Benchmarking Digital Video: Measurements, Analysis, Improvements and Lessons Learned *

Richard Gerber and Ladan Gharai
Department of Computer Science
University of Maryland
College Park, MD 20742
{rich, ladan}@cs.umd.edu
Tel: (301) 405-2710
FAX: (301) 405-6707

May 9, 1995

Abstract

Digital video applications can often push a computer's resources to their limit. They require massive amounts of storage, high IO transfer rates, and fast display refresh times. And if software is involved in the decompression process, the CPU will often end up over-utilized. Resource demands have a direct effect on the quality of the delivered video; this results in a complex "load-balancing" problem, which must be solved with both quantitative and qualitative metrics.

In this paper we describe our experiments on media applications, specifically concentrating on the tradeoff analysis involved in tuning video systems. We first postulate a set of hypotheses, and then we describe the controlled set of 240 tests we ran to test them. Our observations confirmed that achieving smooth playback is mainly a problem of coordinating an operating system to the properties of the media.

The first 120 test runs were drawn from a series of 60 videos, which we generated with our own Hi8 equipment. Each test video uniquely instantiated the following variables: *codec, frame size, digitized rate, spatial quality* and *keyframe distribution*. The tests were carried out on two Apple Macintosh platforms: at the lower end a Quadra 950, and at the high-end, a Power PC 7100/80. Our quantitative metrics included average playback rate, as well as the rate's variance over one-second intervals.

This paper contains the results of these experiments, as well our analysis of each variable's importance. The first set of results unveiled some surprising anomalies, which seemed to substantiate our hypotheses – even more than we expected. These anomalies forced us to run an additional 120 tests, whose analysis led to some broad conclusions about digital video specifically, and the capabilities of personal computers in general.

*This research is supported in part by ONR grant N00014-94-10228, NSF grant CCR-9209333, and NSF Young Investigator Award CCR-9357850.

1 Introduction

The term *digital video* is commonly used to describe three seemingly different activities: producing, storing and playing videos on modern computing systems. However, it takes only a casual experience with any of these activities to understand the intimate connection between them. For example, if one uses some expensive, specialized peripherals (including a high-end digitizer, a RAID disk configuration, etc.), one can, in theory, edit an entire broadcast-quality, full-screen video production on an Apple Macintosh computer.

Of course this raises the issue of storage. In the parlance of videography, a digitized, *broadcast-quality* video demands a screen size of 640×480 pixels, a display rate of 30 frames per second (or 29.97 frames per second on NTSC displays), and an RGB color depth of 24 bits per pixel. Some simple multiplication yields a demand for 1.66 Gbytes per minute, or roughly 50 Gbytes for a one-half hour production!

And then there is the problem of playback. Even if the producer has the wherewithal to digitize, edit and store a feature-length video, the chances are high that no consumer would be able play it. The original production will require 50 Gbytes of storage – and demand an effective playback rate of 27 Mbytes per second – whereas a typically well-equipped, consumer-grade workstation possesses no more than 2.5 Gbytes of disk space, with peak transfer rates at about 2.5 Mbytes per second. Moreover, the computer's display logic will probably not include high-end video de-compressor functionality; thus video decompression will be done in software, as will buffer management, synchronizing the video and audio, etc.

Clearly, the quality of a digital video is directly proportional to the computing resources it requires. Thus the problem of tuning a video production to a target platform – and tuning the platform to the video – demands something akin to a traditional "load-balancing" solution, applied with both quantitative and qualitative metrics.

We are studying the field of digital video essentially as an operating systems application, and one that possesses a strong real-time component. On one hand, we have found media applications to be ideal for real-time research. After all, the end-to-end requirements can be clearly and precisely articulated – e.g., 30 full-color frames delivered per second, accompanied by 16-bit digital sound played at 44kHz, with sound and video synchronized five times per second. Also, when the system fails to achieve a desired level of quality, the resulting behavior can easily be perceived by the viewer. Finally, we need not deal with abstractions of applications (as in avionics control problems); rather, we can experiment with media applications on consumer-grade workstations.

Yet real-time video applications possess unique difficulties not present in "hard" real-time systems. Most of these stem from an obvious point: sampling and update rates should be determined by factors related to "human perception," which is not amenable to mathematical models of "acceptable service." It is true that playback rates of 27 Mbytes/sec are not achievable on modern, consumer-grade systems. It is equally true that there are many ways of reducing these rates, by sacrificing quality – e.g., by introducing greater signal loss into digital compression, or by decreasing color-depth, playback rate, playback size, etc. The theory is that if any of these options are chosen, then disk transfer rates are reduced, as are storage requirements, CPU utilizations, demands for RAM buffers, overload sensitivity – with the result being a smoother, more deterministic video.

But it is not at all apparent which option (or combination of options) should be selected to achieve the greatest benefit. Indeed, as we show in this paper, it is not even true that a "quality-reducing" measure necessarily leads to reduced transfer rates (it occasionally has the opposite effect). And it is never a trivial venture to predict which combination of actions leads to the greatest achievable quality at playback.

On the other hand, while more powerful equipment *may* lead to better playback quality, there is no guarantee that this will always be the case. As we show, faster CPUs, faster disks, and wider IO paths will not always translate into better video quality – even in the best "multimedia computers."

1.1 Observations and Hypotheses

The work described in this paper grew out of some informal observations, which led to several hypotheses, which in turn were tested by a controlled series of 240 experiments. The observations were as follows:

- Using several QuickTime [1] movies as samples, and Apple Macintosh computers as our platforms, we found that the selection of codec[1], frame rate, and color quality had unpredictable consequences. For example, a supposedly "smooth" codec like Radius's Cinepak would, at times, lead to more color distortion than Apple's "default" video codec – which is basically a crude motion-JPEG scheme.

- It was not always the case that upgrading equipment led to discernibly better video playback. This was even true when we compared a Quadra 950 to a PowerPC (MC601) 7100/80 – when the PPC machine ran a movie player compiled into "native" MC601 code, and where the platform had 8 Mbytes more memory.

- "Skips" and "jumps" in video playback were noticeable across a wide spectrum of machines and sample videos. In fact, they looked suspiciously like buffer overruns, pre-fetching glitches, and similar, very familiar problems. For example, frames were not dropped at measured, deterministic rates; rather, a second's worth of video would play, then the following second would stall, then play, then stall, etc.

These observations led to a growing confidence that media playback should be studied as a problem of coordinating the operating systems with the properties of the media – and not specifically as a problem of data compression. While compression algorithms can have a profound effect on visual quality, from the perspective of balancing the resources, a specific compression algorithm can (and perhaps should) be treated as a "black box."

That said, we postulated the observations above as the following hypotheses, which we then set out to test.

[1] A codec is a compression/decompression scheme

1. *A faster, more powerful platform will not necessarily lead to better playback quality.*
2. *A movie's size (in bytes), and its "preferred playback rate" (in frames per second) are not always good predictors of quality.*
3. *Movie-playing software could give better performance if it (1) took into account the platform's capabilities, and (2) provided deterministic rate control based on these characteristics.*

1.2 Metrics and Variables

Before setting out to test these hypotheses, we were presented with four initial hurdles: (1) developing sound, quantitative metrics which loosely correspond to "qualitative performance;" (2) selecting suitable, comparable platforms on which we would benchmark our experiments; (3) finding a set of test videos with a controlled spectrum of content (e.g., different color and light densities, sound quality, scene transitions, etc.), which would permit making some general conclusions from our results; and (4) identifying the test variables, whose instantiations would generate our "test runs."

(1) Quantitative Metrics. It is true that "quality" is fundamentally a subjective attribute. However when analyzing clusters of similar experiments, we found two metrics that stood out as roughly correlating to visual quality. They are: (a) total frames displayed vs. total frames in movie, and (b) the display rate's variance (measured in one second quanta) over the course of the movie.

When comparing different "runs" of a single movie, the first metric gives an indication of *average* playback quality over the course of each run. Letting e denote a run of a digital movie, we denote $\mathcal{F}_T(e)$ as the movie's total, "preferred" number of frames that *should* be displayed throughout the run. Alternatively, we let $\mathcal{F}_D(e)$ be the measured number of frames that *actually* get displayed. If $t(e)$ is the movie's duration (in seconds), we can extract the following properties:

$$\begin{aligned} \mathcal{R}_{\mathrm{PREF}}(e) &\stackrel{\mathrm{def}}{=} \frac{\mathcal{F}_T(e)}{t(e)} \quad \text{(Preferred Rate)} \\ \overline{\mathcal{R}}(e) &\stackrel{\mathrm{def}}{=} \frac{\mathcal{F}_D(e)}{t(e)} \quad \text{(Mean Rate)} \end{aligned}$$

That is, $\mathcal{R}_{\mathrm{PREF}}(e)$ is the digitized rate of the movie in frames per second (henceforth abbreviated as "fps"), and $\overline{\mathcal{R}}(e)$ is the effective, mean rate of the movie's playback performance in an experiment e.

But $\mathcal{F}_D(e)$ and $\overline{\mathcal{R}}(e)$ tell only one side of the story, since a test may experience a given $\mathcal{F}_D(e)$ in a variety of ways. It may be due to a nearly uniform rate throughout – for example, when every-other frame is played. Alternatively, there may be large "spikes" during which *very few* frames are displayed. A third scenario is realized when there are be repeated, radical oscillations, i.e., high-rate intervals, followed by low-rate intervals, etc.

