A Software-Only Video Production Switcher for the Internet MBone

Tina Wong* Ketan Mayer-Patel David Simpson Lawrence A. Rowe

Computer Science Division
University of California, Berkeley
{twong,kpatel,davesimp,larry}@cs.berkeley.edu

ABSTRACT

In this paper, we describe the design and implementation of a software video production switcher, `vps`, that improves the quality of MBone broadcasts. `vps` is modeled after the broadcast television industry's studio production switcher. It provides special effects processing to incorporate audience discussions, add titles and other information, and integrate stored videos into the presentation. `vps` is structured to work with other MBone conferencing tools. The ultimate goal is to automate the production of MBone broadcasts.

1 INTRODUCTION

Live programs are produced and broadcast world-wide on the Internet MBone using IP Multicast [1] and the MBone [2] conferencing tools (e.g., `vic` [8], `vat` [7], `wb` [3], `sdr` [5] etc.). Some examples are the NASA Space Shuttle Missions, conference presentations (e.g., Sixth International WWW Conference), and live music performances. These broadcasts usually have audiences ranging from tens to hundreds of viewers distributed world-wide.

We have broadcast the weekly Berkeley Multimedia and Graphics Seminar on the MBone since early 1995. The seminar is produced using the LBL/UCB MBone tools `vic` and `vat` to capture and transmit video and audio streams, respectively. The shared whiteboard tool `wb` is used to distribute postscript slides. A second `wb` is used by the broadcast director (hereafter, director) to communicate with participants in order to debug problems with the transmission and monitor video and audio quality. We are also testing a floor control tool `qb` [6] to facilitate question asking. The current broadcast uses a single camera that mainly focuses on the speaker, and occasionally pans to show other materials such as slides on the overhead projector, a demo running on a workstation, or members of the local audience. This single camera approach is the most common configuration seen in low-budget, small-scale broadcasts on the MBone.

*T. Wong is supported by a GAANN fellowship.

We are working on tools to improve the quality and simplify the production and control of MBone broadcasts. This paper describes the design and implementation of a software-only *video production switcher* (`vps`) that can be used to improve the quality of an MBone broadcast. `vps` is modeled after a *studio production switcher* [9] used in the broadcast television industry. A studio production switcher is a custom-designed hardware device that provides an array of real-time editing and special effects functions. A director can select one of several picture sources (e.g., cameras, videotapes, and still image displays) to be the output. Other sources are generated by the device by applying special effects processing to one or more streams such as inserting titles into a picture, superimposing one picture on another, chroma-keying, and wiping or fading from one picture to another.

`vps` will enhance the quality of an MBone broadcast by providing effects available in a hardware switcher. Specifically, we want to display local and remote audience discussions and feedback, add titles and credits, integrate stored analog and digital videos into the presentation, and incorporate special effects to improve the visual images and retain audience attention. Ultimately, our goal is to automate the production process by integrating `vps` with a broadcast management system [13] that maintains room, equipment and broadcast configurations, observes event schedules, and launches and monitors the MBone tools required to produce a

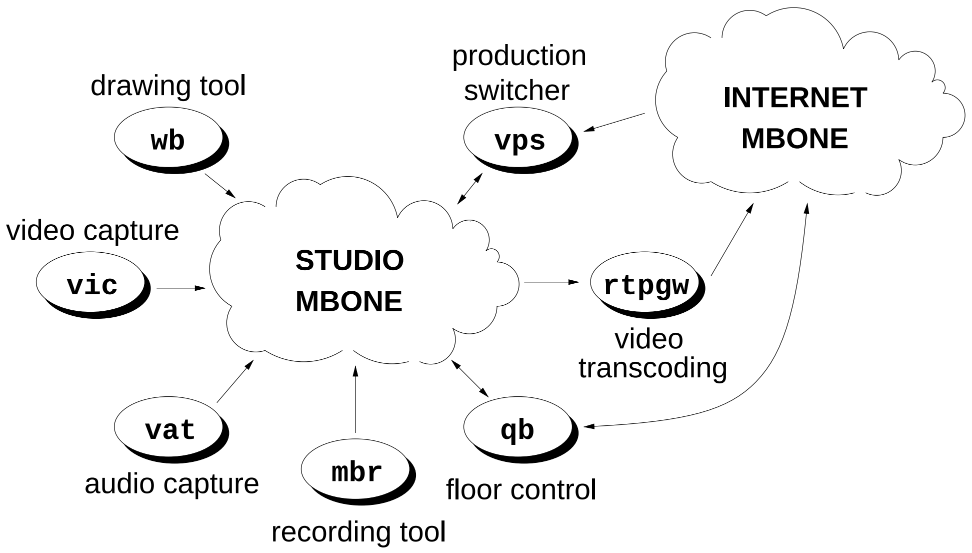

Figure 1: vps in an MBone Broadcast.

broadcast. Figure 1 shows how vps fits into the context of a typical MBone broadcast. The *Studio MBone* is a local domain network connecting the processes required to produce the broadcast. It can support high data rates (e.g., 5 to 30 Mbs) and good quality video streams (e.g., MJPEG video). The public MBone is the Internet and it runs at a considerably lower speed.

A hardware production switcher is a highly developed technology that could be used in an MBone broadcast. However, this solution has several limitations. First, video sources must be converted to a switch-specific analog format before being passed to the switcher and converted back to a digital format and encapsulated as RTP data [11] after processing so that it can be sent to the MBone. vps avoids these conversions by operating on video streams in the RTP representation. Second, a hardware system is not extensible. vps is designed in a modular manner to allow new effects to be added to the system. Third, vps can be controlled by other software to automate decisions by a director through reactive software heuristic technologies. Finally, a hardware switcher has only one user interface. vps can be operated by many GUIs ranging from a simple interface designed for a speaker to a sophisticated interface designed for a skilled director. Moreover, interfaces can be customized for different users.

