The EFX Editing and Effects Environment

Sherman R. Alpert, Mark R. Laff, W. Randall Koons, David A. Epstein, Danny Soroker, David C. Morrill, and Arthur J. Stein
IBM Entertainment Systems

The EFX digital editing and effects environment integrates facilities for nonlinear editing of digitized film, video, and audio with sophisticated image-manipulating special effects. EFX offers an intuitive, visual, direct-manipulation user interface for building multimedia compositions. This front end, discussed here, is coupled with a powerful parallel-processing computer that computes the special effects and plays back the uncompressed digitized film or video in real time.

With the near ubiquity of multimedia delivery and playback, viewers have become more discriminating in their expectations of the content and form of multimedia presentations—they want polished and refined video with sophisticated visual effects. Multimedia producers thus need better and more sophisticated tools for content creation. To that end our group has been developing digital tools for the entertainment industry to facilitate multimedia creation.

One computational tool we have built is the EFX digital editing and effects environment. EFX integrates facilities for nonlinear editing of digitized film, video, and audio with an extensive variety of image-processing special effects. EFX allows users who possess the raw video and audio for multimedia, television, or film to edit and combine these elements into coherent compositions that play out over time.

EFX provides an intuitive, visual, direct-manipulation user interface for building multimedia compositions. It couples this front end to a powerful parallel-processing computer that computes the special effects and plays the uncompressed digitized film or video in real time.

Compositions may incorporate simple transitional effects such as fades, wipes, and dissolves. EFX also provides the full range of effects you would expect from a dedicated studio-quality switcher, plus the entire set of wipes standardized by the Society of Motion Picture and Television Engineers (SMPTE), each with borders of variable softness, variable size, and variable color. Other features include 3D digital video effects (DVEs) such as page turns and flower petals; various color correction and color channel manipulators; a host of image warps such as ripples, vortices, bubbles, and pinches; operators to crop, scale, rotate, cut, and paste images; and multilayer chroma-, alpha-, or matte-keyed composites.

Throughout the design and implementation of EFX, we had two goals:

1. To provide the facilities and operations media editors expect, as well as those necessary to permit—and indeed ease—the multimedia composition process. Our interface decisions were motivated by a desire to develop a system whose use matches the goals, needs, and expectations of our intended users.[1]

2. To provide a system whose primary interaction style is visual, while satisfying and, in fact, making possible goal (1).

To meet these goals, we engaged in user-participatory design, working closely with professional editors. The formative evaluations resulting from this cooperation stimulated a great deal of iterative redesign and benefited the overall design of EFX tremendously.

This article focuses on the EFX front end. Throughout, we attempt to highlight features of EFX that are uncommon or unique with respect to other nonlinear editing systems.

The EFX virtual studio

Before detailing the EFX user interface, we must briefly outline the configuration of a typical EFX "edit suite." The EFX system architecture employs a client-server model: users assemble multimedia compositions by interacting with a personal computer or workstation. The client application running on this workstation communicates image-operation commands via a local area network connection to a back-end server application.

Our current server application resides on an IBM Power Visualization System (PVS), a powerful parallel-processing platform (a full description of the PVS's hardware and software capabilities appears in Epstein et al.[2]). Digitized video and audio files reside on the server, and the server application performs the computations that implement special effects on these media. Images are stored, manipulated, and displayed at full resolution—that is, they are not compressed. Thus, once editing decisions are made and effects are in place,

1070-986X/96/$5.00 © 1996 IEEE

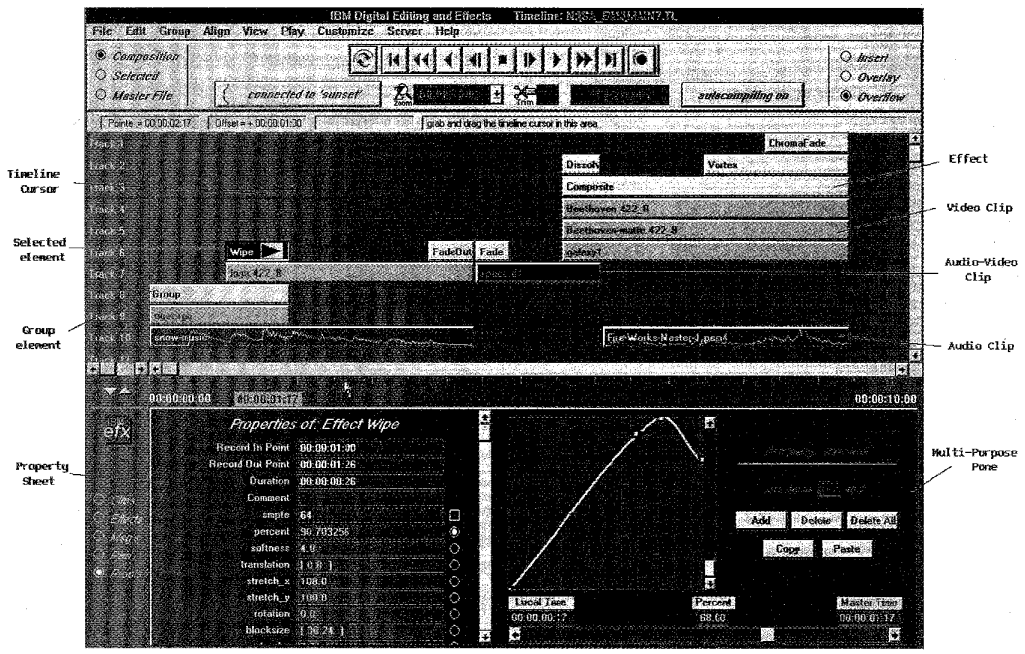


Figure 1. The primary EFX user interface. The timeline appears in the middle section. The interface is "malleable" in that (1) the multipurpose pane is reused for the Clip List, the Effects Palette, a text editor for modifying Effect Definition Files, a subwindow for server messages (errors and others), and the Property Sheet, and (2) the sizes of the timeline and the multipurpose pane are adjustable, ranging from the timeline being completely hidden to occupying the entire area they both now occupy.

sions may occur concurrently. Further, users may rough-cut compositions independently of the back-end server by using the user interface alone. Imagine, for example, a scenario wherein an editor constructs a rough edit on her laptop while commuting to the studio via rail. Once at work, she connects to the server and views the composition-in-progress on the output monitor.

EFX uses the PVS as the system server because we wanted to work with high-quality uncompressed film and video, provide for real-time playback (at 30 frames per second for NTSC video and 24 fps for film), and incorporate a wide range of computationally expensive image-processing effects. Of course, this results in an expensive solution, one that would be labeled a high-end professional system not practical for desktop users—but computer power-per-price doubles about every 1.5 years, so a much lower cost solution with the same capabilities is only time away.

the result can be recorded directly to tape and the process is complete. This procedure contrasts with most nonlinear editing systems, which allow users to view and work only with compressed images. Once editing decisions are made with such systems, the entire process is duplicated in a so-called "on-line" suite using the full-resolution source material on tape.

Further, rather than locking users into a single format as many editing systems do, the image data in EFX may be of multiple resolutions and formats. So, for example, 10-bit RGB digitized film images can be combined with D1 (CCIR601) video images in the same composition.

