图像的傅里叶变换和余弦变换

如何从图片角度理解傅里叶变换

我们所理解的世界，是以时间贯穿的。固定强度的音符随着时间的流逝形成连续的频谱。

这其实是在说，在时域连续的东西，放到频域中可能是几个固定大小的离散值（脉冲），最容易理解的就是余弦波，在时域上无休无止，却可以只用一个值表示（振幅）。而复杂的波形，也可以由几个简单的余弦波叠加得到，也就是只用几个简单的值表示得到。推而广之，只要找到合适的基元，就可以表示任意复杂的东西。**傅里叶基元是**

$$e^{-jN}, j\text{是虚数单位}$$
$$\text{凭什么它是基元：}$$
$$e^{\pi j}+1=0$$

傅里叶解决的便是从时域到频域的转换。可以想象任意复杂的东西都可以被表示为较少量离散的值，则图片的压缩问题似乎得到了解决。(当然还有更多可以应用的领域)

傅里叶变换公式

先给出**一维连续傅里叶变换的公式**：

$$F(u)=\int_{-\infty}^{\infty}f(x)e^{-j2\pi ux}dx$$

然后给出二维离散傅里叶变换公式：

$$F(u,v)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}, u=0...M-1, V=0...n-1$$

虽然达到了变换的目的，但是这个速度实在太慢了，因此，我们采用了好理解又快一点的**分离法**：

$$F(u,v)=\sum_{x=0}^{M-1}e^{-j2\pi\frac{ux}{M}}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi\frac{vy}{N}}$$
$$f(x,y)=\frac{1}{M}\sum_{u=0}^{M-1}e^{j2\pi\frac{ux}{M}}\frac{1}{N}\sum_{v=0}^{N-1}F(u,v)e^{j2\pi\frac{vy}{N}}$$

傅里叶压缩实验

输入是 132X165的图片。由于噪音多为高频信号，因此采用低通滤波的方法尝试压缩，

由于傅里叶的能量集中在幅值较大的地方，我们认为绝对值较大的幅值就大。而且大部分区域的幅值都小到可以忽略。我们只取大于最大值/N的部分，pinp是DFT变换得到的频谱图，如下所示：

```
for (int y = 0; y < N; y++) {
        cnt = 0;
        for (int u = u_b; u < u_e; u++) {
            for (int v = v_b; v < v_e; v++) {
                if (abs(pinp[v][u]) > max/900) {
                        pinp_i = pinp[v][u];
                        cnt++;
                    }
            }}}
```

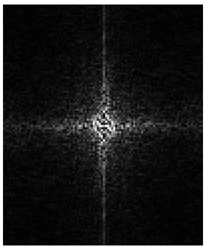
不断调整N的值，最终使用的频谱部分占总面积的百分比及效果如下：

0.022　　　　0.151　　　　0.244

0.386　　　　0.430　　　　原图

可以看到在频谱使用比例在15%左右时已经满足基本观看需求了，在比例达到38.6%时已经与原图没有明显差别了。

在傅里叶变换中得到的频谱图如图所示



可以看到除了中间亮区域外，较暗的，包含更少信息的占了大多数，采用稀疏存储的办法可以达到较好的压缩目的。

傅里叶滤波实验

对频谱的操作还有设置固定的掩码，只对该区域内的频谱进行反变换，由于低通滤波参数更加容易设置，因此我们只对比了低通滤波。

首先，我们采用了较好实现的以方框做掩码的办法，实现代码如下，设定滤波器长宽，中心为图片中心。下图中的数字如33X41意为len_u X lenv

```
int u_b = M / 2 - len_u;
int u_e = M / 2 + len_u;
int v_b = N / 2 - len_v;
int v_e = N / 2 + len_v;
for (int y = 0; y < N; y++) {
    for (int u = u_b; u < u_e; u++) {
        for (int v = v_b; v < v_e; v++) {
        ...}}
    for (int x = 0; x < M; x++) {
        for (int u = u_b; u < u_e; u++) {
        ...}}
    }
```


原图


(0.248)


(0.6303)

可以看到52X66图片画质与原图差别已经不大，但是仍有较明显的振铃效应

接下来采用圆形滤波器，核心实现代码如下：

```
for (int u = u_b; u < u_e; u++) {
    int v_half = sqrt(pow(r, 2) - pow(u - (M / 2), 2));
    v_b = N / 2 - v_half;
    v_e = N / 2 + v_half;
    for (int v = v_b; v < v_e; v++) {...}}
```

效果对比：


0.182圆

0.215圆

0.418圆

与上图对比不难发现，滤波器形状对呈现效果影响不大。在低通滤波的情况下，主要显示的是色块信息，忽略掉了大部分边缘或者噪音

插叙：固定掩码的压缩效果对比

此时也可对比固定形状做掩码时图片压缩的效率，33X41转化为百分比为24.8%，对比图片清晰度如下：

33X41(0.248)0.2443

可见固定掩码做压缩的效率远低于稀疏存储。且由于低通滤波有明显的振铃效应，图片质量大大降低了

余弦变换

由于余弦变换与傅里叶变换有诸多相似性，他们都有对频域和时域变换的一组正交基，余弦变换由于其对偶性，可以忽略复数运算，一定程度上提高了运算效率。

二维离散余弦变换与反变换的公式如下：

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

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

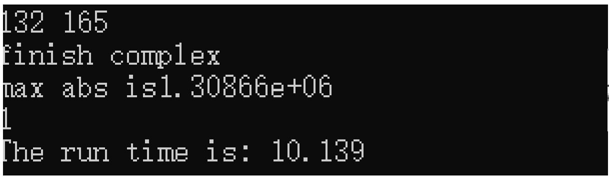
运行时间效率对比

首先我们对比了不使用任何简化的DFT和DCT的运行时间：

```
clock_t startTime, endTime;
    startTime = clock();

    ReverseDFT2(gray);
    endTime = clock();
    cout << "The run time is: " << (double)(endTime - startTime) /
CLOCKS_PER_SEC << endl;
```

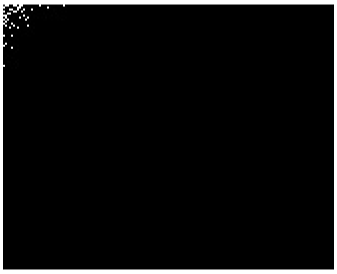
使用方法	花费时间/s
DFT分离计算法	10.139
DCT无优化计算法	155.628
DFT无优化算法	>400s



图片大小为132X165，由公式可知，DFT无优化法时间复杂度O(N^4)， DFT分离法时间复杂度O(N^3)，估算可知，DFT无优化约需1000s出结果。可见DCT确实极大的提升了运算速度。

图片压缩效果对比

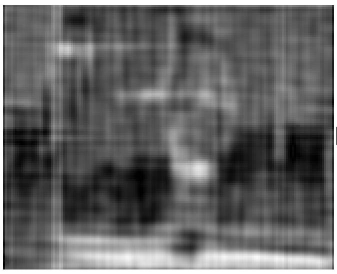
DCT得到的频谱如图：



相比于DFT得到的频谱图，DCT更加集中，下图为均采用稀疏存储法时的效果对比：


IDCT全

IDCT0.494


DFT0.02

DFT0.151

无需多言，DCT在使用率达到40%时，竟然还没有DFT使用率15%的效果好

以上。