Experiments with Digital Video Playback *

Richard Gerber and Ladan Gharai
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, MD 20742
{rich, ladan}@cs.umd.edu

UMD Technical Report CS-TR-3551, UMIACS TR 95-103

Abstract

In this paper we describe our experiments on digital video applications, concentrating on the static and dynamic tradeoffs involved in video playback. Our results were extracted from a controlled series 272 tests, which we ran in three stages.

In the first stage of 120 tests, we used a simple player-monitor tool to evaluate the effects of various static parameters: *compression type, 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 higher end, a Power PC 7100/80. Our quantitative metrics included average playback rate, as well as the rate's variance over one-second intervals.

The first set of experiments unveiled several anomalous latencies. To track them down we ran an additional 120 tests, whose analysis led us to find the locus of the system's bottlenecks. They also let us conclude that a software-only solution was sufficient for good video playback on the systems under observation – provided that the operating system is tuned accordingly.

In the next step we attempted to achieve this goal, by implementing our own video playback software and accompanying device-level handlers. Our emphasis was on achieving a controlled, deterministic coordination between the various system components. An additional set of 32 experiments were carried out on our platforms, which showed significant improvements in our quantitative performance measurements, as well as in visual quality.

Keywords: Digital Video, Performance Analysis, Measurements, Experiment Construction, Operating Systems.

*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

There is usually a wide asymmetry between the workstation on which a digital video is mastered, and the target platforms on which it is played. Video editing systems contain expensive peripherals like full-screen, full color digitizers, high-capacity RAID disk configurations, etc. Due to the quality requirements involved, this kind of equipment is fully warranted: a *broadcast-quality* video demands a resolution of 640×480, a display rate of 30 frames per second, and a color depth of 24 bits per pixel. Some simple multiplication yields a transfer rate of 27 Mbytes per second, or roughly 50 Gbytes of storage for a one-half hour production.

On the other hand, a target system may be an average-level home computer, maybe possessing 2 Gbytes of disk space, with peak transfer rates of 2.5 Mbytes per second. The computer's display logic will usually 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 tracks, etc.

Therefore, the inevitable final step in editing is the attempt to reconcile the vast differences between the producer's workstation, and those of the potential consumers. 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. But this raises two questions:

1. *Since a video can be tuned in a variety of ways, which methods lead to the best results on consumer-grade platforms?*

2. *While only one digital video gets released, there are many potential platforms on which it will get played. Each has its own resource constraints and processing abilities. So in order to achieve smooth, deterministic playback quality on each, a certain amount of dynamic tuning can be done as well. Is this currently done in commercial software, and if not, can it be done?*

As for static tuning, all down-sampling schemes include some sacrifice in quality – e.g., one can control the amount of signal loss used in digital compression, or decrease the color-depth, the digitized rate, frame 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 such "quality-reducing" measures necessarily lead to a reduction in dropped frames.

As for dynamic tuning, assume that video V is mastered at 30 frames per second (henceforth abbreviated as "fps"), and that platform A can play it with minimal dropping of frames. The lesser-endowed platform B may be able to play a statically down-sampled, 15fps version of V. Does this imply that B can play the 30fps version, and on-the-fly tune it down to 15fps? If so, the 30fps version can be the released version. If not, then perhaps the 15fps version should be released instead.

We attempted to quantifiably answer these two questions via a controlled series of of 272 experiments, which we ran in three stages. In the first stage we used a player-monitor tool to evaluate the effects of various static tuning approaches. The results unveiled several anomalous latencies, and to track them down we ran an additional 120 tests, which uncovered where the bottlenecks were. Based on these results, we re-tooled the system's video playback software, and incorporated our changes into a new player-monitor, whose goal was to achieve good dynamic tuning. We evaluated its performance with 32 more experiments, and we compared them with the performance of commercial-grade software.

This paper contains the results of our tests, their analysis, and the system architectural changes that we made based on our conclusions.

1.1 Metrics and Variables

Before setting out, we had to: (1) develop sound, quantitative metrics which loosely corresponded to "qualitative performance;" (2) select suitable, comparable platforms on which we could benchmark our experiments; (3) find 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) identify a set of test variables, whose instantiations would generate our "test runs."

(1) Metrics. 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:

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

That is, $\mathcal{R}_{\mathrm{PREF}}(e)$ is the digitized rate of the movie in 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 $\overline{\mathcal{R}}(e)$ 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.

For example, Figure 1 shows three trials, all of which possess about the same $\overline{\mathcal{R}}$, but which

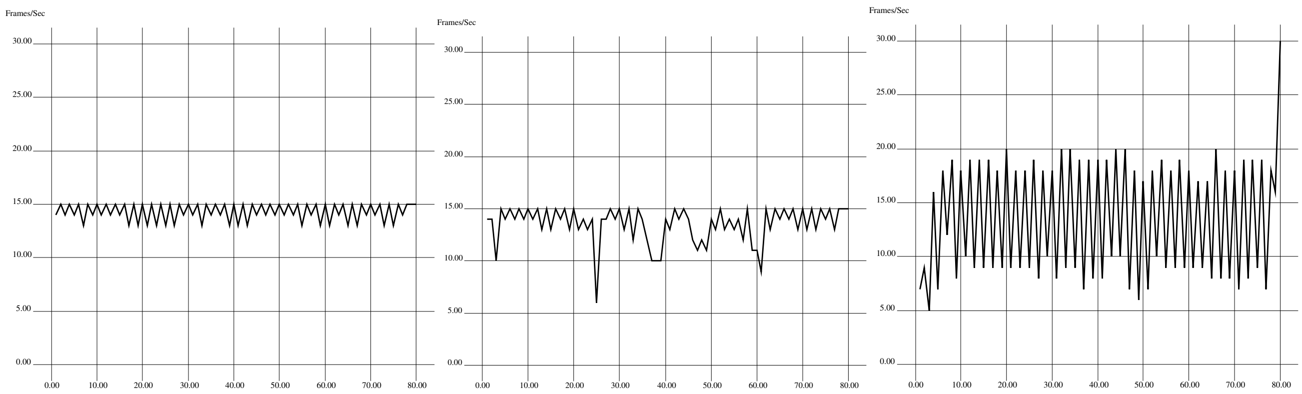

Figure 1: Playing Three Digital Versions of a Video: (1) Smooth, (2) Spikes and (3) Ocillation

show very different quality. To distinguish between these cases (and others), 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, then $\mathcal{R}_{\mu_2}(e)$ is defined as:

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

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

Our decision to use Macintoshes was preceded by a "qualitative" investigation into the video capabilities of different contemporary, affordable workstations.[1] While there is a wide variety of workstations currently available, Apple's products still seem to deliver the best "software-codec" video.[2] This is partially due to the fact that over the years, Apple has invested heavily in optimizing the QuickTime and QuickDraw libraries.[3] But perhaps of greater importance is the fact that at the application level, Mac tasks are largely nonpreemptive and event-driven, while multi-threading is usually a cooperative affair.

While these "features" lead to a very awkward programming style (every application program essentially becomes a system program too), they also end up providing an interference-free platform for evaluating video applications.

But using the Macintosh platform, with its proprietary operating system and codec drivers, presented a unique challenge throughout this work: We had to develop a true exogenous testing strategy, relatively independent of the innards of the system. As it turns out, we were able to

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

[2] A codec is a compression/decompression scheme. A software codec is a set of algorithms in which video compression and/or decompression is performed without special hardware.

[3] QuickTime includes a standard API for interfacing with codecs, and with functions to build, store and play movies. QuickDraw encompasses a set of functions which draw single frames to the screen.

