计能力。
教学重点	离散傅立叶变换
教学难点	离散傅立叶变换
教学内容与时间分配	（1）颜色空间转换（0.7课时） （2）离散傅立叶变换（2.5课时） （3）离散余弦变换（0.8课时） 共计4课时。
习　　题	第14.5.1节（基础知识题）和14.5.2节（程序设计题）。

第14章　图像的空间转换

14.1　空间转换的常见种类

图像的空间转换主要包括空域与空域的转换和空域与频域的转换等两类。其中，空域与空域的转换常见的是颜色空间转换，使用线性变换实现，而空域与频域的转换通常使用离散傅立叶变换或离散余弦变换等方法实现。

14.2　颜色空间转换♫Ж

14.2.1　颜色空间转换的基本方法

1. RGB颜色与灰度级的转换

（1）RGB颜色转换为灰度级。$Y=0.299R+0.587G+0.114B$。

（2）灰度级转换为RGB颜色。$R=G=B=Y$。

2. RGB颜色与CIE XYZ颜色的转换

CIE XYZ颜色模型使用三种假想的标准基色，用X、Y和Z表示产生一种颜色所需要的CIE基色的量。因此，在XYZ模型中描述一种颜色的方式与RGB模型类似。

（1）RGB颜色转换为XYZ颜色。

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.412 & 0.358 & 0.180 \\ 0.213 & 0.715 & 0.072 \\ 0.019 & 0.119 & 0.950 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

（2）XYZ颜色转换为RGB颜色。

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 3.240 & -1.537 & -0.499 \\ -0.969 & 1.876 & 0.042 \\ 0.056 & -0.204 & 1.057 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

3. RGB颜色与YCrCb颜色的转换

YCrCb是JPEG采用的颜色模型。YCrCb颜色空间的一个重要特性是亮度信号Y和色差信号C_r、C_b是分离的。亮度Y=0.299R+0.587G+0.114B，色差C_r和C_b分别由R-Y和B-Y按照不同比例压缩得到。

（1）RGB颜色转换为YCrCb颜色。

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ C_r = 0.713(R - Y) + \delta \\ C_b = 0.564(B - Y) + \delta \end{cases}$$

（2）YCrCb颜色转换为RGB颜色。

$$\begin{cases} R = Y + 1.403(C_r - \delta) \\ G = Y - 0.344(C_r - \delta) - 0.714(C_b - \delta) \\ B = Y + 1.773(C_b - \delta) \end{cases}$$

对于8位图像，δ=128，对于16位图像，δ=32768，对于浮点数图像，δ=0.5。

4. RGB颜色与HSV颜色的转换

HSV颜色模型使用色相H、饱和度S和色明度V表示一种颜色。其中色相H是一个角度，从0度到360度变化，红色对应0度，绿色对应120度，蓝色对应240度。饱和度S表示颜色的纯度，从0到1变化，纯色对应1，灰度颜色对应0。色明度V表示颜色的明亮程度，从0到1变化，最亮的颜色对应1，黑色对应0。

（1）RGB颜色转换为HSV颜色。假定RGB图像和HSV图像都是浮点数图像，各分量值从0到1变化。

$$V_0 = \min(R,G,B), V_1 = \max(R,G,B)$$

$$V = V_1$$

$$S = \begin{cases} (V - V_0)/V & V \neq 0 \\ 0 & V = 0 \end{cases}$$

$$H = \begin{cases} (1/6)(G-B)/S & V = R \\ 2/6 + (1/6)(B-R)/S & V = G \\ 4/6 + (1/6)(R-G)/S & V = B \end{cases}$$

$$H = \text{fract}(H+1)$$

（2）HSV颜色转换为RGB颜色。假定RGB图像和HSV图像都是浮点数图像，各分量值从0到1变化。若 $V=0$，则 $R=G=B=0$，否则，令 $V_1=V$，$V_0=(1-S)V$，考虑 H 的取值情况。

当 $0\le H<1/6$ 时，$R=V_1$，$B=V_0$，$G=V_0+6HS$。

当 $1/6\le H<2/6$ 时，$G=V_1$，$B=V_0$，$R=V_0-(6H-2)S$。

当 $2/6\le H<3/6$ 时，$G=V_1$，$R=V_0$，$B=V_0+(6H-2)S$。

当 $3/6\le H<4/6$ 时，$B=V_1$，$R=V_0$，$G=V_0-(6H-4)S$。

当 $4/6\le H<5/6$ 时，$B=V_1$，$G=V_0$，$R=V_0+(6H-4)S$。

当 $5/6\le H<1$ 时，$R=V_1$，$G=V_0$，$B=V_0-6HS$。

5. RGB颜色转换为HLS颜色

HLS颜色模型使用色相H、亮度L和饱和度S表示一种颜色。其中色相H是一个角度，从0度到360度变化，红色对应0度，绿色对应120度，蓝色对应240度。亮度L表示颜色的明亮程度，从0到1变化，白色对应1，黑色对应0，纯色对应0.5。饱和度S表示颜色的纯度，从0到1变化，纯色对应1，灰度颜色对应0。

（1）RGB颜色转换为HLS颜色。假定RGB图像和HLS图像都是浮点数图像，各分量值从0到1变化。

$$V_0 = \min(R,G,B), V_1 = \max(R,G,B)$$

$$L = (V_1 + V_0)/2$$

$$S = \begin{cases} (V_1 - V_0)/(2L) & L < 0.5 \\ (V_1 - V_0)/(2 - 2L) & L \geq 0.5 \end{cases}$$

$$H = \begin{cases} (1/6)(G-B)/S & V_1 = R \\ 2/6 + (1/6)(B-R)/S & V_1 = G \\ 4/6 + (1/6)(R-G)/S & V_1 = B \end{cases}$$

$$H = \text{fract}(H+1)$$

（2）HLS颜色转换为RGB颜色。假定RGB图像和HLS图像都是浮点数图像，各分量值从0到1变化。

当 $L=0$ 或 $L=1$ 时，$R=G=B=L$。

当 $0<L<1$ 时，若 $L<0.5$，则令 $V_1=(S+1)L$，$V_0=2L-V_1$，否则，令 $V_1=L+S-LS$，$V_0=2L-V_1$，考虑 H 的取值情况。

当 $0\le H<1/6$ 时，$R=V_1$，$B=V_0$，$G=V_0+6HS$。

当 $1/6\le H<2/6$ 时，$G=V_1$，$B=V_0$，$R=V_0-(6H-2)S$。

当 $2/6\le H<3/6$ 时，$G=V_1$，$R=V_0$，$B=V_0+(6H-2)S$。

当 $3/6\le H<4/6$ 时，$B=V_1$，$R=V_0$，$G=V_0-(6H-4)S$。

当 $4/6\le H<5/6$ 时，$B=V_1$，$G=V_0$，$R=V_0+(6H-4)S$。

当 $5/6\le H<1$ 时，$R=V_1$，$G=V_0$，$B=V_0-6HS$。

12.4.2 OpenCV中的相关函数

OpenCV使用cvtColor()函数进行颜色空间转换。

【函数原型】void cvtColor(InputArray src, OutputArray dst, int code);

【功能】输入图像从一个颜色空间转换为另外一个颜色空间。

【参数】

- src和dst：输入数组和输出数组，8位、16位或单精度数图像。
- code：标志颜色转换操作的常数，通常选用
  - COLOR_BGR2GRAY、COLOR_GRAY2BGR（BGR颜色与灰度级的转换）
  - COLOR_BGR2XYZ、COLOR_XYZ2BGR（BGR颜色与XYZ颜色的转换）
  - COLOR_BGR2YCrCb、COLOR_YCrCb2BGR（BGR颜色与YCrCb颜色的转换）
  - COLOR_BGR2HSV、COLOR_HSV2BGR（BGR颜色与HSV颜色的转换）
  - COLOR_BGR2HLS、COLOR_HLS2BGR（BGR颜色与HLS颜色的转换）

12.4.3 应用举例

下述例子演示了一种比较简陋的将灰度图像转换成彩色图像的方法，程序运行结果如图14-1所示。

图14-1 颜色空间转换示例

```
// CvtColor.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("lena.jpg", 0); // 装入灰度图像
    if(X.empty()) return -1; // 装入失败
    imshow("Source", X); // 显示源图像
    // 源图像转换为3通道浮点数图像
    cvtColor(X, X, COLOR_GRAY2BGR); // 3通道字节图像
    X = Mat3f(X) / 255; // 3通道浮点数图像, 像素值 in [0, 1]
    Mat Y, HSV, HLS; // 初始结果图像, HSV图像, HLS图像
    cvtColor(X, Y, COLOR_HLS2BGR); // 较真(初始结果)
    cvtColor(X, HSV, COLOR_BGR2HSV); // 偏红(修正)
    cvtColor(X, HLS, COLOR_BGR2HLS); // 偏绿(修正)
    // 结果=较真×0.75+偏红×0.1+偏绿×0.15
    Y = 0.75 * Y + 0.1 * HSV + 0.15 * HLS;
    Y /= norm(Y, NORM_INF); // 增强亮度和对比度
    imshow("Result", Y); // 显示结果
```

```
    waitKey();
}
```

14.3 离散傅立叶变换

傅立叶变换与前述各种以像素位置或像素值为处理对象的图像处理技术不同，它以图像中灰度的变化频率为处理对象，是一种重要的图像处理技术。

在图像处理中使用傅立叶变换的思想来源于阿贝二次成像理论。该理论表明，物函数的一次傅立叶变换（正变换）反映了物函数在系统频谱面上的频率分布。如果在频谱面上做某些处理（如滤波），再做第二次傅立叶变换（逆变换），就能改变物函数的某些特征，以达到人们要求的结果。图14-2给出了一个阿贝二次成像的例子，该例子在频谱面上只保留了高频成分（可以理解为强调了突变），达到了突出边缘的目的。

图14-2　阿贝二次成像的例子

14.3.1 复数的表示形式及转换

为了更好地理解傅立叶变换，这里回顾一下复数的直角坐标形式表示和极坐标形式表示。

1. 两种表示形式

（1）直角坐标形式。复数的直角坐标形式表示为 (x,y)，其中，x 和 y 都是一个实数，分别称为实部和虚部。在引入虚数单位 $i=\sqrt{-1}$ 后，复数的直角坐标形式 (x,y) 可以表示为加法形式 $x+yi$。

