Multimedia based on object models: Some whys and hows

by V. M. Bove, Jr.

In this paper I describe some of the design issues and research questions associated with object-based video coding algorithms, as well as the new applications made possible. I propose a hardware and software strategy to cope with the computational demands (stream-based computing combined with automatic resource management) and also briefly introduce object-based audio representations that are linked to the video representations.

While the computational requirements for the current round of multimedia standards and applications are proving manageable, several trends can be identified that will dramatically increase the processing demands. One of these will be a shift to an object-based representation, in which video of real scenes is described not as sequences of frames but rather as collections of modeled objects that are encoded by machine-vision algorithms and decoded according to scripting information. Although the shift from images to models has to date largely taken place among researchers seeking significantly more compression than is available from standard coders, the most significant impact may be the new forms of interactivity and personalization these representations enable.

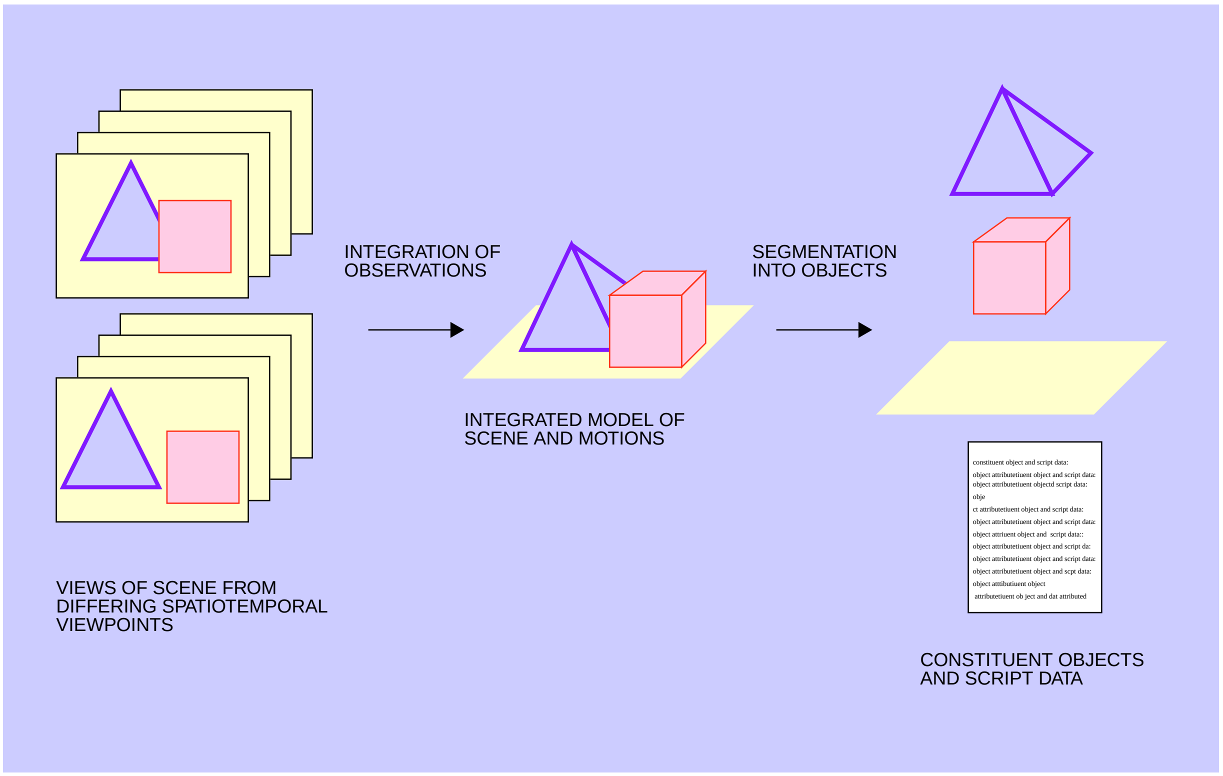
Future multimedia

It is understandable how one might conclude—seeing the array of products dedicated to computer-moderated authoring and playback of digital audio and video—that nearly all the algorithmic and infrastructure problems of multimedia are either solved or nearly solved. But the transform-based, or software-optimized algorithms on which these systems and applications are based are just the first round in a move toward potentially much more useful, efficient, and computationally demanding forms of communication.

Much of current multimedia is essentially a more efficient extrapolation of past analog video and audio applications. Users of multimedia systems certainly desire compression efficiency (or equivalently, more quality at a given bit rate), but taking useful advantage of the computational abilities of multimedia systems in more than a signal-processing sense will require more semantics. A representation that segments information in a manner that is content-driven rather than arbitrary (e.g., localized, modeled sound sources and acoustical environments rather than speaker channels; coherent objects rather than blocks of pixels) might not only achieve more compression,[1] but could also make relevant features apparent in the compressed bit stream. Potential benefits include:

- New production and post-production methods
- Intelligent database search
- Easier authoring of interactive or personalized content
- Better support for distributed storage
- Assembly of content "on-the-fly" from disparate elements

Figure 1 An object-based video encoding process


INTEGRATION OF OBSERVATIONS
SEGMENTATION INTO OBJECTS
INTEGRATED MODEL OF SCENE AND MOTIONS
constituent object and script data:
object attributetiuent object and script data:
object attributetiuent objectd script data:
obje
ct attributetiuent object and script data:
object attributetiuent object and script data:
object attriuent object and script data::
object attributetiuent object and script da:
object attributetiuent object and script data:
object attributetiuent object and scpt data:
object atttibutiuent object
attributetiuent ob ject and dat attributed
VIEWS OF SCENE FROM DIFFERING SPATIOTEMPORAL VIEWPOINTS
CONSTITUENT OBJECTS AND SCRIPT DATA

Object-based video

Let us consider in some detail the case of video compression. Most current algorithms are based on frames, groups of frames, and regular subsections of frames. Compression inefficiency is traded for computational efficiency and regularity, by not taking into account the actual structure of the scene represented by the video frames. Because objects in the world do not correspond to regular subarrays of pixels, and because *x-y* translation does not correspond very well to an image-plane projection of their motion in three-dimensional space, there is a fundamental model mismatch between the world and the video data. Although some clever research has been done on interpreting MPEG (Moving Pictures Experts Group) formatted bit streams to find scene changes and object or camera motions,[2] or performing simple modifications to the video data without fully decoding the bit stream,[3] ultimately semantics and flexibility should be designed-in features of a video representation and not serendipitous afterthoughts.

