数字媒体技术选修实验
（一共两个必选实验）

刘绍辉
shliu@hit.edu.cn

哈尔滨工业大学计算机科学与技术学院

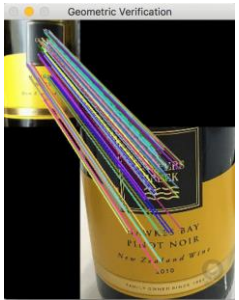
2021春

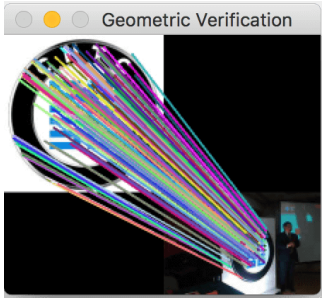
实验一

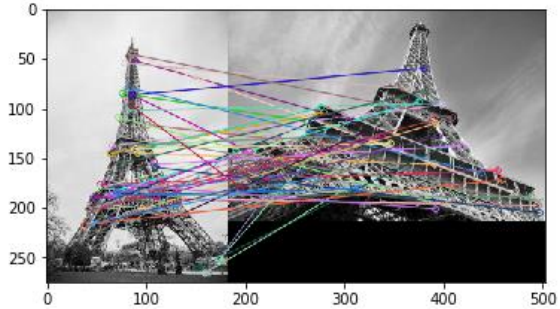
- 实验一
  - 熟悉编程环境
  - 熟悉BMP图像的结构，编程实现BMP图像的阅读和显示
    - 能获取图像任意一点的像素值
    - 能将**图像分成任意块大小**，并**置乱块**的位置；能指定区域内的图像分块并置乱块的大小
  - 能阅读wav音频文件，并将**原始的PCM音频数据显示**出来，并画出其大小示意图（画出波形图）
  - 能调用DFT，DCT对图像和音频进行变换处理
    - 对**图像进行二维DCT和DFT正变换和反变换**，并显示正变换后的图像，注意如何才能获得**更好的显示**效果
    - 对图像进行**分块8*8DCT变换**后，将其中的64个DCT系数按照**Zigzag扫描排序**，**设定保留的DCT系数作为函数参数，然后逆变换**，并显示逆变换后的图像，比较原始图像和该图像的**PSNR值**。
  - 能调用JBIG的编码器和解码器，对二值图像进行编码压缩和解码，并在界面上进行显示。
  - 有以上知识的同学尝试下列任务
    - 熟悉JPEG压缩的流程，对上述BMP图像按照8*8块分块后进行压缩
    - 对JPEG图像进行加密处理

实验二

- 用手机拍摄任意一段视频，然后对其中的图像进行SIFT、ORB、HOG特征提取，并对其中的主要目标进行对齐操作。
  - 选择某个目标，从手机或别的摄像机设备对其进行多角度、多距离的拍摄（可以是桌上的某个瓶子或者杯子等常见设备，蹲下拍摄一只蚂蚁爬行的场景，宠物运动的场景，也可以是远场景，如站在天桥上拍摄一段马路上的车流，在寝室拍摄一段地面的视频，在教师拍摄一段楼下人群行走的视频，或车辆的视频等）
  - 采用opencv等基本平台提取视频帧进行播放控制
  - 对视频中的任意两帧，对其中的同一个物体进行特征提取（例如SIFT特征，或者ORB特征，或者HOG特征等，都可以调用现成的特征函数），然后根据特征进行对齐（如随机一致同意RANSAC，或与ORB-SLAM2里面类似），注意其中的目标可能有仿射变换，或者别的运动造成的位置、尺度等的变化。（就如视频中有目标移动，或者拍摄时手部有抖动），然后在对齐的图像之间画上直线







实验二

- 在上述匹配的基础上，继续实现以下功能
  - 两幅图像上多个目标匹配，然后确认哪些目标出现了异常，例如，停车场上，哪些位置的车辆发生了变化，港口中，哪些船舶发生了移动，几场上，哪些飞机飞走了或者移动了
  - 视频中，连续视频帧中运动目标的移动有多少个像素，每个移动目标都检测出来其相对位移，然后估计其相对速度（拍摄者和运动目标的相对速度）
  - 在同一幅图像中，尝试用这种匹配的思想来检测复制粘贴操作（将图像的某一个区域拷贝粘贴到同一图像的不同位置，来形成多个目标）

