Burkhard K. Neidecker-Lutz
Robert Ulichney

Software Motion Pictures

Software motion pictures is a method of generating digital video on general-purpose desktop computers without using special decompression hardware. The compression algorithm is designed for rapid decompression in software and generates deterministic data rates for use from CD-ROM and network connections. The decompression part offers device independence and integrates well with existing window systems and application programming interfaces. Software motion pictures features a portable, low-cost solution to digital video playback.

The necessary initial investment is one of the major obstacles in making video a generic data type, like graphics and text, in general-purpose computer systems. The ability to display video usually requires some combination of specialized frame buffer, decompression hardware, and a high-speed network.

A software-only method of generating a video display provides an attractive way of solving the problems of cost and general access but poses challenging questions in terms of efficiency. Although several digital video standards either exist or have been proposed, their computational complexity exceeds the power of most current desktop systems.[1] In addition, a compression algorithm alone does not address the integration with existing window system hardware and software.

Software motion pictures (SMP) is both a video compression algorithm and a complete software implementation of that algorithm. SMP was specifically designed to address all the issues concerning integration with desktop systems. A typical application of SMP on a low-end workstation is to play back color digital video at a resolution of 320 by 240 pixels with a coded data rate of 1.1 megabits per second. On a DECstation 5000 Model 240 HX workstation, this task uses less than 25 percent of the overall machine resources.

Together with suitable audio support (audio support is beyond the scope of this paper), software motion pictures provides portable, low-cost digital video playback.

The SMP Product

Digital supplies SMP in several forms. The most complete version of SMP comes with the XMedia Toolkit. This toolkit is primarily designed for developers of multimedia applications who include the SMP functionality inside their own applications. Figure 1 shows the user controls as displayed on a workstation screen. SMP players are also available on Digital's freeware compact disc (CD) for use with Alpha AXP workstations running the DEC OSF/1 AXP operating system. In addition, SMP playback is included with several Digital products such as the video help utility on the SPIN (sound picture information networks) application, as well as other vendors' products, such as the MediaImpact multimedia authoring system.[2]

In the XMedia Toolkit, access to the SMP functions is possible through X applications, command line utilities, and C language libraries. The applications and utilities support simple editing operations, frame capture, compression, and other functions. Most of these features are intended for use by producers of simple file formats called SMP clips.

The decompression functionality is offered as an X toolkit widget that readily integrates into the Open Software Foundation's (OSF) Motif-based applications. Multiple SMP codecs (compressors/decompressors) on a given screen all share the same color resources with one another and with the Display PostScript X-server extension, which is offered by all major workstation vendors. It also plays well with the standard color allocations used in the Macintosh QuickDraw rendering system and Microsoft Windows standard color allocations.

To facilitate flexible but simple access to entire films of SMP frames, SMP defines SMP clips. Rather than publishing that file format directly, all applications and widgets are accessed through an encapsulating library. This method allows future releases to have application-transparent changes to the underlying file structure and completely different ways to store and obtain SMP frames.

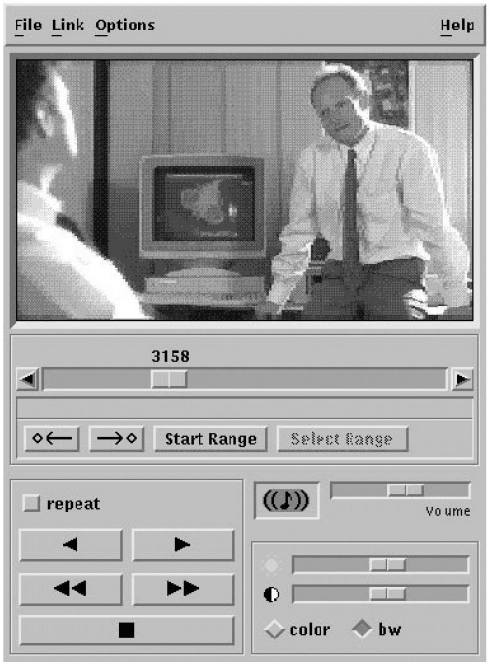

Figure 1 User Controls as Displayed on the Workstation Screen

An example of the latter is the storage of SMP clips directly in a relational database system in which no files exist, such as SQL Multimedia. The video data is stored directly in database records, and the client receives the data through the standard remote database access protocols. At the receiving client, the SMP clip library is used to generate a virtual SMP clip for the application program by substituting a new read function.

