DivX MPEG-4 Codec and Its Interface

March 13, 2002

Version	Key Author	Date Prepared	Reviewed by	Review Date
1.0 - Initial Draft	Adam Li	6/24/2001	Eugene Kuznetsov	5/15/2001
1.1 – Decore Operation	Andrea Graziani	7/3/2001	Eugene Kuznetsov	7/10/2001
1.2 – 5.0 Encore	Adam Li	3/10/2002	John Funnell Eugene Kuznetsov Andrea Graziani Darrius Thompson	3/11/2002 3/12/2002 03/13/2002

TABLE OF CONTENTS

1. Introduction

The DivX MPEG-4 Codec is a Windows installable driver. In other words, it is a DLL (dynamic linked library) that can be installed into the Windows system, and be loaded and called by any Windows applications that need to access the function of the codec.

The name of the DivX MPEG-4 Codec DLL is DivX.dll.

The center of DivX MPEG-4 Codec is the DivX codec core. It is the engine of the codec. The codec core processes either the video image or MPEG-4 bitstream, and it uses the compression and decompression to convert information between the formats. The codec core includes two parts – an encoder (that compresses the video image into MPEG-4 bitstreams) and a decoder (that decompresses the MPEG-4 bitstream back into video images). The encoder core is named "encore", and the decoder core "decore".

The DivX.dll encapsules the DivX codec core, so the Windows application can access the codec core through the Windows installable driver interface. In addition, application can also access the codec core functionalities by directly call the codec core functions.

This document discribes the interface into DivX MPEG-4 Codec through both Windows installable driver interface and the codec core interface.

2. Interface Through the Windows Installable Driver

As specified by Windows, the entry point to the DivX MPEG-4 Codec through the Windows installable driver interface is through the DriverProc() function. DivX MPEG-4 Codec follows and processes all the Video for Windows (VfW) messages. For more details on the messages, please refer to the the page on Microsoft site:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/hh/multimed/avifile_8dgz.asp

In particular, message ICM_CONFIG will instruct the codec DLL to pop up a configuration window, enabling users to configure the process and parameters for the DivX MPEG-4 Codec. The significance of the parameters can be found in later sections of this document.

3. Interface Through the Codec Core Interface - Operation Overview

Before we go into the detail of the interface through the codec core functions, first let us go over a brief overview on the operation process of the codec core.

The encoding by the codec will have the following process:

For single-pass encoding:

1. The encore() is called to initialize a new instance and its coding parameters, references and other necessary information.
2. The encore() is called once for each frame to be encoded. The input will be the video frame to codec and its coding parameter. The output will be the compressed MPEG-4 bitstream.
3. After all the video frames are completed. The encore() is called one more time to end the instance and release all the resources allocated for it.

For two-pass encoding:

The above single-pass encoding will be executed twice. In the first pass, the codec will measure and record the complexity of the video (without writing the actually bitstream). The result is the analyzed to determine the best parameters for each frame of encoding (currently this is done outside of encore). In the second pass, the codec will encode the video accordingly and output the actual MPEG-4 bitstream.

The decoding by the codec will have the following process:

1. The decore() is called to initialize a new instance and its coding parameters, references and other necessary information.
2. The decore() is called once for each frame to be encoded. The input will be the compressed MPEG-4 bitstream.The output will be the decodec video frame.
3. After the entire bitstream is completed. The decore() is called one more time to end the instance and release all the resources allocated for it.

4. Codec Core Interface Prototype

The interface has the following prototype:

```
// the prototype of the encore() - main encode engine entrance
int encore(
      void *handle,           // handle - the handle of the calling entity
      unsigned long enc_opt,  // enc_opt      - the option for encoding, see below
      void *param1,           // param1 - the pointer to input parameter structure
      void *param2            // param2 - the pointer to output parameter structure
);

// the prototype of the decore() - main encode engine entrance
int decore(
      void *handle,           // handle - the handle of the calling entity
      unsigned long dec_opt,  // enc_opt      - the option for decoding, see below
      void *param1,           // param1 - the pointer to input parameter structure
      void *param2            // param2 - the pointer to output parameter structure
);
```

handle is a unique unsigned long interger assigned by the codec core for each encore / decore instance. The codec core will remember the corresponding coding parameters and reference pictures for each unique handle. It can be any value as long as it is unique. The calling application / thread that calls the codec core will provide with the handle so the core knows which instance reference this coming operation is associated with. The handle is set and returned to the calling thread when an initialization request is sent to the codec.