The first case (e.g., Figure 1(1)) may yield good overall visual quality; it depends on whether a rate of 15 fps is acceptable for the video content. However, this type of behavior has one thing going for it – it is deterministic, in that playback persistently hovers around $\overline{\mathcal{R}}(e)$. As for the second

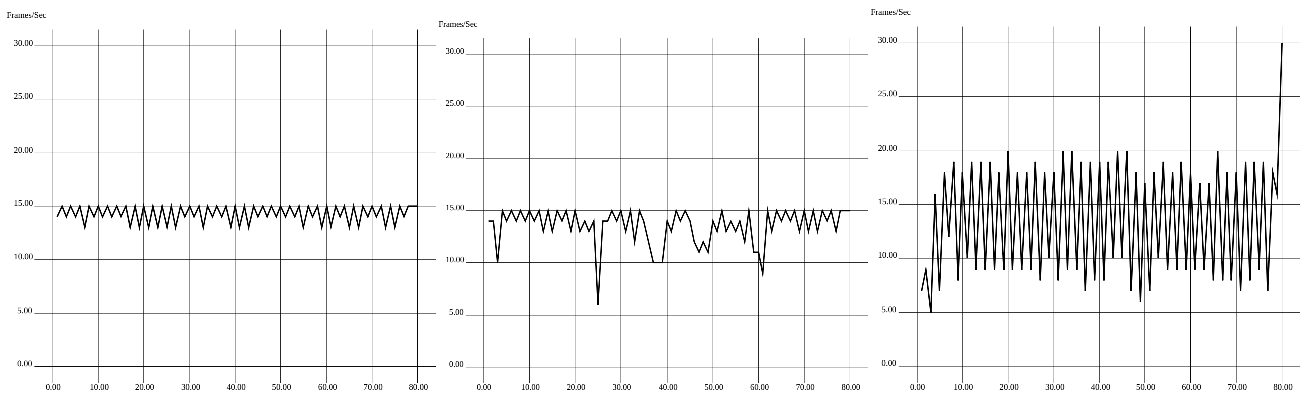

Figure 1: Playing Three Digital Versions of a Video: (1) Smooth, (2) Spikes and (3) Ocillation

case (e.g., Figure 1(2)), its four-five spikes are tolerable over the course of an eighty second video. The net esthetic effect is akin to the occasional "noticeable jitter" seen in a television set. But the third case (Figure 1(3)) is definitely observable, and the visual experience is like a watching a slide show of randomly selected, blurry pictures.

To distinguish between these very different cases, we use the frame rate's variance (per second quantum). That is, letting $\mathcal{F}_i(e)$ be the number of frames displayed during the i^{th} second of e, $\mathcal{R}_{\mu_2}(e)$ adequately serves to capture the degree of "flicker," where

$$\mathcal{R}_{\mu_2}(e) \stackrel{\text{def}}{=} \frac{\sum_{i=0}^{t(e)} (\overline{\mathcal{R}}(e) - \mathcal{F}_i(e))^2}{t(e)}$$

(2) Suitable, Comparable Platforms. We selected two Apple Macintosh computers to be our test platforms – Quadra 950's (at the low end), and Power PC 7100/80's (at the high end). (In Section 3 we detail the differing specifications of these platforms, and we identify the variables we maintained in both.)

Our decision to use Macintoshes was preceded by an exhaustive, "qualitative" investigation into the video capabilities of different contemporary, affordable workstations.[2] While there is a wide variety of workstations currently available, Apple's products still seem to deliver the best "software-codec" video.[3] This is partially due to the fact that over the years, Apple has invested heavily in optimizing the QuickTime and QuickDraw libraries.[4] But perhaps of greater importance is the fact that at the application level, Mac tasks are still largely nonpreemptive and event-driven, while multi-threading is mainly cooperative.[5] System services are still invoked via fast traps, and the

[2] We intentionally excluded specialized high-end equipment, such as SGI's graphics workstations.

[3] A software codec is a compression/decompression scheme in which video processing is performed without special hardware.

[4] QuickTime includes a set of software codecs, and a set of functions to build, store and play movies. QuickDraw is used by QuickTime, and it encompasses a set of functions which draw single frames to the screen.

[5] While a preemptive threads package exists, as of this writing it is still rather inefficient.

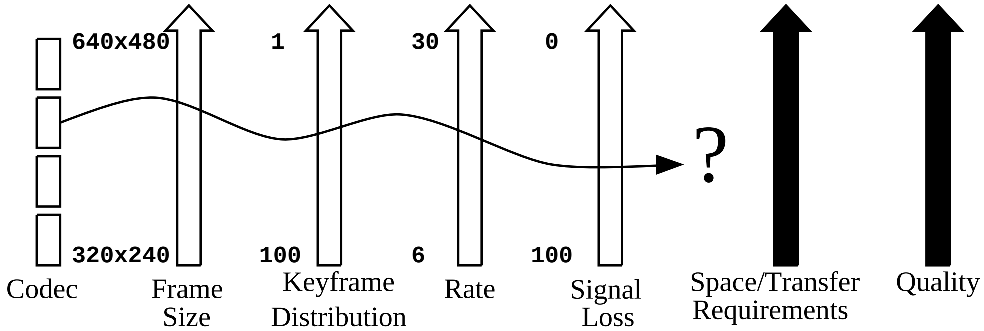

Figure 2: Instantiating the Variables

RAM-resident kernel system includes functionality usually not associated with operating systems – e.g., high-level graphics services.

While these "features" lead to a very awkward programming style (every application program is essentially a system program too), they end up providing a streamlined platform for evaluating video applications.

(3) Test Videos. We first experimented with sample videos from various sources, i.e., from Apple's developer CDs, from Internet locations, and digitized from commercial VHS tapes. None of these options were very satisfying. In the first two cases, movies were already digitized and compressed by others, according to their own needs, and all were of poor quality. Thus our re-digitized, experimental copies would have been generated from inferior originals. A similar problem held in the case of the VHS copies of commercial movies, which possessed a large amount of noise, and led to highly distorted digital copies. Most of all, we had no control over getting a sufficiently wide spectrum of changes in motion, light, sound, etc.

But there was a natural way to control all of the variables involved, which was record, digitize, edit, and assemble our own movie with our own video equipment. Aside from giving us ultimate control over content, this decision had two ancillary benefits. First, we could compare the quality of the computer-based, digital experiments with our original Hi8 clips played on a high-quality, S-video television. Second, we were forced to learn our application in its entirety. In fact we found that digital video is currently in a state similar to that of traditional real-time systems *circa* 1985. As we explain in the sequel, producing a video on a computer is still a matter of trial-and-error.

(4) Variables. Once we had our digital "master," we produced sixty different copies, with each possessing a set of uniquely instantiated variables. What are our variables? We identified five which play dominant roles in determining the quality of the final result (Figure 2): *codec type*, *frame size*, *digitized rate* (or $\mathcal{R}_{\mathrm{PREF}}$), *spatial quality* and *keyframe distribution*. The last two variables are artifacts of digital compression: spatial quality is the amount of original signal loss permitted in each compressed frame, whereas the *keyframe distribution* is the distribution of frames which are stored as compressed, still pictures – while the remainder are processed via interpolation relative to the keyframes.

All other variables, such as color depth and sound quality, were kept fixed. (Throughout we maintained an RGB color depth of 16 bits, and 16-bit stereo sound, sampled at 44kHz.)

While all of these variables have an effect on the ultimate playback quality (and potentially the transfer rates required), the actual contribution of each is not easy to determine. Picture size, for example, is immediately identifiable – and most people would agree that "bigger is better." Yet while too big a palette will lead to untenable data rates, depending on the other parameters chosen, reducing the picture size may not lead to a reduction in data size. As for frame rates, in some presentations the human eye cannot perceive the qualitative differences between playback rates of 30, 20 or 15 fps – it is usually a function of type and amount of inter-frame motion.

Moreover, these variables are not at all independent, and the interplay between them is highly nonlinear. Codec selection always has an effect on tolerable keyframe distributions, as well as on acceptable spatial quality. Yet the actual relationships are usually content-dependent, and will vary over the course of a presentation. A videographer typically juggles these issues to maintain a reasonable level of quality, while simultaneously reducing transfer rates to fall within the ability of some target playback system. But the the process of determining a good instantiation of the variables is at best highly interactive and ad hoc.

1.3 Remainder of the Paper.

Armed with our experiments and metrics, we measured the performance of all 60 variable instantiations on our two platforms. This paper contains the results of these experiments, as well our preliminary analysis of each variable's importance. This data unveiled some very surprising anomalies, which seemed to substantiate our hypotheses – even more than we expected. In fact, these anomalies forced us to run yet another 120 tests, whose examination led to some rather broad conclusions about digital video specifically, and about personal computers in general.

The remainder of this paper is organized as follows. In Section 2 we survey some of the related work in the area of digital video. Then in Section 3, we describe how we generated our test cases, and how we controlled the variables involved. In Section 4 we describe our test results, and we analyze the significance of each variable. We conclude in Section 5 with some remarks on our future work.

2 Related Work

Within computer science the of area digital video has largely been studied as a networking problem, without the "client-centric" focus which we take in this paper. Nonetheless, some recent papers on distributed systems touch on many of the same issues, albiet from a different perspective. For example, Stone and Jeffay's study of delay and jitter management [5] is highly relevant to our focus. As they note, a network's display latency effectively determines client's frame rate, and that latency is, for obvious reasons, inversely proportional to permissible display rates. As they have found in networked traffic – and as we have found in dealing with disks and compression software – a balance must be found between a stream's jitter and its delivery rate. Stone and Jeffay prescribe a *queue monitoring* policy for dyanamic adjustment of display latency, which supports low-latency conferences with acceptable gap-rates.

Our focus on IO and data paths is echoed in [2], which proposes a means of optimizing the transmission of compressed videos. While a network may be able to transmit the video frames, the destination station may end up being the bottleneck. After all, MPEG compression ratios may be as high as 200:1; thus the demand on the receiving station's operating system may be overwhelming. Display quality may decrease not as a result of network capability, but rather due to insufficient I/O throughput. In this work a *splice* mechanism is introduced, in which an application can associate a kernel-level data source with its sink point; this allows for a direct point-to-point data path between source and sink, obviating unnecessary kernel interference.

Finally, the abilities of our player-monitor are similar to that described in [6], where MPEG-encoded video streams can be generated expressly for providing statistical information on the data, such as distribution of ATM cells per frame, auto-correclation and cell inter-arrival times.