While most researchers in the field are searching for more efficient compression, these other considerations are also driving the recent interest in model-based or analysis-synthesis video representations, in which moving scenes are represented as component objects that are reassembled according to scripting information to produce images for viewing. A number of methods are currently under investigation, including segmented coders that identify coherent two-dimensional regions or layers by examining motion[4,5] (or equivalently, color or texture[6]), and the fitting of three-dimensional object models to image sequences.[7] The added compression efficiency of structured video generally comes about because the more accurate transmitted model and the greater computational ability of the receiver permit better prediction. In layered or region-segmented two-dimensional (2D) coders, for example, the transmitted motion parameters are intended to be a more correct approximation for the image-plane projections of objects moving in space than would result from (*x*,*y*) vectors computed on an arbitrary square grid. Added memory in the decoder also eliminates the need to retransmit information about occluded and revealed regions.[8] Yet more computationally intensive algorithms can under certain circumstances estimate camera or object motion in space.[9] Much more research remains to be done in the machine-vision-style scene analysis methods for these higher-level coders, as well as appropriate statistical coders for the resulting objects.

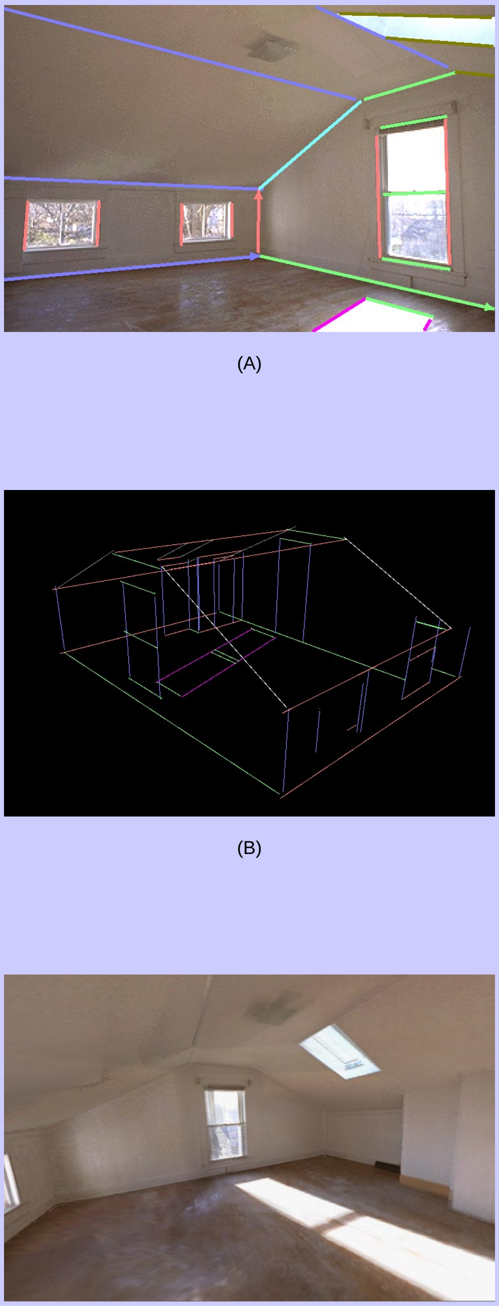
Analysis

There are two fundamental—and perhaps at first glance contradictory—analysis operations needed in an object-based video encoding process: observation integration and information segmentation (see Figure 1). In Figure 1, the encoding process for object-based video can be seen as integration of information from frames taken at differing times and viewpoints into a scene model, which then can be segmented into objects and scripting data. Editing or post-production operations are performed on the output of this process. These operations are best considered not as totally independent steps, but rather as cooperating with each other, possibly inseparably as in some iterative algorithms. The *observation integration* operation involves the finding of correspondences among observations of a moving scene, whether from one camera at different times, spatially separated simultaneous cameras, or current observations and *a priori* knowledge. Many machine-vision techniques (e.g., structure-from-motion, stereopsis, depth-from-focus, motion modeling) can be seen as examples of this concept. This integration process does not have to operate only on the input video frames, but might be aided by other cues such as instrumented cameras that can sense position or acceleration.[10] The *information segmentation* operation seeks also to find correspondences, but among sections of the integrated information that seem to belong to coherent objects in space. For instance, in a 2D layered video coding algorithm like that of Adelson and Wang[4] the finding of optical flow fields mapping one frame into the next might be seen as the integration stage, while applying a higher level motion model and grouping together pixels that share common parameters therein is the segmentation stage.

Two projects from the Television of Tomorrow consortium's recent research further illustrate the segmentation and integration aspects of object-based video, and also provide good examples of what we are calling human-supervised scene analysis. In this analysis we avoid having to solve the entire "vision problem" by asking a user to supply a small amount of starting information and perhaps a continuing reasonableness check while the encoding system runs. Such methods, of course, are more appropriate for media that undergo a post-production stage than for live video. An integration example is that of interpreting a group of uncalibrated 2D images of a static architectural scene (perhaps a movie set) as a single merged 3D model by using perspective effects to infer vanishing points, focal length, and camera orientation (Figure 2). Here a human can indicate in a rough sense the relative orientations of the 2D views one to another, greatly simplifying the algorithm's search space for feature correspondences.[11] In Figure 2, perspective is used to interpret uncalibrated 2D images as 3D. Figure 2A shows one view of a room with extracted edges overlaid. All similarly colored lines have been interpreted as parallel in a three-dimensional space. Figure 2B shows a wire-frame version of a model produced by merging information from five photographs. Figure 2C is a rendering of a synthetic viewpoint from the merged model.

