Technical Note TN2044

Decompressing DV frames and accessing the pixels

CONTENTS

This technical note discusses using Decompression Sequences to decompress DV frames to an RGB offscreen destination, for the purpose of accessing each frames pixels directly.

The same steps can be applied to any QuickTime supported compressed image format represented by an Image Description. The compressed image data can be sourced from a QuickTime Movie, the Sequence Grabber or from an application supplied buffer.

[Apr 11 2002]

Introduction

The overall procedure of decompression single DV frames from a buffer, to an RGB offscreen, then accessing the pixels directly is straightforward, especially if you don't care about scheduled decompression. However, there are a couple of different ways to go about it. You could for example, treat each frame as a single image, and use `DecompressImage` or `FDecompressImage` to decompress the "single-frame" into a pixel map. While this is fine for a single image, it is not the fastest way to decompress a series of images of the same format, sharing a common image description represented by the `ImageDescription` structure.

A more efficient method is the use of a Decompression Sequence.

The Image Compression Manager provides a set of APIs allowing developers to decompress sequences of images sharing a common image description. Each image in the sequence is referred to as a frame. A decompression sequence is started by calling DecompressSequenceBeginS, and each frame in the sequence is processed by calling DecompressSequenceFrameWhen. Upon completion, the sequence is closed with a call to CDSequenceEnd. Checking the status of the current operation can be done by calling CDSequenceBusy. There are also a number of DSequence Parameter APIs allowing the manipulation of the parameters controlling the decompression sequence, these include the source rectangle, transformation matrix and accuracy to name a few.

Getting Started:

Determine where to decompress the image, build an image description describing the source data, decide how much of the image to decompress, and build a mapping matrix for the operation. These parameters must be specified in the call to `DecompressSequenceBeginS` when making a decompression request.

The destination is specified as a graphics port and the image source size is described by a rectangle, in the coordinate system of the source image. NULL can be used to specify the entire source image. To support transformations such as scaling to a destination of a different size, specify a matrix describing how the image is to be mapped into the destination graphics port.

> **IMPORTANT:**
> If your code passes in a smaller (eg. quarter-size) source rectangle with the intention of drawing cropped unscaled DV to a smaller (eg. quarter-size) destination, a DV-specific bug in QuickTime 5 will cause this decompression request to be misinterpreted, scaling the frame to fit. The correct interpretation of this request is to draw the top-left corner of the DV frame cropped at normal size.
>
> This bug will be fixed in QuickTime 6. If your code was behaving as intended because of this bug, make sure to fix your code to use a matrix in the call to DecompressSequenceBeginS, scaling the frame to fit the offscreen gworld. This approach will work in all versions of QuickTime.

You can figure out the size of the source image by examining the image description structure associated with the image or

if you are intimately familiar with the image data you can create the image description yourself. The image description structure contains information defining the characteristics of the compressed image or sequence. One image description structure may be associated with one or more compressed frames. See Figure 1.

For a detailed discussion regarding the representation of uncompressed Y´CbCr video in an Image Description structure, including Image Description Extensions refer to Ice Floe #19.

The ImageDescription:

Create an Image Description for your compressed data. See Listing 1.

```
struct ImageDescription {
    long idSize; // total size of ImageDescription including extra data
                 // (CLUTs and other per sequence data)
    CodecType cType;           // what kind of codec compressed this data
    long      resvd1;          // reserved for Apple use
    short     resvd2;          // reserved for Apple use
    short     dataRefIndex;    // set to zero
    short     version;         // which version is this data
    short     revisionLevel;   // what version of that codec did this
    long      vendor;          // whose codec compressed this data
    CodecQ    temporalQuality; // what was the temporal quality factor
    CodecQ    spatialQuality;  // what was the spatial quality factor
    short     width;           // how many pixels wide is this data
    short     height;          // how many pixels high is this data
    Fixed     hRes;            // horizontal resolution
    Fixed     vRes;            // vertical resolution
    long      dataSize;        // if known, the size of data for this
                               // image descriptor
    short     frameCount;      // number of frames this description
                               // applies to
    Str31     name;            // name of codec ( in case not installed )
    short     depth;           // what depth is this data (1-32) or
                               // (33-40 grayscale)
    short     clutID;          // clut id or if 0 clut follows
                               // or -1 if no clut
};
```