3 Experiment Construction and Variable Instantiation

In this section we provide an overview of the methods used in constructing our test videos. Some of the techniques we used are traditionally not included under the rubric of "computer science," much less "realtime systems;" rather, they are considered more relating to "video production." But as we have found, it is difficult to experiment on the "science" of media systems without having some control over video content. In this section we include some of the processes we used to generate our test cases. Our goal here is to provide sufficient detail so that others can, if they desire, repeat these experiments.

Figure 3 roughly outlines the process we used to produce our tests. Without belaboring the details, we carried out the following steps: (1) Sketching out a "story-board," in which we included scenes with different levels of light, color, motion, etc.; (2) Using a Hi8 video camera to photograph about 3-4 hours worth of video "clips," which included a superset of our "story-board;" (3) Reviewing our clips, selecting portions of interest, and then using a JPEG hardware codec to digitize them into a 7100/80 – and storing them on a 2.1G disk; (4) Using Adobe Premiere 4.0 to assemble a rough prototype of our final "cut," whose duration is about 5 minutes; (5) Employing sound-processing software to sample and filter "background music" from CDs, and inserting them in the movie at appropriate levels; (5) Using Adobe Premiere, in conjunction with our digitizer and the Mac's sound capabilities, to produce a final "mix" – which was still in a "hardware-codec" format; (6) Selecting an 80 second segment from the movie, which contained sufficient contrast to stress the capabilities of compression and playback software; (7) Using the Premiere/Digitizer combination in conjunction with the Quicktime libraries to generate 60 software-digitized copies from this 80 second segment – each representing a separate instantiation of our variables; and (8)

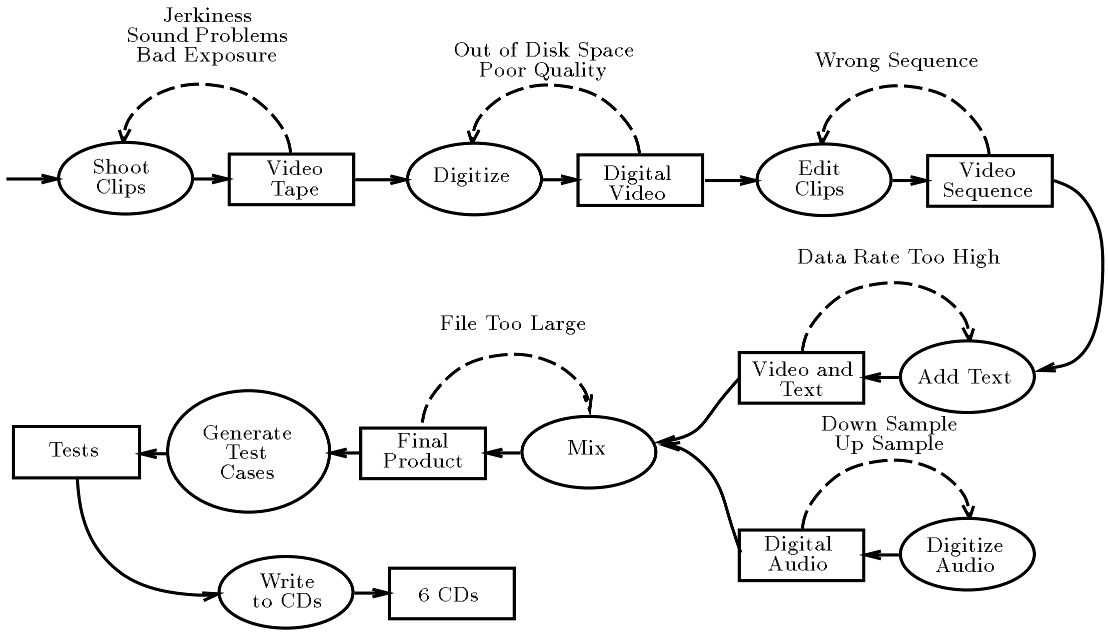

Figure 3: Experiment Production Process

Archiving our 60 files on six CDs. We were then able to run the tests and monitor their results via our TimeWare testing tool, which we built for this purpose.

As Figure 3 makes clear, this process is hardly a linear one, with "back-edges" at almost every stage. Aside from the obvious esthetic difficulties that arise in a film production, there were more serious problems associated with computing issues involved. For example, we constantly had to adjust the quality of our digitized samples to accommodate the transfer rates and capacity of our SCSI disks – while balancing these limitations against our eventual goal, i.e., to produce 60 "paradigm" test runs. Indeed, the process of striking the appropriate balance required several months of time.

3.1 Hardware and Software

Table 3.1 presents the equipment we used. While most of the specifications are self-explanatory, a few comments are relevant here.

Digitizer: The Radius VideoVision Studio (or VVS) is essentially a hardware JPEG codec capable of realtime, 30fps full-screen digitization and compression (or equivalently, de-compression on playback). While the cost of this small card exceeded that of Platform 1, the investment was necessary for producing digital "master copies" of clean Hi8 originals. The advantages of using a good hardware codec are obvious: one need only let the tape roll, and the compression logic is capable of handling every frame in real-time. In fact, the VVS is capable of digitizing frame sizes 640x480 with little signal loss, and no "drop-out" frames. Since JPEG compression basically involves running a Huffman filter over frames mapped onto an 8x8 grid, the VVS can process the grid's 64 squares in parallel.

Computing Equipment	
Test Platform 1: PPC 7100/80	Test Platform 2: Quadra 950
CPU: MC601 80MHz, Floating Point	CPU: MC68040 33 MHz, Floating Point
Memory: 24Mb	Memory: 16Mb
Internal Hard Drive: 696 Mbyte, IBM DSAS3720	Internal Hard Drive: 234Mb, Quantum LP2435
CD: Apple 300i (2-speed, Internal)	CD: Apple 300i (2-speed, External)
OS: System 7.5, Minimal Extensions	OS: System 7.5, Minimal Extensions
Disk Drivers: From FWB	Disk Drivers: From FWB

Essential Peripherals	Video Equipment
External Hard Drive: 2.1Gbyte, Seagate ST12400N	Cameras: Sony CCD TR700s Hi8 Format
Digitizer: Radius VideoVision Studio 2.0	Time Code, HiFi Stereo, S-video In/Out
CD Writer: Pinnacle Micro RCD 1000	VCR: Sony EVS7000 Hi8 Format
Tape Drive: Exabyte 2000	Time Code, Single-Frame Advance/Retract
	Edit Marks, PCM Audio Dubbing, S-video

Software
Metrowerks Gold C/C++ Compiler (PPC/68K Compiler/Debugger)
Adobe Premiere, Adobe Photoshop (Video and Photo Editing Tools)
Macromedia SoundEdit 16 (Sound Sampler and Filtering Tools)
Assorted Quicktime Tools (For “massaging” Quicktime Movies)
TimeWare Movie Monitor (Our monitoring program to test playback)

Table 1: Hardware and Software Used to Generate and Run Experiments

All of this stated, we were unable to use this equipment at its peak potential – our 2.1Gbyte disk was not fast enough to keep up with the required transfer rates, nor was it sufficiently large to store more than a single minute of footage. Even after being processed by the compression logic, some scenes still produced blistering data rates of 20 Mbyte/sec, which would quickly fill up any large disk.

After determining the ability of the system via trial-and-error, we found a reasonable compromise between quality and capacity: digitizing at 30fps, but with a frame size of 320x240, and with a small amount of signal loss. For the sake of our experiments we considered this to be our “baseline,” ranking it at “100% quality.” All subsequent software-compressed copies were generated from the VVS-digitized “master” at this baseline quality.

The outstanding performance of the hardware codec raises the following question: if silicon can do the job so well, why should we bother measuring software-codec playback performance? One answer is obvious: most users will not invest in a digitizer that costs more than the computer itself. But another reason stems from the limitations inherent in the high-priced, real-time JPEG cards. That is, if one is purely interested in transmitting and viewing videos, then JPEG is perhaps the worst compression format to use. JPEG cards do not perform inter-frame compression, and they produce significantlly more data than most systems can accomodate. On the other hand, full-field, full-color, realtime MPEG (and Motion-JPEG) compression cards are still very specialized peripherals, priced well over $100,000.

But there is even a better reason why we should be interested in purely software schemes. As we show in the sequel, when the system software is tuned appropriately, a good software-codec is more than capable of delivering high-quality video. Conversely, if the operating system imposes high latencies on large IO transfers, even the fanciest hardware codec will probably fail to live up to its rated potential.

Hard Disks: Our "main" disk drive was a 2.1 Gbyte Seagate "Barracuda," which we used for digitizing our clips, as well as in monitoring the playback quality of each test run. Using a variety of commercial benchmarking tools, we measured the Barracuda's normal transfer rates as follows:

Read Transfers:	2790 bytes/sec
Write Transfers:	3100 bytes/sec

All of the disk drivers were installed using FWB's Hard Disk Toolset, and we also used FWB's utilities for formatting and partitioning. In particular, we disabled re-mapping bad sectors to the end of the disk; rather, we configured the drivers to simply skip them. This minimized the amount of head movement in both sampling and playback.

3.2 Video Content: February 14, 1995

The video clips were photographed during the week of 2/14/95 at the University of Maryland. They captured "typical" scenes of the university and the Computer Science department, with students and faculty in relatively informal settings. The objective was to procure a sufficiently wide spectrum of changes in light, color and motion within – and between – the scenes. To this extent, about 50% of our test video consists of brightly-lit, exterior scenes, with the remainder shot indoors. The longest scene has a duration of 13 seconds, and is mostly static, whereas there is 15 second "collage" of active short scenes, each of which consumes roughly 1 second of time. About 15% of the movie contains dialague set over background music (to check synchronization), while the remaining "soundtrack" contains only music.

The digitized clips were edited using Adobe Premiere 4.0, with which we also inserted title sequences. The final "cut" is a 5-minute video occupying about 270 Mbytes of space, with $\mathcal{R}_{\mathrm{PREF}}$ set to 30 fps. We copied an 80-second portion of this "final cut" into a separate "experiment" file, which then formed the "master" for each test case.

