# Cinepak For Jaguar

## 1. Introduction

This documents describes ***Cinepak for Jaguar***, a combination of utilities and code that has been developed to enable creation of high-quality video material which can be played back from the Jaguar CD-ROM. Playback rates of 30 frames per second are possible even with full-screen (320x200), 16-bit per pixel images. In fact, even higher resolutions and/or frame rates are possible provided the overall data rate is reasonable.

The ***Cinepak For Jaguar*** package is based upon Radius' proprietary Cinepak video compression technology[1], which was specifically developed for this type of application; it consists of the following main elements:

1. Interface definition and linkable object code for the Cinepak decompressor.

2. Definition of a file format which interleaves audio and video in a manner suitable for playback on Jaguar, together with sample playback code which illustrates how to manage the periodic access to the CD-ROM and maintain synchronization between audio and video.

3. A utility to convert Cinepak-encoded QuickTime movies to the Jaguar Cinepak film format and perform necessary manipulations prior to recording on CD-ROM.

4. Three sample Jaguar films on CD-ROM.

The Cinepak decompressor and the interface to it are discussed in the **Cinepak Decompressor** section. The Jaguar film format is discussed in the **Jaguar Film Format** section. The details of the sample player code are described in the **Sample Playback Code** section. The use of the film conversion utility is discussed in the **Jaguar Cinepak Utility For Macintosh** section. The content of a sample Jaguar CD-ROM containing Cinepak films is briefly described in the **Sample Jaguar Films** section. The layout of film data on a CD-ROM is discussed in the **Using A Jaguar Cinepak Film With CD-ROM** section.

## 2. Cinepak Decompressor

Decoding of the Cinepak bitstream and writing the decompressed pixel data to the frame buffer are handled almost entirely by the GPU in the Jaguar system. The 68000 plays a minor role in parsing the bitstream and setting up pointers to various data structures.

The Cinepak decompressor code consists of two object modules, *codec.o* and *gpucode.og*, for the 68000 and the GPU, respectively. In addition, several flags must be defined, storage for auxiliary data must be reserved and the 68000 interrupt service routine must be used to coordinate bus activity between 68000 and GPU.

[1] Cinepak was originally developed by *SuperMac Technology*, which merged with *Radius, Inc.* in 1994.

In this section, we define the interface to the two code modules and briefly describe the operation of the flags. For an example of how these elements are incorporated in playing a Jaguar film, see the **Sample Playback Code** section.

## 2.1 68000 Module

The *codec.o* module consists of approximately 700 bytes of 68000 code. There are three user callable functions, *CheckKeyFrame*, *PreDecompress*, and *Decompress*. The interfaces to these routines is specified below.

All the routines preserve all 68000 registers.

All parameters used by these routines are passed on the stack. The return value is also returned on the stack. Cleaning up the stack upon return from any of these three routines is the responsibility of the calling program.

### 2.1.1 CheckKeyFrame()

This routine is called to determine whether or not the current frame is a key frame.[2]

| Stack Offset | Size | Description |
|---|---|---|
| 4(a7) | 4 | Return value. Must be set to 0 prior to entry. Will be set to 1 upon exit if key frame is detected. |
| (a7) | 4 | Address of start of frame. |

**Table 2.1 — 68000 stack setup before call to *CheckKeyFrame*.**

### 2.1.2 PreDecompress()

This routine is called to set up the tables needed to draw pixels on the display.

| Stack Offset | Size | Description |
|---|---|---|
| 10(a7) | 4 | Return value. Value prior to entry is not important.<br>0 = returned upon successful completion<br>non-zero = Error occurred. |
| 6(a7) | 4 | Address of $3000 byte auxiliary Cinepak data buffer (see section 2.4) |
| 2(a7) | 4 | Address of start of frame in Cinepak bitstream. |
| (a7) | 2 | Flag which indicates video data type:<br>0 = Cinepak compressed-RGB format<br>1 = Atari CRY format or expanded RGB |

**Table 2.2 — 68000 stack setup before call to *PreDecompress*.**

[2] Cinepak generally relies upon frame differencing to compress video data; however, the encoder periodically inserts a *key frame* into the data stream. Such a frame can be decompressed without reference to any frames which precede it. A key frame may either occur naturally as a result of an abrupt change of scene, or can be injected into the data stream at a prescribed rate to aid random access or resynchronization with audio.

 

### 2.1.3 Decompress()

This is the routine that actually displays the pixels.

| Stack Offset | Size | Description |
|---|---|---|
| 16(a7) | 4 | Value prior to entry is not important. Returns:<br>0 = successful completion<br>non-zero = error |
| 12(a7) | 4 | Address of $3000 byte auxiliary Cinepak data buffer (see section 2.4) |
| 8(a7) | 4 | Address of start of frame in bitstream. |
| 4(a7) | 4 | Frame buffer address of top left corner of image. |
| 2(a7) | 2 | Bytes per row in frame buffer |
| (a7) | 2 | Phrase Interleave Factor |

**Table 2.3** — 68000 stack setup before call to *Decompress*.

The latest version of Cinepak for Jaguar supports phrase interleaving for faster double or triple buffering schemes. If zero is passed as the *phrase interleave factor*, Cinepak will perform normally, writing its data contiguously in memory. A phrase interleaving factor of one will cause one phrase to be skipped for every one written. A phrase interleaving factor of two will cause two phrases to be skipped for every one written, and so on. This is done in a way that is compatible with similar features in the Object Processor and Blitter. By interleaving the buffers which must be blitted back and forth, the frequency of DRAM page faults drops signifigantly, increasing the available bus bandwidth.

### 2.1.4 HaltCPK()

This routine shuts down the Cinepak decompression code running in the GPU at the end of the current frame. It takes no parameters. To restart Cinepak you must start from the beginning again.

## 2.2 GPU Module

The *gpucode.og* module consists of approximately 2200 bytes of relocatable GPU code. The labels *DECOMP_S* and *DECOMP_E* defined in the *gpucode.og* module are used to locate the beginning and end of the Cinepak GPU code so that it may be copied to the GPU's internal RAM for execution.

After the code has been copied over to internal GPU RAM, the GPU is started. The GPU code detects the address at which it has been loaded by looking at the *GPUOffset* variable and then patches all instructions and table values which are position-dependent. It then notifies the 68000 via the *GPU_READY* flag (see Section 2.3) that it is ready to perform decompression tasks.

The Cinepak GPU code may be run from either register bank with some limitations. By default, Cinepak assumes it will run from Bank #0 and will set R31 to point to ten longwords of interrupt stack that it provides. As Cinepak requires registers R0-R27 (and R28-R31 are reserved for interrupts), if you run Cinepak in Bank #0, any interrupt code must preserve all Bank #0 registers. To run Cinepak in Bank #1 you must perform the following steps:

1. Load the Cinepak GPU code into GPU RAM.
2. Load a small startup stub somewhere else in GPU RAM.

3. Have the startup stub provide interrupt stack space and store the location in R31.
4. Switch to the second register bank.
5. Using the information in GPU_OFFSET, jump to the head of the Cinepak code.

When these above steps are performed, Cinepak will harmlessly change R31 in Bank #1 and continue to run from Bank #1. Interrupts (which must run in Bank #0) may then use R0-R27 of Bank #0 without saving them.

Once the system has been initialized, all GPU functions are invoked from within the routines in the *codec.o* module; no attempt should ever be made by your code to directly access the GPU decompression functions.

While the GPU is executing decompression functions, the 68000 is halted (a *stop #$2000* instruction is executed within *codec.o*). When the GPU finishes its task, it interrupts the 68000; the interrupt service routine sets a semaphore which is polled within *codec.o* to reawaken the 68000. This mechanism provides a 5-10% improvement in performance by minimizing GPU/68000 bus contention, and should not be circumvented.

The sample player program includes a utility subroutine named *LoadGPU* in the *util.s* file. This routine copies the GPU code from *gpucode.og* into GPU memory (see section 5.5). The load address is offset from the base of GPU memory by the constant value *GPU_OFFSET*, defined in the application-specific CINEPAK.INC include file. This offset is necessary to avoid collision with the GPU interrupt vectors.

Sample code for the GPU startup sequence appears in the module *player.s* (see Section 5), in the vicinity of label *WaitGPU*.

## 2.3 Flags

Storage for two flag variables must be declared within the DRAM address space. These are defined in Table 2.4. The initial values of these flags are not important.

| Flag | Size | Description |
|---|---|---|
| *semaphore* | 2 | Cleared within *codec.o* upon invocation of GPU task. Set by interrupt service routine upon completion of GPU task. |
| *GPUOffset* | 4 | Relocation offset of GPU code. Before you execute the GPU code from *gpucode.og*, this variable must be set to the offset from the beginning of GPU internal RAM (G_RAM) where the GPU code has been loaded.<br><br>The sample player program sets this to the constant value *GPU_OFFSET* at time GPU code is loaded. |

**Table 2.4** — Flags declared in DRAM address space.

An additional flag is declared (internal to *gpucode.og*) within GPU internal address space and must be accessed by the 68000, as defined in Table 2.5.

 

| Flag | Size | Description |
|---|---|---|
| *GPU_READY* | 4 | Cleared by 68000 prior to GPU startup. Set by GPU when initialization procedure has been completed.<br><br>To account for GPU code relocation, you must add the value of *GPUOffset* to this symbol in order to get the correct address. (For an example, see the code immediately before the *WaitForGPU* label in the sample program's *player.s* source file.) |

**Table 2.5 —** Flag declared in GPU internal address space.

## 2.4 Auxiliary Data

The *PreCompress* and *Decompress* routines require storage space in DRAM for auxiliary data structures, distinct from the Cinepak data bitstream. This buffer must be $3000 bytes in length and reside on a long-word boundary. Your Cinepak playback application must pass the address of a suitable buffer each time these functions are called. (Note that the same buffer may be used for both functions.)

# 3. Jaguar Film Format

The Cinepak bitstream is simply a source for a continuous stream of video; the bitstream contains no information pertaining to time, frame rate, or synchronization of video with other media such as audio.

To provide a time reference and synchronization among different media, the Cinepak bitstream must be embedded in some higher-level structure that is aware of time and the existence of media other than video. The Jaguar film format has been devised to meet these requirements.

The Jaguar film format exists in two flavors:

1) Smooth. This format is useful for playback of multiple low-resolution (for example, less than 160x100) films or a single film of higher resolution, provided in either case that the duration is very short (usually 3 or 4 seconds maximum). In this case, all the film data could be stored and played from ROM, or could be retrieved from the CD-ROM in a single brief access and loaded into DRAM for playing.

2) Chunky. This format is designed for playback of longer films that cannot fit in DRAM all at once. Here, periodic access to the CD-ROM is required on a continuing basis, so some mechanism must be incorporated in the film structure for locating and identifying the film data that are needed for display at a particular time.

The film formats are described in detail in the sections 3.1 and 3.2.