Figure 1. Image Description structure.

```
// Create an Image Description describing the compressed data,
// you are responsible for disposing of the returned handle.
// Modify the Image description values to deal with any other
// compressed data formats - for example PAL (kDVCPALCodecType)
ImageDescriptionHandle MakeImageDescriptionForNTSCDV(void)
{
    ImageDescriptionHandle hImageDescription = NULL;

    hImageDescription =
        (ImageDescriptionHandle)
        NewHandleClear(sizeof(ImageDescription));

    if (NULL != hImageDescription) {
        (**hImageDescription).idSize = sizeof(ImageDescription);
        (**hImageDescription).cType = kDVCNTSCCodecType;

                // DV has no temporalQuality
        (**hImageDescription).temporalQuality = 0;
        (**hImageDescription).spatialQuality = codecNormalQuality;
        (**hImageDescription).width = 720;
        (**hImageDescription).height = 480;
        (**hImageDescription).hRes = 72 << 16;
        (**hImageDescription).vRes = 72 << 16;
        (**hImageDescription).frameCount = 1;
        (**hImageDescription).depth = 24;
        (**hImageDescription).clutID = -1;
    }

    return hImageDescription;
}
```

Listing 1. Create an Image Description describing the compressed data.

The Offscreen:

Use `QTNewGWorld` to create an offscreen to decompress into. See Figure 2. Developers should note 24-bit (`k24RGBPixelFormat`) is not the best destination for the DV Codec, 32-bit (`k32ARGBPixelFormat`) is a more efficient (faster) destination.

Make a 32-bit RGB offscreen. See Listing 2.

```
OSErr QTNewGWorld(
    GWorldPtr   *offscreenGWorld, // on return, a pointer to the GWorld
    OSType      PixelFormat;      // the new GWorlds pixel format
    const Rect  *boundsRect,      // boundary and port rectangle
    CTabHandle  cTable,           // a ColorTable - NULL for default
    GDHandle    aGDevice,         // a GDevice - set to NULL
    GWorldFlags flags);           // flags - set to 0 for default
```

Figure 2. `QTNewGWorld` API.

```
// Make a 32bit GWorld for the decompression destination -
// - 720 x 480 for DV. This function locks the pixel map
OSErr MakeGWorld(short inWidth, short inHeight,
                           GWorldPtr *outDestGWorld)
{
    Rect  theBounds = {0, 0};

    OSErr err = noErr;

    theBounds.right  = inWidth;
    theBounds.bottom = inHeight;
    *outDestGWorld = NULL;

    err = QTNewGWorld(outDestGWorld,     // return a pointer to
                                         // the offscreen
                      k32ARGBPixelFormat, // the new GWorlds pixel
                                         // format
                      &theBounds,        // boundry and port rect
                                         // for the offscreen PixMap
                      NULL,              // handle to a ColorTable
                      NULL,              // handle to a GDevice
                      0);                // flags

    if (noErr == err)
        // call LockPixels to prevent the base address for
        // an offscreen pixel image from being moved while you
        // draw into or copy from its pixel map
        LockPixels(GetGWorldPixMap(*outDestGWorld));

    return err;
}
```

Listing 2. Make a 32-bit `GWorld` for the decompression destination.

Setting up the Sequence:

Perform the setup required to decompress an image sequence by calling the Image Compression Manager's `DecompressSequenceBeginS` function. See Figure 3. While `DecompressSequenceBegin` can also be used, it has been superseded by `DecompressSequenceBeginS`.