This paper describes the design of vps including the GUI interface and the implementation of the current system. It is organized as follows. Section 2 presents an example of vps in use. Section 3 describes the vps software architecture. The implementation of vps is described in section 4. Section 5 talks about future work and section 6 summarizes the paper.

2 AN EXAMPLE SCENARIO

We describe how vps can improve the quality of an MBone broadcast by illustrating its use through an example scenario.

Before beginning the scenario, we first describe the two GUI interfaces to vps: the *director's console* and the *speaker's console*, shown in Figures 2 and 3, respectively. The director's console produces the content of the broadcast. The *main window* of this console has an *editor area* at the top and a *preview area* at the bottom. The director uses the editor area to choose a specific effect editor and to configure the parameters of that effect. The preview area shows thumbnails of video sources including the results of applying an effect. The director can click on a thumbnail to see more information about that video. The *output window* of the director's console shows the current video being broadcast. This window also describes the broadcast multicast session, if applicable. The speaker's console allows the speaker to incorporate stored videos into the lecture. It has a preview area similar to the director's console. The speaker clicks on a thumbnail

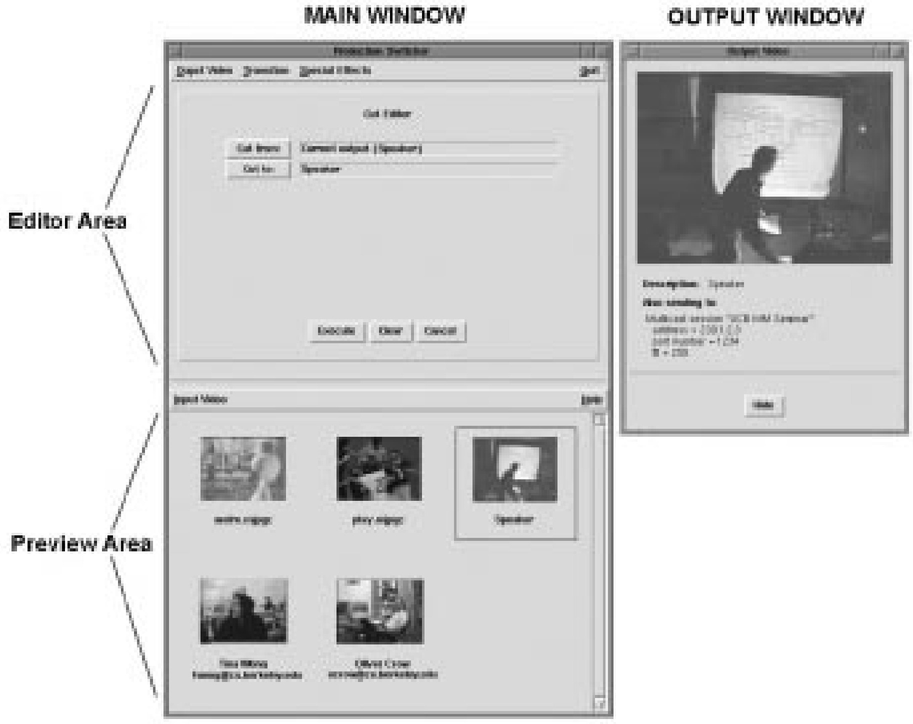

Figure 2: The Director's Console.

to select and and bring up a VCR-like player to playback a video.

The following shows how `vps` can be used in this scenario. Suppose a seminar is being conducted on the Berkeley campus. Students on campus attend the seminar in the lecture room, and remote viewers join in virtually by watching the broadcast on the MBone.

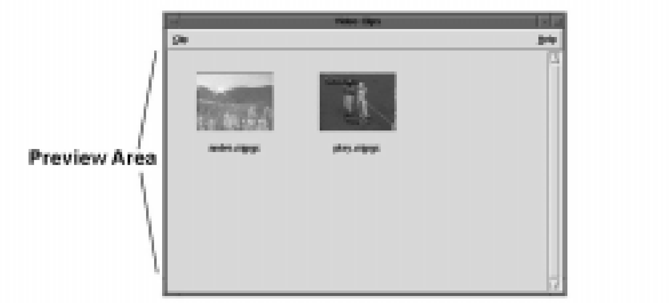

Figure 3: The Speaker's Console.

Viewing Sources and Monitoring the Broadcast

Suppose there are two cameras in the seminar room: one focusing on the speaker and one facing the audience. Suppose further that participants at several remote locations also have digital cameras attached to their workstations, and the speaker has several videos to accompany her lecture. The director uses these videos [1] to produce the content of the broadcast. He previews them in the preview area of the director's console. He also monitors the broadcast with the output window. New video sources can be added at any time during the broadcast. For example, a remote viewer who joins late can still be a `vps` source and part of the lecture broadcast.

Beginning a Lecture

A short time before the lecture starts, the director switches from a still image that identifies the program to a picture showing the speaker. He uses the *cut editor* of the director's console to select this picture and switch sources. He then uses the *subtitle editor* to insert the seminar title and the speaker name onto the picture. After a minute or so, the director removes the titles by switching to the original picture. Figure 4 shows screen shots that illustrates the opening of the lecture.