（2）极坐标形式。复数的极坐标形式表示为 (ρ,θ)，其中，ρ 是一个非负实数，θ 通常选用 $0\sim2\pi$ 中的实数。在数字图像处理中，通常将这里的 ρ 和 θ 分别称为振幅和相位。在引入虚数单位 $i=\sqrt{-1}$ 后，复数的极坐标形式 (ρ,θ) 可以表示为指数形式 $\rho e^{i\theta}$。

2. 两种表示形式之间的转换

（1）直角坐标形式转换为极坐标形式。由直角坐标形式 (x,y) 计算极坐标形式 (ρ,θ) 的方法是 $\big(\rho,\ \theta\big)=\Big(\sqrt{x^2+y^2},\ \arctan(y,x)\Big)$。其中，$\theta$ 的取值范围是 $0\sim 2\pi$。

（2）极坐标形式转换为直角坐标形式。由极坐标形式 (ρ,θ) 计算直角坐标形式 (x,y) 的方法是 $(x,\ y)=(\rho\cos\theta,\ \rho\sin\theta)$。

（3）复数的绝对值。复数 $z=x+yi$ 的绝对值定义为 $\big|z\big|=\sqrt{x^2+y^2}$。显然，$\Big|\rho e^{i\theta}\Big|=\rho$。

3. 共轭复数

复数 $z=x+yi$ 的共轭复数定义为 $\overline{z}=\mathrm{conj}(z)=x-yi$。容易证明，共轭复数满足下列性质。

（1）复数 $\rho e^{i\theta}$ 的共轭复数为 $\rho e^{-i\theta}$。

（2）复数 z 满足 $\overline{z}=z$ 当且仅当 z 是一个实数。

（3）任何复数 z 和 w 都满足 $\mathrm{conj}(z\odot w)=\overline{z}\odot\overline{w}$，其中 $\odot\in\{+,-,\times,\div\}$。

（4）若 $|z|=1$，则 $\overline{z}=z^{-1}$，且 $\mathrm{conj}(z^k)=z^{-k}=\overline{z}^k$。

4. OpenCV中的相关函数

（1）转换为极坐标形式。使用cartToPolar()。

【函数原型】void cartToPolar(InputArray x, InputArray y, OutputArray magnitude, OutputArray angle, bool angleInDegrees = false);

【功能】直角坐标形式转换为对应的极坐标形式。

【参数】

- x, y：分别是实部数组和虚部数组，必须是浮点数数组，且大小和类型必须一致。
- magnitude，angle：分别是振幅数组和相位数组，大小和类型与源数组一致，必要时重建。
- angleInDegrees：指定相位的单位，false表示弧度（范围是0～2π），true表示度（范围是0~360度）。

（2）转换为直角坐标形式。使用polarToCart()。

【函数原型】void polarToCart(InputArray magnitude, InputArray angle, OutputArray x, OutputArray y, bool angleInDegrees = false);

【功能】极坐标形式转换为对应的直角坐标形式。

【参数】

- magnitude, angle：分别是振幅数组和相位数组，必须是浮点数数组，且大小和类型必须一致。
- *x*, *y*：分别是实部数组和虚部数组，大小和类型与源数组一致，必要时重建。
- angleInDegrees：指定相位的单位，false表示弧度（范围是0～2π），true表示度（范围是0~360度）。

（3）振幅和相位。使用magnitude()和phase()。

【函数原型】

- void magnitude(InputArray x, InputArray y, OutputArray magnitude);
- void phase(InputArray x, InputArray y, OutputArray angle, bool angleInDegrees = false);

【功能】

- magnitude()计算直角坐标形式对应的振幅。
- phase()计算直角坐标形式对应的相位。

【参数】

- x, y：分别是实部数组和虚部数组，必须是浮点数数组，且大小和类型必须一致。
- magnitude, angle：分别是振幅数组和相位数组，大小和类型与源数组一致，必要时重建。
- angleInDegrees：指定相位的单位，false表示弧度（范围是$0 \sim 2\pi$），true表示度（范围是0~360度）。

14.3.2 离散傅立叶变换

因为数字图像是离散函数，所以这里只考虑离散傅立叶变换。

1. 表达式

离散傅立叶变换的表达式为

$$F(u)=\sum_{x=0}^{N-1} f(x)(e^{-2\pi i/N})^{ux}\ ,\quad u\in Z$$

离散傅立叶变换的逆变换为

$$f(x)=\frac{1}{N}\sum_{u=0}^{N-1} F(u)(e^{2\pi i/N})^{ux}\ ,\quad x\in Z$$

【注】

- $i=\sqrt{-1}$ 是虚数单位。
- 在数字图像处理中，通常规定 $u=0,1,...,N-1$，$x=0,1,...,N-1$。
- $1/N$ 可用于正变换或逆变换，在OpenCV中通常用于逆变换。

2. 单位根

给定复数 ω 和正整数 n，若 n 是使得 $\omega^n = 1$ 的最小正整数，则称 ω 为一个 n 次单位根。显然，$\omega = e^{2\pi i/n}$ 和 $\omega = e^{-2\pi i/n}$ 都是一个 n 次单位根。

n 次单位根 ω 有下列性质。

（1）周期性。序列 $\ldots, \omega^{-2}, \omega^{-1}, \omega^0, \omega^1, \omega^2, \ldots$ 的周期是 n。

（2）不重复性。$\omega^0, \omega^1, \ldots, \omega^{n-1}$ 各不相同。

（3）共轭对称性。$\omega^{n-k} = \mathrm{conj}(\omega^k)$，$k$ 是整数。

（4）共轭根。$\overline{\omega} = \mathrm{conj}(\omega)$ 也是一个 n 次单位根。

（5）相反性。若 n 是偶数，则 $\omega^{n/2} = -1$，从而 $\omega^{k+n/2} = -\omega^k$。

（6）低次单位根。若正整数 k 能够整除 n，则 ω^k 是一个 n/k 次单位根。

【证明#】

（1）周期性。$\omega^{k+n} = \omega^k \omega^n = \omega^k$。

（2）不重复性。若存在 $0 \le j < k < n$，使得 $\omega^j = \omega^k$，则 $\omega^{k-j} = \omega^k / \omega^j = 1$，与 ω 是一个 n 次单位根矛盾。

（3）共轭对称性。$\omega^{n-k}=\omega^{-k}=\text{conj}(\omega^k)$。

（4）共轭根。显然，$\overline{\omega}^n=1$。若存在$0\le m<n$，使得$\overline{\omega}^m=1$，则$\omega^{n-m}=\omega^{-m}=\overline{\omega}^m=1$，与$\omega$是一个$n$次单位根矛盾。

（5）相反性。由单位根的共轭对称性可知，当n是偶数时，$\text{conj}(\omega^{n/2})=\omega^{n/2}$，所以$\omega^{n/2}$是实数。由$\left|\omega^{n/2}\right|=1$可知，$\omega^{n/2}$只可能是-1或1。因为$\omega$是一个$n$次单位根，所以$\omega^{n/2}=-1$。

（6）低次单位根。若存在$0\le m<n/k$，使得$\omega^{km}=1$，则由$km<n$可知，该假设与ω是一个n次单位根相矛盾。

（3）共轭对称性。$\omega^{n-k}=\text{conj}(\omega^k)$，$k$是整数。

（4）共轭根。$\overline{\omega}=\text{conj}(\omega)$也是一个$n$次单位根。

（5）相反性。若n是偶数，则$\omega^{n/2}=-1$，从而$\omega^{k+n/2}=-\omega^k$。

（6）低次单位根。若k是一个能够整除n的正整数，则ω^k是一个n/k次单位根。

$$F(u)=\sum_{x=0}^{N-1} f(x)(e^{-2\pi i/N})^{ux}$$

3. 离散傅立叶变换的性质

为了将离散傅立叶变换的正变换和逆变换统一考虑，这里讨论变换 $F(u)=\sum_{x=0}^{N-1} f(x)\omega^{ux}$，其中 $u\in Z$，ω 是一个 N 次单位根。显然，若 $\omega=e^{-2\pi i/N}$，则 $F(u)$ 为离散傅立叶正变换，若 $\omega=e^{2\pi i/N}$，则 $F(u)/N$ 为离散傅立叶逆变换。

变换 $F(u)=\sum_{x=0}^{N-1} f(x)\omega^{ux}$ 具有下列性质。

（1）周期性。$F(u+N)=F(u)$。

（2）平移性。若 $g(x)=f(x-x_0)$，则 $G(u)=\omega^{ux_0}F(u)$。

（3）线性组合性。若 $g(x)=\alpha f_1(x)+\beta f_2(x)$，则 $G(u)=\alpha F_1(u)+\beta F_2(u)$。

（4）平均值。$\overline{f(x)}=F(0)/N$。

（5）共轭对称性。如果 $f(x)$ 是实数函数，则 $F(N-u)=\mathrm{conj}(F(u))$。

（6）实数性。当 $f(x)$ 是实数函数时，$F(0)$ 是实数；而且，若 N 是偶数，则 $F(N/2)$ 是实数。

$$f(x)=\frac{1}{N}\sum_{u=0}^{N-1} F(u)(e^{2\pi i/N})^{ux}$$

【证明#】

（1）周期性。

$$F(u+N)=\sum_{x=0}^{N-1}f(x)\omega^{(u+N)x}=\sum_{x=0}^{N-1}f(x)(\omega^{u+N})^{x}$$
$$=\sum_{x=0}^{N-1}f(x)\omega^{ux}=F(u)$$

（2）平移性。

$$G(u)=\sum_{x=0}^{N-1}f(x-x_0)\omega^{ux}=\sum_{x=0}^{N-1}f(x)\omega^{u(x+x_0)}$$
$$=\omega^{ux_0}\sum_{x=0}^{N-1}f(x)\omega^{ux}=\omega^{ux_0}F(u)$$

（3）线性组合性。

$$G(u)=\sum_{x=0}^{N-1}(\alpha f_1(x)+\beta f_2(x))\omega^{ux}$$
$$=\alpha\sum_{x=0}^{N-1}f_1(x)\omega^{ux}+\beta\sum_{x=0}^{N-1}f_2(x)\omega^{ux}=\alpha F_1(u)+\beta F_2(u)$$

