

数字图像处理

指导教师：胡晓雁

电子邮件：huxy@bnu.edu.cn

北京师范大学信息科学与技术学院

数字图像文件格式

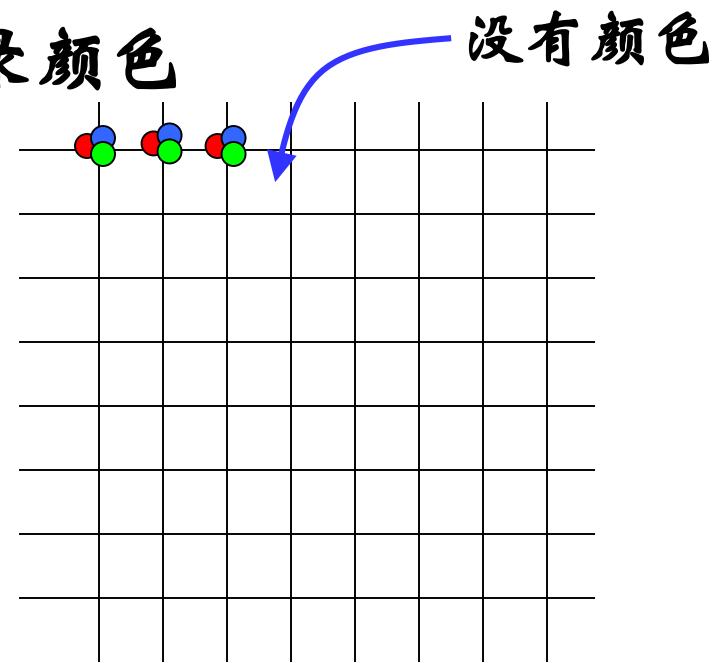
- ✓ 如何存储数字图像
- ✓ BMP文件格式
- ✓ 使用标准C++读取BMP文件

如何存储数字图像

- ✓ 数字图像文件格式:
- ✓ **BMP: Windows 位图**
- ✓ **GIF: CompuServe Gif**
- ✓ **JPEG: 静态图像专家组制定**
- ✓ **PNG: 便携网络图形**
- ✓ **PCX: PCX格式**
- ✓ **TGA: Targa**
- ✓ **TIFF: 标记图像格式**
- ✓ **PSD: PhotoShop专用格式**
- ✓ **DDS: 微软DirectX图像格式**

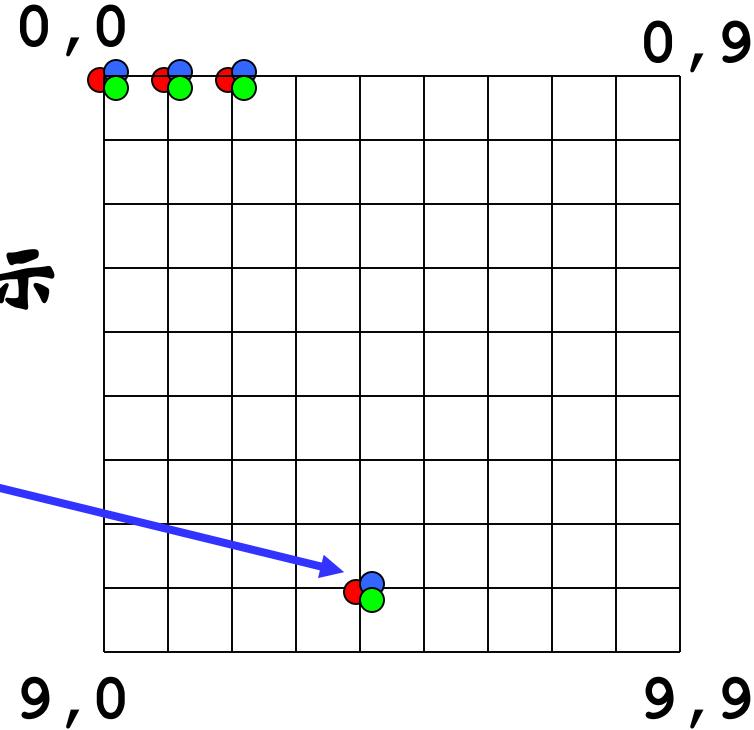
如何存储数字图像

- ✓ 图像的空间坐标和颜色值均不连续取值
- ✓ 例如，数码相机所拍摄的数字图像
- ✓ 空间坐标取值不连续
 - ✓ 一般在网格点位置上才记录颜色
- ✓ 颜色取值不连续
 - ✓ 每个网格点上的颜色取值只有有限种选择