***Atari's existing sample Cinepak player code only knows how to play Chunky-format Cinepak Films. If your program needs to play smooth films, the changes would needed would be minor.***

## 3.1 Smooth Format

Table 3.1 defines the structure of a smooth film at the highest level.

| Field | Size | Description |
|---|---|---|
| *Frame Header* | 16 | Global film header |
| *Frame Description* | 20 | Frame size and compression type |
| *Audio Description* | 20 | Audio data format description |
| *Sample Table* | 16 + (*n* * 16) | Index to film samples which follow; *n* is number of samples |
| *Film Samples* | variable | Audio blocks and video frames |

**Table 3.1** — Smooth film format.

The *frame header* identifies the ensuing data as a Jaguar film and gives the offset to the start of the film data; its structure is defined in Table 2.1. The *frame description* provides information about pixel resolution and the format of the compressed video; Table 3.3 describes this structure. The *audio description* contains information about the format of any audio data included in the film. This is discussed in Table 3.5. (Note that some older Jaguar Cinepak films may not include this field.) The *sample table* provides a time-based index to the ensuing audio and video data which form the actual content of the film; Table 3.7 defines the structure of the sample table.

At the *film sample* level, the data stream is interleaved blocks of audio and video sample information; the *time* field of the sample record holds the key to the multiplexing scheme (see discussion following Table 3.8). The audio data itself uses the format defined by the film's audio description atom. The video data stream is in the proprietary Cinepak format, which is interpreted by the Cinepak decompressor.

### 3.1.1 Frame Header

| Field | Size | Description |
|---|---|---|
| *Header* | 4 | Human readable tag: 'FILM' |
| *AtomSize* | 4 | Size of film header, plus ensuing frame description and sample table |
| *Version* | 4 | Not used |
| *Reserved* | 4 | Not used |

**Table 3.2** — Structure of frame header.

The frame header is a 16-byte structure comprised of four long-word fields. The *Header* field is a human-readable tag, 'FILM', which identifies the ensuing global data structure as a Jaguar film. The *AtomSize* field gives the offset in bytes from the start of the header to the beginning of the audio and video data records; this offset includes the size of the frame header itself, plus the sizes of the ensuing frame description and sample table structures. The *Version* and *Reserved* fields are not currently used; developers are free to use these as they wish.

 

### 3.1.2 Frame Description Atom

| Field | Size | Description |
|---|---|---|
| *Header* | 4 | Human readable tag: 'FDSC' |
| *AtomSize* | 4 | Size of frame description atom (=20) |
| *CType* | 4 | Human readable compression type:<br><br>'cvid' = Cinepak compressed-RGB format<br>'$CRY' = Expanded Atari CRY format<br>'$RGB = Expanded RGB format |
| *Height* | 4 | Number of display lines |
| *Width* | 4 | Number of pixels per line |

**Table 3.3** — Structure of frame description atom.

The frame description is a 20-byte structure comprised of five long-word fields. The *Header* field is a human-readable tag, 'FDSC', which identifies the structure as a frame description. The *AtomSize* field contains the size of the frame description atom (i.e. 20 bytes). The *CType* field contains a human-readable code which identifies the format of the compressed video; two modes are recognized:

| Value | Meaning |
|---|---|
| 'cvid' | Cinepak compressed-RGB format |
| '$CRY' | Cinepak Expanded Atari CRY format |
| '$RGB | Cinepak Expanded RGB format |

**Table 3.4** — Frame Description Atom *CType* values

The *Height* and *Width* fields specify the vertical and horizontal resolution of the video in pixels.

### 3.1.3 Audio Description Atom

| Field | Size | Description |
|---|---|---|
| *Header* | 4 | Human readable tag: 'ADSC' |
| *AtomSize* | 4 | Size of audio description atom (=20) |
| *AudioData* | 4 | Audio Data Description |
| *SCLK* | 4 | SCLK timer value for audio playback |
| *AudioDrift* | 4 | Drift rate value used adjust audio sample rate |

**Table 3.5** — Structure of audio description atom.

The audio description atom is a 20-byte structure that defines the format of the audio data contained in the Cinepak film so that it may be played back properly. The *Header* field is a human-readable tag 'ADSC' which identifies the structure as an audio description atom. The *AtomSize* field specifies the size of the structure (20 bytes).

The *AudioData* field is a bitmapped flag that defines the data format of the audio, i.e. mono or stereo, compressed or non-compressed, 8-bit samples or 16-bit samples, and so forth. See Table 3.6 for a definition of the meanings of each bit. Note that the proper utilization of this information is the responsiblity of the Cinepak player application.

| Bits | Meaning |
|---|---|
| 0 | 0 = Mono, 1 = Stereo |
| 1 | 0 = 8-bit, 1 = 16-bit |
| 2-7 | Audio Compression Type:<br><br>0 = uncompressed<br>1 = $n^2$ compression<br>other values are reserved |
| 8-30 | Reserved |
| 31 | Two's Complement audio flag |

**Table 3.6 — Audio description flag bits**

The *SCLK* field contains the value which should be used with the Jaguar's SCLK timer to set the DSP interrupt frequency for audio playback. In Jaguar Cinepak films which have no audio information, the *SCLK* field will be set to -1 ($FFFFFFFF)[3].

The *AudioDrift* field specifies a 32-bit value that can be used by the player program's audio playback code to account for the difference between the audio data's original sample rate and the actual playback rate on the Jaguar. This value is added to an accumulator during each DSP sample rate interrupt. When a carry is generated, instead of proceeding to the next sample as usual, the current sample is reused instead. The audio drift rate is derived from the formula:

$$DriftRate = \frac{2^{32}}{SourceSampleRate \div (SourceSampleRate - JaguarSampleRate)}$$

The Jaguar sample rate is determined by:

$$VideoClockRate = 26590906Hz\ (NTSC),\ 26593900Hz\ (PAL)$$

$$JaguarSampleRate = \left\{ \frac{VideoClockRate}{2 \times (SCLK + 1)} \div 32 \right\}$$

You can work backwards from the *DriftRate* value and the Jaguar Sample Rate to get the original sample rate. You might do this, for example, in the event that you wanted to change the DSP code to perform linear interpolation to adjust the playback sample rate, rather than simply repeating samples. The formula for this is:

$$SourceSampleRate = JaguarSampleRate + \frac{JaguarSampleRate}{2^{32} \div DriftRate}$$

Note that older Jaguar Cinepak films may not contain an Audio Description Atom. If none is found, the player code should typically default to expecting 8-bit mono at a 22050 Hz (original) sample rate.

[3] This will only be true for films converted with versions of the *Jaguar Cinepak Utilities* dated June 1995 and later.

 

### 3.1.4 Sample Table Atom

| Field | Size | Description |
|---|---|---|
| *Header* | 4 | Human readable tag: 'STAB' |
| *AtomSize* | 4 | Size of sample table atom |
| *Scale* | 4 | Time scale of film |
| *Count* | 4 | Number of sample records in table |
| Sample records | 16 * Count | Array of sample records |

**Table 3.7** — Structure of sample table atom.

The sample table is a data structure which provides a time-based index to the film samples[4], i.e. blocks of audio and frames of video. The header is a human-readable tag, 'STAB', which identifies the structure as a sample table. The atom size field contains the size of the sample table atom, which encompasses the ensuing sample records.

The scale field provides the time scale for the film, in fractional units of a second, i.e. the unit of time is the reciprocal of the scale. A value of 600 is commonly used in QuickTime movies, as it is the lowest common multiple of the common rates of 24, 25 and 30 frames per second. The *MovieToFilm* tool does not alter the time scale embedded in the QuickTime movie when a Jaguar film is created.

The count field gives the number of sample records which immediately follow it; the sample record structure is defined in Table 3.8.

### 3.1.5 Sample Record

| Field | Size | Description |
|---|---|---|
| *Start* | 4 | Start of sample |
| *Size* | 4 | Number of bytes in sample |
| *Time* | 4 | Time at which to play sample |
| *Duration* | 4 | Duration of playback interval for sample |

**Table 3.8** — Structure of sample record.

The *start* field gives the starting address of the sample referenced by the sample record, relative to the end of the sample table. The end of the sample table coincides with the end of the frame header (see Table 3.2).

The *size* field gives the size of the referenced sample in bytes. Adding the *start* and *size* fields of the current sample record yields the value in the start field of the next sample record.

The 31 least-significant bits in the *time* field of the sample record give the time at which the referenced sample is scheduled to be played, in the units specified by the *scale* field of the sample table. If the value is $7FFFFFFF that indicates that the referenced sample (block) contains audio, not video, which should be played immediately following the end of the previous audio sample (block).

[4] The "sample" terminology is, unfortunately, somewhat ambiguous. In the context of a Cinepak film, it refers to a set of data which may be either audio or video. In the context of audio, it conventionally refers to the 8-bit or 16-bit datum which is read or written to a DAC. Where possibility for confusion exists, we use the terminology "block" to indicate the aggregate.

The most significant bit of the *time* field indicates a *shadow sync sample* (0) or not (1); this is a carry-over from QuickTime that should be ignored by the sample player code.[5]

The *duration* field of the sample record gives the play duration of the referenced sample, in units of the time scale. For an audio sample (block), the duration is meaningless; addition of the time and duration fields of the current video sample record yields the value in the time field of the next video sample record.

## 3.2 Chunky Format

The chunky format contains all the ingredients of the smooth format, except that additional structures are embedded in the data stream to partition it in time and provide mechanisms for random access on a CD-ROM disc. The highest-level structure is shown in Table 3.9.

| Field | Size | Description |
|---|---|---|
| *Frame header* | 16 | Global film header |
| *Frame description* | 20 | Frame size and compression type |
| *Audio Description* | 20 | Audio data format description |
| *Chunk table* | 16 + (*n* * 16) | Index to chunk data which follow; *n* is number of chunks |
| *Chunk data* | variable | Time-sequential chunks of film samples |

**Table 3.9** — Chunky film format.

The *frame header*, *frame description*, and *audio description* fields are identical to those already defined for the smooth format (see Table 3.2 and Table 3.3), except that the frame header atom size encompasses the ensuing chunk table.

The *chunk table* and *chunk data* fields are new fields especially created for chunky format films; they are defined in Table 3.9 and Table 3.11, respectively.

### 3.2.1 Chunk Table Atom

| Field | Size | Description |
|---|---|---|
| *Header* | 4 | Human readable tag: 'CTAB' |
| *AtomSize* | 4 | Size of chunk table atom |
| *Scale* | 4 | Time scale of film |
| *Count* | 4 | Number of chunk records in table |
| Chunk records | 16 * *Count* | Array of chunk records |

**Table 3.10** — Structure of chunk table atom.

The chunk table bears a close resemblance to its counterpart, the sample table (see Table 3.7). The differences are that the atom header 'CTAB' identifies it as a chunk table, and the chunk record, defined in Table 3.11, is a minor variation on the previously defined sample record.

[5] For more information, see the book *Inside Macintosh: QuickTime*, Addison-Wesley Publishing Company, 1993, pages. 2-134 to 2-135.