3.3 Variables

Next we produced 60 test files, each of which containing an instantiation of our variables: *codec type*, *frame size*, *digitized rate* (or $\mathcal{R}_{\mathrm{PREF}}$), *spatial quality* and *keyframe distribution*. Other variables that could have been altered – but were not – were color depth and sound quality. Across all experiments they remained fixed, respectively at 16-bit color, and 16-bit stereo sampled at 44kHz.

Formally, the variable space ranges over

$$Codec \times Rate \times KFD \times Size \times Quality$$

where

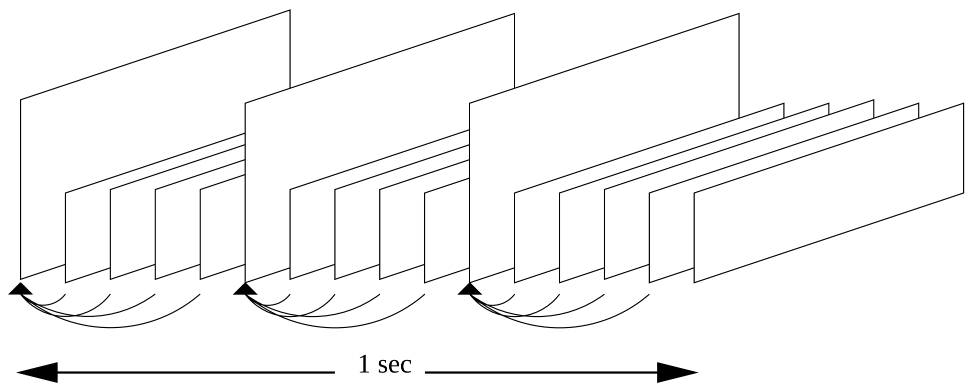

Figure 4: One-Second Strip of frames, with $Rate = 15$ and $KFD = 3$

1. $Codec \in \{\text{C}, \text{V}, \text{J}\}$ denotes the compression scheme used. Here "C" is Radius's Cinepak codec, "V" is Apple's "Default Video" Motion-JPEG codec, and "J" stands for QuickTime's "Photo," frame-by-frame, still-JPEG codec.

2. $Rate \in \{15, 30\}$ denotes preferred playback rate, or $\mathcal{R}_{\text{PREF}}$.

3. $KFD \in \{1, 3, 5, 10\}$ denotes the keyframe distribution used. For example, if $KFD = 5$ this means that every fifth frame is a keyframe. (Note that when the "J" codec is used, KFD is always set to 1, since all frames in a JPEG movie can be considered keyframes.)

4. $Size \in \{\text{half}, \text{quat}\}$ denotes the frame size, where "half" is 320x160 pixels, and where "quat" is 160x80.

5. $Quality \in \{75, 100\}$ denotes the degree of spatial quality that is maintained in the re-digitized test file. When $Quality = 75$, this implies that the codec attempts to keep 75% of the original quality, and $Quality = 100$ means that the codec is used at its "best possible" setting.

Our test instances have labels like "C/30/3/half/75," which denotes a movie re-digitized in the Cinepak codec at 30 fps, with one keyframe every third frame, in a frame size of 320x160 with a 75% quality index.

Codecs and Keyframes. JPEG [3] is basically a compression standard for still-pictures, which can produce nearly lossless digital copies. However, it turned out to be a poor performer at playback time, and we used it as our "high watermark" for image quality, while simultaneously as a "low watermark" for motion quality. (Note that the VVS has no problem handling realtime de-compression at 30 fps, but this is difficult to achieve in software.)

The other two techniques, "Video" and "Cinepak," fall into the category of *motion*-JPEG. Whereas still-JPEG involves running a Huffman filter over each frame individually, motion-JPEG also employ inter-frame (or "temporal") compression. In this scheme keyframes are compressed as still pictures, while the other frames are interpolated relative to their preceeding keyframes (as in Figure 4).

QuickTime does this in a rather crude (but effective) manner: when temporally compressing a frame k relative to its keyframe i, k's pixel-map is simply subtracted from that of i, and the result is compressed. Note that this is analogous (but not identical) to P-frames in MPEG [4], which determines the fields of motion relative to last I-frame, and then stores the corresponding motion-vectors.

While MPEG-compression can be a complex (and time-consuming) operation, re-compressing QuickTime movies in software can also require an enormous amount of CPU time. In particular, each re-compression of our 80-second test in Cinepak required about two hours.

3.4 Running the Tests

Our TimeWare monitor tool is an efficient QuickTime player, which interacts with QuickDraw to determine which frames are displayed, and which are dropped. It keeps this frame-by-frame information in a running bit-vector associated with each movie; after testing the movie, the bit-vector is used to generate a set of playback-performance measures (such as $\mathcal{F}_D$, $\mathcal{R}_{\mu_2}$, $\overline{\mathcal{R}}$, etc.).

Although the Movie Monitor is efficient – and the code requires a minuscule amount of memory – whenever it executes it turns into a memory hog. In fact, it grabs all of the memory currently available (usually about 20Mbytes) on our systems. This is for a good reason: the greater its memory partition, the more buffers can be afforded to the QuickTime decompression software. We set out to test the capabilities of video playback performance, not the speed of garbage collection.

We also eliminated the effects of disk fragmentation. Whenever we ran a test sequence, we first reformatted out disk drive, then we copied a set of our test CDs onto the disk, after which we again re-optimized the entire disk (which eliminated any internal fragmentation). We also used the same physical disk on both of our test machines, hand-carrying it back and forth between them.

Finally, every test was run at least five times; surprisingly, our metrics $\mathcal{F}_D$ and $\mathcal{R}_{\mu_2}$ showed very low deviation across the trials. We attribute this to the tight interaction between Mac application programs and the operating system; i.e., the system rarely "steals" CPU cycles from an application without the application yielding time.

4 Test Results and Analysis

The results of our initial half-screen tests are listed in Figure 5; the remaining test results are in the Appendix. To better understand the "highlights" of the half-screen tests, we have extracted the $\mathcal{F}_D$ data onto the bar charts in Figures 6-7.

Platform and Codec Variables. One need not delve into the numbers to make some course observations, which stand out in the bar charts. First, excluding a few notable exceptions (to which we return shortly), the PPC tests show superior results to those run on the Quadra. Another fairly

Test Movie	Size			PPC			Quadra		
Codec/FPS/KFD/Size/Quality	Total	Video	$\mathcal{F}_T$	$\mathcal{F}_D$	$\overline{\mathcal{R}}$	$\mathcal{R}_{\mu_2}$	$\mathcal{F}_D$	$\overline{\mathcal{R}}$	$\mathcal{R}_{\mu_2}$
C 30 1 half 100	65668948	51556948	2400	1837	22.96	5.30	1090	13.62	29.56
C 30 3 half 100	55288196	41176196	2400	1884	23.55	5.17	1323	16.54	31.15
C 30 5 half 100	53200812	39088812	2400	1882	23.52	7.55	1341	16.76	28.59
C 30 10 half 100	51599160	37487160	2400	1893	23.66	8.63	1315	16.44	29.67
C 30 1 half 75	56272532	42160532	2400	1868	23.35	6.65	1066	13.32	33.66
C 30 3 half 75	48114560	34002560	2400	1978	24.73	5.25	1461	18.26	25.09
C 30 5 half 75	46467008	32355008	2400	2015	25.19	5.49	1557	19.46	24.22
C 30 10 half 75	45253496	31141496	2400	2038	25.48	5.70	1607	20.09	24.17
V 30 1 half 100	138782648	124670648	2400	528	6.60	17.97	271	3.39	1.91
V 30 3 half 100	126319334	112207334	2400	809	10.11	39.94	255	3.19	1.74
V 30 5 half 100	123699495	109587495	2400	847	10.59	42.59	267	3.34	2.24
V 30 10 half 100	121666111	107554111	2400	802	10.03	44.64	218	2.73	1.05
V 30 1 half 75	110761808	96649808	2400	994	12.43	29.94	336	4.20	1.16
V 30 3 half 75	89538819	75426819	2400	1446	18.07	25.38	388	4.85	7.66
V 30 5 half 75	85251859	71139859	2400	1430	17.88	27.29	471	5.89	22.98
V 30 10 half 75	81892471	67780471	2400	1551	19.39	24.11	507	6.34	39.47
J 30 1 half 100	94238136	80126136	2400	313	3.91	0.29	112	1.40	0.24
J 30 1 half 75	76101834	61989834	2400	325	4.06	0.28	124	1.55	0.25
C 15 1 half 100	39892376	25780376	1200	1129	14.11	1.04	1113	13.91	1.99
C 15 3 half 100	34932804	20820804	1200	1148	14.35	0.57	1117	13.96	1.60
C 15 5 half 100	33894160	19782160	1200	1140	14.25	0.76	1117	13.96	1.60
C 15 1 half 75	35187128	21075128	1200	1139	14.24	0.75	1117	13.96	1.60
C 15 3 half 75	31257064	17145064	1200	1142	14.28	0.69	1116	13.95	1.78
C 15 5 half 75	30448024	16336024	1200	1142	14.28	0.69	1116	13.95	1.78
V 15 1 half 100	76406147	62294147	1200	1083	13.54	2.92	593	7.41	8.49
V 15 3 half 100	71468196	57356196	1200	1102	13.78	1.98	724	9.05	9.19
V 15 5 half 100	70419248	56307248	1200	1097	13.71	1.81	747	9.34	9.44
V 15 1 half 75	62437907	48325907	1200	1144	14.30	0.71	692	8.65	7.08
V 15 3 half 75	53301189	39189189	1200	1129	14.11	1.49	894	11.18	8.36
V 15 5 half 75	51404010	37292010	1200	1138	14.22	0.82	945	11.81	7.66
J 15 1 half 100	109158425	95046425	1200	209	2.61	0.41	63	0.79	0.17
J 15 1 half 75	45082267	30970267	1200	330	4.12	0.76	143	1.79	0.17

