# XviD 应用编程接口（API）简介（v0.1）

（陈云川 ybc2084@163.com UESTC,CD 2007 年 4 月 23 日）

## 0 序

XviD 是一款开源的 MPEG-4 视频编解码器。XviD 的 API 接口定义得非常清晰，其三个核心接口函数（xvid_global()，xvid_decore()和 xvid_encore()）和插件函数都具有统一的形式，大大简化了程序员的工作。本文将简单介绍 XviD 的三个核心 API 接口函数，对于 Xvid 的插件函数未做说明，计划在本文下一版中增加。

## 1 版本

XviD 的版本号定义为$major.$minor.$patch 的形式。当版本更新时：

- 如果 API 接口没有发生改变，则增加$patch；
- 如果 API 接口发生了改变，但仍然向后兼容，则增加$minor；
- 如果 API 接口发生了重大变化，则增加$major。

很多 XviD 结构体都包含一个 version 成员，用于指定所采用的 XviD 的版本。正确的初始化方法是：先将结构体全部清空为 0，然后再设置 version 成员。如下所示：

```
memset(&struct,0,sizeof(struct));
struct.version = XVID_VERSION;
```

与 XviD 版本有关的宏定义为：

```
#define XVID_MAKE_VERSION(a,b,c)   ((((a)&0xff)<<16) | (((b)&0xff)<<8) | ((c)&0xff))
#define XVID_VERSION_MAJOR(a)      ((char)(((a)>>16) & 0xff))
#define XVID_VERSION_MINOR(a)      ((char)(((a)>> 8) & 0xff))
#define XVID_VERSION_PATCH(a)      ((char)(((a)>> 0) & 0xff))

#define XVID_MAKE_API(a,b)         ((((a)&0xff)<<16) | (((b)&0xff)<<0))
#define XVID_API_MAJOR(a)          (((a)>>16) & 0xff)
#define XVID_API_MINOR(a)          (((a)>> 0) & 0xff)

#define XVID_VERSION               XVID_MAKE_VERSION(1,1,2)
#define XVID_API                   XVID_MAKE_API(4, 1)
```

XviD 中存在两个版本，一个是 XviD 库本身的版本，即 XVID_VERSION；另一个是

XviD 的 API 接口的版本，即 XVID_API。XVID_VERSION 定义为一个 24 位的无符号整数；XVID_API 定义为一个 16 位的无符号整数。

XVID_MAKE_VERSION(a,b,c)宏的作用是生成 XVID_VERSION，其中 a、b、c 分别是 XviD 版本号中的$major、$minor 和$patch。

XVID_VERSION_MAJOR(a) 用于从 XVID_VERSION 中抽取出 $major ；XVID_VERSION_MINOR(a) 用于从 XVID_VERSION 中抽取出 $minor ；XVID_VERSION_PATCH(a)用于从 XVID_VERSION 中抽取出$patch。

# 2 错误码

所有 XviD 函数都以返回小于 0 的值表明出现了错误。定义的错误码如下：

```
#define XVID_ERR_FAIL      -1    /* general fault */
#define XVID_ERR_MEMORY    -2    /* memory allocation error */
#define XVID_ERR_FORMAT    -3    /* file format error */
#define XVID_ERR_VERSION   -4    /* structure version not supported */
#define XVID_ERR_END       -5    /* encoder only; end of stream reached */
```

XVID_ERR_FAIL 表示一般性错误；XVID_ERR_MEMORY 表示内存分配错误；XVID_ERR_FORMAT 表示文件格式错误；XVID_ERR_VERSION 表明程序中使用了不再被支持的结构体版本；XVID_ERR_END 是编码器独有的错误码，表示已经到达流末尾。

# 3 色场空间

XviD 中定义的色场空间如下：

```
#define XVID_CSP_PLANAR   (1<< 0)  /* 4:2:0 planar (==I420, except for pointers/strides) */
#define XVID_CSP_USER     XVID_CSP_PLANAR
#define XVID_CSP_I420     (1<< 1)  /* 4:2:0 planar */
#define XVID_CSP_YV12     (1<< 2)  /* 4:2:0 planar */
#define XVID_CSP_YUY2     (1<< 3)  /* 4:2:2 packed */
#define XVID_CSP_UYVY     (1<< 4)  /* 4:2:2 packed */
#define XVID_CSP_YVYU     (1<< 5)  /* 4:2:2 packed */
#define XVID_CSP_BGRA     (1<< 6)  /* 32-bit bgra packed */
#define XVID_CSP_ABGR     (1<< 7)  /* 32-bit abgr packed */
#define XVID_CSP_RGBA     (1<< 8)  /* 32-bit rgba packed */
#define XVID_CSP_ARGB     (1<<15)  /* 32-bit argb packed */
#define XVID_CSP_BGR      (1<< 9)  /* 24-bit bgr packed */
#define XVID_CSP_RGB555   (1<<10)  /* 16-bit rgb555 packed */
#define XVID_CSP_RGB565   (1<<11)  /* 16-bit rgb565 packed */
```

```
#define XVID_CSP_SLICE      (1<<12) /* decoder only: 4:2:0 planar, per slice rendering */
#define XVID_CSP_INTERNAL   (1<<13) /* decoder only: 4:2:0 planar, returns ptrs to internal buffers */
#define XVID_CSP_NULL       (1<<14) /* decoder only: dont output anything */
#define XVID_CSP_VFLIP      (1<<31) /* vertical flip mask */
```

尽管 XviD 定义的色场空间比较多，但实际上常用的只有 XVID_CSP_I420、XVID_CSP_YV12、XVID_CSP_BGR、XVID_CSP_RGB555、XVID_CSP_RGB565 这样几种。

作者个人的观点：XVID_CSP_PLANAR、XVID_CSP_USER、XVID_CSP_I420、XVID_CSP_YV12 都属于 YUV4:2:0 色场空间，采用 YUV 平面格式存放；XVID_CSP_YUY2、XVID_CSP_UYVY、XVID_CSP_YVYU 都属于 YUV4:2:2 色场空间，采用 YUV 紧缩格式存放。XviD 似乎没有提供对 YUV4:4:4 和 YUV4:1:1 色场空间的支持——也可能是 MPEG-4 标准不支持。无论如何，常用的 YUV 色场空间是 YUV4:2:0。

# 4 profile 和 level 定义

XviD 中定义了对三种 MPEG-4 Profile@Level 组合的支持：Simple（S）、Advanced Realtime Simple（ARTS）、Advanced Simple（AS）。

```
#define XVID_PROFILE_S_L0       0x08        /* simple */
#define XVID_PROFILE_S_L1       0x01
#define XVID_PROFILE_S_L2       0x02
#define XVID_PROFILE_S_L3       0x03
#define XVID_PROFILE_ARTS_L1    0x91        /* advanced realtime simple */
#define XVID_PROFILE_ARTS_L2    0x92
#define XVID_PROFILE_ARTS_L3    0x93
#define XVID_PROFILE_ARTS_L4    0x94
#define XVID_PROFILE_AS_L0      0xf0        /* advanced simple */
#define XVID_PROFILE_AS_L1      0xf1
#define XVID_PROFILE_AS_L2      0xf2
#define XVID_PROFILE_AS_L3      0xf3
#define XVID_PROFILE_AS_L4      0xf4
```