Another case in which minimal human interaction helps greatly is making a video coder based on object segmentation. The question to be addressed is what is an object: a foreground, a person in the foreground, the person's shirt? Depending on the application, the answers are different (e.g., in authoring an interactive clothing catalog, "the person's shirt" might be desirable). Researchers have used motion or texture to segment video for added compression, but if the segmentation is contextually determined, and is to be used for something like "hot buttons," searchable databases, or on-the-fly assembly of content, then the segmentation may be a multidimensional function of parameters such as color, texture, motion, spatial coherence, and other factors. We have developed a software package that permits a content creator implicitly to indicate the segmentation for a video sequence by quickly dragging the cursor across representative points for each desired object for only one video frame (see Figure 3).[12] The system then finds regions throughout the video sequence that have corresponding color, texture, and motion. The underlying statistical model is capable of identifying regions even if they have multimodal distributions in one or more of these parameters (i.e., the system can correctly imply that a desired region is either red or yellow but not orange). In Figure 3, the top portion shows a frame of a video sequence with user-indicated points overlaid. The bottom portion of the figure shows the final result: a segmentation of the entire video

Figure 2 Information integration example where perspective is used to interpret uncalibrated 2D images as 3D, and to merge multiple views into a single 3D scene model


(A)

(B)

(C)

sequence based on motion, color, texture, and spatial coherence.

Synthesis

We do not anticipate that existing algorithms will disappear when newer ones are developed, nor do we suggest that there is a single best description for all applications or content. Thus we need something more flexible than a dedicated hardware solution, providing real-time computer graphics rendering and compositing capability as well as the digital signal processing required for undoing transforms and entropy coders, and handling error signals. Given a sufficiently powerful and flexible decoder, the way in which a scene is described and the forms of the constituent objects represent an originator-specified trade-off: increased encoding complexity, pipeline delay, and risk (some scenes simply may not lend themselves to high-level descriptions given current analysis algorithms), versus increased flexibility, semantics, compression, and quality. In order to drive our scene-analysis work in directions that are a good match to advanced multimedia applications, and to enable the prototyping of such applications and the tools for authoring them, we have for several years been examining frameworks for object-based video decoders.[13] A portion of this work has involved trying to identify a core set of information "objects" that might make up the compressed bit streams for a variety of encoding methods, as well as the operations that must be performed by the decoder. For the algorithms with which we have significant experience, we identify the following object types:

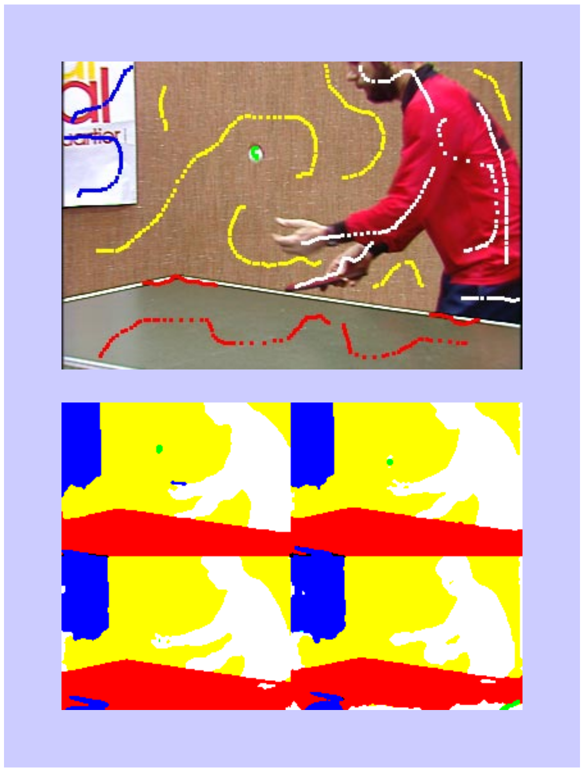
- 2D objects: These are arrays of pixels, possibly transparent in places to support layering.
- 2 1/2-D objects: These are 2D objects, with the addition of *z-buffers*, which provide a distance value for each pixel to be used in compositing. Effectively, a 3D object becomes a 2 1/2-D object after rendering.
- 3D objects: These are standard computer-graphics objects, which require rendering before viewing.
- Explicit transformations: These specify spatial remapping to be applied to objects. They may take forms such as a dense optical flow field, a sparse set of motion vectors, or a parametric warp.
- Error signals: This is an array of values that might be added to a rendered or transformed object, or to the entire composited image, as in a predictive coder. When a precomputed model is used to represent a scene with changing, directional lighting or

moving shadows, it may be more efficient to represent the error as partly multiplicative and partly additive; this is still an open area for research.

We have been considering a number of different decoder architectures, ranging from a “kit of parts” that could be connected in various ways, to a flexible pipeline, as illustrated in Figure 4. In every case the first step performed upon each set of data is decompression, which reverses transform or other (e.g., wavelet, fractal) coding, quantization, run-length coding, and variable-length coding as appropriate. The last step before display is a z-buffer compositing operation, which allows layering of multiple 2D objects or hidden-surface removal for 3D objects. Assembling the objects into a final image requires that they be accompanied by scripting information. In the case of the pipeline mentioned above, the script controls the operation of the functional blocks, the flow of data through them, and the configuration of the flexible data paths. This scripting language might be as simple as a bit field in a data packet header, or as complex as a programming language. Our current language, ISIS, is Scheme-like, but based on arrays rather than lists to simplify memory management, and with a number of special constructs such as a data type called a “timeline” that can have numeric values inserted at arbitrary real-numbered locations but can then be evaluated at any other real-number index by interpolation.[14] In any event, like PostScript**, the language will rarely be edited directly, being rather the result of a scene analysis algorithm or a content authoring tool. In interactive or personalized applications, the script control flow and operational parameters for functional blocks might depend on user actions or state variables (such as user identity, history, or display circumstances). The illustrated processing model should not be taken as a literal hardware architecture. We have implemented it as an application program on the Cheops system (described next) but other implementations such as specialized hardware, parallel processors, or software-only on general-purpose processors are equally possible.

As of this writing, several standardization efforts are moving in directions that address some of the issues I have outlined above. Although MPEG-4 is more concerned with working on very low bandwidth channels than with image quality or data flexibility, proposals have included support for object segmentation and multiple representation types. Virtual Reality Modeling Language (VRML) is first and foremost a language for describing virtual environments rather than efficiently coding real ones, but some of the proposed extensions echo related concerns.