（1）周期性。$F(u+N)=F(u)$。

（2）平移性。若$g(x)=f(x-x_0)$，则$G(u)=\omega^{ux_0}F(u)$。

（3）线性组合性。若$g(x)=\alpha f_1(x)+\beta f_2(x)$，则$G(u)=\alpha F_1(u)+\beta F_2(u)$。

（4）平均值。

$$\frac{1}{N}F(0)=\frac{1}{N}\sum_{x=0}^{N-1}f(x)\omega^{0x}=\frac{1}{N}\sum_{x=0}^{N-1}f(x)=\overline{f(x)}$$

（5）共轭对称性。

$$\begin{aligned}F(N-u)&=\sum_{x=0}^{N-1}f(x)\omega^{(N-u)x}=\sum_{x=0}^{N-1}f(x)\omega^{Nx-ux}\\&=\sum_{x=0}^{N-1}f(x)\mathrm{conj}(\omega^{ux})=\sum_{x=0}^{N-1}\mathrm{conj}(f(x)\omega^{ux})\\&=\mathrm{conj}(\sum_{x=0}^{N-1}f(x)\omega^{ux})=\mathrm{conj}(F(u))\end{aligned}$$

（6）实数性。由 $F(0)=N\overline{f(x)}$ 可知，$F(0)$ 是实数。由 $\mathrm{conj}(F(N/2))=F(N-N/2)=F(N/2)$ 可知，$F(N/2)$ 是实数。

（4）平均值。$\overline{f(x)}=F(0)/N$。

（5）共轭对称性。如果 $f(x)$ 是实数函数，则 $F(N-u)=\mathrm{conj}(F(u))$。

（6）实数性。当 $f(x)$ 是实数函数时，$F(0)$ 是实数；而且，若 N 是偶数，则 $F(N/2)$ 是实数。

4. 快速傅立叶变换

（1）直接计算的耗时。容易说明，若直接使用 $F(u)=\sum_{x=0}^{N-1} f(x)\omega^{ux}$ 计算 $F(0),F(1),\ldots,F(N-1)$，则耗时正比于 N^2（参见下列参考程序）。

```
template<class T, class T2> // 一维DFT(元素类型, 单位根类型)
auto MyDFT(Mat_<T> f, T2 w) // 原函数, 单位根
{   int n = f.total();
    T ws[n]; // 保存w^i, 共n个元素
    ws[0] = 1; // w^0
    for(int i = 1; i < n; ++i)
        ws[i] = (T)w * ws[i - 1]; // w^i
    Mat_<T> g(f.size(), 0); // 结果函数, 初始值为0
    for(int u = 0; u < n; ++u) // g(u) = sum{ f(x) * w^(ux) }
        for(int x = 0; x < n; ++x)
            g(u) += f(x) * ws[u * x % n];
    return g;
}
```

这里介绍一种耗时正比于 $N\log N$ 的快速方法。

（2）快速算法推导。假定 N 是2的正整数幂， $M = N/2$ 。

$$\begin{aligned} F(u) &= \sum_{x=0}^{N-1} f(x)\omega^{ux} \\ &= \sum_{x=0}^{M-1} f(2x)\omega^{u(2x)} + \sum_{x=0}^{M-1} f(2x+1)\omega^{u(2x+1)} \\ &= \sum_{x=0}^{M-1} f(2x)(\omega^2)^{ux} + \omega^u \sum_{x=0}^{M-1} f(2x+1)(\omega^2)^{ux} \end{aligned}$$

设 $f_0(x) = f(2x)$ ， $f_1(x) = f(2x+1)$ ， $\widehat{\omega} = \omega^2$ ， $F_0(u) = \sum_{x=0}^{M-1} f_0(x)\widehat{\omega}^{ux}$ ，

$F_1(u) = \sum_{x=0}^{M-1} f_1(x)\widehat{\omega}^{ux}$ ，则

$$\begin{aligned} F(u) &= F_0(u) + \omega^u F_1(u) \\ F(u+M) &= F_0(u+M) + \omega^{u+M} F_1(u+M) \\ &= F_0(u) - \omega^u F_1(u) \end{aligned}$$

$$F(u)=\sum_{x=0}^{N-1} f(x)\omega^{ux}$$

（3）快速算法的实现。可以使用下列参考程序实现。

```
template<class T, class T2> // 一维DFT(元素类型, 单位根类型)
auto MyFFT(Mat_<T> f, T2 w) // 原函数, 单位根
{   int n = f.total();
    if(n <= 1) return f.clone(); // 只有一个元素时不会变化
    int m = n / 2;
    Mat_<T> f0(m, 1), f1(m, 1); // f(2x)和f(2x+1)的值
    for(int x = 0; x < m; ++x)
        f0(x) = f(2 * x), f1(x) = f(2 * x + 1);
    auto g0 = MyFFT(f0, w * w), g1 = MyFFT(f1, w * w);
    Mat_<T> g(f.size()); // 结果函数
    T wu = 1; // w^u, u=0
    for(int u = 0; u < m; ++u)
    {   g(u) = g0(u) + wu * g1(u);
        g(u + m) = g0(u) - wu * g1(u);
        wu *= w; // w^(u+1)=(w^u)*w
    }
    return g;
}
```

$$F(u)=F_0(u)+w^u F_1(u)$$
$$F(u+M)=F_0(u)-w^u F_1(u)$$

（4）快速算法的耗时。设 $T(N)$ 是当输入规模为 N（假定 $N=2^m$）时的耗时，则存在 $C_1>0$，使得 $T(N)\le 2T(N/2)+C_1N$。

$$\begin{aligned}T(2^m)&\le 2T(2^{m-1})+C_1N\\&\le 2(2T(2^{m-2})+C_1N/2)+C_1N\\&\le 2^2T(2^{m-2})+2C_1N\\&\cdots\\&\le 2^{m-1}T(1)+(m-1)C_1N\end{aligned}$$

选取 $C\ge\max(T(1),C_1)$，则

$$\begin{aligned}T(N)&=T(2^m)\\&\le 2^{m-1}C+(m-1)CN\\&\le CN+(m-1)CN\\&=mCN\\&=CN\log N\end{aligned}$$

（5）最佳DFT大小。对于 $f(x)$ 为实数函数的情况，如果 N 不是2的正整数幂，则最佳DFT大小为 $N_e = 2^{\lceil \log_2 N \rceil}$。此时，对于正变换，可将 $f(x)$ 扩充为

$$f_e(x) = \begin{cases} f(x) & 0 \le x < N \\ 0 & \text{else} \end{cases}$$

对于逆变换，可将 $F(u)$ 扩充为

$$F_e(u) = \begin{cases} F(u) & 0 \le u < N \\ \text{conj}(F(M-u)) & \text{else} \end{cases}$$

14.3.3 二维离散傅立叶变换

因为数字图像是二维离散函数，所以需要考虑二维离散傅立叶变换。

1. 表达式

二维离散傅立叶变换的表达式为

$$F(u,v)=\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)\left(e^{-2\pi i/M}\right)^{ux}\left(e^{-2\pi i/N}\right)^{vy}\ ,\quad u,v\in Z$$

二维离散傅立叶变换的逆变换为

$$f(x,y)=\frac{1}{MN}\sum_{v=0}^{N-1}\sum_{u=0}^{M-1}F(u,v)\left(e^{2\pi i/M}\right)^{ux}\left(e^{2\pi i/N}\right)^{vy}\ ,\quad x,y\in Z$$

【注】在数字图像处理中，通常规定 $u=0,1,...,M-1$， $v=0,1,...,N-1$，$x=0,1,...,M-1$， $y=0,1,...,N-1$。

2. 二维离散傅立叶变换的性质

如果 ω_x 是一个 M 次单位根，ω_y 是一个 N 次单位根，那么变换 $F(u,v)=\sum_{y=0}^{N-1}\sum_{x=0}^{M-1} f(x,y)\omega_x^{ux}\omega_y^{vy}$ 具有下列性质。

（1）周期性。$F(u+M,v)=F(u,v+N)=F(u+M,v+N)=F(u,v)$。

（2）平移性。若 $g(x,y)=f(x-x_0,y-y_0)$，则 $G(u,v)=\omega_x^{ux_0}\omega_y^{vy_0}F(u,v)$。

（3）线性组合性。若 $g(x,y)=\alpha f_1(x,y)+\beta f_2(x,y)$，则 $G(u)=\alpha F_1(u,v)+\beta F_2(u,v)$。

（4）平均值。$\overline{f(x,y)}=F(0,0)/(MN)$。

（5）共轭对称性。如果 $f(x,y)$ 是实数函数，则 $F(M-u,N-v)=\mathrm{conj}(F(u,v))$。

（6）实数性。当 $f(x,y)$ 是实数函数时，$F(0,0)$ 是实数；而且，若 M 是偶数，则 $F(M/2,0)$ 是实数；若 N 是偶数，则 $F(0,N/2)$ 是实数；若 M 和 N 都是偶数，则 $F(M/2,N/2)$ 是实数。

【证明#】

（1）周期性。只写出 $F(u,v+N)=F(u,v)$ 的证明，其余可类似写出。

$$F(u,v+N)=\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)\omega_x^{ux}\omega_y^{(v+N)y}$$
$$=\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)\omega_x^{ux}\omega_y^{vy}=F(u,v)$$

（2）平移性。

$$G(u,v)=\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x-x_0,y-y_0)\omega_x^{ux}\omega_y^{vy}$$
$$=\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)\omega_x^{u(x+x_0)}\omega_y^{v(y+y_0)}$$
$$=\omega_x^{ux_0}\omega_y^{vy_0}\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)\omega_x^{ux}\omega_y^{vy}=\omega_x^{ux_0}\omega_y^{vy_0}F(u,v)$$

（3）线性组合性。与一维变换的证明完全类似，不再赘述。

（1）周期性。$F(u+M,v)=F(u,v+N)$
$=F(u+M,v+N)=F(u,v)$。

（2）平移性。若 $g(x,y)=f(x-x_0,y-y_0)$，则 $G(u,v)=\omega_x^{ux_0}\omega_y^{vy_0}F(u,v)$。