XviD 没有定义对 MPEG-4 中其它 Profile@Level 组合的支持。

# 5 像素幅型比（Pixel Aspect Ration）

像素幅型比表示的是屏幕上一个像素点的宽度和高度的比值，简写为 PAR（即 Pixel Aspect Ratio 的缩写）。XviD 定义了对如下几种标准的像素幅型比的支持：

```
#define XVID_PAR_11_VGA      1   /* 1:1 vga (square), default if supplied PAR is n
ot a valid value */
#define XVID_PAR_43_PAL      2   /* 4:3 pal (12:11 625-line) */
#define XVID_PAR_43_NTSC     3   /* 4:3 ntsc (10:11 525-line) */
#define XVID_PAR_169_PAL     4   /* 16:9 pal (16:11 625-line) */
#define XVID_PAR_169_NTSC    5   /* 16:9 ntsc (40:33 525-line) */
#define XVID_PAR_EXT         15  /* extended par; use par_width, par_height */
```

视频应用中常见的像素幅型比有两种：4:3 和 16:9，后者更具有剧场效果。PAL 和 NTSC 是两种电视标准（另外一种电视标准是 SECAM），这两种电视标准都是隔行扫描时代的产物。PAL 规定的场率是 50 场/秒（即 25 帧/秒），扫描线行数是 525 行；NTSC 规定的场率是 60 场/秒（即 30 帧/秒），扫描线行数是 625 行。

# 6 帧类型

XviD 定义了如下帧类型：

```
#define XVID_TYPE_VOL        -1  /* decoder only: vol was decoded */
#define XVID_TYPE_NOTHING    0   /* decoder only (encoder stats): nothing was dec
oded/encoded */
#define XVID_TYPE_AUTO       0   /* encoder: automatically determine coding type
*/
#define XVID_TYPE_IVOP       1   /* intra frame */
#define XVID_TYPE_PVOP       2   /* predicted frame */
#define XVID_TYPE_BVOP       3   /* bidirectionally encoded */
#define XVID_TYPE_SVOP       4   /* predicted+sprite frame */
```

在解码的时候，常见的做法是根据帧类型的不同而进行不同的处理。XVID_TYPE_VOL、XVID_TYPE_IVOP、XVID_TYPE_PVOP、XVID_TYPE_BVOP 较常用。作者暂不清楚 XVID_TYPE_SVOP 的含义。

# 7 xvid_global()函数

函数原型：

```
int xvid_global(void *handle, int opt, void *param1, void *param2);
```

功能：

- 全局初始化；
- 获取 XviD 和硬件功能信息（全局信息）；
- 色场空间转换。

参数：

- handle，是一个句柄，代表着一个 xvid global 操作实例，
- opt，指定要执行的操作，对应于上面三种功能，opt 分别可取 XVID_GBL_INIT、XVID_GBL_INFO、XVID_GBL_CONVERT 这样三个值；
- param1，是对应操作的入口参数或者出口参数。当 opt 为 XVID_GBL_INIT 时，param1 是入口参数，应当为一个 xvid_gbl_init_t 类型的指针；当 opt 为 XVID_GBL_INFO 时，param1 是出口参数，应当为一个 xvid_gbl_info_t 类型的指针；当 opt 为 XVID_GBL_CONVERT 时，param1 是入口参数，应当为一个 xvid_gbl_convert_t 类型的指针；
- param2，这个参数将被忽略，通常应当设置为 NULL。

返回值：

- 成功：返回 0；
- 失败：返回相应错误码（<0）。

## 7.1 全局初始化

当 xvid_global()函数的第二个参数 opt 为 XVID_GBL_INIT 时，xvid_global()函数对整个 XviD 进行初始化。初始化的内容包括要用到 CPU 的哪些功能以及要采用哪一个调试级别。XviD 的灵活之处在于：它既可以自行确定使用 CPU 的哪些功能，也能够让编程者来决定如何发掘 CPU 的处理能力。

opt 参数为 XVID_GBL_INIT 时，xvid_global()函数的 param1 参数应该为一个 xvid_gbl_init_t 类型的指针。xvid_gbl_init_t 定义为：

```
/* XVID_GBL_INIT param1 */
typedef struct {
    int version;
    unsigned int cpu_flags; /* [in:opt] zero = autodetect cpu; */
                            /* XVID_CPU_FORCE|{cpu features} = force cpu features */
    int debug;              /* [in:opt] debug level */
} xvid_gbl_init_t;
```

前面说过，XviD 的 API 接口中所有的结构体类型都包含一个 version 成员，应当将其设置为所采用的 XviD 的版本。

对于 cpu_flags 成员，当其值为 0 时，将由 XviD 自动探测所用 CPU 的处理能力。当想强制使用 CPU 的某种特殊功能时，必须同时为 cpu_flags 指定 XVID_CPU_FORCE 标志和相关的功能标志，比如，要强制使用 IA32 体系结构的 MMX 指令，就必须将 cpu_flags 进行如下设置：

```
struct.cpu_flags = XVID_CPU_FORCE | XVID_CPU_MMX;
```

XviD 支持的 CPU 功能列表如下所示，其中，XVID_CPU_ALTIVEC 是专门针对 PPC 体系结构的。其它的大部分功能都是针对 IA32 体系结构的。

```
#define XVID_CPU_FORCE      (1<<31)     /* force passed cpu flags */
#define XVID_CPU_ASM        (1<< 7)     /* native assembly */
/* ARCH_IS_IA32 */
#define XVID_CPU_MMX        (1<< 0)     /* mmx : pentiumMMX,k6 */
#define XVID_CPU_MMXEXT     (1<< 1)     /* mmx-ext : pentium2, athlon */
#define XVID_CPU_SSE        (1<< 2)     /* sse : pentium3, athlonXP */
#define XVID_CPU_SSE2       (1<< 3)     /* sse2 : pentium4, athlon64 */
#define XVID_CPU_3DNOW      (1<< 4)     /* 3dnow : k6-2 */
#define XVID_CPU_3DNOWEXT   (1<< 5)     /* 3dnow-ext : athlon */
#define XVID_CPU_TSC        (1<< 6)     /* tsc : Pentium */
/* ARCH_IS_PPC */
#define XVID_CPU_ALTIVEC    (1<< 0)     /* altivec */
```

xvid_gbl_init_t 结构体中的 debug 成员用来表明 XviD 的调试级别，不同的调试级别表明了不同的出错原因。实际上，这和 Linux 内核编程中的打印函数 printk()很相似。XviD 定义了如下调试级别：

```
#define XVID_DEBUG_ERROR        (1<< 0)
#define XVID_DEBUG_STARTCODE    (1<< 1)
#define XVID_DEBUG_HEADER       (1<< 2)
#define XVID_DEBUG_TIMECODE     (1<< 3)
#define XVID_DEBUG_MB           (1<< 4)
#define XVID_DEBUG_COEFF        (1<< 5)
#define XVID_DEBUG_MV           (1<< 6)
#define XVID_DEBUG_RC           (1<< 7)
#define XVID_DEBUG_DEBUG        (1<<31)
```

由于每一种调试级别实际上都是以一个标志位来表示的，因此实际上可以指定多重调试级别。

下面是一个采用 xvid_global()函数进行全局初始化的示例代码：

```
xvid_gbl_init_t     xvid_gbl_init;
/* Clear the structure with zeros */
memset(&xvid_gbl_init, 0, sizeof(xvid_gbl_init_t));
/* Version */
xvid_gbl_init.version = XVID_VERSION;
/* CPU setting */
xvid_gbl_init.cpu_flags = XVID_CPU_FORCE | XVID_CPU_MMX;
/* Debug setting */
xvid_gbl_init.debug = XVID_DEBUG_ERROR | XVID_DEBUG_COEFF;
/* Take in effect */
```

```
xvid_global(NULL, XVID_GBL_INIT, &xvid_gbl_init, NULL);
```

简单说明一下上述代码：首先将 xvid_gbl_init 结构体清零；然后依次设置版本、CPU 功能、调试级别，也就是分别给三个结构体成员赋值。这里，指定采用 CPU 的 MMX 指令，同时，指定了两个调试级别：XVID_DEBUG_ERROR 和 XVID_DEBUG_COEFF。最后，调用 xvid_global()使这些设置生效。注意到 xvid_global()函数的第一个参数 handle 和最后一个参数 param2 都被置为 NULL 指针。