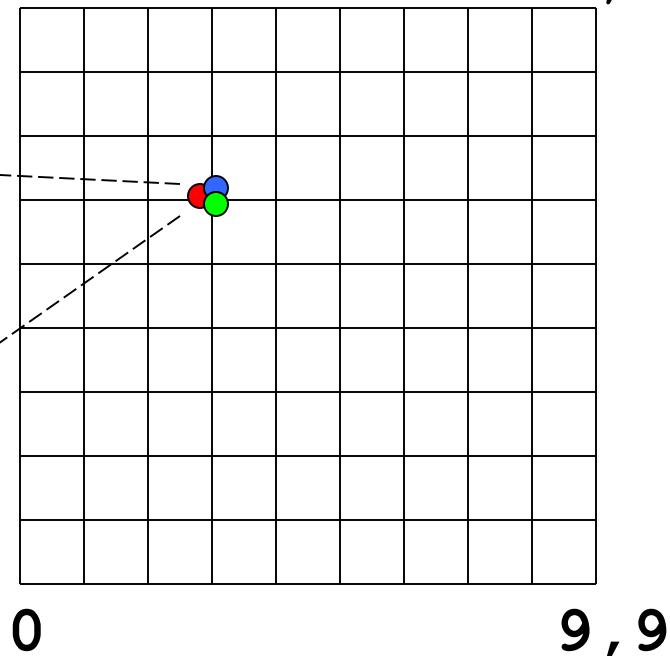
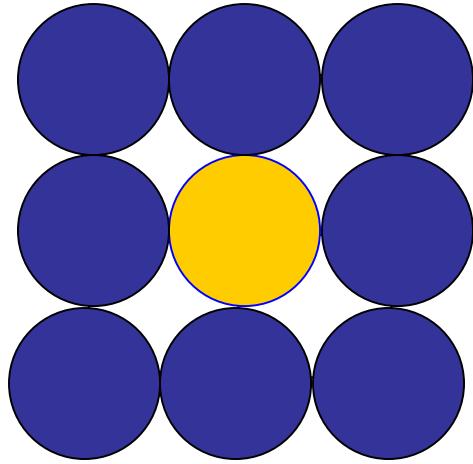
如何存储数字图像

- 右图显示的是一大小为 10×10 的数字图像
- 其中每个格点都具有一个颜色值
- 这里的格点称为像素
- $10 \times 10 = 100$ 个像素
- 我们可以用二维坐标表示
- 像素 (8, 4)