[1] These videos might be stored on a video file server or replayed on a VCR.

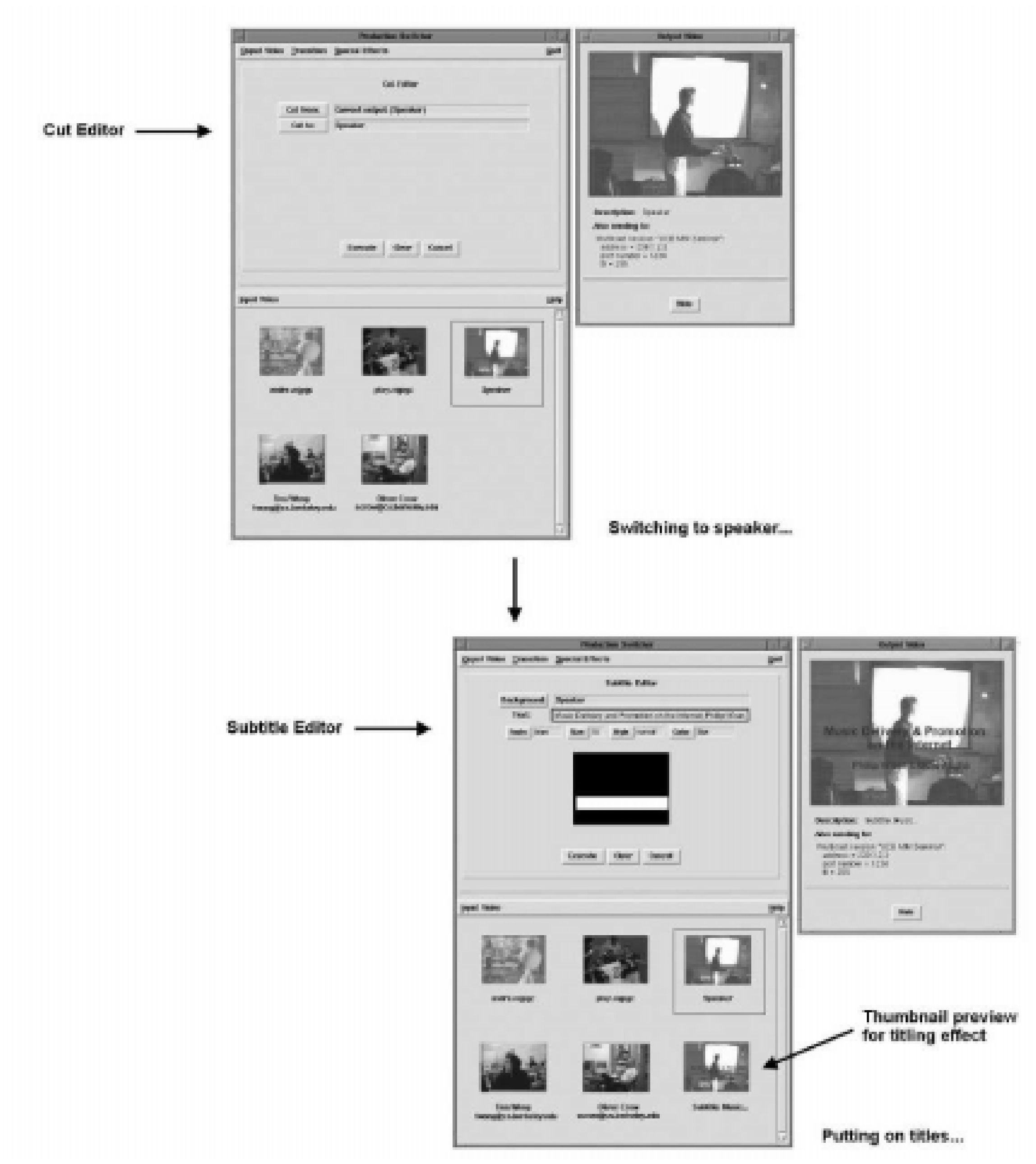

Figure 4: Producing Opening Phase of Lecture.

Playing Stored Video
At some point in the lecture, the speaker wants to show a video. She uses the speaker's console to select and play the video. Depending on her pace, she can use the VCR-like controls to play, stop, rewind, fast-forward or restart the video. The director's console provides the same playback facilities so the director can assume this task. See Figure 5 for a screen shot of the VCR-like player.

Incorporating Audience Discussions
A local audience member raises his hand to notify the speaker that he has a question. A few remote viewers also indicate their desire to ask a question or comment by entering a request into the floor control tool. The floor moderator signals a remote viewer to ask her question. The director notices that a video of this remote viewer is available because it is being previewed on the director's console. The floor control tool might send a "grant floor" message to all tools. The director's or speaker's