Figure 3 Information segmentation example where user-supplied data for one frame result in a segmentation of an entire video sequence



Computational strategies

As might be imagined, such a scenario for video requires powerful and reconfigurable computing, but in a compact, inexpensive, and ideally easily programmable form. Examination of the characteristics of the data and the algorithms has led my research group to the use of the concept of *streams* in multimedia processing. The stream mechanism is a mapping of a multidimensional data array into an ordered one-dimensional sequence of data by means of an access pattern. In a stream-based system, one does not think of a processing element reading or writing memory. Instead, an access pattern turns a source array into a one-dimensional sequence that flows through a processing element and then (through an access pattern again) into a stored (or played, or displayed) destina-

Figure 4 **Model for a receiver capable of decoding compressed object-based video (from Reference 13). This diagram represents the image data-flow portion of the processing pipeline; script data, perhaps in conjunction with user interactions, control the operation of each stage. This process can decode a variety of coding methods from an ordinary hybrid predictive coder to high-level computer graphics descriptions of scenes. Purple data paths are inactive, while black data paths are active. Dashed data paths interconnect in an algorithm-dependent manner.**

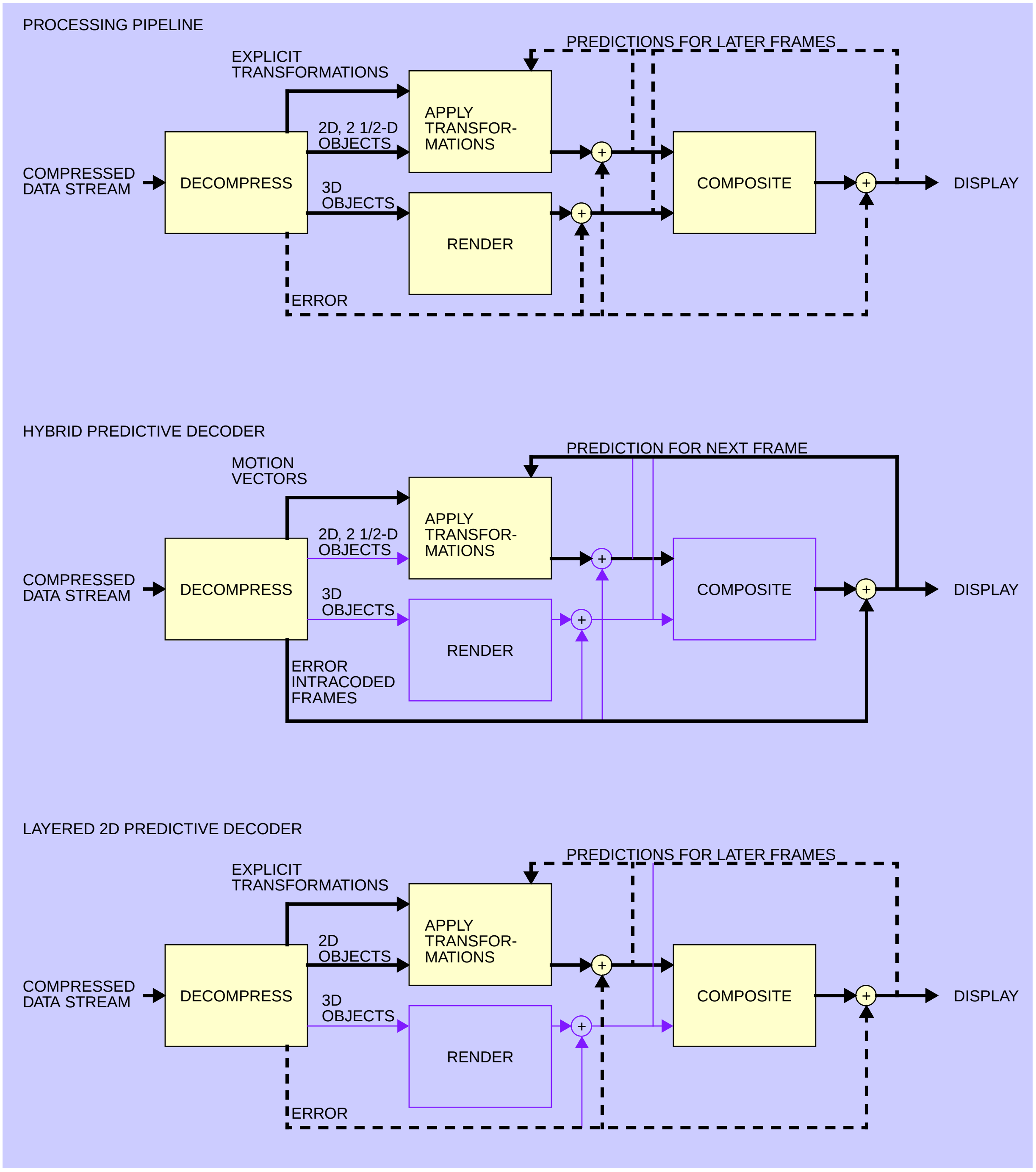
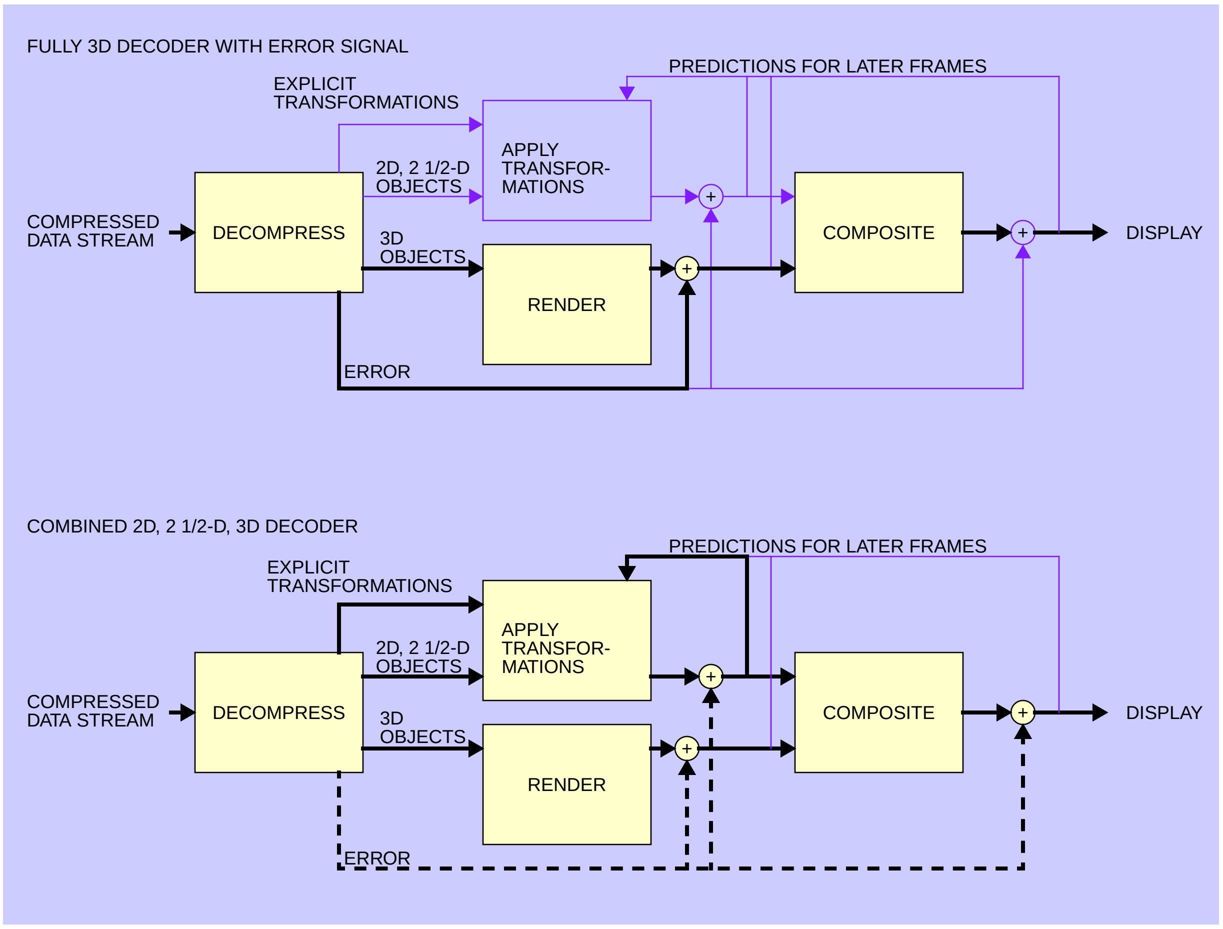