## 7.2 获取全局信息

当 xvid_global()函数的第二个参数 opt 为 XVID_GBL_INFO 时，xvid_global()函数将用于获取 xvid 处理器硬件信息。xvid_global()函数的第三个参数 param1 为出口参数，指向一个 xvid_gbl_info_t 类型的结构体。

```
/* XVID_GBL_INFO param1 */
typedef struct {
    int version;
    int actual_version;          /* [out] returns the actual xvidcore version */
    const char * build;          /* [out] if !null, points to description of this xvid core bu
ild */
    unsigned int cpu_flags;      /* [out] detected cpu features */
    int num_threads;             /* [out] detected number of cpus/threads */
} xvid_gbl_info_t;
```

在调用 xvid_global()函数获取信息之前，应该设置 xvid_gbl_info_t 结构体的 version 成员为当前的 XviD 版本，而在实际的输出结果中，actual_version 表示的是 XviD 的实际版本；如果 build 指针不为空，则是指向一个描述字符串以表明 xvid 的构建版本；cpu_flags 用于表明 CPU 能够支持的功能集合；num_threads 表示线程数（作者未能搞清楚这个变量的含义）。

一个通过 xvid_global()函数获取 XviD 版本信息和 CPU 信息的示例程序如下所示：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "xvid.h"

int
main(void)
{
    xvid_gbl_info_t xvid_gbl_info;

    /* Clear xvid_gbl_info with 0s */
    memset(&xvid_gbl_info, 0, sizeof(xvid_gbl_info_t));
```

```
    /* Version */
    xvid_gbl_info.version = XVID_VERSION;

    /* Take effect */
    xvid_global(NULL, XVID_GBL_INFO, &xvid_gbl_info, NULL);

    /* print */
    printf("actual version: %d.%d.%d\n",
                XVID_VERSION_MAJOR(xvid_gbl_info.actual_version),
                XVID_VERSION_MINOR(xvid_gbl_info.actual_version),
                XVID_VERSION_PATCH(xvid_gbl_info.actual_version));
    if (xvid_gbl_info.cpu_flags | XVID_CPU_MMX)
        printf("CPU: MMX\n");
    if (xvid_gbl_info.build)
        printf("build: %s\n", xvid_gbl_info.build);
    printf("threads: %d\n", xvid_gbl_info.num_threads);

    return (0);
}
```

编译上述程序，运行结果如下：

```
[root@cyc test]# ./xvid_test
actual version: 1.1.2
CPU: MMX
build: xvid-1.1.2
threads: 0
```

## 7.3 色场空间转换

xvid_global()函数的第三个功能是完成色场空间的转换。此时 xvid_global()函数的第二个参数 opt 为 XVID_GBL_CONVERT，第三个参数 param1 应指向一个 xvid_gbl_convert_t 类型的结构体。

xvid_gbl_convert_t 定义为：

```
/* XVID_GBL_CONVERT param1 */
typedef struct {
    int version;
    xvid_image_t input;     /* [in] input image & colorspace */
    xvid_image_t output;    /* [in] output image & colorspace */
    int width;              /* [in] width */
    int height;             /* [in] height */
```

```
    int interlacing;            /* [in] interlacing */
} xvid_gbl_convert_t;
```

解释一下 xvid_gbl_convert_t 的各个成员：version 的含义不用再解释了，前面已多次说明；input 表示输入图像以及相应的色场空间；output 表示输出图像以及相应的色场空间；width 表示要转换的图像的宽度；height 表示要转换的图像的高度；interlacing 是一个标志，用来表明图像是否是隔行扫描图像。xvid_gbl_convert_t 中的所有成员都是输入参数。

input 和 output 都是 xvid_image_t 类型的变量，在 XviD 中将 xvid_image_t 结构体定义为：

```
typedef struct {
    int csp;            /* [in] colorspace; or with XVID_CSP_VFLIP to perform vertical fl
ip */
    void * plane[4];    /* [in] image plane ptrs */
    int stride[4];      /* [in] image stride; "bytes per row"*/
} xvid_image_t;
```

xvid_image_t 结构体的第一个成员 csp 表示色场空间，第二个成员 plane 是一个 4 元素的指针数组，数组中的每一个指针指向一个输出色场平面；第四个参数 stride 也是一个 4 元素的数组，数组中的每一个元素都用于指明在相应色场平面上的步长为多少，所谓步长实际上就是指一行像素点的个数。

（示例程序暂略）

# 8 xvid_decore()函数

XviD 解码的核心函数是 xvid_decore()。xvid_decore()的函数原型如下：

```
int xvid_decore(void *handle, int opt, void *param1, void *param2);
```

xvid_decore()函数的几个参数的含义分别是：

- handle，是一个句柄，用来代表一个 XviD 解码操作实例；
- opt，指定 xvid_decore()函数要执行的操作。opt 可取如下值之一：XVID_DEC_CREATE、XVID_DEC_DESTROY、XVID_DEC_DECODE。和 xvid_global()函数不同，在执行解码操作时，必须先创建一个 XviD 解码实例（XVID_DEC_CREATE），然后进入解码循环（XVID_DEC_DECODE），在解码循环完成之后应当销毁一开始创建的 XviD 解码实例（XVID_DEC_DESTROY）。
- param1，是一个通用指针类型，具体应该指向什么根据 opt 参数的取值而定，比如，当 opt 的值为 XVID_DEC_CREATE 时，param1 应该指向一个 xvid_dec_create_t 类型的结构体。对于 opt 为 XVID_DEC_CREATE 和 XVID_DEC_DECODE 的情形，param1 都是必要参数。对于 opt 为 XVID_DEC_DESTROY 的情况，param1 参数将被忽略，通常设置为 NULL 即可。

- param2，是一个可选参数，只在 opt 为 XVID_DEC_DECODE 的时候有效，指向一个 xvid_dec_stats_t 类型的结构体。

下面就依次来看看 xvid_decore()函数都是如何调用的。

## 8.1 创建 xvid 解码实例

在使用 XviD 进行解码之前，必须先创建 XviD 解码实例。创建 xvid 解码实例的具体做法是：首先初始化一个 xvid_dec_create_t 类型的结构体，然后将其地址做为 param1 参数传递给 xvid_decore()函数，其中，opt 参数指定为 XVID_DEC_CREATE。第一个参数 handle 和 param2 参数都可以指定为 NULL。

在 xvid 中，xvid_dec_create_t 定义如下：

```
/* XVID_DEC_CREATE param 1
    image width & height may be specified here when the dimensions are
    known in advance. */
typedef struct {
    int version;
    int width;          /* [in:opt] image width */
    int height;         /* [in:opt] image width */
    void * handle;      /* [out]      decore context handle */
} xvid_dec_create_t;
```

在 xvid_dec_create_t 的几个成员中：

- version，是采用的 XviD 的版本，按照前面的方式初始化即可；
- width，是解码后的图像的宽度，如果解码之前已经事先知道，可设置，否则设置为 0 即可。width 是输入参数；
- height，是解码后的图像的高度，如果解码之前已经事先知道，可设置，否则设置为 0 即可。height 是输入参数；
- handle，是一个句柄。之所以有这个成员，其原因是：通常在调用 xvid_decore()函数创建 XviD 解码实例的时候，都将 xvid_decore()函数的第一个参数 handle 设置为 NULL，在创建 xvid 解码实例完成之后再从 xvid_dec_create_t 的 handle 成员中得到这个句柄。此后在调用 xvid_decore()函数进行解码的时候就可以使用这个句柄了。Handle 成员是输出参数。

示例：