Figure 5: PPC 7100/80 and Quadra 950/33, 320 $\times$ 240 – Played off Disk

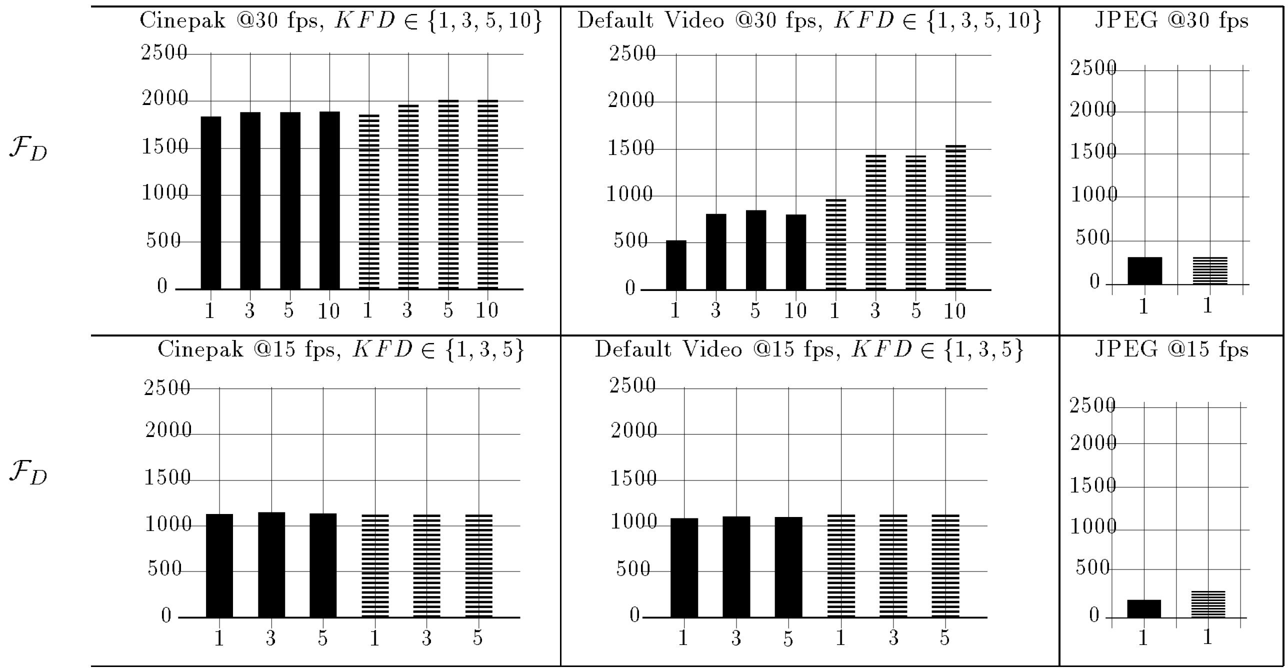

Figure 6: $\mathcal{F}_D$ of All Half-Screen Runs Played on PPC 7100/80 off of Disk. Keyframes are Numbered 1-10. Solid Bars Denote $Quality = 100$, Striped Bars Denote $Quality = 75$

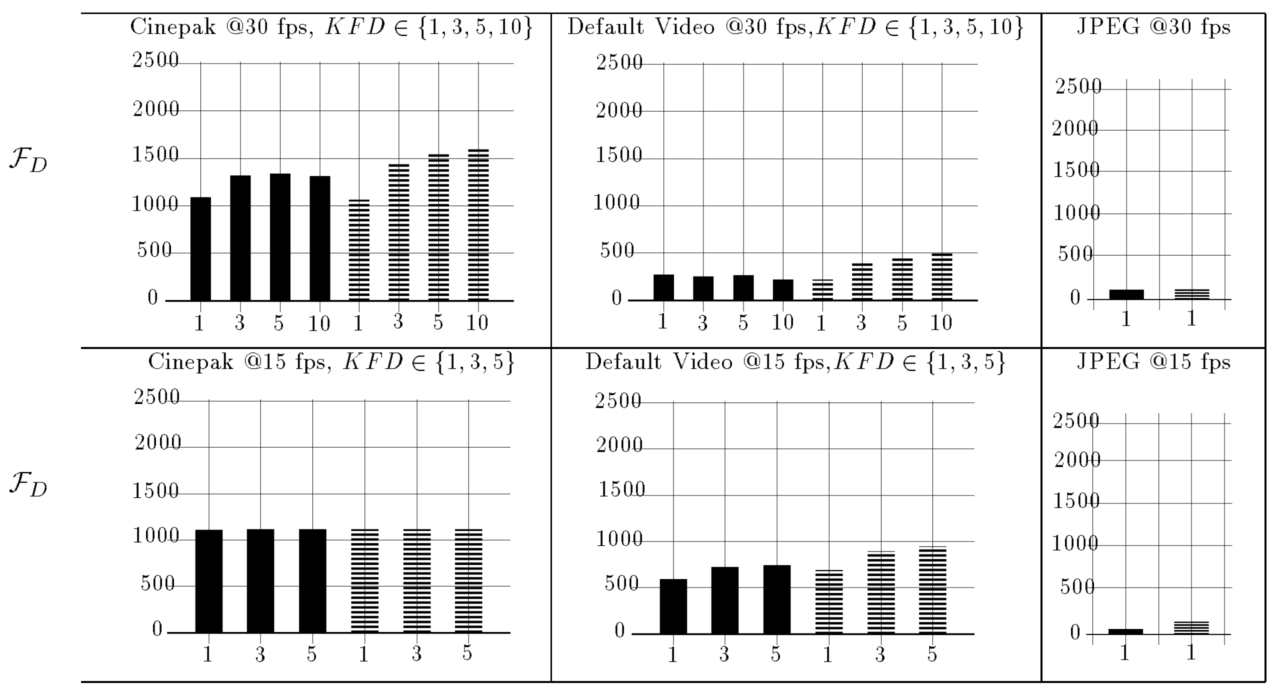

Figure 7: $\mathcal{F}_D$ of All Half-Screen Runs Played on Quadra 950/33 off of Disk. Keyframes are Numbered 1-10. Solid Bars Denote $Quality = 100$, Striped Bars Denote $Quality = 75$

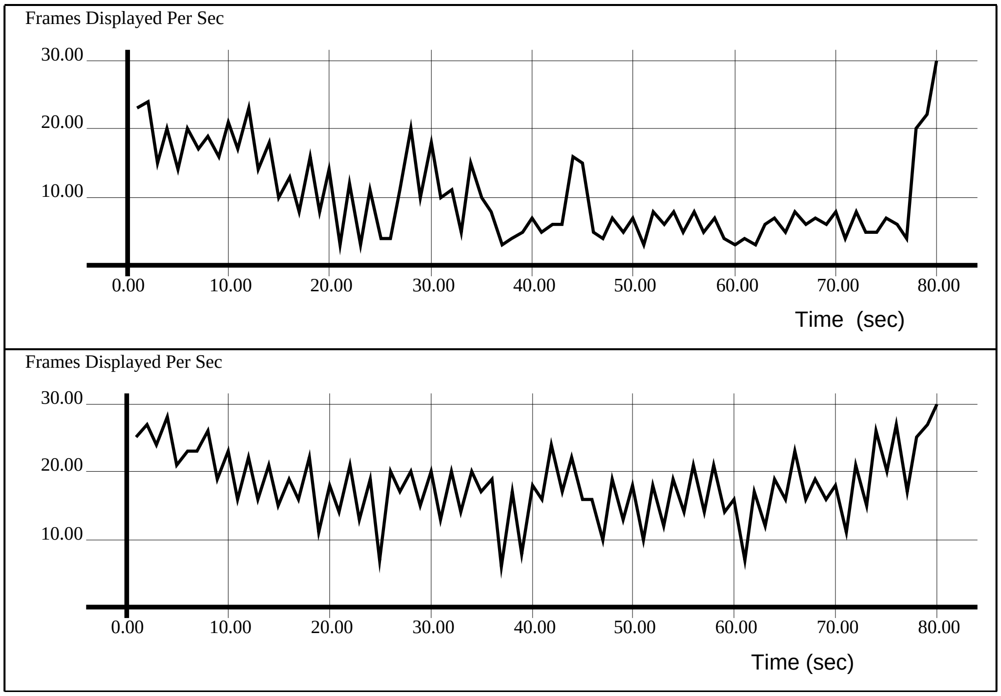

Figure 8: V/30/3/half/100 (above) and V/30/1/half/75 (below)

obvious observation is that JPEG is a very poor performer, while Cinepak shows the best average performance in most test cases. Yet we found that when Default Video and Cinepak have similar performance (as, for example, on the PPC at 15 fps), the Video tests are esthetically superior. In certain scenes, Cinepak's signal loss tends to result in a "grid pattern" appearing over uniform color fields, no doubt an artifact of the JPEG-style YUV compression. A similar, though less annoying phenomenon, emerges in Default Video compression, which tends at times to result in "blurry" edges between differing fields of intensity. However an esthetic judgement like this is really a matter of personal taste.

Compression Variables. As for the effects of compression, it appears that reducing spatial quality hardly leads to uniform benefits across the tests; moreover, the results of temporal compression are highly unpredictable. But there are two groups of trials on which these factors seem to have a more pronounced impact: the PPC/Video tests digitized at 30 fps, and the Quadra/Cinepak tests digitized at 15 fps. Figure 8 shows two of the Default Video results from the PPC, in which the only difference is spatial quality. Note that while both runs show a significant amount of jitter, a reduction to 75% quality performs significantly better. This is true across all of our statistics – $\mathcal{F}_D$ (809 vs. 1446 frames displayed), the proportional $\overline{\mathcal{R}}$ (10.11 vs. 18.07), and $\mathcal{R}_{\mu_2}$ (39.94 vs. 25.38). Neither variance is stellar (one need only examine the graphs to see the reason); yet the playback quality is noticeably superior in the 75% run. Similar statements can be made when comparing the role of KFD in the Cinepak/Quadra/30fps runs.

