Image Transcoding

4/2/96 - Draft 2

Note

This is an early draft of this document and of this interface. Any and all comments are appreciated. Please send comments, clarifications, and questions to Peter Hoddie (email: hoddie.jp@applelink.apple.com).

Introduction

QuickTime 2.2 provides support for a new kind of image manipulation: image transcoding. Until now, if a user needed to convert from compressed image data into another compressed image format, it was necessary to decompress the compressed image data to RGB pixels, and then compress the RGB pixels into the new format. For certain types of compressed image data, it is possible to convert directly from one compressed format to another. This direct conversion process is called image transcoding. Transcoding has two advantages over the traditional decompress/compress approach to converting the format of compressed data. First, the operation is usually substantially faster as much of the data can be copied directly from the source image data format to the destination image data format. Secondly, the operation is usually more accurate as decompressing and recompressing provides two steps for introducing round off and other errors. By directly transcoding, the opportunities for small errors to be introduced or accumulate are substantially reduced.

Overview

QuickTime's image transcoding support is contained within the Image Compression Manager. Image transcoding can be invoked either explicitly, using new API's in the Image Compression Manager, or implicitly, by using existing routines for decompressing images.

QuickTime's support for decompressing images has been enhanced so that if a request is issued to decompress an image, but no image decompressor component is installed for that image format, QuickTime will attempt to locate an image transcoder to convert the image data into a supported format. This transcoding is performed completely transparently to the calling application. This automatic image transcoding is supported both for QuickTime movie's and for compressed image data stored in QuickDraw pictures.

QuickTime also provides an API for applications to transcode images. These API's make it possible for any application to take compressed image data and transcode it into another format. This capability is useful for applications which create QuickTime movies by combining pieces of other QuickTime movies. These applications often convert the format of the compressed image data by decompressing the image and then recompressing it to the new format. If no other processing is to be performed on the compressed data, an image transcoder can be used instead to increase the speed and fidelity of the operation.

As with most other services in QuickTime, the details of image transcoding are handled by components. Image transcoder components are new with QuickTime 2.2. Image

transcoder components are used by the Image Compression Manager to perform both implicit and explicit image transcoding as described above. Application developers that perform image transcoding do not interact directly with the image transcoder components, but rather interact with the Image Compression Manager. The Image Compression Manager takes care of the details of working with image transcoder components. If developers wish to add new image transcoding operations to QuickTime, they can write an image transcoder component.

Image Transcoding Routines

The Image Compression Manager uses the idea of an image sequence when compressing or decompressing data. An image sequence allows QuickTime to make certain optimizations because it knows that a similar operation will be repeated multiple times (i.e. images will be repeatedly compressed to the same image data format). Similarly, the Image Compression Manager's support for image transcoding is based on the idea of an image transcoding sequence. The image transcode sequence identifier is an opaque value as shown below.

```
typedef long ImageTranscodeSequence;
```

To create an image transcoding sequence, use the ImageTranscodeSequenceBegin routine.

```
pascal OSErr ImageTranscodeSequenceBegin(ImageTranscodeSequence
      *its, ImageDescriptionHandle srcDesc, OSType destType,
      ImageDescriptionHandle *dstDesc)
```

ImageTranscodeSequenceBegin returns the image transcode sequence identifier in the "its" parameter. If the operation fails, the value pointed to by the "its" parameter is set to nil. The "srcDesc" parameter is the image description for the source compressed image data. The "destType" parameter is the desired compression format to transcode the source data into. The "dstDesc" parameter is used to return an image description for the data which will be generated by the image transcoding sequence. The caller of the ImageTranscodeSequenceBegin routine is responsible for disposing of the image description which is returned in the "dstDesc" parameter. If no transcoder is available to perform the requested transcoding operation, an error of handlerNotFound is returned.

When an image transcoding sequence is complete, use ImageTranscodeSequenceEnd to dispose of the image transcoding sequence.

```
pascal OSErr ImageTranscodeSequenceEnd(ImageTranscodeSequence its)
```

The only parameter to ImageTranscodeSequenceEnd is the identifier of the image transcoder sequence to dispose. It is safe to pass a value of 0 to this routine.