```
xvid_dec_create_t xvid_dec_create;
void *dec_handle = NULL;

/* Reset the structure with zeros */
memset(&xvid_dec_create, 0, sizeof(xvid_dec_create_t));
/* Version */
```

```
xvid_dec_create.version = XVID_VERSION;
/*
* Image dimensions -- set to 0, xvidcore will resize when ever it is
* needed
*/
xvid_dec_create.width = 0;
xvid_dec_create.height = 0;

ret = xvid_decore(NULL, XVID_DEC_CREATE, &xvid_dec_create, NULL);
dec_handle = xvid_dec_create.handle;
```

上面的这段示例代码定义了一个句柄 dec_handle（其实是一个无类型指针），在调用 xvid_decore()函数完成 XviD 解码实例的创建工作之后，通过

```
dec_handle = xvid_dec_create.handle;
```

将创建的句柄指定给了 dec_handle。

## 8.2 解码

在创建了解码实例之后，接下来就是开始真正的解码了。解码也是通过 xvid_decore()函数来完成的。解码要用到两个结构体类型：xvid_dec_frame_t 和 xvid_dec_stats_t，xvid_dec_frame_t 指定给 xvid_decore()函数的 param1 参数，xvid_dec_stats_t 指定给 xvid_decore()函数的 param2 参数。尽管 param2 是可选参数，但是通常情况下都是应该指定的，因为通过它才能知道解码过程的状态。此外，xvid_decore()函数的第一个参数 handle 应该为创建 XviD 解码实例时返回的 xvid_dec_create_t 类型的结构体中的 handle 成员。在进行解码操作时，xvid_decore()函数的 opt 参数应当设置为 XVID_DEC_DECODE。

XviD 对 xvid_dec_frame_t 类型的定义如下：

```
typedef struct {
    int version;
    int general;            /* [in:opt] general flags */
    void *bitstream;        /* [in]     bitstream (read from)*/
    int length;             /* [in]     bitstream length */
    xvid_image_t output;    /* [in]     output image (written to) */
/* ------- v1.1.x ------- */
    int brightness;         /* [in]     brightness offset (0=none) */
} xvid_dec_frame_t;
```

xvid_dec_frame_t 中各个成员的作用简单说明如下：

- general，是一个可选输入参数。这个参数的取值可以是下面这些标志取或的结果：

```
/* general flags */
#define XVID_LOWDELAY         (1<<0)  /* lowdelay mode  */
#define XVID_DISCONTINUITY    (1<<1)  /* indicates break in stream */
#define XVID_DEBLOCKY         (1<<2)  /* perform luma deblocking */
#define XVID_DEBLOCKUV        (1<<3)  /* perform chroma deblocking */
#define XVID_FILMEFFECT       (1<<4)  /* adds film grain */
#define XVID_DERINGUV         (1<<5)  /* perform chroma deringing, requires debloc
king to work */
#define XVID_DERINGY          (1<<6)  /* perform luma deringing, requires deblocki
ng to work */
#define XVID_DEC_FAST         (1<<29) /* disable postprocessing to decrease cpu usa
ge *todo* */
#define XVID_DEC_DROP         (1<<30) /* drop bframes to decrease cpu usage *todo
* */
#define XVID_DEC_PREROLL      (1<<31) /* decode as fast as you can, don't even sh
ow output *todo* */
```

这些标志控制着解码器的某些功能：

- XVID_LOWDELAY 以低延迟模式解码；
- XVID_DISCONTINUITY 以非连续模式解码；
- XVID_DEBLOCKY 去除亮度块状化（猜测）；
- XVID_DEBLOCKUV 去除色差块状化（猜测）；
- XVID_FILMEFFECT 添加影片效果（猜测）；
- XVID_DERINGUV 去除色差“振铃”(一种图像畸变)，要求开启去除块状化标志；
- XVID_DERINGY 去除亮度“振铃”，要求开启去除块状化标志；
- XVID_DEC_FAST 禁用后处理以减少 CPU 使用，暂未实现；
- XVID_DEC_DROP 丢弃 B 帧以减少 CPU 使用，暂未实现；
- XVID_DEC_PREROLL 尽可能快地解码，甚至不显示输出。

通常情况下，如果不需要上面的这些功能，可以将 general 设置为 0。

- bitstream，是一个输入参数，指向要解码的输入流；
- length，是一个输入参数，表明 bitstream 所指向的输入码流的长度；
- output，输入参数，是一个 xvid_image_t 类型的变量，用来指定解码后的输出流，xvid_image_t 类型前面已有说明，这里再次列举如下：

```
typedef struct {
    int csp;            /* [in] colorspace; or with XVID_CSP_VFLIP to perform vertical fl
ip */
```

```
    void * plane[4];   /* [in] image plane ptrs */
    int stride[4];     /* [in] image stride; "bytes per row"*/
} xvid_image_t;
```

- csp，是一个输入参数，指明输出应当采用的色场空间，如果想将输入图像进行垂直翻转，那么可以将 csp 指定为 XVID_CSP_VFLIP；
- plane，输入参数，是一个数组，用来指向要输出的颜色平面——最多可以有 4 个输出平面。实际上，一个 plane 元素就指向一个输出缓冲区，用于存放相应的色彩平面。如果解码后的输出图像只有一个色彩平面，那么只使用 plane[0]；
- stride，输入参数，也是一个数组，用来指出每一个平面的步长，简单说步长就是一行像素点所占的空间大小，以字节为单位。如果有多个平面，可以为每个平面设置一个步长值。

xvid_decore()函数的另一个可选参数为 xvid_dec_stats_t 类型的指针：

```
/* XVID_DEC_DECODE param2 :: optional */
typedef struct
{
    int version;

    int type;                   /* [out] output data type */
    union {
        struct {                /* type>0 {XVID_TYPE_IVOP,XVID_TYPE_PVOP,X
VID_TYPE_BVOP,XVID_TYPE_SVOP} */
            int general;        /* [out] flags */
            int time_base;      /* [out] time base */
            int time_increment; /* [out] time increment */

            /* XXX: external deblocking stuff */
            int * qscale;       /* [out] pointer to quantizer table */
            int qscale_stride;  /* [out] quantizer scale stride */

        } vop;
        struct {  /* XVID_TYPE_VOL */
            int general;        /* [out] flags */
            int width;          /* [out] width */
            int height;         /* [out] height */
            int par;            /* [out] pixel aspect ratio (refer to XVID_PAR_xxx a
bove) */
            int par_width;      /* [out] aspect ratio width  [1..255] */
            int par_height;     /* [out] aspect ratio height [1..255] */
        } vol;
    } data;
```

```
} xvid_dec_stats_t;
```

xvid_dec_stats_t 类型中最重要的部分是 data，这是一个联合，联合的两个成员分别是两个结构体，结构体 vop 描述了一个视频对象平面（Video Object Plane）；而结构体 vol 则描述了一个视频对象层（Video Object Layer）。

xvid_dec_stats_t 类型中，type 成员的作用是用来确定解出来的帧的类型，有关帧的类型前面已做说明。当 type 的值为 XVID_TYPE_VOL 时，data 联合中存放的是 vol；当 type 的值为 XVID_TYPE_IVOP、XVID_TYPE_PVOP、XVID_TYPE_BVOP、XVID_TYPE_SVOP 之一时，data 联合中存放的是 vop。

首先来解释 vol 结构体中的各个成员：

- general 中保存的是解码器解码时所采用的标志；
- width 是解码出来的图像的宽度，以像素为单位，
- height 是解码出来的图像的高度，以像素为单位；
- par 是解码出来的图像的像素幅型比；
- 如果解码后的图像不是标准的 NTSC 或者 PAL 幅型比，则 par_width 和 par_height 就是实际所采用的像素幅型比。

对于 vop 结构体中的各个成员：

- general 中保存的是解码器解码时所采用的标志；
- time_base 是相对于开始解码时的时间长度（猜测）；
- time_increment 是相对于上一帧的时间间隔；
- qscale 指向外部提供的量化表（猜测）；
- qscale_stride 代表量化器的缩放步长（具体含义未搞清楚）。

前面说过，如果事先知道解出来的图像的大小，可以在创建 XviD 解码实例的时候指定解码后的图像的大小；如果不知道的话，也没有关系，XviD 可以在解码的过程中知道解出来的图像的大小，这就是通过 xvid_dec_stats_t 中的 vol 结构体来实现的。

示例代码：

