AVI视频格式

AVI简介

- 微软AVI（ Audio Video Interleave ）文件格式使用RIFF规范定义，AVI由应用程序捕获、编辑和回放音-视频序列。
- 通常，AVI文件包含多个不同数据类型的流。多数AVI序列同时使用音频和视频流。一个简单的AVI只包含视频数据不包含音频数据。
- AVI类型
  - AVI 1.0
  - Open-DML 1996年发布1.02
  - Hybride-Files

- RIFX文件格式与RIFF文件格式是相同的，除了所有多字节按Motorola字节顺序（大端机）。

Tip: RIFX

AVI Data

- 'RIFF' fileSize fileType (data)
- data包含chunks和lists

dwFourCC dwSize Data

Data常常被补齐到最近的WORD边界。 dwSize包含有效数据大小，不包含补齐部分的大小

```
typedef struct {
DWORD dwFourCC //(dc = video, wb
= audio, tx = text)
DWORD dwSize
BYTE data[dwSize] // 包含 headers或video/audio数据
} CHUNK;
```

CHUNK

- FourCC全称Four-Character Codes，是由4个字符（4 bytes）组成，是一种独立标示视频数据流格式的四字节
- FourCC是在编程中非常常用的东西，一般用作标示符。比如wav、avi等RIFF文件的标签头标示，Quake 3的模型文件.md3中也大量存在等于“IDP3”的FOURCC。它是一个32位的标示符

Tips: FourCC

'LIST' dwSize dwFourCC Data
dwSize包括dwFourCC和Data的大小。

```
typedef struct {
DWORD dwList //'LIST'
DWORD dwSize
DWORD dwFourCC // LIST类型，如’hdrl’
BYTE Data[dwSize-4] //包含Lists或Chunks
} LIST;
```

LIST

AVI主要的LIST和CHUNK

- ‘hdrl’ LIST:定义了数据格式和第一个必需的LIST
- ‘movi’ LIST: 包含了AVI序列的数据和第2个必需的LIST

```
RIFF ('AVI '
      LIST ('hdrl' ... )
      LIST ('movi' ... )
      ['idx1' (<AVI Index>) ]
     )
```

- ‘idx1’ CHUNK:包含的索引
- AVI文件必须以一定顺序定义以上三个分量。

```
RIFF ('AVI '
      LIST ('hdrl'
            'avih'(<Main AVI Header>)
            LIST ('strl'
                  'strh'(<Stream header>)
                  'strf'(<Stream format>)
                  [ 'strd'(<Additional header data>) ]
                  [ 'strn'(<Stream name>) ]
                  ...
                 )
             ...
           )
      LIST ('movi'
            {SubChunk | LIST ('rec '
                              SubChunk1
                              SubChunk2
                              ...
                             )
               ...
            }
            ...
           )
      ['idx1' (<AVI Index>) ]
     )
```

AVI主头部

- ‘hdrl’ list以 ‘avih’ chunk开始。avih表示了主头部包含了AVI的一些全局信息，例如：流的数量，AVI序列的高与宽.

- 该chunk定义为AVIMAINHEADER结构

```
typedef struct _avimainheader {
  FOURCC fcc;
  DWORD  cb;
  DWORD  dwMicroSecPerFrame;
  DWORD  dwMaxBytesPerSec;
  DWORD  dwPaddingGranularity;
  DWORD  dwFlags;
  DWORD  dwTotalFrames;
  DWORD  dwInitialFrames;
  DWORD  dwStreams;
  DWORD  dwSuggestedBufferSize;
  DWORD  dwWidth;
  DWORD  dwHeight;
  DWORD  dwReserved[4];
} AVIMAINHEADER;
```

AVIMAINHEADER

AVI流的头部

- 主头部后跟着1个或多个‘strl’ list，第一个‘strl’ chunk对应流0#，第2个对应stream1#
- 一个 ‘strl’ list是每个数据流所必须的
- 每个 ‘strl’ list包含文件中流的信息，必须包含1个流的头chunk (‘strh’) 和1个流的格式chunk ('strf‘)
- 此外，1个 ‘strl’ list可能包含一个流-头部数据chunk (‘strd’)和流的名字chunk ('strn')
- 头chunk (‘strh’) 由结构**AVISTREAMHEADER**定义

```
typedef struct _avistreamheader {
  FOURCC fcc;
  DWORD  cb;
  FOURCC fccType;
  FOURCC fccHandler;
  DWORD  dwFlags;
  WORD   wPriority;
  WORD   wLanguage;
  DWORD  dwInitialFrames;
  DWORD  dwScale;
  DWORD  dwRate;
  DWORD  dwStart;
  DWORD  dwLength;
  DWORD  dwSuggestedBufferSize;
  DWORD  dwQuality;
  DWORD  dwSampleSize;
  struct {
    short int left;
    short int top;
    short int right;
    short int bottom;
  } rcFrame;
} AVISTREAMHEADER;
```