The SMP product also contains image converters that translate to and from the popular PBMPLUS family of image formats, allowing import and export to about 70 different image formats, including the Digital Document Interchange Format (DDIF). This allows the use of almost any image format as input for creation of SMP clips.

Historical Background and Requirements

In 1989 Digital's Distributed Multimedia Group experimented briefly with an algorithm called color cell compression (CCC) that had been described in 1986 by Campbell et al.[3] CCC is a coding method that is optimized for rapid decompression of color images in software. We built a demonstrator that rapidly displayed CCC-coded images in a loop to create a motion video effect. The demonstrator then served as our study vehicle to create a usable product for digital video playback.

Performing digital video entirely in software would stress the systems at all levels (I/O, processor, and graphics), so we needed to establish upper bounds for what we could hope to achieve with our desktop systems and workstations.

From the user's perspective, large sizes and high frame rates are desirable. These features need to be balanced with the limitations of real hardware. We modeled the data path through which digital video would have to flow in the system and measured the available resources on the slowest system we would use, a DECstation 2100. This workstation has a 12.5-megahertz (MHz) MIPS R2000 processor and a simple, 8-bit color frame buffer.

By merging this measurement with user feedback concerning the smallest acceptable image size and frame rate, we set our performance goal to play back movies of size 240 by 320 on the slowest DECstation processor with an 8-bit display at 15 frames per second. Smaller viewing sizes are almost invisible on a typical high-resolution workstation screen.

We settled for a frame rate of 15 frames per second. This rate is reasonably smooth: to the human eye, it appears as motion rather than separate images. It can be generated easily from 30-frame source material, such as standard video used in North America and Japan, by taking every other frame. Consequently, on the DECstation 2100 we would have at most

$$\frac{12.5 \times 10^6 \text{ clock cycles/second}}{(320 \times 240 \times 15) \text{ pixels/second}} = 10.85 \text{ clock cycles per pixel}$$

Thus, we must average no more than (approximately) ten machine instructions to decode and render each pixel to the screen.

In order to set our target for compression efficiency, we looked at the volume of data and possible distribution methods. CD-ROM looked promising, and this data rate was also chosen by the Motion Picture Experts Group (MPEG)-1 standard.[4] Hence our coded data rate goal was to maintain

a coded data rate for this size and frame rate that would allow playback from a CD-ROM. To achieve this goal, we limited the coded data rate for the video component to 135 to 142 kilobytes per second for video, leaving 8 to 15 kilobytes per second for audio. In addition, we had to limit fluctuations of the coded data rate to allow sensible use of bandwidth reservation protocols for playback over a network without complex buffering schemes.

More interesting were the issues that became apparent when we attempted to use the prototype for real applications. The digital video material had to be usable on a wide range of display types, and due to its large volume, keeping specialized versions for different displays was prohibitive. We would have to adapt the rendition of the coded material to the device-dependent color capabilities of the target display at run time.

Our design center used 8-bit color-mapped displays. These were (and still are) the most common color displays, and the demonstrator was based on them. Integration of the video into applications in a multitasking environment necessitated that computational as well as color resources were available for use by other applications. The system would have to perform cooperative sharing of the scarce color resources on displays with limited color capabilities.

From the perspective of portability, we needed to conform to existing X11 interfaces, without any hidden back doors into the window system. The X Window System affords no direct way of writing into the frame buffer. Rather, the MITSHM extension is used to write an image into a shared memory segment, and then the X server must copy it into the frame buffer. This method would impact our already strained CPU budget for the codec operation. We would need to decompress video in our code and have the X server perform a copy operation of the decompressed video to the screen, again using the main CPU. Quick measurements showed that the copy alone would use approximately 50 percent of the CPU budget for an 8-bit frame buffer, and another 5 to 10 percent would be used by reading the coded data from I/O devices.

With approximately five clock cycles per pixel yet to be rendered, it became clear why none of the standard video algorithms was of any use for such a task. We went back to the original CCC algorithm and started the development of software motion pictures.

