Video Browsing by Direct Manipulation

Pierre Dragicevic, Gonzalo Ramos, Jacobo Bibliowicz,
Derek Nowrouzezahrai, Ravin Balakrishnan, Karan Singh
Department of Computer Science
University of Toronto
www.dgp.toronto.edu
{dragice, bonzo, jacky, derek, ravin, karan} @ dgp.toronto.edu

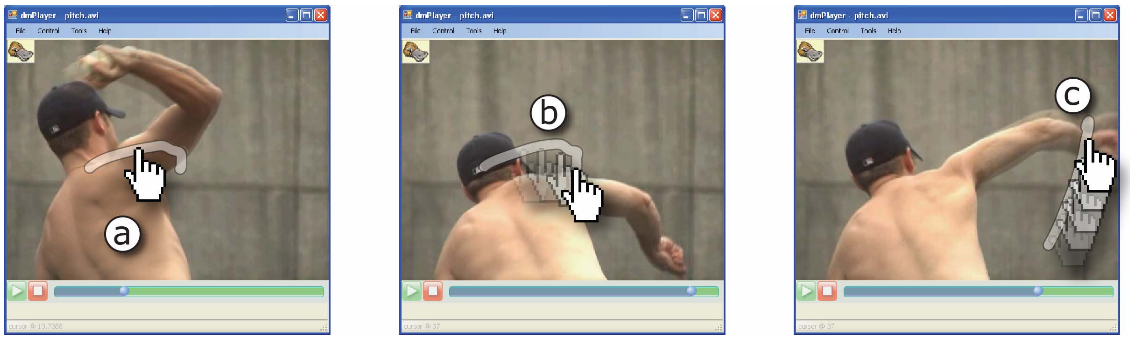

Figure 1. The Direct Manipulation Video Player used to scrutinize a baseball pitcher's motion: (a) pressing on the pitcher's shoulder displays its estimated trajectory ; (b) the user moves forward in the timeline by dragging the shoulder; (c) the user slowly goes back by dragging the pitcher's wrist. The size of the pointer has been exaggerated.

ABSTRACT
We present a novel method for browsing videos by directly dragging video content. This method brings the benefits of direct manipulation to an activity which is typically experienced via an indirect linear time slider. We show how direct manipulation can be supported in a video player by: 1) automatically extracting motions from video files using well-established computer vision methods; 2) using a technique called *relative flow dragging* that enables directly controlling the playback of these motions within the timeline of the motion rather than the overall video timeline.

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

General terms: Algorithms, Design, Human Factors

Keywords: video browsing, direct manipulation.

INTRODUCTION

Despite the widespread adoption and clear benefits [4, 13, 25] of direct manipulation, it has not been adopted consistently across all computer applications. There are still a number of tasks, such as browsing video, that could benefit from better application of direct manipulation principles. Most video players include a seeker bar that provides linear access to the video timeline. While this is a useful tool for interacting with videos, like many GUI widgets, it is no more than an intermediary between a user and the true object to be manipulated [4]. Often this object is in the video image itself, and it might be beneficial for users to be able to directly manipulate it. For example, a coach analyzing limb motions of a baseball pitcher might occasionally want to directly drag the pitcher's body as an alternative way to travel through the video timeline (Figure 1).

Video browsing through direct manipulation can provide the user with a fulfilling sense of control, similar to what they experience when controlling a video game character [13]. It can also facilitate tasks that focus on local and accurate browsing. To enable direct manipulation in a video, each pixel of a frame would have a corresponding "local seeker" that provides control over the object's motion. This control is achieved through a one-to-one mapping between the stimulus and the response [7], which can significantly facilitate tasks occurring in image space.

Along with its potential benefits, direct manipulation of video content presents several interesting research challenges. Firstly, one needs be able to infer or detect the underlying motion paths that are present in a video. There are numerous approaches one could use to address this issue and at first glance it is not obvious which is best suited to the task. Secondly, dragging video content is fundamentally different from dragging traditional GUI entities since the content to be manipulated is constrained to motions determined by that content itself rather than a predefined widget. Furthermore, these motions can be more complex than those of traditional interface widgets. Video scenes can involve non-rigid deformations and may include moving backgrounds which, in turn, affects the way a user perceives object motion. In this paper, we explore solutions to these challenges.