The conclusion we reached is as follows: that when $\overline{\mathcal{R}}$ was over $(\frac{3}{4})\mathcal{R}_{\mathrm{PREF}}$, additional spatial

$\mathcal{F}_D$

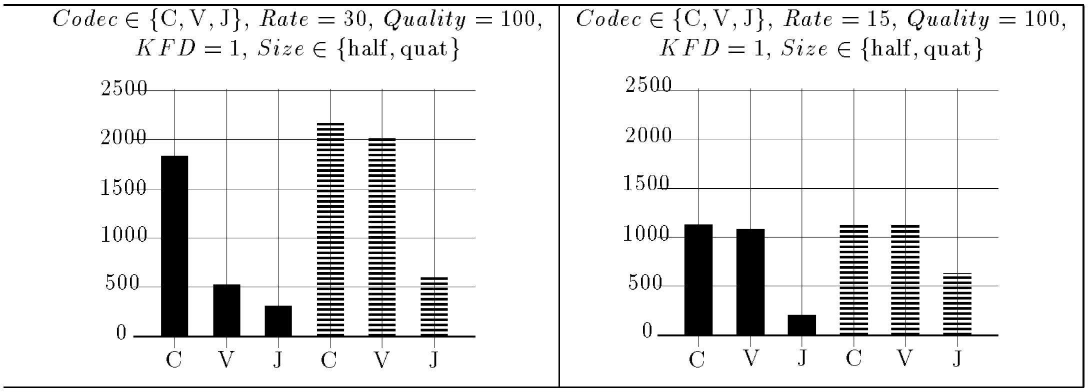

Figure 9: $\mathcal{F}_D$ High-Quality PPC 7100/80 Runs, Where Screen Size is Altered. Solid Bars Denote $Size$ = half, Striped Bars Denote $Size$ = quat

and temporal compression failed to raise it (but only realized a degradation of quality). This is a bit surprising: comparing the size of the "C/30/1/half/100" video to that of "C/30/10/half/75," we see a reduction from 65 Mbytes to 45 Mbytes, i.e., the transfer-rate is reduced by over 30%.

The other observation is that there are many instances where even with a large amount of spatial and temporal compression, abysmal performance remains abysmal. For example, comparing the Quadra's treatment of "V/15/1/half/100" to that of "V/15/5/half/75" – a whopping reduction in size from 139 Mbytes to 82 Mbytes – we end up playing 353 more frames, which still fails to achieve decent visual quality. This is also evident when comparing the 25% quality reduction from "J/15/1/half/100" to "J/15/1/half/75," which reduces the file size from 109 Mbytes to 45 Mbytes! The performance, however, still does not rise about 4.12 fps (on the PPC) or 1.79 fps (on the Quadra).

The Effect of Frame Size. Since a 160x120 field contains 1/4 the number of pixels of a 320x240 field, one would expect the "quat" video track sizes (and transfer rates) to be much lower than the corresponding "half" track sizes. And examining the data in Figures 5-12, one sees that this is true: going from "C/30/1/half/100" to "C/30/1/quat/100" we get a video track reduction from 51 Mbytes to 15 Mbytes; from "V/30/1/half/100" to "V/30/1/quat/100" we get a reduction from 124 Mbytes to 32 Mbytes; and "J/30/1/half/100" reduces from 95 Mbytes to 32 Mbytes in "J/30/1/quat/100." (Note that the sound track remains a constant 14 Mbytes.)

But watching a video on a quarter-screen palette is hardly satisfying; thus the reduced size should, one hopes, pay off in superior playback quality. Does it?

Figure 9 compares the effects of frame size on selected 30fps and 15fps trials. In the case of Default Video at 30fps, the answer is a resounding "yes." In fact, the effective playback rate $\overline{\mathcal{R}}$ increases four-fold from 6.6 to 25.36 – echoing the 75% reduction in transfer rate. As for the other samples, the result is less clear. JPEG's performance increases substantially, but still, it is hardly acceptable. And while Cinepak's rate improves from 22.96 fps to 27.32 fps, most people would prefer the larger frame size to a relatively small increase in the display rate.

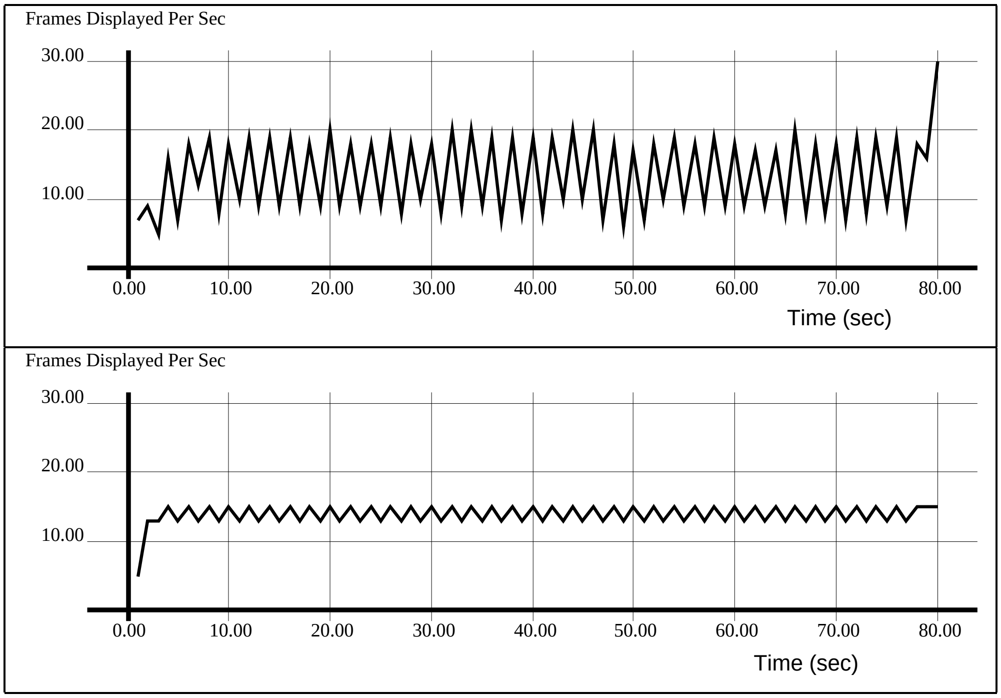

Figure 10: C/30/1/half/100 (above) and C/15/1/half/100 (below)

As for the 15fps runs, the Default Video and Cinepak codecs show only minimal improvement; e.g., the significant frame-size reduction buys Cinepak a rate increase of 0.18 fps, from 14.11 to 14.29. Clearly for the this codec on the PPC at 15 fps, substantially lowering the data-rate abuts against the law of diminishing returns.

But this raises an interesting, perplexing question: Exactly what does it take to achieve the full 15fps playback rate? We return to this question shortly.

The Effects of Rate Changes. Considering the *Rate* variable, the Default Video statistics in Figures 6-7 show situations where 15 fps tests realize superior playback performance to their corresponding 30 fps versions. This is true in all of the Video test results on the Quadra, and strikingly so in the 100%-quality runs on the PPC. Even though this is a very fast CPU, with a sufficiently good disk, the Quicktime de-compression logic still seems to "thrash" on the 30 fps version. It does better with more deterministic (albeit lower) playback rates.

A similar situation arises when a reduction in $\mathcal{R}_{\mathrm{PREF}}$ leaves $\overline{\mathcal{R}}$ almost unchanged, but radically enhances the movie's visual quality. In Figure 10, we compare the results of two Quadra/Cinepak playback runs, "C/30/1/half/100" (with $\overline{\mathcal{R}} = 13.62$) and "C/15/1/half/100" (with $\overline{\mathcal{R}} = 13.91$). Figure 10 shows the on-line behavior of the two; this illustrates why we also use variances to compare two runs with similar average behavior. Visually, the 15 fps version looks smooth and continuous, while the 30 fps playback is jerky (as its graph portrays).

But what is really causing the oscillation in 30fps trial? There are three potential bottlenecks: (1) the QuickDraw screen-manager, (2) the de-compressor, or (3) the IO channel between the hard drive and the codec software.

Case (1) would imply that the de-compressor works faster than the screen manager. Since our player/driver streams the codec's output directly to QuickDraw (for screen updates), perhaps more frames get decompressed than can be drawn. This would have the following result: a bursty period (i.e., a crest) occurs when the de-compressor's input buffer is full of compressed frames. Each frame is then quickly converted, and then streamed to the screen handler. But if QuickDraw cannot keep up with the update rate, since the real-time clock continues to advance, the the de-compressor is periodically forced to purge its buffers – and then request additional chunks of frames from the file system. The resulting latency causes the wave's trough, or low watermark.

But since decompression is a compute-intesive activity, Case (2) is more convincing – that the de-compressor is slower than both QuickDraw and the IO. In this case, perhaps the Cinepak codec attempts to consume frames at a rate of 30fps, while it can only decompress them at a rate of, say 20fps. The result is a repetitive purge of the input queue, followed by a (belated) request to transfer a new set frames from disk. This can easily explain the wave's "trough."

If the bottleneck is Case (3) – i.e., slow IO transfers – we could have the following scenario: That QuickTime and QuickDraw are sufficiently fast to keep up with the display-rate demands; that instead the problem lies in unduly long latencies associated with large IO transfers. This can easily account for the regular oscillation in display rates. That is, whenever the codec's input buffer contains compressed frames, they can quickly be de-compressed and drawn to the screen. On the other hand, IO transfers will result in prolonged periods of the buffer being empty.

The 15fps test demands only one-half of the display rate, and 3/5 of the data-transfer rate (see Figure 5. So this test puts less pressure on all of the three potential bottlenecks – which explains the 15fps's low-amplitude waveform (in Figure 10).