### 3.2.2 Chunk Record

| Field | Size | Description |
|---|---|---|
| Start | 4 | Start of chunk |
| Size | 4 | Number of bytes in chunk |
| Time | 4 | Time at which to play chunk |
| Sync pattern | 4 | Sync pattern for chunk |

**Table 3.11** — Structure of chunk record.

The chunk record is identical to the sample record (see Table 3.8), except that the duration field of the latter is replaced by the sync pattern field. This 4-byte field specifies the pattern that is replicated to form the sync marker for the chunk in the data stream.

### 3.2.3 Chunk Data

| Field | Size | Description |
|---|---|---|
| Sync | 64 | Sync marker used to locate chunk within data stream |
| Sample table | 16 + (*n* * 16) | Index to film samples which follow; *n* is number of samples |
| Film samples | variable | Audio blocks and video frames |

**Table 3.12** — Chunk data format.

The chunk data element begins with 64-byte sync marker. This is followed by the sample table and film sample data for all film samples which fall within the time boundaries of the chunk. The structure of the sample table is identical to that for the smooth format (see Table 3.7); however, the addressing of film samples by the start field is *local to the chunk*. The zero base is the end of the sample table, in analogy with the addressing for a smooth film.

## 4. Using A Jaguar Cinepak Film With CD-ROM

### 4.1 Introduction

Once you have created your film and converted it to the chunky Jaguar Cinepak format using the *SmoothToChunky* option of the ***Jaguar Cinepak Utilities*** program, you are ready to put the film onto a CD-ROM disc so that it may be played on the Jaguar. We will presume for now that you are using just one film per CD-ROM track.

The smooth format Jaguar Cinepak Film created by *SmoothToChunky* is used to create a track file using the ***Jaguar CD Track Creator*** program (see the **Jaguar CD-ROM** chapter). This puts the correct Jaguar CD-ROM track wrapper around your film data and gives you a track file that you can feed directly to your CD-ROM mastering software in order to make a CD-ROM disc.

Unfortunately, some CD-ROM mastering software packages do not have the ability to take a raw binary file and use it to create a track. They may require that the file must look like an AIFF or WAV audio file (even if that's not really what kind of data it contains). The AIFF or WAVE file wrapper is removed prior to the data being written to the disc. The current version of the ***Jaguar CD Track Creator*** has no option to add an AIFF or WAV wrapper to the files it creates; this must be done as an

additional step with a separate program. (The MKAIF tool supplied as part of the Jaguar sound & music package can be used for this purpose right now, but this feature will be added to future versions of the ***Jaguar CD Track Creator***.)

### 4.1.1 Film To AIFF

An early approach to the AIFF requirements of CD-ROM mastering software was the *FilmToAIFF* option of the ***Jaguar Cinepak Utilities*** program, which takes a Jaguar Cinepak Film and creates a new file with an AIFF audio file wrapper around the original data. This option should no longer be used.[6] First, it only works with Jaguar Cinepak Film files, which isn't the only thing you'll need to put onto a Jaguar CD disc. Also, it presumes that there will only be one Cinepak film in each CD-ROM track, which may not be the case if you have a lot of small movies instead of a few big ones. Finally, the files it creates do not follow the standard Jaguar CD-ROM track specification, so it cannot be used to create a master CD-ROM disc ready for production.

If your player code was originally set up to expect a film processed by *FilmToAIFF*, there are a few things to watch for when you change it over. First of all, *FilmToAIFF* has an option to put an extra wrapper around the film data.[7] This places 56446 bytes of leader data (all "A" characters) before the Jaguar Cinepak film data. Some older versions of Atari's sample player program expect to find this data and use an offset value defined by the *LEADER* equate to skip ahead by this amount on each read from the CD. If you stop using *FilmToAIFF*, you should make sure that your player software no longer does this. Also, *FilmToAIFF* inserts a 64-byte sync header with all "1" characters immediately before your Jaguar Cinepak film data. The player probably uses this to locate the start of the film. If this is the case, you must change it to look for the partition header created when you build a track file using the ***Jaguar CD Track Creator*** program.[8]

See the **Jaguar CD-ROM** chapter for more information on CD mastering considerations.

### 4.1.2 Other CD Mastering Considerations

Note that some older CD mastering software automatically inserts two seconds worth of silence at the start of each audio track. This results in extra data at the start of the track. Some versions of the sample Cinepak player code include a *SILENCE* equate that is used to skip past this data in a similar manner to the *LEADER* equate mentioned eariler. See the chapter **Jaguar CD-ROM** for more information.

## 5. Sample Playback Code

This section gives a comprehensive description of the sample code which is provided to demonstrate playback of Jaguar films from CD-ROM. The example is based on a film in the chunky format. The smooth format, being a subset, would not be as illustrative.

---

6 The *FilmToAIFF* option is still available in the current version of the ***Jaguar Cinepak Utilities*** program, but will probably be removed from future versions.

7 See section 8.5 for more detailed information on the *FilmToAIFF* conversion.

8 See the **Jaguar CD-ROM** chapter for detailed information on the ***Jaguar CD Track Creator*** program.

 

## 5.1 Modules Supplied

The sample code consists of the following source modules, in alphabetical order:

| | | |
|---|---|---|
| *player.inc* | *clear.s* | *dspcode.das* |
| *intserv.s* | *lister.s* | *memory.inc* |
| *player.s* | *utils.s* | *vidinit.s* |

A *makefile* is also provided to build the executable *player* code.

**Warning!** *Please note that the current version of the sample Cinepak player programs is not intended as a general example of Jaguar programming. It is intended to specifically demonstrate the use of the Cinepak decompression code, and nothing else.* ***Do not use this example to obtain startup code or as a shell for creating your own programs.***

## 5.2 Memory Map

The system DRAM and ROM emulator memory map is shown in Table 5.1. Relevant symbol definitions are contained in the module *memory.inc*.

| Address Range | Description |
|---|---|
| $0 - $3FFF | Exception vectors, CD-BIOS |
| $4000 - $57BF* | Player executable code |
| $57C0* - $FFFF | Not used |
| $10000 - $31BFF | Frame buffer |
| $31C00 - $33FFF | Not used |
| $34000 - $36FFF | Auxiliary Cinepak data |
| $37000 - $37FFF | Not used |
| $38000 - $137FFF | Film buffer (chunk table and film data) |
| $138000 - $13803F | Overflow (GPU fills beyond end of buffer) |
| $138040 - $1FFFFF | Not used |
| $800000 - $8FFFFF | Stub, reserved |
| $900000 - $9FFFFF | Debug history |
| * = Approximate address, may change with different versions of player program. | |

**Table 5.1** — DRAM and ROM emulator memory map.

The memory map may be freely rearranged, or compacted if necessary; however, there are several restrictions:

1. The base of the frame buffer (currently $10000) must be phrase-aligned.

2. The base of the auxiliary Cinepak data area (currently $34000) must be long-aligned.

3. The base of the film buffer (currently $38000) must be long-aligned.

## 5.3 Key Parameters

In this section, we describe several key parameters, defined in *player.inc*, which either have major impact on the behavior of the system or interact with similar parameters in the tools.

The *CBUF_SIZE* equate controls the size of the circular buffer which is used to store the chunk table and film data. It is currently set at 1 MByte, although the size may be reduced, particularly for low-resolution or short-duration films. The *HEAD_START* equate, which guarantees a minimum separation between read and write pointers upon startup must be adjusted along with *CBUF_SIZE*; maintaining the current ratio of 75% should be adequate.

The *GPU_OFFSET* equate determines the offset from the base of GPU internal RAM at which the Cinepak decompressor code is loaded. During initialization, its value is copied to the variable location *GPUOffset*, which the GPU code uses to relocate portions of its own code and data.

The *FILM_SYNC* equate must correspond to the 4-byte partition sync marker that is repeated 16 times (for 64 bytes total) immediately before the film data begins. The player code uses this to locate the beginning of the film data after it is ready from the CD. This sync marker is inserted in front of the Jaguar Cinepak film data by the ***Jaguar CD Track Creator*** program when you create the track files for the CD.[9] The *FilmToAIFF* option of the ***Jaguar Cinepak Utilties*** program always creates a sync pattern of "1111".[10]

[MF1]The *DRIFT_RATE* equate is used to account for the difference between the sample rate of the original audio data in the original QuickTime movie and the actual playback rate on the Jaguar. (See sections 3.1.3 and 5.6 for more information.)

### 5.3.1 << I don't see the AUDIO_LAG anywhere in CPKDEMO except in one place in PLAYER.S... It seems to me that this information is either misleading or incomplete, or else we wouldn't be able to work with different sized audio blocks... and we do.>>

**The *AUDIO_LAG* equate is a critical parameter in the calculation of when to start reading data from the CD-ROM. It is tied to the parameters *AUD_CHUNK* and *SAMP_RATE*, which represent the size of the audio blocks in the film data stream and the audio sample rate, respectively. The *AUD_CHUNK* parameter must correspond to the *kSoundChunkSize* parameter in the *MovieToFilm* tool.**

The *MAX_DELAY* equate limits how far the system can fall behind real-time display of video before it starts skipping video frames to catch up; it is currently set at 1/24 second. Because only key frames are displayed during the catch-up process, the video will appear jerky while this is happening. If this is too objectionable, the delay can be relaxed to 1/12 second. (Note that only fairly high throughput films should have problems with the video falling behind.)

---

[9] See the **Jaguar CD Mastering** section of the **Jaguar CD-ROM** chapter for more information on the ***Jaguar CD Track Creator*** tool.

[10] Atari recommends that you no longer use *FilmToAIFF*. See the **Using A Jaguar Cinepak Film With CD-ROM** section for more information.

 

The *SILENCE* and *LEADER* equates are used in computation of the time code for the beginning of each track, and must be consistent with how the CD is actually recorded. The *SILENCE* equate is used to keep track of any extra blank space which may be placed at the beginning of a CD track by your CD mastering software.[11] The ideal amount is zero, but some CD-ROM mastering software packages may not give you any choice. The *LEADER* equate should be set to 0 unless you are using *FilmToAIFF*, in which case you should set it to 24. (These values are based on a number of CD data blocks, which are 2352 bytes each.)

The *MARGIN* equate causes the seek to occur ahead of the target, in order to guarantee that the data stream is valid at the actual point of interest. In the sample code, *MARGIN* is set to 16 blocks; *this value should not be tampered with.*

The *SYNC_SIZE* parameter represents the number of bytes in the sync marker that is found before the film header or a chunk of data within the film. This should always be 64.[MF2]

The *SRCH_WIN* parameter controls how many blocks into the input buffer the *FindSync* routine will look for the sync marker pattern before giving up and returning an error. Its value is closely linked to that of *MARGIN* and should not be changed.