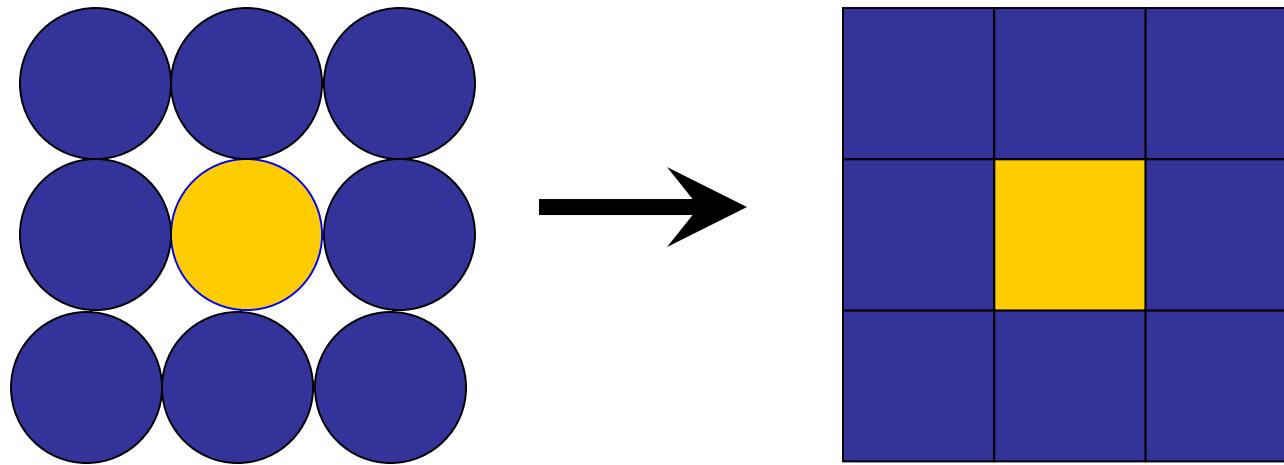
如何存储数字图像

- 当我们显示这幅数字图像时的一个问题：
- 像素有多大？
- $r < 0.5 * \text{GridSize}$
- 否则就会发生像素的重叠



如何存储数字图像

- 但是在确保不会有像素重叠的情况下，另一个问题：像素之间始终有缝隙！
- 因此，再进一步，不妨把像素想象称为一个网格大小的方块

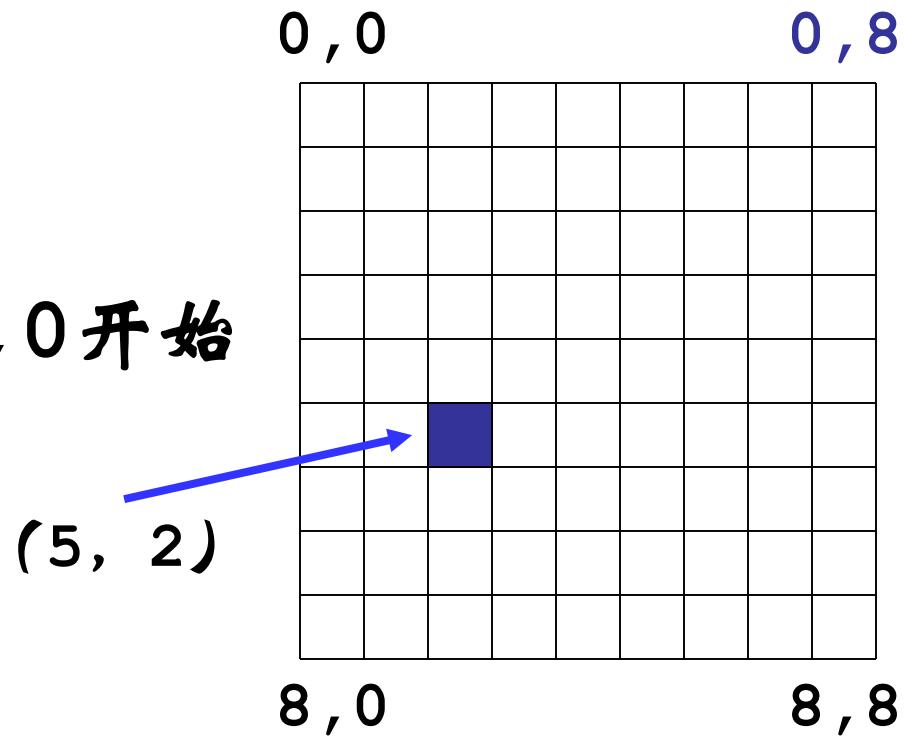






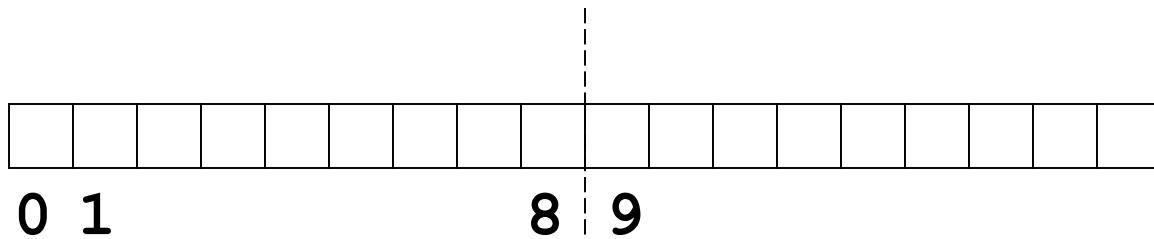
如何存储数字图像

- ✓ 注意这种表达方式与前面的细微区别
- ✓ $9 \times 9 = 81$ 个像素
- ✓ 像素坐标是二维的
- ✓ 像素坐标是离散的
- ✓ 像素坐标一般从 $0, 0$ 开始
- ✓ 类比：二维数组



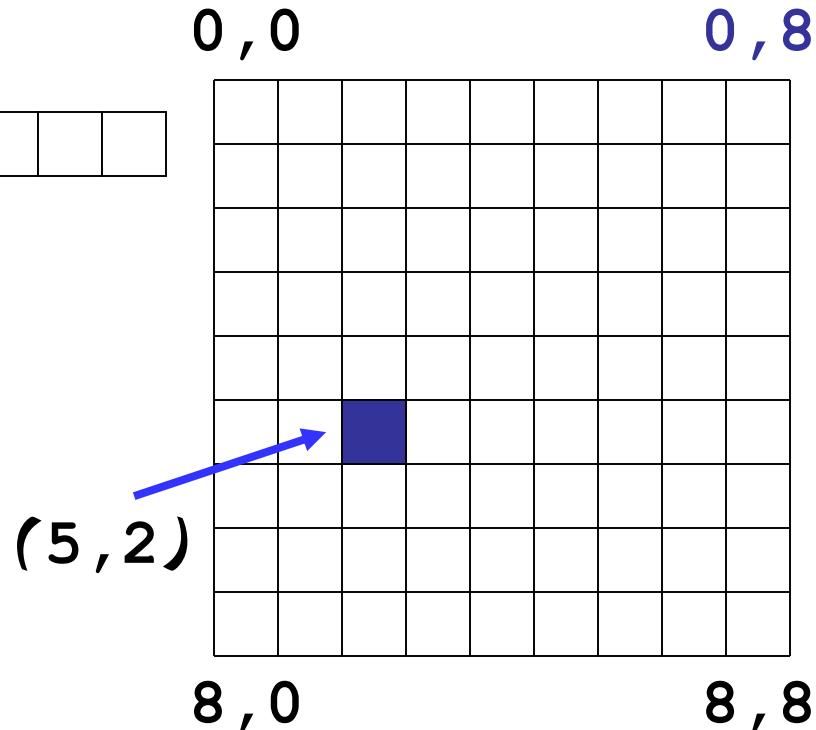
如何存储数字图像

- ✓ 计算机中的内存是一维的模型
- ✓ 因此需要将二维图像转换成一维数据



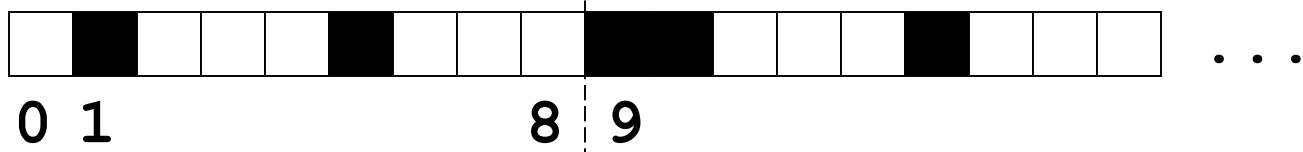
$$\text{index}(5, 2) = 5 \times 9 + 2 = 47$$