（3）线性组合性。若 $g(x,y)=\alpha f_1(x,y)+\beta f_2(x,y)$，则 $G(u)=\alpha F_1(u,v)+\beta F_2(u,v)$。

（4）平均值。

$$\frac{1}{MN}F(0,0)=\frac{1}{MN}\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)\omega_x^{0x}\omega_y^{0y}$$
$$=\frac{1}{MN}\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)=\overline{f(x,y)}$$

（5）共轭对称性。

$$F(M-u,N-v)=\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)\omega_x^{(M-u)x}\omega_y^{(N-v)y}$$
$$=\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)\mathrm{conj}(\omega_x^{ux})\mathrm{conj}(\omega_y^{vy})$$
$$=\mathrm{conj}(\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)\omega_x^{ux}\omega_y^{vy})$$
$$=\mathrm{conj}(\overline{F(u,v)})$$

（6）实数性。由 $F(0,0)=MN\overline{f(x,y)}$ 可知，$F(0,0)$ 是实数。若 M 是偶数，则由 $\mathrm{conj}(F(M/2,0))=F(M-M/2,M-0)=F(M/2,0)$ 可知，$F(M/2,0)$ 是实数。其余类似，不再赘述。

（4）平均值。$\overline{f(x,y)}=F(0,0)/(MN)$。

（5）共轭对称性。如果 $f(x,y)$ 是实数函数，则 $F(M-u,N-v)=\mathrm{conj}(F(u,v))$。

（6）实数性。当 $f(x,y)$ 是实数函数时，$F(0,0)$ 是实数；而且，若 M 是偶数，则 $F(M/2,0)$ 是实数；若 N 是偶数，则 $F(0,N/2)$ 是实数；若 M 和 N 都是偶数，则 $F(M/2,N/2)$ 是实数。

3. 二维离散傅立叶变换的快速计算

（1）计算方法。实际上，二维离散傅立叶变换和二维离散傅立叶逆变换都可以通过两次相应的一维变换得到。

【证明】

$$F(u,v)=\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)\omega_x^{ux}\omega_y^{vy}=\sum_{y=0}^{N-1}\left[\sum_{x=0}^{M-1}f(x,y)\omega_x^{ux}\right]\omega_y^{vy}$$

设

$$g(u,y)=\sum_{x=0}^{M-1}f(x,y)\omega_x^{ux}$$

$$F(u)=\sum_{x=0}^{N-1}f(x)\omega^{ux}$$

则

$$F(u,v)=\sum_{y=0}^{N-1}g(u,y)\omega_y^{vy}$$

显然，若 $\omega_x=e^{-2\pi i/M}$，$\omega_y=e^{-2\pi i/N}$，则 $F(u,v)$ 为二维离散傅立叶正变换，若 $\omega_x=e^{2\pi i/M}$，$\omega_y=e^{2\pi i/N}$，则 $F(u,v)/MN$ 为二维离散傅立叶逆变换。

（2）参考实现。由上述证明可知，二维离散傅立叶变换和逆变换都可以由两个步骤完成，即首先对原数组逐行变换，然后对逐行变换的结果逐列变换。如果 M 和 N 都是2的正整数次幂，则二维离散傅立叶变换和逆变换都可以使用下列参考程序实现快速计算。

```
template<class T, class T2> // 二维DFT(元素类型, 单位根类型)
auto MyFFT(Mat_<T> f, T2 wx, T2 wy) // 原函数, 单位根(wx, wy)
{   Mat_<T> g(f.size()); // 保存逐行变换结果
    for(int y = 0; y < f.rows; ++y) // 逐行变换(固定y, x-->u)
        MyFFT(f.row(y), wx).copyTo(g.row(y));
    Mat_<T> h(f.size()); // 保存逐列变换结果
    for(int u = 0; u < f.cols; ++u) // 逐列变换(固定u, y-->v)
        MyFFT(g.col(u), wy).copyTo(h.col(u));
    return h;
}
```

$$g(u,y)=\sum_{x=0}^{M-1} f(x,y)\omega_x^{ux}$$

$$F(u,v)=\sum_{y=0}^{N-1} g(u,y)\omega_y^{vy}$$

（3）最佳DFT大小。对于 $f(x,y)$ 为实数函数的情况，如果 M 或 N 不是2的正整数幂，则最佳DFT大小为 $M_e = 2^{\lceil \log_2 N \rceil}$ 和 $N_e = 2^{\lceil \log_2 N \rceil}$。此时，对于正变换，可将 $f(x,y)$ 扩充为

$$f_e(x,y) = \begin{cases} f(x,y) & 0 \le x < M \text{ and } 0 \le y < N \\ 0 & \text{else} \end{cases}$$

对于逆变换，可将 $F(u,v)$ 扩充为

$$F_e(u) = \begin{cases} F(u,v) & 0 \le u < M \text{ and } 0 \le v < N \\ \text{conj}(F(M_e - u, N_e - v)) & \text{else} \end{cases}$$

由此可知，如果 M 或 N 不是2的正整数次幂，则二维离散傅立叶变换和逆变换可以使用下列参考程序实现快速计算。

```
template<class T> // 二维DFT变换(元素类型)
auto MyFFT(Mat_<T> f, bool inverse = false) // 原函数, 是否逆变换
{   auto [w, h] = f.size();
    int nx = exp2(ceil(log2(w))), ny = exp2(ceil(log2(h)));
    Mat_<T> g(ny, nx); // 扩充后函数, 结果函数
    for(int x = 0; x < nx; ++x) // 扩充
        for(int y = 0; y < ny; ++y)
            if(x < w and y < h)
                g(y, x) = f(y, x);
            else if(not inverse) // 正变换
                g(y, x) = 0;
            else // 逆变换(共轭对称, 周期性)
                g(y, x) = conj(f((ny - y) % ny, (nx - x) % nx));
    auto wx = exp(-2 * CV_PI / nx * 1i), wy = exp(-2 * CV_PI / ny * 1i);
    if(not inverse) // 正变换
        g = MyFFT(g, wx, wy);
    else // 逆变换
        g = MyFFT(g, conj(wx), conj(wy)) / (nx * ny);
    return g({0, 0, w, h}); // 只输出与原函数对应的部分
}
```

4. 一个二维离散傅立叶变换的计算示例

【问题】请计算对下列实数矩阵进行傅立叶正变换后的变换结果（不缩放结果）。

1	7	4	0
9	4	8	8

1	7	4	0
9	4	8	8

【解答】$\omega_x = e^{-2\pi i/4} = -i,\ \omega_y = e^{-2\pi i/2} = -1$。

（1）逐行变换。

第0行（$y=0$，$u=0,1,2,3$）

$$g(0,0) = f(0,0) + f(1,0) + f(2,0) + f(3,0) = 1+7+4+0 = 12$$

$$\begin{aligned} g(1,0) &= f(0,0) + f(1,0)\omega_x + f(2,0)\omega_x^2 + f(3,0)\omega_x^3 \\ &= 1 + 7(-i) + 4(-i)^2 + 0(-i)^3 = -3-7i \end{aligned}$$

$$\begin{aligned} g(2,0) &= f(0,0) + f(1,0)\omega_x^2 + f(2,0)\omega_x^4 + f(3,0)\omega_x^6 \\ &= 1 + 7(-i)^2 + 4(-i)^4 + 0(-i)^6 = -2 \end{aligned}$$

$$g(3,0) = \mathrm{conj}(g(4-3,0)) = -3-7i \quad \text{（共轭对称）}$$

$F(N-u) = \mathrm{conj}(F(u))$

$$g(u,y) = \sum_{x=0}^{M-1} f(x,y)\omega_x^{ux}$$

12	$-3-7i$	-2	$-3+7i$
29	$1+4i$	5	$1-4i$

1	7	4	0
9	4	8	8

第1行（ $y=1$， $u=0,1,2,3$ ）

$$g(0,1)=f(0,1)+f(1,1)+f(2,1)+f(3,1)=9+4+8+8=29$$

$$\begin{aligned} g(1,1)&=f(0,1)+f(1,1)\omega_x+f(2,1)\omega_x^2+f(3,1)\omega_x^3 \\ &=9+4(-i)+8(-i)^2+8(-i)^3=1+4i \end{aligned}$$

$$\begin{aligned} g(2,1)&=f(0,1)+f(1,1)\omega_x^2+f(2,1)\omega_x^4+f(3,1)\omega_x^6 \\ &=9+4(-i)^2+8(-i)^4+8(-i)^6=5 \end{aligned}$$

