Video Bench

Final Report

SEng 480a / CSc 586a

James Chisan
Jeff Cockburn
Reid Garner
Azarin Jazayeri
Piotr Kaminski
Jesse Wesson

University of Victoria
April 11, 2003

Table of Contents

List of Figures

1. Introduction

The Video Bench project aims to explore the design space of collaborative video editing through direct gestural manipulation. To achieve this goal, we designed and implemented a prototype video editing system that takes advantage of the features of the DiamondTouch device.

1.1. Target Audience

The target audience for our user interface (and thus our prototype) is the casual user who requires only basic editing functionality and prizes ease of use and convenience over fancy features. Furthermore, we expect our users to do the editing collaboratively, in small groups of 2 to 4 people.

Sample users include parents editing home video footage (perhaps with the assistance of children), researchers editing user study records, and perhaps even film professionals collaboratively roughing out the structure of a scene or reviewing dailies. The specific operations that we allow our users to perform are further detailed in Chapter 3.

1.2. Scope Restrictions

The prototype's only purpose was to allow us to experiment with and evaluate our proposed video manipulation techniques, so many traditional concerns were outside the scope of this project. The system's performance was important only insofar as it qualitatively affected the nature of the users' interactions with the system. While some degree of responsiveness is necessary to maintain the illusion of direct manipulation, the system did not need to scale to higher workloads. In the same vein, the system did not need to be able to handle high-resolution or lossless video formats; when testing, we used low-resolution videos that could be decoded efficiently.

In consideration of the system's target audience, we chose not to support advanced video editing functionality such as separate audio tracks, special effects, or colour adjustment. While undoubtedly useful, such advanced functions are not necessary to perform basic video editing. It will be interesting to investigate whether these more abstract functions are amenable to direct manipulation once we've proven that our basic approach is sound (see Section 6.2).

1.3. Outline

The rest of this report is structured as follows. Chapter 2 provides background information on video editing systems and related research topics. Chapter 3 introduces the fundamental concepts of the Video Bench and illustrates the primitive operations made available to the users. Chapter 4 discusses implementation issues, and Chapter 5 describes the results of an informal system evaluation. Chapter 6 concludes with a list of achievements and ideas for future work.

2. Background

Film and video editing is hardly a new art. Without delving overmuch into history, this section describes the operation of a handful of industrial video editing systems, and presents some of the research in this area and other relevant fields.

2.1. Commercial Video Editing Systems

Nonlinear video editing systems have been available for some time. Originally consisting of turnkey solutions, newer systems can be run on a variety of workstations. With the emergence of digital camcorders, consumer applications have also become available.

One of the major players in the professional space is Avid and their software package Media Composer. It is able to handle multiple media sources from traditional film to DV and output to a variety of formats including NTSC, High Definition TV, and others. Apple has also recently introduced two applications into this space, Final Cut Pro and iMovie. Final Cut Pro is a professional system in the vein of Avid's Composer but does not rely on proprietary hardware for support. iMovie (see Figure 1), on the other hand, is geared towards the home user. This application only supports DV video and does not offer all the features of the first two but is still a powerful tool for creating rich video.

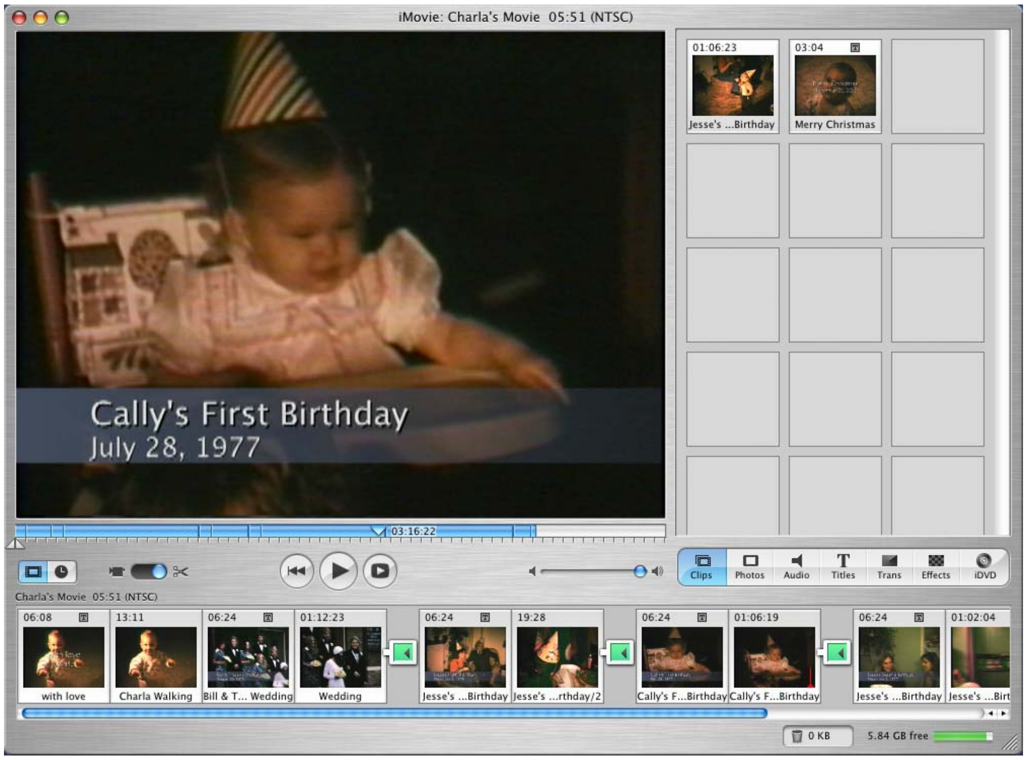


Figure 1. A typical iMovie window

All of these packages rely on the same metaphors and have similar user interfaces. There are two main objects represented, the film being created and the clips that are used to create it. These clips are usual presented in a bin of some sort allowing for easy retrieval and subsequent inclusion in the film. A preview window is supplied to review the clips and film, while the higher end packages provide multiple windows and allow for real-time screening of changes. Finally, the editing of the film uses a timeline (see Figure 2) that provides access to the individual frames. This is where the fine-grained editing occurs; transitions can be inserted and effects applied to the specific segments of the film. The timeline can also be zoomed, but doing so normally loses the context.

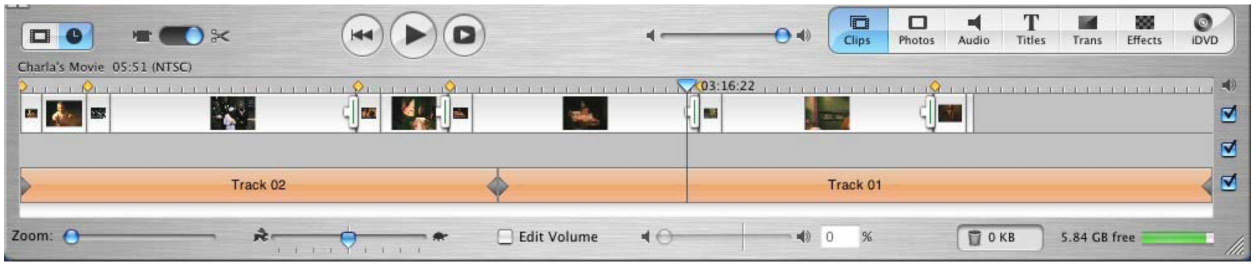


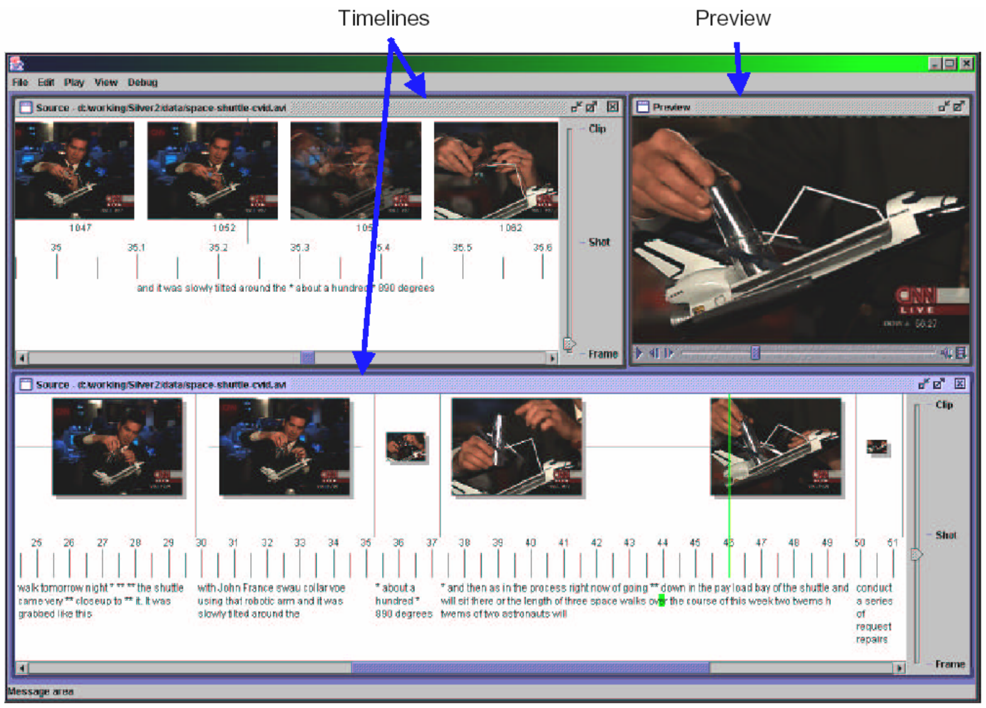
Figure 2. A detailed timeline in iMovie

Very few commercial systems support direct manipulation or gestural editing. Quantel, a provider of professional editing stations, provides some support for gestural editing in their top-of-the-line systems. The workstations are equipped with a graphics tablet, and the user can use the pen to control some features of the application—details are hard to come by. However, this cannot really be considered direct manipulation, since the user interface (apparently) still employs the traditional timeline, and the input tablet is separate from the display.

The Wacom Cintiq tablet and the recently introduced Tablet PCs address this last objection by combining the screen with the input surface. The Cintiq literature makes allusions to film editing, but seems to recommend the use of Final Cut Pro. It is unclear what advantage, if any, is derived from having direct input in a user interface designed to be operated with a mouse.