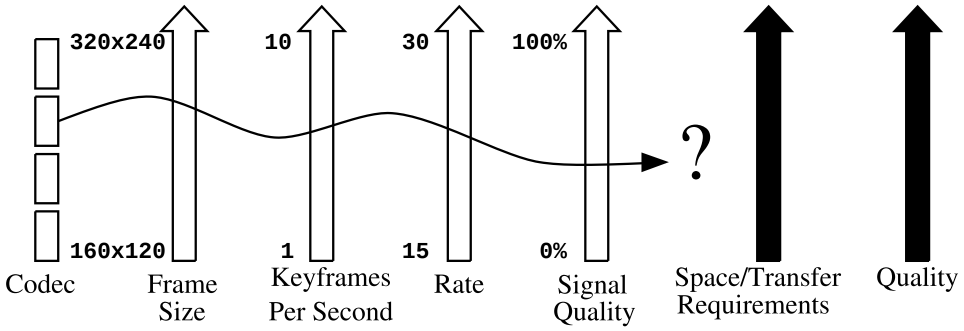

Figure 2: Instantiating the Variables

incrementally deduce, and then pinpoint where various bottlenecks lay. We also were able to implement key optimizations – but only after analyzing the results of a great many tests.

(3) Test Videos. 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. This decision gave us ultimate control over content, and allowed us to get a sufficiently wide spectrum of changes in motion, light, sound, etc.

We were then able to compare the quality of the computer-based, digital experiments with our original, analogue Hi8 clips played on a high-quality VCR.

(4) Variables. Once we had our digital "master," we produced sixty different copies, with each possessing a set of uniquely instantiated variables. We identified five variables 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 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.

Other variables, such as sound quality, were kept fixed. (Throughout we maintained 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 – hence complicating the static tuning problem. Picture size, for example, is immediately identifiable, and most people would agree that "bigger is better." Yet while too big a frame size will lead to untenable data rates, depending on the other parameters chosen, reducing it may not lead to a sizeable reduction. 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. Hence the need for some degree of dynamic tuning as well – even in the highest-end system.

1.2 Remainder of the Paper

The remainder of this paper is organized as follows. In Section 2 we survey some of the related systems work in digital video. Then in Section 3, we describe how we generated our test cases, and how we controlled the variables involved. In Sections 4 we describe our first series of test results, and we analyze the significance of each variable. In Section 5 we track down the bottlenecks realized in the first series, using a second series of experiments. In Section 6 we give the design of our improved video-playback software, and we present the results we obtained with it. We conclude in Section 7.

2 Related Work

To date, the area of digital video has been treated largely as a problem of system design and implementation, without the experimental focus which we take in this paper. Nonetheless, some recent papers touch on many of the same issues we raise in our experiments, and they come to some similar conclusions.

Stone and Jeffay's study of delay and jitter management [8] is highly relevant to our work on dynamic tuning. 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 dynamic adjustment of display latency, which supports low-latency conferences with acceptable gap-rates.

A related issue is achieving graceful degradation of service in the event of network congestion. One approach to this problem is for the client to adaptively *scale* the playback rate by deterministically dropping some of its frames. If this is done properly, the radical oscillations seen in Figure 1(3) should not occur, and ideally $\mathcal{R}_{\mu_2}(e)$ will be close to zero. This is the approach taken in the the Nemesis [5] project, which uses a *predictive prefetch* algorithm to scale a client's input streams. One limitation is the exclusive reliance on the JPEG codec – thus, the prefetch algorithm assumes that any frame may either be retrieved or dropped. This type of rate-based scaling is significantly more complicated when applied to codecs with inter-frame dependencies, such as Cinepak, MPEG, etc.

The system described in [1] scales not only the rate, but also the spatial resolution of a video stream. This is done by packaging three versions of every frame, with each offering a monotonic improvement over the previous one. The first is a 160x120 abstraction of the original picture; the next is the residue term which, when added to the 160x120 image, achieves a resolution of 320x240. The final version is another residue which can be added to the 320x240 image, resulting in full 640x480 resolution. At any point in the process the codec can stop improving the current frame, and proceed to the next. Of course, this flexibility is achieved by using a custom codec, which was designed specifically for this purpose.

Our focus on IO and data paths is echoed in [2, 3], 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. 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.

Two additional playback-enhancing techniques concentrate on optimizing disk performance; these are *disk scheduling* and *block placement*. Simulation studies conducted in [7] show that good performance for single streams can be realized by both CSCAN and SCAN-EDF – a hybrid of the traditional SCAN technique, and the "earliest-deadline-first" strategy used in real-time thread schedulers. Moreover, both algorithms can support a number of concurrent streams, and both have reasonable response times for aperiodic requests. Another technique, the Group Sweeping Scheme [4], is a hybrid of round-robin and SCAN. A number of "groups" are scheduled via round-robin, whereas within each group the SCAN algorithm is used. To a large extent, this allows for compromising between the disk-head's ability to "sweep up" physically neighboring blocks, and the temporal requirements imposed on concurrent, time-based media streams.

Our experiments could not test this important aspect of the system architecture. The reason is fairly simple: if one wishes to use off-the-shelf disks, then one must live with the vendor-supplied, proprietary policies which are hard-coded into controller's micro-program. These include block scheduling, lookahead-buffer maintenance, local caching, etc. On the positive side, however, disk controllers are increasingly being optimized for "multimedia systems" – which usually translates into good "sustained" read/write performance over contiguous blocks.

But there is an obvious relationship between disk scheduling and placement, and placement is an area where the multimedia system designer can have an immediate effect. The trade-offs are enunciated in [4]: contiguous placement is optimal for most disk-schedulers, but it requires allocating a huge "chunk" of consecutive tracks to each file. This policy is only manageable for read-only files, and it can introduce significant external fragmentation. On the other hand, scattered placement is more space-efficient, and streamlines the management of read-write files. The downside is that a large number of intra-file seeks will be required – which can inject a large performance penalty.

In this paper we are concerned with testing read-only performance, and each test is run under nearly optimal disk conditions. We first de-fragment the entire file system, and then we optimize each file into single chunks of contiguous blocks.

Finally, the abilities of our player-monitor are similar to that described in [9], where MPEGencoded video streams can be generated expressly for providing statistical information on the data, such as distribution of ATM cells per frame, auto-correlation 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. Our 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 dialogue 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 80 seconds, with $\mathcal{R}_{\mathrm{PREF}}$ set to 30 fps, and it formed the "digital master" for each test case. Without belaboring the details, we carried out the following steps: (1) Using a Hi8 video camera to photograph about 3-4 hours worth of video "clips," which included scenes with different levels of light, color, motion, etc.; (2) Using a JPEG hardware codec to digitize the clips into a 7100/80 – and storing them on a 2.1G disk; (3) Using Adobe Premiere 4.0 to assemble an 80 second master, which contained sufficient contrast to stress the capabilities of compression and playback software; (4) Employing sound-processing software to sample and filter "background music" from CDs, and inserting them in the movie at appropriate levels; (5) Using the Premiere/Digitizer combination in conjunction with the Quicktime libraries to generate 60 software-digitized copies from our master – each copy representing a separate instantiation of our variables; and (6) Archiving our 60 files on six CDs (Figure 3).

We were then able to run the tests and monitor their our testing tools, which we built for this purpose.

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). The VVS is capable of digitizing frame sizes 640x480 with little signal loss, and no "drop-out" frames. JPEG compression possesses a large degree of parallelism, which a hardware codec can easily exploit.