$g(3,1)=\mathrm{conj}(g(4-3,1))=1-4i$ （共轭对称）

$F(N-u)=\mathrm{conj}(F(u))$

由此可知，逐行变换后可得到下列复数矩阵。

12	$-3-7i$	-2	$-3+7i$
29	$1+4i$	5	$1-4i$

$$g(u,y)=\sum_{x=0}^{M-1} f(x,y)\omega_x^{ux}$$

12	$-3-7i$	-2	$-3+7i$
29	$1+4i$	5	$1-4i$

（2）逐列变换。

第0列（ $u=0$ ， $v=0,1$ ）

$$F(0,0)=g(0,0)+g(0,1)=12+29=41$$

$$F(0,1)=g(0,0)+g(0,1)\omega_y=12+29*(-1)=-17$$

第1列（ $u=1$ ， $v=0,1$ ）

$$F(1,0)=g(1,0)+g(1,1)=(-3-7i)+(1+4i)=-2-3i$$

$$F(1,1)=g(1,0)+g(1,1)\omega_y=(-3-7i)+(1+4i)*(-1)=-4-11i$$

第2列（ $u=2$ ， $v=0,1$ ）

$$F(2,0)=g(2,0)+g(2,1)=-2+5=3$$

$$F(2,1)=g(2,0)+g(2,1)\omega_y=-2+5*(-1)=-7$$

$$F(u,v)=\sum_{y=0}^{N-1} g(u,y)\omega_y^{vy}$$

41	$-2-3i$	3	$-2+3i$
-17	$-4-11i$	-7	$-4+11i$

$F(M-u,N-v)=\mathrm{conj}(F(u,v))$

第3列（ $u=3$ ， $v=0,1$ ）

$F(3,0)=\mathrm{conj}(F(4-3,2-0))=\mathrm{conj}(F(1,0))$ （共轭对称，周期性）

$=-2+3i$

$F(3,1)=\mathrm{conj}(F(4-3,2-1))=-4+11i$ （共轭对称）

由此可知，逐列变换后可得到下列复数矩阵。

41	$-2-3i$	3	$-2+3i$
-17	$-4-11i$	-7	$-4+11i$

$$F(M-u,N-v)=\mathrm{conj}(F(u,v))$$

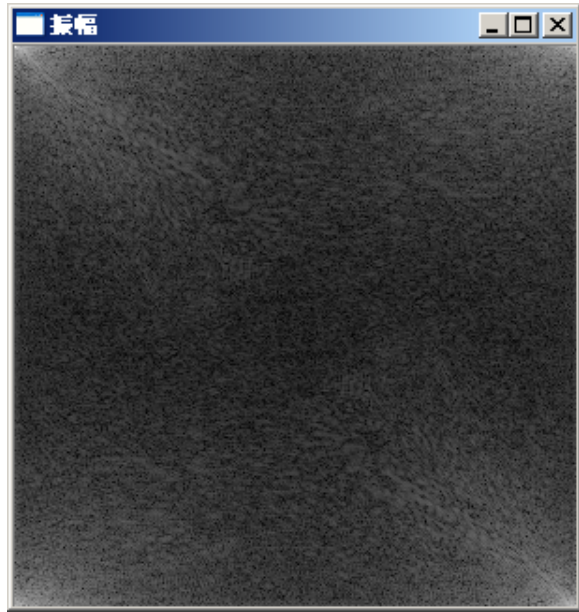
5. 图像傅立叶变换的物理含义

源图像在进行傅里叶变换后，坐标值(u, v)代表频率或梯度，对应于源图像中像素值的变化速度或突变程度，振幅$|F(u,v)|$代表相应频率或梯度的频度或份量。

在变换结果中，低频成分或低梯度部分靠近原点（原点位于结果矩阵或频谱图像的四角），是突变不太明显的成分，对应源图像中的主体和背景等部分，高频成分或高梯度部分远离原点，是突变比较明显的成分，对应源图像中的边界或边缘等部分。

通常，对一幅有意义的源图像进行傅立叶变换，变换结果或频谱图像中靠近原点周围（靠近四角）的振幅比较大，远离原点（远离四角）的振幅比较小，也就是说源图像中低频部分的份量比较多，高频部分的份量比较少。这符合我们的直观感受，一幅非极端情形的图像通常是边缘部分比较少，颜色或灰度相近的区域比较多。

因为源图像是离散的实数函数，根据二维离散傅立叶变换的共轭对称性容易知道，振幅$|F(u, v)|$是关于中心$(M/2, N/2)$对称的。



14.3.4 OpenCV对傅立叶变换的支持

1. 压缩存储

根据二维离散傅立叶变换的实数性和共轭对称性可以知道，当二维离散傅立叶变换的输入是实数数据（即虚部为0）时，变换结果可以使用如图14-3所示的压缩格式存储（单通道数组）。这种压缩存储的变换结果通常用于表示一个傅立叶正变换的变换结果或者一个傅立叶逆变换的输入。

压缩前							压缩后					
	0	1	2	3	4	5	0	1	2	3	4	5
0	2	4+8*i*	8+3*i*	3	8-3*i*	4-8*i*	2	4	8	8	3	3
1	4+7*i*	5+5*i*	2+3*i*	9+3*i*	5-8*i*	6-1*i*	4	5	5	2	3	9
2	6+5*i*	7+2*i*	6+6*i*	5+6*i*	6-8*i*	8-7*i*	7	7	2	6	6	3
3	3	1+9*i*	2+2*i*	4	2-2*i*	1-9*i*	6	1	9	2	2	5
4	6-5*i*	8+7*i*	6+8*i*	5-6*i*	6-6*i*	7-2*i*	5	8	7	6	8	6
5	4-7*i*	6+1*i*	5+8*i*	9-3*i*	2-3*i*	5-5*i*	3	6	1	5	8	4

图14-3　压缩和解压缩过程演示

压缩和解压缩的具体过程可以参考图14-3依据二维离散傅立叶变换的实数性和共轭对称性自行构造。

2. 相关函数

（1）离散傅立叶正变换。

【函数原型】void dft(InputArray src, OutputArray dst, int flags = 0);

【功能】执行单通道或双通道浮点数数组的离散傅立叶正变换。

【参数】

- src：输入数组，可以是实数数组（单通道）或复数数组（双通道）。
- dst：输出数组，类型和大小依赖于flags，必要时重建。
- flags：变换标志，通常选用下列值之一。
  - DFT_COMPLEX_OUTPUT（输出双通道数组，每个元素存储一个复数）
  - DFT_REAL_OUTPUT（使用压缩格式输出单通道数组）

【说明】如果没有在变换标志中指定DFT_COMPLEX_OUTPUT和DFT_REAL_OUTPUT，则输出数组与源数组通道数相同。

（2）离散傅立叶逆变换。

【函数原型】void idft(InputArray src, OutputArray dst, int flags = 0);

【功能】执行单通道或双通道浮点数数组的离散傅立叶逆变换。

【参数】

- src：输入数组，可以是实数数组（单通道）或复数数组（双通道）。
- dst：输出数组，类型和大小依赖于flags，必要时重建。
- flags：变换标志，通常使用DFT_SCALE（缩放结果）与下列值之一的组合。
  - DFT_COMPLEX_OUTPUT（输出双通道数组，每个元素存储一个复数）
  - DFT_REAL_OUTPUT（输出单通道数组，只存储实部）

【说明】

- 单通道的源数组是使用压缩格式存储的离散傅立叶正变换的结果。
- 如果没有在变换标志中指定DFT_COMPLEX_OUTPUT和DFT_REAL_OUTPUT，则输出数组与源数组通道数相同。

14.3.5 几个示例

1. 傅立叶变换的简单使用

下列程序首先创建一个实数矩阵，然后对该矩阵进行傅立叶正变换，最后对得到的结果进行傅立叶逆变换。

```
// DFT_Simple.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;
int main()
{   Mat X(4, 4, CV_32F), Y;
    for(auto &v : (Mat1f)X) v = rand() % 9 + 1;
    auto fmt = Formatter::FMT_PYTHON; // Python风格
    cout << format(X, fmt) << endl;
    dft(X, Y, DFT_COMPLEX_OUTPUT); // 正变换，双通道存储
    cout << format(Y, fmt) << endl;
    // 逆变换，只存储实部
    idft(Y, X, DFT_SCALE | DFT_REAL_OUTPUT);
    cout << format(X, fmt) << endl;
}
```

```
[[6, 9, 8, 5],
 [9, 2, 4, 1],
 [8, 3, 9, 3],
 [8, 7, 8, 6]]
```

```
[[[96, 0], [2, -6], [24, 0], [2, 6]],
 [[5, 13], [-1, -9], [-11, -7], [-1, -1]],
 [[6, 0], [-8, -2], [-2, 0], [-8, 2]],
 [[5, -13], [-1, 1], [-11, 7], [-1, 9]]]
```

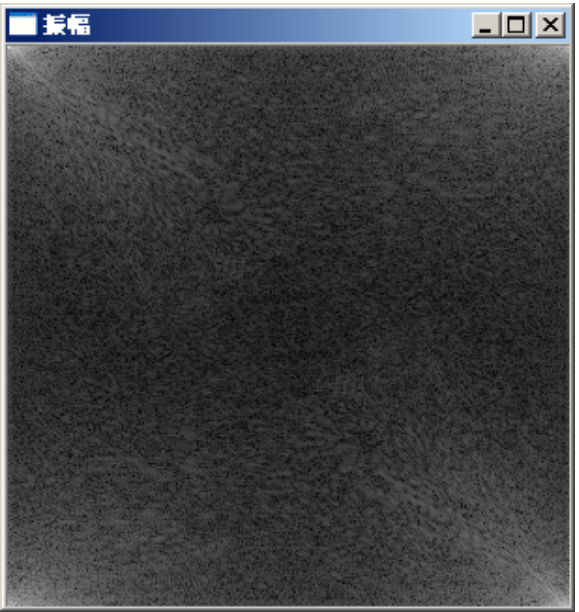
```
[[6, 9, 8, 5],
 [9, 2, 4, 1],
 [8, 3, 9, 3],
 [8, 7, 8, 6]]
```

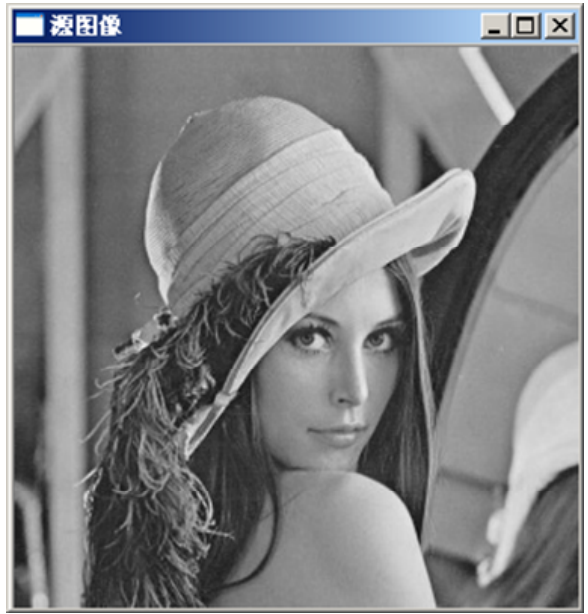
首先创建一个实数矩阵，然后对该矩阵进行傅立叶正变换，最后对得到的结果进行傅立叶逆变换。