2.2. Video Editing Research

There seems to have been relatively little academic research into video editing systems. At the moment, the most prominent effort is the Silver project (see Figure 3). [LMC+]


Timelines
Preview
File Edit Play View Debug
Source - d:working/Silver2/data/space-shuttle-cvid.avi
Preview
Clip
Shot
Frame
Message area

and it was slowly tilted around the * about a hundred* 890 degrees

walk tomorrow night * ** ** the shuttle came very ** closeup to ** it. It was grabbed like this

with John France swau collar voe using that robotic arm and it was slowly tilted around the

* about a hundred * 890 degrees

* and then as in the process right now of going ** down in the pay load bay of the shuttle and will sit there or the length of three space walks over the course of this week two twems h twems of two astronauts will

conduct a series of request repairs

Figure 3. Main user interface of Silver 2

The main thrust of the project seems to be to take the traditional timeline paradigm and augment it with extra information and context. The application principally use lenses and semantic zooming (focus + context) to achieve this goal (see Figure 4), and if available also displays a transcript along the timeline. While these techniques may make the visualization more effective (the results seem inconclusive), the project does not appear to innovate as far as the actual editing operations are concerned.

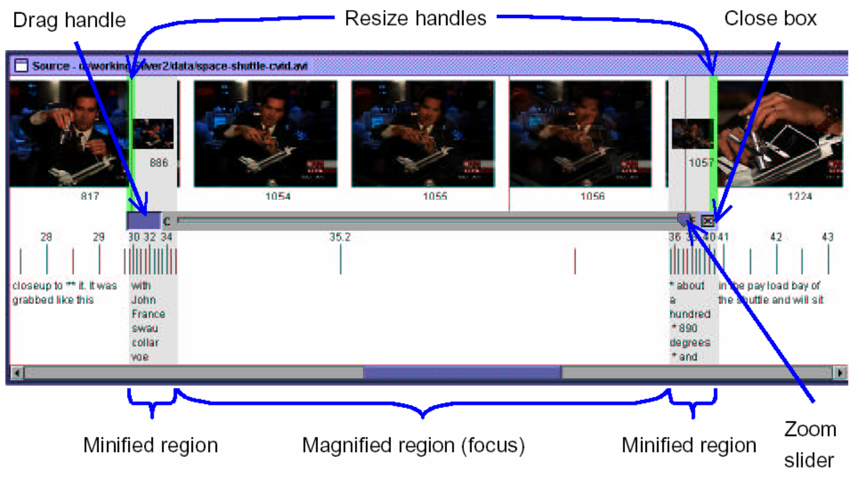


Figure 4. Silver 2 lens

Some older research has investigated the issue of video editing through direct manipulation more closely. Most projects seem to focus on the idea that, due to the large amount of data contained in videos, direct manipulation is impractical without system assistance. IMPACT [UMY91] aims to automate the process of separating clips from a video through automatic scene and key frame detection. It also uses object motion analysis to provide an abstract visual "description" of a cut. The resulting, hopefully meaningful segments can then be composed by the user through drag and drop operations. Hitchcock [GBC+00] takes this approach to the extreme, by having the system determine which cuts are good and how they should be composed given only a target video duration. The user can affect the decision process, but does not perform any video editing in the traditional sense. VideoScheme [MGM93] takes a programmer's approach to resolving the automation issue, by integrating the video editor with a Scheme scripting system, but this approach is irrelevant considering our target audience.

Finally, the video mosaic system [MP94] is an augmented reality system that associates video clips with a hand-drawn storyboard. The storyboard is drawn on a piece of paper and lies on a desk, while video is displayed either besides it or directly on top of it. Manipulation seems to be applied exclusively to the storyboard; the video display is just for viewing.

In summary, while most aspects of our proposed Video Bench system have been experimented with before, nobody has yet attempted to bring them all together in the fashion described below.

2.3. Gesture Research

In the mid 1980's there was some research that looked at the technical and practical issues of direct manipulation input. [KB91][Bux92] In particular, much of the work we have identified so far concentrates on recognition of simple gestures made by a single point input device (such as a mouse or pen). However, some research has sought to explore multipoint input on a horizontal surface. In fact, Lee [LBS85] describes a device that is more sophisticated than the DT in that it has a complete matrix of point sensors across the surface of the board, rather than the DT's cruder horizontal and vertical sensor arrays (see Section 4.2.1). The complete matrix allows the system to accurately determine where the user is interacting with the board, even with multiple contact points. Presumably, the DT made design tradeoffs in order to keep production costs reasonable, and it does offer the added feature of being able to distinguish between multiple users.

There is also some more recent research that concerns itself with the problem of handwriting recognition [Rub91]. Parts of this work related to stroke classification may be relevant for some of the gestures we are planning to support. For example, the circular jog gesture must be recognized as such based on its shape, not just on the objects it happens to intersect. In general, pattern classification is a broad and ongoing area of research. Current research efforts seem aimed towards 3D gesture recognition of hands, bodies or other moving objects, either via image analysis or with the use of a sensor glove.

2.4. Scene Segmentation Research

In traditional cinematography, a scene is the basic building block used to compose and convey ideas. The ability to detect scenes allows a system to operate at a coarser, more semantically meaningful level than frame-by-frame manipulation. There are several scene boundary extraction techniques currently in use. Clustering techniques use shot homogeneity to cluster similar scenes with respect to visual similarity and time locality. Segmentation, on the other hand, looks at differences between shots to determine scene boundaries. If shots are found to be different based on visual content and time locality they are grouped into different scenes. [WCC01] Both of these techniques are based on grouping shots with similar visual properties together.

The cinematic model for scene detection attempts to give a higher-level semantic meaning to extracted scenes. A typical scene is a series of locations or dramatically related shots. Many films are non-linear, which spreads related scenes throughout the timeline. The cinematic model approach brings the related scenes together providing useful semantic content. This approach starts off by using either the segmentation or clustering method to detect scene boundaries, but then takes the extra step of linking related scenes together based on visual correlation.

One of the problems that come up in pixel-based visual pattern detection is finding false boundaries. [BM99] Fast camera panning and zooming are common techniques, and both can result in false boundary detection due to the rapid change in pixel information. Some methods that try to address these problems include histogram analysis and temporal window pixel analysis. The histogram approach looks at the colour or intensity statistics to determine scene boundaries. The temporal window technique uses several separate windows within a shot and determines the relative changes within the windows. This technique addresses issues such as zooming and panning quite well.

3. Conceptual Design

Before the advent of digital non-linear video editing, the only way to edit moving pictures was to get down-and-dirty with the rolls of film. The editor had to handle the footage manually, cutting strips with a knife, hanging them on racks for later use (see upper-left corner of Figure 5), before splicing them together with clear tape. The process was painful and error-prone, but it had a certain physical *thereness* that has been lost in the abstract modern systems. The Video Bench tries to recapture the feeling of hands-on manipulation while keeping all the advantages of non-linear editing.

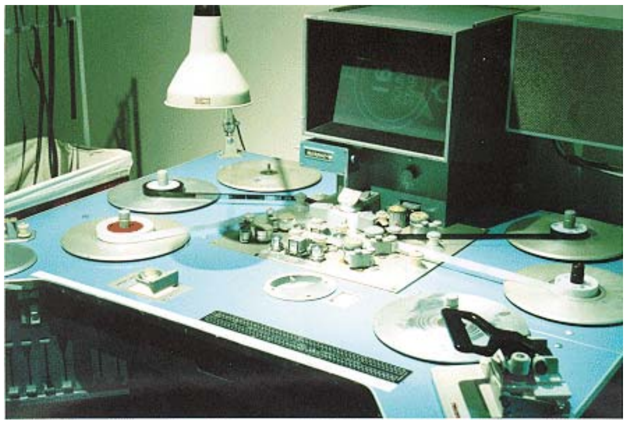


Figure 5. A Steenbeck six-plate flatbed editor

This chapter explores our user interface design for the prototype, by annotating screenshots and explaining the primitive operations that can be performed by the user. All operations can be controlled using either a mouse or fingers, and both modes of operation are presented in each diagram for easier comparison. The few details that have changed since the proposal are noted in the text.

3.1. Benchtop

Figure 6 shows a view of a typical benchtop. The primary metaphor is strips of film scattered on a tabletop, showing key frames of their footage. Notice the lack of any handles or widgets: a gestural interface should make those unnecessary, and they would just clutter the desktop and detract from the metaphor.

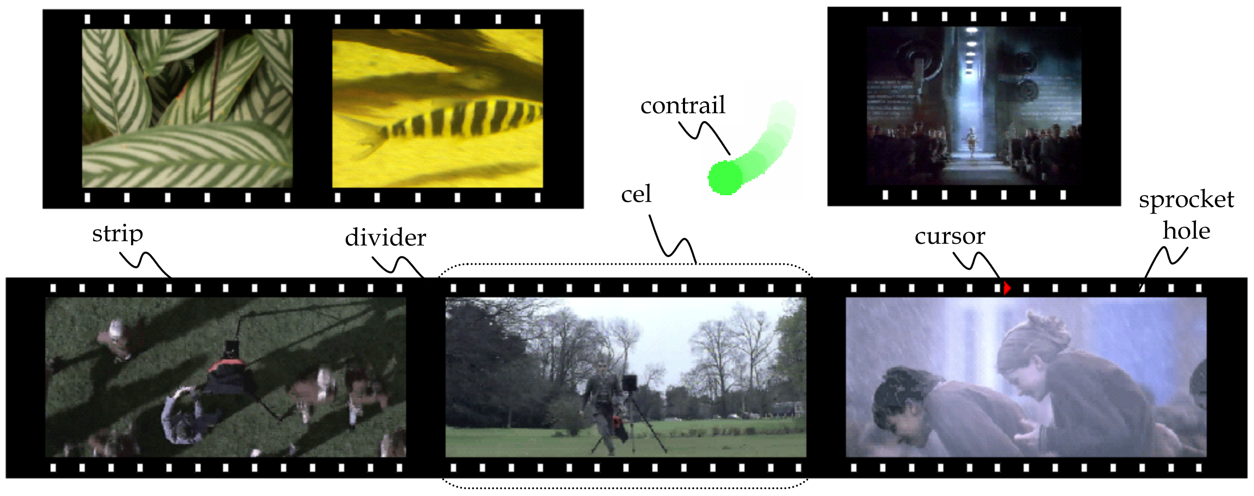


Figure 6. Benchtop with legend

This is also a good time to define some terms that we'll need to use when describing the operations allowed by the bench. Each file that a user imports is called a *clip*, and it is composed of atomic *frames*. Neither of these is directly represented in the user interface, but the concepts are needed for precise explanations of some of the operations.

The visible elements are identified in Figure 6. Logically, a *strip* represents a sequence of frames. Visually, a strip is a sequence of *cels*[1] bordered by *sprocket holes*. Each cel holds a part of the sequence of frames of the parent strip, with the density of the sprocket holes indicating the quantity. Normally, each cel displays the key frame [GBW01] of its sequence, but the user's operations can affect this. If another frame is being displayed, a *cursor* ranging over the width of the cel indicates the frame's position within the cel's frame sequence.