enc_opt / dec_opt are the parameters to instruct the core on the operations it needs to perform. param1 / param2 are two parameters whose meaning depending on the operations. Usually, param1 passes a pointer to the parameter structure that inputs to the codec, and param2 passes a pointer to the parameter structure that outputs from the codec.

5. Encore Operation

The encore has the following encoding options.

```
// encore options (the enc_opt parameter of encore())
#define ENC_OPT_INIT          0  // initialize the encoder, return a handle
#define ENC_OPT_RELEASE       1  // release all the resource associated with the handle
#define ENC_OPT_ENCODE        2  // encode a single frame in one-pass mode
#define ENC_OPT_ENCODE_VBR    3  // encode a single frame
                                 // not using internal rate control algorithm
#define ENC_OPT_VERSION       4  // return information on version of codec interface
#define ENC_OPT_INTERNAL_ERROR_INFO     5      // currently not used
```

The encore return value takes the following values.

```
// return code of encore()
#define ENC_BUFFER            -2
#define ENC_FAIL              -1
#define ENC_OK                0
#define ENC_MEMORY            1
#define ENC_BAD_FORMAT        2
#define ENC_INTERNAL          3
```

5.1. Encore – Initialization

Before initialization, caller should verify that version of codec binary interface corresponds to the version of used header file. To do this, version of codec binary interface can be retrieved using the following sample code:

```
int iVersionBinary = encore(0, ENC_OPT_VERSION, 0, 0);
int iVersionHeader = ENC_VERSION;
if(iVersionBinary != iVersionHeader)
// … interfaces are incompatible, encoding can't be performed
        return –1;
```

Encore initialization process goes when ENC_OPT_INIT is set at enc_opt. The encore will initialize a new instance associated with handle.

When ENC_OPT_INIT is set, the calling thread need to provide param1 pointing to the following data structure. param2 has no meaning and should be set to NULL.

```
typedef struct _ENC_PARAM_ {
      int x_dim;               // the x dimension of the frames to be encoded
      int y_dim;               // the y dimension of the frames to be encoded
      float framerate;         // the frame rate of the sequence to be encoded
      long bitrate;            // the bitrate of the target encoded stream
      int rc_period;           // the intended rate control averaging period
      int rc_reaction_period;  // the reaction period for rate control
      int rc_reaction_ratio;   // the ratio for down/up rate control
      int max_quantizer;       // the upper limit of the quantizer
      int min_quantizer;       // the lower limit of the quantizer
       int max_key_interval;    // the maximum interval between key frames
      int deinterlace;         // fast deinterlace
      int quality;             // the quality of compression ( 1 - fastest, 5 - best )
      void *handle;            // the empty handle, which will be filled by encore
      EXTENSIONS extensions;
} ENC_PARAM;
```

The structure EXTENSIONS is defined below.

Encore returns ENC_OK if all the encore process goes all right. Encore returns ENC_MEMORY if there is a memory allocation error.

At least the parameters x_dim and y_dim must be initialized (valid range: 0<x_dim<=1920, 0<y_dim<=1280, both dimensions should be even). The other parameters can be set to 0, in which case they'll be initialized to default values, or can be specifyed directly.

The rc_period is the averaging period for the rate control algorithm, or how fast the RC forgets the rate history. A larger rc_period value usually results in more accurate overall rate. However, it should not be too large compared to the length of the sequence. A common value used is 2000.

The rc_reaction_period controls how fast the RC adopt to recent scenes. A larger rc_reaction_period value usually results in better high motion and worse low motion scene. A common value used is 10.

The rc_reaction_ratio controls the relative sentivity in reaction to high or low motion scenes. A larger rc_reaction_ratio value uaually results in better high motion scene but larger bit consumption.
A common value used is 20.