$$\text{index}(i, j) = i \times 9 + j$$



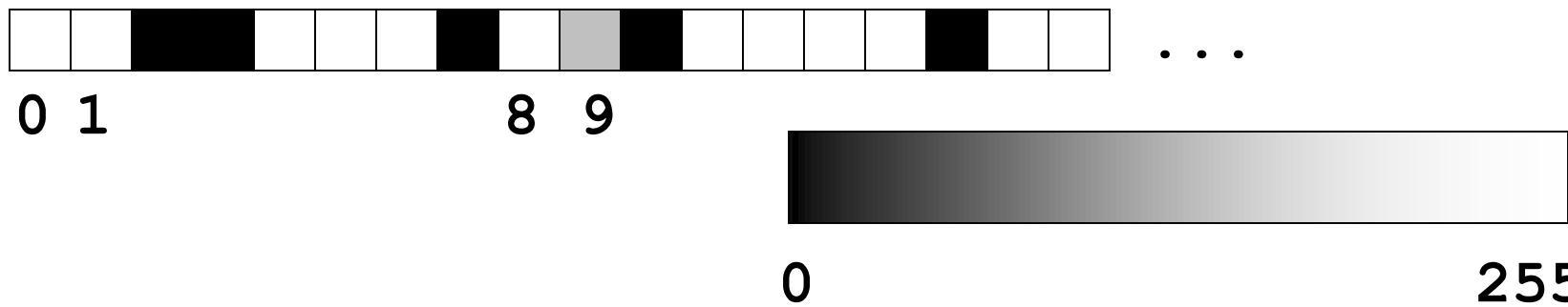
如何存储数字图像

- ✓ 如何存储像素的颜色值？
- ✓ 例如第9个像素（如不说明，均从0开始）
- ✓ 1个字节只能存储256个不同的值
- ✓ 首先考虑二值图像，0是黑色，1是白色
- ✓ 一个字节只存储0或1，这样做有点浪费
(可以更紧凑一点，让一个字节存8个像素)



如何存储数字图像

- ✓ 二值图像颜色太少了（颜色数最少的数字图像）
- ✓ 1个字节能表示0~255个不同的值，因此可以对二值图像进行一些扩充，在黑到白之间插入一些灰度级，这样就能表示灰度图像了，不同的数值代表不同的灰度级别



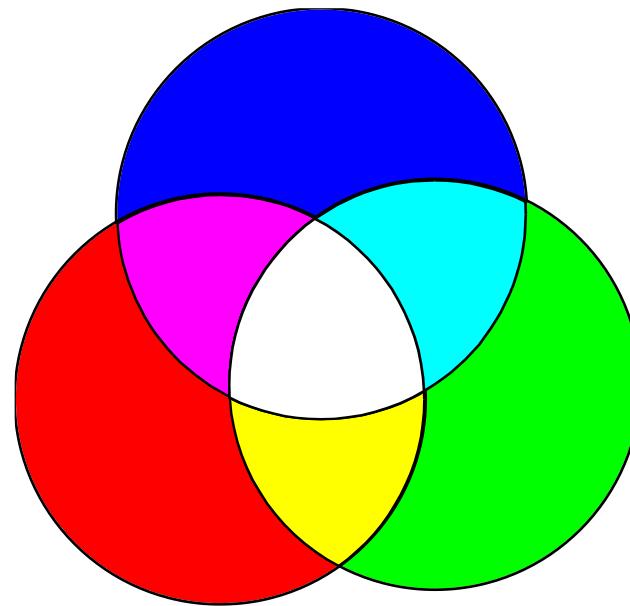
如何存储数字图像

- ✓ 即使有256种颜色，但是始终灰度图像，如何表示彩色图像？
- ✓ 这涉及到色彩的表示，有许多颜色模型：
- ✓ RGB、HSV/HSB、XYZ、CMY、CMKY...
- ✓ 我们比较熟悉的是RGB颜色模型，即红绿蓝三基色颜色模型，该模型指出颜色可以由红、绿、蓝三基色按不同比例混合而得
- ✓ 大部分数字图像都使用RGB模型表示彩色图像

如何存储数字图像

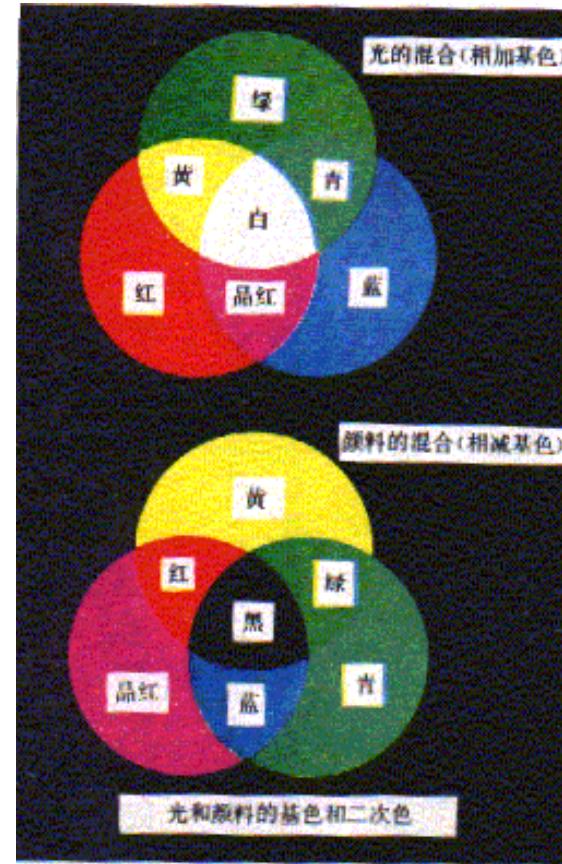
RGB三基色模型

加色系统



如何存储数字图像

- ✓ RGB的比例，设1是该基色的满比例，而0是不含该基色
- ✓ 则RGB的各颜色分量比例均在0~1之间
- ✓ 因为数字图像的颜色取值是离散的，因此不是0~1之间的所有比例值都能表示
- ✓ 例如：白色 (1: 1: 1)
黄色 (1: 1: 0)