The position of pointing devices (whether mouse or fingers) can optionally be tracked with *contrails*. The head of the contrail follows the movement of the pointer (leaving a fading trail behind), providing direct and obvious feedback on the application's idea of the position of the pointers. This could be particularly useful when the detection algorithms are not completely reliable, as is the case for the Diamond Touch (see Section

[1] These would normally be called frames, but this would have been easily confused with the actual frames that make up a clip. Also, in real filmstrips, frames are laid out vertically. We have chosen to use the horizontal orientation instead so that some of the most important operations that require two hands could be performed more easily, as dictated by human physiology.

4.2.2). Control is further enhanced by highlighting the relevant visual elements before (and while) an action takes place, so the user is never in doubt about the operation they are about to trigger. Visual continuity is also improved by animating all transitions: visual elements move smoothly between locations, and fade in (out) when appearing (disappearing).

3.2. Play Toggle

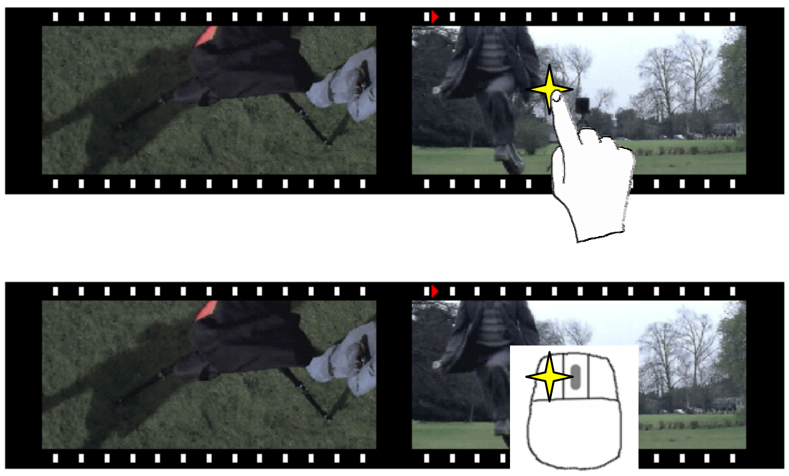


Figure 7. Toggling a cel

To edit videos, one must first be able to watch them: the most basic operation is to play back a sequence of frames. The user can start and stop playback by tapping or left-clicking inside a cel, as shown in Figure 7. Toggling an inert cel starts playback from its first frame,[2] within the cel itself, animating the cursor to match the video's progress. Toggling a playing cel pauses playback, and toggling it again resumes playback from the current frame. Double-tapping (or double left clicking) a cel resets it to the key frame.

Normally, playback is isolated to individual cels. However, if a playing cel reaches its last frame and the next cel in the strip is showing its key frame, playback automatically transfers to the next cel, while the original cel resets itself (after a short pause to avoid visual discontinuity). This mechanism integrates a projection of time onto the horizontal axis (the strip of cels) with its original dimension (frames played within a cel), and allows a whole strip to be played by toggling its first cel, no matter how many cels it contains.

[2] Since the key frame showing in an inert cel is not usually the first frame of the segment beneath it, this causes a visual discontinuity and may be confusing, but captures the most common usage. An alternative would be to forgo key frames altogether and always play from the frame currently showing.

3.3. Spread and Fold

While a strip on the bench is meant to evoke the idea of a real filmstrip, it cannot be a one-to-one representation: a real filmstrip has far too many frames to fit on a typical display. Thus, the strip is an abstraction where each cel stands for a number of frames in the video. This is a kind of semantic zoom that can be used to provide context + detail. Figure 8 and Figure 9 show the spread and fold operations that allow the user to zoom in and out, respectively.

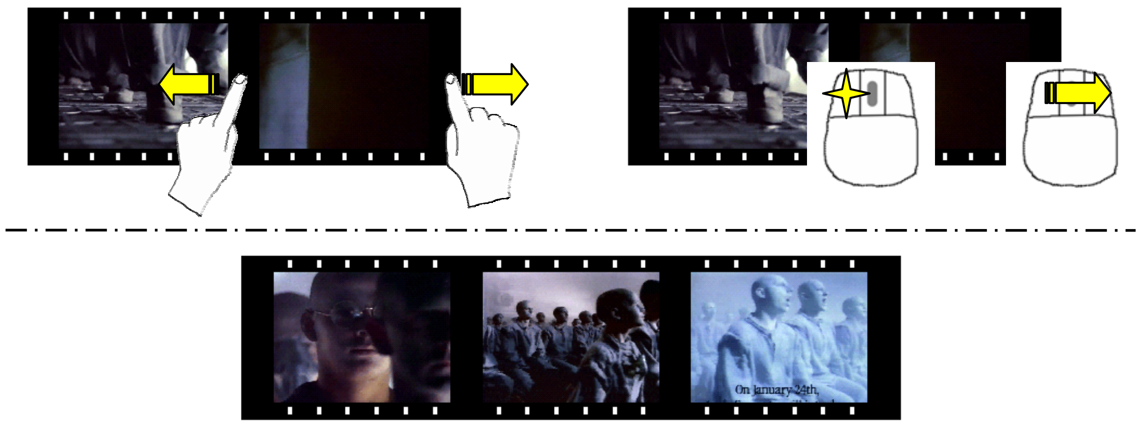


Figure 8. Spreading a strip

To spread by touch, the user positions her fingers between cels and moves them apart. The span selected by the fingers stretches, and additional cels are inserted to fill in the extra space; cells outside the selected span are not affected. To spread using a mouse, the user first left-clicks to select the divider at one end of the desired span, then left-drags the divider at the other end of the span to create more space. Note that the number of frames within the span remains constant, so the density of cels must go down, as reflected by the sprocket holes. At the end of the operation, the strip automatically collapses to the smallest size that can hold the desired number of frames, discarding any unused space created during the spreading process.

The only remaining question, then, is how to repartition the span's frames between the newly increased number of cels. The easiest option is simply to divide the frames evenly, so each cel within the span has the same density. However, this is unlikely to lead to a satisfying user experience, since the key frames displayed in the cels may not be representative of the cels' contents, and the cels' dividers (which are the easiest points at which to split strips, see Section 3.5) would not be conveniently located. For these reasons, an algorithm for detecting key frames and segmenting clips into scenes may well be the key to making the Video Bench usable (see Section 2.4 for details).

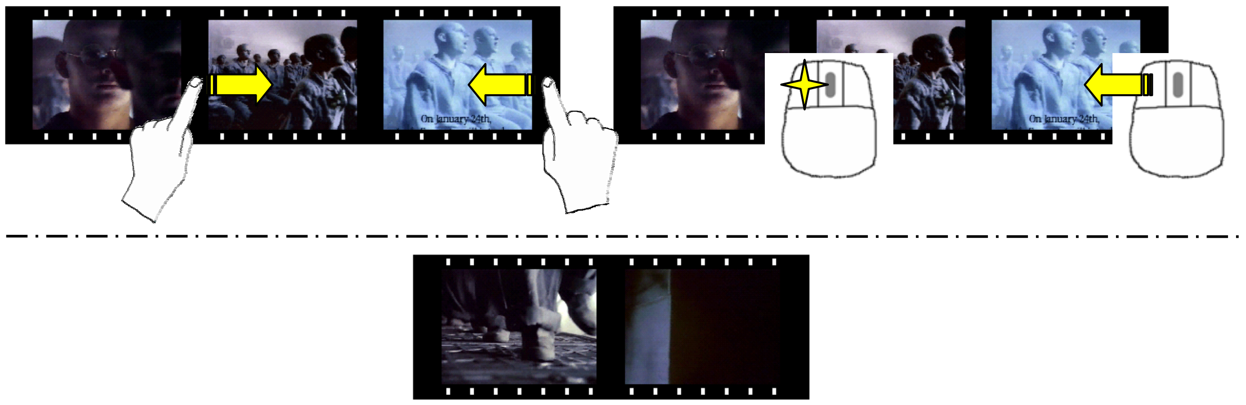


Figure 9. Folding a strip

Folding a strip is the reverse of spreading it. The user selects a span of cels and slides the dividers together, shortening it. As the space disappears, cels are removed and frames rebalanced between the remaining ones. Once again, cels outside the selected span are not affected.

3.4. Move

To maintain the illusion of a desktop with bits of film scattered on it, the user must be able to shuffle the strips around the surface. This common operation is executed by "grabbing" a strip with multiple fingers (or with the middle mouse button) and dragging it to another location, then releasing it (as shown in Figure 10). Strips can overlap, but grabbing a strip immediately moves it into the foreground.

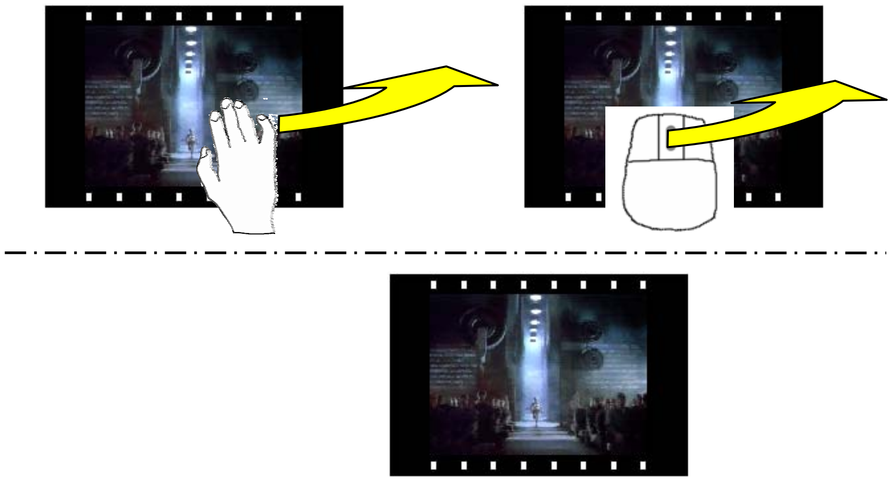


Figure 10. Moving a strip

We were originally planning to let the user rotate the strip as she moves it. In a collaborative setting, it's unlikely that all the participants will be seated on the same side of the table. If they were working semi-independently, they would probably like "their" strips to be right side up. Even if there is only one user, it is unnatural to keep everything perfectly aligned. Indeed, it seems likely that the orientation of an object can play an important part in organization and recognition processes. Unfortunately, due to implementation issues (see sections 4.1.1 and 4.2.1), we had to drop this feature from our prototype.

3.5. Split, Splice and Jog

We now come to the heart of any video editing system: the ability to affect the order and presence of frames. The Video Bench fulfills this requirement by letting the user split (Figure 11) and splice (Figure 12) strips. Splitting a strip between two cels divides it into two strips, each containing the frames on either side of the splitting point. The gesture used to split is a slicing movement across a divider, or a right-click with the mouse.