## 5.4 Key Variables

Table 5.2 lists several key variables in the system (declared near the end of *player.s*), and describes their function.

| Variable | Size | Description |
|---|---|---|
| *buffChunks* | 4 | Number of complete chunks in circular buffer. Value is computed in *SetNextGroup* subroutine. |
| *catchUp* | 2 | Flag which controls frame-skipping mechanism when video falls behind real time. Set if time slip exceeds *maxDelay*. Cleared when next key frame is encountered. |
| *cBufBase* | 4 | Base of circular buffer. Chunk table is relocated to *FILM_BASE*. Circular buffer starts immediately following chunk table. |
| *cBufSize* | 4 | Size of circular buffer (*CBUF_END - cBufBase*). |
| *CDWritePtr* | 4 | Current GPU write pointer for CD-ROM data. Value is updated by calling *GetCDWritePtr* subroutine. |
| *CRYmovie* | 2 | Flag indicates Cinepak compressed-RGB color format (0) or Atari CRY format (1). |
| *deltaTime* | 4 | Difference threshold between expiration time of circular buffer contents and current time, below which the next CD-ROM read activity is initiated. |
| *dest* | 4 | Frame buffer address of top left corner of image. |
| *filmChunks* | 4 | Total number of chunks remaining. to be read from CD-ROM Decremented in *SetNextGroup* subroutine. |
| *maxDelay* | 4 | Maximum amount video can fall behind real time before catch-up process is begun. Value must be computed because time scale of film is not known until run time. |
| *mediaOffset* | 4 | Offset in bytes from start of film on CD-ROM to first audio or video data. |
| *nextBufAddr* | 4 | Location in circular buffer of first chunk that will be played after expiration of current buffer contents. Computed in *SetNextGroup* subroutine. |

[11] See the **Jaguar CD Mastering** section of the **Jaguar CD-ROM** chapter for more information.

| Variable | Size | Description |
|---|---|---|
| *playPhase* | 2 | Flag keeps track of activity while CD-ROM is playing:<br>0: no activity;<br>1: playing initiated;<br>2: sync for next group of chunks detected<br>3: inhibit further play (end of film) |
| *pNextGroup* | 4 | Pointer to chunk record of first chunk in group that will be played after expiration of current contents of circular buffer. Computed in *SetNextGroup* subroutine. |
| *semaphore* | 2 | Semaphore used to awaken the 68000 after GPU has finished decompression task. Cleared by the 68000 when GPU task is initiated. Set upon receipt of GPU interrupt by the 68000. |
| *time* | 6 | 48-bit time in Q16 format. Set to zero when film playing is started. Updated during vertical interval interrupt service routine. |
| *timeIncr* | 4 | 32-bit time increment in Q16 format. It is the ratio of the time scale of the film to the vertical interval tick rate. This increment is added to *time* during vertical interval interrupt service routine. |

**Table 5.2 —** Key variables in system.

## 5.5 Utilities

Several utility routines are provided with the system to hide non-essential details and streamline the main code. These routines are all contained in the module *utils.s*.

Parameter passing to and from these routines is done via registers; the stack is not used. Table 5.3 summarizes the interfaces to the utility routines, along with their functions.

| Routine | Input | Output | Function |
|---|---|---|---|
| *FindSync* | d0: sync pattern<br>a0: starting address | a0: address following end of sync, or 0 if sync not found within SRCH_WIN bytes | Searches data stream beginning at (a0), until sync pattern, input in d0, is located. |
| *GetCDWritePtr* | None | None | Updates *CDWritePtr* location with current position of CD-ROM destination pointer. |
| *GetTimeCode* | d0: data offset from start of media | d0: time code in mm:ss:bb format | Converts byte offset to time code. |
| *LoadDSP* | None | None | Copies DSP program from DRAM to DSP internal memory. |
| *LoadGPU* | None | None | Copies GPU Cinepak decompressor code from DRAM to GPU internal memory and calls CD-BIOS to load support code. Initializes *GPUOffset*, needed for later access to GPU memory. |
| *LongDivide* | d0: unsigned 16-bit divisor<br>d1: unsigned 32-bit dividend | d1: unsigned 32-bit quotient | Performs long division, taking correct account of overflow (quotient exceeds 16 bits). |
| *ReadCDData* | d0: data offset from start of media<br>a0: starting destination address | None | Performs housekeeping on CD-ROM hardware, sets up write pointers, computes time code for seek and initiates CD-ROM playback. |

 

| Routine | Input | Output | Function |
|---|---|---|---|
| *SetNextGroup* | None | None | Computes *buffChunks* and *pNextGroup* for next group of chunks in circular buffer. Adjusts value of *filmChunks*. |
| *Snapshot* | None | None | Dumps 64-byte record of key registers and variables into ROM emulator address space. |

**Table 5.3** — Interfaces to utility routines.

## 5.6 Audio Playback

Audio playback is handled entirely by the DSP (see module *dspcode.das*), although it does use some information which is set up by the 68000 (in *player.s*). The player code looks at the film header for an audio description atom (see section 3.1.3). If one is found, then the information for the audio format is extracted and saved into variables for the DSP code to use. If no audio description is found, the player assumes that any audio data in the film will be mono, 8-bit samples in two's-complement format, with a playback sample rate of 21.867 kHz and original sample rate of 22250 kHz.

Two locations in DSP internal memory are used to pass parameters between the 68000 and the DSP, as shown in Table 5.4.

| Location | Size | Description |
|---|---|---|
| *DSP_ARGS* | 4 | Byte count in audio block |
| *DSP_ARGS+4* | 4 | Starting address of audio block in circular buffer |

**Table 5.4** — Locations used to control audio playback.

When the 68000 encounters an audio block in the circular buffer, it loads the starting address of the block into location *DSP_ARGS+4*, then the the byte count into location *DSP_ARGS*. The code which does this is located just following the *SampleLoop* label in module *player.s*.

The DSP polls the byte count location. When it sees a nonzero value, it reads the value, writes back a zero and reads the starting address of the audio data. On a sample rate interrupt, the DSP reads a byte from the audio buffer, writes it to the DACs and decrements its copy of the byte count. Because of the forward bias of audio in the film data stream (see Section 5.8), the DSP receives a continuous supply of audio data even if the video begins to lag behind schedule. However, should the byte count reach zero, null (silence) samples are written to the DACs until the 68000 next updates the parameters at *DSP_ARGS*.

A third DSP internal memory location, *AUDIO_DRIFT*, is loaded with either the *DriftRate* parameter from the audio description atom (see Section 3.1.3) if one is found, or otherwise from the *DRIFT_RATE* equate defined in the *player.inc* file (see Section 5.3). This must happen before audio playback is initiated. This value is used to adjust for the differences, or "drift", between the original sample rate of the audio data and the interrupt frequency at which it will be played back. After every sample is written to the DACs, the *AUDIO_DRIFT* value is added to an accumulator. When a carry is generated, it means that the error between the two sample rates has accumulated to a full sample, and an input sample

is dropped to compensate for the error. However, because the difference between the sample rates is fairly small[12] there is no discernible impairment in audio quality.

## 5.7 Interrupt Handling

The code for setting up and servicing interrupts to the 68000 is all contained in the module *intserv.s*.

On the vertical interval interrupt, the 68000 must refresh the object list for the object processor and increment the *time* variable. The object list refresh is very compact: only those data in the list which have been destroyed by the object processor need to be reconstructed; the remaining values survive from initialization. The *time* update is straightforward, except that a carry to the upper 16 bits must periodically be handled.

On a GPU interrupt, the 68000 must set the *semaphore* flag to awaken the main decompression task.

## 5.8 Buffer Management

Management of the read and write pointers in the circular buffer is the most difficult technical aspect of film playback.

Figure 5-A illustrates the essentials of the process. The read pointer for the video data being used by the decompression code advances through the circular buffer, consuming data as it goes. Meanwhile, the write pointer for data coming from the CD follows along behind it. Whenever the read pointer reaches the end of the buffer, it is reset to the beginning and the consumption of data continues without interruption. When the write pointer reaches the end of the buffer, the write process is suspended.

The duty cycle for CD-ROM access is the ratio of the combined video/audio data rate to the playback rate from CD-ROM. For a typical high-quality film, the combined rate might be 250 kBytes/sec; with a double-speed CD-ROM (~350 kBytes/sec), this translates to a duty cycle of roughly 70%.

Because the audio sample rate is typically much lower than the compressed video data rate, the audio read pointer, which is controlled by the DSP, advances at a much slower rate than the video read pointer. Note, however, that there can be dramatic differences in audio throughput rates depending on the audio format. For example, uncompressed 16-bit stereo audio at 22 kHz requires 4 times as much data throughput as 8-bit mono.

Since audio and video are multiplexed in the data stream, the audio pointer will periodically jump ahead to the next block of audio in the buffer. For this reason, the audio pointer has a rather jagged trajectory in buffer-time space; however, it always lies within an envelope having the same slope as the trajectory of the video pointer, but offset from it by a constant amount, as shown.

---

12 For example, the difference between an original sample rate of 22250 Hz and a playback sample rate of 21867 is only about 1.7%.

 

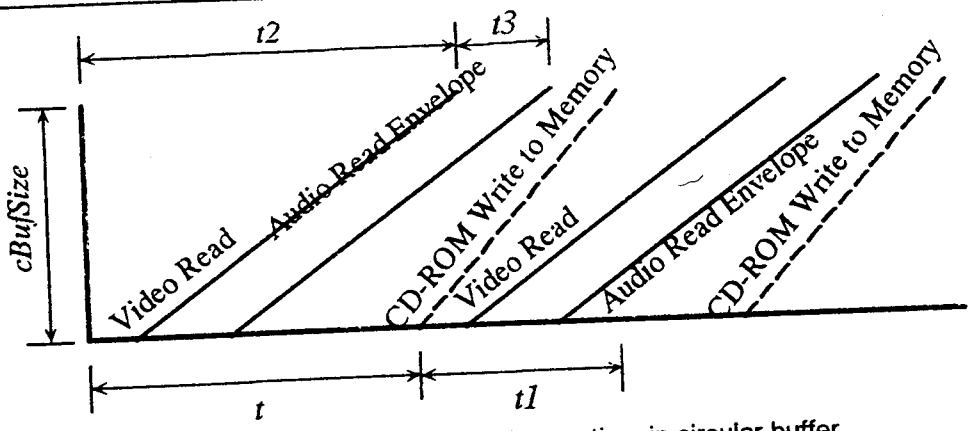


**Figure 5-A** — Pointer trajectories vs. time in circular buffer.

Referring to Figure 5-A, we define four times of interest:

*t* = zero-based time at which writing of CD-ROM data is initiated;
*t1* = time interval required to fill circular buffer;
*t2* = zero-based expiration time for current video data in circular buffer;
*t3* = lag between audio read envelope and trajectory of video read.

The heuristics of the buffer management process are as follows:

- Writing must be initiated late enough that the write pointer does not cross the tail end of the audio read envelope;

- Writing must be initiated soon enough that there is sufficient backlog of fresh data in the circular buffer at the time the video read pointer is reset.