Comparison with Other Video Algorithms

Today (early 1993), a number of digital video compression algorithms are in use. All of them are guarded closely as proprietary and therefore closed, and only one algorithm predates the development of SMP. Although we could not build on experiences with these for our work, we believe the internal working on most of them is similar to SMP with some additions.

A popular method for video compression is frame differencing. Rather than each frame being encoded separately, only those parts of the images that have changed relative to a preceding (or future) frame are encoded (together with the information that the other blocks did not change). This method works well for some input material, for example, in video conferences where the camera does not move. The method fails, however, on almost all other video material.

To enable frame differencing on a wider range of input scenes, a method known as motion estimation is used by some algorithms. The encoder for an image sequence performs a search for blocks that have moved between frames and encodes the motion. This search step is computationally very expensive and usually defeats real-time encoding, even for special-purpose hardware.

One of the earliest algorithms was digital video interactive (DVI) from Intel/IBM. It comes in two variations, real-time video (RTV) and production level video (PLV). RTV uses an unknown block encoding scheme and frame differencing. PLV adds motion estimation to this. RTV is comparable to SMP in compression efficiency, computationally more expensive, and much worse in image quality. PLV cannot be done in software and requires special-purpose supercomputers for compression. Compression efficiency of PLV is about twice as good as SMP, and image quality is somewhat better. The more recent INDEO video boards from Intel use RTV.

In 1992 Apple introduced QuickTime, which contains several video compression codecs. The initial RoadPizza (RP) video codec uses simple frame differencing and a block encoding similar to CCC, but without the color quantization step. (This is a guess based on the visual appearance and performance characteristics.) Compression efficiency of RP is three times worse than SMP, and image quality is comparable on 24-bit displays and much worse than SMP on 8-bit displays. Performance is

difficult to compare since SMP does not yet run on Macintosh computers.

The newer Compact Video (CV) codec introduced in QuickTime version 1.5 is similar to CCC with frame differencing and has compression efficiency much closer to SMP. Image quality on 8-bit displays is still lower than SMP, and compression times are almost unusable (i.e., long).

The newest entry into the market for software video codecs is the video 1 codec in Microsoft's Video for Windows product. Very little is known about it, but it seems to be close to CCC with frame differencing. Finally, Sun Microsystems has included CCC with frame differencing in their upcoming version of the XIL imaging library.

Three well-known standards for image and video compression have been established by the Joint Photographic Experts Group (JPEG) and the Motion Picture Experts Group (MPEG) committees of the International Organization for Standardization (ISO) and by the Comité Consultatif Internationale de Télégraphique et Téléphonique (CCITT). These standards are computationally too expensive to be performed in software in all but the most powerful workstations today.

The Algorithm

The SMP algorithm is a pixel-based, lossy compression algorithm, designed for minimum CPU loading. It features acceptable image quality, medium compression ratios, and a totally predictable coded data rate. No entropy-based or computationally expensive transform-based coding techniques are used. The downside of this approach is a limited image quality and compression ratio; however, for a wide range of applications, SMP quality is sufficient.

Block Truncation Coding

In 1978, the method referred to as block truncation coding (BTC) was independently reported in the United States by Mitchell, Delp, and Carlton and in Japan by Kishimoto, Mitsuya, and Hoshida.[3,5,6,7]

BTC is a gray-scale image compression technique. The image is first segmented into 4 by 4 blocks. For each block, the 16-pixel average is found and used as a threshold. Each pixel is then assigned to a high or a low group in relation to this threshold. An example of the first stage in the coding process is shown in Figure 2a, in which the sample mean is 101. Each pixel in the block is thus truncated to 1 bit, based on this threshold (see Figure 2b).

12	14	15	23
62	100	201	204
190	195	240	41
20	48	206	45

(a) The average of these 16 pixels is 101.

0	0	0	0
0	0	1	1
1	1	1	0
0	0	1	0

(b) The average of 101 is used as a threshold to segment the block.

