Video Browsing by Direct Manipulation

Pierre Dragicevic[2,1], Gonzalo Ramos[1], Jacobo Bibliowicz[1], Derek Nowrouzezahrai[1], Ravin Balakrishnan[1], Karan Singh[1]

[1] Department of Computer Science
University of Toronto
{bonzo, jacky, derek, ravin, karan}@dgp.toronto.edu

[2] INRIA
France
dragice@lri.fr

ABSTRACT

We present a method for browsing videos by directly dragging their content. This method brings the benefits of direct manipulation to an activity typically mediated by widgets. We support this new type of interactivity by: 1) automatically extracting motion data from videos; and 2) a new technique called *relative flow dragging* that lets users control video playback by moving objects of interest along their visual trajectory. We show that this method can outperform the traditional seeker bar in video browsing tasks that focus on visual content rather than time.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

INTRODUCTION

Despite its many advantages [6, 15, 30], direct manipulation has not been adopted consistently across all computer applications, and there remain tasks that could benefit from a better application of its principles. For example, video players provide a seeker bar for linearly browsing through the video's timeline. While effective, like many GUI widgets, it is only an intermediary between the user and the object of interest to be manipulated [6]. Often this object is in the video image itself, and it might be beneficial for users to directly manipulate it. For example, a video analyst studying a pool shoot might want to directly drag a ball to find the precise moment when it hits another ball (Figure 1). Although it does not change the video content, this interaction technique belongs to the category of direct manipulation techniques because it maintains a close match between user input and the system's output.

In addition to providing users with a fulfilling sense of control [15], direct manipulation can also be very efficient. Whereas a seeker bar is excellent for time-centric browsing tasks, direct manipulation might more efficiently support accurate space-centric browsing. As such, both techniques should be viewed *not as rivals but more as complementary tools*. This paper makes several contributions by identifying new classes of direct manipulation techniques, illustrating their use and implementation in a media player, and presenting a study showing that they can yield remarkable gains in performance for space-centric browsing tasks.

CHI 2008, April 5–10, 2008, Florence, Italy.

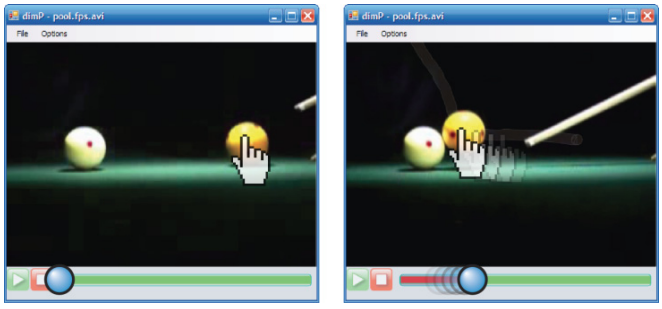

Figure 1: The Direct Manipulation Video Player used to scrutinize a pool ball's motion by directly dragging the ball to the point where it hits another ball.

Directly interacting with video content raises several research challenges. From a technical perspective, one needs be able to extract relevant motion data from videos. From an HCI perspective, dragging video content is fundamentally different from dragging traditional GUI entities. Firstly, the content to be manipulated is not clearly segmented spatially and involves arbitrary deformations. Secondly, this content is constrained to predefined motions. Thirdly, dragging an object also animates the rest of the visual scene, which can produce illusory or "induced" motions affecting motor tasks. We explore solutions to all these challenges. We also outline the differences of our approach from a recent paper [18] exploring similar ideas that were brought to our attention during the review process.

RELATIVE FLOW DRAGGING

Our video navigation interface uses a new technique we call *relative flow dragging* – a general solution to the control of moving imagery through direct manipulation. It decomposes arbitrary 2D motions into individual pixel trajectories, and allows scrubbing over these trajectories in a way that accounts for the phenomenon of induced motion, such as those produced by dynamic backgrounds. We now describe the technique in detail, including how it follows direct manipulation principles, and illustrate its benefits.

Directness

Hutchins et al. [15] argue that the *directness* of a user interface depends on factors such as how *responsive* and *unobtrusive* the system is, and how closely the input language of the user interface *matches* its output language. For example, dragging an object on the GUI is highly direct because the output it generates (the object's motion) is very similar to the input (the user's hand movement).

Beaudouin-Lafon [6] categorized common sources of *indirectness* in interfaces. These include spatial and/or temporal separation between the user's action and the system

response (high *degree of indirection*) and dissimilarities between action and response (low *degree of compatibility*). For example, panning with scrollbars suffers from these two types of indirectness, as opposed to directly dragging a document. A third type of indirectness is present in tasks requiring more degrees of freedom than the available user's input (low *degree of integration*).

Matching Gestures with Motions

Designing an interaction technique often involves deciding how the user's input will be interpreted into changes in the system. Following direct manipulation principles, this should be decided according to the characteristics of the system's outputs, rather than the system's internals. For example, consider an internal variable whose variations cause a motion on screen, such as current time moving a clock's hands. Regardless of this variable's nature, the most direct way for users to control it visually is by specifying the expected motion, i.e., dragging the clock's hand [11].

We argue that designing for direct manipulation involves matching user's gestures with the observed visual motion. This is straightforward when the possible gestures allow us to express all possible visual movements – i.e., when the *gesture space* matches the *motion space*, e.g., using a 2D device to pan a map. However, incompatibilities between motion and gesture spaces are common. Adjusting the input device to the task [9] is an effective but often impractical solution. As a result, a number of techniques have been designed to map a limited 2D gesture language to a wide range of visual motions while preserving an illusion of direct manipulation. Examples are scaling objects, rotating objects [7, 11] and 3D manipulations [32] using a mouse.

The control of high-dimensional motion spaces is a well-recognized problem, especially in the field of 3D GUIs. In comparison, little attention has been paid to the control of low-dimensional spaces. Here we address the issues related to the direct control of motions having only one degree of freedom. Our solution, *relative flow dragging*, is related to three simpler families of direct manipulation techniques: *curvilinear dragging*, *flow dragging* and *relative dragging*.