In terms of the above-defined time values, these constraints translate to:

$$t + t1 > t2 + t3$$
$$t < t2$$

Solving both inequalities for t2 - t and rearranging, we obtain the concise result:

$$0 < t2 - t < t1 - t3$$

The most conservative design strategy is to split the difference, i.e.

$$t2 - t = (t1 - t3)/2$$

this is the approach which has been taken in the sample player code.

The combination (t1 - t3)/2 is referred to as *deltaTime* in the sample code (see also Table 5.2). The value is computed halfway between labels *CalcDest* and *.ClearWindow* in *player.s*. The comparison between t2 - t and *deltaTime* is made just after label *CheckCDPlay*, once it is determined that *playPhase* = 0.

The mechanics of transferring CD-ROM data to the circular buffer are all managed by the GPU interrupt service routine, which is loaded by an initial call to the CD-BIOS routine *CD_init*; this call is made as part of the *LoadGPU* subroutine in module *utils.s* (see Table 5.3). Subroutine *ReadCDData* takes care of all the overhead associated with setting up the BIOS calls to access the CD-ROM, including specification of an "end-of-buffer" address. When the write pointer has advanced to this address, the transfer of data is automatically suspended until the next call to *ReadCDData*; no further intervention by the playback code is required.

## 5.9 Frame Rate Control

The mechanism for frame rate control is fairly simple. The sample record (see Table 3.8) contains a field which indicates the scheduled time for the sample. The clock time, maintained by the vertical interval interrupt, is compared with the scheduled time and the system waits until the two times are the same. The code for doing this appears in *player.s* at label *KillTime*.

If the display of video falls behind schedule by an amount greater than *maxDelay*, then the *catchUp* flag is set and frames are skipped until the next key frame is encountered. When this occurs, the *catchUp* flag is cleared, the key frame is displayed and normal operation resumes. This code appears six to eight instructions on either side of label *LookForKey* in *player.s*.

Under most circumstances, there is ample processing power in the system to play full-screen video at 24 or even 30 frames per second, so the catch-up mode will seldom be activated. However, there may be situations in which developers will also want to use some portion of the GPU processing bandwidth for purposes other than video decompression; in these cases, the catch-up mechanism is essential.

## 5.10 Code Walkthrough

In this section, we give a complete walkthrough of the sample code in *player.s*, highlighting major points of interest along the way.

Before beginning, we define in Table 5.5 the use of several dedicated 68000 registers; this will clarify some of the explanations as we progress. All other registers are available for scratchpad computations.

| Register | Use |
|---|---|
| d4 | Pointer to compressed frame data |
| d5 | Counter for samples remaining in chunk |
| d7 | Counter for chunks remaining in circular buffer |
| a3 | Pointer to current sample record in circular buffer |
| a4 | Pointer to start of current chunk in circular buffer |
| a5 | Pointer to current chunk record in chunk table |

**Table 5.5** — Dedicated 68000 registers in film player code.

Between the start of the code and the label *WaitGPU*, the system is initialized. Much of the code used here -- especially in subroutines -- is either identical to, or a close derivative of early versions of generic

 

Jaguar sample code distributed by Atari. Note, however, that some aspects of this code are no longer considered to be good examples of general Jaguar programming.

The *Lister* subroutine has been modified to store certain entries in the object list in memory for subsequent use by the vertical interrupt interrupt service routine.

The *USE_CDROM* switch, set at assembly time, allows assembly of code that bypasses all access to CD-ROM; this is useful during development for testing short (three- or four-second) films by downloading them into memory from the hard disk.

After the GPU has finished initialization, the first access to the CD-ROM occurs. Data from the CD-ROM will be read into memory starting at location *FILM_BASE*. At label *.ClearWindow*, we allow the write pointer to advance beyond the end of the sync search window, then call *FindSync* to locate the start of the film. At label *CheckFilm*, we verify that the frame header tag (see Table 3.2) follows the film sync.

At labels *RelocTable* and *CopyCT*, the entire chunk table is moved from wherever it happened to land in the buffer to location *FILM_BASE*. Next, the *mediaOffset* variable is computed, since the byte offset for all subsequent accesses to CD-ROM data will be relative to the end of the chunk table. Following this, *cBufBase* and *cBufSize* are determined: the size of the chunk table is subtracted from the total available memory and whatever is left is allocated to the circular buffer. The *cType* field in the frame description atom is tested and the video is switched to CRY if the CRY tag is found.

The value of *dest* is computed at label *CalcDest*. In the sample code, the film is centered on the display; developers will obviously want to adapt this for their own purposes. After this, the *filmChunks* variable is initialized by copying the value from the *Count* field of the chunk table (see Table 3.10). Next, three key time variables are computed: *timeIncr*, *maxDelay* and *deltaTime*. Finally, register *a5* is set to point to the first chunk record (see Table 5.5).

We are now ready to look for the first chunk in the circular buffer. The search begins at *cBufBase*, with a sync pattern given by *$c(a5)*. At label *.ClearWindow*, we again wait to ensure that the write pointer has advanced beyond the end of the search window before calling *FindSync*. Upon returning from *FindSync*, we verify that the sample table header tag (see Table 3.7) follows the chunk sync.

At label *.ChunkOK*, register *a4* is set to point at the start of the chunk and *a3* to point at the sample table for the chunk. A call to *SetNextGroup* is made to determine which chunk will be the target of the next access to CD-ROM.

Two final steps are required before we are ready to play the film. At label *WaitToFill*, we allow the write pointer to get far enough ahead that the read pointer will not catch up to it. At label *WaitForTick*, we restart the vertical interval time clock at zero, since all time references in the film file are zero-based.

Label *ChunkLoop* is the top of the outer program loop. Register *d5* is loaded from the *Count* field of the sample table (see Table 3.7). The *AtomSize* field of the sample table is added to the base address of the sample table in *a3* to determine the address of the first data sample in the chunk; this is transferred to *d4*. Next, *a3* is adjusted to point to the current sample record.

Label *SampleLoop* is the top of the inner program loop. The call to *Snapshot* generates a time history in ROM emulator address space which is very useful for doing post-mortems during development; it should be commented out or deleted in production versions of the code. The *Time* field of the sample record is tested to determine whether the sample is audio or video. If it is audio, the arguments specified in Section 5.6 are passed to the DSP and a branch is taken to the end of the sample loop; otherwise, the program falls through to process video.

At labels *DoVideo* and *KillTime*, the *Time* field of the sample record is read and compared with the current *time* variable. If we are ahead of schedule, we wait until *time* has advanced to the scheduled value; otherwise, we check to see how far behind schedule we have fallen. If the slip exceeds the time specified by *maxDelay*, we begin the catch-up process described in Section 5.9; otherwise, we proceed to display the frame. The stack setup for the call to CheckKeyFrame is specified in Table 2.1.

The call to *ForceDelay* at label *DisplayFrame* can be conditionally assembled to simulate the catch-up process during development; there is no other use for *ForceDelay*. Next the stack is set up for the call to *PreDecompress* (see Table 2.2). Following the return, an error check is performed on the return value. At label *StartDecomp*, the stack is prepared for the call to *Decompress* (see Table 2.3); error checking is likewise performed upon return.

All of the code which manages the dynamics of writing to the circular buffer (excluding the initial write) appears between labels *CheckCDPlay* and *NextSample*. The *playPhase* variable, described in Table 5.2, is the key to controlling this mechanism:

- When *playPhase* is 0, the CD_ROM is not playing and the only task is to check the difference between the expiration time and the clock time and compare this difference with *deltaTime*. Note that the expiration time is recovered from the *Time* field of the chunk record which is addressed by *pNextChunk*. If it is time to start filling the buffer, the CD-ROM is given a seek address determined by the *Start* field of the chunk record pointed to by *pNextChunk*, playing is initiated with a write destination of *cBufBase*, and *playPhase* is set to 1; otherwise, a branch is taken to *NextSample*.

- When *playPhase* is 1, the CD-ROM is playing, and the only task is to locate the start of the next group of chunks in the circular buffer. Before calling *FindSync*, a test is performed to see if the write pointer has progressed beyond the end of the sync search window. If the test fails, the program does not wait, but branches to *NextSample*; this is to avoid needless delay in the middle of a loop that must execute in real time. If the test passes, the following actions are taken:

    - The sync search is begun at *cBufBase*, with a sync pattern specified by the *SyncPattern* field of the chunk record addressed by *pNextChunk*;

    - Error checking is performed;

    - The *nextBufAddr* variable is set at the sync location in the circular buffer and *SetNextGroup* is called to determine which chunk will be the target of the subsequent access to CD-ROM;

    - *playPhase* is set to 2.

 

Once *playPhase* has reached 2, there is nothing further to be done until the count (*d7*) of chunks currently in the circular buffer is exhausted. This situation is handled following label *ResetBuffer* (see below).

At label *NextSample*, the *Size* field of the current sample record is added to the address (*d4*) of the current sample to obtain the address of the next sample, and the pointer (*a3*) to the sample record is advanced to the next record. The counter (*d5*) for the number of samples in the current chunk is decremented, and if not exhausted, a backward branch is taken to *SampleLoop*.

If the sample count (*d5*) is exhausted, the counter (*d7*) for the number of chunks remaining in the buffer is decremented. If there are no chunks left, a branch is taken to *ResetBuffer*; otherwise, the *Size* field of the current chunk record is added to the address (*a4*) of the current chunk to obtain the address of the next chunk in the buffer, and register *a3* is set to point at the sample table for the next chunk. At this point, a test is made for an empty chunk (no video or audio scheduled) and a backward branch is taken to either *ChunkLoop* (not empty) or *NextChunk* (empty).

At label *ResetBuffer*, *d7* is reloaded from the *buffChunks* variable, which is set either in *SetNextGroup* or a few instructions below. If the value loaded is zero, the film is finished and we branch to *Done*.

For a nonzero value, *a5* is advanced to the next chunk record, *a4* is loaded from *nextBufAddr*, *a3* is set up to point to the sample table for the first sample in the new chunk, and *playPhase* is reset to zero. Next, the *filmChunks* variable (maintained by *SetNextGroup)* is tested to see if there are any chunks beyond those about to be processed that must be loaded from the CD-ROM. If so, a backward branch is taken to *ChunkLoop*.

If not, *playPhase* is set to 3 and *buffChunks* is set to zero. The first action inhibits any further access to the CD-ROM; the second causes the program to terminate when the current group of chunks has been exhausted. A backward branch is then taken to *ChunkLoop* to finish playing the film.

## 5.11 Error Trapping

There are several error conditions related to CD-ROM data integrity which are checked by the 68000 and trapped via an *illegal* instruction. When the trap is taken, register *d0* will contain an error code, according to the condition which caused the trap. Table 5.6 summarizes the traps and condition codes.

| Code | Condition |
|---|---|
| $00000000 | No error; playback completed normally |
| $11111111 | Sync pattern not found within search window |
| $22222222 | 'FILM' tag not found at start of film header |
| $33333333 | 'STAB' tag not found at start of sample table |
| $44444444 | Data error detected by *PreDecompress* |
| $55555555 | Data error detected by *Decompress* |

**Table 5.6** — Error codes and conditions.

These traps are useful for development and experimentation. They should never occur during playback of a finished Jaguar film.

