Technical Note TN2060

ICM Drawing non-scheduled frames with QuickTime 6

CONTENTS

This technote discusses changes to how the Image Compression Manager draws non-scheduled frames in QuickTime 6 on Mac OS X.

[Aug 21 2002]

Introduction

Windows on Mac OS X are double-buffered. In other words, every window has an offscreen buffer associated with it, and applications draw into the offscreen buffer (using QuickDraw, Quartz, QuickTime, or OpenGL). When an application is done drawing into the offscreen buffer, the image is copied to the frame buffer by the Quartz Compositor. Sending the image to the Quartz Compositor is called *flushing* .

Usually, Mac OS X handles window flushing for you, and your application does not need to be aware of it. However, there are some changes in the way QuickTime 6 handles this which may make application drawing behave differently.

Back to top

Some terminology

For the purposes of this note:

- *flushing* means sending part of a window's offscreen buffer to the Quartz Compositor for display on the screen.
- a *run loop* is a Core Foundation service that monitors sources of input to a task (such as events), calls callbacks to handle them, and sleeps in between. Carbon and Cocoa event loops are implemented using run loops.
- a *decompression* is any drawing operation mediated by the Image Compression Manager. This includes frames of video, images drawn with Graphics Importer components, and compressed pictures.
- a *scheduled decompression* is a decompression set up to occur at a specific time in the future.
- a *non-scheduled decompression* is a decompression that happens immediately.

Graphics Importers and compressed pictures always use non-scheduled decompression. When playing a movie, QuickTime tries to use scheduled decompression for drawing all video frames, because this allows for more accurate frame timing. However, certain things can cause QuickTime to fall back to non-scheduled decompression. These include video codecs that don't support scheduling and cases when there is not enough CPU power available to play the movie.

Back to top

What happens when my application draws using QuickDraw?

QuickDraw maintains a *dirty region* associated with each `GrafPort`. When an application draws into a `GrafPort`, the area drawn to is added to the ports dirty region. When your application returns to its run loop, Carbon walks the list of windows and flushes their dirty regions by calling `QDFlushPortBuffer`. This means that even if drawing operations took a while, everything appears on the screen in one crisp refresh.

Back to top

What did QuickTime 5 do?

QuickTime 5 did the same thing for both scheduled and non-scheduled decompressions: as soon as the drawing was complete, QuickTime would flush the image to the screen.

This is the correct thing to do for playing movies: every video frame should be displayed as soon as possible. However, it meant that applications that mixed still images drawn using Graphics Importers (for example) with other QuickDraw drawing would look bad, because the still images would be displayed on the screen before everything else -- not in one crisp refresh.

Back to top

What changed in QuickTime 6?

After scheduled decompressions, QuickTime 6 flushes images to the screen.

After non-scheduled decompressions, QuickTime 6 does not flush. Instead, it adds the drawn area to QuickDraw's dirty region for that `GrafPort`, expecting the run loop to flush the dirty region along with any other drawing. So applications that mix Graphics Importers and QuickDraw will get crisper screen updates than before.

Back to top

So, what's the problem?

While this change to the Image Compression Manager brings non-scheduled drawing into line with QuickDraw drawing, it means you may not actually see the results of a drawing operation if your application:

1. uses an on-screen port that is not a Carbon window, or
2. doesn't let the run loop execute regularly (e.g., sitting in a tight loop with no calls to `WaitNextEvent` or not returning control from a Carbon event handler).

In case you're curious, one example of an onscreen port that is not a Carbon window is the application dock tile port, which you can create by calling `BeginQDContextForApplicationDockTile`.

Back to top

What should I do if my application encounters this problem?

To avoid these drawing problems, applications are advised to implement one or more of the following:

1. let the run loop run, if drawing is in a Carbon window.
2. call `QDFlushPortBuffer` after any non-scheduled decompression, to flush explicitly.
3. set the Image Compression Manager sequence `codecDSequenceFlushInsteadOfDirtying` flag for your decompression sequence causing the Image Compression Manager to implicitly flush.This can be done using the `SetDSequenceFlags` API.
4. set the movie play hint `hintsFlushVideoInsteadOfDirtying` for your movie. This can be done using the `SetMoviePlayHints` API.

Let's go into each of these options in more detail.

1. Let the run loop run.

All sorts of things stop happening when you don't let the run loop run. Carbon event loop timers don't fire; your application doesn't receive events (so the dock will report that your application is not responding); and dirty regions in windows don't get flushed. If possible, look into restructuring your code so that your event handler returns. In some cases, using Carbon event loop timers can simplify your code *and* make it less CPU-greedy.