RELATIVE FLOW DRAGGING

Our video navigation interface uses a new technique we call *relative flow dragging*. Relative flow dragging is a general direct manipulation solution to the control of arbitrary motions that are limited to one degree of freedom. It decomposes complex motions into individual trajectories in a way that accounts for the phenomenon of induced motion (i.e., illusions of motion produced by dynamic backgrounds). We present the idea of relative flow dragging in a general context, describe how it relates to the concept of direct manipulation and illustrate its benefits.

Maximizing Directness

Hutchins et al. [13] proposed that *directness* depends on several factors, such as how *responsive* and *unobtrusive* a system is, and how closely the input language of the user interface *matches* its output language. For example, dragging an object is highly direct because the output it generates (an object moving on the screen) is very similar to the input (the user's hand moving).

Michel Beaudouin-Lafon [4] 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 the action and the response (low *degree of compatibility*). For example, panning with scrollbars suffers from these two types of indirectness, as opposed to directly dragging the document. A third type of indirectness involves tasks that require more degrees of freedom than provided by the user's input (low *degree of integration*).

Matching Gestures with Motions

Designing an interaction technique in general involves deciding how the user's *input* will be interpreted into changes in the *system*. Direct manipulation teaches us that such a decision should be made according to the characteristics of the system's *outputs*, rather than the internals of the system. For example, consider an internal variable whose variations cause a motion on screen (e.g., time causes a clock's hand to rotate). Regardless of the nature of this variable, the most direct visual way for the user to control it is by specifying the expected motion (e.g., dragging the clock's hand).

We propose 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*. For example, using a 2D device to pan a map. However, incompatibilities between motion spaces and gesture spaces are common. Adjusting the input device to the task [7] is an effective but often impractical solution. As a result, a number of techniques have been designed to map a limited 2D gestural language to a wide range of visual motions while preserving an illusion of direct manipulation. Examples are scaling objects, rotating objects [4] and performing 3D manipulations [27] using a mouse.

The control of high-dimensional motion spaces has long been recognized as an important problem, especially in the field of 3D interaction. In comparison, little attention has been paid to the control of low-dimensional spaces. *Relative flow dragging*, our general solution to the control of motions with one degree of freedom, is related to three other families of direct manipulation techniques: *curvilinear dragging*, *flow dragging* and *relative dragging*.

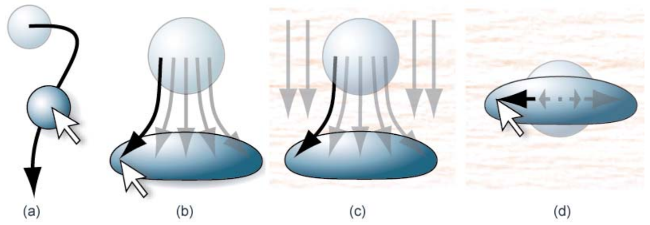

Figure 2: Forms of constrained direct manipulation techniques: (a) *curvilinear dragging*; (b) *flow dragging*; (c) flow dragging with a moving background; (d) *relative flow dragging* takes perceived motion into account.

Curvilinear Dragging

Curvilinear dragging consists of moving a point constrained along a 2D curve using a 2D pointing device (Figure 2a). Examples of curvilinear dragging techniques can be found in current user interfaces, although they mostly involve straight lines. For example, scrollbars, sliders and menus project 2D mouse movements onto a 1D axis [4]. Cascading menus and flow menus are not direct manipulation techniques per se, but involve steering, an action similar to that of following a curve [1, 11].

Examples of curvilinear dragging on arbitrary curves also exist. For example, the 3D modeling software Maya allows users to move contact points along 3D curves [2]. The Cabri-Géomètre mathematics learning environment [12] allows the exploration of geometric figures by dragging construction points that can be constrained to curves. Ngo et al.'s [19] system allows users to animate parameterized graphics by dragging objects along their trajectories.

Flow Dragging

Flow dragging is a generalization of curvilinear dragging. It involves direct manipulation of *continuous arbitrary motions* having only *one degree of freedom*. The term "arbitrary" means that the motions are not limited to translations, but can involve any type of visual transformation or deformation, such as a bouncing rubber ball. The term "one degree of freedom" does not mean that the motion occurs in 1D space, but that it can be parameterized with a unique variable, such as time.