```
#define   IN_BITSTREAM_SIZE      1024
#define   OUT_BITSTREAM_SIZE     1024
#define   STRIDE                 352

unsigned char istream[IN_BITSTREAM_SIZE];
unsigned char ostream[OUT_BITSTREAM_SIZE];
int istream_size = IN_BITSTREAM_SIZE;
xvid_dec_frame_t xvid_dec_frame;
xvid_dec_stats_t xvid_dec_stats;

/* Reset all structures */
```

```
memset(&xvid_dec_frame, 0, sizeof(xvid_dec_frame_t));
memset(&xvid_dec_stats, 0, sizeof(xvid_dec_stats_t));

/* Set version */
xvid_dec_frame.version = XVID_VERSION;
xvid_dec_stats.version = XVID_VERSION;

/* No general flags to set */
xvid_dec_frame.general = 0;

/* Input stream */
xvid_dec_frame.bitstream = istream;
xvid_dec_frame.length = istream_size;

/* Output frame structure */
xvid_dec_frame.output.plane[0]  = ostream;
xvid_dec_frame.output.stride[0] = STRIDE;
xvid_dec_frame.output.csp = XVID_CSP_UYVY;

/* submit to xvide decode core */
xvid_decore(dec_handle, XVID_DEC_DECODE, &xvid_dec_frame, &xvid_dec_stats);
```

在 xvid_decore()函数中，dec_handle 是上一节创建 XviD 解码实例时得到的那个句柄。上面的实例代码中没有加入从文件或者其它媒介（比如网络）读取输入码流的过程，解码得到的图像采用的是 UYVY 格式（一种 YUV 4:2:2 紧缩格式），采用单平面输出。另外，通常情况下的做法是在一个循环中不断解码，根据解码的状态决定对解码得到的码流如何处理，比如，如果解码正常，那么可能会将得到图像通过显示设备输出或者保存到文件中。而如果解码得到的是 vol，那么可能会根据需要调整输出图像缓冲区的大小，等等。

## 8.3 销毁 xvid 解码实例

销毁 XviD 解码实例比较简单。方法如下：将 xvid_decore()函数的第一个参数 handle 设为创建 XviD 解码实例后得到的句柄；第二个参数 opt 设置为 XVID_DEC_DESTROY；参数 param1 和 param2 都设为 NULL。销毁 XviD 解码实例之后，不能再通过其对输入码流进行解码。

示例代码：

```
xvid_decore(dec_handle, XVID_DEC_DESTROY, NULL, NULL);
```

上述代码中的第一个参数 dec_handle 是创建 XviD 解码实例之后得到的。

# 9 xvid_encore()函数

xvid_encore()是对输入的原始图像进行编码，输出符合 MPEG-4 标准规定的码流。与 xvid_decore()函数的工作流程一样：xvid_encore()函数的使用方法也是先创建一个 XviD 编码实例；然后执行实际的编码操作；最后，在所有的解码过程都结束之后，销毁所创建的 XviD 编码实例。不过，与 xvid_decore()相比，xvid_encore()函数的使用要复杂一些，主要原因是编码过程的参数设置要麻烦一些。xvid_encore()函数原型定义如下：

```
int xvid_encore(void *handle, int opt, void *param1, void *param2);
```

xvid_encore()函数的几个参数的含义分别是：

- handle，编码实例的句柄，用来代表一个 xvid 编码操作实例；
- opt，指定 xvid_encore()函数要执行的操作。opt 可取如下值之一：XVID_ENC_CREATE、XVID_ENC_DESTROY、XVID_ENC_ENCODE。在执行编码操作时，必须先创建一个 XviD 编码实例（XVID_ENC_CREATE），然后进入编码循环（XVID_ENC_ENCODE），在编码循环完成之后应当销毁所创建的 XviD 编码实例（XVID_ENC_DESTROY）。
- param1，是一个通用指针类型，具体应该指向什么根据 opt 参数的取值而定。当 opt 的值为 XVID_ENC_CREATE 时，param1 应该指向一个 xvid_enc_create_t 类型的结构体。对于 opt 为 XVID_ENC_CREATE 和 XVID_ENC_ENCODE 的情形，param1 都是必要参数。对于 opt 为 XVID_ENC_DESTROY 的情况，param1 应当为 NULL。
- param2，是一个可选参数，只在 opt 为 XVID_ENC_ENCODE 的时候有效，指向一个 xvid_dec_stats_t 类型的结构体。

返回值：

- 成功：对于 opt 为 XVID_ENC_CREATE 和 XVID_ENC_DESTROY 的情形，xvid_encore()函数返回 0 表示成功，对于 opt 为 XVID_ENC_ENCODE 的情形，xvid_encore()函数返回输出的字节数表示执行成功；
- 失败：对于 opt 为 XVID_ENC_CREATE 和 XVID_ENC_DESTROY 的情形，xvid_encore()函数返回小于 0 的错误码表示执行失败，对于 opt 为 XVID_ENC_ENCODE 的情形，xvid_encore()函数返回 0 表示该帧不应该被写入；

下面就来依次看看 xvid_encore()函数是如何完成不同的功能的。

## 9.1 创建 xvid 编码实例

在创建 XviD 编码实例时，应当将 xvid_encore()函数的 param1 参数设为一个 xvid_enc_create_t 类型的结构体的地址。xvid_enc_create_t 类型定义如下：

```
/*----------------------------------------------------------------------
```

```
 * xvid_enc_create_t structure definition
 *
 * This structure is passed as param1 during an instance creation (operation
 * XVID_ENC_CREATE)
 *----------------------------------------------------------------------------*/

typedef struct {
    int version;

    int profile;                /* [in] profile@level; refer to XVID_PROFILE_xxx */
    int width;                  /* [in] frame dimensions; width, pixel units */
    int height;                 /* [in] frame dimensions; height, pixel units */

    int num_zones;              /* [in:opt] number of bitrate zones */
    xvid_enc_zone_t * zones;    /*              ^^ zone array */

    int num_plugins;            /* [in:opt] number of plugins */
    xvid_enc_plugin_t * plugins;/*              ^^ plugin array */

    int num_threads;            /* [in:opt] number of threads */
    int max_bframes;            /* [in:opt] max sequential bframes (0=disable bframe
s) */

    int global;                 /* [in:opt] global flags; controls encoding behavior */

    /* --- vol-based stuff; included here for convenience */
    int fincr;                  /* [in:opt] framerate increment; set to zero for variabl
e framerate */
    int fbase;                  /* [in] framerate base frame_duration = fincr/fbase se
conds*/
    /* ---------------------------------------------- */

    /* --- vop-based; included here for convenience */
    int max_key_interval;       /* [in:opt] the maximum interval between key frames
 */

    int frame_drop_ratio;       /* [in:opt] frame dropping: 0=drop none... 100=drop a
ll */

    int bquant_ratio;           /* [in:opt] bframe quantizer multipier/offeset; used to
decide bframes quant when bquant==-1 */
    int bquant_offset;          /* bquant = (avg(past_ref_quant,future_ref_quant)*bqua
nt_ratio + bquant_offset) / 100 */
```