Figure 2 Block Truncation Coding of a 4 by 4 Block

For each of the two groups, the average is then calculated again, giving a low average, a, and a high average, b. Mathematically, the first and second statistical moments of the block are preserved. Therefore, for a block of m pixels, with q pixels greater than the sample mean $\bar{x}^2$, and sample variance $\bar{\sigma}^2$, it can be shown that

$$a = \bar{x} - \bar{\sigma}\sqrt{q/(m-q)}$$
$$b = \bar{x} + \bar{\sigma}\sqrt{(m-q)/q}$$

More intuitively, the bit mask represents the shape of things in the block, and the average luminance and contrast of the block contents are preserved. With this coding method, for blocks of 4 by 4 pixels and 8-bit gray values, a 16-bit mask and two 8-bit values encode the 16 pixels in 32 bits for a rate of 2.0 bits per pixel.

Color Cell Compression

Lema and Mitchell first extended BTC to color by employing a luminance-chrominance space.[8] However, the direction taken by Campbell et al. was computationally faster for decode.[3] In this approach, a luminance value is computed for each pixel. As in the BTC algorithm, the sample mean of the luminance in each 4 by 4 block is used to segment pixels into low and high groups based on luminance values only. The 24-bit color values assigned to the low and high groups are found by independently solving for the 8-bit red, green, and

blue values. This allows each block to be represented by a 16-bit mask and two 24-bit color values, for a coding rate of 4 bits per pixel.

The 24-bit values are mapped to a set of 256 8-bit color index values by means of a histogram-based palette selection scheme known as the median cut algorithm.[9] Thus every block can be represented by two 8-bit color indices and the 16-bit mask, yielding 2 bits per pixel; however, each image frame must also send the table of 256 24-bit color values.

Software Motion Pictures Compression

With our goal of 320 by 240 image resolution playback at 15 frames per second, straight CCC coding would have resulted in a data stream of more than 292 kilobytes per second, which is well beyond the capabilities of standard CD-ROM drives. Thus SMP needed to improve the compression ratio of CCC approximately twofold.

Given that we could not apply any of the more expensive compression techniques, we looked for computationally cheap data-reduction techniques. Since most of these techniques negatively impact image quality, we needed a visual test bed to judge the impact of each change.

We computed the images off-line for a short sequence, frame by frame, and then preloaded the images into the workstation memory. The player program then moved the images to the frame buffer in a loop, allowing us to view the results as they would be seen in the final version. The use of this technique provided two advantages. First, we could discover motion artifacts that were invisible in any individual frame. Second, we could judge the covering aspects of motion, which tends to brush over some defects that look objectionable in a still frame.

At first, interframe or frame difference coding looked like a reasonable technique for achieving better compression results without sacrificing image quality, but this was highly dependent on the nature of the input material. Due to the low CPU budget, we could not use any of the more elaborate motion compensation algorithms, so even slight movements in the input video material largely defeated frame differencing. Typically, we achieved only 10 percent better compression with interframe coding, while introducing considerable complexity to the compression and decoding operations. As a result, we dropped interframe coding and made SMP a pure intraframe method, simplifying editing operations and random access to digitized material. At the same time, this opened up use of SMP for still image applications.

To reach our final compression ratio goal of approximately 1 bit per pixel, we settled for a combination of two subsampling techniques. Similar techniques have been independently described by Pins, who conducted an exhaustive search and evaluation of compression techniques.[10] His findings served as a check on our experiments.

Blocks with a low ratio of foreground-to-background luminance (a metric that can be interpreted as contrast) are represented in SMP by a single color and no mask. This reduces the coded representation to a single byte compared to 4 bytes in CCC, which amounts to a fourfold subsampling of such blocks. No chrominance information enters into this decision. It is surprising, but even very marked chrominance differences in foreground/background pairs are readily accepted by the human eye.

With the introduction of a second kind of block, additional encoding information was necessary to distinguish normal (structured) CCC blocks from the subsampled (flat) blocks. In the SMP encoding, this is handled by a bitmap with one bit flagging each block.

Because the adaptive subsampling alone did not yield enough data reduction for our compression goal, we added fixed subsampling for the structured blocks. The horizontal resolution of the structured blocks in SMP is halved relative to CCC by horizontally averaging two neighboring pixels, which reduces the number of bits in the mask from 16 to 8. This reduction leads to blurred vertical edges but looks reasonable for natural video images. Fixed subsampling allowed the encoding of structured blocks with 3 bytes instead of 4 bytes.