The max_key_interval sets the maximum interval between the key (INTRA) frames. In the one-pass mode, the key frame is automatically inserted in the encoding when codec detects scene change. In the case that the scene goes a long stretch without a cut, a key frame will be inserted to insure the interval to be always less or equal than the set maximum key frame interval.

The deinterlace option is currently ignored by x86 non-MMX machines.

The quality parameter determines the motion estimation algorithm the encore will perform on the input frames. For the higher quality settings, more thorough motion search will be performed. This will usually results in a better match of the blocks, and hence fewer bits needed for coding the residue texture errors. In the other words, the quality of the decoded video will be better for the same resulting bitrate.

```
typedef struct
{
	int non_mpeg_4;
	int use_bidirect;
	int obmc;
	int data_partitioning;
	int mpeg_2_quant;
	int quarter_pel;
	int bidir_quant_multiplier;// / 0x1000
	int intra_frame_threshold; // 1 ... 100, 0 default value ( 50 )
	int psychovisual;
	double pv_strength_frame;
	double pv_strength_MB;
	int testing_param;		/* only used in testing */
	int use_gmc;
	int gmc_sensitivity;

	/* IVTC/deinterlace */
	int interlace_mode;

	/* crop/resize things */
	int enable_crop;
	int enable_resize;
	int resize_width;
	int resize_height;
	int crop_left;
	int crop_right;
	int crop_top;
	int crop_bottom;
	int resize_mode;		// 0==bilinear, 1==bicubic
	double bicubic_B;		// spline parameter
	double bicubic_C;		// spline parameter
```

```
        /* tsfilter things */
        int temporal_enable;
        int spatial_passes ;
        double temporal_level ;
        double spatial_level  ;

        const char* mv_file;
        int mv_direction;
        // also here:
        // any other special features
} EXTENSIONS;
```

The signal parameters in the following should be set to non-zero values to make signal activate the feature, and zero values to signal not-activate of the feature. The non_mpeg_4 parameter signals special testing coding mode that is not MPEG-4 compliant. The use_bidirect parameter signals using of B-frame video coding. The bidir_quant_multiplier signals the multiplier used for the B-frame quantizer relative to the I and P-frames times 0x1000. For example, a multiplier of 0x2000 means the quantizer used for the B-frames are 2 times the quantizer used in P-frame that is following the B-frame. The obmc parameter signals using of over boundary motion compensation in coding. It is not implemented yet. The data_partitioning signals using of data partitioning. The mpeg2_quant signals using of mpeg2 quantizer. The quarter_pel signals using of quarter pixel motion compensation mode. The intra_frame_threshold signals a threshold for the codec to decide between I and P frames. The default value is 50. The psychovisual parameter signals the using of psychovisual enhancement mode. The pv_strength_frame and pv_strength_MB convey the strength of the psychovisual enhancement at frame and MB level. The value should be between 0 and 1. The testing_parameter should be set to 0. The use_gmc parameter signals the using of global motion compensation in the codec. The gmc_sensitivity gives the sensitivity that will be used in global motion compensation decisions. It is currently unused.

The interlace_mode may be set to one of three settings:

0 = All frames as progressive – no processing here,
1 = All frames as interlaced - an adaptive frame deinterlace will be applied to every frame
2 = Intelligent IVTC / deinterlace - warrants its own paragraph…

The Intelligent IVTC deinterlace is a special mode of the codec that allows it to process content that is any mixture of interlaced, frame progressive and 3:2 pulldown material. Interlaced sections will be deinterlaced and encoded at frame rate. Frame progressive sections are encoded as-is. An IVTC process is applied to 3:2 pulldown material before encoding. The IVTC processes means that the one-to-one relationship between input bitmaps and output encoded frames is broken.

The encoder is able to crop and / or resize the image prior to encoding. To crop the image, first set enable_crop to 1 (default is 0 – off). The four crop parameters crop_left, crop_right, crop_top and crop_bottom must then be set. For example, setting crop_top = 8 and the others to 0 will remove the top 8 lines from the picture.