Playback of audio and video from the PVS server occurs via video controllers (frame buffers) attached to monitors and, optionally, to tape recording devices. Hence, the standard configuration for an individual EFX editing station is a "two-headed" model: the user interface resides on one monitor attached to a PC, and the full-resolution computed video appears on a second monitor attached to a PVS frame buffer. Multiple EFX edit stations may be connected to a single server, and multiple edit ses-

Composing in time: The visual EFX interface

Composing a multimedia piece is equivalent to programming in time. Users must specify "where" in time particular video and audio clips start, how long each lasts, which play before which others, and so on. Thus, our user interface must incorporate a representation for time. We must also represent effect-image *relationships*—when do effects occur, and to what images do they apply? EFX provides an intuitive user interface to accommodate this two-dimensional programming task.

In EFX, multimedia compositions are defined visually, via the topology of rectangular media elements placed and manipulated within a graphical, timeline-based environment. Users control media elements representing clips and effects by pointing, clicking, and dragging within this timeline.

It is commonly understood and agreed in the human-computer interaction community that direct-manipulation interfaces of this sort benefit

users because they are intuitive, comparatively easy to learn, and concomitantly easy to use.[3,4] Rather than an abstract computational language, composing is accomplished via a visual language[5] that matches the way users might intuitively think about media composition.

The "two-dimensional" timeline

Figure 1 shows the main EFX interface. Media compositions are defined in the timeline, which is delineated by *timecodes*. Timecodes are the standard time representation for video work, consisting of the hour, minute, second, and frame number defining a point in time in a composition. They use the format HH:MM:SS:FF.

The timeline is populated with multiple vertically layered "tracks" that serve as containers for any number of media elements (video clips, audio clips, and effects). Together, these interface elements provide a comprehensible yet powerful metaphor for controlling both the time and effects-layering aspects inherent in the media composition process.

As mentioned, our interface's first requirement is to encode time. Compositions in EFX are laid out in time horizontally, with earlier timecodes to the left of later ones. Media elements are added to the timeline and positioned at physical locations that represent temporal locations within a composition. Each media element is portrayed as a rectangular bar whose left and right edges specify the element's starting and ending times, respectively.

The second dimension the interface must represent concerns the dependencies among effects and other media. That is, we require a visual mechanism for specifying which media a particular effect is to act upon. For this we employ the vertical dimension of the timeline. Effects are applied to media that co-occur in time and that are beneath them. Thus, at any single point in time in a composition there may exist multiple clips and effects in multiple tracks. Each effect is applied to the media below it, and when multiple effects co-occur in time, they are iteratively applied in a bottom-up, vertically layered fashion.

As a simple illustrative example, suppose we wish to have a color video clip appear black-and-white in a composition. This requires applying a *monochrome* effect to the clip. To do so in the context of the EFX timeline, we simply need to (1) have the *monochrome* effect temporally span the duration of the clip (that is, the effect must start at, or before, the start of the clip and end at, or later than, the end of the clip); and (2) place the effect element in a higher track than the clip (see Figure 2a). The effect then acts upon the video clip beneath it.

The layering metaphor in EFX goes further than this simple example portrays in that it allows for the iterative application of effects: multiple effects may be layered on top of other effects to any extent required by the composition. "Inputs" to an effect need not be simple video or audio clips, just the appropriate media for the particular effect; visual effect "outputs" are merely image data, identical to digitized clip data. Our example *monochrome* effect, for example, expects one visual image as its subject. That image may, of course, be a raw video clip, but it may also be the resultant output of another effect. So, extending our example, if we wanted our video to appear as a black-and-white photographic negative, we could apply a *negate* effect to the result of the *monochrome* effect. In EFX, we would add a *negate* element to the timeline in any track above the *monochrome* effect, ensuring that the *negate* spans the duration of the *monochrome* element (see Figure 2b).

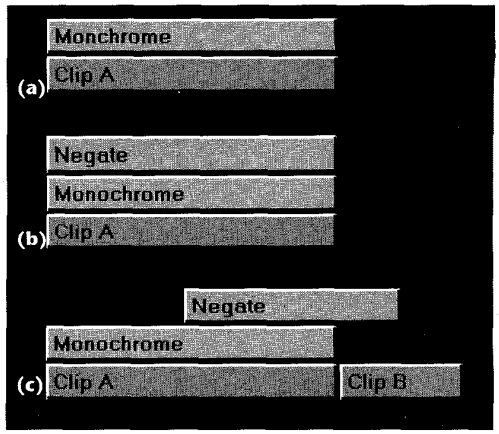


Figure 2. EFX's 2D timeline and effects layering scheme. The horizontal dimension represents time; the vertical dimension encodes effect-image relationships. (a) The elements required to apply a monochrome effect to the entire duration of a video clip; (b) applying a negate effect to the image resulting from the monochrome effect; (c) positioning the negate to act on the later portion of the monochrome effect's output and the early portion of a second raw clip.

Another typical use of EFX's multi-effects capability is the creation of multilayer composites, where the result of one composite becomes the input to a second, and so on. This facilitates, for example, the placement of multiple foreground images from multiple source materials onto a single background image. EFX imposes no limit on the depth of such effect-and-image layering; the user may add new tracks to the timeline at will, and effects may be layered to any extent required by the user's needs or imagination. (Of course, there is a concomitant downside—increasing the number of layered effects means increased processing time to compute the final image.)

Thus, we have a 2D, purely visual strategy for media composition. (Compare, for example,

Figure 3. The Effects Palette.

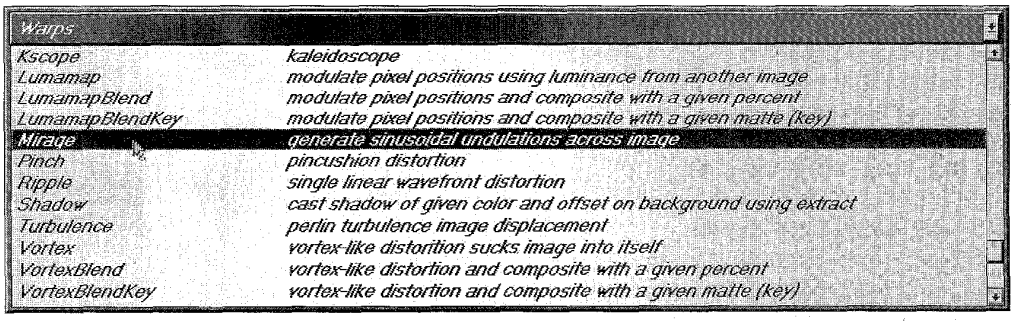
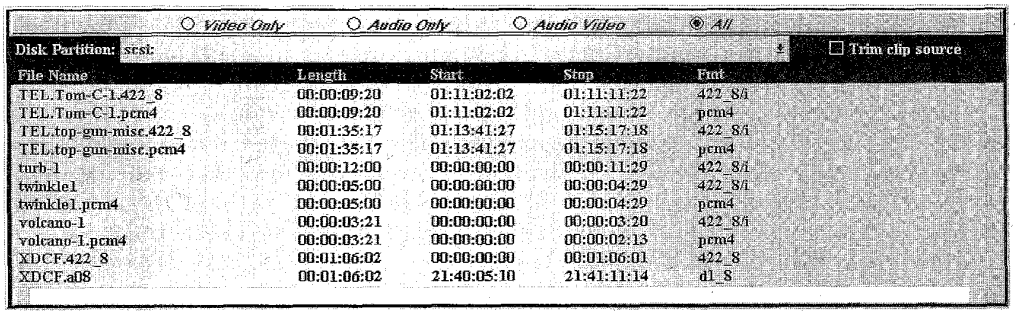


Figure 4. The Clip List.