Implementing flow dragging amounts to implementing a *curvilinear dragging* technique on a family of curves: since the whole motion has only one degree of freedom, each point of the object has a unique trajectory (see Figure 2b). Depending on how flow dragging is implemented, this trajectory can be dynamically replaced by nearby trajectories or can remain the same throughout the drag operation.

Relative Dragging

We presented direct manipulation design as a problem of matching gestures with motions. However, our experience with video browsing suggests that gestures should be matched with *perceived* motion rather than real motion. The apparent motion of an object is heavily influenced by the motion of its surroundings, a phenomenon known as *induced motion,* or the Duncker illusion [36]. Research suggests that induced motion also affects motor tasks [23].

Although induced motion is a complex phenomenon [21], it suggests that people perceive relative rather than absolute motion. Direct manipulation techniques that focus on the control of relative motion will be called *relative dragging* techniques (Figure 2c,d). Simple examples of relative dragging techniques are "automatic scrolling" techniques. They involve combining pointer and background motion so that their relative motion matches the user's hand movement. A particular case is keeping the pointer or object of interest still and moving the background in the opposite direction. Such an approach is used in Microsoft's accessibility magnifier to give the user access to a larger workspace. It is also very common in 2D action games.

Flow dragging is best implemented using a *relative dragging* approach, because flows are likely to produce induced motion. For example, let us suppose that the deformation of the rubber ball occurs with a background motion (Figure 2c). Curvilinear dragging on actual trajectories might be difficult because they are very different from what the user actually perceives. Dragging is facilitated if background motion is subtracted from the actual trajectories (Figure 2d). We call this method *relative flow dragging*.

Challenges

The design and implementation of relative flow dragging faces two key challenges. The first difficulty is the extraction of visual trajectories, as well as the computation of relative motions. We describe how this can be done with acceptable accuracy on video content. The second issue is support for curvilinear dragging: although anecdotal examples of curvilinear dragging exist, they do not behave well on arbitrary curves. This problem will also be addressed.

A more general challenge of relative flow dragging is to understand the situations in which it might be beneficial. Although few examples exist, we believe that it deserves to be more widely supported. For example, consider an information visualization application supporting dynamic queries. Manipulating a slider will typically change the parameters of the query, which in turn moves the datapoints of a scatter-plot. Supporting relative flow dragging on the scatter-plot could nicely complement the slider technique.

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)[1]. In addition to providing all the standard controls of a traditional movie player (Figure 3b), DimP allows users to go back and forth through the video's timeline by directly manipulating the video content.

DimP supports automatic extraction of motion trajectories. When a video is loaded into DimP, it first checks if it is already accompanied with motion information. If it is not, this information is automatically extracted then saved as a separate file associated with the video. We believe that it is reasonable to expect motion information to be created and potentially authored off-line. The loaded video is fully uncompressed, so that we have immediate access to the bitmap corresponding to each frame.

Figures 1, 3 and 4 illustrate DimP in action. Via three scenarios, we illustrate how DimP provides a richer way to navigate the video stream 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. We might want to rewind to the point in the video where the car is in the process of parking. On a traditional video player we have to drag the seeker bar's handle until we visually detect the frame of interest. With DimP, we can click and drag 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 finishing parking (Figure 3b). From that point onwards (or backwards) we can use the traditional seeker bar to find out, for example, who is leaving the car (Figure 3c).

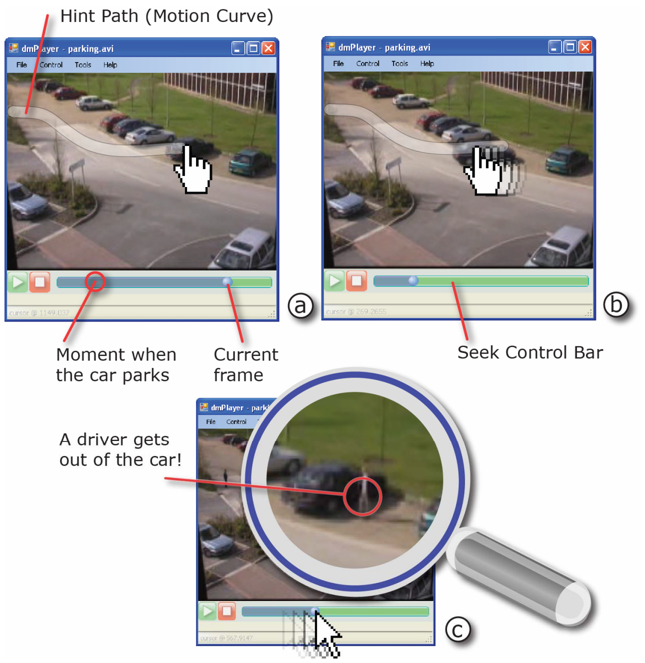

Figure 3: DimP can help find a particular event in a video. a) The user clicks on a parked car, its trajectory is shown. b) The user "detaches" the car from its parking location, which takes the player back to the point where the car was parking. c) The user browses through the temporal vicinity of this moment to find the person that is exiting the car.