By default, the size of the encoded image will be the dimensions of the input image after subtracting any crop values. To encode at a different resolution, set enable_resize = 1 (default is 0 – off) and the resize_width and resize_height to the desired encoding resultion. Currently, two resize algorithms are implanted: Bilinear and bicubic. To use the bilinear resize, set resize_mode = 0 (default). For the bicubic resize, set resize_mode = 1. The constants bicubic_B and bicubic_C are spline parameters which influence the characteristics of the resized image. It is recommended to use the Cartmull-Rom Spline (resize_mode = 0, bicubic_B = 0, bicubic_C = 0.5) or the bilinear algorithm (resize_mode = 1) according to personal preference.

Two noise-reduction prefilters are incorporated into the encoder. One is a temporal filter and the other spatial. To enable the temporal filter, set temporal_enable = 1 (default is 0 – off). spatial_level should then be set in the range 0.0 (off) to 1.0 (full strength spatial filtering). To enable the spatial filter, set

spatial_passes in the range 1 to 3 (default is 0 – off). Likewise, then set spatial_level in the range 0.0 (off) to 1.0 (full strength spatial filtering).

The string mv_file gives the name of the file that codec used to same some coding information in the first pass of the 2-pass encoding. The same file will be provided in the second pass to provide the information to expedite the encoding process.

5.2. Encore – Encoding

Encore encoding process starts when ENC_OPT_ENCODE or ENC_OPT_ENCODE_VBR is set at enc_opt. The encore will encode the input video frame using the coding parameter and reference frame associated with the handle.

When ENC_OPT_ENCODE is set, encore will analyze the input frame and automatically detect the scene changes. The quant and intra instruction inputs (see below) will be ignored.

When ENC_OPT_ENCODE_VBR is set, encore will encode the input frame following the quant and intra instructions in the input.

In this operation, the calling thread need to provide param1 and param2 pointing to the following data struture.

```
typedef struct _ENC_FRAME_ {
      void *image;        // the image frame to be encoded
      void *bitstream;    // the buffer for encoded bitstream
      int length;         // the length of the encoded bitstream
      int colorspace;     // the format of image frame
      int quant;          // quantizer for this frame; only used in VBR modes
      int intra;          // force this frame to be intra/inter; only used in VBR 2-pass
      const void *mvs;    // optional pointer to array of motion vectors ( currently
unused )
} ENC_FRAME;
```

The image points to the input bitmap. The bitstream points to a buffer large enough to hold the output MPEG-4 bitstream. Checks for buffer overflow are too expensive and it will be almost impossible to recover from such overflow. Thus, no checks for buffer overflow will be done. Theoretical upper limit of frame size is around 6 bytes / pixel or 2.5 Mb for 720x576 frame. On success, encore will also set length to indicate how many bytes are written into the bitstream buffer.
The colorspace indicate the color space the input image is in. The value of colorspace must be one of the following.

```
#define ENC_CSP_RGB24    0    // common 24-bit RGB, ordered as b-g-r
#define ENC_CSP_YV12     1    // planar YUV, U & V subsampled by 2 in both directions,
    average 12 bit per pixel; order of components y-v-u
#define ENC_CSP_YUY2     2    // packed YUV, U and V subsampled by 2 horizontally,
    average 16 bit per pixel; order of components y-u-y-v
#define ENC_CSP_UYVY     3    // same as above, but order of components is u-y-v-y
#define ENC_CSP_I420     4    // same as ENC_CSP_YV12, but chroma components are
    swapped (order y-u-v)
#define ENC_CSP_IYUV     ENC_CSP_I420
#define ENC_CSP_RGB32    5    // 32-bit RGB, order of components b-g-r, one byte unused
```

Encoder is most effective in modes ENC_CSP_I420 and ENC_CSP_YV12. Conversion from mode ENC_CSP_UYVY is currently not optimized.