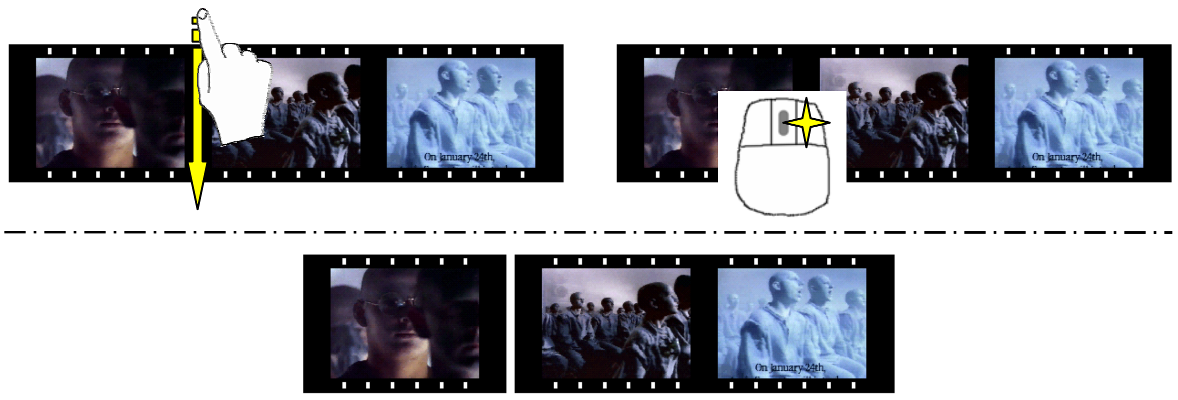


Figure 11. Splitting a strip

Splicing is the natural opposite, combining two strips into one. In the original proposal, the gesture for splicing required both candidate strips to be grabbed. Having control of both objects would have made it easier to match them up, and prevented the accidental splicing caused by mistakenly dropping a strip close to another one. However, having two loci of control is impossible with a mouse, and turned out to be very difficult to achieve with the Diamond Touch as well, so we dropped the idea. Instead, dropping a strip onto another one with matching dividers overlapping will splice them together; we try to prevent accidents by giving a clear visual cue of the imminent action.

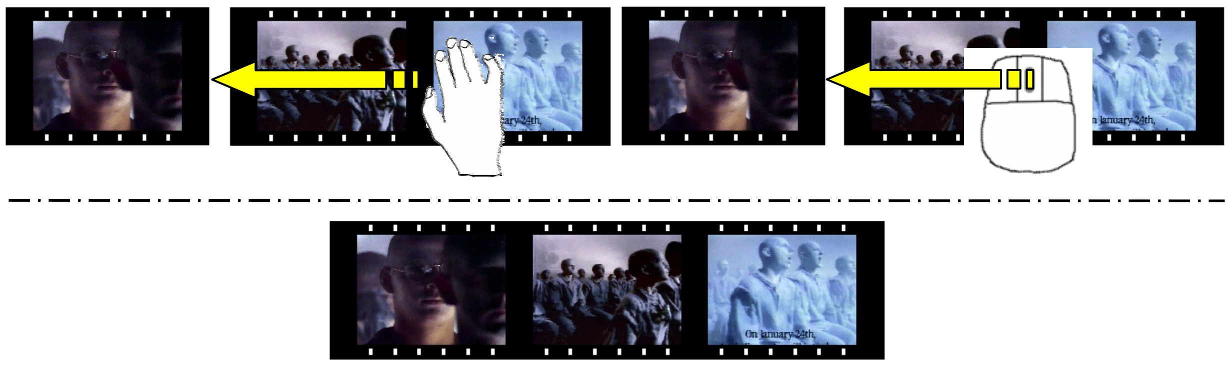


Figure 12. Splicing two strips

Combined with spreading and folding, these operations are sufficient to perform all basic editing actions. The user can always keep spreading a strip until the right division point is picked between cels, then split the strip at that point. This is hardly practical, though, especially since in the worst case the user might need to spread the strip down to frame level. By that time, the strip would likely be hundreds of frames long, and far exceed the display's boundaries. Though we can hope that the automated scene detection will be adequate some of the time, another method is needed.

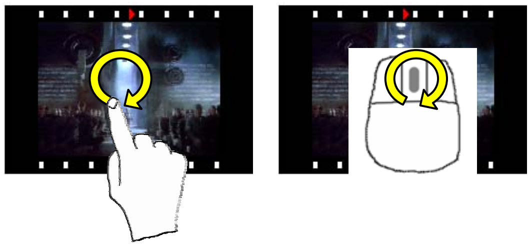


Figure 13. Jogging through a cel

To give the user more control over split points, we introduce the "jogging" operation (Figure 13) so named after the jog dial common to video equipment. By running her finger clockwise and counter clockwise in a circular motion[3], or by rolling the mouse wheel, the user can control the frame displayed in the cel. The bigger the circle made by her finger, the faster the frames advance, making the gesture practical no matter how dense the cel is; there is no equivalent step size control for the mouse. The cursor provides continuous feedback on the current frame's location within the cut.

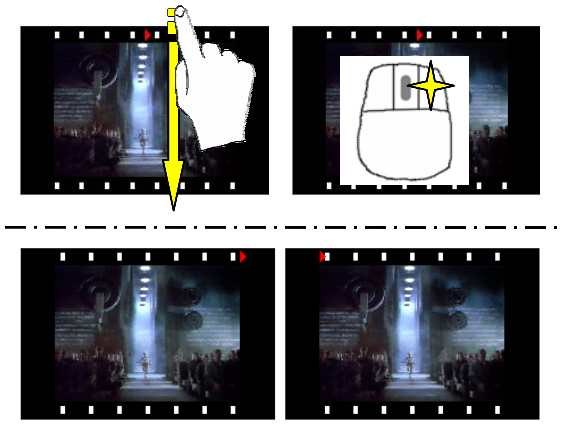


Figure 14. Splitting a strip mid-cel

Once a cel is selected within a frame, either through jogging or simply by pausing playback at the right moment, the user can split the strip mid-cel (Figure 14) by pulling her finger *through* the cel rather than between them, or by right-clicking on the cel itself. This results in two strips, with the one on the right starting with the cel's selected frame, while all frames before the selected one go into the strip on the left. The newly split cels are automatically cued to their first and last frame, respectively, to clearly show the point at which the split was made.

Notice that a mid-cel split combines a split and a spread operation; the two cells obtained from the one that was split will have lower densities. We can take advantage of

[3] Though the initial movement must be made within the cel, further movements can take place outside, so that the projection of the cel's contents isn't blocked by the user's hand.

this by remembering the split's location and, even should the strips be recombined, preferring the user's choice of segmentation to any automatically computed ones.

3.6. Grow and Shrink

The final operation envisioned would allow the user to control the size of a cel. The grow (Figure 15) and shrink (Figure 16) operations scale the cel on the horizontal and vertical axes while maintaining the correct aspect ratio. When the gesture is complete, the cel is automatically recentred within the strip to prevent excessive deformation. The operations are analogous to spread and fold, but affect the physical dimensions of the cel rather than the sequence of frames behind it. They are performed in a similar manner, by either grasping the top and bottom edge with fingers, or by left-dragging an edge with the mouse[4]. The user is prevented from shrinking a cel below a minimum size, calculated from the centreline of the strip. This makes it impossible to shrink a cel to minimum size using the mouse, since it only allows control of one edge at a time.

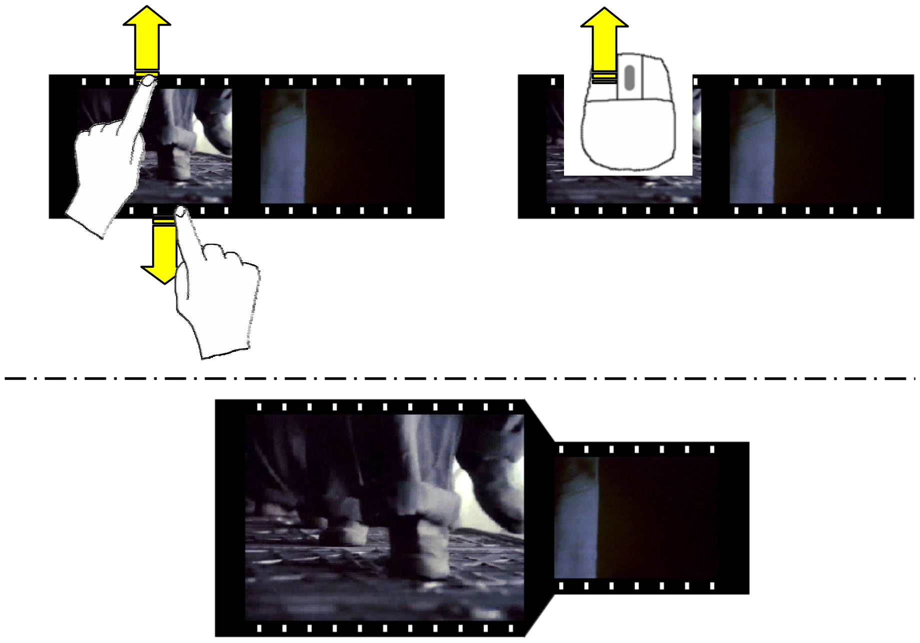


Figure 15. Growing a cel

Shrink/grow is a kind of zoom that provides focus + context, since the surrounding cels are pushed away (or brought in closer), but do not change size themselves. The frame

[4] Unlike spreading, it is unnecessary to first select the other boundary, since edges are strictly paired.

density of the zoomed cel doesn't change either, which is represented by keeping the sprocket holes' density (i.e. spacing) constant.