# 6. Sample Jaguar Films

Three sample Jaguar films are provided on CD-ROM for demonstration purposes; any of the three films can be played using the sample code without modification. The film material has been approved for distribution and can be freely used for demonstration or evaluation.

Table 6.1 summarizes the characteristics of the three sample films:

| | Excerpt from "Jaws" | "Escape" sequence from Star Wars | Excerpt from "Back To the Future 3" |
|---|---|---|---|
| Resolution | 288 x 136 | 288 x 216 | 288 x 216 |
| Pixel depth | 16 bits | 16 bits | 16 bits |
| Color format | Cinepak RGB | Cinepak RGB | Cinepak RGB |
| Frame rate | 24 fps | 24 fps | 24 fps |
| Compressed video rate | 220 kB/sec | 260 kB/sec | 280 kB/sec |
| Audio sample rate | 22251.5 Hz | 22251.6 Hz | 22249 Hz |
| Chunk duration | 1.0 sec | 0.5 sec | 0.5 sec |
| Film duration | 2:33 min | 1:09 min | 1:08 min |

**Table 6.1** — sample Jaguar films.

All CD-ROMs are single-session with the film data recorded on track zero. This is the format expected by the sample player code.

 

## 7. Notice

Cinepak is a registered trademark of Radius, Inc. Jaguar is a registered trademark of Atari Corporation. QuickTime, Macintosh and MPW are registered trademarks of Apple Computer, Inc. Think C is a registered trademark of Symantec Corporation. CoSA and After Effects are registered trademarks of The Company of Science and Art.

# 8. Jaguar Cinepak Utility For Macintosh

The ***Jaguar Cinepak Utility*** program runs on the Apple Macintosh under System 6.1 or later (older versions of System/Finder may work, but have not been tested). The QuickTime extensions must also be loaded. When you run the program, you'll see a screen that looks like this:

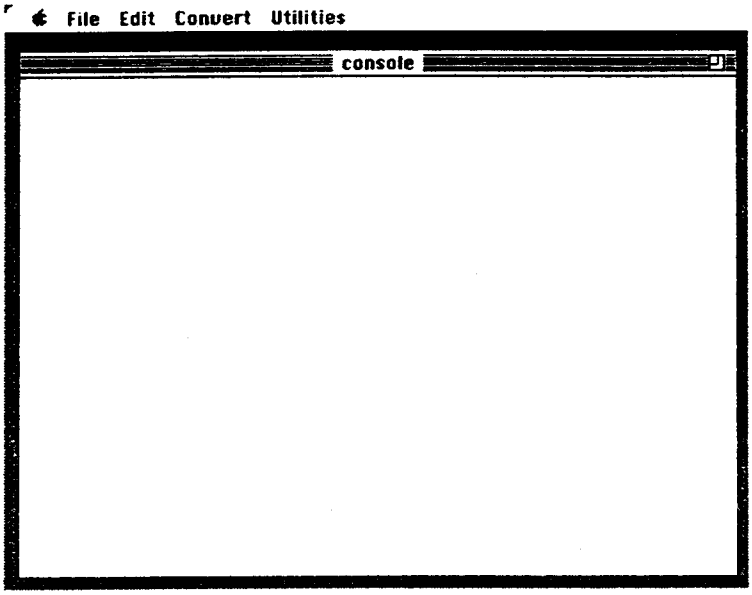


**Figure 8-A** — Jaguar Cinepak Utilities Screen

We'll assume that you know how to run programs and generally use the Macintosh computer. If this isn't true, please look through your Macintosh user's manual before attempting to run the ***Jaguar Cinepak Utility***.

The program displays a console window where messages from the various conversion functions will appear, as well as a menu bar at the top. The menus and the items they contain are described below.

## 8.1.1 *File* Menu

The *File* menu has just a single choice that allows you to quit the program.

## 8.1.2 *Edit* Menu

The Edit menu has the standard list of choices for Cut, Copy, Clear, and Paste. However, these options are not yet functional in this version of the utility. In future versions, they will allow you to edit the text shown in the console window.

## 8.1.3 *Convert* Menu

There are six conversion options available in the *Convert* menu. The first four are: *Movie To Film*, *RGB To Cry*, *Smooth To Chunky*, and *Film To AIFF*. These described in additional detail below.

 

The last two choices are *Convert A QuickTime Movie* and *Convert QuickTime Movie Batch.* These options allow you to combine the individual conversion steps represented by the top four menu choices. This is discussed in further detail below.

### 8.1.4 Utilities Menu

There is also a *Utilities* menu with options for displaying information about Jaguar Cinepak Film files and QuickTime movie files. These are discussed further detail below.

## 8.2 Movie To Film

The *Movie To Film* function allows you to convert a standard QuickTime Cinepak movie to the smooth Jaguar film format. Selecting this menu item leads to a dialog box that allows you to select the input file and the output file and conversion options as shown below:

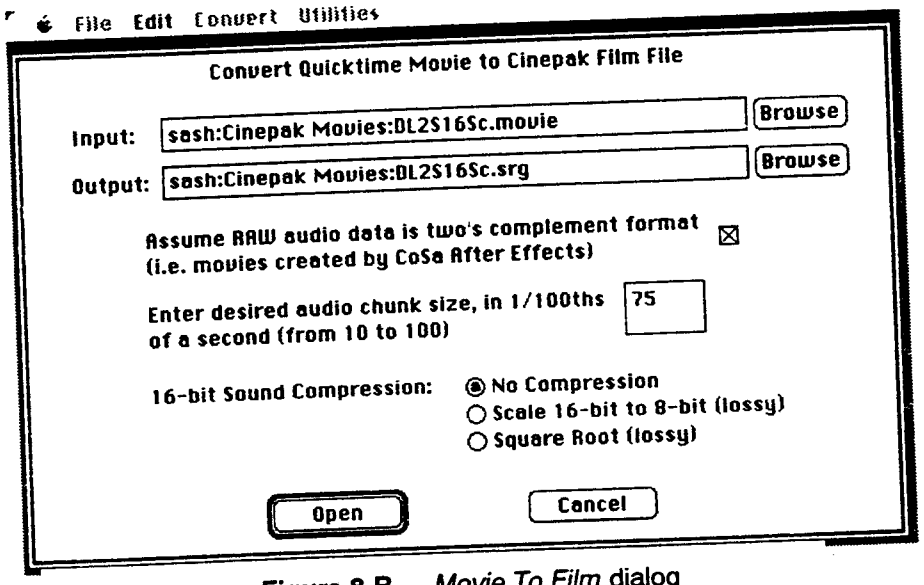


**Figure 8-B** — *Movie To Film* dialog

The input file must be an existing QuickTime Cinepak movie. You can type in the name of the file yourself, or you can click on the **Browse** button at the end of the **Input** field and the standard Macintosh file selector will appear and allow you to select the desired filename. In the event that the **Output** field is blank when you **Browse** for the input field, the input filename you select will be used to guess at the desired output filename. You may either use the guess directly or edit it as required.

The output file name may be specified by typing in a name or by selecting the **Browse** button and using the standard Macintosh file selector that appears. Any existing file with the same name as the output file will be overwritten. If you use the file selector to enter the output filename, you will be given a warning, but not if you simply type it in. *Note: Using a filename extension of ".SRG" is recommended.*

The *Assume RAW Audio Data...* checkbox allows you to inhibit the conversion of "Raw" audio tracks in the source QuickTime movie to the "Two's Complement" format needed for proper playback on the Jaguar.[13]

Audio data from the source movie is placed into the destination file in chunks interleaved with the video data. The length of each audio chunk is specified by the *Enter Audio Chunk Size...* edit box. This value is specified as *n*/100ths of a second, and should ordinarily be about 3/4 the size of the chunk size you will later specify in the *Smooth To Chunky* conversion process. The default size is 75/100ths of a second. Note that the actual amount of data placed into the audio chunk depends on the format of the audio data. If you use 16-bit stereo audio it will take 4 bytes per sample, versus 1 byte per sample for 8-bit mono.

Assuming an audio chunk size of 75/100ths of second, and video running at 24 frames per second, the audio will be placed into the destination file in the following way: the first audio chunk will be placed in the destination file immediately after the first frame of video. The second audio chunk will be inserted after video frame #10. The remaining audio chunks will be inserted every 18 video frames. This forward temporal bias in the audio stream means that the audio will play interrupted, as we will always have a little more audio remaining in the buffer than we have video, even in cases where the video playback starts to lag behind real time.

You may specify audio chunk sizes from 10/100ths to 1 second. If you later specify chunk sizes less than 1.0 seconds long in *Smooth To Chunky*, you should reduce the audio chunk size accordingly. However, please note that changing the audio chunk size to less than 3/4 of the chunk size later specified in *Smooth To Chunky* may affect the audio playback of the movie. If you have problems, try increasing the audio chunk size.

If the source QuickTime movie has a 16-bit audio track, then you have the option of compressing the audio data. There are two ways to do this. The first method is to simply scale the 16-bit samples to 8-bit. The second method uses a special square root compression algorithm. Each 16-bit audio sample is converted to an 8-bit encoded value as follows:

```
8-bit encoded value = sqr(original sample value / 2)
```

The 8-bit encoded values are then placed into the destination film file. During playback, these encoded sample values are expanded back to 16-bit. This compression method is still lossy (i.e. the output is not quite the same as the input), but the results are usually more pleasing to the ear than simply scaling 16-bit values to 8-bit.

---

[13] QuickTime movies typically specify either a "RAW" audio track or a "Two's Complement" audio track. The "Raw" type is normally the binary-offset format that is the default audio format used by the Macintosh. However, "Raw" also means the actual data format is not precisely defined, and some "Raw" audio tracks may not require conversion. This is the case with movies created by ***Adobe (CoSA) After Effects***, for example. Selecting the *Assume RAW Audio Data...* checkbox will inhibit the conversion of "Raw" audio tracks.

QuickTime movies that specify a "Two's Complement" audio track will normally not be converted regardless of the checkbox setting. However, if you hold down the **Shift+Command** keys on the keyboard when selecting the menu choices *Movie To Film*, *Convert A QuickTime Movie*, or *Convert QuickTime Batch*, these tracks will be converted if the checkbox is not selected. (Remember, the checkbox says "the audio is already Two's Complement, leave it alone.")

 

The actual *Movie To Film* conversion process is also accessed through the *Convert A QuickTime Movie* and *Convert QuickTime Batch* options.

## 8.3 RGB To CRY

The *RGB To CRY* function expands Cinepak-compressed RGB video data in a smooth format Jaguar Cinepak film to either CRY or RGB uncompressed. The movie's smooth film structure is not changed.