When encoding is performed with ENC_OPT_ENCODE, quant and intra fields of ENC_FRAME structure are ignored. Encore2 provides a possibility to more accurately control the encoding process. To use this feature, you have to pass ENC_OPT_ENCORE_VBR as an argument of encore(). In this case, the quant

instruct the encore to encode the current frame with the specified quantizer. Valid range of this field is 1 to 31, with 1 giving highest quality and 31 giving lowest bitstream size. The Intra forces the current frame to be encoder as INTRA frame (when Intra = 1) and INTER frame (when Intra = 0). When Intra is set to –1, the internal decision method is adopted.

The mvs member allows to improve encoder performance by specifying motion information. This feature can be found useful for MPEG-1/2 to DivX transcoders, but most probably you don't need it. See mv_hint.h for detailed information about it.

The ENC_RESULT parameter is used for the encore to return some results of the encoding operation.

```
typedef struct _ENC_RESULT_ {
      int isKeyFrame;           // the current frame is encoded as a key frame
      int quantizer;            // the quantizer used to encode the current frame
      int texture_bits;         // the number of bits used for texture coding
      int motion_bits;          // the number of bits used for motion vectors
      int total_bits;           // the total number of bits used for the current frame
} ENC_RESULT;
```

The isKeyFrame variable is set to 1 if the currented frame is encoded as a key frame, otherwise it is set to 0.

Encore returns ENC_OK if all the encore process goes all right. Encore returns ENC_BAD_FORMAT if the input frame format does not match the format set at initialization with the current handle.

When the interlace_mode is set to 2 (“Intelligent IVTC / deinterlace”) the one-to-one relationship between bitmaps and encoded frames is broken. For example, several ENC_OPT_ENCODE operations may be performed before the encoder begins to return bitstream. Similarly, calling ENC_OPT_ENCODE with a new frame may allowencore to encode two or more frames due to the ivtc process. The ENC_OPT_ENCODE calling convention has been enhanced to deal with this special case. If the encoder cannot return compressed bitstream due to the ivtc process, then it returns with ENC_FRAME::length < 0. If the encoder does produce bitstream (i.e. ENC_FRAME::length >= 0), then it is possible that it can produce a second or third frame. Repeated calls should be made to encore, with ENC_FRAME::image set to 0, until ENC_FRAME::length < 0.

5.3. Encore – Release

Encore releasing process starts when ENC_OPT_RELEASE is set at enc_opt. The encore will purge all the information and release all the resources allocated for handle, and delete handle from its database.

When ENC_OPT_RELEASE is set, both param1 and param2 have no meaning and should be set to NULL.

Encore returns ENC_OK.

6. Decore2 Operation

The decore has the following decoding options.

```
// decore options
#define DEC_OPT_MEMORY_REQS      0
#define DEC_OPT_INIT             1
#define DEC_OPT_RELEASE          2
#define DEC_OPT_SETPP            3 // set postprocessing mode
#define DEC_OPT_SETOUT           4 // set output mode
#define DEC_OPT_FRAME            5
#define DEC_OPT_GAMMA            7
#define DEC_OPT_VERSION          8
#define DEC_OPT_INIT_VOL         12
#define DEC_OPT_CONVERTYUV       13
#define DEC_OPT_CONVERTYV12      14
#define DEC_OPT_GETVOLINFO       15
```

The decore returns the following values.

```
// decore return values
#define DEC_OK            0
#define DEC_MEMORY        1
#define DEC_BAD_FORMAT    2
#define DEC_EXIT          3
```

6.1. Decore – Initialization

Before initializing decore(), the application should allocate the data structures. To do this, the application will call decore() setting DEC_OPT_MEMORY_REQS as dec_opt. When DEC_OPT_MEMORY_REQS is set, the calling thread need to provide the input parameter (param1) pointing to a DEC_PARAM data structure and the output parameter (param2) pointing to a DEC_MEM_REQS structure. The two data structures mentioned are defined below.

```
typedef struct _DEC_PARAM_
{
    int x_dim;            // x dimension of the frames to be decoded
    int y_dim;            // y dimension of the frames to be decoded
    int output_format;    // output color format
    int time_incr;
    int codec_version;
    int build_number;
    DEC_BUFFERS buffers;
} DEC_PARAM;

typedef struct _DEC_MEM_REQS_
{
    unsigned long mp4_edged_ref_buffers_size;
    unsigned long mp4_edged_for_buffers_size;
    unsigned long mp4_display_buffers_size;
    unsigned long mp4_state_size;
    unsigned long mp4_tables_size;
    unsigned long mp4_stream_size;
    unsigned long mp4_reference_size;
} DEC_MEM_REQS;
```