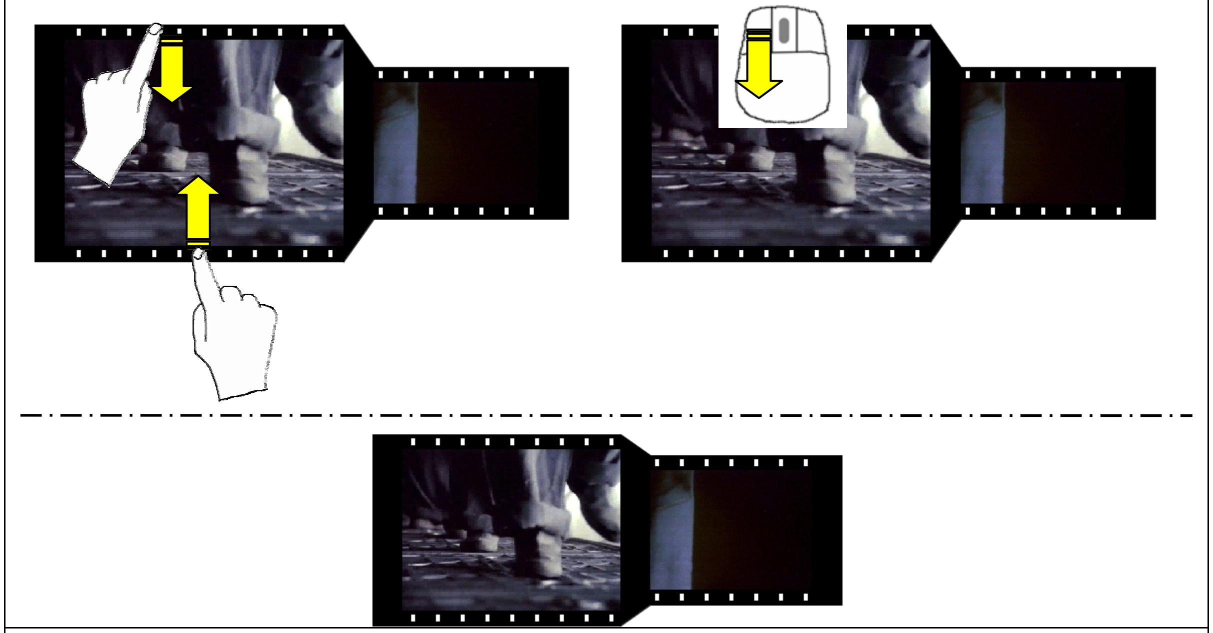


Figure 16. Shrinking a cel

3.7. Tools: Trash, Clone and Squish

It is likely that, as the user edits a video, they will end up with scraps of footage that are no longer needed and just clutter up the bench: we need to provide a trashcan to get rid of them. Conversely, when experimenting with different cuts, one copy of a strip may not be enough—the user might want to use a shot more than once, or hang on to a previous version of a strip for safe-keeping, in case the new edits don't work out. The easiest way to cater to these requirements is to let the user quickly clone any strip. Finally, when growing and shrinking cels, it would be convenient to be able to quickly return a whole strip to its minimum size.

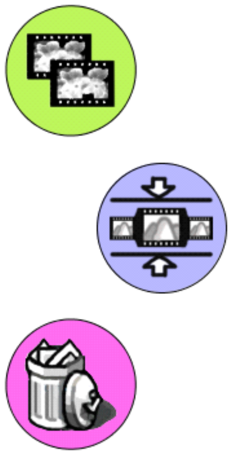


Figure 17. Tool palette

All of these operations can be modeled as a "splice" with a special target, brought up by a grasp (or middle-click) on an empty area of the bench. Using a common gesture to execute extended operations lowers the user's conceptual overhead, and having the special targets pop up on demand is both convenient and keeps the bench uncluttered. The design is also extensible, since more targets can be added to support other parameterless single-strip operations. The various targets are differentiated through the use of different colours and icons (Figure 17).

4. Implementation

This chapter provides details of the design's implementation, explains the challenges faced, and documents the current prototype's limitations. We chose Java as our implementation environment, since it's a solid general-purpose language with extensive libraries that can be used for rapid prototyping, and (most importantly) well known by all the authors. We standardized on JDK 1.4, in order to take advantage of some recently introduced features and its generally improved performance. Many (though not all) developers used the Eclipse IDE, backed by a common CVS repository. Setting this up proved surprisingly troublesome, but once in place the infrastructure was very reliable.

As a point of interest, the application is cross-platform[5] and has been tested on both Windows XP and Mac OS X.

4.1. Rendering

The core component of the Video Bench is its display, the canvas. The canvas displays all the strips and their key frames, allows them to be layered and translated, and to play video within the cels. There are two distinct but closely related subsystems: canvas drawing and video manipulation.

4.1.1. Canvas Drawing

We used the 1.02-beta version of Piccolo as our drawing framework. Piccolo handles hierarchical nested coordinate systems that can be arbitrarily transformed, and is

[5] With the exception of the Diamond Touch device interface, which requires native access to a machine's serial ports. This driver can probably be converted to pure Java once the low-level communication libraries are added to the JDK.

hooked into the Swing event loop for input and animation. Being fairly small, Piccolo was easy to learn, but this simplicity was also its downfall. It made many things possible without making them easy. Since it's a beta version, it also has its share of bugs that we needed to fix. Piccolo could benefit from further refinement and the addition of more useful "extras".

Drawing contrails proved particularly difficult to accomplish. Each contrail is a series of straight-line strokes painted with an alpha gradient.[6] However, to overlap properly, the segments first need to be drawn using the Porter-Duff Source alpha compositing rule, then composed on top of the canvas using the Source Over rule to preserve the transparency. Since Piccolo doesn't support rendering part of the node hierarchy to a side buffer, we needed to completely override the rendering logic for the contrail layer and manage our own buffer allocation and blitting.

We also encountered many problems with rotation (some caused by bugs, others probably by our lack of understanding of the framework), and quickly gave up on rotating strips. Rotation would have increased the complexity of all inter-strip layout algorithms and many of the transformation animations. It turned out later that the gesture detection engine couldn't deal with arbitrary rotation anyway (see Section 4.2.1). We still hope to reintroduce rotation in a later iteration: what little of it worked looked very promising.

4.1.2. Video Framework

The other piece of functionality needed is control over video playback. We investigated the two leading media frameworks, QuickTime for Java and the Java Media Framework (JMF), and decided to use the latter. The current implementation of QuickTime for Java is not compatible with JDK 1.4, and in any case it's only a thin wrapper around the C library that lacks the flexibility we require. JMF, on the other hand, exposes its rendering path in an object-oriented manner and has a history of successful use for video editing in the Silver 2 project.

---

[6] Java does not provide facilities for tapered strokes, or for smoothed curves filled with a gradient.

We hooked up JMF to Piccolo by writing a custom JMF video renderer that writes each frame into a buffered image supplied by the canvas subsystem. Whenever possible, the renderer requests that the codec deliver data in a format suitable for direct blitting into video memory, thus optimizing performance. We use a triple-buffering system to reconcile the video and canvas rendering threads. At all times, one buffer is owned by the canvas to refresh the screen, one buffer holds the next fully rendered frame, and one buffer is used for writing the next frame. The video renderer asynchronously notifies the canvas whenever a new frame is ready, and the canvas picks it up as part of its normal screen refresh loop. While this arrangement requires extra memory for the buffers, it is very robust: if the canvas refresh falls behind, it will simply skip frames and thus remain in sync with the other tracks of the video that are being rendered independently (e.g. sound).

Since JMF does not offer any abstractions of granularity finer than an entire piece of media (clip), we also needed to write custom decorators that would emulate the finer structure of segments and composites, enabling editing. We achieved this by manually controlling the start and stop times of the underlying player, and by manually starting the next player in a composite whenever the previous one reaches the end of its segment. While expedient, this technique sometimes causes stuttering over splice points, depending on the underlying platform's performance.

4.2. Gesture Recognition

Gesture recognition is a substantial component of our system as we endeavour to leverage the unique abilities of the DiamondTouch device (DT). Through natural direct manipulation of video artefacts displayed on the table, we anticipate users will be able to conduct most operations by making simple physical gestures on the table.

4.2.1. Multipoint Detection

Unfortunately, due to its construction the DT provides rather rudimentary sensor data. It uses a set of horizontal and vertical sensors that span the entire width and length (respectively) of the board; these sensors can detect if there is a touch somewhere along their span. This is enough information when only one point is being touched at a time (by each user), but is insufficient to accurately indicate positions when two or more points are being touched at the same time. There is inherent ambiguity since only the "projection" of the contact on the board is detectable (see Figure 18).

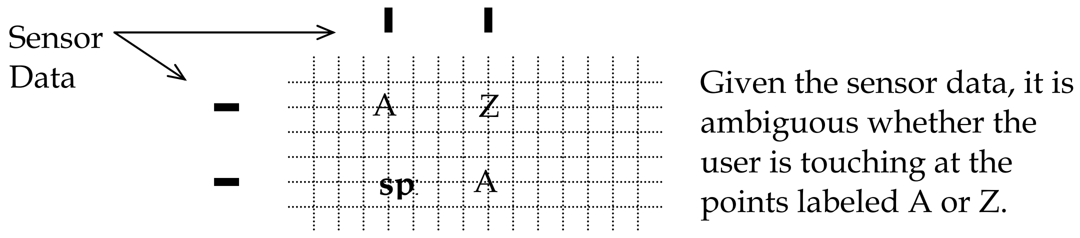

Sensor Data
A
Z
sp
A
Given the sensor data, it is ambiguous whether the user is touching at the points labeled A or Z.

Figure 18. DiamondTouch sensor data ambiguity

Multipoint gestures thus presented considerable challenge due to the rather crude data available from the DT. Clearly, the position ambiguity must be resolved in order to respond appropriately to users' direct manipulations. Fortunately, our system is limited to multipoint gestures that rely on only two accurate points of input; it probably would have been impossible to detect more than two points accurately and consistently.

The disambiguation algorithm is based on a simple concept that exploits a user's natural tendency to put down one finger (first point) before choosing the second point. By tracking the initial point we have a basis for comparison as subsequent data becomes available. If we imagine the ambiguous reading as a four-corner square, the algorithm essentially chooses which two new corners are closest to the previously detected points. Unfortunately, this algorithm is not foolproof since the choice of corners is ambiguous whenever the user "crosses the axis" (Figure 19). For the multipoint gestures that we support this is not a concern since, in the situation below, we are primarily concerned with movement along the X axis, which both possibilities capture accurately. However, this simplification no longer holds if the axis of interest is not aligned with either the X or Y axis, thus precluding arbitrary strip rotation.

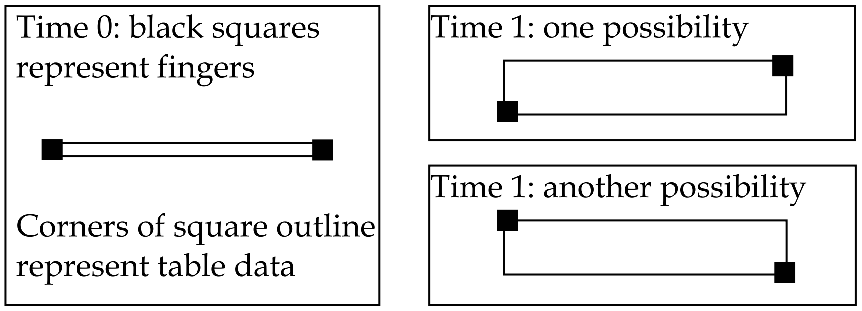

Time 0: black squares represent fingers
Corners of square outline represent table data
Time 1: one possibility
Time 1: another possibility

Figure 19. Axis crossover ambiguity

4.2.2. Hardware Issues

Ultimately, the DT's design philosophy is to blame for the difficulties we encountered. The table's primary feature is the multi-user aspect, so its multipoint capabilities are somewhat limited by design. While multi-user interactions are a unique feature, multi-point interaction is very natural once you sit down at the table, suggesting that perhaps more attention could be paid to improve the multi-touch capabilities of the board.