Koegel et al.[6]) The temporal aspect inherent in media compositions is represented by the horizontal dimension of the timeline. The layering aspect, which permits the iterative application of effects, is represented by the timeline's vertical dimension, where multiple elements may occur concurrently.

Numerous timeline-based interfaces for multimedia composing have been suggested by researchers as well as being employed in a number of commercial editing tools.[7-9] Some of these systems share common characteristics with EFX; however, EFX goes further in a number of ways. First, the EFX timeline does not contain a fixed number of tracks, nor are any tracks dedicated to a single media type. Rather than limiting users to, say, one or two image tracks, a single effects track, and a fixed number of audio tracks, the EFX timeline is free-form: it lets users place elements as they wish. Timeline elements of differing types (video-only, audio-only, audio-video, effects) are differentiated simply by distinct colors (as in Figure 1). So, if a user wishes to have elements representing a video segment and its accompanying audio clip in contiguous tracks without an intervening effects track, she may. Of course, if a user chooses, separate tracks may each contain a specific media type, and track titles may be easily modified to more meaningfully reflect the tracks' contents (for example, "voice-over"). However, EFX leaves these organizational issues up to users' preferences without imposing a limiting structure.

Second, the EFX timeline allows for any level of iterative layering of effects. As we've seen, rather than limiting users to a fixed number of tracks, new tracks may be added and effects may be layered to whatever extent users find necessary.

Adding media elements to the timeline

At the bottom of the main EFX window is the "multipurpose pane." At any time, this single portion of the interface displays any one of multiple subwindows. The goal is to conserve valuable screen space by reusing a single area for many purposes. To the left of the multipurpose pane, a group of radio buttons governs which particular subwindow is visible at any given time.

Two of the subwindows are used to add new media elements to a composition in the timeline. The first, the Effects Palette, appears when the user selects the "Effects" radio button (see Figure 3). The upper portion of the palette contains a drop-down list of selectable special-effects categories. The list below that contains the effects available within the currently selected category. The second subwindow, the Clip List, displays information about the digitized video and audio clips residing on the back-end server's disk array (see Figure 4). Users may create these files via EFX's "capture" facility, which can digitize source material from

up to four tapes simultaneously. The Clip List may also be filtered using the radio buttons at the top of the pane so that it displays those clips containing only video (that is, no audio), only audio, both audio and video, or all of the above.

Users add media elements to a composition by first selecting a clip in the Clip List or an effect in the Effects Palette, then specifying visually where the new element should appear in the timeline (representing when the element will appear in the composition). The user drags out a rectangle for the element within the timeline, with the rectangle's left and right edges defining the element's start and stop times and its top edge specifying the element's track. In the case of clips, the rectangle's width—representing the duration of the clip segment in the composition—is constrained by the length of the clip's source material.

Once elements reside in the timeline, users can modify them directly via the mouse. For example, a user could reposition an element in time (and track) by simply dragging it to a new location. An element may be trimmed (that is, its duration and start or stop time changed) by grabbing and dragging either of its sides to stretch or compress its length.

Media elements may be selected in several ways, making them available to generic operations such as cut, copy, and paste, and domain-specific operations such as splitting, trimming, and temporal alignment. Additionally, in EFX, unlike other nonlinear editors, tracks themselves are modifiable as entities. Users can select entire tracks of elements with a single mouse click, making them subject to operations such as cut, copy, paste, and insert.

All media elements in an EFX composition have equal "status." A number of digital editing systems embed transitional effects, such as fades, within clip elements and portray them on the screen as, for example, an oblique line crossing two adjacent clip elements (see Rubin[9]). In EFX, all effects are first-class timeline elements. They are all directly accessible and manipulable by the same mechanisms—all may be dragged to new positions, directly resized, and so on. Also, as we'll discuss later, all possess individual, unique properties that govern their precise appearance and behavior and are available for user modification on a per-element basis.

Multiple editing modes

When new elements are added to the timeline and when existing elements are dragged and dropped, we must decide where the elements should ultimately reside. For example, if a media element is dragged on top of another, what happens? Should the dropped element overwrite a portion of the element "beneath" it? Should the dragged element jump to a position just after the other element? The EFX timeline possesses multiple modes that govern the behavior of the system in these situations.

The first timeline mode, *insert* mode, simulates film editing, where splicing a clip into a longer piece moves all subsequent clips to points later in time. While in insert mode, as an element is dragged, EFX displays a dynamic insert indicator showing where the element will be placed if the user drops it at that point. The location of the insertion point is that keyframe (start or stop point of existing timeline elements) nearest the left edge of the dragged element. When the user drops the element, it is inserted with its head aligned at the most recent insertion point, and all elements to its right are "pushed" further to the right. When the user deletes an element, all elements on its right "spring back" to the left.

Overlay mode mimics video editing. When a clip is recorded onto a particular segment of video tape, it overwrites whatever was there previously. Thus, in overlay mode, when the user drops an element on top of other elements, it remains there, overwriting (and thus deleting from the composition) those elements, or portions of elements, beneath it.

In *overflow* mode, the temporal integrity of all media elements is always maintained—no elements are deleted or repositioned in time. Instead, when element *A* is dropped on top of element *B*, EFX makes room for both of them: Element *B* remains in its current position, and EFX attempts to relocate *A* in the track below *B*, without modifying *A*'s start or stop time. If other elements in that lower track temporally overlap *A*, EFX automatically inserts a new track below *B*'s track, pushes all those tracks (and elements) below *B* down one track, and places element *A* in the newly inserted track.

Viewing the composition output

With EFX we are, of course, creating a visual product. During the composition creation process, users must be able to simultaneously view the results of their ongoing work. Thus, at all times EFX displays the *current result image* on the video monitor. This "current" image results from all the clips and effects traversed by EFX's *timeline cursor*, a line

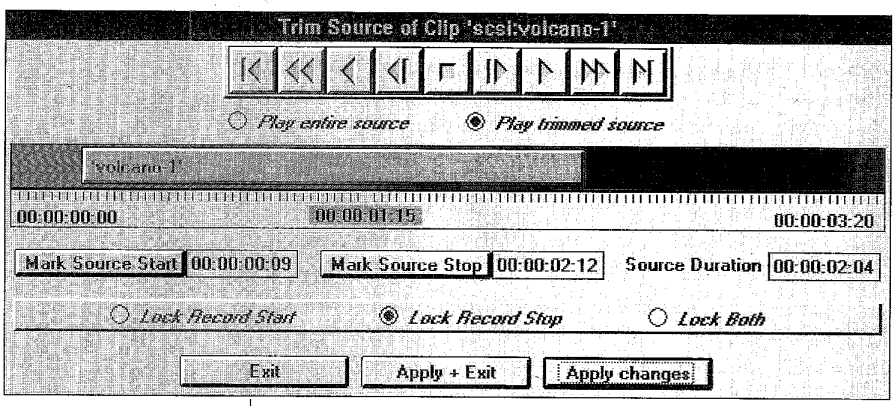


Figure 5. The Trim Clip Source window. With this control, users may trim the source material associated with a clip.

that extends vertically through the timeline tracks and indicates the current timecode location within the composition (see Figure 1). Users may scan (or "scrub") a composition or directly jump to any point within it by manipulating this "current" position. That is, they can click at specific timecode locations in the area near the bottom of the timeline or drag the timeline cursor with the mouse. As the user manipulates the timeline cursor, the video monitor updates to show the image created by the combination of media elements at the precise point indicated by the cursor's position. Thus, EFX supports true random access of a composition's resultant full-resolution output.