But even in the 30fps case, the bottlenecks could be reduced if there were better, more controlled coordination between the three components involved. That is, if the platform is only capable of playing a 30fps movie at 20fps, then all of these components should be "tuned" to work at a nearly constant, deterministic rate of 20 fps.

Before we investigate this issue, we first must discover the true bottleneck in the process. In doing so, we find some statistics that really seem to stand out.

The RAM Tests. When we attempted to determine the highest performance capabilities of our two platforms, our measurements presented us with some surprising anomalies. Consider, for example, the half-screen/PPC/Cinepak runs listed in Figure 5. Note that all of the 30fps tests display at least 1800 out of 2400 total frames in the 80 second segment. *Why, then, is it the case that not a single 15fps test displays all (or almost all) of its 1200 frames?* Since the codec "proves" that it can run at a rate of 22fps when "asked" to run at 30fps, one would expect it to be capable of running at 15fps "upon request."

This phenomenon is somewhat analogous to the opposite one – i.e., the cases in which a movie plays better at 15fps than it does at 30fps. In fact, both sets of data focused our attention on Case (3) above – that high IO latencies "rob" the decompressor of useful CPU time. Moreover, since this

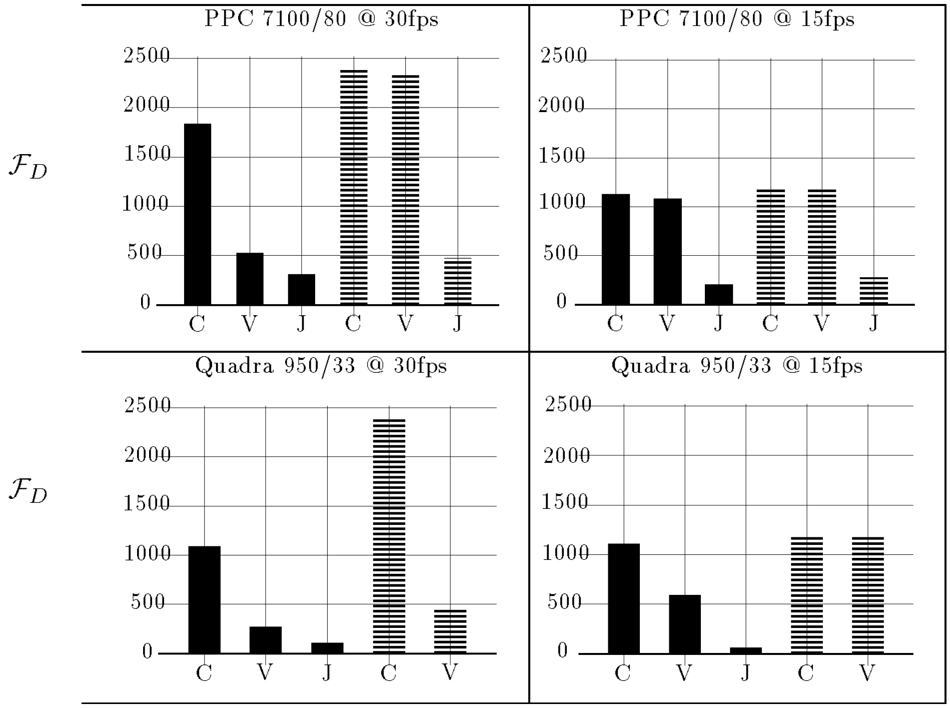

Figure 11: $320 * 240$, Highest temporal and spatial compression, Disk vs. RAM playback. Results from the disk runs are portrayed with solid bars, whereas the RAM runs are in striped bars.

occurs in such a periodic, predictable fashion (as in Figure 10), we postulate that this overhead is due mainly to a large amount of synchrony in IO setups.

We were not able to test this specific issue, since it basically involves reverse-engineering the Quicktime library. However it was easy to test our main hypothesis – that the bottleneck lies in the file system/IO subsystem. We simply extended the TimeWare Movie Monitor to play our test movies in eight 10 second segments – with each segment first prefetched into RAM, and then played out of RAM. During an IO transfer (which is typically quite brief, but noticeable), the monitor stops its movie-progress clock, and then resets it after it the transfer is done. The net result is a tool which measures the performance of the CPU and the display drivers, but *not* the IO subsystem.

Figure 4, and the figures in the Appendix, illustrate the differences between the disk-based tests, and those played out of RAM. The contrast is striking: the PPC Cinepak and Video versions play almost every single frame, when run out of RAM. This is true for *all* of the runs. Most incredibly, the Default Video test climbs from $\overline{\mathcal{R}} = 6.60$ to $\overline{\mathcal{R}} = 29.48$.

With one set of notable exceptions, the Quadra's performance improvement is not as uniform. The exceptions are all of the Cinepak 30 fps runs, which climb to nearly perfect display rates like 29.98, 30.00, 29.85, etc. In fact, the Cinepak tests do marginally better on the Quadra than on the PPC (but the differences are minute, since both achieve near-perfection).

As for the Default Video tests, the performance increase is marked, but not as much as on the PPC. This leads us to conclude that the decompression algorithm is sufficiently compute-intensive to stress the 68040/33, while the 7100 can handle the half-screen decompression easily.

But after all, it should not be a surprise that the disk is a bottleneck. "Common wisdom" dictates that the hard drive will always be the culprit, since (1) it is a physical device, limited by seek times, rotational delays, etc., and (2) it is connected via a SCSI bus, not the local bus.

But in this case the numbers do not lead to such a conclusion. The PPC hardware, for example, affords the ability to perform asynchronous, DMA'd block transfers. As we mentioned in Section 3, our disk is capable of handling read transfers at 2.7 Mbytes/sec. Since the largest file is "only" 138 Mbytes, the maximum demand from *any* of our tests is 1.7 Mbytes/sec – a large number, to be sure, but not one that will choke the system. Moreover, a data rate of 1.7 Mbytes/sec is not high enough for the DMA's memory "cycle-stealing" to have a large impact. Accounting for IO overhead, perhaps the Disk-based, Cinepak/PPC tests should run at 27 fps, but not 23 fps.

So the conclusion seems obvious: that disk transfers incur a large latency, and that IO – even "asynchronous," DMA'd, IO – is not sufficiently asynchronous for the QuickTime codecs to run at their peak rates. While the Mac arguably possesses the best consumer-grade video software, it could be significantly improved by achieving efficient coordination between the file system and the codec software. Of course, this would be a nontrivial task, since it involves streaming DMA'd IO directly into user space, while simultaneously providing "regular" IO services for other processes.

On the bright side, there is good news: that the PPC is capable of displaying high-quality video using a software-only solution. Indeed, full-screen, software decompression should be achievable within the near future – that is, if IO operations can be coordinated with the required display rates.

5 Conclusion

We conclude by re-examining our initial hypotheses.

1. A faster, more powerful platform will not necessarily lead to better playback quality.

In general, the PPC performed better the Quadra, which was not a surprising result. However in the RAM-based, 30fps Cinepak tests, the Quadra outperformed the PPC. And as we showed in our last series of tests, hardware was not really the problem in video playback – it was the operating system.

2. A movie's size (in bytes), and its playback rate (in frames per second) are not always good predictors of quality.

This phenomenon manifested itself many times – especially in the cases where the 15fps tests outperformed their corresponding 30fps versions.

3. Movie-playing software could give better performance if it (1) took into account the platform's capabilities, and (2) provided deterministic rate control based on these characteristics.

This was the point of the paper, and it raises the subject of our future plans in the area.

We have now located what we believe is the most significant bottleneck facing video-playback software. We are currently working to eliminating this problem, by implementing a direct, doublebuffered channel between the disk driver and the decompression software. A direct, DMA'd "data pipe" will, for example, obviate the need for redundant copy-in/copy-out operations. (Fall and Pasqule came to the same conclusion in [2].) Needless to say, a "Quicktime-aware" driver will have to be safe with respect other processes in the system. With this enhancement, and with a wider IO path, we are confident that we will have a software-only solution for displaying high-quality, full-screen videos

References

[1] Apple Computer Corporation. *Inside the Machintosh: Quicktime*. Addison Wesley, 1994.

[2] Kevin Fall and Joseph Pasquale. Improving Continuous-Media Playback Performance with In-Kernel Data Paths. *Proceedings of IEEE*, pages 100–109, 1994.

[3] Gregory K.Wallace. The jpeg still picture compression standard. *Communications of the ACM*, 34(4), 4 1991.

[4] Ralf Steinmetz. Compression Techniques in Multimedia Systems. Technical Report 43.9307, IBM European Networking Center, Vangerowstrabe 18, 69020 Heidelberg, Germany, 1993.

[5] Donald L. Stone and Kevin Jeffay. An Empirical Study of Delay Jitter Management Policies. *Multimedia Systems*, 2(6):267–279, 1995.

[6] Toshiyuki Urabe, Hassan Afzal, Grace Ho, and Pramod Pancha Magda El Zarki. MPEGTool: An X Window-Based MPEG Encoder and Statistics Tool. *Multimedia Systems*, 1(6):220–229, 1994.