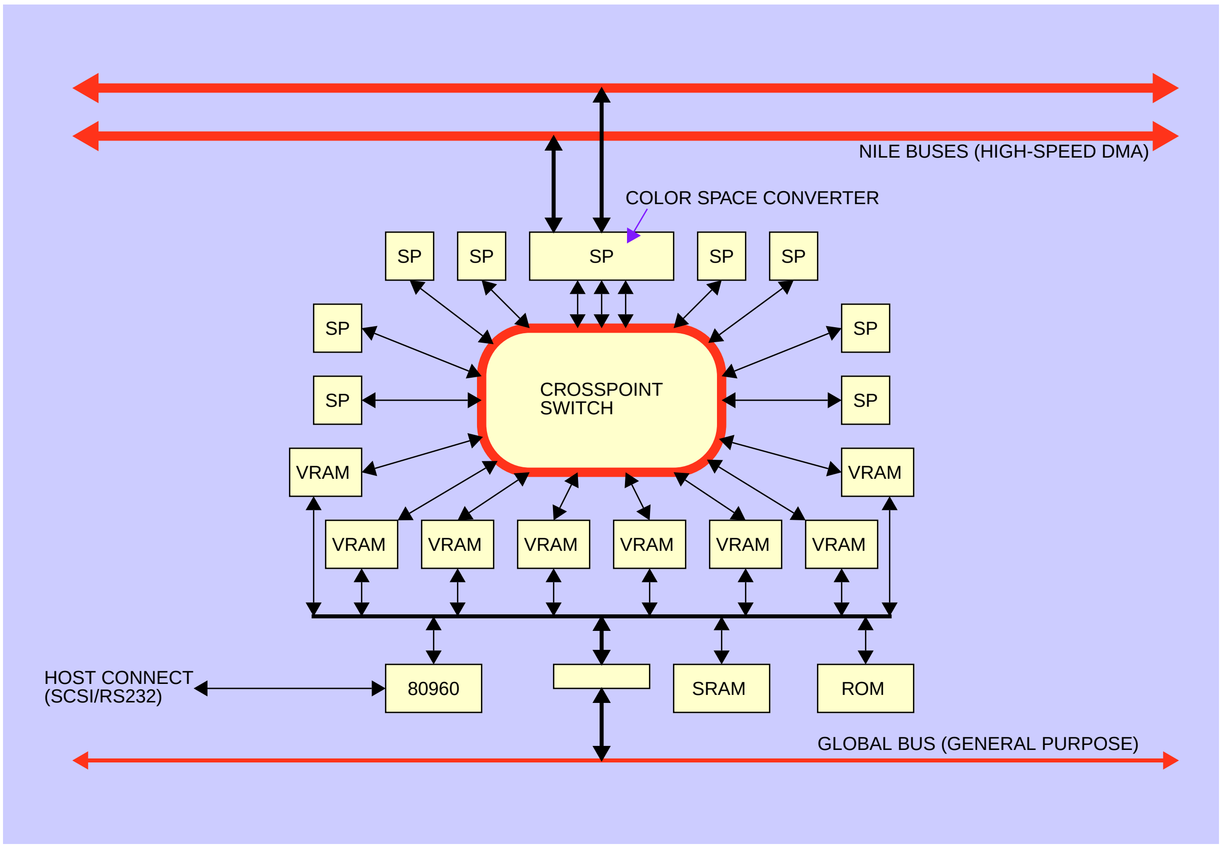
Figure 4 (continued)



tion array. The algorithm can therefore easily be described in terms of a graph structure connecting storage buffers with computational elements—thus process parallelism is expressed explicitly, and execution can be parallelized at run time, supporting multitasking and hardware scalability. In Reference 15, we explain how a suitable description of the dimensional mapping can also describe data parallelism by making explicit the subsections of larger arrays that can be processed independently if multiple suitable processing elements are available. Another advantage of such a representation is that graphical programming methods can be applied relatively easily. Additional efficiency is obtained by avoiding address calculation in the processing elements, which in many cases can pose a significant computational load. The stream mechanism is particularly well suited to algorithms where the same operations are applied to a large amount of data, such as those used in audio and video processing.