A user may also change any layer (that is, any element) traversed by the timeline cursor, and the output monitor will automatically update to reflect this new set of elements. Thus, much as in a spreadsheet, where changing a number in a single cell immediately propagates updates to all related cells, any change to an element under the timeline cursor—whether modification of the element's properties or substitution of an entirely new element—immediately propagates to effects above it and to the resulting output image.

Trimming clips

No task is more intrinsic to the editing process than trimming video and audio segments. The editor needs to include just the right material in the composition, with just the right rhythm, just the right duration, and just the right starting and stopping moments.

We have already seen some methods of trimming elements within the timeline, thus changing their temporal location or duration within the target composition: Users may slide composition elements in time by simple dragging and change start or stop times by sliding the appropriate edge of an element. Additionally, multiple elements may be precisely trimmed simultaneously. After selecting the elements of interest, users may trim multiple elements' start or stop times to the left or right with simple keystrokes. Of course, with either direct manipulation or keystrokes, stretching of a clip element is constrained by the duration of the source material associated with that clip.

While the above methods allow for trimming of audio and video within the target composition, EFX also provides the ability to trim the source material underlying those timeline-resident elements. At the very least, for example, a video segment shot for a dramatic piece must be trimmed to begin sometime after the director yells "Action!" and before the director's "Cut!" Or, a clip might occupy just the right temporal location and duration in the timeline, but the source trim is "off"—without changing the timeline element's position in any way, we might need to "slide" the source "underneath" that element, that is, have it start and end sooner or later within the original material. Or, a timeline element might end at just the right time for the composition, but the editor wants to try out different starting points within its source material. To accommodate these requirements, the user may open a Trim Clip Source window on any timeline element.

The Trim Clip Source facility, shown in Figure 5, includes a "mini" timeline whose duration and boundary timecodes represent the entire source material available in a single raw clip. The sole media element in this timeline represents the temporal location and duration of the trimmed segment within the entire source clip, that is, the portion of the clip that will actually be used in the composition.

While the Trim Clip Source window is in use, the frame buffer monitor displays video from this source material rather than the main timeline's output. As in the main timeline, the image actually displayed depends on the location of the timeline cursor within the mini-timeline.

The Trim Clip Source window provides a number of methods for trimming the source. To slide the source under a composition element without changing its length, the user need only slide (drag) the mini-timeline's element. Alternatively, the start and stop times of the trimmed source may be modified, again by dragging one of the

element's sides. As the user does so, the local timeline cursor moves with the dragged edge, and the video monitor displays the corresponding frames of the source material, allowing the user to watch for the desired moment at which the trim should occur.

Another method for trimming the source start or stop is to drag the timeline cursor itself, watching or listening for the correct moment for the trim, then pressing the "Mark start" or "Mark stop" button. Or, if the precise timecode for the start or stop is known beforehand without visually scanning the source material, the user may simply type the timecode in the appropriate entry field and press the appropriate "Mark" button.

Customizing effects: Property sheets

In addition to placing effects where appropriate in the timeline, users require more complete control over the precise behavior and appearance of effects. The appearance of special effects and transitions is governed by their unique properties, or set of controlling parameters. Further, each parameter may vary in value over the effect's temporal span. For example, the amplitude of a *bubble* might start small and increase over time.

To gain access to effect properties, users open the *property sheet*. (Figure 1 shows the main user interface with a property sheet opened on the currently selected element.) Property sheets provide both textual and graphical control over an effect's time-varying parameters. In this way, EFX permits full, user-controlled customization of effects. The result is a virtually infinite number of effects. Further, effects such as accelerating dissolves or oscillating wipes may be defined (and iteratively redefined) for repeated, consistent performance, rather than requiring manual overrides (as with studio switchers), which might be difficult to replicate.

As shown in Figure 1, the property sheet for an effect contains radio buttons associated with each of its user-modifiable parametric properties. When one of these buttons is selected, the property sheet's graph view (in the right portion of the property sheet) changes to portray the graphical representation of the associated property. This graph defines the property's value and how it changes over time. The horizontal dimension represents time, and the vertical dimension represents the parameter's value at each point of the effect's duration. Directly manipulating this curve modifies the parameter's values. Thus, users can customize effects via purely visual mechanisms.

For example, suppose we have a simple left-to-right *wipe*. By default, it traverses the video image in a linear fashion. That is, as time progresses, the *wipe* travels continuously from left to right across the image. The position of the *wipe*'s leading edge is defined by its "percent" property. If we were to look at the default graph view for "percent," its curve would be linear, proceeding from the lower left to the upper right. The linear progression represented by such a curve implies that at its start (at time 0 of the curve), the *wipe* will have progressed 0 percent across the image; at the midpoint (in time), it will have progressed 50 percent across; and by the end, the *wipe* will have progressed to 100 percent.

In some situations an editor might not wish such an orderly progression. Manipulating the curve via the mouse alters how the *wipe* progresses over time. For example, if the editor wants the *wipe* to be at 100 percent prior to the end of the *wipe* element's duration, she may double click on the curve at, say, its midpoint. Doing so creates a new keyframe for the curve, represented visually by the addition of a "handle" (a small square) at that point. Since all of the curve's keyframes represent draggable points, the editor would simply drag this keyframe to the top of the graph, representing 100 percent. She might also drag the last keyframe in the curve (at the end of the effect's duration) to a lower position, representing something less than 100 percent (see Figure 1). EFX automatically provides for smooth transitioning between keyframe values by interpolation: values for all points between keyframes are supplied via a spline passing through the keyframes. The curve would thus result in a *wipe* that completely traverses the image left to right, then "backs off" toward the left.