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 a reasonable amount of footage. 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

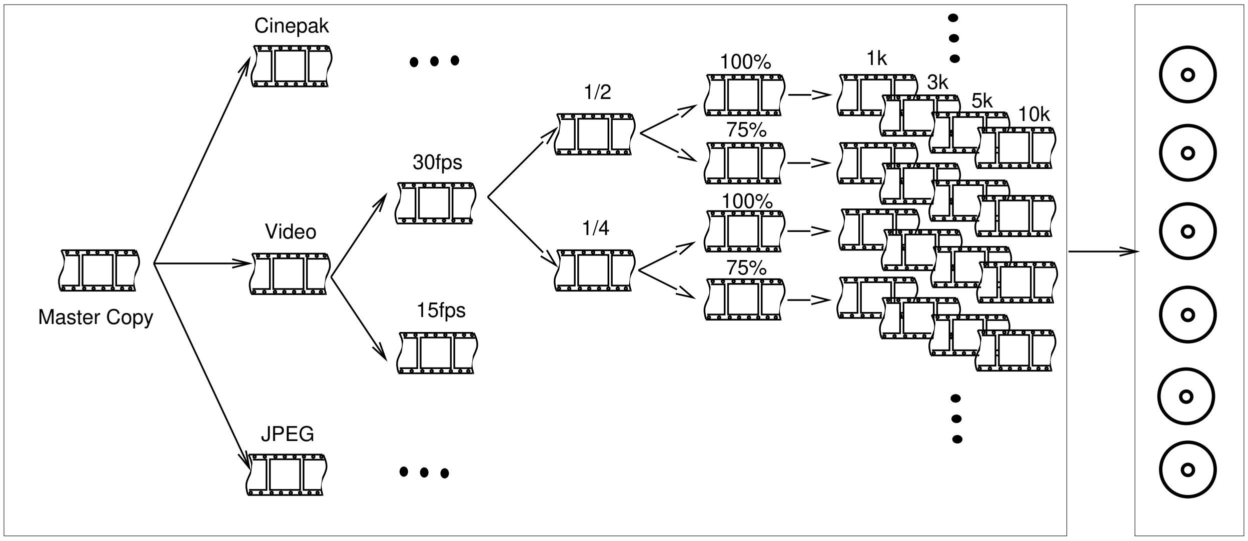

Figure 3: Test Movie Instantiation.

Computing Equipment	
Test Platform 1: PPC 7100/80 CPU: MC601 80MHz, Floating Point Memory: 24Mb Internal Hard Drive: 696 Mbyte, IBM DSAS3720 CD: Apple 300i (2-speed, Internal) OS: System 7.5, Minimal Extensions Disk Drivers: From FWB	Test Platform 2: Quadra 950 CPU: MC68040 33 MHz, Floating Point Memory: 16Mb Internal Hard Drive: 234Mb, Quantum LP2435 CD: Apple 300i (2-speed, External) OS: System 7.5, Minimal Extensions Disk Drivers: From FWB
Essential Peripherals	**Video Equipment**
External Hard Drive: 2.1Gbyte, Seagate ST12400N Digitizer: Radius VideoVision Studio 2.0 CD Writer: Pinnacle Micro RCD 1000 Tape Drive: Exabyte 2000	Cameras: Sony CCD TR700s Hi8 Format Time Code, HiFi Stereo, S-video In/Out VCR: Sony EVS7000 Hi8 Format 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

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 the real answer lies in the asymmetry between the producer's system and that of the video consumer. While a real-time JPEG capture card may be suitable for a high-priced video workstation, if one is purely interested in transmitting and viewing videos, then such an arrangement is probably the worst alternative. JPEG cards do not perform inter-frame compression, and they produce significantly more data than most systems can accommodate. On the other hand, full-field, full-color, realtime MPEG (and Motion-JPEG) decompression cards are still very specialized peripherals, and are quite expensive.

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 Kbytes/sec
Write Transfers:	3100 Kbytes/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 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*. Anther variable that could have been altered – but was not – was sound quality. Across all experiment it remained fixed, at 16-bit stereo sampled at 44kHz.

Formally, the variable space ranges over

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

where

1. $Codec \in \{\mathrm{C}, \mathrm{V}, \mathrm{J}\}$ denotes the compression scheme used. Here "C" is Radius's Cinepak

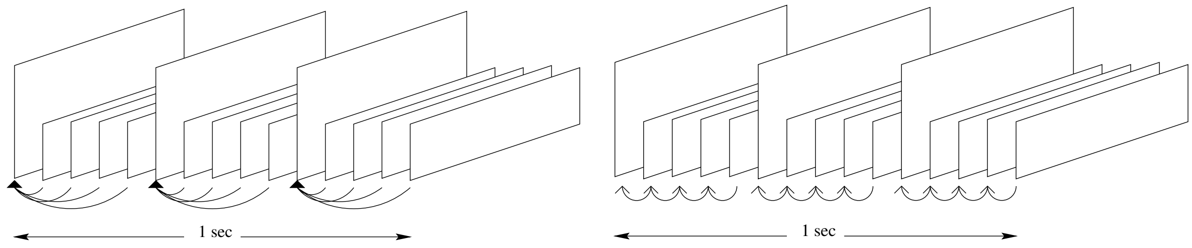

Figure 4: One-Second Strip of Frames, with $Rate = 15$ and $KFD = 5$. Apple Video (Left) and Cinepak (Right).

codec, "V" is the "Apple Video" codec (sometimes known as "Road Pizza"), and "J" stands for QuickTime's frame-by-frame, still-JPEG codec.

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

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. $Size \in \{\mathrm{half}, \mathrm{quat}\}$ denotes the frame size along one axis, 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 [6] is basically a compression standard for still-pictures, which can produce nearly lossless digital copies. 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.

Whereas still-JPEG involves processing each frame individually, the Apple Video and Cinepak codecs 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 "Video," or relative to their preceding neighboring frames as in "Cinepak" (Figure 4).

QuickTime does this in a rather crude (but effective) manner: when temporally compressing Apple Video frame k relative to its keyframe i, k's pixel-map is simply subtracted from that of i, and the result is compressed. This type of compression can often be a time-consuming process. For example, each re-compression of our 80-second test in Cinepak required about two hours.

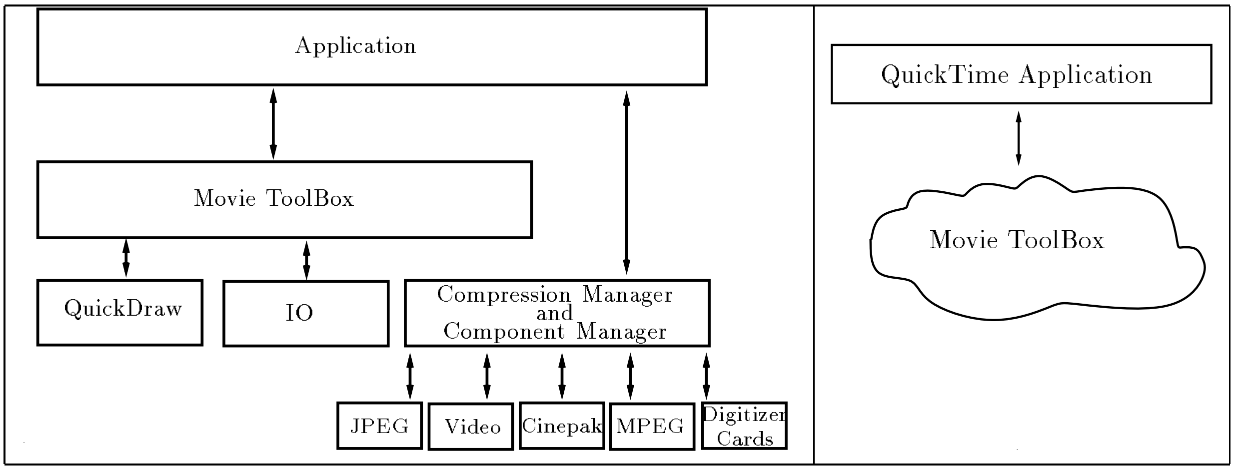

Figure 5: Quicktime Structure (left), and its interface to Movie Monitor I(right)

3.3 System Factors

We eliminated the effects of disk fragmentation. Whenever we ran a test sequence, we first reformatted our 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 external and 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 $\overline{\mathcal{R}}$ 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 Series I: Static Tuning

We now turn to our first set of experiments, in which we tested all possible "static tuning" decisions made over the domains of our variables. The results were obtained by a player-monitor tool, which uses the high-level video playback API included in Quicktime – i.e., the same functions used in almost all movie players. We first describe the monitor's architecture, and proceed to analyze the test results.