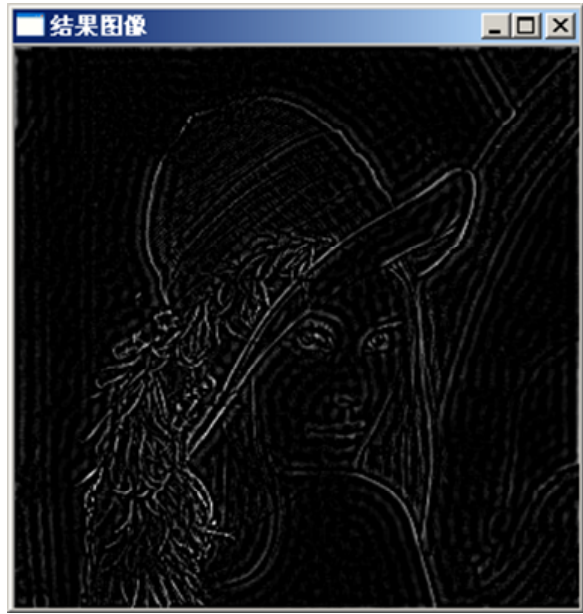
2. 图像的傅立叶变换

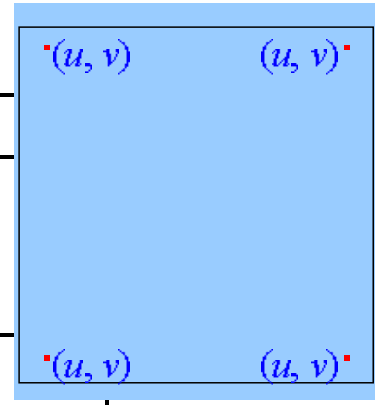
这里给出一个通过傅立叶变换突出边缘的例子。源图像在进行傅里叶变换后，低频成分靠近频谱图像的四角，高频成分靠近频谱图像的中心。为了达到突出边缘的目的，需要在频谱面上过滤掉低频成分只保留高频成分。显然，只需要在频谱图像中将靠近四角的元素改为0，就可以达到突出边缘的目的。

下列程序首先对一幅灰度图像进行傅立叶正变换，然后在变换结果中将靠近四角的元素改为0，最后对得到的结果进行傅立叶逆变换。程序运行结果如图14-2所示。










```
// DFT_Image.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat src = imread("lena.jpg", 0); // 载入灰度图像
    if(src.empty()) return -1; // 载入图像失败
    imshow("源图像", src); // 显示源图像
    // 正变换
    Mat1f X = src; // 源图像转换为浮点数图像
    Mat2f Y; // 输出复数图像
    dft(X, Y, DFT_COMPLEX_OUTPUT);
    // 将靠近四角的元素改为0(与角部距离小于40)
    auto [w, h] = X.size(); // 源图像列数和行数
    for(int u = 0; u < w; ++u)
        for(int v = 0; v < h; ++v)
        {   // 记下相对最近角部的偏移量
            int x = min(u, w - u), y = min(v, h - v);
            if(x * x + y * y < 40 * 40)
                Y(v, u) = 0; // 行列号为(v, u)
        }
```

```
    // 逆变换，只存储实部
    idft(Y, X, DFT_SCALE | DFT_REAL_OUTPUT);
    normalize(X, X, 1, 0, NORM_INF); // 增强高亮度和对比度
    imshow("结果图像", X); // 显示结果
    waitKey();
}
```

3. 傅立叶变换的频域图像

通常使用振幅生成频域图像，也可称为频谱图像。生成频域图像时通常需要进行对数变换和归一化等处理。对数变换用于增强频谱图像的灰度级细节，归一化用于增强频谱图像的对比度，便于观察。

下列程序使用图像形式给出了对一幅灰度图像进行傅立叶正变换后得到的结果。在显示结果图像时，对变换结果进行了计算振幅、对数变换和归一化等处理。程序运行结果如图14-4所示。

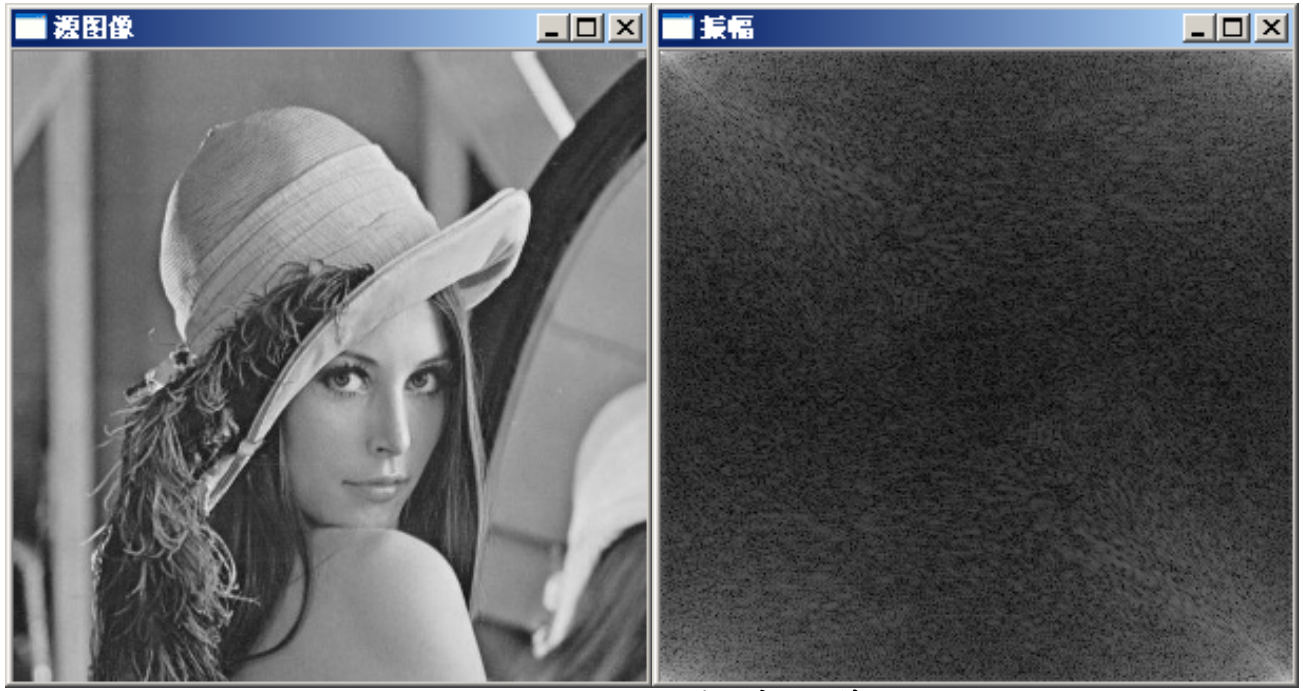


图14-4　DFT频域图像

```
// DFT_Spectrum.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("lena.jpg", 0); // 载入灰度图像
    if(X.empty()) return -1; // 载入图像失败
    imshow("源图像", X); // 显示源图像
    // 正变换(先转换为浮点数图像, 输出复数图像)
    dft((Mat1f)X, X, DFT_COMPLEX_OUTPUT);
    // 频谱图像, 只显示振幅
    Mat Re, Im, Mag; // 实部和虚部, 频谱的振幅
    // 将DFT图像分为实部和虚部
    extractChannel(X, Re, 0), extractChannel(X, Im, 1);
    magnitude(Re, Im, Mag); // Mag = (Re^2 + Im^2)^0.5
    // 用对数变换Mag = log(Mag + 1)增强灰度级细节
    log(Mag + 1, Mag);
    normalize(Mag, Mag, 1, 0, NORM_INF); // 增强高亮度和对比度
    imshow("振幅", Mag); // 显示振幅图像
    waitKey();
}
```

14.3.6# 频域图像的压缩存储和中心化

1. 离散傅立叶变换的压缩存储

（1）压缩方法。将复数矩阵X压缩成实数矩阵Y。

```
template<class T> // 辅助函数, 用于计算压缩数据的首列和尾列
void pack(Mat_<complex<T>> X, Mat_<T> Y, int jx, int jy)
{   // 由源数据的jx列计算压缩数据的第jy列
    int h = X.rows; // 行数简写
    Y(0, jy) = X(0, jx).real(); // 计算第jy列的首行, 其中X(0, jx)的虚部是0
    if(h % 2 == 0) // 若行数是偶数, 则计算第jy列的尾行
        Y(h - 1, jy) = X(h / 2, jx).real(); // 其中X(h/2, jx)的虚部是0
    for(int i = 1; i <= (h - 1) / 2; ++i) // 计算第jy列的中间行
        Y(2 * i - 1, jy) = X(i, jx).real(), // 奇数行是实部
        Y(2 * i, jy) = X(i, jx).imag(); // 偶数行是虚部
}
```

	压缩前						压缩后					
	0	1	2	3	4	5	0	1	2	3	4	5
0	2	4+8*i*	8+3*i*	3	8-3*i*	4-8*i*	2	4	8	8	3	3
1	4+7*i*	5+5*i*	2+3*i*	9+3*i*	5-8*i*	6-1*i*	4	5	5	2	3	9
2	6+5*i*	7+2*i*	6+6*i*	5+6*i*	6-8*i*	8-7*i*	7	7	2	6	6	3
3	3	1+9*i*	2+2*i*	4	2-2*i*	1-9*i*	6	1	9	2	2	5
4	6-5*i*	8+7*i*	6+8*i*	5-6*i*	6-6*i*	7-2*i*	5	8	7	6	8	6
5	4-7*i*	6+1*i*	5+8*i*	9-3*i*	2-3*i*	5-5*i*	3	6	1	5	8	4

```
template<class T> // 辅助函数, 用于计算压缩数据的中间列
void pack(Mat_<complex<T>> X, Mat_<T> Y, int i)
{   // 由源数据的i行计算压缩数据的第i行(只含中间列)
    int w = X.cols; // 列数简写
    for(int j = 1; j <= (w - 1) / 2; ++j)
        Y(i, 2 * j - 1) = X(i, j).real(), // 奇数列是实部
        Y(i, 2 * j) = X(i, j).imag(); // 偶数列是虚部
}
```

```
template<class T> // 将复数矩阵X压缩成实数矩阵Y
void pack(Mat_<complex<T>> X, Mat_<T> Y)
{   auto [w, h] = X.size(); // 列数和行数
    pack(X, Y, 0, 0); // 计算压缩数据的首列
    if(w % 2 == 0) // 若列数是偶数, 则计算压缩数据的尾列
        pack(X, Y, w / 2, w - 1);
    for(int i = 0; i < h; ++i) // 计算压缩数据的中间列
        pack(X, Y, i);
}
```