Curvilinear Dragging

Curvilinear dragging consists of moving a point constrained to a 2D curve using a 2D pointing device (Figure 2a). Examples of curvilinear dragging abound in current GUIs but they mostly involve straight lines. For example, scrollbars, sliders and menus project 2D mouse movements onto a 1D axis [6]. Cascading and flow menus are not direct manipulation techniques per se, but involve steering, an action similar to that of following a curve [1, 13].

Curvilinear dragging on arbitrary curves has also been used in specific applications. The 3D modeling tool Maya allows points to be moved along 3D curves [2]. The Cabri-Géomètre learning environment [14] allows studying geometric figures by dragging construction points on curves. Baudel et al. [4] proposed a curvilinear drawing technique based on a French curve metaphor. Ngo et al.'s [23] system lets its users animate parameterized graphics by dragging objects along their trajectories.

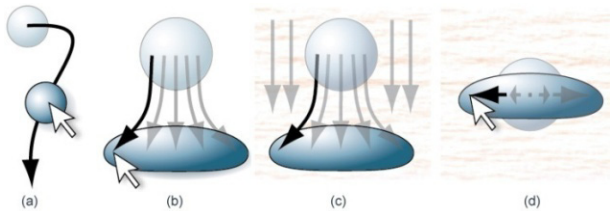

Figure 2: Three classes of constrained direct manipulation techniques: (a) *curvilinear dragging*; (b) *flow dragging*; (c, d) *relative flow dragging* compensates for moving backgrounds.

Flow Dragging

Flow dragging is a generalization of curvilinear dragging. It involves direct manipulation of *arbitrary motions* having only *one degree of freedom*. By "arbitrary" we mean that besides translations, the motions can include any visual transformation, such as the deformations of a bouncing ball. By "one degree of freedom" we mean that the whole motion can be mapped to a scalar variable, such as time.

Supporting *flow dragging* simply requires supporting *curvilinear dragging* on a family of curves: since the motion has only one degree of freedom, each point of the image has a well-defined trajectory which can be invoked upon a button press (Figure 2b).

Relative Dragging

We see the design of direct manipulation techniques as a problem of matching gestures with *observed* visual motions. The observed motion of an object is influenced by the motion of its surroundings, a phenomenon known as *induced motion* or the Duncker illusion [26, 41]. Research suggests that induced motion also affects motor tasks [28].

This phenomenon tells us that people perceive *relative* rather than *absolute* motion, thus we call direct manipulation techniques focusing on the control of relative motions *relative dragging* techniques. For example, one can combine pointer and background motion so that their relative movement matches a user's hand movement. This can be achieved by moving background objects in the opposite direction of the hand's motion. Such an approach is very common in scrolling-based 2D arcade games.

Since complex moving imagery is likely to produce induced motions, flow dragging is best combined with a relative dragging approach. For example, suppose that the deformation of a rubber ball occurs with a background motion (Figure 2c). Dragging on actual trajectories might be difficult because they are different from the motions the user sees. Manipulation can be facilitated by subtracting background motion from these trajectories (Figure 2d). We call this method *relative flow dragging*.

Challenges

There are two key challenges in the design and implementation of relative flow dragging. First is the extraction of trajectories and computation of relative motions. Second is the design of curvilinear dragging: although anecdotal examples of curvilinear dragging exist, they do not behave well on arbitrary curves. In the following sections, we present a system that supports relative flow dragging, and then discuss in detail how we addressed these two issues.

THE DIRECT MANIPULATION VIDEO PLAYER

We implemented and tested relative flow dragging in an interactive video player prototype that we call DimP (for **Di**rect **m**anipulation **P**layer). In addition to providing all the standard controls of a traditional movie player (Figure 1), DimP allows users to go back and forth through the video's timeline by directly manipulating the video content, in a way similar to Kimber et al.'s system [18]. DimP can be downloaded at www.aviz.fr/dimp

When DimP loads a video, it first checks if the video is already accompanied with motion data. If it is not, this information is automatically extracted then saved as a separate file. We believe that it is reasonable to expect motion information to be created and potentially authored off-line. The loaded video is then fully uncompressed into memory to allow immediate access to any video frame.

Figures 1, 3 and 4 illustrate DimP in action. Via three scenarios, we show how DimP can provide a radically different way to navigate video streams as well as transcend some inherent limitations of the traditional seeker bar.

Scenario 1: A surveillance video shows a car that has been parked in a particular spot all night (Figure 3). We might want to access the point in the video where the car is in the process of parking. On a traditional video player we have to carefully drag the seeker bar's handle until we visually detect the frame of interest. With DimP, we can detach the parked car (Figure 3a) directly out of the parking spot. This action immediately takes us to a point in the video just moments away from when the car has finished parking (Figure 3b). From that point onwards (or backwards) we can use the traditional seeker bar as a complementary tool to find out, for example, who is leaving the car (Figure 3c).

Scenario 2: A video of a scene on a busy street, where many cars and people are buzzing about, is being watched (Figure 4). Cars in the scene regularly accelerate, slow down or stop. Let us suppose that we wish to advance frames so that a particular car arrives at a particular location. This is difficult to achieve using the seeker bar, since not only is the mapping between the bar's handle and the car's movement unknown to the user, but it is also not linear in time and space. A smooth controlled dragging on the seeker bar can result in an erratic non-uniform movement of the car. This disparity stems from the indirectness of the seeker bar and is addressed by letting users simply drag the car over the video scene at the rate they see fit.