实验三（选做）

- 跨媒体的情感分类
  - 根据图像、视频、音频，分别提取其特征，根据特征对其表达的情感进行分类
  - 是否可以在各种多媒体的特征表示之间建立转换关系，从而可以用图像特征来检索具有同样情感属性的音频和视频，或者用任意一种表达其属性。
  - 情感属性可以换为：正能量性，积极性等，并进一步给出定量分数。

要求

- 最后提交实验报告（包括原理介绍、论文阅读、测试数据集和测试结果等）和源代码以及可执行程序
- 具体参见模板。

位图格式

- 位图格式
  - 每行字节数必须是4的整数倍
  - 8比特及其以下图像都带有调色板，采用调色板的索引值来表示图像的像素值，因此可以是彩色的，例如GIF是8比特图像
  - 8比特以上的图像一般没有调色板，直接将图像的RGB值放在相应的位置上
  - 位图文件：14字节的文件头+40字节的信息头+[调色板]+像素数值

BMP图像结构

- **BITMAPFILEHEADER(14 bytes)**

```
typedef struct tagBITMAPFILEHEADER {
        WORD bfType;
        DWORD bfSize;
        WORD bfReserved1;
        WORD bfReserved2;
        DWORD bfOffBits;
} BITMAPFILEHEADER, *PBITMAPFILEHEADER;
```

BITMAP图像结构

- **BITMAPINFO**

```
typedef struct tagBITMAPINFO {
        BITMAPINFOHEADER bmiHeader;
        RGBQUAD bmiColors[1];
} BITMAPINFO, *PBITMAPINFO;
```

BITMAP图像结构

- **BITMAPINFOHEADER(40 Bytes)**

```
typedef struct tagBITMAPINFOHEADER{
        DWORD biSize;
        LONG biWidth;
        LONG biHeight;
        WORD biPlanes;(1)
        WORD biBitCount;
        DWORD biCompression;
        DWORD biSizeImage;
        LONG biXPelsPerMeter;
        LONG biYPelsPerMeter;
        DWORD biClrUsed;
        DWORD biClrImportant;
} BITMAPINFOHEADER, *PBITMAPINFOHEADER;
```

BITMAP图像结构

- **RGBQUAD**

```
typedef struct tagRGBQUAD {
        BYTE rgbBlue;
        BYTE rgbGreen;
        BYTE rgbRed;
        BYTE rgbReserved;
} RGBQUAD;
```

颜色表的起始位置

- pColor = ((LPSTR)pBitmapInfo + (WORD)(pBitmapInfo->bmiHeader.biSize));

位图字节的起始值和长度

- 起始位置：
  - PBITMAPFILEHEADER.bfOffBits;
- 长度：
  - PBITMAPFILEHEADER.bfSize - PBITMAPFILEHEADER.bfOffBits;