4.1 Movie Monitor I

QuickTime presents a uniform API for Mac media applications[4], and it includes internal interfaces to most of the commonly used codecs, as well as to third party digitizer cards. QuickTime has a layered architecture (Figure 5), and an application interacts with it via functions contained

[4] QuickTime for Windows is also available for playback on PCs, and some of QuickTime's functionality has also been ported to Unix.

in either the Movie ToolBox or the Compression Manager. As a rule, the typical movie player will deal exclusively with the ToolBox, whereas a video editing program (like Adobe Premiere) will also directly invoke the lower-level codec controlling functions in the Compression Manager.

A QuickTime file (called type *MooV* on the Mac) consists of two parts: (1) a hierarchal data structure containing frame-by-frame information about the movie (e.g., size, duration, file location); and (2) the movie data itself. All of the information in the data structure can be extracted via function calls to the Movie ToolBox.

Our Movie Monitor I is just a typical, though somewhat scaled down, video player for the Macintosh. Like most players, it repeatedly calls the ToolBox function **MovieTasks()** to retrieve, buffer, decompress and display frames, and to synchronize the audio and video streams. The monitor also maintains running information on the frames that get displayed, and those that are dropped. It keeps this frame-by-frame data 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 on our systems (usually about 20Mbytes). 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.

4.2 Test Results

The results of our initial half-screen tests are listed in Figure 15 (in the Appendix); to better understand the "highlights" of the half-screen tests, we present their $\overline{\mathcal{R}}$ data in Figures 6-7.

Platform and Codec Variables. One need not delve into the numbers to make some coarse 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 obvious observation is that JPEG is a very poor performer, while Cinepak shows the best average performance in most test cases.

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 30 fps.

Examining all of the tests, the conclusion we reached is as follows: that when $\overline{\mathcal{R}}$ was over $(\frac{3}{4})\mathcal{R}_{\mathrm{PREF}}$, additional spatial and temporal compression failed to raise it further (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

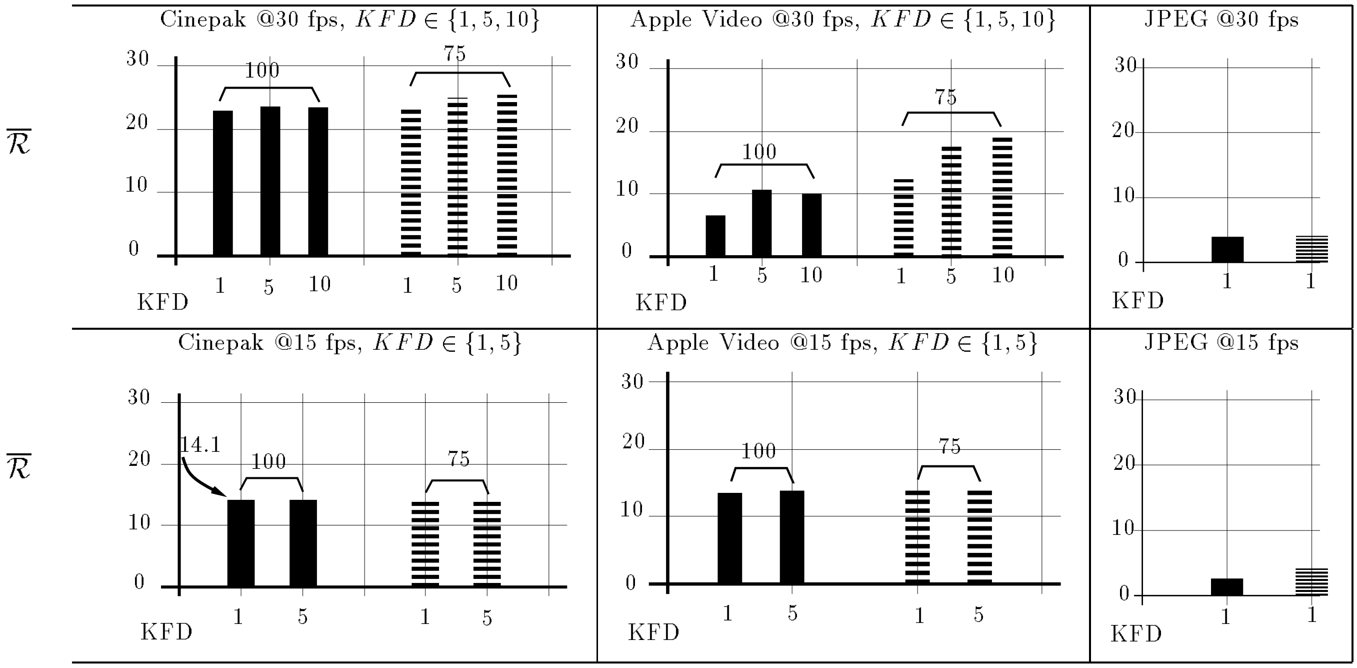

Figure 6: $\overline{\mathcal{R}}$ of All Half-Screen Runs Played on PPC 7100/80 off of Disk. Keyframes are Numbered 1-10. Solid Bars Denote $Quality = 100$, Striped Bars Denote $Quality = 75$

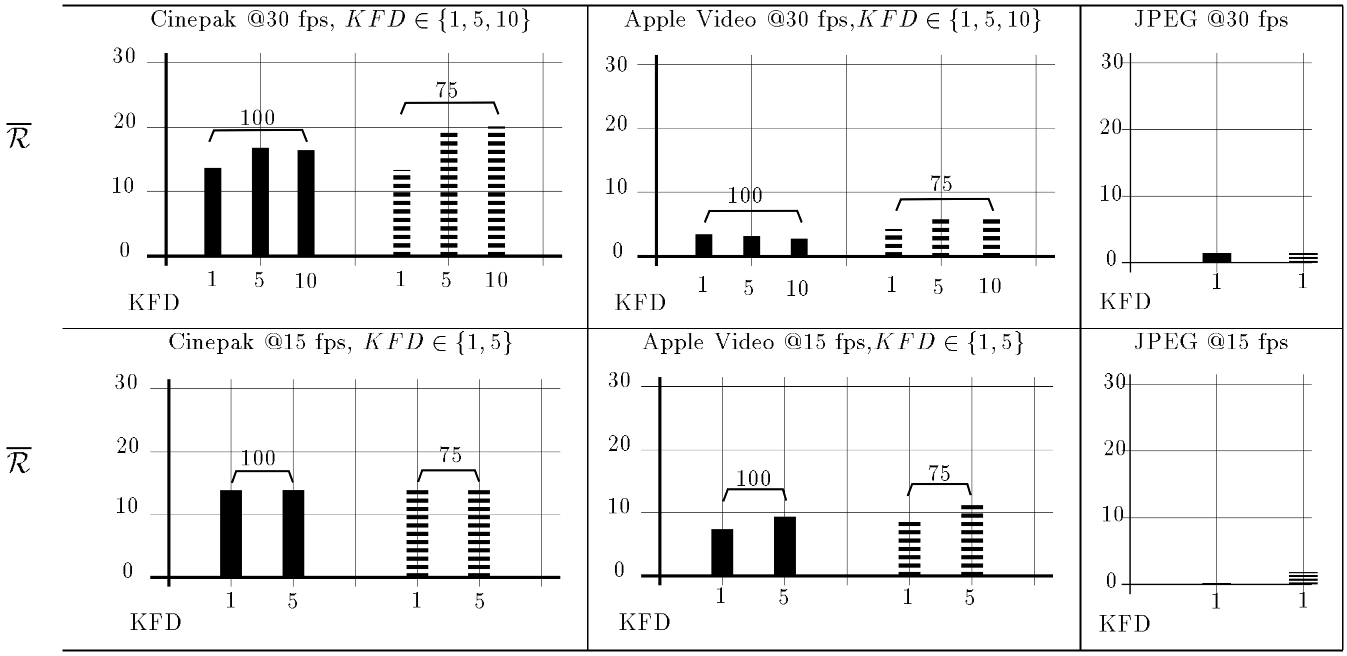

Figure 7: $\overline{\mathcal{R}}$ of All Half-Screen Runs Played on Quadra 950/33 off of Disk. Keyframes are Numbered 1-10. Solid Bars Denote $Quality = 100$, Striped Bars Denote $Quality = 75$