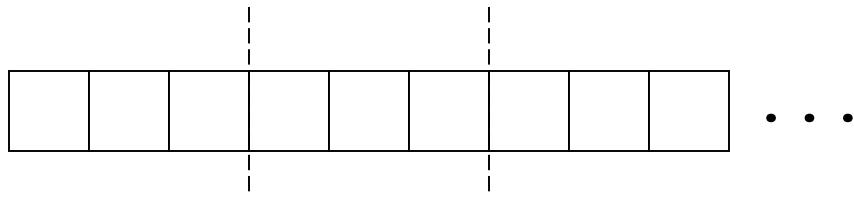
如何存储数字图像

- ✓ 一种简单的方法就是对0~1进行等分
- ✓ 我们可以用一个字节表示一个颜色分量的比例： $0/255, 1/255, \dots, 255/255$, 共有0~1的256个级别，注意，0和1的值可以取到
- ✓ 这样RGB就需要3个字节，能表示RGB混合比例 $256 \times 256 \times 256 = 2^{24}$ 种，即可以表示16777216种不同的颜色（16M色），这些颜色已经足够接近真实的色彩世界了，因此俗称这种颜色表示方法为真彩色（True Color）

如何存储数字图像

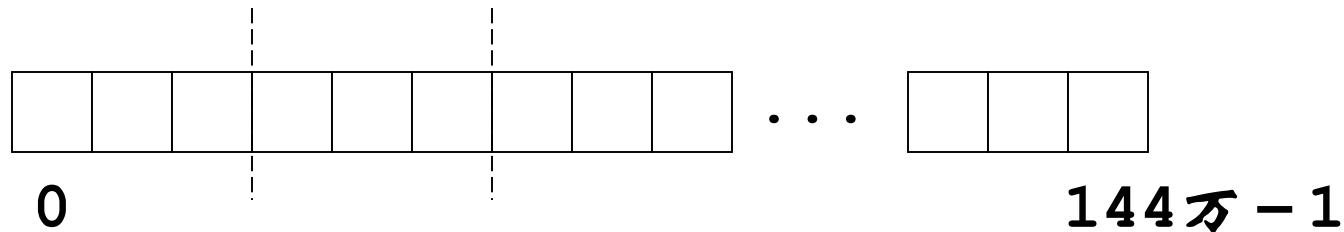
- ✓ 这样，存储一幅 9×9 的图像，需要保存 $9 \times 9 = 81$ 个像素，而每个像素需要3个字节，因此需要 $9 \times 9 \times 3 = 243$ 个字节

- ✓ 保存成一维数组，每个像素占3个字节
- ✓ 第*i*个像素占据字节：
 - ✓ $i \times 3, i \times 3 + 1, i \times 3 + 2$



如何存储数字图像

- 一幅典型的图像具有很多的像素，例如一幅图像有 800×600 个像素，因此需要保存 $800 \times 600 = 48$ 万个像素，而每个像素需要3个字节以表示色彩，因此需要存储该幅图像一共需要 $800 \times 600 \times 3 = 144$ 万个字节，约为1.3M bytes



如何存储数字图像

- 当一幅图像中没有太多的颜色，例如某幅图像只用到了几十种颜色，可以进行压缩



如何存储数字图像

- ✓ 对于实际使用的颜色数不太多的图像，可以不必对每个像素使用3个字节来表示
- ✓ 例如一幅图像，只含有不多于256种彩色颜色，此时可以把这些颜色都列举出来，进行编号，对于每个像素，只需用一个字节保存一个索引值到相应的颜色即可，这样的图像可以节约大量的存储空间
- ✓ 将所有可用的颜色列举出来的，保存在一个数组，称为“调色板”

如何存储数字图像

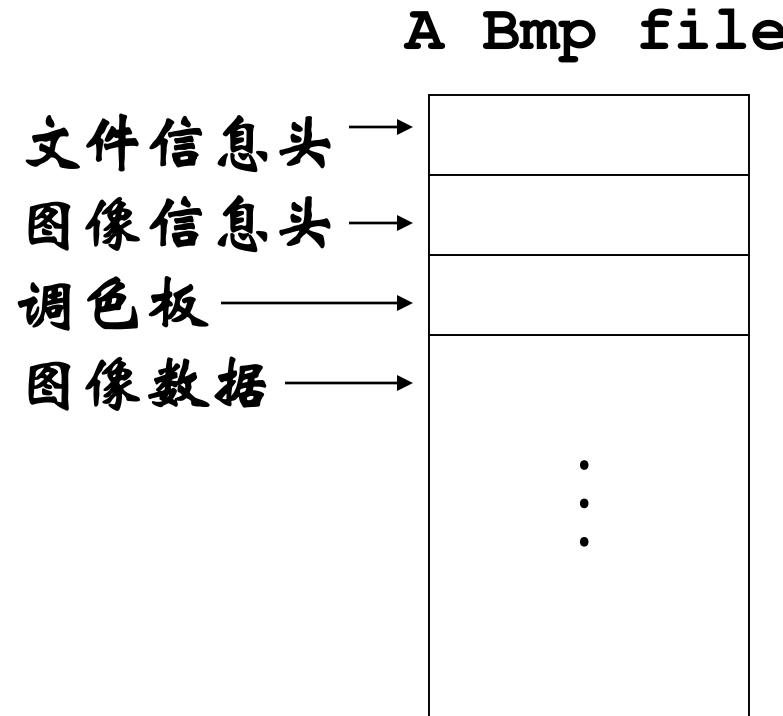
- ✓ 使用索引的方法，根据索引的取值范围，决定图像的颜色数目
- ✓ 例如使用1位作为索引，可以表示2值图像
- ✓ 使用2位索引，可以表示4色图像
- ✓ 使用4位索引，可以表示16色图像
- ✓ 使用8位索引，可以表示256色图像
- ✓ 使用2个字节，可以表示64k色？需要保存的调色板可能过大，不是很合算了

