1.   PlayStation MDEC Still Image Compression & Decompression

The PlayStation MDEC is designed to quickly decode data encoded using a discrete cosine transform method. This is an encoding scheme that is used for both still images and video. The PlayStation MDEC performs the calculation-intensive inverse DCT operation used to decode these images at high speed.

Perhaps the main use of the PlayStation MDEC is to decode data used for playback of full-motion video animations, but the MDEC is not restricted to video data. One of common questions asked by PlayStation developers is how does one use the MDEC to decompress still images quickly and easily? The streaming movie format used for full-motion video can also be used to store and retrieve still images, but there's an easier way. In this document, we'll tell you a little bit more about the MDEC compression method and go through the steps for creating and displaying an MDEC still image.

1.1.   What is an MDEC Still Image?

An MDEC still image is a bitmapped graphic that has been compressed using the MOVCONV tool or similar utility. The compression method itself is very similar to the popular JPEG format[1], although the actual data is not directly compatible. The basic steps to encoding an MDEC still image are:[2]

1) Divide the image into 16x16 pixel sections known as "macroblocks". Steps 2 through 6 are then repeated for each macroblock.

2) Convert the color space of the macroblock from RGB to YUV (unless already in YUV). (This step can optionally be performed on the entire image prior to step 1.) This allows the brightness and color portions of the image to be compressed separately.

3) Perform a Discrete Cosine Transform (DCT) on the block. This converts the data from a sequence of pixel values into a matrix of frequency components that describe the colors found in the block.

Two things are worth noting here. First, steps 2 and 3, aside from some inevitable round-off error, are not lossy. If we were to go backwards at this point and decode the image back to the original format, it would still be virtually identical to the original. Second, no data compression has taken place so far. The intermediate stage of the image takes up just as much room as the original.

4) Perform a quantization step on the frequency components. This is the "lossy" stage of the encoding process. The data is altered so that minor variations in color or brightness from one pixel to the next are reduced or eliminated. Because the human eye is more sensitive to changes in brightness than in color, the brightness and color portions of the image are quantized separately, using different quantization tables. This allows for

---

[1] The JPEG and MDEC formats both use a "lossy" image compression method, meaning what you get after decompression is not identical to the image that was originally compressed. The greater the compression ratio, the greater the loss in image quality. MDEC is an acronym for "Motion DECoder" and refers to Sony's flavor of JPEG-style image compression that is used for both still and moving images. JPEG is an acronym for "Joint Photographic Experts Group" which is the name of the group of people that came up with this whole method of compression in the first place.

[2] This is a VERY superficial description of JPEG-style compression. There are all sorts of books and references on the subject if you want to know more.

greater quantization of the color components, which will improve compression without altering the appearance of the image significantly. With moderate compression ratios, the difference is small enough to go unnoticed in anything less than a direct comparison with the original image.

5) Run-length compress the quantized data. This is done in a zig-zag diagonal path from the top left of the matrix. After quanitization, many of the low frequency components in the matrix are reduced to zero, and there are usually other repeating value sequences. This allows for efficient lossless run-length compression.

6) Compress the RLE compressed DCT data using the Huffman coding method, a lossless compression method that is very efficient at compressing data with a lot of repeating values. The data is converted to a stream of bit codes of varying lengths (Huffman is also referred to in the PlayStation documentation as "Variable Length Coding"). The most common values in the original data are encoded using just a few bits of information, while less-frequent values use more bits.

As you can see, the actual compression is done only in steps 5 & 6. The previous steps are done so that the data can be compressed more efficiently.

As we'll discuss in section 1.2, using the MOVCONV utility to do these operations results in a BS (bitstream) file. The BS file format is very simple. It simply contains a small header followed by enough bitstream data to describe an image.

1.2. Creating An MDEC Still Image

Creating an MDEC still image is a simple task using the MOVCONV tool under MS Windows. You're probably already familiar with this tool because it allows you to convert MS Windows AVI movie files into the MDEC-compressed video format used by the PlayStation. It can also be used to compress still images.

Your starting point needs to be either a PlayStation TIM format graphic file or a raw 24-bit RGB or YUV data file.[3] TIM images can be either palette-based or direct 15-bit or 24-bit true color. Palette-based images will be converted to direct mode automatically during the process. Note that everything will later be decompressed into either 15-bit or 24-bit RGB direct mode regardless of the format of the source image.

Note that an image must have a width and height that is a multiple of 16 for it to be properly compressed and decompressed.

1) Run MOVCONV. Select your source image by clicking on the *Ref…* button in the Input section and then using the file selector. For our example, this will be C:\PSX\HALFDOME.TIM. Note that the output filename is already filled in. You should see a window resembling that shown in *Figure 1*.