Drivers for the hardware could also be improved. While the driver interface is expectedly low level it could be improved by providing consistent "data-transformations" on all points of data. For instance, the driver only provides interpolation on a single data point rather than the four-corner data points that it also delivers in every data frame. In our implementation we had to implement this manually in our own work instead of relying on the driver.

In addition, some data from the board seems to be slightly erroneous in some cases. For instance, when users put down a second finger, the first data point sent seems to be very inaccurate, an issue we resolved by throwing it away when the situation is detected. In addition, in practice, users occasionally momentarily lose contact with the board. To address this we only considered users to have stopped touching the board after a prolonged period of no contact, about 3 data frames[7] depending on the operation.

[7] Three data frames equates to approximately a fifth of a second at the 15Hz data rate of the board.

4.2.3. Jog and Split Gestures

Some of the gestures used in the application were first developed for use with a mouse. This was for two reasons: first, access to the DT was limited, and second, to determine if it was possible to detect the gestures at all. Implementation of gestures with a mouse turned out to be a relatively easy task. The accuracy of the mouse made gestures such as jogging easy to detect in a small area, giving hope for eventual touch gesture detection. A down side to this approach was that a mouse only provides one input channel, whereas we wanted to make gestures that used both hands or that used multiple fingers. However, we were able to develop the single-finger split and jog gestures using the mouse as input.

When it came time to port the mouse gestures to the DT we ran into some complications. The first gesture that we ported was jogging, and even though the code was developed so that it could be easily adapted, it turned out to need fine-tuning. It seemed that the DT would stop providing input data either because the user was not making a strong enough contact or because the user was moving too fast, whereas with a mouse you are guaranteed a constant and consistent data stream. We had to modify our gesture to be fault tolerant and to accept occasional bad data.

The jog gesture does not actually detect a circle, but rather a square-like shape. It watches the motion of the pointer and figures out the direction it is traveling. Once the direction changes it determines if it is now going in a valid direction, i.e. if it was originally traveling up it should now be going left or right. At this point we can determine the user's intended direction of rotation. The recognizer continues to watch the data stream and ensures that subsequent movements are in the right direction. Once the user is doing a forward jog, they cannot reverse directions unless the lift their finger and start a new jog going backwards. The reason for this is that if you allow for mid jog changes it makes it hard to detect if the user wanted to jog or is just drawing lines from left to right. This limitation proved unpopular with users (see Section 5.4).

Armed with experience, the split gesture was a lot easier to port. The split gesture is simpler, only needing to determine if the direction of travel is vertical. All recognizers also rely on information from the canvas to ensure that the detected gesture is applicable to the objects it traverses. For example, the split gesture requires that the pointer start on an empty piece of canvas, travel in an upwards or downwards direction, pass through a divider or a cel, finally stopping on an empty area.

4.2.4. Other Gestures

Move. A move operation is triggered any time a bounding box that is more than 100 pixels wide covers a strip. In practice, this condition is triggered whenever the user puts her hand over a strip as if to grasp it. Once the operation begins, the restriction on the width of the bounding box is lifted, to compensate for transient losses of contact. When the user stops contacting the board the movement is complete, and the canvas module automatically checks for splices.

Grow/Shrink and **Spread/Fold.** These operations are triggered whenever one contact point is over an edge (divider) and a second point touches an opposite edge (divider) on the same cel (strip). Once the gesture has been recognized, only movement along the operation's axis is considered.

Play toggle. This operation is triggered anytime there is a single contact point on a cel; the play is performed when the finger is raised from the board. Recognizers for some operations (e.g. jog) can override the impending action if the user decides to extend the gesture (e.g. makes a circular gesture). The double-tap reset to key frame was not implemented.

4.3. Scene Detection

We would like to provide the user with a method of scene based editing by enabling some kind of scene boundary detection capability within our system. Ideally, we would like the user to initially be presented with a broad overview of the scenes in their clips.

They could then take a scene and expand it to reveal the lesser scenes within, a process that could be carried out recursively all the way down to the atomic frame level.

4.3.1. Current Implementation

Video segmentation breaks up a clip into smaller pieces that enabled semantic time zooming. However, there are many factors involved in determining where to segment a given video, many of which centre around the user's intention. Sometimes a user may just want to see more detail within a given section, in which case only a simple time-based segmentation is required. In other cases users may wish to segment a video based on scenes. This requires a more complex video analysis based segmentation. There are a myriad of different ways a user may wish segment any given video clip. The user may even wish to segment different areas of the same video clip in different ways. As developers we can make educated guesses as to how a user may wish to segment a video, but in the end it is the user who knows. The key to any video segmentation mechanism will be to allow the user the freedom they require to perform their intended tasks, while at the same time shielding them from the underlying details.

Spreading gives more screen real estate to the content of the selected strip span. Where there once was one cel being displayed there may now be two or three cells, each of which will display content that was in the original cel but may not have been visible. Currently, our video segmentation implementation is limited to time based segmentation. When a user zooms in on a given cel, that cel is broken down into equal length segments. For example, if a cel contains 2 seconds worth of video and the user spreads that cel into two cels, the frames will be partitioned into two equal segments of 1 second each. The time partitioning is done on the fly, eliminating the need for any video analysis or pre-processing.

4.3.2. Segmentation and Clustering

Time based segmentation takes no account of video content other than video length. However, there is a rich source of data in the video itself. We would like to take advantage of this information source and provide the user with scene based segmentation, and perhaps even scene clustering capabilities. Conceptually, a film editor breaks a movie down into smaller sections, most likely based on the individual scenes in the movie. For example, a video may start with a scene in the kitchen, cut to a scene in a smoky French café, flash quickly to some shady activity in a back alley then cut back to the kitchen scene. We would like to be able to automatically find these scenes in a movie and base segmentation around individual scenes rather than time.

A simple use case scenario would start with a user opening some video clip and initially being shown some number of scenes. We will call these the top-level scenes. These scenes could be chosen on a time basis so as to provide an even overview of the video. The user may then decide that they want to work on a scene that they know is near the beginning of the video, but the scene is not currently visible. They stretch out the first segment shown which reveals some of the scenes between the first two top-level scenes. They can repeat this stretching until they get to the desired scene. If the user has some idea about the ordering of scenes within a video it should be quite easy to pull out the exact scene desired. This removes the need to jog and allows the user to easily pull out the desired scene and work on it individually or within the context of the rest of the video.

However, with the addition of scene boundary detection comes the problem of determining what type of zooming the user wishes to perform. Do they want to zoom in to a particular scene, or do they just want an overall time-based zoom? Some simple heuristics could base the type of zooming on the zoom context. If the user is at a coarse level of granularity it is likely that they would wish to perform scene based zooming. If a user is focused on a single scene it is likely that they would wish to perform time-based zooming. Of course, these hypotheses would require user testing to determine if they are useful.

In addition to scene-based segmentation, an editor may be trying to decide where and how to re-cut a given scene, which may by spread all over the video. This would be an ideal situation for scene clustering. We could analyze scenes and determine which scenes are similar. This would enable an editor to easily find all the scenes in the kitchen, or all the scenes in the smoky French café. This capability is inherently non-linear and hence may not integrate well with the gestures metaphor, but it could be very useful nonetheless.

4.3.3. Algorithmic Speculations

It seems that it would be very difficult to develop an explicit algorithm that would determine scene boundaries accurately and cluster scenes meaningfully. However, we may be able to address this challenge indirectly by developing an adaptive scene detection system. We could train the system to detect scene boundaries and teach it how to group scenes meaningfully. Furthermore, we could have the users themselves train the system to perform as they see fit. This approach has worked very well for language recognition and other "fuzzy" problems.

One way to implement this technique would be to analyze individual frames and store several pieces of data for each frame, such as average contrast, color levels, gain etc… However, we need not analyze each frame in its entirety; we need only look at a minutia of each frame. This is often the approach taken in biometric data analysis as there is a vast amount of data but only a small percentage of it is of any interest. We could then store data about each minutia point, from which we could perform our analysis. As a user trains the system, they would essentially be telling the system which minutia values to look for when performing scene boundary detection and scene grouping. Values such as the rate of change of a value over time, or average value differences could easily be compared and learned by the system.

Unfortunately, due to time constraints these future considerations could not be implemented or fully explored. However, I think that the value added to the system would be considerable had they been put in place. The approaches mentioned above would allow for a very customizable system, and a very flexible system overall. As users requirements would likely change from user to user, this would make for a much richer and useful tool.

5. Evaluation

The direct manipulation video editing interface was evaluated by informally soliciting the opinions of a few users that we let play with our system. Ideally, this type of evaluation should be carried out iteratively throughout the different design and implementation stages. However, due to time constraints we only did a post-implementation evaluation test. This provided feedback on whether the system met the expectations of users.

5.1. Sample

We gathered a small sample of test subjects for this evaluation. This limited the formal accuracy of the evaluation, but still provided a lot of useful feedback. The test subjects were computer science undergraduates and a visual arts graduate. They ranged from moderate to novice users in using video editing systems, which included iMovie, Premiere, Final Cut Pro, Media 100, and Microsoft Movie Maker. All had been thoroughly exposed to the Video Bench concepts before the study began, further limiting the applicability of the results.

5.2. Method

Before each user tested our system we gave them some general directions about how they should use the Video Bench system. They were also provided with a user manual (see Appendix C), which describes verbally and pictorially how to use the system. We asked them to perform specific tasks as well as do their own exploration of the system. All subjects used the touch interface first, then repeated their experiments using the mouse later. Guidance was provided for the test subject as needed, if they required any assistance or had any questions.

Each subject was encouraged to "think aloud" while using the system, verbalizing what she was thinking or what she wanted to achieve. We observed the users while they were using the system and asked them short questions to encourage dialogue between evaluators and users. The test subjects were surveyed afterwards via a questionnaire (see Appendix B). Three methods of questioning were used in the questionnaire: scalar, open-ended and multiple choice. A Likert 5 point scale measuring agreement or disagreement with statements was used. We used both positive and negative statements for rating in the questionnaire, to ensure that the test subjects were reading the questions carefully.

5.3. Questionnaire Results

These results were obtained after the users had tested the system with both the mouse and touch interfaces.

The users felt that they needed to learn a lot before they could use the system on their own and that they needed the support of a technical person initially for guidance. They did not feel confident when using the system.

