Changes to Image Compression Manager
8/18/92

CompactVideo Compressor

The CompactVideo Compressor is a newly developed compressor that is best suited to compressing 16-bit and 24-bit video sequences. It is a lossy algorithm that is highly asymmetrical. In other words, it takes significantly longer to compress a frame than it does to decompress that frame. Compressing a 24-bit 640x480 image on a Mac IIfx takes approximately 4 minutes, achieving a compression ratio of 13.5:1. Decompressing the image takes less than a second.

Compared to the original Video compressor, CompactVideo obtains higher compression ratios, better image quality, and faster playback speeds. When used in conjunction with MovieShop, CompactVideo can constrain data rates to user definable levels. This is particularly important when compressing material for playback from CD.

For best quality results, CompactVideo should be used on raw source data that has never been compressed with a highly lossy compressor before (including CompactVideo).

Known Problems in CompactVideo:

• Low memory situations are not handled gracefully. All memory is currently allocated in the application heap instead of the heap with the most available memory. Failure of memory allocation is not always detected. Banding and spooling of large images is not handled.

• Bad frames are occasionally generated when a data rate significantly less than what the compressor can achieve is requested.

Alignment Calls

New window alignment calls have been added to provide a mechanism for positioning and dragging windows to optimal screen positions based on the depth of the screen. Rectangles on 1-bit and 2-bit screens are horizontally aligned to multiples of 16 pixels, 4-bit to multiples of 8, 8-bit to multiples of 4, and 16-bit to multiples of 2. Alignment on 32-bit screens only occurs on 68040 class machines or greater and then it is to multiples of 4 pixels. When the alignment rectangle crosses more than one screen, the alignment of the strictest screen is used. Decompression to non-optimally aligned destinations can reduce performance by as much as 50% so these calls should be used whenever possible.

The default alignment behavior will be adequate for the vast majority of cases where alignment is desired. However, when alignment characteristics specific to piece of video hardware or for some other reason is needed, AligmentProcs give the caller a way of overriding the default alignment. The alignment proc is called with a rectangle in global screen coordinates that has already been aligned using the default behavior. The alignment proc then has the option of applying some addition alignment criteria to the rectangle, such as vertical alignment of some

form. In the case of supporting hardware alignment, it is the proc's responsibility to determine if the rectangle applies to specific device it cares about. See the Sequence Grabber for more information on alignment procs and video hardware.

```
typedef pascal void (*AlignmentProcPtr)(Rect *rp, long refcon);

typedef struct {
        AlignmentProcPtr      alignmentProc;
        long                  alignmentRefCon;
} AlignmentProcRecord, *AlignmentProcRecordPtr;
```

```
pascal void
AlignWindow(WindowPtr wp, Boolean front, const Rect *alignmentRect,
AlignmentProcRecordPtr alignmentProc);
```

AlignWindow is similar to the Window Manager call MoveWindow but is used to move a window the nearest optimal alignment position. The WindowPtr wp and Boolean front are identical to those of the MoveWindow call. The alignmentRect parameter is a rectangle in window coordinates that allows you to align the window to a rectangle within the window. Pass nil to align to the bounds of the window. The alignmentProc parameter allows you to provide your own alignment behavior. Pass nil to use the default behavior.

```
pascal void
DragAlignedWindow(WindowPtr wp, Point startPt, Rect *boundsRect, Rect *alignmentRect,
AlignmentProcRecordPtr alignmentProc);
```

DragAlignedWindow implements the Window Manager call DragWindow but adds the ability to drag the window along an optimal alignment grid. The WindowPtr, startPt, and boundsRect parameters are identical to those of DragWindow. The alignmentRect parameter is a rectangle in window coordinates that allows you to align the window to a rectangle within the window. Pass nil to align to the bounds of the window. The alignmentProc parameter allows you to provide your own alignment behavior. Pass nil to use the default behavior.

```
pascal long
DragAlignedGrayRgn(RgnHandle theRgn, Point startPt, Rect *boundsRect, Rect *slopRect,
short axis, ProcPtr actionProc, Rect *alignmentRect, AlignmentProcRecordPtr alignmentProc);
```

DragAlignedGrayRgn is nearly identical to the Window Manager call DragGrayRgn. The alignmentRect allows alignment to a rectangle within the bounds of theRgn. Pass nil to align to the bounds of theRgn. The alignmentProc parameter allows you to provide your own alignment behavior. Pass nil to use the default behavior. This call will not usually need to be made directly.

```
pascal void
AlignScreenRect(Rect *rp, AlignmentProcRecordPtr alignmentProc);
```

Given a rectangle in global screen coordinates, align the rectangle to the strictest screen the rectangle intersects. The alignmentProc parameter allows you to provide your own alignment behavior. Pass nil to use the default behavior. This call will not usually need to be made

directly.