We reapplied these ideas to the original gray-scale block truncation algorithm. We added a variation to the format that does not use a color look-up table but interprets the foreground and background colors directly as luminance values. Images compressed in this format code gray-scale input material more compactly (there is no need to transmit the leading color look-up table as in CCC); they also do not suffer from the quantization band effects inherent in the color quantization used in the CCC algorithm.

We varied the ratio of flat to structured blocks to effect a trade-off between image quality and compression ratio; however, the range of useful settings is relatively small. If too few structured blocks are allocated, the image essentially is scaled down

fourfold, which makes the image look very blocky. If too many structured blocks are allocated, regions of the image that have little detail are encoded with unnecessary overhead. Over the wide range of images we tested, allocating between 30 percent and 50 percent of structured blocks worked best, yielding a bit rate of 0.9 to 1.0 bits per pixel. For color images, the overhead of the color table (768 bytes) must be added.

Decompression

The most challenging part of the design of the SMP system, given the performance requirements, is the decompression step. Efficient rendering techniques of block-truncation coding are well known for certain classes of output devices.[3] SMP improves on the implementations described in the literature by complementing the raw algorithm with efficient, device-independent rendering engines.[3,5,8,10,11] To maximize code efficiency, a separate decompression routine is used for each display situation, rather than using conditionals in a more generic routine. The current implementation can render to 1-, 8-, and 24-bit displays.

Decompression of BTC involves filling 4 by 4 blocks of pixels with two colors under a mask. Because the size and alignment of the blocks is known, a very fast, fully unrolled code sequence can be used. Changes of brightness and contrast of the image can be rapidly adapted to different viewing conditions by manipulating the entries of the colormap of the SMP encoding. Most of the work lies in adaptation of the color content of the decompressed data to the device characteristics of the frame buffer.

For displays with full-color capabilities (24-bit true color), the process is straightforward. The main problem is performing the copy of the decompressed video to the screen. Since 24-bit data is usually allocated in 32-bit words, the amount of data to copy is four times the 8-bit case. Typically, SMP spends 90 percent of the CPU time in the screen copy on 24-bit systems.

The more common and interesting case is to decompress to an 8-bit color representation. Given that SMP is an 8-bit, color-indexed format, it would seem straightforward to download the SMP frame color table to the window system color table and fill the image with the pixel indices directly. This method is impractical for two reasons. First, most window systems (including X11) do not allow reservation of all 256 colors in the hardware color tables. Typically, applications and window managers use a few of the entries for system colors and cursors. Quantizing down to a smaller number of colors (such as 240) could overcome this drawback to a certain degree; however, it would make the SMP-coded material dependent on the device characteristics of a particular window system.

The second and much more problematic aspect is that the SMP frames in a sequence usually have different color tables. Consequently, each frame requires a change of color table that causes a kaleidoscopic effect for the windows of other applications on the screen. In fact, flashing cannot be eliminated within the SMP window itself.

Neither X11 nor other popular window systems such as Microsoft Windows allow reload of the color table and the content of an image at the same time. Therefore, regardless of whether the color table or image contents is modified first, a flashing color effect takes place in the SMP window. It may seem that the update would have to be done in a single screen refresh time as opposed to simultaneously. This is true but irrelevant. Most window systems do not allow for such fine-grain synchronization; and for performance reasons, it was unrealistic to expect to be able to update the image in a single, vertical blanking period.

Alternative suggestions to avoid this problem have been proposed in the literature. One suggestion is to use a single color table for the entire sequence of frames.[10,11] This method is computationally expensive and fails for long sequences and editing operations. Another proposes quantization to less than half of the available colors or partial updates of the color map and use of plane masks.[11] This alternative is not particularly portable between different window systems, and the use of plane masks can have a disastrous impact on performance for some frame-buffer implementations such as the CX adapter in the DECstation product line.

Neither of these methods addresses the issue of monochrome displays or the use of multiple simultaneous SMP movies on a single display. (This effect can be witnessed in Sun Microsystems' recent addition of CCC coding to their XIL library.) To keep device influence out of the compressed material and to enable the use of SMP on a wide range of devices and window systems, a generic decoupling step was added between the colors in the SMP frame and the device colors used for rendition on the screen.

A well-known technique for matching color images to devices with a limited color resolution is dithering. Dithering trades spatial resolution for an apparent increase in color and luminance resolution of the display device. The decrease in spatial resolution is less of an issue for SMP images because of their inherently limited spatial resolution capability. Thus the only challenge was the computational cost of performing dithering in real time.