To transcode a frame of image data, use ImageTranscodeFrame, after having created the image transcoder sequence using ImageTranscodeSequenceBegin.

```
pascal OSErr ImageTranscodeFrame(ImageTranscodeSequence its, void
      *srcData, long srcDataSize, void **dstData, long
      *dstDataSize)
```

The “its” parameter specifies the image transcoding sequence to use to perform the transcoding operation. The “srcData” field contains a pointer to the source data to transcode. The source data must be locked. The “srcDataSize” field indicates the size of the compressed source image data in bytes. A pointer to the transcoded image data is returned in the “dstData” parameter, and the size of the transcoded image data is returned in the “dstDataSize” parameter. The caller of this routine is responsible for disposing of the transcoded data using the ImageTranscodeDisposeFrameData routine.

When the transcoded image data returned by ImageTranscodeFrame is no longer needed, call ImageTranscodeDisposeFrameData to dispose of the data.

```
pascal OSErr ImageTranscodeDisposeFrameData(ImageTranscodeSequence
      its, void *dstData)
```

The “its” parameter” specifies the image transcoding sequence that was used to generate the transcoded data. The “dstData” parameters is a pointer to the transcoded image data generated by the ImageTranscodeFrame routine. The ImageTranscodeDisposeFrameData routine must be used to dispose of the transcoded data. Only the image transcoder that generated the data knows how to properly dispose it.

Image Transcoder Components

It is only necessary to understand image transcoder components if you are writing an image transcoder. To perform image transcoding, you should use the services provided by the Image Compression Manager.

Image transcoder components are standard Component Manager components. See Inside Macintosh: More Toolbox for details on creating components.

Image transcoder components have a type of ‘imtc’ as defined below.

```
enum {
      ImageTranscodererComponentType = 'imtc'
};
```

The sub-type field of the component defines the compressed image data format that the transcoder accepts as an input. The manufacturer field of the component defines the compressed image data format that the transcoder generates as output.

```
pascal ComponentResult
      ImageTranscoderBeginSequence(ImageTranscoderComponent ci,
      ImageDescriptionHandle srcDesc, ImageDescriptionHandle
      *dstDesc)
```

ImageTranscoderBeginSequence specifies the format of source compressed image data format in the “srcDesc” parameter. The image transcoder should allocate a new image description and returned in the “dstDesc” parameter. The new image description should be a completely filled out image description which is sufficient for correctly decompressing the data which will be generated by subsequent calls to ImageTranscoderConvert.

```
pascal ComponentResult
      ImageTranscoderConvert(ImageTranscoderComponent ci, void
      *srcData, long srcDataSize, void **dstData, long
      *dstDataSize)
```

ImageTranscoderConvert performs the actual image transcoding operation. A pointer to the source compressed image data is provided in the “srcData” parameter and the size in bytes of the source data is specified in the “srcDataSize” parameter. The image transcoder component is responsible for allocating storage for the transcoded data, transcoding the data, and returning a pointer to the transcoded data in the “dstData” parameter. The image transcoder’s ImageTranscoderDisposeData routine will be called to dispose of the transcoded data. The size of the transcoded data in bytes should be returned in the “dstDataSize” parameter.

The memory allocated to store the transcoded image data must not be in an unlocked handle. Even if the image transcoding operation can be done in place, the transcoded data must be placed in a separate block of memory from the source data. The image transcoder component must not write back into the source image data.