The application must provide a meaningful param1 containing the correct dimension of the frames. The decore will give back to the application information on the buffers and data structure size are needed throught param2. The application must also set the codec_version and the build_number of the codec that has been used to encode the stream that must be decoded by decore. A build_number of 0 will be ignored. If a stream claim conformance to the ISO/IEC 14496-2 standard, it's suggested to set as codec_version 500 (last version of the codec).

At this point the application should allocate the required memory and then initialize decore.
The memory that decore will use is described in the DEC_BUFFERS structure, defined as:

```
typedef struct _DEC_BUFFERS_
{
    void * mp4_edged_ref_buffers;
    void * mp4_edged_for_buffers;
    void * mp4_edged_back_buffers;
    void * mp4_display_buffers;
    void * mp4_state;
    void * mp4_tables;
    void * mp4_stream;
    void * mp4_reference;
} DEC_BUFFERS;
```

The time_incr field in DEC_PARAM indicates the number of evenly spaced ticks within one modulo time. This information is usually present in the VOL header of each stream (each stream contains one instance of the VOL). A default value for time_incr should always be indicated.

Decore initialization process goes when DEC_OPT_INIT is set at dec_opt. The decore will initialize a new instance associated with handle.

When DEC_OPT_INIT is set, the calling thread need to provide param1 pointing to a valid DEC_PARAM data structure. In this case, param2 has no meaning and should be set to NULL.

Decore returns DEC_OK if all the decore process goes all right. Decore returns DEC_MEMORY if there is memory allocation error.

The output value must be choosed between the following valid formats:

```
// supported output formats
#define DEC_YUY2           1
#define DEC_YUV2           DEC_YUY2
#define DEC_UYVY           2
#define DEC_420            3
#define DEC_RGB32          4
#define DEC_RGB32_INV      5
#define DEC_RGB24          6
#define DEC_RGB24_INV      7
#define DEC_RGB555         8
#define DEC_RGB555_INV     9
#define DEC_RGB565         10
#define DEC_RGB565_INV     11
#define DEC_USER           12
#define DEC_YV12           13
#define DEC_ARGB           14
```

DEC_YUY2 and DEC_UYVY are packed YUV formats. DEC_420 is planar YUV with chrominance planes subsampled by 2 in both directions. DEC_RGB* formats correspond to different flavors of RGB with or without vertical flipping of the output.

The last format (DEC_USER) provides user with ability to manually perform colorspace conversion with optimal efficiency. See next paragraph for more details.

DEC_ARGB output is the RGB 32bit output with the last byte set to opaque (255) instead of transparent (0) like in DEC_RGB32.

6.2. Decore – Decoding

Decore decoding process starts when DEC_OPT_FRAME is set at dec_opt. The decore will decode the input bitstream using the coding parameter and the reference frame associated with the handle.

In this operation, the caller need to provide param1 pointing to the following data struture. param2 has no meaning and should be set to NULL.

```
typedef struct _DEC_FRAME_ {
       void *bmp;          // the 24-bit bitmap to be encoded
       void *bitstream;    // the buffer for encoded bitstream
       long length;        // the length of the encoded bitstream
       int render_flag;    // 1: render the bitmap, 0: avoid to render the output
       int stride;         // decoded bitmap stride
} DEC_FRAME;
```

The bitstream points to a buffer holding the output MPEG-4 bitstream. The length indicates how many bytes in the bitstream buffer holds the actual bitstream. The bmp points to a bitmap to hold the output image. The render_flag is used to speed up the decoder if it is in late. In particular, if render_flag is 0, the decoder will not produce a valid bmp. The renderer should avoid as a consequence to display the returned bmp. The stride of the output bitmap must be indicated using the appropriate field.