Fortunately, we found a dithering algorithm that allowed both good quality and high speed.[12] It reduces quantization and mapping to a few table look-up operations, which have a trivial hardware implementation (random access memory) and a reasonable software implementation with a few adds, shifts, and loads.

The general software implementation of the dithering algorithm takes 12 instructions in the MIPS instruction set to map a single pixel to its output representation. For SMP decoding, two different colors at most are in each 4 by 4 block. With this distribution, the cost of dithering is spread over the 16 pixels in each block.

Another optimization used heavily in the 8-bit decoder is to manipulate 4 pixels simultaneously with a single machine instruction. This technique increases performance for decompressing and dithering to 3.2 instructions per pixel in the MIPS instruction set, including all loop overhead, decoding of the encoded data stream, and adjusting contrast and brightness of the image (2.7 instructions per pixel for gray-scale). This efficiency is achieved by careful merging of the decoding, decompression, and dithering phases into a single block of code and avoiding intermediate results written to memory. The cost of the 1-bit and 24-bit decoders is the same or lower (3.2 and 2.9 instructions per pixel, respectively).

Compression

The SMP compressor takes an input image, a desired coded image size, and an output buffer as arguments. It operates in five phases:

- Input scaling (optional)
- Block truncation (luminance)
- Flat block selection
- Color quantization (color SMP only)
- Encoding and output writing

Although the initial scaling is not strictly part of the SMP algorithm, it is necessary for different input sources. Fast scaling is offered as part of both the library and the command-line SMP compressors. Instead of simple subsampling, true averaging is used to ensure maximum input image quality.

The block truncation phase makes two passes through each 4 by 4 block of the input. The first pass calculates the luminance of each individual pixel and sums them to find the average luminance of the entire block. The second pass partitions the pixel pairs into the foreground and background sets and calculates their respective luminance and chrominance averages.

The flat-block-selection phase uses the desired compression ratio to decide how many blocks can be kept as structured blocks and how many need to be converted to flat blocks. The luminance difference of the blocks is calculated, and blocks in the low-contrast range are marked for transition to flat blocks. Because the total average was calculated for each block in the preceding phase, no additional calculations are needed for the conversion of blocks, and the mask is thrown away. Colors are entered into a search structure during this phase.

The color quantization phase uses a median cut algorithm, biased to ensure good coverage of the color contents of the image rather than minimize the overall quantization error. The bias method ensures that small, colored objects are not lost due to large, smoothly shaded areas getting the lion's share of the color allocations. These small objects often are the important features in motion sequences and have a high visibility despite their small size.

The final encoding phase builds the color table and matches the foreground/background colors of the blocks to the best matches in the chosen color table.

The gray-scale compression can be much faster because neither the quantization nor the matching step need be performed. Also, only one-third of the uncompressed video data is usually read in, making gray-scale compression fast enough to enable real-time compression on faster workstations and video-conferencing type applications.

This speed is partly due to the 8-bit restriction in the mask of each structured block. This restriction permits the algorithm to store all intermediate results of the block truncation step in registers on typical reduced instruction set computer (RISC) machines with 32 registers. The entire gray-scale

compression algorithm can be done on a MIPS R3000 with 8 machine instructions per input pixel on average, all overhead (except input scaling) included.

Unfortunately, for color processing, SMP compression remains an off-line, non-real-time process, albeit a reasonably fast one at 220 instructions per pixel. A 25-MHz R3000 processor can process more than 40,000 frames in 24 hours (DECstation 5000 Model 200, 320 by 240 at 15 frames per second, TX/PIP as frame grabber), equivalent to 45 minutes of compressed video material per day. The more recent DEC 3000 AXP Model 500 workstation improves this number threefold, so special-purpose hardware for compression is unnecessary even for color SMP.

Portability

A crucial part of the SMP design for portability is the placement of the original SMP codec on the client side of the X Window System. This allows porting and use of SMP on other systems, without being at the mercy of a particular system vendor for integration of the codec into their X server or window system.

This placement is enabled by the efficiency of the SMP decompression engine, which allows many spare cycles for performing the copy of the decompressed, device-dependent video to the window system.