In implementation, a stream system may consist of specialized processors using hardware to implement the stream communications, or general-purpose processors in a shared-memory configuration using memory buffers to perform all communications. Or, ideally, it may combine general-purpose and specialized processors into a heterogeneous system using both stream implementations. Even machines with a single general-purpose processor may utilize the stream mechanism to their advantage. While such a machine is incapable of exploiting either control or data parallelism of the granularity provided by streams, performance improvements may still be

Figure 5 The Cheops processor connects specialized processors to DMA-equipped banks of memory through a crosspoint switch. Resource allocation and user interaction are handled by an Intel 80960CF processor. The Nile Buses are high-speed DMA channels used for transferring data from module to module.


NILE BUSES (HIGH-SPEED DMA)
COLOR SPACE CONVERTER
SP
CROSSPOINT SWITCH
VRAM
HOST CONNECT (SCSI/RS232)
80960
SRAM
ROM
GLOBAL BUS (GENERAL PURPOSE)

obtained as a result of the clearer definition of the data access patterns. The goals of a stream mechanism within the Von Neumann machine model are similar to those of compiler vectorization techniques. While unable to exploit the additional data parallelism provided by partitioning a stream, a single processor machine may nonetheless benefit from partitioning of large data sets such as images. An overview and bibliography on the topic of streams may be found in Reference 15.

Our first implementation of a stream-based system for video processing was Cheops,[16] which attempts to combine the efficiency of specialized hardware with the programmability of general-purpose processors. The Cheops processor module (Figure 5) is a board-level system containing up to eight heterogeneous specialized processing units (stream processors) connected through a full crosspoint switch to a set of eight memory units. Each memory unit is independently capable of sourcing or storing one stream, through the use of an integral multidimensional direct memory access (DMA) controller. A general-purpose processor (an Intel 80960CF) is used to execute both the resource manager and algorithm segments that do not map well onto the specialized stream processors provided. The crosspoint switch is semi-statically switched by the resource manager to configure a processing pipeline for a particular stream segment, while a separate hardware handshake mechanism is used to synchronize the actual stream flow.

The stream processors are optimized for performing a set of common operations from multidimensional digital signal processing (DSP) and computer graphics. One processor, for example, consists of eight 16-bit multiply-accumulate units, a tree of adders for combining the results, 64 words of local storage, and a programmable operation sequencer. It is used for convolution, matrix and vector multiplication, and simpler stream functions using multiplication or addition (such as scaling and mixing). Other processors are specialized for motion estimation, data controlled memory reading and writing (for motion compensation, image warping, or hidden surface removal), block transform coding, and basis vector superposition.

If only one thread needs to be executed at a time, and if enough computational elements are available that none needs to be reused in the execution path, it is possible to set up a system like Cheops as a fully static pipeline: one set of crosspoint connections is made for the entire thread. More typically, though, we must reuse the elements (for example, in a predictive transform encoder, the discrete cosine transform [DCT] processor that codes the error signal might also need to do the inverse discrete cosine transform [IDCT] in the prediction loop), and the real-time programming needed for circuit-switching and processing element reconfiguration becomes unwieldy. The situation is yet worse if we hope to share resources among multiple execution threads. These considerations led us to consider automated resource management, which also nearly handles the problem that in Cheops the specialized processors are on removable submodules, such that the hardware configuration may change from time to time or from machine to machine. In order to allow a software application to execute on differently configured Cheops systems, at least a simple resource-management process was needed. The logical follow-on was to automatically parallelize the execution as much as permitted by the hardware configuration. This transparent, run-time parallelization can equivalently be viewed in terms of hardware scalability. The management strategies we considered were

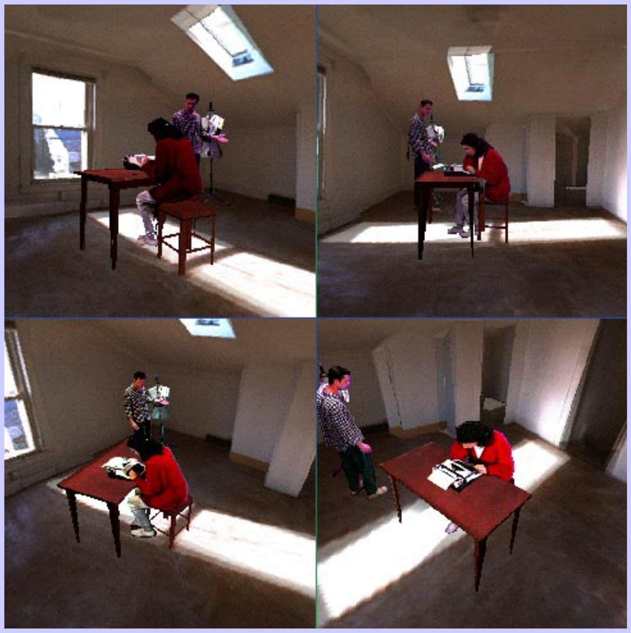
- *Static:* Processing tasks are assigned to processing devices at compile time. This method involves no run-time overhead, and does not support multitasking or hardware scalability.
- *Run time:* A management process assigns devices for the entire processing pipeline just before the program is run. This involves minimal run-time overhead, and does support hardware scalability, but not multitasking.
- *On-the-fly:* Each stage in the process is assigned to a device as the prerequisite data and devices become available. This is the method used in Cheops. In exchange for the flexibility, the overhead costs are significant.

A user program for Cheops—a process that handles file I/O, interaction, and so forth—contacts the resource manager by making a function call whose argument is a pointer to a linked-list data structure describing the data-flow portion of the algorithm. An individual stream transfer making up a subpart of the data-flow graph then occurs when all prerequisite data, an appropriate stream processor, and a destination are available. Transfers may optionally also be made dependent upon a real-time clock to permit process synchronization with video sources or displays. The programmer may associate a callback routine with any individual transfer, to inform the invoking process of the status of the data-flow processing. As noted above, there is a computational cost associated with on-the-fly resource management. Even when the manager was made as lightweight as possible (in which case it could not be very clever) we have found that it consumes the majority of the general-purpose cycles of the CPU. In future designs we intend to use a dedicated resource-management CPU, so that other demanding computations cannot affect scheduling performance.