```
    int  min_quant[3];              /*  [in:opt]  */
    int  max_quant[3];              /*  [in:opt]  */
    /*  ------------------------------------------  */

    void  *handle;                  /*  [out]  encoder  instance  handle  */
}  xvid_enc_create_t;
```

下面对 xvid_enc_create_t 中的成员逐一进行简要说明：

- version，应当设置为所用的 xvid 的版本，同前；
- profile，应当设置为准备采用的 profile 和 level 组合，XviD 编码器将据此决定如何在编码质量和压缩效率以及运算强度之间进行折衷。关于 profile 和 level 的定义在前面已有说明。比如，如果将 profile 设定为 XVID_PROFILE_S_L0，就表明希望 XviD 编码器按照 MPEG-4 Simple profile 的 level 0 进行编码。profile 是一个输入参数；
- width，为输入帧的宽度，以像素为单位。width 是一个输入参数；
- height，为输入帧的高度，以像素为单位。height 是一个输入参数；
- num_zones，**似乎**是码流 zone 数目（作用未搞清楚），这是一个可选输入参数；
- zones，**似乎**是 zone 数组（作用未搞清楚），这是一个可选输入参数；
- num_plugins，插件数目，这是一个可选输入参数；
- plugins，插件数组，这是一个可选输入参数；
- num_threads，线程数，（作用未知），这是一个可选输入参数；
- max_bframes，最大 B 帧数，设为 0 禁用 B 帧，这是一个可选输入参数；
- global，全局标志位，这些标志位控制着 XviD 编码器的行为。这是一个可选输入参数，当不需要控制编码器的行为时，将其设为 0 即可。global 中允许的标志位如下（作者窃以为这些标志都是为视频编码专家准备的）：

```
/*----------------------------------------------------------------------
 * "Global"  flags
 *
 * These  flags  are  used  for  xvid_enc_create_t->global  field  during  instance
 * creation  (operation  XVID_ENC_CREATE)
 *---------------------------------------------------------------------*/

#define  XVID_GLOBAL_PACKED                  (1<<0)  /*  packed  bitstream  */
#define  XVID_GLOBAL_CLOSED_GOP              (1<<1)  /*  closed_gop:  was  DX50BVOP
 dx50  bvop  compatibility  */
#define  XVID_GLOBAL_EXTRASTATS_ENABLE (1<<2)
#if  0
#define  XVID_GLOBAL_VOL_AT_IVOP             (1<<3)  /*  write  vol  at  every  ivop:  WIN
```

```
32/divx compatibility */
#define XVID_GLOBAL_FORCE_VOL          (1<<4) /* when vol-based parameters ar
e changed, insert an ivop NOT recommended */
#endif
#define XVID_GLOBAL_DIVX5_USERDATA     (1<<5) /* write divx5 userdata string
                                   this is implied if XVID_GLOBAL_PACKED is set */
```

- fincr，帧率增量，设为 0 时为可变帧率（作用未完全搞清楚），这是一个可选输入参数；
- fbase，这个成员用于确定帧之间的间隔时间，计算公式为：帧间间隔=fincr/fbase 秒；
- max_key_interval，关键帧（即 Intra frame）与关键帧之间的最大间隔，单位为帧。比如，如果设为 50，就表示两个关键帧之间应该有 50 个 P 帧或者 B 帧。这是一个可选输入参数；
- frame_drop_ratio，为丢弃帧占总帧数的比例，取值为 0 到 100 之间，为 0 时不丢弃帧，为 100 时丢弃所有帧。这是一个可选输入参数；
- bquant_ratio，B 帧量化器 multipier（作用未搞清楚），这是一个可选输入参数；
- bquant_offset，B 帧量化器偏移（作用未搞清楚）。**似乎**是通过下面的公式 bquant = (avg(past_ref_quant,future_ref_quant)*bquant_ratio + bquant_offset) / 100 计算 B 帧的量化器系数，在此公式中用到了 bquant_ratio 和 bquant_offset；
- min_quant[3]，最小量化系数（作用未知），可选输入参数；
- max_quant[3]，最大量化系数（作用未知），可选输入参数；
- handle，在创建 xvid 编码实例之后，xvid_encore()函数将通过这个成员返回一个编码实例的句柄。这个成员是一个输出参数。

尽管 xvid_enc_create_t 中需要设置的参数比较多，但是很多都是可选参数，必须设置的参数仅下面这样几个：version、profile、width、height。因此，下面就给出一个最简单的示例代码（未经过验证）：

```
#define   CIF_ROW     352
#define   CIF_COL     288

void * enc_handle = NULL;
xvid_enc_create_t xvid_enc_create;

/* Version again */
memset(&xvid_enc_create, 0, sizeof(xvid_enc_create_t));
xvid_enc_create.version = XVID_VERSION;

/* Width and Height of input frames */
xvid_enc_create.width = CIF_ROW;
xvid_enc_create.height = CIF_COL;
```

```
xvid_enc_create.profile = XVID_PROFILE_S_L0;

xvid_encore(NULL, XVID_ENC_CREATE, &xvid_enc_create, NULL);

/* Retrieve the encoder instance from the structure */
enc_handle = xvid_enc_create.handle;
```

在调用 xvid_encore()函数创建 XviD 编码实例的时候，第一个参数 handle 指定为 NULL（实际上即使指定了值也会被 XviD 的 API 函数忽略掉），在创建完成之后，从 xvid_enc_create 中取出创建的编码实例的句柄赋给 enc_handle。此后，在编码过程中或者在销毁编码实例的时候，都应该将 xvid_encore()函数的 handle 参数指定为 enc_handle。

## 9.2 编码

在创建了 XviD 编码实例之后，紧接着的事情就是进行真正的编码工作了，这也是通过 xvid_encore()函数完成的。为了完成编码过程，xvid_encore()函数的 param1 参数应当设定为一个指向 xvid_enc_frame_t 类型的结构体，可选参数 param2 可以指向一个 xvid_enc_stats_t 类型的结构体，如果不关心编码状态，也可以将 param2 设置为 NULL。xvid_encore()函数的第一个参数 handle 通常应当设置为创建 XviD 编码实例之后得到的句柄，而 opt 参数应该指定为 XVID_ENC_ENCODE。当 xvid_encore()函数调用成功时，将返回编码后输出的字节数，如果返回 0 则表明当前所在的帧不应该写入。

xvid_enc_frame_t 是 param1 参数应当指向的类型，xvid_enc_frame_t 在 XviD 中定义为：

```
/*----------------------------------------------------------------------
 * xvid_enc_frame_t structure definition
 *
 * This structure is passed as param1 during a frame encoding (operation
 * XVID_ENC_ENCODE)
 *----------------------------------------------------------------------*/

/* out value for the frame structure->type field
 * unlike stats output in param2, this field is not asynchronous and tells
 * the client app, if the frame written into the stream buffer is an ivop
 * usually used for indexing purpose in the container */
#define XVID_KEYFRAME (1<<1)

/* The structure */
typedef struct {
    int version;

    /* VOL related stuff
     * unless XVID_FORCEVOL is set, the encoder will not react to any changes
     * here until the next VOL (keyframe). */
```

```
    int vol_flags;                  /* [in] vol flags */
    unsigned char *quant_intra_matrix;  /* [in:opt] custom intra qmatrix */
    unsigned char *quant_inter_matrix;  /* [in:opt] custom inter qmatrix */

    int par;                        /* [in:opt] pixel aspect ratio (refer to XVID_PAR
_xxx above) */
    int par_width;                  /* [in:opt] aspect ratio width */
    int par_height;                 /* [in:opt] aspect ratio height */