The responsibility for allocating the buffer for the transcoded data has been placed in the transcoder with the intent that some hardware manufacturers may find it useful to place the transcoded data directly into on-board memory on their video board. If the transcoding operation is being performed on a movie, the transcoded data pointer will be almost immediately passed on to a decompressor. If the decompressor is implemented in hardware, some performance may be increased because the transcoded data is already loaded onto the decompression hardware.

```
pascal ComponentResult
      ImageTranscoderDisposeData(ImageTranscoderComponent ci, void
      *dstData)
```

When the client of the image transcoder component is done with a piece of transcoded data, ImageTranscoderDisposeData is called with a pointer to the transcoded data. The image transcoder component should not make any assumptions about the maximum number of outstanding pieces of transcoded data, or the order in which the transcoding data will be disposed.

```
pascal ComponentResult
      ImageTranscoderEndSequence(ImageTranscoderComponent ci)
```

ImageTranscoderEndSequence is called when there are no more frames of data to be transcoded using the parameters specified in the previous call to ImageTranscoderBeginSequence. After ImageTranscoderEndSequence is called, the component will either be closed or receive another call to ImageTranscoderBeginSequence with a different image description (for example, the dimensions of the source image may be different).

Example Image Transcoder Component

The following code is an example of an image transcoder component. It converts from an imaginary compressed data format, ‘bgr ‘ to uncompressed rgb pixels. The transcoding

process is simply copying the source data to the destination and inverting each byte in the process. This example serves to show the format of how an image transcoder might work without getting into an tedious details of a particular image transcoding operation.

```
#include <ImageCompression.h>

pascal ComponentResult main(ComponentParameters *params, Handle storage
);

pascal ComponentResult TestTranscoderBeginSequence (Handle storage,
ImageDescriptionHandle srcDesc, ImageDescriptionHandle *dstDesc);
pascal ComponentResult TestTranscoderConvert (Handle storage, void
*srcData, long srcDataSize, void **dstData, long *dstDataSize);
pascal ComponentResult TestTranscoderDisposeData (Handle storage, void
*dstData);
pascal ComponentResult TestTranscoderEndSequence (Handle storage);

pascal ComponentResult main(ComponentParameters *params, Handle storage
)
{
      ComponentFunctionUPP proc = nil;
      ComponentResult err = noErr;

      switch (params->what) {
            case kComponentOpenSelect:
            case kComponentCloseSelect:
                  break;
            case kImageTranscoderBeginSequenceSelector:
                  proc = (ComponentFunctionUPP)
                        TestTranscoderBeginSequence;
                  break;
            case kImageTranscoderConvertSelector:
                  proc = (ComponentFunctionUPP)TestTranscoderConvert;
                  break;
            case kImageTranscoderDisposeDataSelector:
                  proc = (ComponentFunctionUPP)
                        TestTranscoderDisposeData;
                  break;
            case kImageTranscoderEndSequenceSelector:
                  proc = (ComponentFunctionUPP)
                        TestTranscoderEndSequence;
                  break;
            default:
                  err = badComponentSelector;
                  break;
      }

      if (proc)
            err = CallComponentFunctionWithStorage(storage,
                                    params, proc);

      return err;
}

pascal ComponentResult TestTranscoderBeginSequence (Handle storage,
ImageDescriptionHandle srcDesc, ImageDescriptionHandle *dstDesc)
{
      *dstDesc = srcDesc;
```

```
	HandToHand((Handle *)dstDesc);
	(***dstDesc).cType = 'raw ';

	return noErr;
}

pascal ComponentResult TestTranscoderConvert (Handle storage, void
*srcData, long srcDataSize, void **dstData, long *dstDataSize)
{
	Ptr p;
	OSErr err;

	if (!srcDataSize)
		return paramErr;

	p = NewPtr(srcDataSize);
	err = MemError();
	if (err) return err;

	{
	Ptr p1 = srcData, p2 = p;
	long counter = srcDataSize;
	while (counter--)
		*p2++ = ~*p1++;
	}

	*dstData = p;
	*dstDataSize = srcDataSize;

	return noErr;
}

pascal ComponentResult TestTranscoderDisposeData (Handle storage, void
*dstData)
{
	DisposePtr((Ptr)dstData);

	return noErr;
}

pascal ComponentResult TestTranscoderEndSequence (Handle storage)
{
	return noErr;
}
```