AVISTREAMHEADER

- MRLE - Microsoft Run Length Encoding
- Bitmap

Tips: mrle

Format CHUNK

- 一个流的格式chunk（‘strf’）必须跟在流的头chunk
- ‘strf’定义了流中的数据格式
- 该chunk中的数据依赖于流的格式，对于视频流来说，该信息为 **BITMAPINFO** 结构；对于音频流，来说该信息为 **WAVEFORMATEX** 结构

一、BMP文件结构

1. BMP文件组成

BMP文件由文件头、位图信息头、颜色信息和图形数据四部分组成。

文件头 主要包含文件的大小、文件类型、图像数据偏离文件头的长度等信息；

位图信息头 包含图像的尺寸信息、图像用几个比特数值来表示一个像素、图像是否压缩、图像所用的颜色数等信息。

颜色信息 包含图像所用到的颜色表，显示图像时需用到这个颜色表来生成调色板，但如果图像为真彩色，既图像的每个像素用24个比特来表示，文件中就没有这一块信息，也就不需要操作调色板。

文件中的数据块 表示图像的相应的像素值，需要注意的是：图像的像素值在文件中的存放顺序为从左到右，从下到上，也就是说，在BMP文件中首先存放的是图像的最后一行像素，最后才存储图像的第一行像素，但对与同一行的像素，则是按照先左边后右边的的顺序存储的；

另外,文件存储图像的每一行像素值时，如果存储该行像素值所占的字节数为4的倍数，则正常存储，否则，需要在后端补0，凑足4的倍数。

2. BMP文件头

BMP文件头数据结构含有BMP文件的类型、文件大小和位图起始位置等信息。其结构定义如下:

```
typedef struct tagBITMAPFILEHEADER
{
  WORD bfType; // 位图文件的类型，必须为“BM”
  DWORD bfSize; // 位图文件的大小，以字节为单位
  WORD bfReserved1; // 位图文件保留字，必须为0
  WORD bfReserved2; // 位图文件保留字，必须为0
  DWORD bfOffBits; // 位图数据的起始位置，以相对于位图文件头的偏
                   移量表示，以字节为单位
} BITMAPFILEHEADER；该结构占据14个字节。
```

3. 位图信息头

BMP位图信息头数据用于说明位图的尺寸等信息。其结构如下：

```
typedef struct tagBITMAPINFOHEADER
{
  DWORD biSize; // 本结构所占用字节数
  LONG biWidth; // 位图的宽度，以像素为单位
  LONG biHeight; // 位图的高度，以像素为单位
  WORD biPlanes; // 目标设备的平面数不清，必须为1
   WORD biBitCount// 每个像素所需的位数，必须是1(双色), 4(16色)，8(256色)或24(真彩色)
    DWORD biCompression; // 位图压缩类型，必须是 0(不压缩),1(BI_RLE8压缩类 型)或2(BI_RLE4压缩类型)之一
  DWORD biSizeImage; // 位图的大小，以字节为单位
  LONG biXPelsPerMeter; // 位图水平分辨率，每米像素数
  LONG biYPelsPerMeter; // 位图垂直分辨率，每米像素数
  DWORD biClrUsed;// 位图实际使用的颜色表中的颜色数
  DWORD biClrImportant;// 位图显示过程中重要的颜色数
} BITMAPINFOHEADER；该结构占据40个字节。
```

注意：对于BMP文件格式，在处理单色图像和真彩色图像的时候，无论图像数据多么庞大，都不对图像数据进行任何压缩处理，一般情况下，如果位图采用压缩格式，那么16色图像采用RLE4压缩算法，256色图像采用RLE8压缩算法。

4. 颜色表

颜色表用于说明位图中的颜色，它有若干个表项，每一个表项是一个RGBQUAD类型的结构，定义一种颜色。

RGBQUAD结构的定义如下:

```
typedef struct tagRGBQUAD
{
  BYTErgbBlue;// 蓝色的亮度(值范围为0-255)
  BYTErgbGreen; // 绿色的亮度(值范围为0-255)
  BYTErgbRed; // 红色的亮度(值范围为0-255)
  BYTErgbReserved;// 保留，必须为0
} RGBQUAD;
```