数字图像文件格式

- ✓ 如何存储数字图像
- ✓ BMP文件格式
- ✓ 使用标准C++读取BMP文件

Bmp 文件格式

- 让我们以BMP文件为例子，看看一个具体的文件格式是如何表示图像保存图像的



Bmp文件格式

文件信息头

偏移	长度	名称	描述
0	2	bfType	ASCII码 “BM”
2	4	bfSize	文件长度
6	2	bfReserved	0
8	2	bfReserved	0
10	4	bfOffbits	图像开始偏移量

注意：机器单位长度，一般为字节

Bmp文件格式

图像信息头

偏移	长度	名称	描述
14	4	biSize	信息头大小
18	4	biWidth	图像宽度
22	4	biHeight	图像高度
26	2	biPlanes	图像位平面数目
28	2	biBitCount	像素位数目

Bmp文件格式

图像信息头（接上）

偏移	长度	名称	描述
30	4	biCompression	压缩类型
34	4	biSizeImage	压缩图像大小
38	4	biXPelsPerMeter	水平分辨率
42	4	biYPelsPerMeter	垂直分辨率
46	4	biClrUsed	彩色数目
50	4	biClrImportant	“重要颜色”数目

Bmp文件格式

图像信息头（接上）

偏移	长度	名称	描述
54	$4 \times N$	bmiColors	调色板

N由biBitCount决定

BMP文件格式

- ✓ 重要信息：
- ✓ 颜色结构
- ✓ 颜芻数
- ✓ 颜色表
- ✓ 图像像素宽度和高度
- ✓ 图像数据的存储结构

BMP文件格式

- ✓ 颜色结构
- ✓ BMP文件的颜色使用RGB模型
- ✓ 不过其存储结构在颜色分量的顺序上是按照BGR顺序来存放的

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

BMP文件格式

- 颜色数
- bitCount
- 1 2^1 2种颜色 通常是黑白图像
- 4 2^4 16种颜色
- 8 2^8 256种颜色
- 24 2^{24} 真彩色 16M种颜色
- 32 2^{24} 真彩色，另有256级Alpha值

BMP文件格式

- ✓ 颜色表
- ✓ 当bitCount的值为1时，图像数据是真正的位图，即每一比特位表示一个像素；当bitCount的值为4或8时，图像数据保存的是该像素颜色的索引值，取值范围分别为0~0XF及0~0xFF
- ✓ 此时BMP在图像头后面保存调色板（颜色表），bitCount=1，保存2个RGBQUAD，=4，保存16个，=8，保存256个RGBQUAD

BMP文件格式

- ✓ 当bitCount = 24时，图像数据中每3个字节表示一个像素的颜色值
- ✓ 当bitCount = 32时，图像数据中每4个字节表示一个像素的颜色值和Alpha值
- ✓ 当bitCount等于24或32时，BMP文件中不再保存颜色表，此时图像数据紧接着图像头的内容

BMP文件格式

- ✓ 图像像素宽度和高度，BMP图像头中的宽度和高度是指图像的像素个数，
- ✓ 高度是指图像的行数
- ✓ 宽度是指每一行有多少个像素
- ✓ 图像宽度和高度合起来称为图像的分辨率或解析度或精度（Resolution）

BMP文件格式

- ✓ 图像数据的存储结构
- ✓ 图像数据存储每个像素的颜色值或者颜色索引
(根据索引在颜色表中找到相应的颜色值)
- ✓ 像素的组织方式是行优先模式，即先保存第零行的像素，再保存第一行的所有像素，依次类推...
- ✓ 像素的保存顺序：从图像的左下角开始逐行保存像素，从左到右依次保存该行的所有像素，从下到上依次保存所有行的像素

BMP文件格式

- ✓ 每一行的字节数：需要保证每一行的字节数是4的倍数，如果不到4的倍数，补充若干个0凑足字节数
- ✓ 假设某图像的一个像素占bitCounts位数据，一行共有width个像素
 - ✓ 一行占据totalBits=width×bitCounts位，
 - ✓ 一行占据nBytes=(totalBits + 7) / 8个字节
 - ✓ 对齐到4的倍数totalBytes=(nBytes+3) / 4个字节

图像数据的压缩

- ✓ 为什么要压缩图像数据？
- ✓ 800×600 的彩色图像需要存储 48 万个像素，每个像素 3 个字节，则 144 万字节，约合 1.44M 字节
- ✓ 目前数码照相机的照片分辨率普遍达到 2600×2000 ，存储这样一幅图像需要多少字节？
- ✓ 将来的分辨率将更高，一台 Canon IXUS960 相机的最高分辨率达到 4000×3000 ，这又需要多少字节？