The "percent" parameter is scalar. That is, at any point in time it is a single numeric value, and this value might or might not change over time. Effects may contain time-varying parameters of other types. Nonetheless, they are still represented graphically. For example, the "region" property of a *crop* effect specifies the rectangular portion of the image to extract (to be pasted into another image, for example). Logically, the graph view for this property displays a rectangle that the user may directly shape and size by dragging. At different keyframes, the rectangle may differ in size; thus, it may vary over time as well. A *paste* effect possesses a property that specifies the top-left position for pasting a rectangular image. The graph view for this type of property—that is, a point within an image—lets the user visually specify the point by

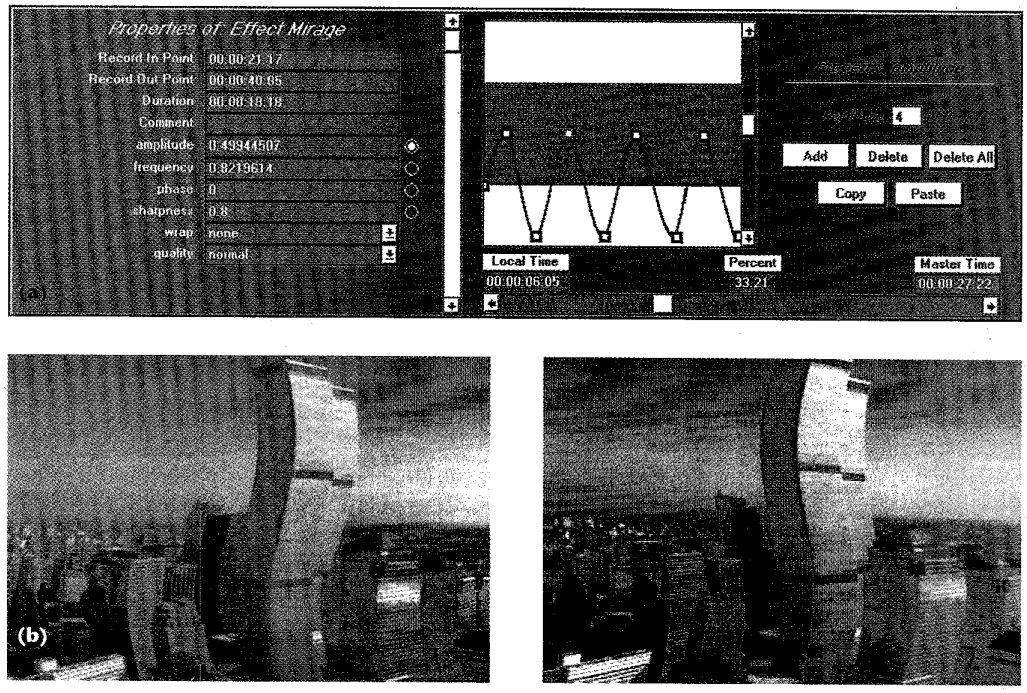


Figure 6. New York City dances. (a) The property sheet for a mirage effect being applied to a New York skyline flyby. The curve for the "amplitude" property is shown. The keyframes for the property occur at downbeats of an accompanying music clip, and alternating positive and negative values in the curve result in "bending" the image in opposite directions at each keyframe. The interpolated values between each keyframe cause the building to sway smoothly between keyframe extremes. (b) Two still images from the resulting video.

dragging a small square to the desired location.

Note that as the user manipulates the curve, or the rectangle, or the point, in a property sheet, the resultant image displayed on the frame buffer monitor continuously and simultaneously updates. So, the user is not "flying blind," approximating placement of the *paste*, for example. As she manipulates the property sheet's graph view, the user sees the resulting image.

The value of an effect parameter at each keyframe may also be modified textually. Once a user clicks on a keyframe, the parameter's value at that point appears in the associated textual entry field. Modifying the entry-field text automatically modifies the curve. EFX provides this mechanism in addition to direct modification of the curve because, in some situations, it provides more precise parameter specification; in other instances, the visual mechanisms may be superior or easier. EFX provides both and leaves the method to users' preferences.

Property sheets thus vastly broaden the range of special effects. Users invoke their creativity not only in deciding which effects should occur at specific points in a composition, but in customizing such effects in completely unique ways. This opens a world of creative possibilities. One of the authors, for example, applied what EFX calls a *mirage* effect to a video "flyby" of the New York City skyline. In EFX, a *mirage* effect "bends" an image, creating a vertically undulating result; the amount and direction of undulation depend on the effect's "amplitude" property. He added keyframes to the curve for "amplitude" so that each keyframe occurred at a downbeat of an accompanying music clip. Then he dragged the curve's keyframes into a sawtooth-like wave form, with alternating keyframes specifying a positive and negative "amplitude," respectively (see Figure 6a). This reversed the direction of "bending" of the image at each keyframe. The resulting composition portrays the World Trade Center and the rest of the lower Manhattan skyline dancing back and forth to the beat of the music (see Figure 6b).

Temporal alignment

Multimedia composition demands specific types of temporal coordination among media elements. We might require two (or more) media elements to start at precisely the same time (for example, a special effect and a video clip). We might want multiple elements to end at the same timecode or to occupy exactly the same time span (that is, to start and stop at the same times, respectively). One element might have to start just when another element ends (two video clips with a simple cut transition between them, for example), or an element might have to precisely span the temporal overlap of two other elements (such as with a dissolve transition "between" two clips). We might wish to have a media element exactly fill the gap in time between two other timeline elements, and so on.[7] Thus, our application requires the ability to specify such temporal relationships among elements.

The syntax and semantics of the visual language of the EFX timeline imply that aligning elements in time means graphically aligning them in the vertical dimension. The solution, then, is to provide a rich repertoire of mechanisms for the graphical—and thus, temporal—alignment of media elements.

Alignment with the help of "gravity." The simplest—and crudest—method of aligning elements is by directly dragging them within the timeline. If we wish the start time of clip *B* to align with the stop time of clip *A*, we simply drag *B* directly next to (to the right of) *A*. This method "works," but is, at best, time consuming.

Alignment can also be performed with the help of "gravity." Normally, dragged elements move smoothly, tracking the mouse location. However, when gravity is in effect, a dragged element "snaps" to the next significant alignment location, or keyframe, as the cursor moves left or right.[10] Here, "keyframe" refers to the start and stop times of all media elements currently in the timeline. Gravity may be applied to the left or right edge of the dragged element, so its left or right edge snaps to the next keyframe. This method greatly simplifies alignment to the start or stop timecode of another element: As dragging occurs, the dragged element automatically snaps into alignment with other elements' keyframes.

Making this more concrete, assume a timeline with only two elements. Effect *B* is somewhere off to the right of clip *A*. When we apply gravity to the left edge of *B* and drag *B* to the left, it first jumps to a position where its left edge is aligned with *A*'s right edge. As dragging continues to the left, *B* snaps to a position such that the left edges of *A* and *B* align exactly.

More automatic alignment facilities. The above methods, although useful for some situations, might prove cumbersome or time-consuming for others. Further—and perhaps more importantly—they only handle a limited subset of the necessary alignments (aligning an element's start or stop time to another element's start or stop). Our domain requires mechanisms for many types of temporal alignment; thus, EFX offers several other, more automatic, alignment facilities. We look at two of them here: menu alignment and immediate alignment of new timeline elements.

1. *Menu alignment.* In the most straightforward of EFX's automatic alignment methods, a user selects two or more elements to align, then chooses an operation from the "Align" pull-down menu (or presses the associated keyboard shortcut keys). Most menu-based alignments of timeline elements depend on selecting a unique primary element, the element to which others will be aligned in some manner. When multiple elements are selected, the primary element is the one selected first. So, for example, a user might select element *A*, then extend-select elements *B* and *C*; *A* is the primary and *B* and *C* are the secondary elements for alignment.

The user may select as many secondary elements as desired. When an alignment operation is invoked, the primary element always remains unchanged, and all of the secondaries are modified in some way to align with the primary (dependent, of course, on the type of alignment chosen). For example, the user might select several elements and choose "Align Start Times by Moving" from the "Align" menu. As a result, all secondary elements move to locations where their start times precisely align with the primary element's left edge. Another example appears in Figure 7.