Scenario 3: DimP can be useful in scenarios where a user is interested in analyzing a particular video scene. For example a sports coach is interested in studying a spring board diver's performance or a physics student wants to follow the intricacies of a complex pool shoot. Using direct manipulation, the coach has access to the many complicated movements involved in throwing a ball or twisting one's body in mid-air. Likewise we can also imagine the student switching his/her attention between different key points of

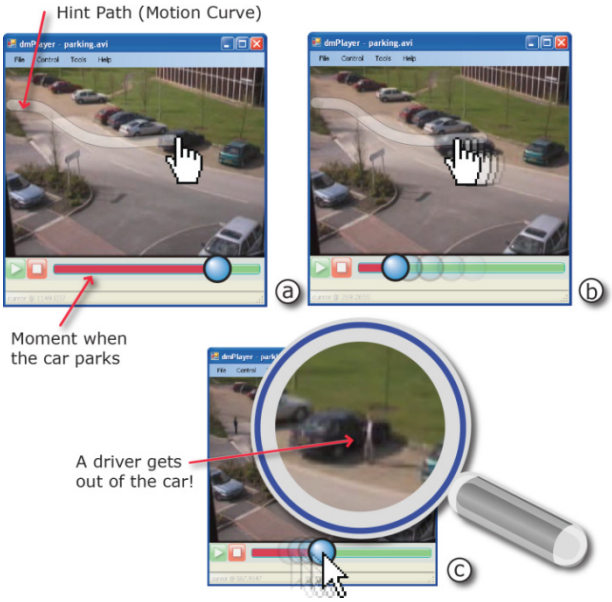

Figure 3: DimP can help find a particular event in a video. a) The user clicks on a parked car, its trajectory is shown. b) The user detaches the car from its parking location, which takes the player to the point where the car was parking. c) The user drags the seeker bar to temporally browse around this point in space to find the person exiting the car.

Figure 4: Background stabilization on a scene where the camera pans upwards. (a) The actual motion of a car is hard to perceive by users. (b) Background stabilization presents users with a motion path close to the car's perceived motion. (c) When the car is dragged, background stabilization shifts the video frames and leaves a trail of previously displayed frames.

the scene, such as the point where a ball is hit by the cue, or when it impacts another one. All these operations can be performed by direct manipulation in the video viewport, without having to resort to the seeker bar (Figure 1). Hint paths are useful in these scenarios because motion trails are known to help animation and motion analysis [37].

Trajectory Visualization

One of the biggest challenges we faced in the design of the visual feedback in DimP is the trade-off between providing the illusion of direct manipulation and guiding users when they stray from a particular motion path. It is important to provide a trajectory visualization that is unobtrusive and at the same time helpful when necessary.

Preview

We change the appearance of the pointer's cursor from an arrow to the shape of a hand whenever the pointer is hovering above a region of the video frame where the motion is significant. Users can then engage in direct manipulation by simply clicking and dragging over the video's frame. As a user clicks over the video frame, the system displays a hint path corresponding to the estimated trajectory of the point located under the mouse cursor (Figures 3 to 5). The hint path has been made very subtle in order to reinforce the metaphor of direct manipulation.

Emphasizing

Since objects in a video scene follow prescribed trajectories, users cannot drag them to arbitrary locations. DimP however maps mouse movements to object motions in a very conservative way, so that users *do not have to carefully follow their path*. For feedback purposes, we emphasize the hint path as the user's dragging path diverges from an object's motion (Figure 5). If a user drags the pointer far away from the hint path for more than 2 seconds, we play a "snapping" sound and terminate the interaction.

Position Feedback

We display a red cursor over the hint path that indicates the position over the trajectory curve corresponding to the pointer's location. If the pointer perfectly follows the path, the cursor is located under the pointer. As the pointer's path diverges from the object's motion path, the cursor provides extra feedback as to the effects of the dragging operation (Figure 5b). This feedback also allows for video navigation with sub-frame accuracy.

Background Stabilization

Pilot user testing sessions suggested that camera pans could make video manipulation very difficult, an issue that can be addressed with *relative dragging*. DimP supports relative dragging by compensating for background motion.

Figure 4 illustrates how background stabilization helps to match the user's gesture with object's perceived motion. In this example, an upward camera pan makes a car's motion a downwards one, even though the car is seen moving up the street (Figure 4a). It is difficult for users to perceive or reproduce this absolute motion. Figure 4b shows the result of subtracting the global motion from the real motion: an upwards path closer to the car's perceived motion.

Figure 5: As the pointer drifts from horse's trajectory, the curve's style changes from (a) subtle to (b) emphasized.

Our background stabilization technique shifts video frames within the player's window as a user drags a particular object. Video frames leave a trail of blurred grayscale images which aid in putting the currently displayed video frame in context with the video scene (Figure 4c). This is similar to stitched panoramas or video mosaic techniques [12, 16]. Once the direct manipulation ends, the current frame springs back to fill the player's window frame.

TRAJECTORY EXTRACTION FROM VIDEOS

Supporting relative flow dragging in a video player requires knowledge of the motions occurring in the video – i.e., having a function that maps any point in any video frame to a curve representing its trajectory over time. We considered three approaches for obtaining this information:

Manual Annotation. We initially collected user reactions to relative flow dragging with a Java prototype that requires specifying the motion of points of interest across video frames. Manual annotation is a reasonable approach in the context of multimedia authoring [25] and can be complemented with automatic techniques. Still, fully automatic solutions are needed for video consumers to be able to experience direct manipulation on arbitrary video files.

Metadata Extraction. Most video formats already contain motion information used for decoding purposes [35, 39]. At this point we have no control as to the quality of this metadata, but this strategy is worth exploring in the future.

Automatic Estimation. Motions can be estimated with acceptable accuracy using video processing and computer vision techniques. This is the approach we used in DimP.

Computer Vision Approaches

Automatically estimating motions from an image sequence is a classical computer vision problem that is still actively investigated [5, 17, 31, 33, 35, 39, 40]. Thus, instead of proposing a new method, we aim to use well-established vision techniques to demonstrate the feasibility of built-in support for the direct manipulation of video. We group motion estimation approaches into two categories:

Object Tracking: tracking involves following the motion of one or several objects of interest over a video sequence. Applications include motion capture and surveillance [40].

Optical Flow: optical flow computation estimates pixel velocities between adjacent video frames, without enforc-

ing time consistency. Applications include video compression, 3D reconstruction and robot servo control [5].