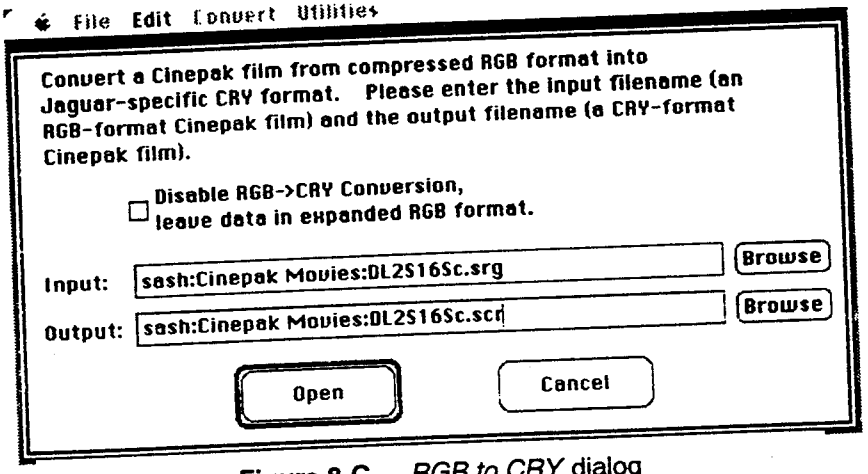


**Figure 8-C** — *RGB to CRY* dialog

the a smooth-format Jaguar film from the Cinepak compressed-RGB color format to either the Atari Jaguar CRY format, without altering the smooth film structure. Selecting this menu item will lead to a dialog box where you can select the input file, output file, and conversion options.

The input file must be an existing Jaguar Cinepak film in compressed-RGB format previously converted with *Movie To Film*. You can type in the name of the file yourself, or you can click on the **Browse** button at the end of the **Input** field and the standard Macintosh file selector will appear and allow you to select the desired filename. In the event that the **Output** field is blank when you **Browse** for the input field, the input filename you select will be used to guess at the desired output filename. You may either use the guess directly or edit it as required.

The output file name may be specified by typing in a name or by selecting the **Browse** button and using the standard Macintosh file selector that appears. Any existing file with the same name as the output file will be overwritten. If you use the file selector to enter the output filename, you will be given a warning, but not if you simply type it in. *Note: Using a filename extension of ".SRG" is recommended for movies with RGB video, or ".SCR" for movies with CRY video.*

The *RGB To CRY* function first decompresses the proprietary Cinepak RGB color data to a non-compressed RGB format. Checking the *Disable RGB->CRY Conversion...* checkbox disables the final conversion of this data to CRY mode.

Note that the decompression operation performed by *RGB To CRY* increases the amount of data needed to represent each frame of video, so various entries in the header and sample table are also adjusted to

reflect the change. The increase in size of the resulting film is typically about 10%, so there is minimal penalty in either storage or CD-ROM access requirements.

Cinepak films using non-compressed RGB or CRY video will consume about 10-15% less GPU processing bandwidth on playback than the same film using compressed-RGB video. The reason is that the processing step which converts from compressed to expanded RGB is bypassed (having already been done off-line). For certain highly complex movies where the frame rate may fall slightly short of 24 fps, developers may wish to take advantage of this time savings in order to squeeze maximum performance out of the system.

The actual *RGB To CRY* conversion process is also accessed through the *Convert A QuickTime Movie* and *Convert QuickTime Movie Batch* options.

## 8.4 Smooth To Chunky

The *Smooth To Chunky* menu item converts a Jaguar film from the smooth file format to the chunky format. Selecting this menu item will lead to a dialog box where you can select the input file, output file, and conversion options.

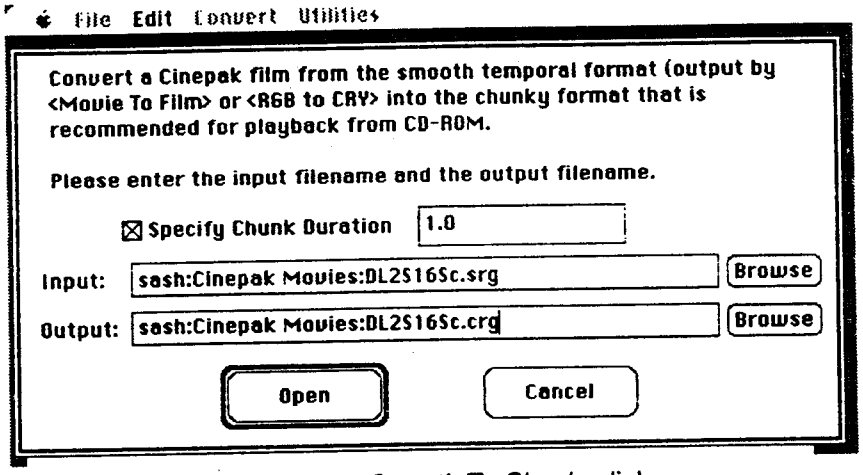


**Figure 8-D** — *Smooth To Chunky* dialog

The input file must be an existing smooth-format Jaguar Cinepak film previously created by either *Movie To Film* or *RGB To CRY*. You can type in the name of the file yourself, or you can click on the **Browse** button at the end of the **Input** field and the standard Macintosh file selector will appear and allow you to select the desired filename. In the event that the **Output** field is blank when you **Browse** for the input field, the input filename you select will be used to guess at the desired output filename. You may either use the guess directly or edit it as required.

The output file name may be specified by typing in a name or by selecting the **Browse** button and using the standard Macintosh file selector that appears. Any existing file with the same name as the output file will be overwritten. If you use the file selector to enter the output filename, you will be given a warning, but not if you simply type it in. *Note: Using a filename extension of ".CRG" is recommended for movies with RGB video, or ".CCR" for movies with CRY video.*

 

Checking the *Specify Chunk Duration* checkbox will cause an edit box to appear where you can specify the chunk duration, in seconds, of each chunk of data that will be placed into the destination file. If this option is not invoked, a default value of 0.25 seconds is used.

The *Smooth To Chunky* function takes a smooth-format Jaguar Cinepak film and converts it into a chunky-format Jaguar Cinepak film. This essentially takes audio chunks and frames of video from the smooth file and places them into chunks of a particular playback duration.

See sections 3 to 3.2 for further information on the smooth and chunky Cinepak film formats.

A chunk never begins with audio; an audio block which happens to fall on a chunk boundary is always incorporated as the final data element in the earlier of the two chunks.

The global chunk table is inserted near the beginning of the file, following the frame description atom. Synchronization blocks and local (i.e. intra-chunk) sample tables are inserted at the beginning of the film data for each chunk. This is so that the data for each individual chunk may be reliably located when reading from CD-ROM.

Developers are free to experiment with the chunk duration. On the low end, the value is limited by the increase in the length of the chunk table, which must be stored in its entirety in DRAM. On the high end, the duration is limited by increasingly inefficient use of DRAM buffer space. Incomplete chunks at the end of the circular buffer constitute wasted storage; the fraction of wasted buffer space will increase with progressively larger chunk durations.

The actual *Smooth To Chunky* conversion process is also accessed through the *Convert A QuickTime Movie* and *Convert QuickTime Movie Batch* options.

## 8.5 Film To AIFF

**Warning!** *The* FilmToAIFF *option was designed in response to certain CD Mastering software packages which could not accept a raw binary file and use it to create a CD track. Note that the files written by* FilmToAIFF *do not follow the standard Jaguar CD track format specification. Using* FilmToAIFF *is no longer recommended. Use the* ***Jaguar CD Track Creator*** *program instead. See the information in the* ***Jaguar CD Mastering*** *section of the* ***Jaguar CD-ROM*** *chapter, as well as section 4.1.1 of this chapter for additional information.*

The *Film To AIFF* menu item converts a Jaguar film file to an AIFF file, suitable as input to CD-ROM recording software which accepts audio files in this format. Selecting this menu item will lead to a dialog box where you can select the input file, output file, and conversion options.

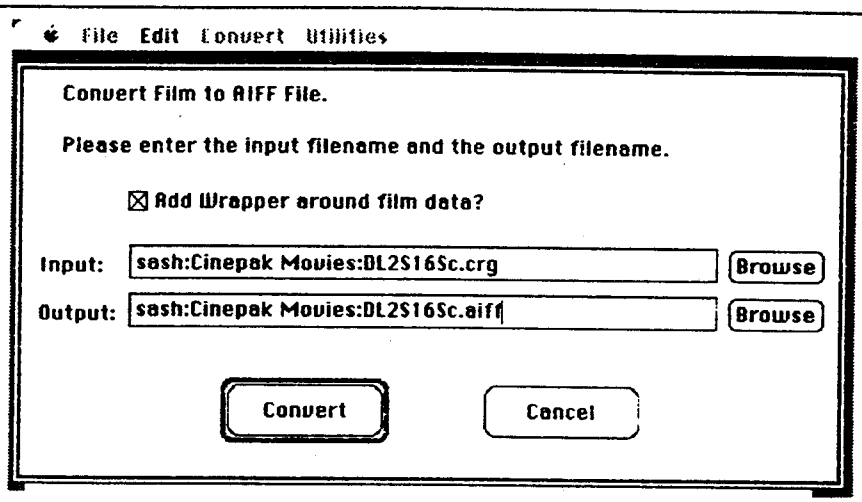


**Figure 8-E** — *Film To AIFF* dialog

The input file must be an existing Jaguar Cinepak film in either smooth or chunky format created by *Movie To Film, RGB To CRY,* or *Smooth To Chunky*. You can type in the name of the file yourself, or you can click on the **Browse** button at the end of the **Input** field and the standard Macintosh file selector will appear and allow you to select the desired filename. In the event that the **Output** field is blank when you **Browse** for the input field, the input filename you select will be used to guess at the desired output filename. You may either use the guess directly or edit it as required.

The output file name may be specified by typing in a name or by selecting the **Browse** button and using the standard Macintosh file selector that appears. Any existing file with the same name as the output file will be overwritten. If you use the file selector to enter the output filename, you will be given a warning, but not if you simply type it in. *Note: Using a filename extension of ".AIFF" is recommended.*

There is also a checkbox for an option that is used to cause the film data to be "wrapped" by the header/sync and tailer data structures defined in Table 4.1 before the AIFF file header is added.

This tool is included primarily as a convenience to those developers using CD-ROM mastering software which cannot do this conversion or which do not accept raw data files as input. [MF3]

Developers who choose to use or adapt *Film To AIFF* should be aware of three work-arounds in the code which have been introduced to compensate for bugs in the driver software that was used in creating the sample CD-ROM:

- The header and tailer sizes are increased by two bytes each to preserve long alignment of the film data on the recorded medium (see references to *HACK_SIZE* in the definitions of *HEAD_SIZE* and *TAIL_SIZE*);
- *SYNC_SIZE* is omitted from the computation of *fileSize*;
- The *numSampleFrames* field of *commonChunk* does not correctly account for the number of channels (=2) and the number of bytes per sample (=2).

 

The latter two work-arounds are needed to prevent spurious failure of the recording process and the attendant destruction of a CD-ROM.

The actual *Film To AIFF* conversion process is also accessed through the *Convert A QuickTime Movie* and *Convert QuickTime Movie Batch* options.