颜色表中**RGBQUAD**结构数据的个数由**BITMAPINFOHEADER** 中的**biBitCount**项来确定，当**biBitCount=1,4,8**时，分别有**2,16,256**个颜色表项，当**biBitCount=24**时，图像为真彩色，图像中每个像素的颜色用三个字节表示，分别对应**R**、**G**、**B**值，图像文件没有颜色表项。位图信息头和颜色表组成位图信息，**BITMAPINFO**结构定义如下:

```
typedef struct tagBITMAPINFO
{
        BITMAPINFOHEADER bmiHeader; // 位图信息头
        RGBQUAD bmiColors[1]; // 颜色表
} BITMAPINFO;
```

注意：**RGBQUAD**数据结构中，增加了一个保留字段**rgbReserved**，它不代表任何颜色，必须取固定的值为“**0**”，同时，**RGBQUAD**结构中定义的颜色值中，红色、绿色和蓝色的排列顺序与一般真彩色图像文件的颜色数据排列顺序恰好相反，既：若某个位图中的一个像素点的颜色的描述为“**00**，**00**，**ff**，**00**”，则表示该点为红色，而不是蓝色。

5. 位图数据

位图数据记录了位图的每一个像素值或该对应像素的颜色表的索引值，图像记录顺序是在扫描行内是从左到右,扫描行之间是从下到上。这种格式我们又称为Bottom_Up位图，当然与之相对的还有Up_Down形式的位图，它的记录顺序是从上到下的，对于这种形式的位图，也不存在压缩形式。位图的一个像素值所占的字节数：

当biBitCount=1时，8个像素占1个字节；
当biBitCount=4时，2个像素占1个字节；
当biBitCount=8时，1个像素占1个字节；
当biBitCount=24时,1个像素占3个字节，此时图像为真彩色图像。

当图像不是为真彩色时，图像文件中包含颜色表，位图的数据表示对应像素点在颜色表中相应的索引值，当为真彩色时，每一个像素用三个字节表示图像相应像素点彩色值，每个字节分别对应R、G、B分量的值，这时候图像文件中没有颜色表。上面我已经讲过了，Windows规定图像文件中一个扫描行所占的字节数必须是4的倍数(即以字为单位),不足的以0填充，图像文件中一个扫描行所占的字节数计算方法：

DataSizePerLine= (biWidth* biBitCount+31)/8；// 一个扫描行所占的字节数

位图数据的大小按下式计算(不压缩情况下)：

DataSize= DataSizePerLine* biHeight。

- Encode mode
- Absolute mode

BI_RLE4/8

Compressed data	Expanded data
03 04	04 04 04
05 06	06 06 06 06 06
00 03 45 56 67 00	45 56 67
02 78	78 78
00 02 05 01	Move 5 right and 1 down
02 78	78 78
00 00	End of line
09 1E	1E 1E 1E 1E 1E 1E 1E 1E 1E
00 01	End of RLE bitmap

BI_RLE8

Compressed data	Expanded data
03 04	0 4 0
05 06	0 6 0 6 0
00 06 45 56 67 00	4 5 5 6 6 7
04 78	7 8 7 8
00 02 05 01	Move 5 right and 1 down
04 78	7 8 7 8
00 00	End of line
09 1E	1 E 1 E 1 E 1 E 1
00 01	End of RLE bitmap

BI_RLE4

- ‘strn’ chunk包含一些有关该流的文本描述

strn CHUNK

JUNK

- AVI数据对齐需插入'JUNK' chunks
- 应用程序不需要处理 'JUNK' chunk.

流数据

- ‘movi’ List包含流中的实际数据，例如视频帧和音频采样，或被分组到‘rec ’ LIST中。

Two-character code	Description
db	Uncompressed video frame
dc	Compressed video frame
pc	Palette change
wb	Audio data

- 可选的(‘idx1’) chunk跟在 ‘movi’ list后
- 索引包含数据chunks列表和它们在文件中的位置
- 每个数据chunk的索引由**AVIOLDINDEX**结构定义
- 如果文件含有索引，需要设置**AVIMAINHEADER**结构的**dwFlags**为 AVIF_HASINDEX flag.

idx1 CHUNK

```
typedef struct _avioldindex {
  FOURCC fcc;
  DWORD  cb;
  struct _avioldindex_entry {
    DWORD dwChunkId;
    DWORD dwFlags;
    DWORD dwOffset;
    DWORD dwSize;
  } aIndex[];
} AVIOLDINDEX;
```

idx1 CHUNK