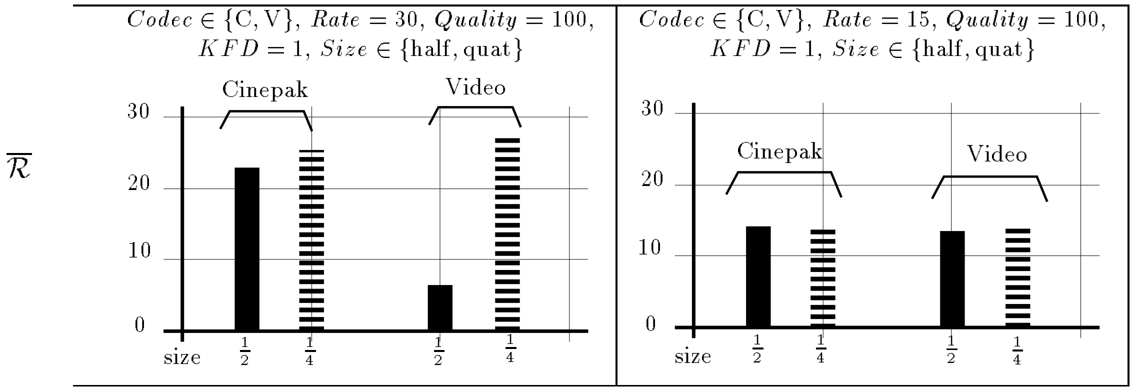

Figure 8: $\overline{\mathcal{R}}$ High-Quality PPC 7100/80 Runs, Where Screen Size is Altered. Solid Bars Denote $Size = \text{half}$, Striped Bars Denote $Size = \text{quat}$

and temporal compression, poor performance poor abysmal. For example, comparing the Quadra's treatment of "V/15/1/half/100" to that of "V/15/5/half/75" – a 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 above 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 Figure 15 and Figure 16, one sees that this is often 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 160x120 screen is hardly satisfying; thus the reduced size should, one hopes, pay off in many fewer dropped frames. Does it?

Figure 8 compares the effects of frame size on selected 30fps and 15fps trials. In the case of Apple Video at 30fps, the answer is "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. 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.

As for the 15fps runs, the Apple 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. That is, once $\mathcal{R}_{\text{PREF}}$ is set to 15 fps, substantially lowering the data-rate by reducing frame size clearly 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.

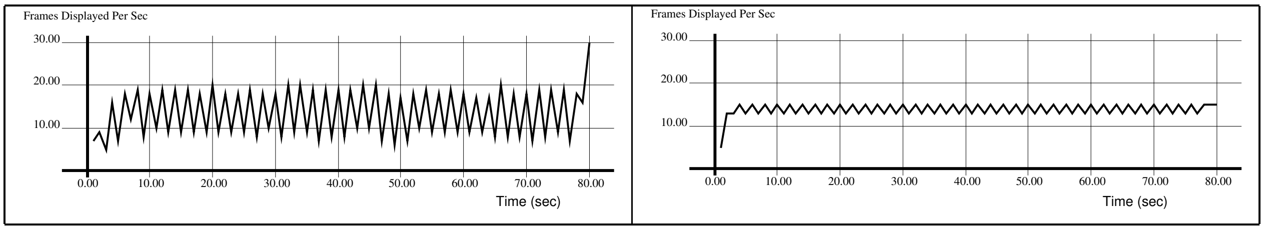

Figure 9: C/30/1/half/100 (left) and C/15/1/half/100 (right)

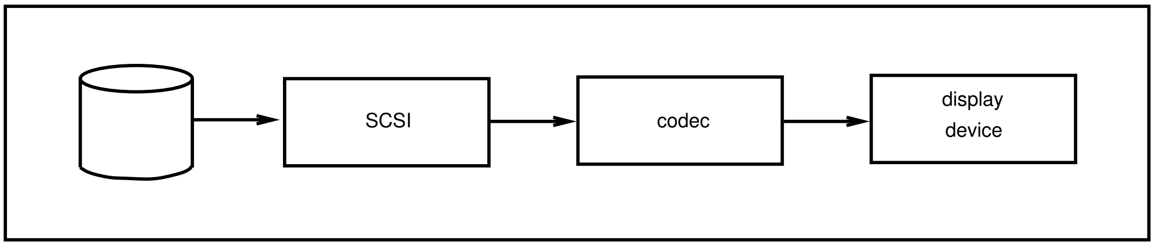

Figure 10: System components involved in video playback.

The Effects of Rate Changes. Considering the *Rate* variable, the Apple Video statistics in Figures 6-7 show situations where 15 fps tests realize a higher playback rate than 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 9, 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 $\overline{\mathcal{R}} = 13.91$). Tracking the on-line behavior of the two, it is easy to see 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).

5 Test Series II: Tracking the Bottlenecks

What accounts for the high amount of jitter realized in some of the 30fps trials? Three potential bottlenecks suggest themselves: (1) the display manager, (2) the codec or (3) the IO channel between the hard drive and the codec (Figure 10).

We considered Case (1) unlikely, since intuitively it seemed that decompressing a frame would be much more time-consuming than simply displaying it. Also, the codec's output gets streamed directly to QuickDraw (for screen updates), and QuickDraw is probably the most finely-tuned part of the Macintosh. To substantiate this intuition, we set up a small test: we measured the CPU time required to decompress a series of 1200 frames; then we measured the time it took to both decompress and display them. Within the granularity of the platform's clock, the two times were more or less the same.

So exploring Case (2), perhaps the de-compressor is much slower than the IO. In this case, the following scenario could play out: while the QuickTime drivers attempt to retrieve frames at a rate of 30fps, the frames can only be decompressed 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.

But Case (3) – i.e., unduly slow IO transfers – could also account for the regular oscillation in display rates. The premise here is that whenever a codec's input buffer contains compressed frames, they can quickly be de-compressed and drawn on the screen. On the other hand, IO transfers will result in prolonged periods of the buffer being empty.

A 15fps test demands only one-half of the display rate of its 30fps counterpart, and a proportionally lower IO transfer rate (see Figure 15). So it will put less pressure on *both* potential bottlenecks – which probably explains the 15fps's low-amplitude waveform (in Figure 9).

But it seems that the bottlenecks involved in the 30fps tests could be reduced if there were at least some degree of dynamic tuning, or perhaps more controlled coordination between the 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.

The RAM Tests. Our measurements presented us with even more surprising anomalies. Consider, for example, the half-screen/PPC/Cinepak runs listed in Figure 15. 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 platform "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 - i.e., the case in which a movie plays better at 15fps than it does at 30fps.