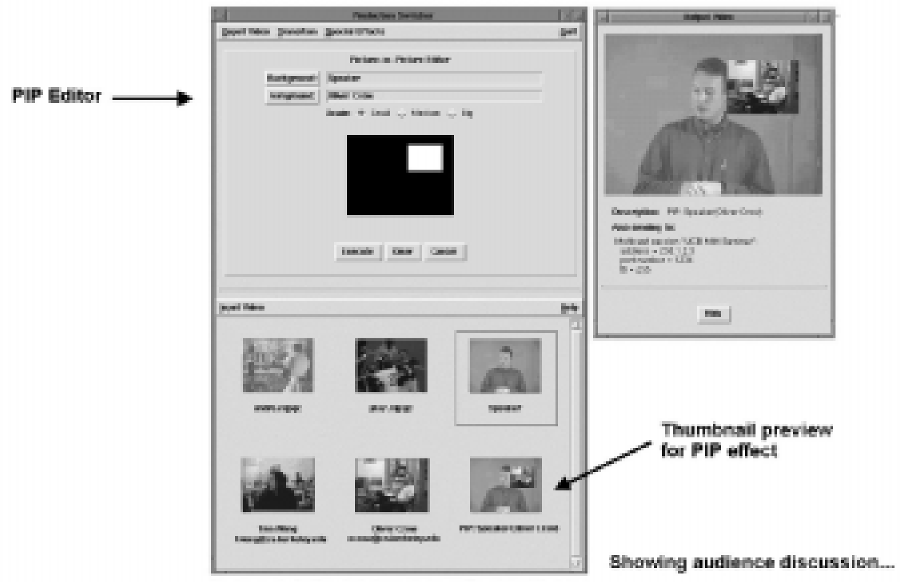
PIP Editor
Thumbnail preview for PIP effect
Showing audience discussion...

Figure 6: Incorporating Audience Discussions.

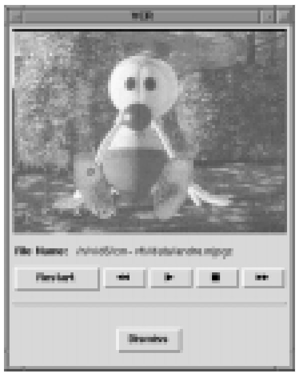

Figure 5: VCR-like Player.

console can display a thumbnail in the preview area when it receives this message if it includes a video source or still image. `vps` could automatically switch to a stream that showed the speaker and questioner in side-by-side windows. This example illustrates automatic switching. This action can also be invoked manually by using the *picture-in-picture (PIP) editor.* See Figure 6 for screen shots.

Finishing a Lecture

At the end of the lecture, the director uses the *fade editor* to execute a fade transition from the speaker to a black screen. On the black screen, he uses the subtitle editor to put up acknowledgments and credits to thank the speaker, people organizing the seminar, and an advertisement for the next event. Figure 7 shows an example.

3 SOFTWARE ARCHITECTURE

This section describes the `vps` architecture. `vps` is composed of multiple processes: a *video file server*, a *broadcaster*, one or more *effects (fx) processors* managed by an *fx server*, and two *user interfaces*. These processes exchange control messages and transmit video data to each other using RTP over IP Multicast on the Studio MBone, and receive data from the public MBone. Figure 8 illustrates the processes in this software architecture.

`vps` is decomposed into multiple processes in order to build a distributed system which can utilize more resources, facilitate future extensions in effects processing, and make modifications to the user interfaces. The system is implemented with the Continuous Media Toolkit (CMT) [4] which is described in more detail in section 4.

3.1 Studio MBone

The Studio MBone is an RTP session [11] with a single multicast group and two port numbers [2] on which `vps` processes transmit video data and control messages. This multicast address is well known to the processes and can be configured as a command-line argument at system startup. One

[2]This multicast address can be chosen through the *session directory protocol*[5] to avoid conflicting with other multicast groups. Since the Studio MBone spans Berkeley, we only need to allocate a multicast address not in use on campus.

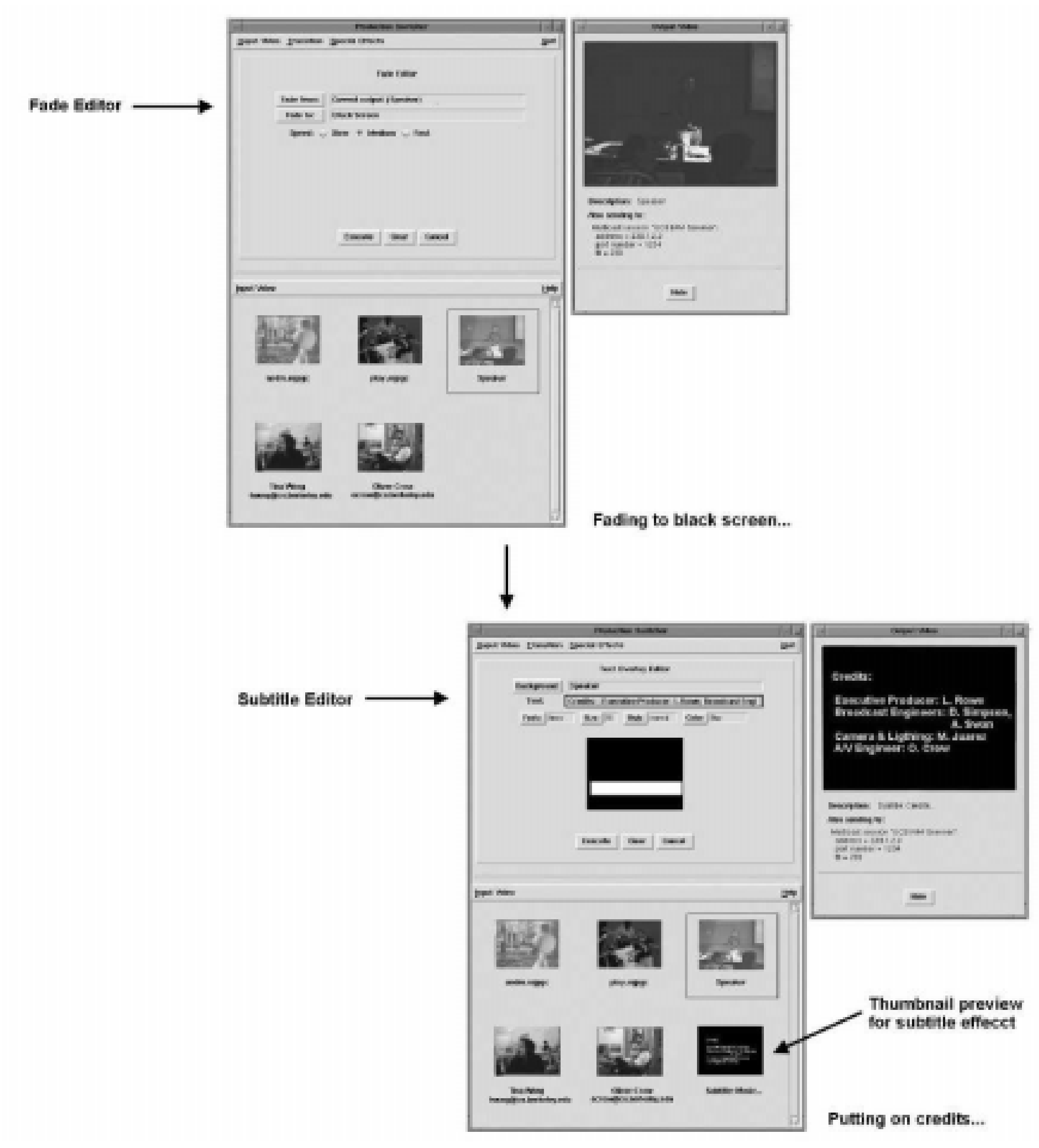
Fade Editor
Fading to black screen...
Subtitle Editor
Thumbnail preview for subtitle effecct
Putting on credits...