Image Description Extension Calls

The Set and GetImageDesciptionExtension calls of QuickTime 1.0 have been augmented with calls that supply additional information about the extensions and the ability to remove them once added. These calls parallel the UserData calls in the MovieToolbox.

pascal OSErr
RemoveImageDescriptionExtension(ImageDescription **desc, long type, long index);

Remove the extension specified by type and ones based index from the description handle. Passing type == 0 means match any extension, with the index parameter becoming an index into all of the extensions.

pascal OSErr
CountImageDescriptionExtensionType(ImageDescription **desc, long type, long *count);

Count the number of extensions in the given description handle of a specified type. Passing type == 0 means match any extension, and return a count of all of the extensions.

pascal OSErr
GetNextImageDescriptionExtensionType(ImageDescription **desc, long *type);

In an effort to create the QuickTime call with the longest name, GetNext'Type has been added to return the next extension type found after the one passed in the type parameter. Passing 0 for type means return the first type found. If zero is returned, no more types could be found.

Sequence Calls

pascal OSErr
SetCSequenceFrameNumber(ImageSequence seqID, long frameNumber);

SetCSequenceFrameNumber is used to inform the compressor being used for the sequence that frames are being compressed out of order. This information is only necessary for compressors that are sequence sensitive.

pascal OSErr
GetCSequenceFrameNumber(ImageSequence seqID, long *frameNumber);

GetCSequenceFrameNumber returns the current frame number of the given sequence.

pascal OSErr
GetCSequenceKeyFrameRate(ImageSequence seqID, long *keyframerate);

This call provides a way of determining the current key frame rate of a sequence. It balances the SetCSequenceKeyFrameRate call available in QuickTime 1.0.

Miscellaneous Calls

pascal OSErr
GetBestDeviceRect(GDHandle *gdh, Rect *rp);

GetBestDeviceRect selects the deepest from all available GDevices, while treating 16 and 32-bit screens as being equal depth. If there are multiple 16 and 32-bit monitors available, it selects the 16 or 32-bit device that the mouse is currently on. If the mouse is not on one of the devices in question, the first of those in the list is chosen.

The rectangle returned is from the gdRect of the chosen device. The rectangle is adjusted for the height of the menu bar if the device is the main device. Note that this call does not center a rectangle on a device, it only returns the rectangle for the best device. If you not need either the GDHandle or Rect returned, pass nil for either of those parameters.

pascal QDErr
NewImageGWorld(GWorldPtr *gworld, ImageDescription **idh, GWorldFlags flags);

NewImageGWorld is a convenience call that creates a GWorld based on the width, height, depth and color table of the given image description. It will select the appropriate color table using the depth field or custom color table in the image description, and it will create a 32-bit deep gworld if the image description depth is 24. The flags are passed directly through to NewGWorld. The caller is responsible for disposing the GWorld using DisposeGWorld.

Changes to Existing Calls

DrawTrimmedPicture/File
FCompressPicture/File

The meaning of the doDither parameter these four calls take has been expanded to:

```
#define defaultDither       0
#define forceDither         1
#define suppressDither      2
```

defaultDither means respect the dithering in the source picture/file, forceDither means dither whether the source uses dithering or not, and suppressDither means don't use dithering in any case. When used with QuickTime 1.0, suppressDither would be interpreted as forceDither.

The ability to suppress dithering can be useful if, for example, you have a 32-bit color JPEG image being drawn into a 8-bit buffer with a custom color table from the image. In that case, dither would not be necessary and probably not desirable, particularly if the buffer was then going to be compressed with the Graphics compressor.