They thought the most people would not learn to use the system quickly since it was not very easy or intuitive to use. They also attributed this problem to the fact that the various functions of the system were not well integrated. Therefore, they felt that they would not use a system like this frequently.

They users were asked whether they would purchase the system for video editing at home if the system became commercially available. They felt that their decision would mainly be based on retesting a more complete and final version of the product. Some also had concerns about the cost as well as the hardware requirements.

5.4. Observation Results

The users felt that the tool is very futuristic and that it has a lot of potential with further development. While they seemed thrilled and excited upon seeing the new system, they seemed uncertain as to how to proceed when seated in the operator position.

The users seem to enjoy using both the mouse and the direct touch manipulation options, since they serve slightly different functions. For example, the mouse seems to be better for precise clip jogging, while operations such as split and fold are more intuitive with touch gestures.

Our test subjects were asked to repeat the tasks done via touch manipulation with the mouse. They seemed to have a great deal of difficulty at first with this transition, but as time passed they seemed to become just as comfortable using the mouse. Certain functions such as play toggle were simple and intuitive to activate with the mouse; however, others functions such as grow/shrink were more difficult and required reference to the user manual. Some test subjects preferred the mouse operations over hand manipulations since it provided them with more precision. In general, the edges of the table provided the least precise contact. Some functions were also flaky in practice: the program sometimes seemed to “lose” the cel that the user was growing, and the tools menu jumped around the bench at random.

A problem that was encountered several times was that it is difficult for the users to get back to the first or last frame of a cel. They need to manually jog through to rewind to the beginning or forward to the end, and this process can be time consuming if the cut is lengthy. Some users thought that they could drag the cursor directly to cue the video. Another problem with the dial jog was if a user was rotating the dial in one direction then reversed directions, the program seems to completely stop or slow down dramatically before returning to normal operational speed. This seemed to confuse and worry users.

The users liked the contrails and the highlighting used when performing functions such as join. Other features were controversial, e.g. mixing the sound of two movie clips being playing simultaneously. Also, using the mouse, some incomplete strip rotation functionality was left in the code, attached to the mouse wheel. Since the wheel also served as the middle mouse button, users often rotated strips while moving them without realizing it. While natural use was the intent of this operation binding, it should have been removed prior to testing since rotation was not fully implemented and could easily crash the application.

5.5. Suggestions

Better space management is required, since the film strips can easily clutter the available space. For example, if the tools menu happened to be nearby when spreading a strip, sometimes the strip would get accidentally trashed, duplicated, or squished, frustrating the users. Part of this problem can be attributed to the small size of the DiamondTouch table. The original two-handed splice design would've also helped in this respect. Due to the space constraints, the system is probably not suitable for use by simultaneous multiple users at this time.

Furthermore, when attempting to move strips, if a user tried to use two widespread fingers of one hand the system sometimes failed to recognize the gesture. Most users resorted to using two hands for moving strips. Since most users would have preferred the single-handed method, the move gesture recognizer needs to be fine-tuned. Also, in general, more precise touch detection is required with the tool, especially for stopping at the correct cel.

Users requested some extra features while using the system. The requests included an explicit time line view for the clips, and also some sort of separate audio manipulation. Many users were looking for some sort of undo function to recover from their mistakes; this issue was compounded by the accidental application of the irreversible trash tool. Even if a general undo function is not possible, it should at least be possible to recover deleted strips. It would also be useful if there was an option for coarse but quick fast forwarding or rewinding of cuts, perhaps through direct dragging of the cursor (if it's made larger).

When using the mouse operations, error messages were displayed if the user tried an invalid operation. Users really liked this feature, since it would immediately inform them of their mistake. These error messages should also be extended to the hand manipulation.

It might be nice to introduce an outline for the cels or wrap the cels in a white box to clearly separate the dividers from a dark cel. This boundary would be particularly useful for mouse operations where the hit area to activate a function is very precise. Another visual issue is some tools are the same color as the highlight, making it impossible to tell when they're about to be activated. It would be more helpful to use separate colors. Finally, some functions need to be renamed. For example, although the words "splice" and "split" have different (opposite!) meanings, their similar sound confused some users. It might be better to change "splice" to "join".

6. Conclusion

The project was successful. The Video Bench prototype achieved most of its goals in a very tight time frame, though it is still far from a practically usable system. The prototype is visually attractive, and users seemed to enjoy their experience, even while dealing with the frustrations common to prototypes.

6.1. Achievements

We have integrated gesture recognition and video rendering with a layered canvas system. The multipoint gesture recognition on the Diamond Touch table is a novel contribution, and the Video Bench seems to be the only video editor that supports direct gestural manipulation with continuous feedback. Early evaluation results indicate that direct gestural editing may be practical for certain types of tasks, though further research is needed to determine whether it is valuable.

Unfortunately, we had to sacrifice some features that proved difficult to implement, namely strip rotation, multi-hand gestures and scene segmentation. We also didn't get around to testing the system with multiple users, though the prototype technically supports this functionality.

6.2. Future Work

First and foremost, the project needs further evaluation. A formal procedure should be followed to compare the system's utility against other video editors, as well as comparing the touch, mouse interaction and mixed interaction modes. The system should also be evaluated in multi-user cooperative settings.

On the implementation side, we should complete the work on strip rotation, enhancing simultaneous multi-user interaction. This will require a revamping of the gesture recognition engine. The scene segmentation and clustering engine also needs to be implemented, and a better solution found for playing composite cuts without stuttering. The system would probably benefit from some overall optimization work.

We eagerly await improvements to the Diamond Touch table. A larger work surface and more precise and faster contact detection would go a long way to improving the system's usability.

Finally, for long-term future research, it would be interesting to investigate ways to extend the system to deal with multiple separable tracks (e.g. audio and video), and special effects (transitions, fades). It might also be worth looking into integrating the direct manipulation paradigm with a traditional timeline, to take advantage of the strengths of both approaches. The current prototype provides an excellent basis for future research.

References

[Bux92] William Buxton: *Continuous Hand-Gesture Driven Input*. In Proceedings of the 5th annual ACM symposium on User interface software and technology, 1992, p. 199-208.

[GBC+00] Andreas Girgensohn, John Boreczky, Patrick Chiu, et al.: *A semi-automatic approach to home video editing*. In Proceedings of the 13th annual ACM symposium on User interface software and technology, p. 81-89, 2000.

[GBW01] Andreas Girgensohn, John Boreczky, Lynn Wilcox: *Keyframe-Based User Interfaces for Digital Video*. IEEE Computer, 2001, p. 61-67.

[KB91] Gordon Kurtenbach, William Buxton: *Issues in Combining Marking and Direct Manipulation Techniques*. In Proceedings of the 4th annual ACM symposium on User interface software and technology, 1991, p. 137-144.

[LBS85] SK Lee, William Buxton, K. C. Smith: *A Multi-touch Three Dimensional Touch-Sensitive Tablet*. In Proceedings of the SIGCHI conference on Human factors in computing systems, 1985, p. 21-25.

[LMC+] A. Chris Long, Brad A. Myers, Juan Casares, Scott M. Stevens, and Albert Corbett: *Video Editing Using Lenses and Semantic Zooming*. Submitted for publication.

[MGM93] James Matthews, Peter Gloor, Fillia Makedon: *VideoScheme: a programmable video editing systems for automation and media recognition*. Proceedings of the first ACM international conference on Multimedia, p. 419-426, 1993.

[MP94] W. Mackay, D. Pagani: *Video mosaic: laying out time in a physical space*. In Proceedings of the second ACM international conference on Multimedia, p. 165-172, 1994.

[Rub91] Dean Rubine: *Specifying Gestures by Example*. In Proceedings of the 18th annual conference on Computer graphics and interactive techniques, p. 329-337, 1991.

[SG99] Bikash Sabata, Moises Goldszmidt: *Fusion of Multiple Cues for Video Segmentation*. SRI International, 1999.

[UMY91] Hirotada Ueda, Takafumi Miyatake, Satoshi Yoshizawa: *IMPACT: an interactive natural-motion-picture dedicated multimedia authoring system*. In Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology, p. 343-350, 1991.

[WCC01] Jihua Wang, Tat-Seng Chua, Liping Chen: *Cinematic-Based Model for Scene Boundary Detection*. National University of Singapore, 2001.

[Wil97] Mark Willey: *Design and Implementation of a Stroke Library*. 1997.

A. Statement of Contributions

James Chisan

In addition to my work on the gesture sections of the initial proposal and final report, I was responsible for making sense of the DiamondTouch table data. This consisted of several sub-components: calling and setting-up the DT drivers appropriately, transformations of input data according to world coordinate system and calibration parameters, facilitating calibration, performing finger tracking, gesture coordination and finally integration of the gesture system into the rest of the program. Reid and I collaborated extensively to implement play/stop, spread/fold and move gestures.

Azarin Jazayeri

I wrote the Evaluation section of the original proposal, as well as the Evaluation chapter in the final report. I made the original set of power point slides for the presentation (although Piotr modified it to better suit his presentation). I directed the user testing and evaluation by preparing the user manual and the questionnaire, and by observing the test subjects

Note: As stated in our proposal, I was also going to work on the mouse manipulation. I got to a good start and made the designs for simulating the hand manipulations using the mouse and completed the readings (papers and documentations of tools used) that Piotr told me about. However, due to illness and shortage of time, I was unable to finish the task. I took me quite a bit of time to actually get everything set up (Eclipse, JMF, and other tools used) with a lot of help from Piotr. By the time this was completed and I was reading the code written it was much too late for me contribute to coding.

Jeff Cockburn

My contributions to the project focused on video playback and video segmentation. I collaborated with Jesse and Piotr to graft the JMF into our project. As expected using any new technology the ratio of time spent learning the JMF to generating usable code was far more in favour of learning than implementation. Unfortunately my involvement with the JMF dug into the time I was able to dedicate to video segmentation. I was able to develop a simple time-based segmentation algorithm to perform segmentation, but this did not provide the semantic zooming we desired. I developed some code to detect scene boundaries within a video but the results of segmentation were unreliable and processing was extremely slow. Some problems included detecting scene transitions such as fade-outs, zooming and quick panning. I wanted to implement a trainable system for scene detection; however, my lack of experience in this area required far more research than time constraints allowed.

Reid Garner

I mainly coded gestures with James; this work included integrating with the DiamondTouch display, which required a lot of debugging and tweaking. I wrote about implementing some of the gestures and problems that were encountered with them as well as the DiamondTouch for the final report.

Piotr Kaminski

I came up with the original concept, the user interface and the design of all the operations. I wrote the introduction, video research background, operation design, canvas implementation and conclusion sections in the proposal and final report, and thoroughly edited both documents. I designed the system architecture and implemented all canvas drawing and mouse manipulation. I also did extensive refactoring on the video subsystem, and a fair amount of general debugging (except gestures). I prepared and delivered the in-class presentation, based loosely on draft slides provided by Azarin.