图像数据的压缩

- ✓ 为什么图像数据可以被大幅度压缩？

	格式 (F) JPEG
	宽度 (W) 1440
	高度 (H) 900
	颜色 (C) 真彩色
	页数 (P) 1
	文件大小 (S) 91.5 KB
	未压缩大小 (U) 3.7 MB
	压缩比率 (R) 41.5

图像数据的压缩

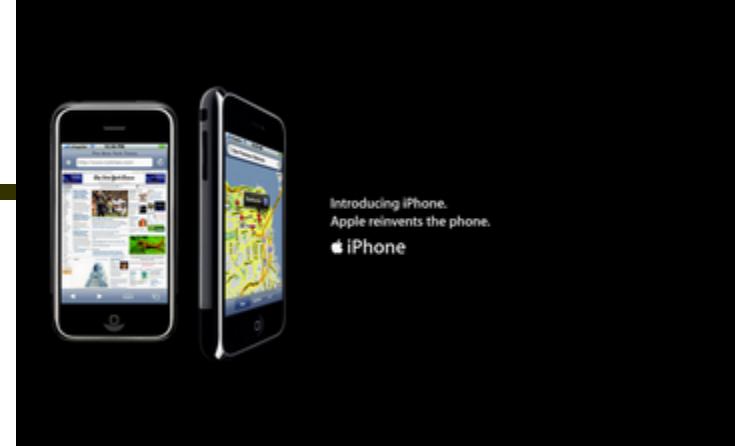
- ✓ 为什么图像数据可以被大幅度压缩？
- ✓ 图像数据中存在冗余信息，剔除掉这些冗余信息，即实现了图像数据的压缩
- ✓ 例如：图像中有大量颜色相同或者相近的像素，我们可以根据颜色信息的冗余度来设计一种简单的图像数据压缩方案
 - ✓ 实际上，使用颜色表索引本身就是一种形式的压缩
 - ✓ 行程编码 run length method

例：iPhone.BMP

24位真彩色，无调色板数据

文件头 14字节

图像头 40字节



00000000h:	42 4D B6 53 3B 00 00 00 00 00 36 00 00 00 28 00	; BM機;.....6 ...(.)
00000010h:	00 00 A0 05 00 00 84 03 00 00 01 00 18 00 00 00	; ...?..?.....
00000020h:	00 00 80 53 3B 00 00 00 00 00 00 00 00 00 00 00	; ..€s;.....
00000030h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
00000040h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
00000050h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
00000060h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
00000070h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
00000080h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
00000090h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
000000a0h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
000000b0h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
000000c0h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
000000d0h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
000000e0h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
000000f0h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
00000100h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;
00000110h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	;

例：iPhone.BMP

文件头		
Type	19778 (BM)	4D 42
Size	3888054	B6 53 3B 00
reserved1	0	00 00
reserved2	0	00 00
offset	54	36 00 00 00

例：iPhone.BMP

图像头		
size	40	28 00 00 00
width	1440	A0 05 00 00
height	900	84 03 00 00
plane	1	01 00
bitcount	24	18 00
compression	0	00 00 00 00
imageSize	3888000	80 53 3B 00
Others...	0 0 0 0	00.....00

数字图像文件格式

- ✓ 如何存储数字图像
- ✓ BMP文件格式
- ✓ 使用标准C++读取BMP文件

wingdi.h

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

wingdi.h

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

使用标准C++读取BMP文件

- ✓ BMP类：
- ✓ 封装BMP文件的相关数据结构
- ✓ 解析BMP文件
- ✓ 分配内存，装载数据
- ✓ 释放内存

使用标准C++读取BMP文件

```
struct BitMapFileHeader
{
    // unsigned short bfType; moved to
    // class BitMap
    unsigned int    bfSize;
    unsigned short  bfReserved1;
    unsigned short  bfReserved2;
    unsigned int    bfOffBits;
};
```

使用标准C++读取BMP文件

```
struct BitMapInfoHeader
{
    unsigned int biSize;
    int biWidth;
    int biHeight;
    unsigned short biPlanes;
    unsigned short biBitCount;
    unsigned int biCompression;
    unsigned int biSizeImage;
    int biXPelsPerMeter;
    int biYPelsPerMeter;
    unsigned int biClrUsed;
    unsigned int biClrImportant;
};
```

使用标准C++读取BMP文件

```
struct RGBQuad
{
    unsigned char rgbBlue;
    unsigned char rgbGreen;
    unsigned char rgbRed;
    unsigned char rgbReserved;
};
```

使用标准C++读取BMP文件

```
class BitMap
{
public:
    enum DataType{ UNKNOW, RGB, RGBA,
FORMATTED_RGB } ;
    DataType dataType;

    unsigned short bmfHeader_bfType;
    struct BitMapFileHeader bmfHeader;
    struct BitMapInfoHeader bmiHeader;
    struct RGBQuad * pPalette;
    unsigned char * pData;
    . . . .
};
```

使用标准C++读取BMP文件

```
BitMap () : dataType (UNKNOW) ,  
             pPalette (NULL) ,  
             pData (NULL)  
{  
    resetBMFHeader ();  
    resetBMIHeader ();  
}  
  
~BitMap ()  
{  
    releasePalette ();  
    releaseImageData ();  
}
```