Both approaches have pros and cons. Tracking algorithms are efficient at following objects over long periods of time [17]. Using a tracking approach, Kimber et al. [18] extract moving objects and try to maintain their identity in the presence of discontinuities due to merging, splitting or occlusion. However, such techniques are very sophisticated and never 100% reliable. In contrast, optical flow approaches do not extract objects and hence do not handle merging or occlusion. But they can track a wide range of motions, such as the movements of a fluid.

In light of these tradeoffs, we believe that a sparse estimation of optical flows is an adequate first step for a general-purpose video player.

Extraction of Feature Flows

Our implementation uses a feature-based optical flow estimation scheme [20], where one interpolates the motion of salient feature points between frames. We reused Nowozin's C# implementation of SIFT (Scale-Invariant Feature Transforms) [24, 20], a robust feature extraction method that can match photos with different viewing angles or lighting conditions. SIFT has been used successfully in applications ranging from panorama creation [24], 3D reconstruction [34] and video tracking [33].

Feature flows are computed by detecting and matching SIFT feature points on two consecutive frames. Each match gives a motion vector. Unmatched features are discarded. A complete absence of matches is a good estimator for scene cuts. Once all of the video's feature flows are found, trajectory curves can be generated on-the-fly.

Construction of Trajectory Curves

Optical flows are inferred from feature flows by nearest-neighbor interpolation. The trajectory curve going through a given pixel of a particular video frame is then built by adding up flows forward and backwards in time. Since SIFT feature points have sub-pixel accuracy, cumulative errors are negligible. The process stops as soon as an unknown flow (scene cut) is encountered.

Finally, each of the vertices of a trajectory curve is tagged with the frame number it corresponds to. The floating-point video frame number of an arbitrary point on the curve is obtained by linear interpolation.

Estimation of Background Motion

We implemented a greedy screen-space binary partitioning scheme to find the most dense motion region in the space of feature motions. This algorithm yields the dominant or "most representative" feature displacement on a given pair of frames, which is identified with background translation and subtracted from the feature flow.

Our binary partitioning scheme is computationally cheap and has produced results that received positive response from early users of our system. More advanced tools such as K-means or Mean-Shift clustering algorithms can be used to detect multiple coherent motions and refine the computation of relative motion [10].

Current Limitations

Pilot user testing sessions revealed that the trajectories generated by our system match users' expectations fairly well, especially for videos involving large continuous motions. However, some limitations remain:

Speed: feature extraction & matching can be costly. While our implementation takes about 5s per frame on a typical desktop computer, research suggests that feature detection and matching can be faster, if not real-time [20, 33].

Sensitivity: in order to keep computation times acceptable, we sub-sample video frames to 128×128 grayscale pixels before processing. As a result, small objects are not tracked. Faster feature extraction algorithms could work on higher resolution images, thus improving sensitivity.

Discontinuous Motions: our implementation does not remember features which have been briefly occluded or moved outside the camera view. This issue is intrinsic to the optical flow paradigm and can be partly addressed by the integration of tracking techniques [40, 18].

Induced Motion: while straightforward, our background extraction method makes several simplifying assumptions about induced motion, e.g., it does not detect multiple coherent motions, nor does it account for non-translational induced motions [26].

Scalability: At this time, DimP works best for relatively short video clips. In addition to processing time, current scalability issues concern memory (videos are wholly uncompressed to support fast browsing) and lags produced by the construction of very long trajectory curves.

The trajectory curves we produce hold all the information we need for the direct manipulation of video. With this information, we can reflect a point's motion over a trajectory curve back into changes on the video frame number. In the next section, we elaborate on how users control these curvilinear motions using 2D mouse movements.

CURVILINEAR DRAGGING DESIGN

Flow dragging requires supporting curvilinear dragging on multiple curves. We assume a simple model where no dynamic transition between curves occurs, thus we only consider the case where curves remain fixed once they have been invoked by a mouse press. We also assume background motion has been subtracted from trajectory curves, and focus on the curvilinear dragging problem.

Requirements

There are different ways of mapping dragging gestures to curvilinear motions, i.e., moving a point along a 2D curve using a 2D input device. While the "correct" behavior for a curvilinear dragging method is subjective to users' expectations, we can postulate five basic requirements:

Multi-Scale: users should be able to perform both slow/fine dragging as well as fast/coarse dragging. If a curve has small high-frequency and large low-frequency components, the user should be able to explore it locally, but also rapidly transition to other regions of the curve (Figure 6a).

Arc-Length Continuity: it is desirable to favor continuity in terms of arc-length variation, e.g., if the user follows a loop and goes through its intersection, the algorithm should not jump to a different part of the curve (Figure 6b).

Directional Continuity: variations that preserve the arc-length direction of motion are favorable, e.g., users following a cusp should not go back to where they came from (Figure 6c). This allows for navigating through the whole curve with a single gesture, even if the curve has U-turns.

Proximity: a curvilinear dragging method should minimize spatial indirection [6], e.g., when the pointer is still, the offset between its current position and the corresponding point on the curve should be minimized.

Responsiveness: the method should also minimize temporal indirection [6], i.e., pointer motions should rapidly reflect on the curve, without noticeable delays in the interaction.

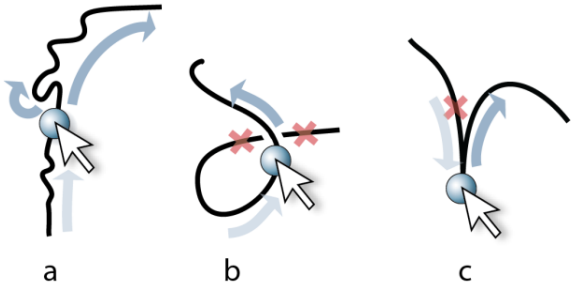

Figure 6: Three requirements for a curvilinear dragging technique: (a) multi-scale navigation, (b) arc-length continuity, and (c) directional continuity.

Existing Solutions

Current applications use simple solutions to the problem of curvilinear dragging. While straightforward, these solutions do not fully satisfy our proposed set of requirements. Projecting the pointer's location onto a line, for example, is effective but only applies to linear trajectories, e.g., sliders.