A Additional Test Results

Test Movie	Size			PPC			Quadra		
Codec/FPS/KFD/Size/Quality	Total	Video	$\mathcal{F}_T$	$\mathcal{F}_D$	$\overline{\mathcal{R}}$	$\mathcal{R}_{\mu_2}$	$\mathcal{F}_D$	$\overline{\mathcal{R}}$	$\mathcal{R}_{\mu_2}$
C 30 1 quat 100	29766828	15654828	2400	2186	27.32	2.94	2157	26.96	8.37
C 30 3 quat 100	27746068	13634068	2400	2166	27.07	4.43	2149	26.86	9.03
C 30 5 quat 100	27310408	13198408	2400	2189	27.36	3.55	2192	27.40	7.84
C 30 10 quat 100	26985084	12873084	2400	2194	27.43	3.37	2173	27.16	6.88
C 30 1 quat 75	29449976	15337976	2400	2185	27.31	3.76	2176	27.20	8.15
C 30 3 quat 75	26442288	12330288	2400	2190	27.38	3.46	2171	27.14	8.68
C 30 5 quat 75	25860228	11748228	2400	2193	27.41	3.02	2185	27.31	8.44
C 30 10 quat 75	25421516	11309516	2400	2193	27.41	3.17	2166	27.07	6.36
pV 30 1 quat 100	46998831	32886831	2400	2029	25.36	5.93	1903	23.79	8.92
V 30 3 quat 100	42843008	28731008	2400	2084	26.05	4.94	1968	24.60	11.82
V 30 10 quat 100	41322935	27210935	2400	2107	26.34	5.19	2037	25.46	9.22
V 30 1 quat 75	38400471	24288471	2400	2129	26.61	3.64	2023	25.29	8.52
V 30 3 quat 75	32259404	18147404	2400	2155	26.94	4.04	2103	26.29	9.60
V 30 10 quat 75	30045289	15933289	2400	2174	27.18	3.31	2144	26.80	7.69
J 30 1 quat 100	78918203	64806203	2400	616	7.70	1.01	177	2.21	0.37
J 30 1 quat 75	36138726	22026726	2400	1006	12.57	3.27	444	5.55	1.12
C 15 1 quat 100	21939432	7827432	1200	1143	14.29	0.67	1124	14.05	1.24
C 15 3 quat 100	21037560	6925560	1200	1138	14.22	0.77	1122	14.03	1.39
C 15 5 quat 100	20823272	6711272	1200	1136	14.20	0.88	1119	13.99	1.42
C 15 1 quat 75	21780904	7668904	1200	1146	14.32	0.61	1120	14.00	1.26
C 15 3 quat 75	20319956	6207956	1200	1144	14.30	0.66	1122	14.03	1.39
C 15 5 quat 75	20029612	5917612	1200	1143	14.29	0.67	1119	13.99	1.42
V 15 1 quat 100	30546740	16434740	1200	1146	14.32	0.61	1149	14.36	0.80
V 15 3 quat 100	28887319	14775319	1200	1141	14.26	0.71	1124	14.05	1.24
V 15 1 quat 75	26254172	12142172	1200	1140	14.25	0.73	1141	14.26	0.96
V 15 3 quat 75	23659209	9547209	1200	1143	14.29	0.67	1131	14.14	1.13
J 15 1 quat 100	46508263	32396263	1200	632	7.90	1.48	219	2.74	0.24
J 15 1 quat 75	25122290	11010290	1200	1050	13.12	1.32	511	6.39	0.78

Figure 12: PPC 7100/80 and Quadra 950/33, 160 × 120 – Played off Disk

Test Movie	Size			PPC			Quadra		
Codec/FPS/KFD/Size/Quality	Total	Video	$\mathcal{F}_T$	$\mathcal{F}_D$	$\overline{\mathcal{R}}$	$\mathcal{R}_{\mu_2}$	$\mathcal{F}_D$	$\overline{\mathcal{R}}$	$\mathcal{R}_{\mu_2}$
C 30 1 half 100	65668948	51556948	2400	2397	29.96	0.04	2398	29.98	0.05
C 30 3 half 100	55288196	41176196	2400	2393	29.91	0.08	2399	29.99	0.01
C 30 5 half 100	53200812	39088812	2400	2392	29.90	0.11	2400	30.00	0.00
C 30 10 half 100	51599160	37487160	2400	2389	29.86	0.14	2388	29.85	0.90
C 30 1 half 75	56272532	42160532	2400	2399	29.99	0.01	2388	29.85	0.13
C 30 3 half 75	48114560	34002560	2400	2389	29.86	0.12	2396	29.95	0.05
C 30 5 half 75	46467008	32355008	2400	2392	29.90	0.11	2396	29.95	0.05
C 30 10 half 75	45253496	31141496	2400	2394	29.93	0.07	2400	30.00	0.00
V 30 1 half 100	138782648	124670648	2400	2358	29.48	0.55	461	5.76	73.18
V 30 3 half 100	126319334	112207334	2400	2370	29.62	1.61	1516	18.95	47.70
V 30 5 half 100	123699495	109587495	2400	2375	29.69	1.02	1662	20.77	50.62
V 30 10 half 100	121666111	107554111	2400	2384	29.80	0.49	1809	22.61	60.69
V 30 1 half 75	110761808	96649808	2400	2351	29.39	0.66	296	3.70	47.48
V 30 3 half 75	89538819	75426819	2400	2371	29.64	1.43	1280	16.00	70.92
V 30 5 half 75	85251859	71139859	2400	2376	29.70	1.09	1540	19.25	76.51
V 30 10 half 75	81892471	67780471	2400	2372	29.65	0.91	1713	21.41	73.07
J 30 1 half 100	94238136	80126136	2400	473	5.91	0.53	-	-	-
J 30 1 half 75	76101834	61989834	2400	491	6.14	0.72	-	-	-
C 15 1 half 100	39892376	25780376	1200	1200	15.00	0.00	1199	14.99	0.01
C 15 3 half 100	34932804	20820804	1200	1200	15.00	0.00	1200	15.00	0.00
C 15 5 half 100	33894160	19782160	1200	1200	15.00	0.00	1200	15.00	0.00
C 15 1 half 75	35187128	21075128	1200	1200	15.00	0.00	1200	15.00	0.00
C 15 3 half 75	31257064	17145064	1200	1200	15.00	0.00	1200	15.00	0.00
C 15 5 half 75	30448024	16336024	1200	1200	15.00	0.00	1200	15.00	0.00
V 15 1 half 100	76406147	62294147	1200	1200	15.00	0.00	1195	14.94	0.06
V 15 3 half 100	71468196	57356196	1200	1198	14.97	0.11	1197	14.96	0.04
V 15 5 half 100	70419248	56307248	1200	1197	14.96	0.20	1197	14.96	0.04
V 15 1 half 75	62437907	48325907	1200	1200	15.00	0.00	1196	14.95	0.05
V 15 3 half 75	53301189	39189189	1200	1197	14.96	0.20	1197	14.96	0.04
V 15 5 half 75	51404010	37292010	1200	1197	14.96	0.20	1198	14.97	0.02
J 15 1 half 100	109158425	95046425	1200	282	3.52	0.62	-	-	-
J 15 1 half 75	45082267	30970267	1200	528	6.60	0.82	194	2.42	0.27

Figure 13: PPC 7100/80 and Quadra 950/33, 320 $\times$ 240 – Played from RAM

Test Movie	Size			PPC			Quadra		
Codec/FPS/KFD/Size/Quality	Total	Video	$\mathcal{F}_T$	$\mathcal{F}_D$	$\overline{\mathcal{R}}$	$\mathcal{R}_{\mu_2}$	$\mathcal{F}_D$	$\overline{\mathcal{R}}$	$\mathcal{R}_{\mu_2}$
C 30 1 quat 100	29766828	15654828	2400	2399	29.99	0.01	2400	30.00	0.00
C 30 3 quat 100	27746068	13634068	2400	2399	29.99	0.01	2400	30.00	0.00
C 30 5 quat 100	27310408	13198408	2400	2399	29.99	0.01	2400	30.00	0.00
C 30 10 quat 100	26985084	12873084	2400	2398	29.98	0.02	2400	30.00	0.00
C 30 1 quat 75	29449976	15337976	2400	2398	29.98	0.02	2400	30.00	0.00
C 30 3 quat 75	26442288	12330288	2400	2395	29.94	0.06	2400	30.00	0.00
C 30 5 quat 75	25860228	11748228	2400	2399	29.99	0.01	2400	30.00	0.00
C 30 10 quat 75	25421516	11309516	2400	2399	29.99	0.01	2400	30.00	0.00
V 30 1 quat 100	46998831	32886831	2400	2395	29.94	0.06	2400	30.00	0.00
V 30 3 quat 100	42843008	28731008	2400	2391	29.89	1.01	2400	30.00	0.00
V 30 10 quat 100	41322935	27210935	2400	2393	29.91	0.24	2400	30.00	0.00
V 30 1 quat 75	38400471	24288471	2400	2392	29.90	0.09	2399	29.99	0.01
V 30 3 quat 75	32259404	18147404	2400	2389	29.86	1.50	2399	29.99	0.01
V 30 10 quat 75	30045289	15933289	2400	2395	29.94	0.44	2400	30.00	0.00
J 30 1 quat 100	78918203	64806203	2400	859	10.74	1.32	306	3.83	0.42
J 30 1 quat 75	36138726	22026726	2400	1478	18.48	3.05	578	7.22	0.82
C 15 1 quat 100	21939432	7827432	1200	1200	15.00	0.00	1200	15.00	0.00
C 15 3 quat 100	21037560	6925560	1200	1200	15.00	0.00	1200	15.00	0.00
C 15 5 quat 100	20823272	6711272	1200	1200	15.00	0.00	1200	15.00	0.00
C 15 1 quat 75	21780904	7668904	1200	1200	15.00	0.00	1200	15.00	0.00
C 15 3 quat 75	20319956	6207956	1200	1200	15.00	0.00	1200	15.00	0.00
C 15 5 quat 75	20029612	5917612	1200	1200	15.00	0.00	1200	15.00	0.00
V 15 1 quat 100	30546740	16434740	1200	1200	15.00	0.00	1200	15.00	0.00
V 15 3 quat 100	28887319	14775319	1200	1198	14.97	0.11	1200	15.00	0.00
V 15 1 quat 75	26254172	12142172	1200	1200	15.00	0.00	1200	15.00	0.00
V 15 3 quat 75	23659209	9547209	1200	1197	14.96	0.20	1200	15.00	0.00
J 15 1 quat 100	46508263	32396263	1200	858	10.72	1.25	336	4.20	0.33
J 15 1 quat 75	25122290	11010290	1200	1197	14.96	0.04	578	7.22	0.67

Figure 14: PPC 7100/80 and Quadra 950/33, 160 × 120 – Played from RAM