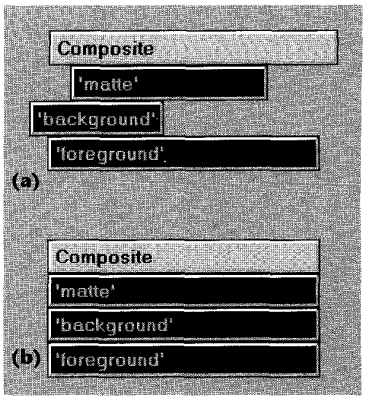


Figure 7. Spatiotemporal alignment of media elements. (a) Four media elements selected for alignment. The bottom clip is the primary selection; the others will align to it. (b) The result of invoking the alignment operation "Both start and stop times" from the pull-down menu.

The "Align" menu includes several different alignment types. The first three are operations to align elements start-to-stop (such that their start and stop timecodes are contiguous), align their start times, or align their stop times. For each of these, the user can direct the alignment operation to either move the secondary elements or stretch them. In the latter case, only one side of each secondary element (representing its start or stop time) changes to bring it into alignment with the primary. The fourth alignment operation lets users coerce all secondary elements to span the same time period as the primary by aligning their start and stop times, respectively.

Two other alignment operations let users select two elements as primaries and a unique third element to be modified by the alignment. The first is the "transition alignment" operation, whereby the third element is modified to precisely span the overlap of the two primary elements. This alignment operation is commonly employed to place a transitional effect, such as a *wipe* or *dissolve*, between two clips (see the *wipe* in Figure 1, for

example). The second such operation, the "fill gap" alignment, modifies the start and stop times of the third element to precisely fill the time gap between the two nonoverlapping primary elements. The three selected elements then become successively contiguous in time.

2. *Immediate alignment of new timeline elements.* Normally, to add a media element to the timeline, users click on an entry in the Effects Palette or Clip List and drag out a rectangle for the new element. Alternatively, they can precisely place new elements in the timeline with specific start and stop timecodes immediately as they are selected in the Effects Palette or Clip List. That is, the user merely clicks on a list entry while pressing an augmenting control key at the keyboard, and the new element appears in the timeline with the appropriate size and alignments, without the user having to manually specify the element's rectangle.

Three types of "immediate" alignment are available upon element creation, corresponding to alignments commonly required in typical editing scenarios. The first is the "hop on top" alignment. We might wish, for example, to add an image-manipulating special effect that exactly "covers" (and thus operates on) a particular video clip. This effect would need to be inserted in a track above the clip and have start and stop times exactly matching the clip's. Or, we might want to add a voice-over audio clip accurately spanning a set of video clips and associated effects. In these cases, the user first selects the elements to be "covered," then clicks on a list entry while pressing the appropriate control key. The new element automatically appears in the timeline in the track above the uppermost selected element, and with start and stop times that precisely span the duration of all the selected timeline elements.

A second common editing scenario involves two clips with a *wipe* or *dissolve* transitioning between them. Again, because this operation is so frequently required, we want to facilitate the immediate placement of transitional effects. Thus, as a variation of the "transition alignment," a user may select two temporally overlapping timeline elements, clips *A* and *B* for example, then click on an Effects Palette entry while holding the appropriate control key. The new effect element instantly appears in a track above both *A* and *B*, and with start and stop times exactly spanning their overlap.

Lastly, EFX offers the direct and speedy replacement of existing timeline elements with new elements. That is, a new element may be added immediately in place of (and thus with the same start and stop times as) another timeline element. This turns out to be useful in a number of situations. For example, it lets EFX users freely experiment with alternative composition choices. A user might create a composition with a particular effects-layering structure or clip-to-clip transitional flow, but experiment by substituting contrasting clips in the same "slot." These clips might be differing versions of the same scene, for instance, perhaps with different camera angles or camera ranges.

Another reason to use the replace operation is to sample alternative special effects applied to a particular clip. The replace-alignment feature lets users do so easily and quickly without having to repeatedly align a new element within the timeline. Thus, "trying out" alternative clips or effects in the same position is simply a matter of repetitively choosing palette entries and seeing the immediately viewable result.

Grouping

Another unique feature of EFX is the ability to group multiple elements, from any number of tracks, into a single composite element. Once the user has created and refined part of a composition—for example, having perfectly sized, aligned, and coordinated a set of video clips, effects, and audio segments—the interrelated elements may be aggregated into a single unit.

A practical benefit of grouping an arbitrarily complex set of elements is that multiple media elements thereby become directly manipulable in concert. Thus, repositioning in time (for example, by dragging), resizing, duplicating (copy/pasting), deleting, and so on may be performed simultaneously on multiple elements. In this way, groups provide for persistent graphical constraints among media elements,[11] since when a group is dragged to a new position, the spatiotemporal relationships among the group's constituent elements are maintained. Contrast this to the alignment operations discussed above, which provide for "one-shot" temporal relationships among media elements: After elements are aligned, the spatial relationships among them may be "broken" by dragging individual elements. In essence, those alignment operations implement temporary graphical constraints.[10]

Groups afford another benefit: Once a set of elements is aggregated into a single unit, the user can think of the group as a "finished" segment of the composition and turn to composing separate,

even discontiguous, portions of the piece. A group is in essence a *software submaster*—a subcomposition in its desired final state, a discrete portion of a larger composition. A group may be ungrouped at any time, returning its constituent elements to the timeline as individual elements in their original, pre-group spatial configuration (relative to one another). Thus, unlike conventional tape submasters or even disk submasters created by a digital editor, groups retain the ability to have any element replaced or any effect parameter altered at any time. Ungrouping also dissolves the persistent constraints among the group's elements.

Compiling a timeline

While constructing a composition in the EFX timeline, and upon its completion, we want to actually view the resultant multimedia piece. To do so, the timeline-based visual/spatial specification of the piece must be translated into a process for computing the audio and visual output.

The back-end server performs the computations necessary to apply special effects to images and display resultant compositions. The server's image-processing application provides not only a rich set of audio and video processing functions but also a general-purpose, programmable, stack-based virtual machine. Both this general computational model and the specific media-manipulation functions are accessible and controllable via a flexible and powerful textual programming language. The client application communicates with the server using this programming language interface.

Thus, to "play" a timeline-resident composition, the EFX client application translates the visually specified composition into a procedural program that, when executed by the server, produces the desired sequence of images, sound, and effects. We call this translation process "compiling a timeline." Once the timeline has been compiled, commands from the user interface to the image server also allow users to control execution of the resultant program, including forward/reverse, fast-forward/reverse, pause, and random access.

To compile a timeline, the EFX front end performs a temporal and topological analysis. At the highest level, a code block is generated for each time "segment" (a portion of the timeline delimited by element start and stop points). Within each segment, the physical layering of the media elements in the timeline determines the sequential ordering of the process encoded in the block. The program schedules these elements for execution in a bottom-up fashion, such that the process for the bottommost element is executed first, and so on up to the topmost media element in a segment.