Figure 7: Producing Closing Phase of Lecture

port number is used for data and the other for control. Administrative scoping or the time-to-live (ttl) field in the RTP session is set to reach all processes in the system. For our broadcasts, the Studio MBone spans our building.

We designed the system so that control messages serve as an interface among the processes. Consequently, internal changes to a process do not affect other processes as long as these messages remain the same. Control Messages provide coordination among `vps` processes. In the current system, messages flow from the user interfaces to the other processes. They request effects processing from the fx server, configure parameters at the broadcaster, notify the broadcaster to switch to another video source, and control video playback at the video file server. Table 1 lists the control messages used in the current system. Although these messages could be unicast to the appropriate destinations, we believe multicast will be more efficient when the system is integrated with other MBone tools. For example, as the floor control tool grants the floor to an audience member, it multicasts a message on the Studio MBone so the responsible `vps` processes can react to the message by switching to the correct video to broadcast and/or requesting effects processing. The Studio MBone is connected to the public MBone by a multicast router. This way, streams from remote participants are automatically passed to the Studio MBone.

Available video streams are source videos and result streams from effects processing. They are multicast on the Studio MBone instead of unicast because multiple processes usually need the same video at one time. For example, the result of an fx processor is needed by the preview area at the director's console, another fx processor for other kinds of processing, and the broadcaster to output to the MBone, all at the same time.

3.2 Stored and Live Video Server

The video file server process serves stored digital videos to other `vps` processes. Stored video playback is controlled by the user interfaces which send control messages to the server. The server plays a video by multicasting the appropriate streams on the Studio MBone.

Live videos originating from the local studio (e.g., camera feeds from the lecture room) or from other video feeds (e.g., cable or satellite receivers) are "served" by the Studio MBone in the sense that the streams are multicast on the associated RTP session. Live videos from remote participants (e.g., cameras attached to student workstations) are sent on a separate RTP session on the public MBone. These streams are multicast on different addresses so that local data and control messages are not forwarded to the public MBone in order to avoid wasting valuable bandwidth. We distinguish each video source within an RTP session with the unique synchronization source identifier field (`ssrc`) specified in the RTP header.

3.3 Broadcaster

The broadcaster process multicasts the `vps` output to the public MBone at the address and port number advertised for the broadcast. The director using the director's console selects a video to be the output and sends a control message to tell the broadcaster to carry out the switching between streams.

3.4 FX Processor and FX Server

An fx processor manipulates one or more video streams to generate special effects. The effects supported by the current `vps` are fade, mix, picture-in-picture (PIP) and subtitle. The fade and mix effects are implemented in the compressed MJPEG domain which means the streams are not fully-decoded before being processed [12]. PIP and subtitle are implemented in the uncompressed YUV domain which means the streams must be fully-decoded before being processed. More details about the effects processing algorithms are presented in the next section.

There can be more than one fx processor depending on the computing resources available. The collection of fx processors are managed by the fx server. It communicates with the other `vps` processes, accepting processing requests from the director's console and assigning them to an fx processor. Fx processors are scheduled using round robin scheduling to ensure load balancing. The result of effects processing is multicast onto the Studio MBone so all `vps` processes can utilize it. For example, the director's console previews the result before it is being switched to the output, and another fx processor uses the result as an input to a different effect.

This design was chosen so the fx server and fx processors can be easily extended without requiring the other to be significantly rewritten or affecting other `vps` processes. Changes in the load balancing policy in the fx server do not affect the internals of the fx processors. Likewise, modifications to the processors, such as implementing effects processing in the raw or compressed domain, are isolated.

To add a new type of effects processing such as chroma-key which is common in television weather forecasts, we would need to include the code to implement this processing into the fx processor, extend the control messages so that the director's console can request the effect, and implement a chroma-key editor so the director can control the effect.