	压缩前						压缩后					
	0	1	2	3	4	5	0	1	2	3	4	5
0	2	4+8*i*	8+3*i*	3	8-3*i*	4-8*i*	2	4	8	8	3	3
1	4+7*i*	5+5*i*	2+3*i*	9+3*i*	5-8*i*	6-1*i*	4	5	5	2	3	9
2	6+5*i*	7+2*i*	6+6*i*	5+6*i*	6-8*i*	8-7*i*	7	7	2	6	6	3
3	3	1+9*i*	2+2*i*	4	2-2*i*	1-9*i*	6	1	9	2	2	5
4	6-5*i*	8+7*i*	6+8*i*	5-6*i*	6-6*i*	7-2*i*	5	8	7	6	8	6
5	4-7*i*	6+1*i*	5+8*i*	9-3*i*	2-3*i*	5-5*i*	3	6	1	5	8	4

（2）解压方法。将实数矩阵X解压成复数矩阵Y。

```
template<class T> // 辅助函数, 用于解压首列和尾列
void unpack(Mat_<T> X, Mat_<complex<T>> Y, int jx, int jy)
{   // 将压缩数据的第jx列解压到第jy列
    int h = X.rows; // 行数简写
    Y(0, jy) = X(0, jx); // 解压第jx列的首行, 其中Y(0, jy)的虚部是0
    if(h % 2 == 0) // 若行数是偶数, 则解压第jx列的尾行
        Y(h / 2, jy) = X(h - 1, jx); // 其中 Y(h/2, jy)的虚部是0
    // 解压第jx列的中间行, 奇数行是实部, 偶数行是虚部
    for(int i = 1; i <= (h - 1) / 2; ++i)
        Y(i, jy) = {X(2 * i - 1, jx), X(2 * i, jx)};
}
```

	压缩前						压缩后					
	0	1	2	3	4	5	0	1	2	3	4	5
0	2	4+8*i*	8+3*i*	3	8-3*i*	4-8*i*	2	4	8	8	3	3
1	4+7*i*	5+5*i*	2+3*i*	9+3*i*	5-8*i*	6-1*i*	4	5	5	2	3	9
2	6+5*i*	7+2*i*	6+6*i*	5+6*i*	6-8*i*	8-7*i*	7	7	2	6	6	3
3	3	1+9*i*	2+2*i*	4	2-2*i*	1-9*i*	6	1	9	2	2	5
4	6-5*i*	8+7*i*	6+8*i*	5-6*i*	6-6*i*	7-2*i*	5	8	7	6	8	6
5	4-7*i*	6+1*i*	5+8*i*	9-3*i*	2-3*i*	5-5*i*	3	6	1	5	8	4

```
template<class T> // 辅助函数, 用于解压中间列
void unpack(Mat_<T> X, Mat_<complex<T>> Y, int i)
{   // 由压缩数据的i行计算源数据的第i行(只含中间列)
    int w = X.cols; // 列数简写
    for(int j = 1; j <= (w - 1) / 2; ++j) // 奇数列是实部, 偶数列是虚部
        Y(i, j) = {X(i, 2 * j - 1), X(i, 2 * j)};
}
```

压缩前							压缩后					
	0	1	2	3	4	5	0	1	2	3	4	5
0	2	4+8*i*	8+3*i*	3	8-3*i*	4-8*i*	2	4	8	8	3	3
1	4+7*i*	5+5*i*	2+3*i*	9+3*i*	5-8*i*	6-1*i*	4	5	5	2	3	9
2	6+5*i*	7+2*i*	6+6*i*	5+6*i*	6-8*i*	8-7*i*	7	7	2	6	6	3
3	3	1+9*i*	2+2*i*	4	2-2*i*	1-9*i*	6	1	9	2	2	5
4	6-5*i*	8+7*i*	6+8*i*	5-6*i*	6-6*i*	7-2*i*	5	8	7	6	8	6
5	4-7*i*	6+1*i*	5+8*i*	9-3*i*	2-3*i*	5-5*i*	3	6	1	5	8	4

```
template<class T> // 将实数矩阵X解压成复数矩阵Y
void unpack(Mat_<T> X, Mat_<complex<T>> Y)
{   auto [w, h] = X.size(); // 列数和行数
    unpack(X, Y, 0, 0); // 解压首列
    if(w % 2 == 0) // 若列数是偶数, 则解压尾列
        unpack(X, Y, w - 1, w / 2);
    for(int i = 0; i < h; ++i) // 解压中间列
        unpack(X, Y, i);
    // 补齐共轭元素
    for(int i = 0; i < h; ++i)
        for(int j = 0; j < w; ++j)
            if(i > h / 2 or j > w / 2)
                Y(i, j) = conj(Y((h - i) % h, (w - j) % w));
}
```

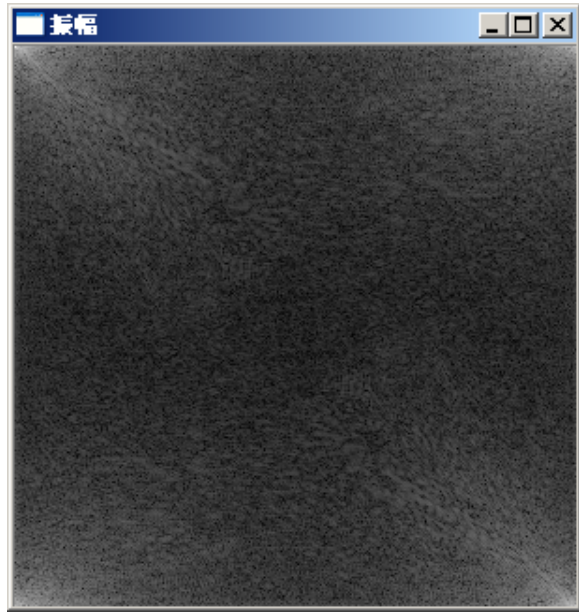
	压缩前						压缩后					
	0	1	2	3	4	5	0	1	2	3	4	5
0	2	4+8*i*	8+3*i*	3	8-3*i*	4-8*i*	2	4	8	8	3	3
1	4+7*i*	5+5*i*	2+3*i*	9+3*i*	5-8*i*	6-1*i*	4	5	5	2	3	9
2	6+5*i*	7+2*i*	6+6*i*	5+6*i*	6-8*i*	8-7*i*	7	7	2	6	6	3
3	3	1+9*i*	2+2*i*	4	2-2*i*	1-9*i*	6	1	9	2	2	5
4	6-5*i*	8+7*i*	6+8*i*	5-6*i*	6-6*i*	7-2*i*	5	8	7	6	8	6
5	4-7*i*	6+1*i*	5+8*i*	9-3*i*	2-3*i*	5-5*i*	3	6	1	5	8	4

2. 频谱图像的中心化

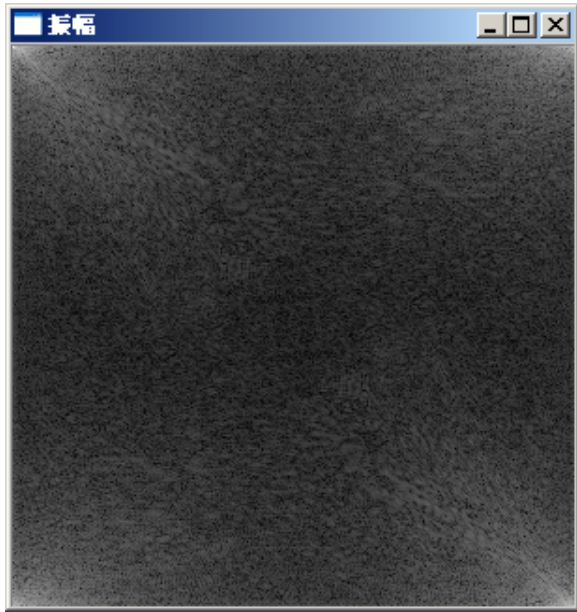
源图像进行傅里叶变换后，低频成分靠近频谱图像的四角，也就是原点位于频谱图像的四角，不便于观察，所以通常需要将低频成分交换到靠近频谱图像的中心（因为对于对称的图像，原点位于中心比原点位于四角要方便观察一些），这可以通过中心化实现。

假设用一条水平线和一条垂直线将频谱图平均分成四块，中心化就是对这四块进行对角线的交换与反对角线的交换，目的是将原点移到频谱图像的中心。

下述函数FFTShift()给出了中心化的一种参考实现。



```
// 中心化，分成四块进行对角交换
Mat fftShift(Mat X)
{   Mat Y(X.size(), X.type()); // 结果
    auto [w, h] = X.size() / 2; // 左上大小
    auto [w2, h2] = X.size() - Size(w, h); // 右下大小
    X({0, 0, w, h}).copyTo(Y({w2, h2, w, h})); // 左上 --> 右下
    X({w, 0, w2, h}).copyTo(Y({0, h2, w2, h})); // 右上 --> 左下
    X({w, h, w2, h2}).copyTo(Y({0, 0, w2, h2})); // 右下 --> 左上
    X({0, h, w, h2}).copyTo(Y({w2, 0, w, h2})); // 左下 --> 右上
    return Y;
}
```



14.4 离散余弦变换[Ж]

同傅立叶变换一样，余弦变换也是以图像中灰度的变化频率为处理对象，是另一种重要的图像处理技术。

14.4.1 相关原理和方法

因为数字图像是离散函数，所以这里也只考虑离散余弦变换。

1. 离散余弦变换

离散余弦变换的表达式为 $F(u)=K(u)\sum_{x=0}^{N-1} f(x)\cos\frac{(2x+1)u\pi}{2N}$，$u\in Z$。

离散余弦变换的逆变换为 $f(x)=\sum_{u=0}^{N-1} K(u)F(u)\cos\frac{(2x+1)u\pi}{2N}$，$x\in Z$。

变换中的 $K(u)$ 定义为 $K(u)=\left((u=0)?\sqrt{1/N}:\sqrt{2/N}\right)$，$u\in Z$。

2. 二维离散余弦变换

因为数字图像是二维离散函数，所以只需要考虑二维离散余弦变换。二维离散余弦变换的表达式为

$$F(u,v)=K(u)K(v)\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)\cos\frac{(2x+1)u\pi}{2M}\cos\frac{(2y+1)v\pi}{2N}$$