Clearly there is little dynamic tuning going on here; moreover, resources are frequently going unused. This is surprising in a robust and hand-tuned piece of software such as Apple's Quicktime, and it motivated us to pursue the next series of tests – to test the ability of the codecs alone, without IO. Thus we simply extended Monitor I 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 11, 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 Apple Video test climbs from $\overline{\mathcal{R}} = 6.60$ to $\overline{\mathcal{R}} = 29.48$.

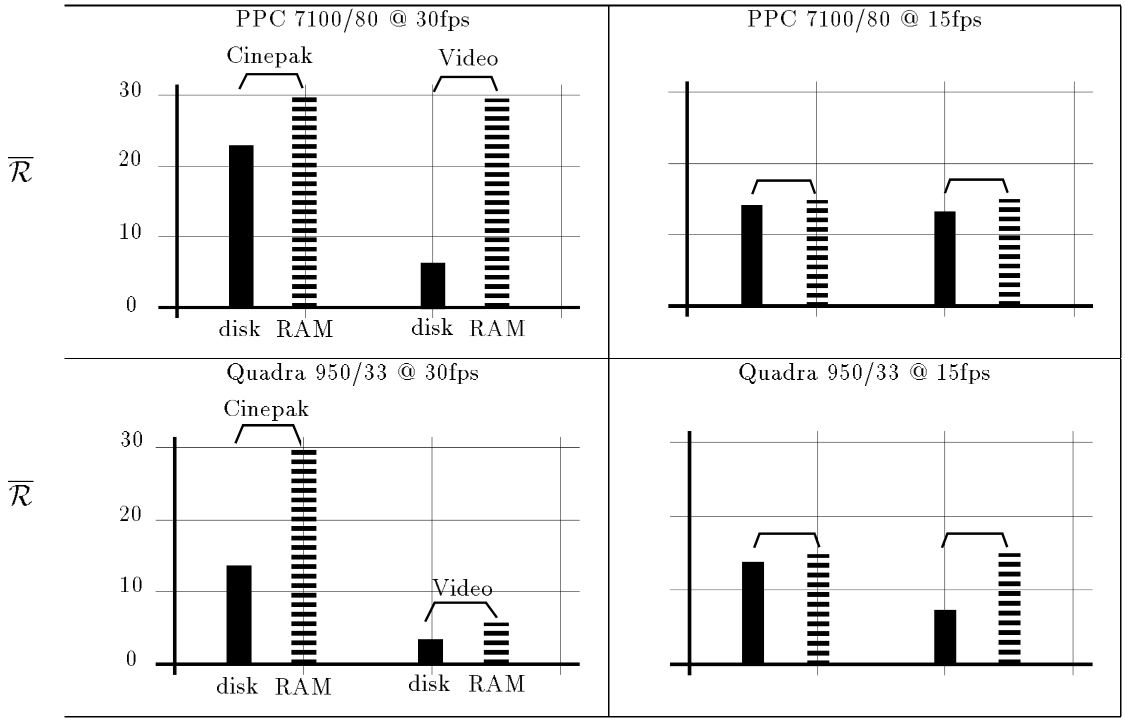

Figure 11: $320 * 240$, Highest temporal and spatial compression, Disk vs. RAM playback. Results from the disk runs are portrayed with solid bars, whereas the RAM runs are in striped bars.

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 they are nearly perfect).

As for the Apple 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.

Can we conclude that the disk is the bottleneck? "Common wisdom" dictates that this is true, 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.

However 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, even including the DMA's memory "cycle-stealing." So it seems that while IO may be asynchronous and DMA'd, it is not sufficiently asynchronous for the QuickTime codecs to run at their peak rates.

Moreover, we hypothesized that if a full pipeline effect were achieved between the system elements in Figure 10, there would be no difficulty in delivering near-30fps data rates on the PPC

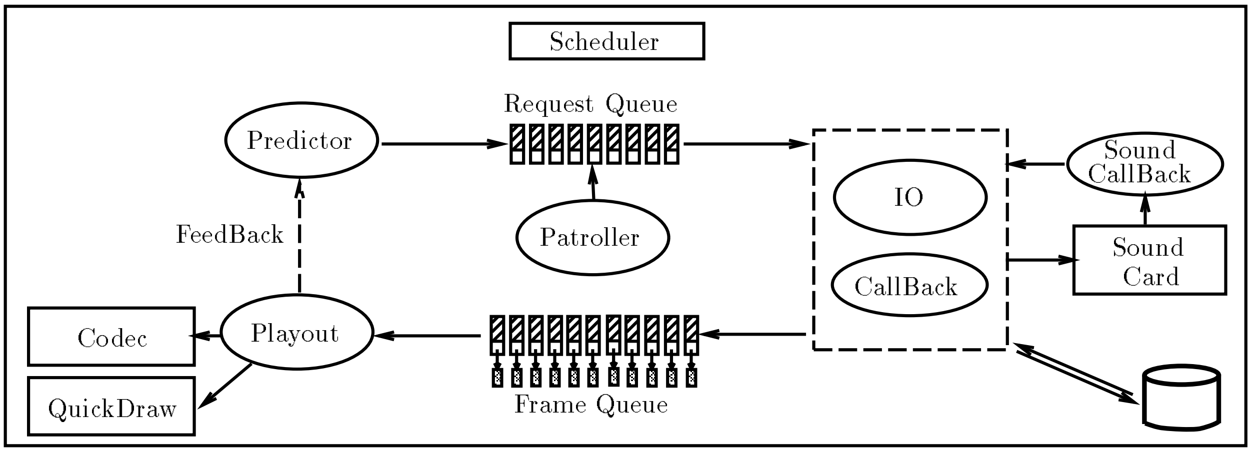

Figure 12: Movie MonitorII

system. To test this hypothesis, we built a new player-monitor which effectively bypasses the Movie ToolBox API. In doing so, we had to support our own buffering and IO functionality, as we describe in the next section.

6 Test Series III: Dynamic Tuning

Our new player-monitor (Movie Monitor II) was designed to dynamically tune digital videos, in part by streaming data from the SCSI disk through the codec software. Accomplishing this objective required implementing our own IO handlers, memory buffering, and performing low-level codec control via QuickTime's Compression Manager. In this section we describe the structure of the Movie Monitor II, and then we present the results of the tests we ran with it.

6.1 The Player Structure

Figure 12 depicts the structure of our player-monitor, which is composed of four threads and two callback functions. As the figure shows, the system operates as a simple feedback loop. Based on the player's past performance, the Predictor thread selects a set of frames to be played in the future, and inserts their frame numbers into the Request Queue. The IO thread removes them, looks up their corresponding file locations, and initiates the appropriate asynchronous IO commands for the SCSI manager. If there are requests for neighboring frames, the IO thread will attempt to bundle as many neighbors as possible within a single IO transfer. By bundling adjacent frames the IO thread (and all associated SCSI handlers) can execute less frequently; this CPU time can be better used by activities such as decompression. But when frame transfers are bundled, none of them can be processed until the entire bundle arrives. For this reason the IO thread only bundles when it gets sufficiently ahead of playout.

When an IO operation completes, its callback function preemptively executes like an interrupt service routine. It inserts a pointer to each transferred frame into the Frame Queue. They subsequently get removed by the Playout Thread which, according to its real-time movie clock, will either display the associated frames, or just discard them. If the frames have been delivered on time, then they get decompressed using the Compression Manager's codec interface, and displayed via QuickDraw.

The final thread used in processing video is called the Patroller. It is responsible for ensuring that none of the outstanding frame requests for the IO are outdated – lest frames end up getting transferred needlessly.