## 8.6 Convert A QuickTime Movie

The *Convert A QuickTime Movie* menu item brings up a dialog that combines the functionality of the separate *Movie To Film*, *RGB To CRY*, *Smooth To Chunky*, and *Film To AIFF* functions into one place. Please see the documentation for those functions before using *Convert A QuickTime Movie*.

The options in the *Convert A QuickTime Movie* dialog correspond to the options in the separate *Movie To Film*, *RGB To CRY*, *Smooth To Chunky*, and *Film To AIFF* dialogs with just a few exceptions, as detailed below.

First, the options currently selected affect the output filename that is automatically created when you *Browse* the input filename. For example, if you have *RGB Compressed* and *Smooth Film* selected, the output name will have an extension of ".SRG". But if you have *CRY Non-Compressed* and *Chunky Film* selected, the output name will have an extension of ".CCR" instead.

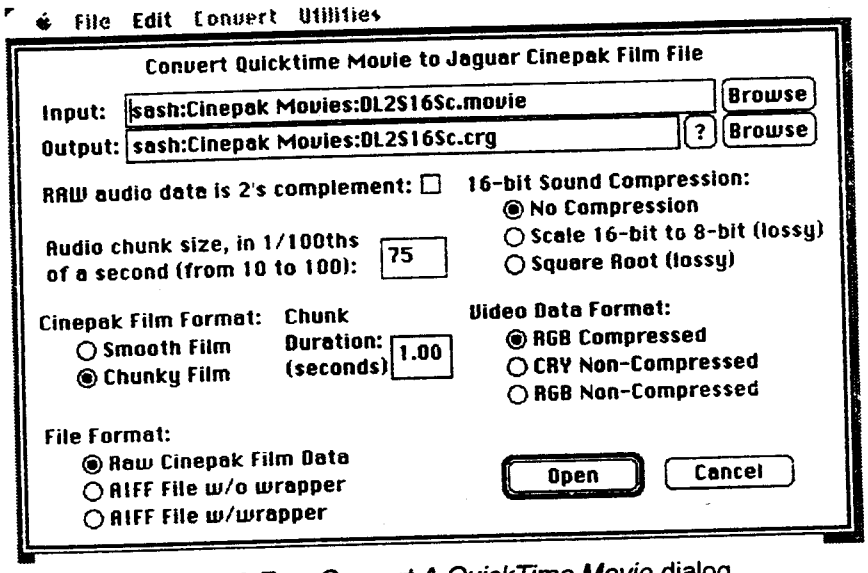


**Figure 8-F** — *Convert A QuickTime Movie* dialog

In the event you want to change the options after having selected the input filename, you can force the dialog to recreate the output filename to match the new options by clicking on the "?" button next to the output filename field's *Browse* button.

Just because the choices are all in one dialog does not change the fact that there are still up to four separate conversion steps involved. When you exit the dialog, *Convert A QuickTime Movie* will call the *Movie To Film* conversion as well as whichever of the three other conversion steps are appropriate for the options you have selected.

Any intermediate files required between the source and final destination will be created and deleted as needed. You will typically need to have approximately 2.2 times as much free disk space available as the size of your source movie. Please note that the amount of free disk space is ***not checked*** prior to the beginning of the conversion process.

Holding down the SHIFT+COMMAND keys when selecting the *Convert A QuickTime Movie* menu item will cause the *Raw Audio Data is Two's Complement* checkbox setting to affect QuickTime movies with the "twos" audio format as well as movies with the "raw" audio format.

## 8.7 Convert QuickTime Movie Batch

The *Convert QuickTime Movie Batch* menu item brings up a file selector which allows you to select the filename of a text file containing a list of QuickTime movie files to be converted. This file may be arbitrarily long and can therefore allow you to process dozens or even hundreds of QuickTime movies at once.

Each line in the batch file must specify a list of desired options and the source filename. You may also specify the destination filename, but if none is specified, one will be created based on the conversion options selected. The available command line options are:

| Option | Description |
|---|---|
| -a{*n*} | Specify audio chunk size. {*n*} is the chunk size in n/100ths of a second. The default value is 75. Must be in range of 10 to 100. |
| -c{*n*} | Chunk duration in seconds for chunky movies. The {*n*} value should be a floating point number. The default is 1.0. Note that this number affects your CD-ROM buffer size requirements: longer chunk durations require a larger buffer. |
| -cmp{*n*} | Compress 16-bit audio (if that's what is in the source movie) {*n*} must be one of:<br><br>0 = No compression (default)<br>1 = Simple 16-bit to 8-bit scaling<br>2 = Square-Root 16-bit to 8-bit compression |
| -f{*n*} | File format. {*n*} represents the desired file format. In most cases<br><br>0 = Raw Cinepak film (default)<br>1 = AIFF w/o wrapper<br>2 = AIFF w/wrapper |
| -film{*n*} | Specify Cinepak film format. {*n*} must be one of:<br><br>0 = Smooth (suitable for small RAM-based movies, not really for CD-ROM)<br>1 = Chunky (default, designed for CD-ROM playback) |
| -twos | Specify that "RAW" audio tracks in source QuickTime movie are Two's complement format and do not need conversion. Note that if the QuickTime source movie has the "twos" flag set on the audio tracks, this conversion is deselected unless you hold down the SHIFT+COMMAND keys when selecting the *Convert QuickTime Batch* menu item (in which case it uses the *-twos* flag). The default for this option is off. |

 

| Option | Description |
|---|---|
| -v{*n*} | Video mode. {*n*} represents the desired video mode and must be one of:<br><br>0 = RGB compressed (default)<br>1 = CRY Expanded<br>2 = RGB Expanded |

These options allow you to select the same items as the various conversion dialog boxes. A typical batch file might look like this:

```
//    This is a comment in my batch file...
#     This is another comment.
//    This is the last (third, actually, in a series of three) comment.

-a37 -film1 -c0.5 -v0 -f0 "sash:Cinepak Movies:DL2S16Sc.Movie"
-a45 -film0 -c0.6 -v1 -cmp2 -f0 "sash:Cinepak Movies:DL3S16Sc.Movie"
-a60 -film1 -c0.75 -v2 -f1 "sash:Cinepak Movies:DL4S16Sc.Movie"
```

Note that any line in a batch file that starts with "#" or "//" is ignored and may be used as a comment. Blank lines are also ignored.

The first line in the example that would be processed specifies an audio chunk size of 37/100ths of a second (**-a37**), a chunky format film (**-film1**), a chunk size of 0.5 seconds (**-c0.5**), RGB-compressed video (**-v0**), and a Raw Cinepak data file (**-f0**).

This command would cause the file "`sash:Cinepak Movies:DL2S16Sc.crg`" to be created from the source file "`sash:Cinepak Movies:DL2S16Sc.Movie`". (Remember, if not otherwise specified, the name of the destination file is always generated automatically based on the conversion options selected.)

In a batch file, all command line options are persistent from one line to the next unless changed. If one command line in the batch file sets up certain options, they remain in effect until changed by another command line. For example, the second example shown above specifes 16-bit audio compression using the square root method (**-cmp2**). The third command line example does not specify any "-cmp" option, so the "-cmp2" from the previous command will carry over.

This option is essentially a batch file version of the *Convert A QuickTime Movie* option, and therefore similar rules apply. In particular, please note that that the individual functions *Movie To Film, RGB To CRY, Smooth To Chunky,* and *Film To AIFF* are called by the batch file processor to perform whatever conversions are required.

When doing batch file processing, the disk-space availability check done by the individual menu choices and dialogs is NOT PERFORMED. So make sure you have sufficient disk space before attempting a batch conversion. Try to ensure that you have about as much free disk space as the total disk space of your source files, plus the size of your largest file. (i.e. if you have 5 files totaling 10mb, and the largest file is 2mb, then you need about 12mb free disk space total. However, keep in mind these are rough estimates and give yourself as much room as possible.

WARNING: Please keep in mind that each movie can take up to several minutes at a time to convert. Large movies can easily take an hour or more. So before you start processing a batchfile with a hundred commands, remember that it could easily take several days to finish. Make sure that that you have a good understanding of the process and always run a reality check using just one or two movies first.

Also, the batch file processing feature removes the necessity of you, the user, having to sit at the computer and guide each file through the conversion process, but it does not reduce the time required to convert each file. Because there is currently no facility for breaking out of the middle of a batch job, it is suggested that you try converting just a few movies at a time until you get a feel for how long the process is going to take. The time required for each of the conversion steps is directly related to the size of the file you are converting, with the exception of CRY-expanded or RGB-expanded video output, which will also depend on the compression ratio of the original video data.

## 8.8 Show Film Info

The *ShowFilm Info* menu item brings up a dialog where you can select a Jaguar film file and select one of three different degrees of verbosity.

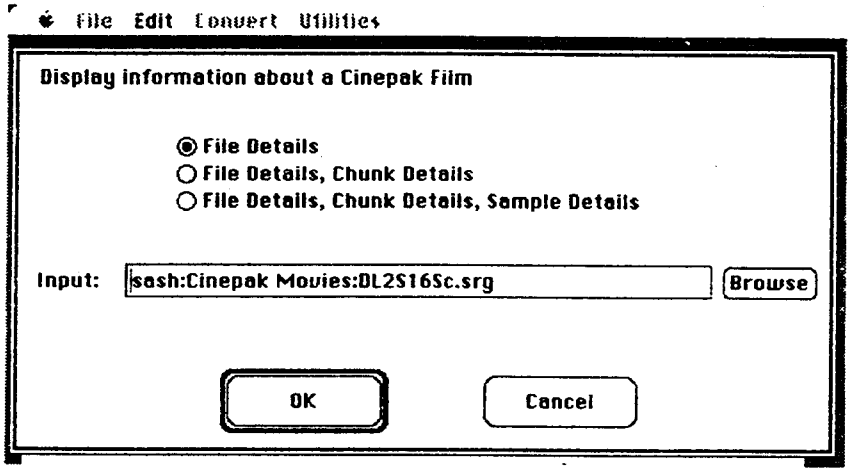


**Figure 8-G** — *Show Jaguar Cinepak Film Info* dialog

To get just the basic information about a Jaguar Cinepak Film, select the *File Details* radio button. To also get the the details for each chunk of the Jaguar Cinepak Film, select the *File Details, Chunk Details* radio button. To get the maximum amount of information, including the details of each block of sample data in the Jaguar Cinepak Film, select the radio button *File Details, Chunk Details, Sample Datails.*

The specified film file will be analyzed and the requested information about the contents will be dumped to the screen. To pause the screen output, hold down the mouse button, and release it when you want to continue. (The information printed is identical to the FILMINFO tool available for MSDOS.)

## 8.9 Show QuickTime Movie Info

The *ShowQuickTime Movie Info* menu item brings up a standard Macintosh File selector and allows you to select a QuickTime movie. This will cause information about the movie, such as the movie length,

 

frame size, number of video frames per second, type of audio tracks, audio data format, and so forth, will be printed into the console window.