Another approach involves constraining the pointer's motion to a tunnel around the curve, i.e., a steering [1] solution. This steering method supports *fine* dragging and, to some extent, *arc-length continuity* and *responsiveness*. However, because users cannot deviate from the tunnel, support for *proximity* and *coarse* exploration is limited.

Using a "closest point" algorithm for curvilinear dragging is another simple solution [3]. This method meets the *proximity* and *responsiveness* criteria; however it does not meet the *arc-length continuity* and *directional continuity* criteria. At the same time, while this method supports *coarse* dragging, it does not allow for *fine* dragging on curves with small high-frequency components.

A better approach is to restrict the search for the closest point to a small neighborhood of the previous closest point [4]. This method supports *arc-length continuity* and *fine* dragging. However, support for *coarse* dragging is limited by the fact that the dragged point can remain stuck into local minima such as the left loop in Figure 6a.

We suggest a different extension of the closest point technique and show how it meets our proposed requirements.

The 3D Distance Method

We build on the closest point algorithm in order to benefit from its high *responsiveness*, its ability to enforce *proximity* and support for *coarse exploration*. We enforce continuity by taking arc-length distance into account when searching for the closest point. This is similar to Kimber et al's method [18], which however uses key frame distance and is hence sensitive to the way the curve is sampled.

Algorithm

The 3D distance method consists of expressing the curve in (*x, y, z*) coordinates instead of *(x, y)*. We do this by mapping a point's *z*-coordinate to its arc-length distance from the curve's origin. This mapping takes the form of a linear function $p_z = k \cdot arclen(p)$, where $k \geq 0$ is a scale factor. The curve's *x* and *y* coordinates are left unchanged.

The pointer's coordinates are also expressed in 3D space with *x, y* unchanged and *z* mapped to the *z*-coordinate of the currently active (dragged) point on the curve.

If C_a is the location of the currently active point on the curve, the 3D distance between the pointer p and any point C on the curve is obtained by the following equation:

$$D = \sqrt{(p_x - C_x)^2 + (p_y - C_y)^2 + (k \cdot \overline{C_a C})^2} \quad (1)$$

where p_x and p_y are the coordinates of the pointer p on the screen, C_x and C_y are the coordinates of the point C on the 2D curve, and $\overline{C_a C}$ is the arc-length distance between C_a and C on the 2D curve. Notice that this algorithm reduces to the standard 2D closest point when $k = 0$.

The initial active point C_a is obtained using a standard 2D closest-point search. Then, on each drag event, the new C_a is the point C which minimizes equation (1).

Jumps

Although our 3D distance algorithm preserves continuity at intersection neighborhoods, a jump will occur if users "insist" on dragging away from the current region. The jumping threshold depends on the value of the *z*-scaling constant *k*. $k \approx 1$ yields good results for video navigation and allows for both local dragging gestures that follow a curve loosely and ballistic motions between distant points of a curve.

Since the 3D distance method tries to preserve continuity, jumps occur less frequently than when using a closest point approach. However, jumps that occur are naturally larger. This is a result of the combined support for arc-length continuity and coarse exploration. The fact that jumps are both larger and more difficult to produce yields a natural interaction style by which local dragging can be bypassed using "pull gestures". Large jumps can be smoothed visually using animations, provided that these animations are fast enough to meet the responsiveness criterion.

Adding Support for Directional Continuity

As stated, the 3D distance method addresses the problem of arc-length continuity but not directional continuity. We add this support by adding a term $k_D > 0$ to the right member of equation (1) whenever $\overline{C_a C}$ and $\overline{C_{a^{(t-1)}} C_a}$ have opposite

signs. This will move the already visited region further away when searching for the closest point and thus preserve directional continuity on cusps. As a result, it will be slightly more difficult to reverse the arc-length direction of motion on the curve, k_D being the travel distance required to go back ($k_D \approx 5$ yields good results for video navigation). Kimber et al [18] use a similar scheme except that k_D and k decrease with time when no drag event occurs.

Maintaining Interactive Rates

The 3D distance method requires computing the distance between P and each of the curve segments. When curves have a large number of vertices, an optimization technique is desirable to ensure interactive system response.

Although we could have used a data structure to optimize our search [21], our distance metric allows a simpler approach. Since the absolute value of the z-component of our distance metric $d_z = |k \cdot \overrightarrow{C_a C}|$ increases monotonically as the candidate point C moves away from C_a, it can be used as a lower bound on the total distance between C and C_a.

This leads to the following algorithm: we search forward and backward along the curve beginning from the initial active point C_a. Each branch of the search halts when the distance d_z is greater than the candidate minimum distance or when the end of the curve is reached.

Curve Filtering

Curves can contain very small high-frequency components that are above the display capabilities and/or the pointing accuracy of the input device. In the case of video trajectory curves, noise can result from motion estimation artifacts, such as feature mismatches. Removing these components will likely improve curvilinear manipulations.

We use Taubin's smoothing algorithm [36] which behaves like a low-pass filter and does not significantly modify curves. This efficiently removes variations too small to be followed, while preserving most of the curve's features, thus still agreeing with the proximity criterion.

Limits of Curvilinear Dragging

The 3D distance method nicely supports curvilinear dragging on moderately complex curves. However, there is a limit to that complexity. For example, we processed a 5 minute uncut video of a couple dancing tango and obtained cluttered curves going back-and-forth many times on the screen. Such curves are difficult to visualize clearly and are naturally difficult to follow. We are currently looking into overcoming this issue by clipping very long curves to a neighborhood of predefined arc-length.

USER STUDY

We argue that, in addition to the feel of direct manipulation it can provide, relative flow dragging can outperform the traditional seeker bar in media browsing tasks that involve targeting in the image space. Targeting in the *image space* means that the user wants to reach a specific visual event or a frame having specific visual characteristics. This is in contrast to targeting in the *time space*, in which one wants to, e.g., reach a particular point in time within the video. We test this hypothesis on DimP through a study that presents users with video navigation tasks requiring them to think in terms of space instead of time.