    /* Other fields that can change on a frame base */

    int fincr;                      /* [in:opt] framerate increment, for variable fram
erate only */
    int vop_flags;                  /* [in] (general)vop-based flags */
    int motion;                     /* [in] ME options */

    xvid_image_t input;             /* [in] input image (read from) */

    int type;                       /* [in:opt] coding type */
    int quant;                      /* [in] frame quantizer; if <=0, automatic (rateco
ntrol) */
    int bframe_threshold;

    void *bitstream;                /* [in:opt] bitstream ptr (written to)*/
    int length;                     /* [in:opt] bitstream length (bytes) */

    int out_flags;                  /* [out] bitstream output flags */
} xvid_enc_frame_t;
```

在 xvid_enc_frame_t 中，各个成员的含义如下所示：

- version，前面已多次说明，应设为当前所用的 XviD 版本；
- vol_flags，包含 vol 标志位，vol_flags 是一个输入参数。xvid 支持的标志位如下：

```
/*-----------------------------------------------------------------------
 * "VOL" flags
 *
 * These flags are used for xvid_enc_frame_t->vol_flags field during frame
 * encoding (operation XVID_ENC_ENCODE)
 *----------------------------------------------------------------------*/

#define XVID_VOL_MPEGQUANT      (1<<0)  /* enable MPEG type quantization */
#define XVID_VOL_EXTRASTATS     (1<<1)  /* enable plane sse stats */
```

```
#define XVID_VOL_QUARTERPEL     (1<<2)  /* enable quarterpel: frames will encode
d as quarterpel */
#define XVID_VOL_GMC            (1<<3)  /* enable GMC; frames will be checked
 for gmc suitability */
#define XVID_VOL_REDUCED_ENABLE     (1<<4)  /* enable reduced resolution vops: f
rames will be checked for rrv suitability */
                                        /* NOTE:  the reduced resolution feature is not
supported anymore. This flag will have no effect! */
#define XVID_VOL_INTERLACING    (1<<5)  /* enable interlaced encoding */
```

- quant_intra_matrix，是一个可选输入参数，可以将其指向一个自定义的帧内（intra）量化矩阵；
- quant_inter_matrix，是一个可选输入参数，可以将其指向一个自定义的帧间（inter）量化矩阵；
- par，是一个可选输入参数，用于指定像素幅型比，其取值应该是几个标准的像素幅型比宏定义中的一个（XVID_PAR_xxx）；
- par_width，是一个可选输入参数，用于指定像素幅型比的宽度；
- par_height，是一个可选输入参数，用于指定像素幅型比的高度；
- fincr，是一个可选输入参数，指定帧率增量，只适用于可变帧率的情形；
- vop_flags，是一个输入参数，其中保存着基于 VOP 的标志位。允许的标志位定义如下所示：

```
/*------------------------------------------------------------------------
 * "VOP" flags
 *
 * These flags are used for xvid_enc_frame_t->vop_flags field during frame
 * encoding (operation XVID_ENC_ENCODE)
 *----------------------------------------------------------------------*/

/* Always valid */
#define XVID_VOP_DEBUG          (1<< 0) /* print debug messages in frames
*/
#define XVID_VOP_HALFPEL        (1<< 1) /* use halfpel interpolation */
#define XVID_VOP_INTER4V        (1<< 2) /* use 4 motion vectors per MB */
#define XVID_VOP_TRELLISQUANT       (1<< 3) /* use trellis based R-D "optimal"
quantization */
#define XVID_VOP_CHROMAOPT      (1<< 4) /* enable chroma optimization pre-fi
lter */
#define XVID_VOP_CARTOON        (1<< 5) /* use 'cartoon mode' */
#define XVID_VOP_GREYSCALE      (1<< 6) /* enable greyscale only mode (eve
n for  color input material chroma is ignored) */
```

```
#define XVID_VOP_HQACPRED               (1<< 7) /* high quality ac prediction */
#define XVID_VOP_MODEDECISION_RD        (1<< 8) /* enable DCT-ME and use it for
mode decision */
#define XVID_VOP_FAST_MODEDECISION_RD (1<<12) /* use simplified R-D mode de
cision */
#define XVID_VOP_RD_BVOP                (1<<13) /* enable rate-distortion mode decisi
on in b-frames */

/* Only valid for vol_flags|=XVID_VOL_INTERLACING */
#define XVID_VOP_TOPFIELDFIRST          (1<< 9) /* set top-field-first flag  */
#define XVID_VOP_ALTERNATESCAN          (1<<10) /* set alternate vertical scan flag */

/* only valid for vol_flags|=XVID_VOL_REDUCED_ENABLED */
#define XVID_VOP_REDUCED                (1<<11) /* reduced resolution vop */
                                                /* NOTE: reduced resolution feature is not s
upported anymore. This flag will have no effect! */
```

- motion，是一个输入参数，其中保存着用于运动估计的标志位。允许的标志位定义如下：

```
/*----------------------------------------------------------------------------
 * "Motion" flags
 *
 * These flags are used for xvid_enc_frame_t->motion field during frame
 * encoding (operation XVID_ENC_ENCODE)
 *--------------------------------------------------------------------------*/

/* Motion Estimation Search Patterns */
#define XVID_ME_ADVANCEDDIAMOND16       (1<< 0) /* use advdiamonds instead of
diamonds as search pattern */
#define XVID_ME_ADVANCEDDIAMOND8        (1<< 1) /* use advdiamond for XVID_
ME_EXTSEARCH8 */
#define XVID_ME_USESQUARES16            (1<< 2) /* use squares instead of diam
onds as search pattern */
#define XVID_ME_USESQUARES8             (1<< 3) /* use square for XVID_ME_E
XTSEARCH8 */

/* SAD operator based flags */
#define XVID_ME_HALFPELREFINE16         (1<< 4)
#define XVID_ME_HALFPELREFINE8          (1<< 6)
#define XVID_ME_QUARTERPELREFINE16      (1<< 7)
#define XVID_ME_QUARTERPELREFINE8       (1<< 8)
#define XVID_ME_GME_REFINE              (1<< 9)
#define XVID_ME_EXTSEARCH16             (1<<10) /* extend PMV by more search
```

```
es */
#define XVID_ME_EXTSEARCH8              (1<<11) /* use diamond/square for exte
nded 8x8 search */
#define XVID_ME_CHROMA_PVOP             (1<<12) /* also use chroma for P_VOP/
S_VOP ME */
#define XVID_ME_CHROMA_BVOP             (1<<13) /* also use chroma for B_VOP
 ME */
#define XVID_ME_FASTREFINE16            (1<<25) /* use low-complexity refineme
nt functions */
#define XVID_ME_FASTREFINE8             (1<<29) /* low-complexity 8x8 sub-bloc
k refinement */

/* Rate Distortion based flags
 * Valid when XVID_VOP_MODEDECISION_RD is enabled */
#define XVID_ME_HALFPELREFINE16_RD      (1<<14) /* perform RD-based halfpel re
finement */
#define XVID_ME_HALFPELREFINE8_RD       (1<<15) /* perform RD-based halfpel re
finement for 8x8 mode */
#define XVID_ME_QUARTERPELREFINE16_RD   (1<<16) /* perform RD-based qpel refin
ement */
#define XVID_ME_QUARTERPELREFINE8_RD    (1<<17) /* perform RD-based qpel refin
ement for 8x8 mode */
#define XVID_ME_EXTSEARCH_RD            (1<<18) /* perform RD-based search usi
ng square pattern enable XVID_ME_EXTSEARCH8 to do this in 8x8 search as well */
#define XVID_ME_CHECKPREDICTION_RD      (1<<19) /* always check vector equal t
o prediction */