---

[3] Note that raw 24-bit YUV image files will take slightly less time to convert since the color space conversion mentioned earlier (item #2 in section 1.1) is no longer required. However, the difference will be fairly small.
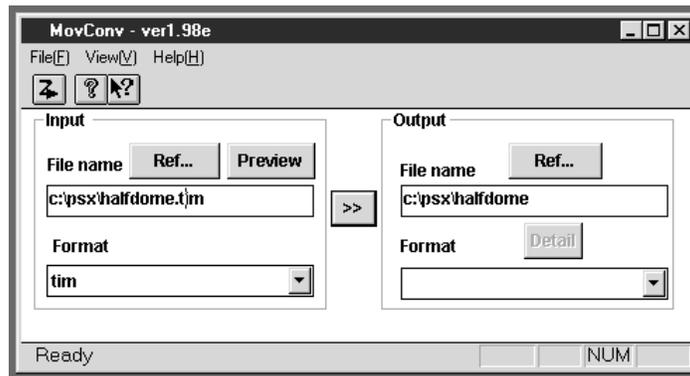
Figure 1 — MovConv Window Example

2) The output filename is not complete because we still haven't specified the output format. Click on the *Format* popup menu in the *Output* section and select the "BS" format, as shown in *Figure 2*.
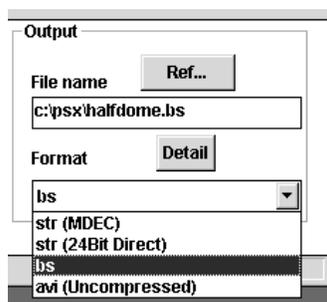
Figure 2 — Output Format Popup Menu

3) Now we must specify the desired size of the compressed image. Click on the *Details* button in the *Output* section of the main window as seen in *Figure 1*. This should result in the *MDEC Parameters* dialog being shown, as seen in *Figure 1*.
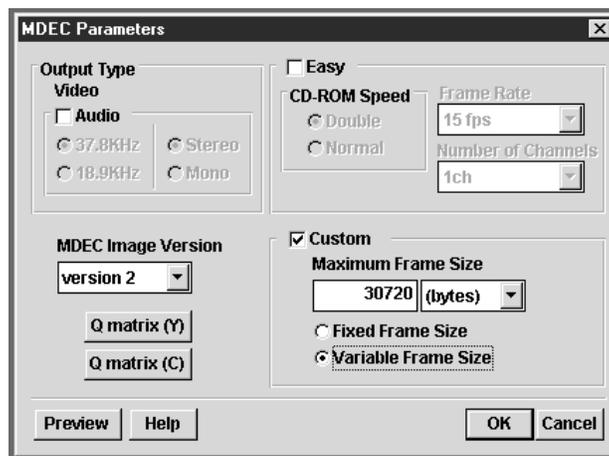
Figure 3 — MDEC Parameters Dialog

4) The only portion of the dialog we are concerned with is the *Custom* section in the bottom right corner. Rather than expressing a compression to quality ratio as a percentage, MOVCONV lets you specify the desired size of the compressed data and adjusts the compression quality as needed to fit. Make sure the *Custom* box is selected.

5) Decide if you want to specify the image size in terms of the number of bytes or the number of CD-ROM sectors. This is controlled by the popup menu underneath and to the right of where it says "*Maximum*

Frame Size". The popup is just a convenience; specifying either 20480 bytes or 10 sectors will provide the same results.[4]

6) We will specify 30720 bytes as our desired compressed image size, so enter "30720" in the left hand box and select "(bytes)" in the right-hand popup menu.

7) Select the *Variable Frame Size* radio button. This will allow MOVCONV to create a smaller file in the event that the image compresses well even with the maximum quality setting.

8) Now we are ready to actually perform the image compression. Click on the ">>" button in the center of the window. A status window will appear to keep you informed about the ongoing status of the operation. Once the compression is completed, you will be returned to the main window to do any additional images.

The result of these steps should be a file named HALFDOME.BS which contains compressed DCT-encoded image data. This can be loaded into memory by your program as required.

1.3. Decompressing An MDEC Still Image

Decompressing an MDEC still image is very simple and requires only a few function calls to LIBPRESS. The first step is to make sure that you have access to your compressed data. It doesn't matter if your data is linked in with your program or loaded from a file on the CD-ROM. For now we'll assume that the compressed data is already loaded and accessed through the *mdec_bitstream* pointer.