The `DecompressSequenceBeginS` call is used to initiate the sequence and specifies many of the parameters that control the sequence-decompression operation. The Image Compression Manager allocates system resources necessary for the operation, and returns a `ImageSequence` ID (`seqID`) uniquely identifying the sequence.

Start a decompression sequence. See Listing 3.

```
OSErr DecompressSequenceBeginS(
    ImageSequence          *seqID,   // returns a unique seqID
    ImageDescriptionHandle desc,     // description of
                                     // compressed data
    Ptr                    data,     // pointer to the compressed
                                     // data for preflight
    long                   dataSize, // size of the data buffer.
    CGrafPtr               port,     // the destination port
    GDHandle               gdh,      // GDevice for the destination
    const Rect             *srcRect, // portion of image to
                                     // decompress
    MatrixRecordPtr        matrix,   // transformation to apply
                                     // during decompress
    short                  mode,     // graphics transfer mode
                                     // for the operation
    RgnHandle              mask,     // mask applied during
                                     // decompression
    CodecFlags             flags,    // intermediate buffer
                                     // allocation flags
    CodecQ                 accuracy  // desired accuracy
                                     // for the operation
    DecompressorComponent  codec);   // decompressor to use -
                                     // - can be special identifier
```

Figure 3. DecompressionSequenceBeginS API.

```
// Signal the beginning of the process of decompressing a
// sequence of frames Using codecHighQuality for the CodecQ
// parameter tells the decompressor to render at the highest
// image quality that can be achieved with reasonable
// performance. Using a lower CodecQ setting may be useful
// is speed is the priority.
OSErr MakeDecompressionSequence(
    ImageDescriptionHandle inImageDescription,
    GWorldPtr inDestGWorld, ImageSequence *outSeqID)
{
    Rect         theSrcBounds = {0, 0};
    Rect         theDestBounds;
    MatrixRecord rMatrix;

    *outSeqID = 0;

    if (NULL == inImageDescription) return paramErr;

    // *** IMPORTANT NOTE DV SOURCE ONLY ***
    // If your code passes in a smaller (eg. quarter-size) source
    // rectangle with the intention of drawing cropped unscaled DV to a
    // smaller (eg. quarter-size) destination, a DV-specific bug in
    // QuickTime 5 will cause this decompression request to be
    // misinterpreted, scaling the frame to fit. The correct
    // interpretationof this request is to draw the top-left corner
    // of the DV frame cropped at normal size. This bug will be
    // fixed in QuickTime 6.  If your code was behaving as intended
    // because of this bug,  make sure to fix your code to use a
    // matrix in the call to DecompressSequenceBeginS, scaling the
    // frame to fit the offscreen gworld. This approach will work
    // in all versions of QuickTime.
    // ************************************

    // create a transformation matrix to scale from the source bounds
    // to the destination bounds. Using NULL for the source rectangle
    // in the call to DecompressSequenceBeginS indicates we want to
    // decompress the entire source image
    GetPortBounds(inDestGWorld, &theDestBounds);
    theSrcBounds.right  = (*inImageDescription)->width;
    theSrcBounds.bottom = (*inImageDescription)->height;

    RectMatrix(&rMatrix, &theSrcBounds, &theDestBounds);

    return DecompressSequenceBeginS(
                outSeqID,           // pointer to field to receive
                                    // unique ID for sequence
                inImageDescription, // handle to image description
                                    // structure
                NULL,               // pointer to compressed image
                                    // data (used
```

```
                                    // for preflight)
                0,                  // image data size
                inDestGWorld,       // port for the DESTINATION image
                NULL,               // grahics device handle, if
                                    // port is set, set
                                    // this to NULL
                NULL,               // source rectangle defining
                                    // the portion of the image
                                    // to decompress - NULL for
                                    // the entire source image
                &rMatrix,           // transformation matrix
                srcCopy,            // transfer mode specifier
                (RgnHandle)NULL,    // clipping region in dest.
                                    // coordinate system to use as
                                    // a mask
                0,                  // flags
                codecHighQuality,   // accuracy in decompression
                anyCodec);          // compressor identifier or
                                    // special identifiers
                                    // ie. bestSpeedCodec
}
```

Listing 3. Signal the beginning of the process of decompressing a sequence of frames.