3.5 User Interfaces

As described in the scenario, there are two user interfaces in `vps`. The director's console is the main control center which provides a set of primitives to manipulate `vps`, and the speaker's console which

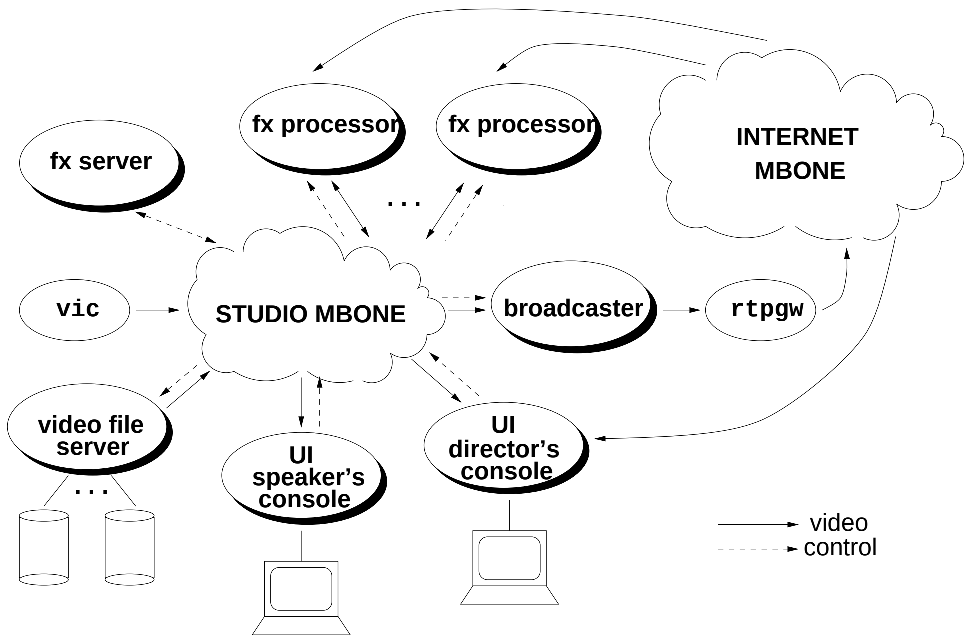

Figure 8: `vps` Software Architecture.

sender	*receiver*	*control name*	*parameters*
`Director's Console`	`Fx Server`	`Processing`	`FX Name, Fx Params`
`Director's Console`	`Video File Server`	`Playback`	`File ID, Playback Speed`
`Speaker's Console`	`Video File Server`	`Playback`	`File ID, Playback Speed`
`Director's Console`	`Broadcaster`	`Switch`	`Video ID`
`Director's Console`	`Broadcaster`	`Configure`	`address/port/ttl`

Table 1: Control messages.

allows the lecturer to integrate stored video into the presentation. Two separate interfaces are provided so that the production process and the lecture can be going on in different rooms. They are written in Tcl/Tk [10] and OTcl [14] and are easy to modify to incorporate better UI designs, as we get more experience using them, and new editors when new effects are included.

The thumbnail previews in the director's and speaker's console are "optimized" in the sense that they are updated infrequently to avoid expensive decoding of each frame in the video. The current implementation displays one frame every one hundred frames of the video. When effects processing is requested by the director, the director's console sends a control message to the fx server to request the processing as described above. The resulting stream is sent back on the Studio MBone so the director's console can display it in the preview area. When a transition is requested, the director sends a control message to the broadcaster process to execute the switch.

3.6 Automating the Production Process

Several aspects of the production process can be automated. The director's console can follow a prepared script and send control messages for switching and effects processing at specified times. For example, the script shown in Table 2 can be used to automate a broadcast. The first part of the script defines variables to be used later on; `startTime` is the advertised starting time of the broadcast (April 15, 1997, 1:00 p.m.), `endTime` is the end time (3:00 p.m. the same day), and `speakerStream` is the camera facing the speaker (ssrc 326232628 in the RTP session 234.1.2.3/1234). The second part of the script automates the opening and closing phases of the broadcast. It first tells the broadcaster process to switch `speakerStream` to the MBone at `startTime`. Then, it requests subtitle processing on `speakerStream` with a text string and assigns the resulting stream to the variable `titledStream`. At time `startTime` + 30 seconds, it switches to the stream specified by `titledStream`. It then switches back to the original video `speakerStream` at `startTime` + 60 seconds. Similar actions are are executed in the closing phase. A potential problem here is that the estimated times are not always accurate, as the lecture can start late and run overtime. These situations should be accounted for in the design of the automation engine.

4 IMPLEMENTATION AND DISCUSSION

This section discusses our current implementation and related issues.

4.1 Status

`vps` is implemented using the Continuous Media Toolkit (CMT). CMT is a portable toolkit of reusable objects operating on media streams that simplifies the development of multimedia applications. The toolkit includes video file and playback objects in MJPEG and H.261 formats, communication objects for unicast (UDP), multicast (RTP), and blocking and non-blocking RPC, synchronization objects to control application behavior, and filter objects that implement effects processing on YUV and MJPEG data. Each process in `vps` is composed of CMT objects connected by the Tcl scripting language. The `vps` code is structured into a hierarchy of classes using OTcl, an object-oriented extension to Tcl developed at MIT. The Tk toolkit is used to build the user interface.

The current implementation is a prototype of the described system. We implemented effects processings in the YUV and MJPEG domains. The automation engine is in its design phases and works closely with the broadcast management system described in the introduction. To demonstrate the system's feasibility, this prototype sends video data unicast (UDP) over the network and control messages via RPC. The implementation is being updated to use IP Multicast for both video data and control messages. At the writing of this paper, the system has gone through two major revisions. The most recent version has approximately 5000 lines of OTcl code.