In a special case of color output format DEC_USER, bmp pointer is treated as a pointer to the structure in the following format:

```
typedef struct _DEC_PICTURE_
{
       void *y;
       void *u;
       void *v;
       int stride_y;
       int stride_uv;
} DEC_PICTURE;
```

Its members will be filled after successful decompression of the output. Fields y, u and v will contain pointers to internal decoder memory buffers, and fields stride_y and stride_uv will contain strides of these buffers (distances in bytes between sequential scanlines). Caller will need to perform clipping and color space conversion by himself.

Warning: these pointers may be valid only until the next call to decore(). Strides may be larger from dimensions of image. Avoid using return values if you passed render_flag=0.

Decore returns DEC_OK if all the decore process goes all right. Decore returns DEC_BAD_FORMAT if the input bitstream does not match the format set at initialization with the current handle.

6.3. Decore – Release

Decore releasing process starts when DEC_OPT_RELEASE is set at dec_opt. The decore will delete handle from its database. After realeasing decore the application should free the allocated data structures and buffers.

When DEC_OPT_RELEASE is set, both param1 and param2 have no meaning and should be set to NULL.

Decore returns DEC_OK.

6.4. Decore – Decoding DivX ;-) 3.11

The current release of decore is also compatible with DivX ;-) 3.11 (also known as MS MPEG-4 v3) video format. Decoding of this format is done in the same way as of DivX 4.x and DivX 5.x, you just have to indicate 311 as codec_version in the structure DEC_PARAM when the decoder is initialized.

6.5. Decore – Extra Settings

6.5.1. Postprocessing Settings

It is possible to change the output format or the postprocessing level of the decoder using as decoding options DEC_OPT_SETOUT, DEC_OPT_SETPP or DEC_OPT_SETPP_ADV.
In particular:

To change the output format of the decoder, the user must set the DEC_PARAM structure according to the output format he wants the decoder to write in bmp, then call decore passing as param1 the DEC_PARAM structure and setting DEC_OPT_SETOUT as decoder option. Here's an example:

```
{
   DEC_PARAM DecParam;
   DecParam.color_depth = 0;
   DecParam.output_format = DEC_RGB32;

   decore(NULL, DEC_OPT_SETOUT, &DecParam, NULL);  // tell decore to output in RGB32 mode

  ...
}
```

To change the postprocessing level of the decoder the user must set the DEC_SET structure (field postproc_level) in accordance with the postprocessing level desired and call the decore API passing as param1 the DEC_SET structure and setting DEC_OPT_SETPP as decoder option.

The DEC_SET structure is defined as:

```
typedef struct _DEC_SET_
{
   int postproc_level; // valid interval are [0..100]

   int deblock_hor_luma;
   int deblock_ver_luma;
   int deblock_hor_chr;
   int deblock_ver_chr;
   int dering_luma;
   int dering_chr;

   int pp_semaphore;} DEC_SET;
```

Note that the valid value for postproc_level are integer numbers between 0 and 100. The other fields are currently not used.
Here's a code example:

```
{
   DEC_SET dec_set;
   dec_set.postproc_level = m_iPPLevel;

   decore(NULL, DEC_OPT_SETPP, &dec_set, NULL);
}
```

6.5.2. Decore Version

When the decoder is called specifying as option DECORE_VERSION, it will return a number that indicates the version of the decoder. The number is composed by 8 digits (aaaammdd) that indicates respectively the year, the month and the day in which the library has been released.

6.5.3. Gamma

The option DEC_OPT_GAMMA permits to modify some properties of the output bitmap like brightness, contrast and saturation. The decoder will interpret param1 as gamma correction index and param2 as correction value. The index indicates which property of the output bitmap must be modified as indicated in the following table:

DEC_GAMMA_BRIGHTNESS	0
DEC_GAMMA_CONTRAST	1
DEC_GAMMA_SATURATION	2

The value field can be any number between –127 and +127.

6.5.4. Init VOL

The option DEC_OPT_INITVOL indicates to the decoder that the stream passed contains only the VOL header. This option can be used to update status information of the decoder as picture width, picture height and time increment resolution.