For clips, the generated program reads frame-specific image data from disk and pushes it onto the stack, making it available to the next element in the sequential process. (The server application's stack can reference image data as well as more conventional scalar and vector data.) An effect's code first pops some number of input images off the stack (varying visual effects require different numbers of input images). When the effect operation completes its computational manipulation of its input images, it leaves its resultant image on the top of the stack. There is no difference between raw clip image data left on the stack and image information placed there by a special-effect function. Each effect, in turn, leaves its resultant image on the stack, allowing successive effects to use this image data as input. In this way, this process implements the iterative effects-layering feature of the EFX timeline.

This process iterates for each frame in the segment. For each frame, after all the elements in the segment have been processed, the image remaining at the top of the stack is taken as the final result and displayed to the frame buffer monitor (and, optionally, written to a digital "record" file). When a segment time boundary is crossed while playing a composition, the new segment's code block is invoked in similar fashion.

User-defined effects

Together, the back end's general-purpose programming model and the textual language controlling it have provided great power and flexibility, helping us greatly reduce the development time required to interface the EFX client to the server. Of equal value is the power that the model and language provide users, who may define new effects in the textual programming language itself.

The stack-oriented textual programming language is based on PostScript,[12] so reference manuals and tutorials are readily available. Using a text editor (provided in the multipurpose pane), users may create what EFX calls a *Programmed Effect*—the general-purpose language lets users employ variables, conditional branching, iteration, function calls, and so on to program additional unique image-processing operations like special effects.

All effects are catalogued in an external *effect definition file* (EDF). Each EDF entry specifies an effect name, its effects category (for listing in the Effects Palette), and the name, type, and default

value for each of its customizable properties.

Before looking further at Programmed Effects, let us first examine the EDF entry for a standard effect:

```
[Effect]
  name = FadeIn
  description = Fade in from black
  category = Fades & Dissolves
  input = img, image
  input = percent, normalized,
                              [0 0 1 1]
  output = img, image
```

As defined here, the *FadeIn* effect expects a single image as input, plus a parameter named "percent" whose type is *normalized* (that is, its values may vary over time, and at each point in time its value lies between 0 and 1). The effect's output is image data. The array supplying the default value for "percent" means that at time 0 (at the start of the *FadeIn*), the value of "percent" is 0, meaning the input image's visibility is 0 (not visible); at time 1 (the end of the effect), the value of "percent" is 1 (the image is 100 percent visible). All non-image input parameters, such as "percent" here, appear in an effect's property sheet and are thus modifiable by users.

Programmed Effects are created by adding new entries to the EDF. A Programmed Effect entry looks much like a standard effect except that it includes the PostScript-like code defining the effect's behavior. Consider this example of a user-defined Programmed Effect:

```
[Programmed Effect]
  name = Blend
  description = Composite with
          correctly shaped matte using
Extract
  category = Graphics, Composite
  input = backgnd, image
  input = foregnd, image
  input = pct, normalized, 1.0
  output = image, image
  code = << >> begin
            [/pct /foregnd /backgnd]
{exch def} forall
            backgnd foregnd White pct
Extract foregnd 1.0 0.0 Composite
         end
```

The code for this Programmed Effect blends the "interesting" (that is, non-black) parts of the foreground image onto the background. The amount of transparency of the foreground image, when blended with the background, is a function of the "pct" parameter. The "Extract" operator (corresponding to the existing *extract* effect) creates an intermediate image, a *matte*, in which each non-black pixel of the foreground is replaced by a grayscale pixel whose brightness corresponds to "pct." For example, if "pct" is 100, the non-black pixels would become white, producing a matte that implies "overlay the non-black pixels of the foreground on top of the background." If "pct" is 50, the non-black pixels would become 50-percent gray, meaning "blend the non-black pixels of the foreground with the background image." That is, the foreground becomes semitransparent, so the background partially "shows through."

The matte is passed to the "Composite" operator, which actually performs the image combination. In other words, *Blend* performs a *composite* without having to supply a separate matte; instead it uses the foreground image to automatically create its own temporary matte. Note also that the Programmed Effect creates (and subsequently destroys) a temporary PostScript dictionary for defining local variables.

The "environment" for coding Programmed Effects thus includes not only the PostScript-like language, but predefined constants such as White and the full palette of existing effects as callable procedures. Once a new Programmed Effect is defined in the EDF, the Effects Palette "knows" to include an entry for it, and the new effect type will be available for inclusion in EFX compositions. Thus, users may define and add effects to EFX with no reprogramming of the client or server applications. Of course, we don't expect all EFX users to possess the skills and knowledge required to do their own effects programming, but the capability is provided for those able to take advantage of it.

Users may also define their own variations of existing EFX effects in a simpler fashion that involves no programming. Users define a *Derived Effect* by specifying a base effect and supplying new default values for any of its user-modifiable parameters. For example, the EFX effect *FadeOut* is simply a Derived Effect based on *FadeIn*. Its definition in the EDF follows:

```
[Derived Effect]
  name = FadeOut
  description = Fade out to black
  category = Fades & Dissolves
  base = Fade
  default = percent, [0 1 1 0]
```

A Derived Effect "inherits" all the attributes of its base effect and overrides the default value of one or more parameters. So, a *FadeOut* resembles

a *FadeIn*, except the default curve defining how "percent" varies over time has the image fully visible at time 0 (at time 0, "percent" is 1) and completely invisible at time 1 ("percent" is 0).

By permitting user-defined effects, either Derived or Programmed, EFX attempts to provide a level of openness not found in other nonlinear editors. Such editors tend to be regarded as "black boxes" with regard to user programming or user-defined functionality.

Conclusion

Taken as a whole, the user interface of EFX provides a rich, visual, intuitive environment for multimedia composing. With EFX, users declaratively and visually specify pieces via the spatial configuration of the media elements directly manipulated within the EFX timeline, in effect providing a visual language for multimedia composing.

A major question regarding many visual language environments is whether they can scale up to real-world tasks. We believe EFX does. We believe its direct manipulation interface is intuitive and easy to use, and offers the functionality required by editors of multimedia pieces.

A number of commercial digital nonlinear editors are currently available.[8,9] Many of EFX's features, such as a timeline-based interface, appear in one or more of these applications, although none of them combine all such features in a single system, as EFX does. The salient characteristics of EFX include

- a free-form timeline that does not coerce users into a system-imposed element structure;
- simple, yet intuitive and powerful, temporal alignment facilities;
- the ability to group sets of elements and thereby provide for persistent temporal constraints;
- Property Sheets, which provide precise user control of all parameters defining the behavior of individual effects and offer the ability to do so graphically via a spline-based curve editor;
- unlimited effects and compositing layering—when necessary, EFX automatically inserts new tracks into a composition, and users may do so as well by explicit command;
- openness—a PostScript-based textual language for communication with the back-end server allows users to create their own unique effects and to define effects derived from existing effects;
- the ability to capture (that is, digitize) media from up to four tape sources simultaneously;
- transitional effects (such as *wipe*, *fade*, and *dissolve*) as explicit, first-class timeline elements; and
- a multiuser capability—multiple users at multiple client workstations running the EFX frontend may be attached to a single server and edit simultaneously.