4.2 Special Effects Processing

The PIP and subtitle effects are applied in the uncompressed domain, and have YUV streams as inputs and outputs. The mix and fade effects are generated in the compressed JPEG domain, which process MJPEG streams and output in the same format. The algorithms that manipulate images in the compressed domain are fully described in [12].

To investigate the performance of the current effects implementation, we conducted experiments to determine the latency of each effects on each frame. We also measured the performance of the MJPEG to YUV decoding operation. The MJPEG and YUV streams used in the measurements are CIF (320x240) sized videos, and are served from local disks to isolate the measurements from network overhead. The measurements were carried out on a 200 MHz Pentium Pro with 32MB of memory and 2GB of disk space.

The results are presented in Table 3. In our implementation, the generation of various effects is inexpensive; it takes approximately 15 to 20 ms to process each frame in YUV. The decoding from MJPEG to YUV is more computationally intensive and thus takes on average 45 ms. Adding the decoding times to the processing times, it takes about 65 ms to complete the YUV processing on each frame. When compared to the mix effect implemented in the compressed domain, we see that it takes on average of 65 ms for processing YUV data, but only 20 ms for processing in the compressed domain. Clearly, effects processing in the compressed domain is much faster than converting a stream to YUV, applying the transformation, and converting back to MJPEG [12]. These measurements only account for the raw computation time needed to generate the operations; we did not look into how other bottlenecks in the `vps` system can affect the performance of effects processing. Other possible I/O bottlenecks may exist in the kernel when the system transmits or receives streams over the network.

The effects processing offered in the current system are simple; they mainly involve memory copies and/or simple calculations. For more complex effects that require greater computation power, such

```
set vps [new VPS ...]
set startTime [new Time ``4 15 1997'' ``13:00'' GMT]
set endTime [new Time ``4 15 1997'' ``15:00'' GMT]
set speakerStream [new LiveStream ``234.1.2.3'' 1234 326232628]
...
at $startTime ``$vps cut $speakerStream''
set introText ``Berkeley Graphics ...''
at $startTime + 1 ``set titledStream [$vps subtitle $speakerStream $introText]''
at $startTime + 30 ``$vps cut $titledStream''
at $startTime + 60 ``$vps cut $speakerStream''
...
at $endTime ``$vps fade $speakerStream black''
set creditsText ``Credits ...''
at $endTime + 1 ``set endStream [$vps subtitle black $creditsText]''
at $endTime + 30 ``$vps cut $endStream''
```

Table 2: OTcl script to automate production process.

Operation	*Latency (ms)*	*Std. dev (ms)*
MJPEG $\rightarrow$ *Decode* $\rightarrow$ YUV	44.62	9.64×10^{-8}
MJPEG $\rightarrow$ *Mix* $\rightarrow$ MJPEG	18.62	9.78×10^{-7}
YUV $\rightarrow$ *Mix* $\rightarrow$ YUV	20.65	2.42×10^{-4}
YUV $\rightarrow$ *PIP* $\rightarrow$ YUV	14.95	2.21×10^{-4}
YUV $\rightarrow$ *Subtitle* $\rightarrow$ YUV	19.62	5.96×10^{-4}

Table 3: Measurements results.

as chroma-key, we need to resort to more sophisticated algorithms. Also, for larger image sizes, such as SCIF video (640x480), it takes at least four times as long to process each frame for many effects. Section 5 discusses future work that will address the ability to execute more complex effects in a way that maintains an acceptable throughput data rate. the Another issue to consider is H.261 video [8]; this is the format almost exclusively used in public MBone broadcasts. Although streams originating in the Studio MBone are higher bitrate streams such as MJPEG, the fx processors need to be enhanced to handle H.261 streams from the public MBone as well.

5 FUTURE WORK

Future work with the software-only video production switcher will be concentrated on the fx server and fx processor. The current `vps` implementation is able to realize simple effects with objects written specifically to perform the desired effect on a particular format of video. This approach is inflexible and requires that a new object to be written for each new video format and desired effect. In addition, the complexity of an object increases with the complexity of the desired effect.

A more flexible approach is to represent complex effects as a combination of simpler functions. Objects written to perform these basic functions can be reused and recombined into different complex effects. As an example, Figure 9 shows a mix effect represented as a combination of simple multiplication and addition functions.

Another requirement of the fx server in the `vps` system is the ability to maintain an acceptable level of throughput independent of the complexity of the desired effect. To meet this requirement, the fx server must exploit parallelism. Three types of parallelism are available: temporal, spa-

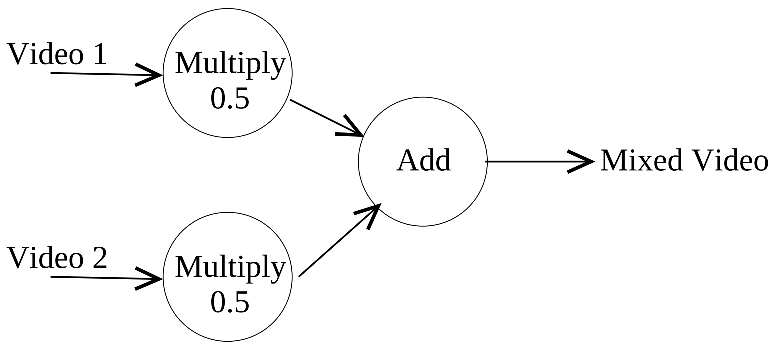

Figure 9: Mix effect as combination of simple functions.

tial, and functional decomposition. Temporal parallelism can be exploited by creating and controlling fx processors to process frames of a video stream independently. For example, the fx server may use one fx processor to process all odd numbered frames and another fx processor to process all even numbered frames. Spatial parallelism can be exploited by splitting the input video streams along spatial boundaries, utilizing separate fx processors to process each region independently, and recombining the resulting streams into one video stream. Functional decomposition takes advantage of representing complex effects as a combination of simpler functions. The fx server can decompose this representation into two or more stages and utilize separate fx processors for each stage.