6.5.5. YUV Color Conversion

When decore can be used to perform color plane conversion using as option DEC_OPT_CONVERTYUV or DEC_OPT_CONVERTYV12. Decoder will receive as param1 a structure of type DEC_FRAME and as param2 the YUV plane (in format 420) to be converted. According to the output format information passed to the decoder during the initalization, the decoder will write in the destination bitmap of DEC_FRAME the color converted plane.

6.5.6. Get Vol Info

Using the option DEC_OPT_GETVOLINFO, the decoder can be used to parse a VOL header and get useful information out of it. The decoder will parse the stream passed to it up to the VOP header and will fill the structure of type DEC_PARAM , passed as param2. Here's an example:

```
{
   DEC_FRAME dec_frame;
   DEC_PARAM dec_param;

   decFrame.bitstream = m_InputStream;
   decFrame.bmp = 0; // not used
   decFrame.length = m_StreamLength;
   decFrame.render_flag = 0;
   decFrame.stride = 0;

   decore(NULL, DEC_OPT_GETVOLINFO, &dec_frame, &dec_param);

   // the decoder returns useful information as dec_param.x_dim, dec_param.y_dim, ...
}
```

6.6. Decore – Example

This is a simple example on how to initialize and release decore():

```
...

DEC_MEM_REQS decMemReqs;
DEC_PARAM decParam;

decParam.x_dim = m_FrameWidth;
decParam.y_dim = m_FrameHeight;
decParam.output_format = m_OutputFormat;
decParam.codec_version = 412; // indicates that the stream is DivX 4.12 compatible
decParam.build_number = 0; // in this case, the build field is ignored
decParam.time_incr = 15; // time_incr default value

decore((long) this, DEC_OPT_MEMORY_REQS, &decParam, &decMemReqs);

// the application allocates the data structures and the buffers
decParam.buffers.mp4_edged_ref_buffers = malloc(decMemReqs.mp4_edged_ref_buffers_size);
decParam.buffers.mp4_edged_for_buffers = malloc(decMemReqs.mp4_edged_for_buffers_size);
decParam.buffers.mp4_edged_back_buffers = malloc(decMemReqs.mp4_edged_back_buffers_size);
decParam.buffers.mp4_display_buffers = malloc(decMemReqs.mp4_display_buffers_size);
decParam.buffers.mp4_state = malloc(decMemReqs.mp4_state_size);
decParam.buffers.mp4_tables = malloc(decMemReqs.mp4_tables_size);
decParam.buffers.mp4_stream = malloc(decMemReqs.mp4_stream_size);
decParam.buffers.mp4_reference = malloc(decMemReqs.mp4_reference_size);

memset(decParam.buffers.mp4_state, 0, decMemReqs.mp4_state_size);
memset(decParam.buffers.mp4_tables, 0, decMemReqs.mp4_tables_size);
memset(decParam.buffers.mp4_stream, 0, decMemReqs.mp4_stream_size);
memset(decParam.buffers.mp4_reference, 0, decMemReqs.mp4_reference_size);

decore((long) this, DEC_OPT_INIT, &decParam, NULL);

// decode frames
{
        DEC_FRAME decFrame;

        decFrame.bitstream = m_InputStream;
        decFrame.bmp = m_OutputBmp;
        decFrame.length = m_StreamLength;
        decFrame.render_flag = 1;
        decFrame.stride = m_OutputBmpWidth;

        while ( decore((long) this, DEC_OPT_FRAME, &decFrame, NULL) == DEC_OK )
                ;
}

decore((long) this, DEC_OPT_RELEASE, NULL, NULL);

free(m_decParam.buffers.mp4_display_buffers);
free(m_decParam.buffers.mp4_edged_for_buffers);
free(m_decParam.buffers.mp4_edged_back_buffers);
free(m_decParam.buffers.mp4_edged_ref_buffers);
free(m_decParam.buffers.mp4_reference);
free(m_decParam.buffers.mp4_state);
free(m_decParam.buffers.mp4_stream);
free(m_decParam.buffers.mp4_tables);

m_decParam.buffers.mp4_reference = NULL;
```