Use the `GWorldPtr` returned from `QTNewGWorld` as the destination port parameter. Using `codecHighQuality` for the `CodecQ` parameter tells the the decompressor to render at the highest image quality that can be achieved with reasonable performance. Using a lower `CodecQ` setting may be useful is speed is the priority.

Decompressing a Frame:

Once the sequence has started, each frame in the sequence is queued up for decompression by calling `DecompressSequenceFrameS` or `DecompressSequenceFrameWhen`. See Figure 4. The unique `ImageSequence` ID returned from `DecompressSequenceBeginS` is passed in as the first parameter.

```
OSErr DecompressSequenceFrameWhen(
    ImageSequence              seqID,              // unique segID
    Ptr                        data,               // pointer to
                                                   // compressed data
    long                       dataSize,           // size of the data
                                                   // buffer
    CodecFlags                 inFlags,            // control flags
    CodecFlags                 *outFlags,          // status flags
    ICMCompletionProcRecordPtr asyncCompletionProc, // async completion
                                                   // proc record
    const ICMFrameTimeRecord   *frameTime);        // frame time
                                                   // information
```

Figure 4. `DecompressSequenceFrameWhen` API.

The Image Compression Manager manages the decompression operation, calls the appropriate codec component to do the work and the frame is decompressed to the location specified in the `DecompressSequenceBeginS` call.

Decompress a frame. See Listing 4.

```
// A simple wrapper around DecompressSequenceFrameWhen
// Ignores in and out flags, no completion proc or
// frameTime specified - decompression operation will
// happen immediately
OSErr DecompressFrameNow(ImageSequence inSequenceID,
                         Ptr inBuffer, long inBufferSize)
{
    return DecompressSequenceFrameWhen(inSequenceID,
            inBuffer, inBufferSize, 0, NULL, NULL, NULL);
}
```

Listing 4a. Simple wrapper around `DecompressSequenceFrameWhen`.

```
// You could use a structure like this to keep track of per
// frame information
typedef struct {
    ImageSequence              seqID;      // decompression
                                           // sequence ID
    Ptr                        pSrcBuffer; // pointer to
                                           // compressed data
    long                       bufSize;    // compressed image
                                           // data size
    ICMCompletionProcRecordPtr
                    pCompletionProc;       // pointer to a
                                           // ICMCompletionProcRecord
    Boolean                    isDestDone; // is the ICM done
                                           // with the dest?
    OSErr                      rc;         // return code
} FrameRecord, *FrameRecordPtr, **FrameRecordHdl;

// Another way to wrap DecompressSequenceFrameWhen
// FrameRecordPtr assumes some type of data structure
// associated with your application which keeps track
// of per frame information
OSErr DecompressFrame(FrameRecordPtr inFrame)
{
    if (inFrame->pCompletionProc) {
        // if we set up a completion proc, store the frame
        // so it can be pulled out when we get called
        inFrame->pCompletionProc->completionRefCon = (long)inFrame;
    }

    return DecompressSequenceFrameWhen(inFrame->seqID,
                inFrame->pSrcBuffer, inFrame->bufSize,
                0, NULL, inFrame->pCompletionProc, NULL);
}
```

Listing 4b. Another wrapper around `DecompressSequenceFrameWhen`.

Asynchronous Decompression:

Decompression can be performed asynchronously by specifying a completion function along with a RefCon in a `ICMCompletionProcRecord`. See Figure 5a. In this case, make sure not to read the decompressed image until the decompressor indicates that the operation is complete by calling your completion function with the `codecCompletionDest` flag set. See Listing 5.