/* Other */
#define XVID_ME_DETECT_STATIC_MOTION    (1<<24) /* speed-up ME by detecting st
ationary scenes */
#define XVID_ME_SKIP_DELTASEARCH        (1<<26) /* speed-up by skipping b-fram
e delta search */
#define XVID_ME_FAST_MODEINTERPOLATE    (1<<27) /* speed-up by partly skipping
 interpolate mode */
#define XVID_ME_BFRAME_EARLYSTOP        (1<<28) /* speed-up by early exiting b-
search */

/* Unused */
#define XVID_ME_UNRESTRICTED16          (1<<20) /* unrestricted ME, not implem
ented */
#define XVID_ME_OVERLAPPING16           (1<<21) /* overlapping ME, not implem
ented */
#define XVID_ME_UNRESTRICTED8           (1<<22) /* unrestricted ME, not implem
ented */
```

```
#define XVID_ME_OVERLAPPING8                (1<<23) /* overlapping ME, not implemented */
```

- input，是一个 xvid_image_t 类型的输入参数，代表着输入流，其中不仅指定了输入流的输入数据，也指定了输入流采用的色场空间；
- type，是一个可选的输入参数，用于指定编码的类型，通常可以指定为 XVID_TYPE_AUTO；
- quant，是一个输入参数，表示帧量化器，如果取值小于等于 0，则自动选择（码率控制）；
- bframe_threshold，（作用未知）；
- bitstream，是一个可选输入参数，指向编码后的输出缓冲区；
- length，是一个可选输入参数，表明输出缓冲区（即 bitstream 所指向的空间）的长度，单位为字节；
- out_flags，是一个输出参数，包含输出码流的相关标志。

当执行编码操作时，xvid_encore()函数的 param2 参数可以指向一个 xvid_enc_stats_t 类型的结构体。xvid_enc_stats_t 类型定义如下：

```
/*----------------------------------------------------------------------------
 * xvid_enc_stats_t structure
 *
 * Used in:
 *   - xvid_plg_data_t structure
 *   - optional parameter in xvid_encore() function
 *
 * .coding_type = XVID_TYPE_NOTHING if the stats are not given
 *---------------------------------------------------------------------------*/

typedef struct {
    int version;

    /* encoding parameters */
    int type;           /* [out] coding type */
    int quant;          /* [out] frame quantizer */
    int vol_flags;      /* [out] vol flags (see above) */
    int vop_flags;      /* [out] vop flags (see above) */

    /* bitrate */
    int length;         /* [out] frame length */

    int hlength;        /* [out] header length (bytes) */
    int kblks;          /* [out] number of blocks compressed as Intra */
```

```
    int mblks;          /* [out] number of blocks compressed as Inter */
    int ublks;          /* [out] number of blocks marked as not_coded */

    int sse_y;          /* [out] Y plane's sse */
    int sse_u;          /* [out] U plane's sse */
    int sse_v;          /* [out] V plane's sse */
} xvid_enc_stats_t;
```

xvid_enc_stats_t 既可以用作 xvid_encore()函数的 param2 参数，也可以包含在 xvid_plg_data_t 类型中（插件数据）。下面的讨论只针对 xvid_enc_stats_t 用作 xvid_encore()函数的 param2 参数的情形。在此种情况下，除了 version 外，xvid_enc_stats_t 中的所有成员都是输出参数，用来返回编码操作执行的状态。各个成员的简单说明如下：

- type，是一个输出参数，表明编码类型；
- quant，是一个输出参数，表明采用的量化器；
- vol_flags，是一个输出参数，表明为编码器指定的 vol 标志；
- vop_flags，是一个输出参数，表明为编码器指定的 vop 标志；
- length，是一个输出参数，编码的一帧的长度；
- hlength，是一个输出参数，编码后的头部长度，单位为字节；
- kblks，是一个输出参数，帧内编码压缩的宏块数目；
- mblks，是一个输出参数，是一个输出参数，帧间编码压缩的宏块数目；
- ublks，是一个输出参数，标记为不编码的宏块数目；
- sse_y，是一个输出参数，Y 平面上的 sse 值（SSE 表示总方误差）；
- sse_u，是一个输出参数，U 平面上的 sse 值；
- sse_v，是一个输出参数，V 平面上的 sse 值。

示例代码：

```
#define   CIF_ROW     352
#define   CIF_COL     288
#define   CSP         XVID_CSP_I420
#define   BPP         1

unsigned char image[CIF_ROW * CIF_COL * BPP];
unsigned char bitstream[1024];

xvid_enc_frame_t xvid_enc_frame;
xvid_enc_stats_t xvid_enc_stats;

/* Version for the frame and the stats */
memset(&xvid_enc_frame, 0, sizeof(xvid_enc_frame_t));
```

```
xvid_enc_frame.version = XVID_VERSION;
memset(&xvid_enc_stats, 0, sizeof(xvid_enc_stats_t));
xvid_enc_stats.version = XVID_VERSION;

/* Bind output buffer */
xvid_enc_frame.bitstream = bitstream;
xvid_enc_frame.length = -1;

/* Bind input buffer */
xvid_enc_frame.input.plane[0] = image;
xvid_enc_frame.input.csp = CSP;
xvid_enc_frame.input.stride[0] = CIF_ROW;

/* Set up core's general features */
xvid_enc_frame.vol_flags = 0;
xvid_enc_frame.vop_flags = 0;

/* Set up motion estimation flags */
xvid_enc_frame.motion = 0;

/* Frame type -- let core decide for us */
xvid_enc_frame.type = XVID_TYPE_AUTO;

/* Force the right quantizer -- It is internally managed by RC plugins */
xvid_enc_frame.quant = 0;

xvid_encore(enc_handle, XVID_ENC_ENCODE, &xvid_enc_frame,
            &xvid_enc_stats);
```

在上述示例代码中，image 中保存的是输入图像；bitstream 用于保存编码后的码流；大部分 xvid_enc_frame 中的成员都采用默认设置，实际上的设置工作集中在设定输入图像和输出流上。设置好 xvid_enc_frame 之后，调用 xvid_encore()函数，其中第一个参数 enc_handle 是创建 XviD 编码实例后返回的实例句柄，编码状态通过 xvid_enc_stats 返回。

## 9.3 销毁 xvid 编码实例

销毁 XviD 编码实例非常简单。做法如下：将 xvid_encore()函数的第一个参数 handle 设为创建 XviD 编码实例后得到的句柄；第二个参数 opt 设置为 XVID_ENC_DESTROY；参数 param1 和 param2 都设为 NULL。销毁 XviD 编码实例之后，不能再用它进行编码。

示例代码：

```
xvid_encore(enc_handle, XVID_ENC_DESTROY, NULL, NULL);
```

上述代码中的第一个参数 enc_handle 是创建 XviD 编码实例之后得到的句柄。

# 10 总结

XviD 的三个核心 API 接口都采用了同样的参数形式（只是形式上相同），不可否认，XviD 采用这样的处理方式，使得应用程序接口非常清晰，非常便于使用。这为通过 XviD 编写应用程序的开发人员提供了不少方便。但是一个比较不好的地方是：到目前为止，还没有看到 XviD 的官方文档。显然，接口清晰易懂并不是不给出官方文档的理由。

另外，在通过 XviD 进行编码的过程中，可以插入相关的插件完成一些额外的操作，比如通过插件对编码情况进行统计或者将编码前后的码流转储到文件中以便于调试或者分析。这些插件函数和前面说明的三个核心函数具有相同的接口形式。XviD 本身定义了六个标准的插件函数，用于计算 psnr（Peek Signal Noise Rate，峰值信噪比，一种通用的衡量视频质量的方法）以及进行码率控制等工作。在本文中没有对这些插件函数进行说明，本文档的下一版本中也许会增加对这些插件函数的说明。