This won't help if your application is drawing into an on-screen port that is not a Carbon window, however, because it won't be found when Carbon walks the window list.

2. Call QDFlushPortBuffer.

This might be appropriate if you have a short animation drawn using QuickTime and it's not convenient for your event handler to return until it's done.

```
void QDFlushPortBuffer( CGrafPtr port, RgnHandle region );
```

- **port** - A pointer to a CGrafPort record.
- **region** - A handle to an explicit region to flush, or pass NULL to flush the current dirty region.

Listing 1. Use a Graphics Importer to draw and rotate an image.

```
WindowRef gWindow;

OSErr DoSillyRotation( const FSSpecPtr inFSSpec )
{
    GraphicsImportComponent importer = 0;
    Rect                    naturalBounds,
                            windowBounds;
    MatrixRecord            matrix;
    UInt32                  rotation = 0;
    OSErr                   err = noErr;

    err = GetGraphicsImporterForFile( inFSSpec, &importer );
    if ( err ) goto bail;

    // get the native size of the image
    err = GraphicsImportGetNaturalBounds( importer, &naturalBounds );
    if ( err ) goto bail;

    windowBounds = naturalBounds;
    OffsetRect( &windowBounds, 10, 45 );
    gWindow = NewCWindow( NULL, &windowBounds,
                          "\pSilly GImport Rotate", true, documentProc,
                          (WindowPtr)-1, true, 0);
    if ( NULL == gWindow ) goto bail;

    // set the graphics port for drawing
    err = GraphicsImportSetGWorld( importer,
                        GetWindowPort( gWindow ), NULL );
    if ( err ) goto bail;

    // do silly rotation
    do {

        // reset the matrix
        SetIdentityMatrix( &matrix );

        // modify the contents of a matrix so that it defines a
        // rotation operation - 'rotation' degrees to the right
        RotateMatrix( &matrix,
```

```
                Long2Fix( rotation ),
                Long2Fix(( naturalBounds.right - naturalBounds.left ) / 2 ),
                Long2Fix(( naturalBounds.bottom - naturalBounds.top ) / 2 ));

        // set the transformation matrix
        GraphicsImportSetMatrix( importer, &matrix );

        // draw
        err = GraphicsImportDraw( importer );
        if ( err ) break;

        // explicitly flush - required under QuickTime 6
        QDFlushPortBuffer( GetWindowPort( gWindow ), NULL );

        rotation += 30;

    } while ( rotation <= 360 );

bail:
    if ( importer ) CloseComponent( importer );

    return err;
}
```

3. Tell the Image Compression Manager to flush.

If you're using Image Compression Manager decompression sequences directly, you can set a flag to tell it to flush as it did in QuickTime 5.

```
enum {
    codecDSequenceFlushInsteadOfDirtying = (1L << 8)
};
```

Listing 2. Use a Decompression Sequence to draw and rotate an image.

```
WindowRef gWindow;

OSErr DoDSeqSillyRotation( const FSSpecPtr inFSSpec )
{
    GraphicsImportComponent importer = 0;
    Rect                    naturalBounds,
                            windowBounds;
    ImageSequence           seqID = 0;
    ImageDescriptionHandle  desc = NULL;
    Ptr                     pData = NULL;
    MatrixRecord            matrix;
    UInt32                  rotation = 0;
    GrafPtr                 savedPort;
    OSErr                   err = noErr;

    GetPort(&savedPort);

    err = GetGraphicsImporterForFile( inFSSpec, &importer );
    if ( err ) goto bail;

    // get the native size of the image associated with the importer
    err = GraphicsImportGetNaturalBounds( importer, &naturalBounds );
    if ( err ) goto bail;
```

```
    windowBounds = naturalBounds;
    OffsetRect( &windowBounds, 10, 45 );
    gWindow = NewCWindow( NULL, &windowBounds,
                          "\pSilly DSequence Rotate", true, documentProc,
                          (WindowPtr)-1, true, 0);
    if ( NULL == gWindow ) goto bail;

    SetPortWindowPort( gWindow );

    // get the image description
    err = GraphicsImportGetImageDescription( importer, &desc );
    if ( err ) goto bail;

    // we need to stick the image data somewhere
    pData = NewPtrClear( (**desc).dataSize );
    if ( MemError() || NULL == pData ) goto bail;

    // get the image data
    err = GraphicsImportReadData( importer, pData, 0, (**desc).dataSize );
    if ( err ) goto bail;

    // *** use a decompression sequence to draw ***

    // begin the sequence
    err = DecompressSequenceBeginS( &seqID, desc, NULL, 0,
                                    GetWindowPort( gWindow ), NULL, NULL,
                                    NULL, srcCopy, NULL, 0, codecNormalQuality,
                                    anyCodec );
    if ( err ) goto bail;

    // under QuickTime 6 setting the codecDSequenceFlushInsteadOfDirtying
    // flag for a decompression sequence causes the Image Compression Manager
    // to implicitly flush - setting this flag has no effect under QuickTime 5
    SetDSequenceFlags( seqID,
                       codecDSequenceFlushInsteadOfDirtying,
                       codecDSequenceFlushInsteadOfDirtying );

    // do a silly rotation
    do {

        // reset the matrix
        SetIdentityMatrix( &matrix );

        // modify the contents of a matrix so that it defines a rotation operation
        // 'rotation' degrees to the right
        RotateMatrix( &matrix,
                      Long2Fix( rotation ),
                      Long2Fix(( naturalBounds.right - naturalBounds.left ) / 2 ),
                      Long2Fix(( naturalBounds.bottom - naturalBounds.top ) / 2 ));

        // set the transformation matrix to use for drawing an image
        SetDSequenceMatrix( seqID, &matrix );

        // draw the image
        err = DecompressSequenceFrameS( seqID, pData, (**desc).dataSize, 0,
                                        NULL, NULL );
        if ( err ) goto bail;

        rotation += 30;

    } while ( rotation <= 360 );
```

```
bail:
    if ( seqID ) CDSequenceEnd( seqID );
    if ( importer ) CloseComponent( importer );
    if ( desc ) DisposeHandle( (Handle)desc );
    if ( pData ) DisposePtr( pData );

    SetPort( savedPort );

    return err;
}
```