The completion function may be called multiple times and must be interrupt safe. Passing in NULL for the `asyncCompletionProc` specifies synchronous decompression.

Additionally, `DecompressSequenceFrameWhen` allows you to schedule decompression by passing in a pointer to a `ICMFrameTimeRecord`. See Figure 5b. This structure contains parameters specifying the frame's time information, including the time at which the frame should be decompressed, its duration, and the playback rate. When implementing scheduled decompression make sure to also implement the `ICMCompletionProc` discussed above. This parameter can be NULL, in which case the decompression operation will happen immediately.

```
// Specifies an image compression completion callback
struct ICMCompletionProcRecord {
    ICMCompletionUPP completionProc;   // UPP accessing your
                                       // ICMCompletionProc
    long             completionRefCon; // refcon for use by
                                       // the callback
};
typedef struct ICMCompletionProcRecord
                    ICMCompletionProcRecord;
typedef ICMCompletionProcRecord *
                    ICMCompletionProcRecordPtr;

// Called by a compressor component upon completion
// of an asynchronous operation
typedef void (*ICMCompletionProcPtr) (OSErr result,
                                short flags, long refcon);
void MyICMCompletionProc(
    OSErr result   // result of current operation
    short flags    // flags indicating which part
                   // of the operation is complete
    long  refcon); // refcon specified in the
                   // ICMCompletionProcRecord
```

Figure 5a. `ICMCompletionProcRecord` and `ICMCompletionProc`

```
// Contains a frame's time information for scheduled
// asynchronous decompression operations.
struct ICMFrameTimeRecord {
    wide     value;            // time the frame is to be displayed
    long     scale;            // units for the frame's display time
    void *   base;             // the time base
    long     duration;         // duration the frame is displayed
                               // must be in the same units as
                               // specified by the scale field.
                               // 0 if duration is unknown
    Fixed    rate;             // the time base's effective rate
    long     recordSize;       // size of this structure
    long     frameNumber;      // 0 if the frame number is not known
    long     flags;            // flags
    wide     virtualStartTime; // conceptual start time
    long     virtualDuration;  // conceptual duration
};
```

Figure 5b. ICMFrameTimeRecord

```
// Sample ICM decompression completion procedure
// This procedure simply checks the status of the codecCompletionDest
// flag which indicates that the Image Compression Manager is done
// with the destination buffer FrameRecordPtr assumes some type of
// data structure associated with your application which keeps
// track of per frame information
// Note: This function may be called multiple times and must be
// interrupt safe
static pascal void DecompressionDone(OSErr inResult,
                                short inFlags, long inRefCon)
{
    FrameRecordPtr theFrame = (FrameRecordPtr)inRefCon;

    if (noErr == inResult) {
        if (codecCompletionDest & inFlags) {

            // the ICM is done with the destination
            theFrame->isDestDone = true;

            ...
        }
    }

    theFrame->rc = inResult;
}
```

Listing 5. Sample ICM Decompression Completion Procedure

Accessing the Pixels:

To access the pixels directly, use GetGWorldPixMap to obtain the PixMapHandle from your destination GWorld, then call GetPixBaseAddr which returns a pointer to the beginning of a pixel image.

GetPixRowBytes or QTGetPixMapHandleRowBytes can be used to retrieve the rowBytes value (the distance, in bytes, from the beginning of one row of the image data to the beginning of the next row of the image data) for a pixel map accessed by a handle, while QTGetPixMapPtrRowBytes will do the same for a pixel map accessed by a pointer. See Figure 6.