Another interesting approach to provide the efficiency of specialized hardware while supporting flexible computing (which might or might not be combined with the stream-based computing discussed above) is suggested by the recent availability of dense logic arrays that can be reprogrammed quickly while in circuit. As there is a certain processor overhead associated with reconfiguration of such devices, efficient resource management will require additional intelligence. In a processing system developed by our group, the logic devices were accompanied by local memory for caching several configurations, reducing the load on the resource-managing CPU.[17] A stream processor consisting of a programmable logic array, a general-purpose microprocessor, and static random access memory (SRAM) has proven able to emulate (at equal or better speed, and in the same board space) several of Cheops's stream processors, with the exception of those capable of high-speed parallel multiplication; a more appropriate architecture would provide hardwired multiply-accumulate (or general arithmetic logic) units connected to the routing logic.

As of this writing, the use of SRAM-based electrically programmable logic devices (EPLDs) in multimedia

Figure 6 **Stills from the video "Wallpaper," in which we used machine-vision methods to build a three-dimensional model of the set. Combined with multiple calibrated camera views of the actors, this modeling permits synthesizing arbitrary views of the scene either in post-production or for interactive display.**



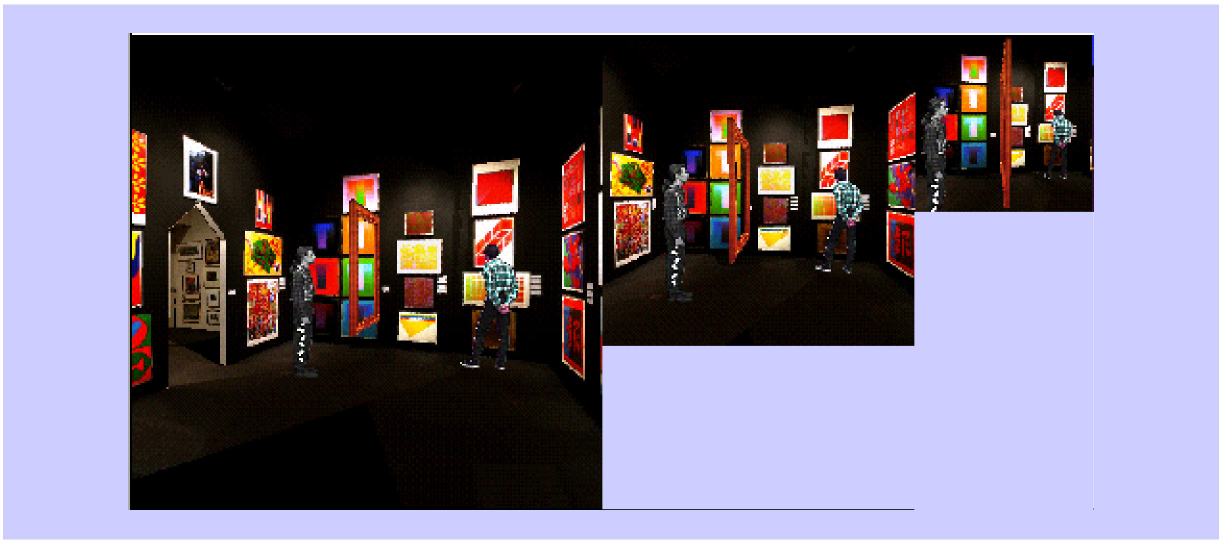
applications is certainly not a cost-effective technique. Should process improvements and manufacturing economies of scale make it so, several questions will remain to be addressed:

- Can functional descriptions of needed tasks be represented in a form that is not tied to the architecture of a specific device, but can easily be interpreted at run time? Besides supporting hardware scalability, such representations will enable "hardware on demand," in which content providers can download specific computational architectures for particular applications.
- Is it possible to automatically and efficiently segment algorithms into portions for which the programmable logic device is more appropriate and portions that are more suited for execution on associated general-purpose processors?
- Can the devices be reconfigured dynamically, effectively paging in hardware functions as parts of a larger algorithm? The answer to this question seems to be yes, based on results reported by other researchers.[18]

Prototype program material

To understand some of the issues involved in object-based-video production, post-production, and interactive or personalized display, we have made several short multimedia productions. We computed three-dimensional models of the unoccupied sets, enabling resynthesis of arbitrary points of view (see Figure 6), and finding the three-dimensional locations of actors moving about in the sets. As we do not yet have methods for producing good 3D models for people, the actors are represented as two-dimensional objects with multiple viewpoints provided by multiple cameras. We have used these productions to explore the idea of "intelligent interoperability," in which either an author or an automatic process can cause the video to display differently (presumably a more optimal display) on different sizes and aspect ratios of displays. The first form of intelligent interoperability we explored was recropping the frame for various sizes and shapes of screens. Thus on a small screen the video might contain more cuts and close-ups than on a large, wide screen.[13] A more advanced behavior that

Figure 7 In this example from the video "The Museum," the focal length of a simulated lens changes when the video is viewed on a smaller screen; thus, the actors in the foreground remain recognizably large while much of the background stays visible.



seems to offer great potential is changing the effective focal length of a simulated lens (see Figure 7), thus allowing the entire background to be visible in a small window while not unduly reducing the size of foreground elements. Doing the latter while maintaining good scene composition may also entail a shift in camera position, as shown in the figure.

We have also handled audio in an object-based fashion: rather than channels corresponding to speakers, sound was represented as a set of localized sources and an acoustical environment in which they are placed. These sound sources were then linked to the visual objects with which they are associated. As directed by the script, perhaps in conjunction with user interaction, the audio is "rendered" for the speakers associated with the video display. Thus, the "soundscape" changes as the visual viewpoint is varied. The auditory synthesis methods are much better understood this time than are the analysis methods. Synthesis involves simulation of the early reverberation process—for each speaker, a separate finite-impulse-response filter is calculated and applied to each sound source to give the effect of the reflections from walls—then a nondirectional diffuse reverberation signal is calculated and added representing steady-state room noise for all the sources and their echoes.[19] Reference 20 discusses an implementation in software on a midrange workstation, which proved able to generate sound for two speakers while running UNIX** and several other tasks. Until the analysis is better developed, changes in production methods and linkage with the video analysis can assist the process. In our production experiments each actor has carried a separate wireless microphone, and the video analysis methods provided the locations from which their voices emanated. We are now looking into using fixed microphone arrays combined with information-maximization methods[21] for unmixing and deconvolution as part of the capture process.