Apparatus & Participants

We used a Dell Precision system running windows XP at 3.2GHz, 2 GB of RAM, a 1280×1024 LCD display, and a mouse. Six males and ten females, 18-44 years old, participated in the study and were recruited through e-mail posting at our university. No compensation was provided.

Task and Stimuli

The study used synthetic videos made by screen-capturing objects moving on a 2D canvas. The videos were 227 to 917 frames long and had a 640×480 resolution. We presented users with two types of task, *ladybug* and *car*:

Ladybug (Figure 7, left): This task involved video clips showing a ladybug flying over four markers, in an unpredictable order with non-uniform speed. Users were instructed to *"find the moment in the video when the ladybug passes over marker X"*.

Car (Figure 7, right): This task involved video clips showing three cars of different shape and colors moving one after the other, at unpredictable times. Each car moved only once and most of the video contained no movement. Users were instructed to *"find the moment in the video when car X starts moving"*.

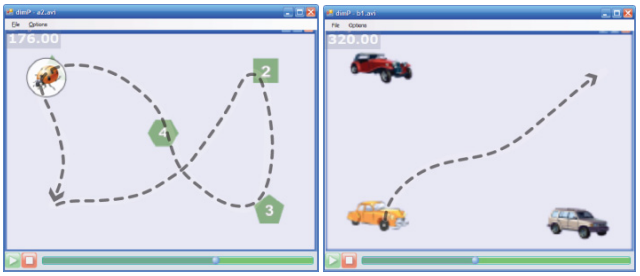

Figure 7: Examples of the *ladybug* (left) and *car* tasks (right). The dashed curves illustrate the objects' trajectory on the video and did not appear like this in the trials.

At the beginning of each trial, the video player (DimP) loaded a new video clip into memory then paused on the first frame. The player included a standard seeker bar of 554×16 pixels and also supported the direct manipulation technique described in the previous sections. All videos have been pre-processed for the latter technique. VCR buttons were also available but they were not needed for the tasks and none of the participants used them.

Procedure and Design

We used a 2 *technique* (*seeker bar, relative flow dragging*) × 2 *task* (*ladybug, car*) within-participant design. We presented three instances of each task that involved a different variant of the same video and a different *target frame*, i.e., frame to reach in the video stream. Target frames were chosen before, during and after the halfway-point of the video's length. They were given to users in a non-explicit way, i.e., through the description of visual events defined by either a marker number (*ladybug*) or a car color (*car*).

The dependent variables were *trial time* and *error*. We computed trial time as the time between the first *Mouse-Down* and the last *Mouse-Up* events of a trial. We computed error as the absolute difference between the number of the target frame and the number of the frame reached by the user divided by the total number of frames. The conditions' order of presentation was counter-balanced across participants. In summary, the experiment consisted of:

16 users × 2 techniques × 2 tasks × 3 instances = 192 trials.

Prior to the study, the experimenter explained the task to the users. Before each technique, users practiced with it using a warm-up video different from those used in the trials. Users were told to do the trials as quickly and accurately as possible. Users advanced to the next trial after declaring a task was completed.

Quantitative Results

The study averaged half an hour per user. We conducted a 2 (*technique*) × 2 (*task*) repeated measures analysis of variance (RM-ANOVA) on the logarithmically transformed trial time and on the error. The logarithm transform corrects for the skewing present in human time response data, and removes the influence of outliers.

We found a main effect for *technique* on time: relative flow dragging was at least 250% faster than using the seeker bar ($F_{1,12}$=69.762, p<0.0001) (Figure **8**).

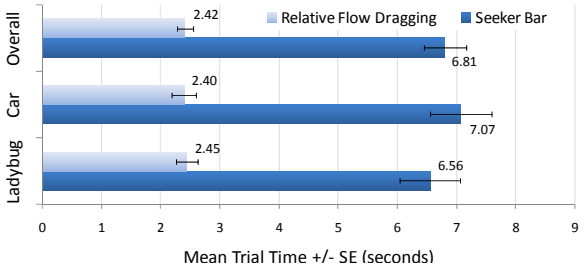

Figure 8: Mean time per task

Users were accurate with both techniques (<1% error). Although they seemed to be more precise with relative flow dragging than the seeker bar, this difference was not statistically significant ($F_{1,12}$=2.848, p=0.117). There was a main effect for *task* on *error* ($F_{1,12}$=13.982, p<0.005) (Figure 9). This result suggests that while conceptually similar, the two tasks demanded different resources from users.

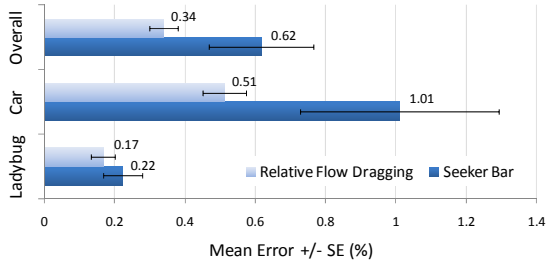

Figure 9: Mean error per task.

Qualitative Results

We asked users to rank their preferences using a Likert scale ranging from 1 (strongly disagree/very dissatisfied) to 7 (strongly agree/very satisfied). Users' preference for relative flow dragging over the seeker bar is consistent with our quantitative results (Table 1).

Users voiced their preference for relative flow dragging: it was "clean, easy to use", produced "immediate results with a high level of precision", "allowed me to interact with the video elements I was interested in, not just video as a whole". A user of 3D systems thought that "for visual constraints this technique is incredibly helpful". One user expressed some disorientation: "it was easier" but "confusing when the object did not move along the same path as the cursor". This expectation was probably reinforced by the synthetic look of the videos.

Technique	Question	Score	Standard Error
Relative Flow Dragging	Tasks were easy	6.38	0.24
	Tasks were fast	6.56	0.16
	Satisfaction	6.13	0.26
Seeker Bar	Tasks were easy	5.06	0.36
	Tasks were fast	4.31	0.42
	Satisfaction	4.69	0.33

Table 1: Summary of qualitative results.