Remember to always call LockPixels to prevent the base address for an offscreen pixel image from being moved while you draw into or copy from its pixel map.

```
// You could use a structure like this for convenience
// to cast the 32bit pixel map as an array of pixels.
typedef struct {
      UInt8 alpha;  // alpha component
      UInt8 red;    // red component
      UInt8 green;  // green component
      UInt8 blue;   // blue component
} ARGBPixelRecord, *ARGBPixelPtr, **ARGBPixelHdl;

PixMapHandle hPixMap     = GetGWorldPixMap(myDestGWorld);
long         theRowBytes = QTGetPixMapHandleRowBytes(hPixMap);
Ptr          pPixels     = GetPixBaseAddr(hPixMap);
```

Figure 6. Accessing Pixels directly

Sequence setup function. See Listing 6.

Ending the Sequence:

After the entire sequence is decompressed, end the process by calling the CDSequenceEnd. Pass in the unique image sequence ID. See Figure 7.

Sequence end function. See Listing 7.

```
OSErr CDSequenceEnd(ImageSequence seqID);
```

Figure 7. CDSequenceEnd API

```
// Sample function demonstrating how one might go about setting up a
// decompression sequence
// Long winded assignments are used for the purpose of this sample
// MyAppObjectPtr assumes some type of data structure associated with
// your application which keeps track of all the important bits
OSErr SetupDecompressionSequenceForDV(MyAppObjectPtr inAppObject)
{
    ImageDescriptionHandle hImageDescription = NULL;
    GWorldPtr              theGWorld = NULL;
    ImageSequence          theSequenceID = 0;
    PixMapHandle           hPixMap = NULL;
    long                   theRowBytes = 0;
    Ptr                    pPixels = NULL;

    OSErr err = noErr;

    hImageDescription = MakeImageDescriptionForDV();
    err = MemError();

    if (hImageDescription) {
        // the GWorld does not have to be the exact same size as
        // the compressed image
        err = MakeGWorld(&theGWorld,
                (*desc)->width, (*desc)->height);
        if (err) goto bail;

        err = MakeDecompressionSequence(hImageDescription,
                                theGWorld, &theSequenceID);
        if (err) goto bail;

        // get the BaseAddress and RowBytes for the pixels
        hPixMap     = GetGWorldPixMap(theGWorld);
        pPixels     = GetPixBaseAddr(hPixMap);
        theRowBytes = QTGetPixMapHandleRowBytes(hPixMap);

        // this is what we need
        inAppObject->dSeqID   = theSequenceID;
        inAppObject->pGWorld  = theGWorld;
        inAppObject->pPixels  = pPixels;
        inAppObject->rowBytes = theRowBytes;
    }

bail:
    if (hImageDescription)
        DisposeHandle((Handle)hImageDescription);

    return err;
}
```

Listing 6. Sample function demonstrating setting up a decompression sequence.

```
// Sample function demonstrating how one might go about ending
// a decompression sequence MyAppObjectPtr assumes some type of
// data structure associated with your application which keeps
// track of all the important bits
OSErr EndDecompressionSequenceForDV(MyAppObjectPtr inAppObject)
{
    OSErr err = noErr;

    // end the decompression sequence
    if (0 != inAppObject->dSeqID) {
        err = CDSequenceEnd(inAppObject->dSeqID);
        inAppObject->dSeqID = 0;
    }

    // dispose of the GWorld
    if (NULL != inAppObject->pGWorld)
        DisposeGWorld(inAppObject->pGWorld);

    inAppObject->pGWorld  = NULL;
    inAppObject->pPixels  = NULL;
    inAppObject->rowBytes = 0;

    return err;
}
```

Listing 7. Sample function demonstrating ending a decompression sequence.

Back to top

References:

QTNewGWorld

ImageDescription Structure

ImageDescription

Working with Image Descriptions

DecompressSequenceBeginS

DecompressSequenceFrameWhen

ICMCompletionProcRecord

ICMCompletionProc

ICMFrameTimeRecord

Working with Sequences

Image Compression Manager

Decompressing Sequences

Changing Sequence-Decompression Parameters

Back to top

Downloadables

Acrobat version of this Note (76K). Download