The use and management of parallelism to realize complex effects will be internal to the fx server. The `vps` interfaces with the fx server only to specify what desired effect is required. The details of realizing the effect by exploiting the available parallelism is left to the fx server. The fx server must address two issues when executing a desired effect. First, how many fx processors will be required to execute the effect and achieve an acceptable throughput? Second, what type of parallelism should be exploited?

To arrive at answers for these questions, we plan on using a hierarchical configuration structure within the fx server. At the topmost level of the hierarchy, the desired effect and the input video streams are represented as they were provided by the `vps` system to the fx server. The fx server evaluates the desired effect to determine if a single fx processor can realize the effect with an acceptable throughput. If this is not possible, the fx server decomposes the desired effect into two or more separate pieces by exploiting one of the three types of parallelism available. Each sub-piece is then recursively treated as a new effect to be realized. The question of what type of parallelism is to be exploited is answered at each level of this configuration hierarchy, allowing for a hybrid mix of temporal, spatial, and functional decomposition. The question of how many fx processors is required is answered by the number of leaves in the configuration hierarchy.

A number of optimizations to improve performance must be made when constructing the configuration hierarchy and executing the effect. One optimization is to balance the time spent building the configuration hierarchy and the time spent actually performing the effect. In most cases, a desired effect will only be required for a finite period of time. For example, a fade effect may be used to transition between two different camera views and will only be required for a relatively short period of time. When an effect is requested, the fx server must evaluate a number of different possible configuration hierarchies that could be used to execute the effect. Essentially, this task amounts to searching the space of possible configurations and predicting the performance of each. The time spent in constructing and evaluating these hierarchies must be proportional to how long the effect will last. To this end, accurate heuristics and cost models to predict the performance of various configurations need to be developed.

Another optimization is to dynamically balance the number of fx processors used by several different configuration hierarchies. Several separate effects may be required by the vps at the same time. For example, the vps may be used to execute a fade transition between one camera view and the result of a chroma-key effect using a different camera view and a still image. The chroma-key effect and the fade effect will be two separate effects that are active at the same time. Although each effect will have its own configuration hierarchy within the fx server, the computing resources available must be shared between them. As new effects are requested, the number of fx processors used by effects already being executed must be dynamically changed to accommodate the needs of the new effect. Similarly, as effects are completed, fx processors that are no longer in use must be distributed to effects that are still under way.

6 SUMMARY

The deployment of IP Multicast and MBone conferencing tools make it possible to produce and broadcast live programs on the Internet. In this paper, we described the design and implementation of a software-only video production switcher, `vps`, that enhances the quality of MBone broadcasts. `vps` provides special effects processing to incorporate audience discussions, add titles and credits, and integrate stored videos into the presentation. We showed an example scenario of `vps` in use. We also discussed the software architecture of `vps`. The developed prototype has demonstrated the feasibility of `vps`. Future work will improve special effects processing.

ACKNOWLEDGMENTS

The first prototype of this system was designed and implemented as a class project by David Simpson and Richard Fromm. Soam Acharya implemented the mix and fade effects in the compressed domain.

References

[1] S. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems (TOCS)*, 18(2):85–110, May 1990.

[2] H. Eriksson. Mbone, the multicast backbone. *Communications of the ACM*, 37(8):54–60, August 1994.

[3] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *ACM SIGCOMM*, pages 342–356, August 1995.

[4] Plateau Multimedia Research Group. The berkeley continuous media toolkit. Documentation available at URL http://bmrc.berkeley.edu/projects/cmt/cmt.html.

[5] J. Handley and V. Jacobson. Sdp: Session description protocol. *Internet Draft, work in progress*, November 1995.

[6] R. Malpani. Floor control for large-scale mbone seminars. Master's thesis, University of California, Berkeley, CA, 1997. Submitted for publication to ACM MM '97.

[7] S. McCanne and V. Jacobson. vat: The lbnl audio conferencing tool. 1995. Available at URL ftp://ftp.cs.berkeley.edu/ucb/sggs/.

[8] S. McCanne and V. Jacobson. vic: A flexible framework for packet video. In *ACM Multimedia*, pages 511–522, San Francisco, CA, November 1995.

[9] G. Millerson. *The Technique of Television Production*. Focal Press, 12 edition, 1990.

[10] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time application, January 1996. RFC 1889. Avaiable at URL ftp://ds.internic.net/rfc/rfc1889.txt.

[12] B. Smith and L. Rowe. Algorithms for manipulating compressed images. *IEEE Computer Graphics and Applications*, September 1993.

[13] A. Swan. An internet mbone broadcast management system. Master's thesis, University of California, Berkeley, CA, 1997. Submitted for publication to ACM MM '97.

[14] D. Wetherall and C. Lindblad. Extending tcl for dynamic object-oriented programming. In *Proceedings of the Tcl/Tk Workshop'95*, Toronto, July 1995.