PREVIOUS WORK ON VIDEO BROWSING

Despite advances in computing hardware, watching a video using regular software remains similar to the way we experience videos on analog video cassette recorders (VCR) [18]. Software video players mostly use traditional VCR controls, enabling playback at different rates. The only significant advance is the seeker bar, allowing partial random access and continuous navigation in the video timeline. Other innovations in browsing include non-linear navigation, visual summaries, content-based video retrieval, and advanced widgets.

Non-Linear Video Browsing

Videos often contain meaningful events, some of which can be extracted automatically, e.g., scene cuts, to support intelligent skip mechanisms [18]. Videos also have segments of different importance, and static scenes that can be detected and used to speed-up video playback [18, 38]. Different levels of importance can be inferred by estimating motion activity. Such information can be used to support adaptive fast-forward, i.e., changing the playback rate so as to maintain a constant "visual pace" [27, 38].

Adaptive fast-forward approaches are related to our technique because they use actual motions in the image space to facilitate video browsing. However, they show their limits in the presence of concurrent motions. For example, if a video has objects moving at very different speeds (e.g., cars and pedestrians), it is not clear which of them should be taken into account. Relative flow dragging addresses this by allowing the user to specify the motion of interest: clicking on a slow pedestrian will provide coarse grain access to a big time segment, whereas clicking on a fast car will provide finer access to a smaller time segment.

Visual Summaries

Combining video content analysis with visualization has been a popular method for supporting searching tasks in long videos. A representative approach for this method

consists of extracting relevant key-frames and organizing them into mosaics or interactive storyboards [38]. Key-frames can be also laid out on the seeker bar to provide an overview of the video [29].

A recent system from Goldman et al. [12] generates schematic storyboards to facilitate video editing tasks. Schematic arrows are generated from the motion of objects of interest specified by the user. These arrows act like sliders, allowing navigation through the video timeline. While this system shares similar ideas with ours, it does not support true direct manipulation, i.e., the video is manipulated through a static storyboard displayed in a separate window.

Content-Based Video Retrieval

Several research efforts have explored the use of images and visual trajectories for indexing and searching video [31, 35]. While some of these tools provide sketch-based GUIs, they all use a conversational interaction paradigm: the user makes a query, and then waits for the results. This is significantly different from our direct manipulation approach, which allows local browsing and exploration.

Advanced Widgets

Video players could also be improved by using sliders that support simultaneous control of position and velocity [29] or position and accuracy [2, 22]. This would increase the seeker bar's resolution, a common limitation of current video players.

In-place widgets can also be used to facilitate video browsing. For example, we could design pop-up seeker bars that adapt to the instantaneous motion of objects of interest. Such widgets would not require the user to follow trajectories but would share some advantages with relative flow dragging, such as its small degree of spatial indirection.

Although such approaches are worth exploring as an alternative to the seeker bar, we believe that only direct manipulation can support fast targeting in the image space, such as bringing a moving object to a location of interest. We intend to assess this claim in future user studies.

Similar Systems

The idea of annotating video clips to support direct manipulation has been previously used in the Dragri multimedia authoring system [25]. A fully automated solution by Kimber et al. has also been recently published and brought to our attention during the review process [18]. These two parallel attempts at solving a similar problem undoubtedly led to some very similar solutions, although significant differences in the motion extraction and curvilinear dragging algorithms have been outlined throughout this paper.

Kimber and al.'s work also mainly focuses on surveillance systems and explores ideas such as tracking across multiple calibrated cameras and manipulating floor plan projections [18]. In contrast, we elaborate on a general technical and conceptual framework for supporting direct manipulation of moving imagery. We also describe the issue of induced motion and show how it can be addressed by background stabilization. Finally, we provide a first assessment of the usability of the technique by the means of a user study.

CONCLUSION AND FUTURE WORK

We presented a new way of browsing videos, which brings the benefits of direct manipulation to an activity previously experienced through indirect means. Commercial media players could potentially benefit from our approach by exploiting motion metadata present in video files [35, 39]. This is especially appealing with the emergence of touch-input handheld multimedia devices.

In addition to the potential benefits to the overall subjective user experience, we showed how direct manipulation can improve user performance on space-centric video navigation tasks. The videos we used in our evaluation have been chosen for illustrative purposes and a more ecological comparison of video browsing techniques is still needed. In particular, it would be beneficial to assess the relative occurrence of space-centric video browsing tasks, as well as the applicability of flow dragging to long video clips.

Our contributions go beyond the implementation of an interactive system and address research challenges such as identifying new classes of direct manipulation techniques, designing a reusable curvilinear dragging method that meets a number of desirable properties, and adding to our understanding of the concept of direct manipulation.

By introducing the concept of relative flow dragging, we also suggest a general mechanism that can bring a new type of interactivity to a variety of graphical applications in which interaction is traditionally mediated by widgets. These include information visualization tools, interactive learning environments [14], GUI revisitation [8], as well as 2D and 3D animation authoring tools [3, 23] where clicking and dragging objects on the canvas can be sometimes more practical than "scrubbing" a timeline.

ACKNOWLEDGEMENTS

Thanks to Patricio Simari, Digby Elliott, Tomer Moscovich, Shahzad Malik, Nigel Morris, Michel Beaudouin-Lafon and members of the dgp lab for their helpful comments. Videos courtesy of www.bskunion.at, PETS2000, Christina J. Hodge and Ambers Christians.

REFERENCES

1. Accot, J. and Zhai, S. (1997). Beyond Fitts' law: models for trajectory-based HCI tasks. *CHI*. p. 295-302.
2. Appert, C. and Fekete, J. (2006). OrthoZoom scroller: 1D Multi-Scale Navigation. *CHI*. P. 21-30.
3. Autodesk Maya. http://www.autodesk.com/
4. Baudel, T., Fitzmaurice, G., Buxton, W., Kurtenbach, G., Tappen, C. and Liepa, P. (2002). *Drawing system using design guides.* US Patent # 6,377,240.
5. Beauchemin, S.S. and Barron, J.L. (1995). The computation of optical flow. *ACM Computing Surveys*, 27(3). p. 433-467.
6. Beaudouin-Lafon, M. (2000). Instrumental Interaction: An interaction model for designing post-WIMP user interfaces. *CHI*. p. 446-453.
7. Beaudouin-Lafon, M. (2001). Novel interaction techniques for overlapping windows. *UIST. p.* 153-154.