Part of the Playout Thread's job is to update feedback information for the Predictor, which is done at every time interval Δ. (For the results displayed in the sequel, Δ was set to $\frac{1}{2}$sec.) The feedback is in the form of a predicted playout rate for the next interval, and the Predictor uses it to dynamically tune its prefetch rate. This scheme is partially aided by the scheduler, which ensures that the Predictor gets at most most $\frac{1}{2}$ second ahead of the Playout thread.

Let t be a multiple of Δ, and let $\mathcal{PR}(t)$ be the predicted playback rate for the time interval $[t, t+\Delta]$. Then if we let $\overline{\mathcal{R}}(t)$ be the rate that the Playout thread *actually* achieved during the interval, $\mathcal{PR}(t+\Delta)$ is calculated as follows:

$$\mathcal{PR}(t+\Delta) = \left[\begin{array}{ll} \alpha\overline{\mathcal{R}}(t) + (1-\alpha)\mathcal{PR}(t) & \text{when } \overline{\mathcal{R}}(t) < \mathcal{PR}(t) \\ min(\overline{\mathcal{R}}(t) + c, \mathcal{R}_{\text{PREF}}) & \text{when } \overline{\mathcal{R}}(t) = \mathcal{PR}(t) \end{array} \right.$$

where for the purposes of our experiments, we set $\alpha = .85$ and $c = 1$. In other words, when playback falls behind its predicted rate, we exponentially average the old prediction with the achieved rate. (This is to smooth out sporadically large frame sizes, or abnormally high decompression times.) But when playback meets its prediction, we gradually ratchet up the new prefetch rate, so that eventually the highest potential quality can be realized.

The objective of our design is to let the system achieve a steady state, so that IO and playback are always working in parallel, at their full capacity. This means the Playout thread should never have to wait for a frame – the IO should always have prefetched it ahead of time, while the Playout thread was processing a previous frame.

Keyframes. Our scheme is significantly complicated by the existence of keyframes; e.g., if a keyframe is dropped, then the the interpolated sequence following has to be discarded. Thus, while $\mathcal{PR}(t)$ is the current predicted rate, the Predictor thread cannot simply fetch frames at a constant frequency. First a decision is made whether an entire sequence will be avoided. If not, its keyframe is requested, as are selected interpolated frames within the sequence. The Playout thread may end up only decompressing – but not playing – the keyframe, so that it can be used to display its dependent, interpolated frames.

Memory Management. We chose to implement our own memory management, and we used a simple fixed-size buffering scheme. At initialization time, the player-monitor determines how much memory M it can allocate, after which it configures its buffer pool. Using a rough rule of thumb we decided that if possible, the number of buffers n should be able to accommodate 1 to 2 seconds of video; this translates to 30 buffers for 15fps movies, and 60 buffers for 30fps movies.

As for the buffer size s, it should be sufficiently large for the IO thread to frequently bundle its transfers. We attempt to set s so that between 3 and 5 average-sized frames can be bundled, and so that s can hold the largest frame in the entire video.

Letting B be our bundling factor, f_{max} be the size of largest frame, and f_{avg} be the average frame size, we attempt to reconcile all of our memory constraints by solving for B, s and n in

$$\begin{array}{ll} (1) & max(f_{max}, B * f_{avg}) \leq s \\ (2) & s * n \leq M \\ (3) & 1 \leq B \leq 5 \\ (4) & \mathcal{R}_{\mathrm{PREF}} \leq n \leq 2 \cdot \mathcal{R}_{\mathrm{PREF}} \end{array}$$

For the tests run in this paper, our system initially sets n to $2 \cdot \mathcal{R}_{\mathrm{PREF}}$, and then maximizes B over equations (1)-(4). With such a scheme Monitor II ends up using significantly less memory than Monitor I.

Sound. The Mac comes equipped with an asynchronous sound card, which handles the physical processing of digital sound. Our player interacts with this device via a simple double-buffering scheme. When one buffer is almost finished being played, the sound card triggers a callback routine, and then switches to the other buffer. The callback initiates an IO transfer for another chuck of sound. Unlike video sound samples cannot be dropped, and so the IO thread gives them priority.

6.2 Test Results

This time we confined our attention to the higher-quality videos, excluding the 160x120 cases, as well as those of 75% spatial quality. Thus our narrower domain was:

$$Rate \in \{15, 30\}, \quad Codec \in \{\mathrm{C}, \mathrm{V}, \mathrm{J}\}, \quad Size = half, \quad Quality = 100, \quad KFD \in \{1, 3, 5, 10\}$$

Figures 13,14 and 19 summarize results we obtained. On the PPC we obtained perfect (or near perfect) playout for *all* Cinepak and Apple Video movies. Overall, the 30fps PPC/Cinepak tests improved by about 6fps, while the 30fps PPC/Video tests improved by 19-23 fps. In particular, the highest quality Apple Video test jumped from $\overline{\mathcal{R}} = 6.6$ to $\overline{\mathcal{R}} = 29.36$. In fact here we can hardly claim that the gains are due to dynamic tuning, since almost every frame is played in every movie. Rather, a better design allows the components to be truly asynchronous, and achieve a higher degree of pipelining.

On the other hand, dynamic tuning is tested on the slower Quadra 950/33. Relying on its own lower-level control, our Monitor II does a far superior job than Monitor I, which uses the proprietary code in the Movie ToolBox. Significantly, the highest quality Quadra/Video tests improves from $\overline{\mathcal{R}} = 3.39$ to $\overline{\mathcal{R}} = 14.41$, and other tests show similar improvement. While 14fps is still not 30fps, it is a definite improvement over 3fps. Further, we no longer see an inverse relationship between

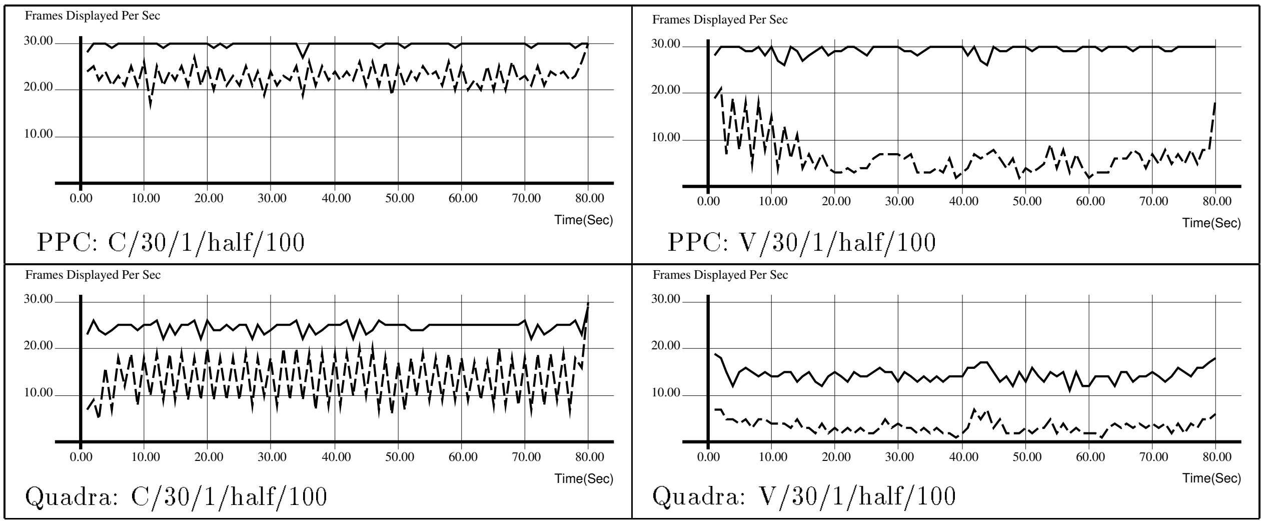

Figure 13: Comparison of playback schemes, for 30fps movies. Solid lines denote MonitorI and dashed lines denote MonitorII

$\mathcal{R}_{\mathrm{PREF}}(e)$ (preferred rate) and $\overline{\mathcal{R}}(e)$ (mean rate).