Note:
Setting the `codecDSequenceFlushInsteadOfDirtying` flag has no effect under QuickTime 5.

4. Tell the Movie Toolbox to flush.

If you're playing a movie, you can set a play hint to tell the movie to flush. For movies that play using scheduled decompression (or using hardware acceleration) this may not appear to be necessary. Animated GIFs use neither, so if you change the movie to an animated GIF and you don't see frames, you may need to set this play hint.

```
enum {
    hintsFlushVideoInsteadOfDirtying = (1L << 22)
};
```

Listing 3. Play an animated GIF file as a movie.

```
WindowRef gWindow;
Movie     gMovie;

OSErr PlayAnimatedGIF( const FSSpecPtr inGifFile )
{
    short     refNum = 0;
    Rect      bounds;
    long      numberOfSamples;
    GrafPtr   savedPort;
    OSErr     err;

    GetPort( &savedPort );

    err = OpenMovieFile( inGifFile, &refNum, fsRdPerm );
    if (err) goto bail;

    // create a movie from the GIF
    err = NewMovieFromFile( &gMovie, refNum, NULL, NULL, newMovieActive, NULL );
    CloseMovieFile( refNum );
    if ( err || NULL == gMovie ) goto bail;

    GetMovieNaturalBoundsRect( gMovie, &bounds );
    OffsetRect( &bounds, -bounds.left, -bounds.top );
    OffsetRect( &bounds, 10, 45 );
    gWindow = NewCWindow( NULL, &bounds,
                          "\pPlay Animated GIF", true, documentProc,
                          (WindowPtr)-1, true, 0 );

    SetPortWindowPort( gWindow );

    // set the movies GWorld
    SetMovieGWorld( gMovie, GetWindowPort( gWindow ), NULL );
```

```
    // get the sample count - this will let us know if it's animated or not
    numberOfSamples =
          GetMediaSampleCount( GetTrackMedia( GetMovieIndTrack( gMovie, 1 ) ));

    // it's animated - under QuickTime 6 set the
    // hintsFlushVideoInsteadOfDirtying play hint
    if ( numberOfSamples > 1 ) {
        SetMoviePlayHints( gMovie,
                           hintsFlushVideoInsteadOfDirtying,
                           hintsFlushVideoInsteadOfDirtying );
   }

    StartMovie( gMovie );

    do {
        MoviesTask( gMovie, 0 );
    } while ( !IsMovieDone( gMovie ) );

bail:
    SetPort( savedPort );

    return err;
}
```

Play hints are passed down to media handlers; media handlers should obey this play hint by making sure whatever they draw gets flushed -- possibly by calling `QDFlushPortBuffer` or by setting the `codecDSequenceFlushInsteadOfDirtying` flag discussed above on any decompression sequences they use.

> **Note:**
> Setting the `hintsFlushVideoInsteadOfDirtying` play hint has no effect under QuickTime 5.

Back to top

References

QDFlushPortBuffer

SetDSequenceFlags

SetMoviePlayHints

Working with Sequences

Back to top

Downloadables

Acrobat version of this Note (44K) Download

Back to top