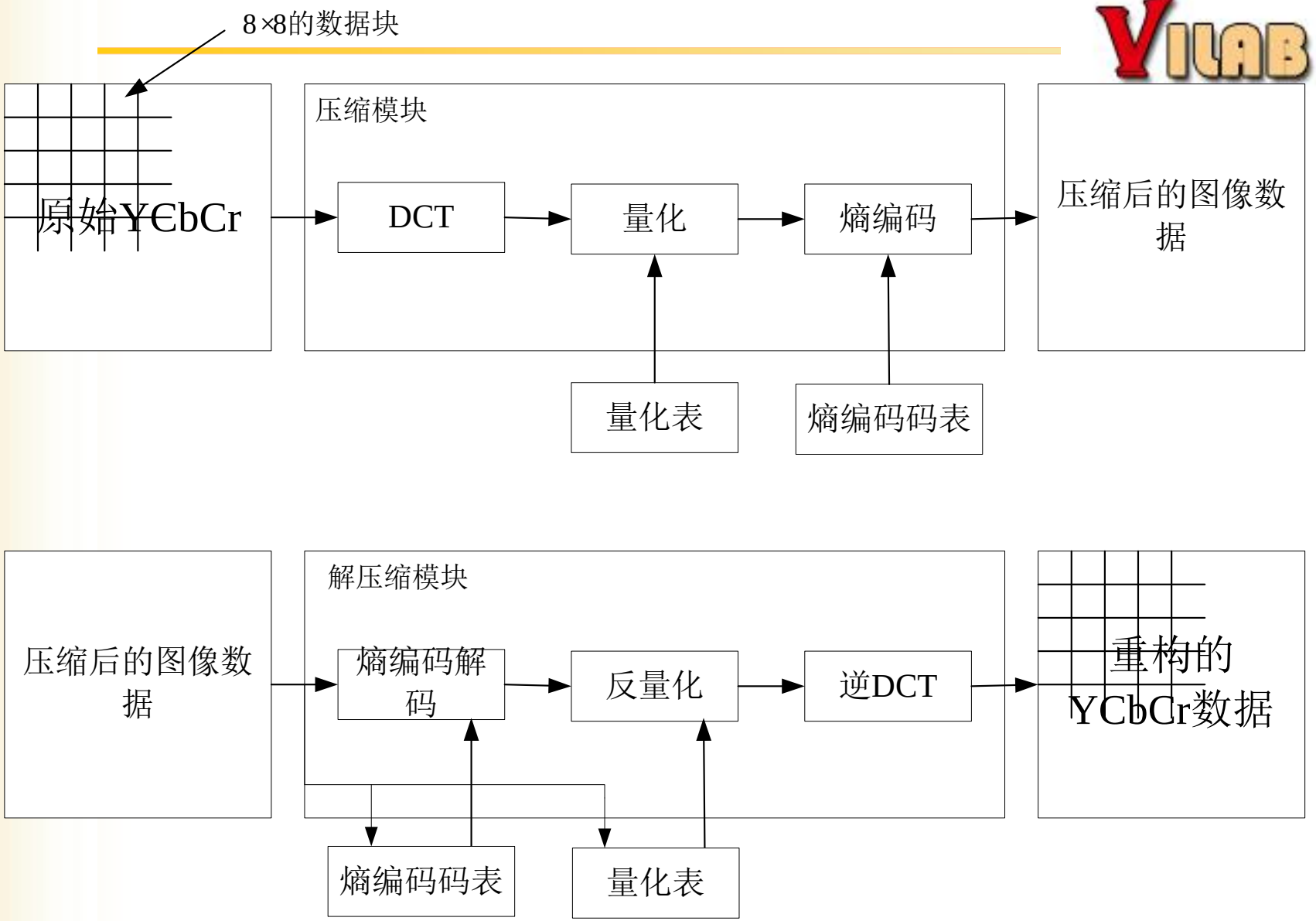
位图显示

- **BOOL StretchBlt(HDC** *hdcDest*, // handle to destination DC **int** *nXOriginDest*, // x-coord of destination upper-left corner **int** *nYOriginDest*, // y-coord of destination upper-left corner **int** *nWidthDest*, // width of destination rectangle **int** *nHeightDest*, // height of destination rectangle **HDC** *hdcSrc*, // handle to source DC **int** *nXOriginSrc*, // x-coord of source upper-left corner **int** *nYOriginSrc*, // y-coord of source upper-left corner **int** *nWidthSrc*, // width of source rectangle **int** *nHeightSrc*, // height of source rectangle **DWORD** *dwRop* // raster operation code **);**

附录：JPEG压缩流程

JPEG图像压缩的基本流程

- RGB->YCbCr
  - 注意Cb+128,Cr+128
- Y-128
- 亮度和色度分量分别分块做DCT变换
- 亮度和色度分量分别用不同的量化矩阵进行量化
  - 注意量化矩阵的计算公式
- DC系数进行处理，ZigZag扫描
- 熵编码



标准量化表和量化因子

$$QuanTable = \begin{cases} round(StdQuanTable \cdot (2 - 0.02 \cdot QualityFactor)) & QualityFactor \geq 50 \\ round(StdQuanTable \cdot (50 / QualityFactor)) & QualityFactor < 50 \end{cases}$$

JPEG推荐的标准亮度量化表

$$StdQuanTable = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}$$

附加

- 如何设计去除块效应的后处理算法
  - JPEG压缩图像随着压缩因子的增大，其块效应愈发明显，主要是由于量化造成边缘区域的失真明显不连续造成的
- 去除块效应算法可以从以下几个方面考虑：
  - 对块的边界处进行平滑处理，可以参考视频编码标准中的环路滤波(de-blocking) 利用JPEG压缩码流中的某些信息来进行补偿，从而减少失真