Our rough numerical analysis proved to be correct. If a video's data transfer requirements can be sustained by the SCSI disk, then asynchronous IO should never create a bottleneck. Direct DMA'd transfers into the player's buffers should not consume appreciable CPU time, thus it can be dedicated entirely to decompression.

Our results all indicate that this is what is happening. Both codecs achieved near-perfect playback rates on the PPC, where the processor is capable of keeping up with both codecs' CPU requirements. However on the slower Quadra, the Cinepak movies were able to achieve higher rates than their Apple Video counterparts. This is because the Apple Video codec is more compute-intensive.

We attribute the improvements in Monitor II to its almost classical producer-consumer structure. The IO thread – playing the role of producer – constantly supplies a steady flow of frames to be consumed by the Playout thread. It is never forced to "wait" for the arrival of a frame, which results in higher display rates with lower variance. This translates directly into better visual quality.

7 Conclusion

In this paper we reported results of three sets of experiments. The first tested a range of static video tuning methods. They also unveiled several anomalies – one of which was that low-rate movies often had superior playback quality to their higher-rate counterparts. To understand why, we ran a second set of tests, which tested the performance of the individual components involved. We concluded that many of the videos would realize far superior playback quality, if only IO and

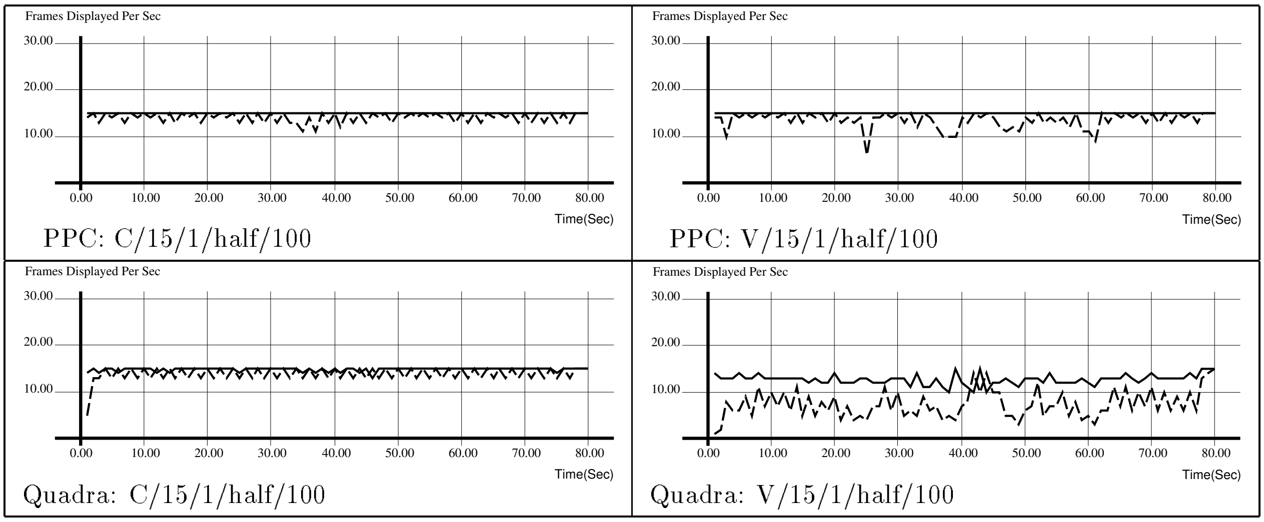

Figure 14: Comparison of playback schemes, for 15fps movies. Solid lines denote MonitorI and dashed lines denote MonitorII

codec operations were sufficiently pipelined.

In the next step we attempted to achieve this goal, by implementing our own video playback software and accompanying device-level handlers. Our emphasis was on achieving a controlled, deterministic coordination between the various system components. An additional set of 32 experiments were carried out on our platforms, which showed significant improvements in our quantitative performance measurements, as well as in visual quality.

In the future, we hope to achieve the same improvements with high quality, full screen digital videos. A 30 fps video with a frame size of 640x480 will put significantly greater pressure on the system's resources, and dynamic tuning will play an even more prominent role.

References

[1] Navin Chaddha, Gerard A.Wall, and Brian Schmidt. An End to End Software Only Scalable Video Delivery System. In *Proceedings of the Workshop on Network and Operating Systems for Digital Audio and Video (NOSSDAV 95*, 1995.

[2] Kevin Fall. *A Peer-to-Peer I/O System in Support of I/O Intensive Workloads.* PhD thesis, University of California at San Diego, 1994.

[3] Kevin Fall and Joseph Pasquale. Improving Continuous-Media Playback Performance with In-Kernel Data Paths. In *Proceedings of the First International IEEE Conference on Multimedia Computing and Systems*, pages 100–109, 1994.

[4] D.James Gemmell, Harrick M.Vin, Dilip D.Kandlur, P.Venkat Rangan, and Lawrence A.Rowe. Multimedia Storage Servers: A tutorial. *IEEE Computer*, pages 40–49, May 1995.

[5] Howard P. Katseff and Bethany S.Robinson. Predictive Prefetch in the Nemesis Multimedia Information Service. In *ACM Multimedia Proceedings*, pages 201–209, 1993.

[6] Gregory K.Wallace. The JPEG Still Picture Compression Standard. *Communications of the ACM*, 34(4), 4 1991.

[7] A.L. Narasimha Reddy and James C.Wyllie. IO Issues in a Multimedia System. *IEEE Computer*, pages 69–74, March 1994.

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

[9] 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 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 15: PPC 7100/80 and Quadra 950/33, 320 $\times$ 240 – Played off Disk, MonitorI

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
V 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 16: PPC 7100/80 and Quadra 950/33, 160 $\times$ 120 – Played off Disk, MonitorI

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 17: PPC 7100/80 and Quadra 950/33, 320 × 240 – Played from RAM, MonitorI

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

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	2386	29.83	0.24	1967	24.59	1.39
C 30 3 half 100	55288196	41176196	2400	2388	29.85	0.11	1946	24.33	3.44
C 30 5 half 100	53200812	39088812	2400	2360	29.50	0.30	2017	25.21	2.21
C 30 10 half 100	51599160	37487160	2400	2365	29.56	0.21	2056	25.70	1.37
V 30 1 half 100	138782648	124670648	2400	2349	29.36	0.93	1153	14.41	2.12
V 30 3 half 100	126319334	112207334	2400	2358	29.48	0.48	930	11.62	25.74
V 30 5 half 100	123699495	109587495	2400	2365	29.56	0.46	1194	14.93	9.70
V 30 10 half 100	121666111	107554111	2400	2357	29.46	0.50	1276	15.95	10.29
J 30 1 half 100	94238136	80126136	2400	251	3.14	0.29	142	1.77	0.22
C 15 1 half 100	39892376	25780376	1200	1200	15.00	0.00	1186	14.82	0.17
C 15 3 half 100	34932804	20820804	1200	1200	15.00	0.00	1183	14.79	0.13
C 15 5 half 100	33894160	19782160	1200	1200	15.00	0.00	1195	14.94	0.12
V 15 1 half 100	76406147	62294147	1200	1200	15.00	0.00	1016	12.70	1.21
V 15 3 half 100	71468196	57356196	1200	1197	14.96	0.02	1036	12.95	0.96
V 15 5 half 100	70419248	56307248	1200	1199	14.99	0.01	1053	13.16	0.90
J 15 1 half 100	109158425	95046425	1200	194	2.42	0.44	75	0.94	0.13

Figure 19: PPC 7100/80 and Quadra 950/33, 320×240 – Played off Disk, MonitorII