Currently, SMP is offered as a product only on the DECstation family of workstations, but it has been ported to a variety of platforms, including

- DEC AXP workstations running the DEC OSF/1 AXP operating system
- Alpha AXP systems running the OpenVMS operating system
- DECpc AXP personal computers running the Windows NT AXP operating system
- VAX systems running the VMS operating system
- Sun SPARCstation
- IBM RS/6000 system
- HP/PA Precision system
- SCO UNIX/Intel
- Microsoft Windows version 3.1

Generally, porting the SMP system to another platform supporting the X Window System requires the selection of two parameters (host byte order and presence of the MITSHM extension) and then a compilation. The same codec source is used on all the above machines; no assembly language or machine-specific optimizations are used or needed.

The port to Microsoft Windows shows that the same base technology can be used with other window systems, although parts specific to the window system had to be rewritten. The codec code is essentially identical, but the extreme shortage of registers in the 80x86 architecture and the lack of reasonable handling of 32-bit pointers in C language under Windows warrant a rewrite in assembly language on this platform. We do not expect this to be an issue on Windows version 3.2, due to be released later in 1993.

Conclusion

Software motion pictures offers a cost-effective, totally portable way of bringing digital video to the desktop without requiring special investments for add-on hardware. Combined with audio facilities, SMP can be used to bring a complete video playback to most desktop systems. The algorithm and implementation were designed to be used from CD-ROMs as well as network connections. SMP seamlessly integrates with the existing windowing system software. Because of its potentially universal availability, SMP can serve an important function as the lowest common denominator for digital video across multiple platforms.

Acknowledgments

We would like to thank all the people who have contributed to making software motion pictures a reality. Particular thanks go to Paul Tallett for writing the original demonstrator and insisting on the importance of a color version. He also implemented the VMS versions. Thanks also to European External Research for making the initial research and later product transition possible. Last but not least, thanks to Susan Angebranndt and her engineering team for their help and confidence in this work.

References

1. Special Issue on Digital Multimedia Systems, *Communications of the ACM,* vol. 34, no. 4 (April 1991).

2. L. Palmer and R. Palmer, "DECspin: A Networked Desktop Videoconferencing Application," *Digital Technical Journal,* vol. 5, no. 2 (Spring 1993, this issue): 65–76.

3. G. Campbell et al., "Two Bit/Pixel Full Color Encoding," *SIGGRAPH'86 Conference Proceedings,* vol. 20, no. 4 (1986): 215–223.

4. D. LeGall, "MPEG: A Video Compression Standard for Multimedia Applications," *Communications of the ACM,* vol. 34, no. 4 (April 1991): 47–58.

5. O. Mitchell, E. Delp, and S. Carlton, "Block Truncation Coding: A New Approach to Image Compression," *Conference Record, IEEE International Conference Communications,* vol. 1 (June 1978): 12B.1.1–12B.1.4.

6. T. Kishimoto, E. Mitsuya, and K. Hoshida, "A Method of Still Picture Coding by Using Statistical Properties" (in Japanese), *Proceedings of the National Conference of the Institute of Electronics and Communications Engineers of Japan,* no. 974 (March 1978).

7. E. Delp and O. Mitchell, "Image Compression Using Block Truncation Coding," *IEEE Transactions on Communications,* vol. COM-27 (1979): 1335–1342.

8. M. Lema and O. Mitchell, "Absolute Moment Block Truncation Coding and Its Application to Color Images," *IEEE Transactions on Communications,* vol. COM-32, no. 10 (1984): 1148–1157.

9. P. Heckbert, "Color Image Quantization for Frame Buffer Display," *Computer Graphics (AMC SIGGRAPH'82 Conference Proceedings),* vol. 16, no. 3 (1982): 297–307.

10. M. Pins, "Analyse und Auswahl von Algorithmen zur Datenkompression unter besonderer Berücksichtigung von Bildern und Bildfolgen," Ph.D. thesis, University of Karlsrühe, 1990.

11. B. Lamparter and W. Effelsberg, "Digitale Filmübertragung und Darstellung im X-Window-System," Lehrstuhl für Praktische Informatik IV, University of Mannheim, 1991.

12. R. Ulichney, "Video Rendering," *Digital Technical Journal,* vol. 5, no. 2 (Spring 1993, this issue): 9–18.