We could, of course, add other features to further speed up and otherwise facilitate multimedia content creation. For example, EFX currently possesses the ability to display an audio waveform, bass detection curve, or silence detection curve for clips containing audio. The bass detection feature can be used to expose the downbeats in a music clip. Since this is represented by a curve over time, we have discussed providing the ability to copy and paste that curve into the Property Sheet graph for an effect parameter. In that way, we could synchronize the dynamic behavior of the effect to the beat of the music (thus, for example, automating the "New York City Dances" scenario). We have also discussed a visual programming tool for defining Programmed Effects. Also, EFX currently lacks audio special effects to complement the wide variety of visual effects already present. We believe the ultimate outcome of adding these features to EFX will be a user-oriented, intuitive editing environment that facilitates the creation of multimedia content. **MM**

Acknowledgments

We would like to thank the professional editors we have worked with and who have used EFX, in particular Churchill Morgan and Jimmy Blyth of Crawford Post in Atlanta and Geoff Wheeler of Hawthorne Productions, for their invaluable design suggestions, wish lists, user testing, and priceless insights before and during EFX's design and implementation. We also thank the following for their work on the back-end server application and special-effects functions: Curt McDowell, Alan Norton, Bob Plotkin, Serge Stretchsinsky, Nidheesh Dubey, Rakesh Mohan, Gabriel Taubin, Rick Kjeldsen, Norm Haas, Ehud Karnin, Gul Ashour, Dov Ramm, and Ezra Silvera. Finally, we

thank Sarah Blanton and Mike Keeler for their help in testing EFX and for Mike's work in putting together the EFX user guide.

References

1. D.A. Norman, "Cognitive Engineering," in *User Centered System Design*, D.A. Norman and S.W. Draper, eds., Erlbaum, Hillsdale, N.J., 1986, pp. 31-61.
2. D.A. Epstein et al., "The IBM Power Visualization System: A Digital Post-Production Suite in a Box," *SMPTE (The Society of Motion Picture and Television Engineers) J.*, Vol. 104, No. 3, Mar. 1995, pp. 125-133.
3. E.L. Hutchins, J.D. Hollan, and D.A. Norman, "Direct Manipulation Interfaces," in *User Centered System Design*, D.A. Norman and S.W. Draper, eds., Erlbaum, Hillsdale, N.J., 1986, pp. 87-124.
4. B. Schneiderman, "Direct Manipulation: A Step Beyond Programming Languages," *Computer*, Vol. 16, No. 8, Aug. 1983, pp. 57-69.
5. B.A. Myers, "Taxonomies of Visual Progamming and Program Visualization," *J. Visual Languages and Computing*, Vol. 1, No. 1, Mar. 1990, pp. 97-123.
6. J.F. Koegel, J.L. Rutledge, and J. Heines, "Visual Programming Abstractions for Interactive Multimedia Presentation Authoring," *Proc. 1992 IEEE Workshop on Visual Languages*, CS Press, Los Alamitos, Calif., 1992, pp. 231-233.
7. T.D.C. Little, "Time-based Media Representation and Delivery," in *Multimedia Systems*, J.F. Koegel Buford, ed., ACM Press, New York, 1994, pp. 175-200.
8. J. Ankeney, "Non-linear Editing Comes of Age," *TV Technology*, Vol. 13, No. 6, May 1995, pp. 28, 50, and 52.
9. M. Rubin, *NonLinear: A Guide to Electronic Film and Video Editing*, 2nd ed., Triad Publishing, Gainesville, Fla., 1992.
10. E.A. Bier and M.C. Stone, "Snap-dragging," in *Siggraph 86 Conf. Proc.*, D.C. Evans and R.J. Athay, eds., ACM Press, New York, 1986, pp. 233-240.
11. S.R. Alpert, "Graceful Interaction with Graphical Constraints," *IEEE Computer Graphics and Applications*, Vol. 13, No. 2, Mar. 1993, pp. 82-91.
12. Adobe Systems, *PostScript Language Reference Manual*, 2nd ed., Addison-Wesley, Reading, Mass., 1990.

Sherman R. Alpert is in the Education Solutions group at IBM's Thomas J. Watson Research Center, designing a multimodal instructional environment for early readers and an environment for integrated math and science exploration. His current research interests include educational computing, multimodal environments, object-oriented programming, and user interface construction. Alpert received an MA in computing in education in 1987 from Columbia University Teachers College and a BS in computer science from the State University of New York at Stony Brook in 1974.

Mark R. Laff has been affiliated with the IBM Thomas J. Watson Research Center since 1961 (when he was six years old). He is currently a research staff member in the Niche Networking department. His research interests include human-computer interaction, usability engineering, computer graphics, and computer system design. Laff received a BS in mathematics from Ithaca College in 1977.

W. Randall Koons, previously an advisory graphics designer with IBM Entertainment Systems, is now an independent video producer. Self-educated in illustration, computer graphics, video production, and user interface design, he won an International Monitor Award for special effects and an IICS Mark of Excellence award for interactive multimedia. His interests include illustration, outdoor activities, and writing.

David A. Epstein is currently the technical consultant to the Vice President of Services, Applications, and Solutions in IBM's Research Division. He received his bachelor's and master's degrees from Harvard University in computer science and spent two years at New York University's Courant Institute of Mathematical Sciences before joining IBM Research. Since 1982 he has been involved in many facets of the graphics industry, from video games to commercial vehicle simulators, scientific visualization, video editing, special effects, and video compression.

Danny Soroker is a research staff member at IBM's Thomas J. Watson Research Center, where he is currently working on C++ programming environments. His research interests include parallel computing, multimedia, algorithms, and design and implementation of software systems. Soroker received a BS in computer engineering and an MS in electrical engineering from the Technion, Israel Institute of Technology, Haifa, in 1981 and 1983, respectively, and a PhD in computer science from the University of California, Berkeley, in 1987. He is a member of the ACM and the IEEE Computer Society.

David C. Morrill is a senior programmer in the Object Technology Products group of IBM in Austin, Texas. His interests include interactive programming languages and environments, GUI builders, 3D graphics and animation, and object-oriented programming systems. Morrill received a BS in mathematics from Pratt Institute in 1974.

Arthur J. Stein is responsible for Internet development for the Telecommunications and Media Division of IBM. He received his BS in physics and mathematics from Lowell University, Lowell, Mass. He has been involved in 2D and 3D computer graphics, image processing, scientific visualization, multimedia, and communications. He designed and managed the scientific visualization lab at Thomas J. Watson Research Center. He is a member of Siggraph, ACM, IEEE Computer Society, and NY Academy of Sciences.

Readers may contact Alpert at IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, e-mail alpert@watson.ibm.com.

Examples on the Web

The video showing the dancing New York skyline of Figure 6 can be viewed on the World Wide Web if you have an MPEG-capable video player. Point your Web browser to the IEEE Computer Society's Web site at http://www.computer.org and follow the menus through Publications to Magazines to *IEEE MultiMedia*'s Spring 1996 issue and this article.

This example demonstrates how EFX-supplied effects can be user-customized in unique ways. We achieved the bending of the skyline using EFX's *mirage* effect, which creates an undulating wave across a video image. The EFX Property Sheet lets users control the direction and size of the undulation as well as how these parameters vary over time.

To produce this video, we reversed the direction at each downbeat of an accompanying music clip (see the main text on p. 22). Thus the "dancing" is synchronized to the beat of the music.

The video was MPEG-compressed using IBM's Digital Compression Facility running on a Power Visualization System.