二维离散余弦变换的逆变换为

$$f(x,y)=\sum_{v=0}^{N-1}\sum_{u=0}^{M-1}K(u)K(v)F(u,v)\cos\frac{(2x+1)u\pi}{2M}\cos\frac{(2y+1)v\pi}{2N}$$

【注】在数字图像处理中，通常规定 $u=0,1,...,M-1$， $v=0,1,...,N-1$，$x=0,1,...,M-1$， $y=0,1,...,N-1$。

14.4.2 OpenCV对离散余弦变换的支持

1. 相关函数

【函数原型】

- void dct(InputArray src, OutputArray dst);
- void idct(InputArray src, OutputArray dst);

【功能】执行一维或二维实数数组的离散余弦变换（正或逆）。

【参数】

- src：输入数组，大小为偶数的单通道实数数组。
- dst：输出数组，类型和大小与输入数组一致，必要时重建。

2. 离散余弦变换的简单使用

这里使用一个简单的例子说明离散余弦变换的简单使用。

```
// DCT_Simple.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;
```

```
int main()
{   Mat1f X(2, 6), Y;
    for(auto &e : X) e = rand() % 9 + 1;
    cout << X << endl;
    dct(X, Y); cout << Y << endl;// DCT正变换
    idct(Y, X); cout << X << endl;// DCT逆变换
}
```

```
[41, 67, 34, 0, 69, 24;
  78, 58, 62, 64, 5, 45]
```

```
[157.90529, 37.820503, 9.899497, -9.5262785, -10.206208, 3.1794908;
  -22.227991, -18.382685, 12.020815, 0.86602533, -60.42075, 28.382685]
```

```
[40.999989, 66.999992, 33.999992, -3.8146973e-06, 68.999992, 23.999989;
  77.999985, 57.999992, 61.999992, 63.999996, 5, 44.999996]
```

3. 图像的离散余弦变换

这里给出一个通过离散余弦变换突出边缘的例子。与离散傅立叶变换不同，源图像在进行离散余弦变换后，低频成分靠近原点，高频成分远离原点，所以，只需要在频谱图像中将靠近原点的元素改为0，就可以达到突出边缘的目的。

下列程序首先对一幅灰度图像进行离散余弦正变换，然后在变换结果中将靠近原点的元素改为0，最后对得到的结果进行离散余弦逆变换。程序运行结果如图14-5所示。

图14-5　灰度图像的离散余弦变换

```
// DCT_Image.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("lena.jpg", 0); // 载入灰度图像
    if(X.empty()) return -1; // 载入图像失败
    imshow("源图像", X);
    dct(Mat1f(X), X); // 正变换(先转换为浮点数图像)
    // 将靠近原点的元素改为0
    for(int u = 0; u < X.cols; ++u)
        for(int v = 0; v < X.rows; ++v)
            if(u * u + v * v < 60 * 60)
                X.at<float>(v, u) = 0; // 行列号为(v, u)
    idct(X, X); // 逆变换
    normalize(X, X, 1, 0, NORM_INF); // 增强高亮度和对比度
    imshow("结果图像", X);
    waitKey();
}
```

4. 离散余弦变换的频域图像

源图像进行离散余弦变换后，生成频域图像时通常需要进行对数变换和归一化等处理。对数变换用于增强频谱图像的灰度级细节，归一化用于增强频谱图像的对比度，便于观察。

下列程序使用图像形式给出了对一幅灰度图像进行离散余弦正变换后得到的结果。在显示结果图像时，对变换结果进行了对数变换和归一化等处理。程序运行结果如图14-6所示。

图14-6　DCT频域图像

```
// DCT_Spectrum.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("lena.jpg", 0); // 载入灰度图像
    if(X.empty()) return -1; // 载入图像失败
    imshow("源图像", X); // 显示源图像
    dct(Mat1f(X), X); // DCT正变换(先转换为浮点数图像)
    // 频谱图像
    log(X + 1, X); // 对数变换X = log(X + 1)用于增强灰度级细节
    normalize(X, X, 1, 0, NORM_INF); // 增强高亮度和对比度
    imshow("频谱图像", X);
    waitKey();
}
```

14.5 练习题

14.5.1 基础知识题

1. 证明n次单位根ω有下列性质。

（1）周期性。序列 $\ldots,\omega^{-2},\omega^{-1},\omega^{0},\omega^{1},\omega^{2},\ldots$ 的周期是 n 。

（2）不重复性。 $\omega^{0},\omega^{1},\ldots,\omega^{n-1}$ 各不相同。

（3）共轭对称性。 $\omega^{n-k}=\mathrm{conj}(\omega^{k})$ ， k 是整数。

（4）共轭根。 $\overline{\omega}=\mathrm{conj}(\omega)$ 也是一个 n 次单位根。

（5）相反性。当 n 是偶数时， $\omega^{k+n/2}=-\omega^{k}$ ，且 ω^{2} 是一个 $n/2$ 次单位根。

2. 设 ω 是一个 N 次单位根，证明变换 $F(u)=\sum_{x=0}^{N-1} f(x)\omega^{ux}$ 有下列性质。

（1）周期性。$F(u+N)=F(u)$。

（2）平移性。若 $g(x)=f(x-x_0)$，则 $G(u)=\omega^{ux_0}F(u)$。

（3）线性组合性。若 $g(x)=\alpha f_1(x)+\beta f_2(x)$，则 $G(u)=\alpha F_1(u)+\beta F_2(u)$。

（4）平均值。$\overline{f(x)}=F(0)/N$。

（5）共轭对称性。如果 $f(x)$ 是实数函数，则 $F(N-u)=\mathrm{conj}(F(u))$。

（6）实数性。当 $f(x)$ 是实数函数时，$F(0)$ 是实数；而且，若 N 是偶数，则 $F(N/2)$ 是实数。

3. 设 ω_x 是一个 M 次单位根，ω_y 是一个 N 次单位根，证明变换 $F(u,v)=\sum_{y=0}^{N-1}\sum_{x=0}^{M-1} f(x,y)\omega_x^{ux}\omega_y^{vy}$ 具有下列性质。

（1）周期性。$F(u+M,v)=F(u,v+N)=F(u+M,v+N)=F(u,v)$。

（2）平移性。若 $g(x,y)=f(x-x_0,y-y_0)$，则 $G(u,v)=\omega_x^{ux_0}\omega_y^{vy_0}F(u,v)$。

（3）线性组合性。若 $g(x,y)=\alpha f_1(x,y)+\beta f_2(x,y)$，则 $G(u)=\alpha F_1(u,v)+\beta F_2(u,v)$。

（4）平均值。$\overline{f(x,y)}=F(0,0)/(MN)$。

（5）共轭对称性。如果 $f(x,y)$ 是实数函数，则 $F(M-u,N-v)=\mathrm{conj}(F(u,v))$。

（6）实数性。当 $f(x,y)$ 是实数函数时，$F(0,0)$ 是实数；而且，若 M 是偶数，则 $F(M/2,0)$ 是实数；若 N 是偶数，则 $F(0,N/2)$ 是实数；若 M 和 N 都是偶数，则 $F(M/2,N/2)$ 是实数。

4. 请计算对下列实数矩阵进行傅立叶正变换后的变换结果（不缩放结果）。

1	7	4	0
9	4	8	8

5. 请计算对下列复数矩阵进行傅立叶逆变换后的变换结果（缩放结果，i是虚数单位）。

41	-2-3i	3	-2+3i
-17	-4-11i	-7	-4+11i

6. 某研究者在对一个4×4实数矩阵进行傅立叶变换时，将变换结果记录在一张草稿纸上。半小时后，由于意外，有部分数据受到了污损，请根据傅立叶变换的性质帮这位研究者恢复被污损的数据，并说明依据。其中，受到污损后的数据如下（i是虚数单位）。

67	-5-8i	-5	-5+8i
-4-19i	10-9i	-11i	
-11	-7-4i	-3	
	-10-7i		

14.5.2 程序设计题

1. 使用OpenCV编写一个演示傅立叶变换和逆变换的程序。该程序首先装入一幅灰度图像并显示该图像，然后对该图像进行傅立叶正变换，对得到的结果进行傅立叶逆变换，显示得到的结果以便与原图像进行比对。

2$^{\text{米}}$. 使用OpenCV编写一个演示离散余弦变换和逆变换的程序。该程序首先装入一幅灰度图像并显示该图像，然后对该图像进行离散余弦正变换，对得到的结果进行离散余弦逆变换，显示得到的结果以便与原图像进行比对。

3. 请为OpenCV提供一个中心化函数FFTShift，该函数的函数原型为“Mat FFTShift(Mat X);”。

14.5.3 阶段实习题

1. 使用OpenCV编写一个演示傅立叶变换和逆变换的程序。该程序首先装入一幅灰度图像，并创建一个滑块（初始值为0，最大值为16），然后对该图像进行傅立叶正变换，在正变换的结果中将小于value（value=滑块位置×250-2500）的元素改为value，最后对得到的结果进行傅立叶逆变换，并显示得到的结果图像。

2. 使用OpenCV编写一个演示离散余弦变换和逆变换的程序。该程序首先装入

一幅灰度图像，并创建一个滑块（初始值为0，最大值为16），然后对该图像进行离散余弦正变换，在正变换的结果中将大于value（value=250-滑块位置×25）的元素改为value，最后对得到的结果进行离散余弦逆变换，并显示得到的结果图像。

3. 傅立叶变换结果的压缩存储。根据二维离散傅立叶变换的实数性和共轭对称性可以知道，当二维离散傅立叶变换的输入是实数数据时，变换结果可以使用如图14-3所示的压缩格式存储。请参考图14-3依据二维离散傅立叶变换的实数性和共轭对称性给出压缩和解压缩的具体过程。

4. 灰度图像的彩色化。首先读入一幅灰度图像，然后通过一幅或几幅彩色图像作为参考图像，将读入的灰度图像变换成彩色图像。可以提供几种结果供程序使用者选择。相关的变换方法请到互联网上查阅。