Jesse Wesson

I provided a brief introduction to commercial non-linear video editing software in the proposal. My area of focus for the project proper was video playback and manipulation. Before we started development, I examined the two media frameworks QuickTime and JMF to see which would support our project best. After we decided on JMF, I started to implement the custom video renderer that would provide the images needed by Piccolo. Understanding JMF proved to be quite an undertaking and took some time but I eventually coded the necessary implementation needed (although behind schedule). This was then incorporated into the design Piotr provided and the stubbed out classes were fleshed out. Ironing out bugs also filled my time spent on this project.

James Chisan

Azarin Jazayeri

Jeff Cockburn

Piotr Kaminski

Reid Garner

Jesse Wesson

B. Evaluation Questionnaire

Informed Consent Information

The general purpose of this study is evaluation of our project. Please be informed that, because this study is both anonymous and confidential, you will not be asked to return a signed consent form along with your questionnaire, as is normally required in studies of this type. Instead, return of your completed questionnaire will be taken as indication of your informed consent.

Please also be informed that:

- Your participation in the study is fully voluntary.
- All information you provide in this study is anonymous, and will be kept strictly confidential. Any report of the study will not identify you personally in any way.
- Although the length of time it takes different people to fill out the questionnaire will vary, for most people it should not take more than 15-20 minutes to complete.

Please answer the questions below:

1. Have you used any video editing system before? If yes, please specify which one?

2. Do you consider yourself a) an **expert** user in using video editing systems
   b) a **moderate** user in using video editing systems
   c) a **novice** user in using video editing systems

3. What did you like/dislike about the system?

4. Do you prefer **both** the mouse option and the direct hand manipulation option or do you prefer **only** direct hand manipulation? Please, explain why?

5. If this system became commercially available would you purchase it for video editing at home? Why or why not?

Please circle the appropriate number.
(5= strongly agree 1= strongly disagree)

6. I need to learn a lot of things before I could get going with this system

   1 2 3 4 5

7. I felt confident using the system

   1 2 3 4 5

8. I think most people would learn to use the system quickly

   1 2 3 4 5

9. I found the various functions well integrated

   1 2 3 4 5

10. I think I would need the support of a technical person to use this system

    1 2 3 4 5

11. I thought the system was easy to use

    1 2 3 4 5

12. How would you rate this program?

    1 2 3 4 5

13. I think I would use a system like this frequently

    1 2 3 4 5

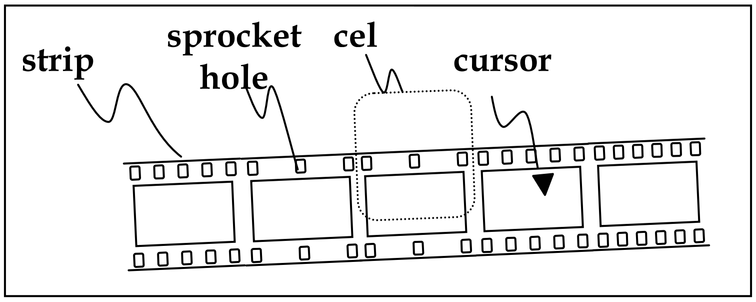
14. Any suggestions/comments.

Thank you for taking the time to fill out this evaluation questionnaire.

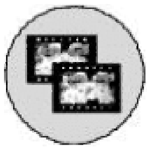
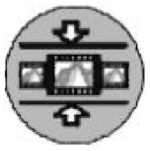
C. User Manual

This is the user manual as it was provided to the evaluation subjects. An updated version will be included with the software package.

Anatomy of a strip



Tools available

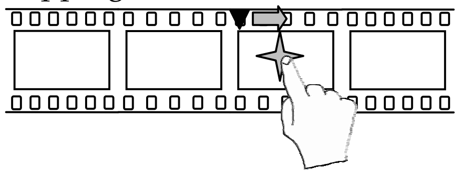
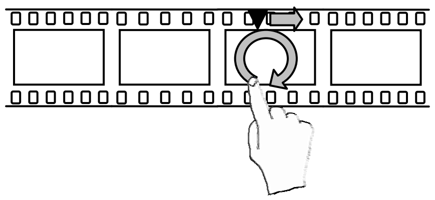
Tool icon	Effect
	Duplicate: makes a copy of the strip
	Trash: deletes the strip
	Squish: shrinks all cels in the strip to their minimum size

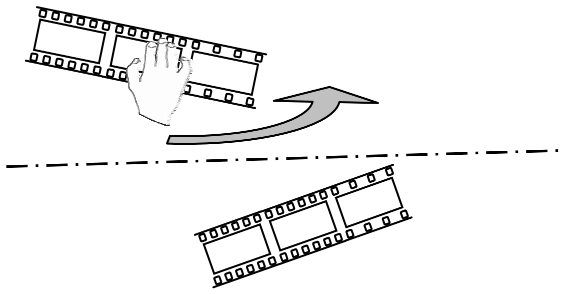
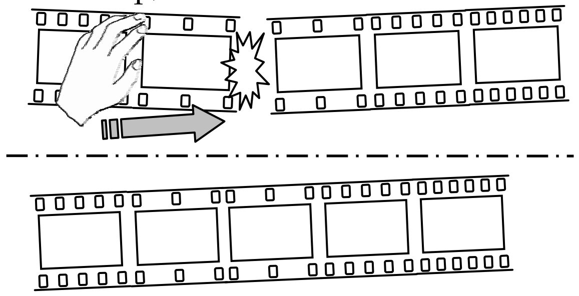
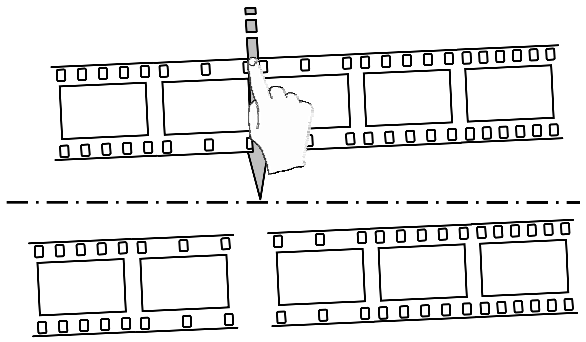
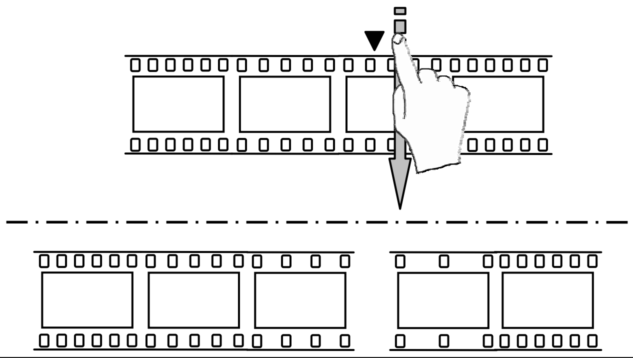
Operations Using the Mouse

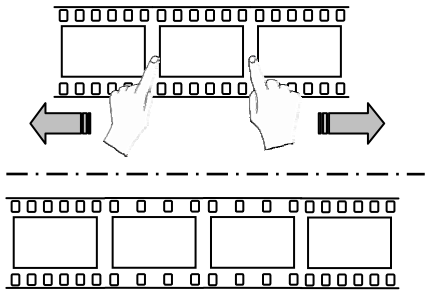
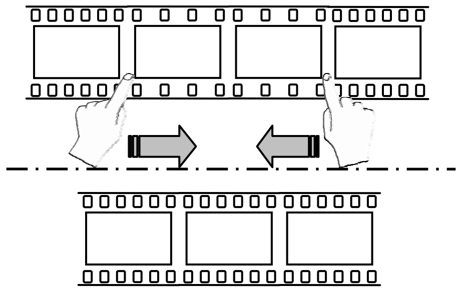
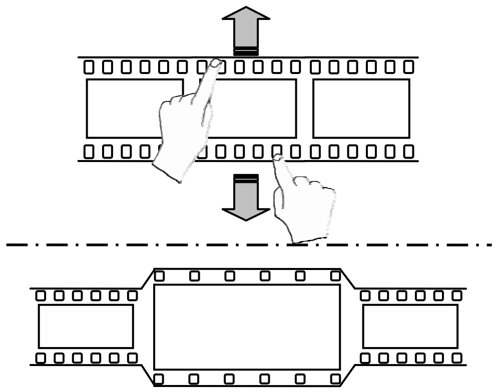
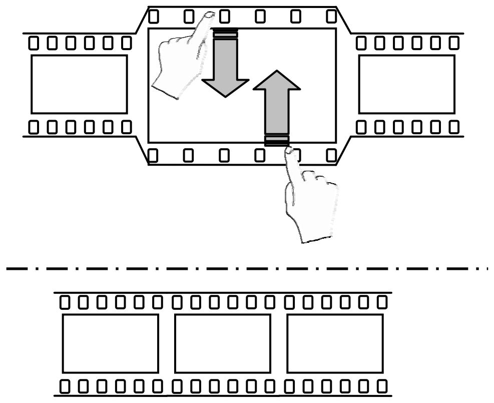
Operation	Mouse
Play / Pause	Left click frame
Reset to key-frame	Double left click frame
Jog	Roll wheel over frame
Move strip	Middle drag strip
Splice strips	Middle drag one strip until end divider lines up with end divider of

	other strip, then release
Split between cels	Right click on divider
Split within cel	Right click on frame
Spread / Fold	Left click on one divider, then left drag another divider on the same strip
Grow / Shrink	Left drag top or bottom edge
Show tools	Middle click on background; middle drag to move tools
Apply tool	Middle drag one strip until an end divider overlaps the desired tool, then release

Operation using hand gestures

Operation	Mouse
Play	Tapping an inert cel 
Pause	Tapping a playing cel pauses playback
Reset to key-frame	Tapping it twice
Jog	Make a small circle within a cel, then run your finger clockwise and counter clockwise in a circular motion 
Move strip	“Grab” a strip and drag it to another location, then release it

	
Splice strips	Middle drag one strip until end divider lines up with end divider of other strip, then release 
Split between cels	Move down the cels from the top to the bottom of the strip, then release 
Split within cel	Pull your finger through the cel 
Spread	Positions your fingers between cels and moves them apart

	
Fold	Select a span of cels and slides your fingers together 
Grow	Drag hand in opposite directions up and down 
Shrink	Drag hand toward each other 
Show tools	Put your hand down on an empty part of the table, the icons will pop-up

Apply tool	Move the film strip to the appropriate icon

Try the following tasks using hand manipulation:
- Remove the beginning (or end) of a clip.
- Extract interesting scenes,
- Duplicate and reorder them.

Now do the above using the mouse.