Conclusions

In order for digital video and audio to be more than just a bandwidth-efficient version of their analog predecessors, the digital representation must permit the processing devices that are a part of multimedia systems to perform useful *content-based* operations upon the data. An object-based description as outlined previously offers this possibility, and permits evolution to more efficient and flexible content representations as improved methods develop.

The reader will perhaps note that I have spent much more time on the display than on the camera. This emphasis is intentional; as the synthesis side is better understood than the analysis side (particularly with respect to real-time processing), we have used our prototype displays to demonstrate the advantages of object-based video and audio representations. Having done so, we may now increase our attention on the capture of appropriate scene information to drive the creative and communicative applications we have enabled.

Acknowledgments

The author wishes to thank numerous students and coworkers who have contributed to the research described in this paper, in particular Stefan Agamanolis, Shawn Becker, Edmond Chalom, and John Watlington. This project has been sponsored by the Television of Tomorrow consortium at the MIT Media Laboratory.

**Trademark or registered trademark of Adobe Systems, Inc. or X/Open Co. Ltd.

Cited references

1. M. J. Biggar, O. J. Morris, and A. G. Constantinides, “Segmented-Image Coding: Performance Comparison with the Discrete Cosine Transform,” *Proceedings of IEE Part F* **135**, No. 2 (April 1988), pp. 121–132.
2. A. Akutsu et al., “Video Indexing Using Motion Vectors,” *Proceedings of SPIE Visual Communications and Image Processing,* Vol. 1818 (1992), pp. 1522–1530.
3. S.-F. Chang and D. G. Messerschmitt, “A New Approach to Decoding and Compositing Motion-Compensated DCT-Based Images,” *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing* (ICASSP-93) (1993), pp. V421–V424.
4. E. A. Adelson and J. Y. A. Wang, “Representing Moving Images with Layers,” *IEEE Transactions on Image Processing* **3**, No. 5, 625–638 (Sept. 1994).
5. M. Irani, S. Hsu, and P. Anandan, “Mosaic-Based Video Compression,” *Proceedings of SPIE Digital Video Compression: Algorithms and Technologies*, Vol. 2419 (1995), pp. 242–253.
6. M. Kunt, A. Ikonomopoulos, and M. Kocher, “Second-Generation Image-Coding Techniques,” *Proceedings of IEEE* **73**, No. 4 (1985), pp. 549–574.
7. H. G. Musmann et al., “Object-Oriented Analysis-Synthesis Coding of Moving Objects,” *Signal Processing: Image Communication* **1**, 117–138 (1989).
8. R. G. Kermode, “Coding for Content: Enhanced Resolution from Coding,” *Proceedings of IEEE International Conference on Image Processing* (ICIP ’95) (1995), pp. 460–463.
9. J. K. Aggarwal and N. Nandhakumar, “On the Computation of Motion from Sequences of Images—A Review,” *Proceedings of IEEE* **76**, No. 8 (August 1988), pp. 917–935.
10. C. Verplaetse, “Inertial Motion Estimating Camera,” S.M. thesis, MIT, Cambridge, MA (1996).
11. S. Becker and V. M. Bove, Jr., “Semiautomatic 3-D Model Extraction from Uncalibrated 2-D Camera Views,” *Proceedings of SPIE Image Synthesis,* Vol. 2410 (1995), pp. 447–461.
12. E. Chalom and V. M. Bove, Jr., “Segmentation of Frames in a Video Sequence Using Motion and Other Attributes,” *Proceedings of SPIE Digital Video Compression: Algorithms and Technologies*, Vol. 2419 (1995), pp. 230–241.
13. V. M. Bove, Jr., “Object-Oriented Television,” *Society of Motion Picture and Television Engineers (SMPTE) Journal* **104**, 803–807 (December 1995).
14. S. Agamanolis, “High-Level Scripting Environments for Interactive Multimedia Systems,” S.M. thesis, MIT, Cambridge, MA (1996).
15. J. A. Watlington and V. M. Bove, Jr., “Stream-Based Computing and Future Television,” *Proceedings of 137th Society of Motion Picture and Television Engineers (SMPTE) Technical Conference* (1995), pp. 69–79.
16. V. M. Bove, Jr. and J. A. Watlington, “Cheops: A Reconfigurable Data-Flow System for Video Processing,” *IEEE Transactions on Circuits and Systems for Video Processing* **5**, 140–149 (April 1995).
17. E. K. Acosta, V. M. Bove, Jr., J. A. Watlington, and R. A. Yu, “Reconfigurable Processor for a Data-Flow Video Processing System,” P*roceedings of SPIE FPGAs for Fast Board Development and Reconfigurable Computing*, Vol. 2607 (1995), pp 83–91.
18. J. Villasenor, C. Jones, and B. Schoner, “Video Communications Using Rapidly Reconfigurable Hardware,” *IEEE Transactions on Circuits and Systems for Video Processing* **5**, 565–567 (December 1995).
19. W. G. Gardner, “The Virtual Acoustic Room,” S.M. thesis, MIT, Cambridge, MA (1992).
20. A. V. Inguilizian, “Building a Better ‘Picture’: Synchronized Structured Sound,” S.M. thesis, MIT, Cambridge, MA (1995).
21. A. J. Bell and T. J. Sejnowski, “An Information-Maximisation Approach to Blind Separation and Blind Deconvolution,” *Neural Computation* **7**, No. 6, 1004–1034 (1995).

Accepted for publication March 19, 1996.

V. Michael Bove, Jr. *MIT Media Laboratory, 20 Ames Street, Cambridge, Massachusetts 02139-4307 (electronic mail: vmb@media.mit.edu).* Dr. Bove holds an S.B.E.E., an S.M. in visual studies, and a Ph.D. in media technology, all from the Massachusetts Institute of Technology. In 1989 he was appointed to the faculty of MIT, where he is currently associate professor of media technology, working in the MIT Media Laboratory. He is current holder of the Alex W. Dreyfoos, Jr. Career Development Professorship. He is the author or coauthor of over 30 journal or conference papers on digital television systems, video processing hardware/software design, scene modeling, and optics. He holds patents on inventions relating to video recording and hard copy, and has been a member of several professional and government committees. In December 1995, *Boston Magazine* named him one of the “People Shaping Boston’s High-Tech Future.” He is serving as general chair of the 1996 ACM multimedia conference. More detail on his research, and downloadable documents, can be found at http://vmb.www. media.mit.edu/people/vmb/.

Reprint Order No. G321-5609.