使用标准C++读取BMP文件

```
class BitMap
{
    .....
    int loadBmpFile(const string& filename);

    inline void resetBMFHeader();
    inline void resetBMIHeader();
    inline void releasePalette();
    inline void releaseImageData();
    inline void releaseData();

};
```

使用标准C++读取BMP文件

✓ 代码说明

- ✓ 结构的特殊处理 `DataType`
- ✓ 初始化列表 `BitMap()`
- ✓ 类中重复功能的处理 `resetBMFHeader`

使用标准C++读取BMP文件

- ✓ //bfType is moved into BitMap class, as a member :
unsigned short bmfHeader_bfType. see declaration
of class BitMap
- ✓ //the reason is that bfType should be size of 2
bytes according to BMP file specification, but in
Visual C ++, the
- ✓ //default alignment size of struct is 8 bytes.
two ways to address this
- ✓ //1: set project setting in VC++ -> C/C++ -> Code
Generation -> struct member alignment -> 1 Byte
(Zp1)
- ✓ //2: in code, add "#pragma pack (2)" before
"struct BitMapFileHeader"
- ✓ //both methods are not clear for teaching
consideration so I just drag bfType out of the
struct BitMapFileHeader

使用标准C++读取BMP文件

- ✓ 初始化列表，基类对象，对象成员，常量成员进行初始化的理想之处

使用标准C++读取BMP文件

- 类中重复功能的处理 `resetBMFHeader`

使用标准C++读取BMP文件

```
inline void BitMap::resetBMFHeader()
{
    bmfHeader_bfType = 0;
    bmfHeader.bfSize = 0;
    bmfHeader.bfReserved1 = 0;
    bmfHeader.bfReserved2 = 0;
    bmfHeader.bfOffBits = 0;
}
```

使用标准C++读取BMP文件

```
inline void BitMap::resetBMHeader()
{
    bmiHeader.biSize=0;
    bmiHeader.biWidth=0;
    bmiHeader.biHeight=0;
    bmiHeader.biPlanes=0;
    bmiHeader.biBitCount=0;
    bmiHeader.biCompression=0;
    bmiHeader.biSizeImage=0;
    bmiHeader.biXPelsPerMeter=0;
    bmiHeader.biYPelsPerMeter=0;
    bmiHeader.biClrUsed=0;
    bmiHeader.biClrImportant=0;
}
```

使用标准C++读取BMP文件

```
inline void BitMap::releasePalette()
{
    if(pPalette)
    {
        delete pPalette;
        pPalette = NULL;
    }
}
```

使用标准C++读取BMP文件

```
inline void BitMap::releaseImageData()
{
    if(pData)
    {
        delete pData;
        pData = NULL;
    }
}
```

使用标准C++读取BMP文件

```
inline void BitMap::releaseData()
{
    dataType = UNKNOW;

    resetBMFHeader();
    resetBMIHeader();
    releasePalette();
    releaseImageData();
}
```

loadBmpFile

- ✓ `int BitMap::loadBmpFile(const string& filename)`
- ✓ 返回值，0表示成功，其他表示出错
- ✓ 参数，标准库中的字符串类，需要包含
`#include <string>`

loadBmpFile

```
int BitMap::loadBmpFile(const string&  
filename)  
{  
    打开文件；  
    释放数据，如果有的话()；  
    读取文件内容；  
    根据颜色数解析图像数据；  
}
```

loadBmpFile—打开文件

```
//ifstream bmpfile(filename.c_str());
ifstream bmpfile;
bmpfile.open(filename.c_str(), ios::binary);

if (!bmpfile) // can't open file filename
{
    return -1;
}
```

loadBmpFile——释放数据，如果有的话

```
// release previous data if any;  
releaseData();
```

loadBmpFile——读取文件内容

```
bmptime.read( (char*) &bmfHeader_bfType,  
              sizeof(bmfHeader_bfType));  
  
//not a bmp file  
if (bmfHeader_bfType != 0x4D42 )  
{  
    return -1;  
}
```

loadBmpFile——读取文件内容

```
// read file header  
bmpfile.read( (char*) &bmfHeader,  
               sizeof(BitMapFileHeader));
```

loadBmpFile——读取文件内容

```
// read info header  
bmpfile.read((char*)&bmiHeader,  
             sizeof(BitMapInfoHeader));  
  
int imageSize = bmfHeader.bfSize -  
               bmfHeader.bfOffBits;
```

loadBmpFile—读取文件内容

```
int imageSize = bmfHeader.bfSize -  
    bmfHeader.bfOffBits;
```

```
int paletteSize = bmfHeader.bfOffBits -  
    sizeof(BitMapFileHeader) -  
    sizeof(BitMapInfoHeader) - 2;
```

```
pPalette = new RGBQuad  
[paletteSize/sizeof(RGBQuad)];
```

loadBmpFile——读取文件内容

```
switch (bmiHeader.biBitCount)
{
case 1: // monoColor Image
case 4: // 16 Color Image
case 8: // 256 Color Image
case 16: // 16 Bits, 64k Colors Image
case 24: // True Color Image
default:
    return -1; // Not support now
}
```

loadBmpFile—读取文件内容

```
case 24: // True Color Image  
    pData = new unsigned char [imageSize];  
    bmpfile.read((char*)pData, imageSize);  
    dataType = RGB;  
    break;
```

使用BitMap类

```
#include "bmp.h"

int main()
{
    BitMap bmpImage;
    bmpImage.loadBmpFile("iPhone.bmp");
}
```