8. Bezerianos, A., Dragicevic, P. and Balakrishnan, R. (2006). Mnemonic rendering: an image-based approach for exposing hidden changes in dynamic displays. *UIST*. p. 159-168.
9. Buxton, W. (1986). There's more to interaction than meets the eye: some issues in manual input. In *User Centered System Design: New Perspectives on Human-Computer Interaction.* Lawrence Erlbaum. p. 19-337.
10. Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8). p. 790-799.
11. Dragicevic, P., Huot, S. and Huot, S. (2002). SpiraClock: a continuous and non-intrusive display for upcoming events. *CHI Extended Abstracts*. p. 604-605.
12. Goldman, D.B., Curless, B., Salesin, D. and Seitz, S.M. (2006). Schematic storyboarding for video visualization and editing. *SIGGRAPH*. p. 862-871.
13. Guimbretière, F. (2000). FlowMenu: combining command, text, and data entry. *UIST*. p. 213-216.
14. Hölzl, R. (1996). How does 'dragging' affect the learning of geometry? *International Journal of Computers for Mathematical Learning*, 1(2). p. 169-187.
15. Hutchins, E.L., Hollan, J.D. and Norman, D.A. (1987). Direct manipulation interfaces. In *Human-Computer interaction: A Multidisciplinary Approach*. R. M. Baecker, Ed. Morgan Kaufmann. p. 468-470.
16. Irani, M., Anadan, P. and Hsu, H. (1995). Mosaic based representations of video sequences and their applications. *Intl. Conference on Computer Vision*. p. 605-611.
17. Kim, C. and Hwang, J. (2002). Fast and automatic video object segmentation and tracking for content-based applications. *IEEE Trans. Circuits and Systems for Video Technology*, 12. p. 122-129.
18. Kimber D., Dunnigan, T., Girgensohn, A., Shipman, F., Turner, T. and Yang, T. (2007). Trailblazing: Video playback control by direct object manipulation. *ICME*. p. 1015-1018.
19. Li, F.C., Gupta, A., Sanocki, E., He, L. and Rui, Y. (2000). Browsing digital video. *CHI*. p. 169-176.
20. Lowe, D.G. (2004), Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2). p. 91-110.
21. Maier, D., Hesser, J. and Männer, R. (2003). Fast and accurate closest point search on triangulated surfaces and its application to motion estimation. WSEAS Intl. Conference on Signal, Speech and Image Processing.
22. Moscovich, T., Hughes, J.F. (2004). Navigating Documents with the Virtual Scroll Ring. *UIST*. P. 57-60.
23. Ngo, T., Cutrell, D., Dana, J., Donald, B., Loeb, L. and Zhu, S. (2000). Accessible animation and customizable graphics via simplicial configuration modeling. *SIGGRAPH*. p. 403-410.
24. Nowozin, S. autopano-sift − Automatic panorama stitching package. http://user.cs.tu-berlin.de/~nowozin/autopano-sift/
25. NTT-AT Dragri. http://www.dragri-fran.com
26. Pack, C. and Mingolla E. (1998). Global induced motion and visual stability in an optic flow illusion. *Vision Research,* 38. p. 3083-3093.
27. Peker, K.A., Divakaran, A. Sun, H. (2001). Constant pace skimming and temporal sub-sampling of video using motion activity. *IEEE International Conference on Image Processing*, Vol. 3. p. 414-417.
28. Proteau, L. and Masson, G. (1997). Visual perception modifies goal-directed movement control: Supporting evidence from a visual perturbation paradigm. *The Quarterly Journal of Experimental Psychology*, 50, 726-741.
29. Ramos, G. and Balakrishnan, R. (2003). Fluid interaction techniques for the control and annotation of digital video. *UIST*. p. 105-114.
30. Schneiderman, B. (1992). Designing the user interface: Effective strategies for effective human-computer interaction. Addison-Wesley.
31. Shim, C. and Chang, J. (2004). Trajectory-based video retrieval for multimedia information systems. *Proc. ADVIS, LNCS 3261*. p. 372-382.
32. Shoemake, K. (1992). ARCBALL: a user interface for specifying three-dimensional orientation using a mouse. *Graphics Interface*. p. 151-156.
33. Sinha, S., Frahm, J.M. and Pollefeys M. (2006). GPU-based video feature tracking and matching. *Tech. Rep. TR06-012, University of North Carolina at Chapel Hill.*
34. Snavely, N., Seitz, S.M. and Szeliski, R. (2006). Photo tourism: exploring photo collections in 3D. *ACM Transactions on Graphics,* 25(3). p. 835-846.
35. Su, C., Liao, I.M. and Fan, K. (2005). A motion-flow-based fast video retrieval system. *ACM SIGMM International Workshop on Multimedia Information Retrieval*. p. 105-112.
36. Taubin, G. (1995). Curve and surface smoothing without shrinkage. *Intl. Conf. on Comp. Vision.* p. 852.
37. Thorne, M., Burke, D. and van de Panne, M. (2004). Motion doodles: an interface for sketching character motion. *SIGGRAPH.* p. 424-431.
38. Truong, B.T. and Venkatesh, S. (2007). Video abstraction: A systematic review and classification. *ACM Transactions on Multimedia Computing, Communications, and Applications,* 3(1). p. 1-37.
39. Wei, J. (2003). An efficient motion estimation,method for MPEG-4 video encoder. *IEEE Transactions on Consumer Electronics,* 49(2). p. 441-446.
40. Yilmaz, A., Javed, O. and Shah, M. (2006). Object tracking: A survey. *ACM Computing Surveys,* 38(4). p. 1-45.
41. Zivotofsky, A.Z. (2004). The Duncker illusion: intersubject variability, brief exposure, and the role of eye movements in its generation. *Investigative Ophthalmology and Visual Science*, 45. p. 2867–2872.