The next step is to reset the MDEC so that it is ready to decompress a new image. This is done via:

```
DecDCTReset(0);    /* reset MDEC */
```

The next step is to decompress the bitstream of Huffman-compressed DCT data. This is also referred to in PlayStation documentation as VLC (variable length coding) data. We'll need a buffer to decompress into. Since the DCT data has been RLE-encoded, it won't be as large as the final image, so we don't know yet how big a buffer we'll need. To find out, we use the ***DecDCTBufSize()*** function:

```
mdec_runlevel = malloc( DecDCTBufSize(mdec_bitstream) );    /* Get buffer */
```

Now we have a buffer set up and accessible via the *mdec_runlevel* pointer. Decoding the VLC data is done via:

```
DecDCTvlc(mdec_bitstream, mdec_runlevel); /* decode VLC */
```

This returns control to your program after the DCT data has been extracted. The next step is to tell the MDEC where the DCT-encoded data is located. This is done via:

```
DecDCTin(mdec_runlevel, 0);    /* send run-level info to DCT */
```

Finally, we request that the MDEC decode the DCT data and give it back to us so that it can be displayed. We must give it a pointer to a buffer where the decoded information will be placed and also tell it how many bytes of image data we want to receive.

```
DecDCTout(mdec_image, total);              /* Ask to receive data */
```

---

[4] A CD-ROM data sector is 2048 bytes.

This will start the MDEC decoding the specified amount of data. It will uncompress the RLE-encoded macroblock information, perform an Inverse-DCT on the result, and then convert the data from YUV back to 15-bit RGB direct format. Note that the output will always be 15-bit direct RGB image data, regardless of the format of the original image before it was compressed.

The *DecDCTout()* function is non-blocking, meaning that the MDEC begins the operation and then control is returned immediately back to your program. This means that your program will have to wait for the decoded image data to be available. This is done via:

```
DecDCToutSync(0);                                /* wait for it to finish */
```

Once this call returns, there is image data available in the *mdec_image* buffer that can now be moved to the frame buffer using the *LoadImage* function. Optionally, you can use the *DecDCToutCallback()* function to define a callback routine that will be called when the decoded data is available.

The decoded image is presented as 16-pixel wide vertical strips which run the height of the image. Therefore, larger images are typically copied over in strips to the desired area in the frame buffer. Note that the entire image does not have to be decoded in one step. A smaller buffer may be used if the image is decoded in sections. See the sample program listing in section 1.4 as an example.

1.4. Sample Program Source Code

The sample program below is derived from one included on the PlayStation Programmer's CD. It's a very simple program that does just the bare minimum required to initialize a display and decompress an MDEC-compressed picture onto it. The sections dealing with the MDEC are highlighted in bold.

```
/*      Simple VLC decode and MDEC in-memory decompression
 *
 *      Copyright (C) 1996 by Sony Corporation, All rights Reserved
 */

#include <sys/types.h>
#include <libetc.h>
#include <libgte.h>
#include <libgpu.h>
#include <libpress.h>

/***************************************************************************
 * Image is compressed in BitStream(BS) format & created by MOVCONV.
 * Note that file does not have information about image size.
 ***************************************************************************/

#define DISPLAY_WIDTH (256)
#define DISPLAY_HEIGHT (224)

u_char mdec_bitstream[] = {
#include "halfdo~1.c"
};

u_long mdec_image[16*DISPLAY_HEIGHT];      /* Buffer large enough for 1 strip */
u_long *mdec_runlevel;                      /* Ptr to buffer to receive DCT data */

/*************************************************************************/

main()
{
int        width  = DISPLAY_WIDTH;
int        height = DISPLAY_HEIGHT;
int        i;
```

```
        DECDCTENV    env;
        DISPENV      disp;
        RECT         rect;

            ResetGraph(0);
            SetGraphDebug(0);          /* set debug mode (0:off, 1:monitor, 2:dump) */
            SetDispMask(1);            /* enable to display (0:inhibit, 1:enable) */

    /* clear frame buffer */

            rect.w = 640;      rect.h = 480;
            rect.x =   0;      rect.y =   0;
            ClearImage(&rect, 0, 0, 0);

    /* setup display environment */

            SetDefDispEnv(&disp, 0, 0, width, 240);
            PutDispEnv(&disp);

    /* Setup MDEC */

            DecDCTReset(0);                                  /* Reset MDEC */
            mdec_runlevel = malloc(DecDCTBufSize(mdec_bitstream));
            DecDCTvlc((void *)mdec_bitstream, mdec_runlevel);     /* Decompress DCT */
            DecDCTin(mdec_runlevel, 0);                      /* Point MDEC at DCT data */

    /* Retrieve image in 16-pixel wide strips */

            rect.w = 16;
            rect.h = height;
            rect.y = (240-height)/2;

            for (rect.x = 0; rect.x < width; rect.x += 16 )
            {
                    DecDCTout(mdec_image, (16*height)/2 );    /* Get data */
                    DecDCToutSync(0);                         /* wait */
                    LoadImage(&rect, mdec_image);             /* Draw that strip */
            }

            DrawSync(0);
            StopCallback();
            free( mdec_runlevel );
    }
```