[1] Dimp is available at www.dgp.toronto.edu/~dragice/dimp

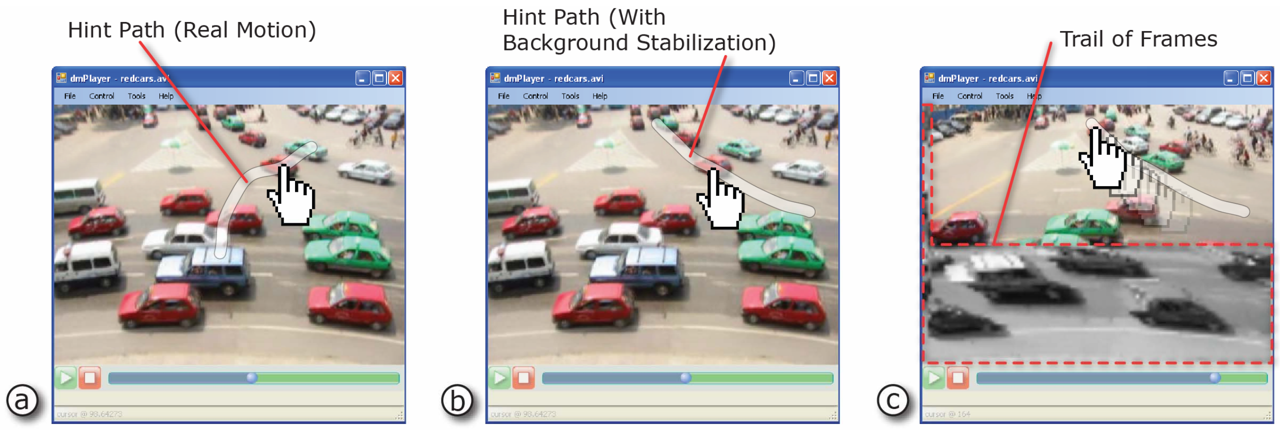

Figure 4: Background stabilization in Dimp, on a traffic scene involving a camera pan upwards. a) The actual motion of a car on the screen, hard to perceive by users. b) Background stabilization presents users with a motion path close to the car's perceived motion. c) When the car is dragged, background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

Scenario 2: a video of a scene on a busy street, where many cars and people are buzzing about, is being watched (Figure 4a). 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. 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 the style of an athlete pitching a baseball or a spring board diver performing a jump. Using direct manipulation, the coach has access to the many complicated movements involved in throwing a ball or twisting ones body in mid-air. We can imagine that the coach may switch his/her attention between different key points of interest in a scene, such as the point where a ball leaves the pitcher's hand, when a diver is about to hit the water or when a particular muscle group is engaged. All these operations can be performed by means of direct manipulation in the video view-port, without having to resort to the seeker bar (Figure 1). Hint paths are also useful in this scenario because motion trails are known to help animation and motion analysis [32].

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 away 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 curve is significant enough. Users can then engage in direct manipulation by simply clicking and dragging over a video's frame. As a user clicks over a region in the video frame where movement occurs, the system displays a subtle hint path corresponding to the estimated motion. Our designs are deliberately subtle and as visually unobtrusive as possible. The video frame remains fully visible while at the same time we reveal the general motion occurring under the point the user clicked on (Figures 1, 3, 4).

Emphasizing

Since objects in a video scene follow prescribed trajectories, users cannot drag them on arbitrary paths. We show this potential mismatch by emphasizing the hint path as the users' dragging path diverges from an object's motion. The farther the divergence, the more opaque the displayed hint path becomes (Figure 5). Finally, if a user drags the pointer past a predetermined threshold distance from the hint path, we play a "snapping" sound and the current instance of direct manipulation is terminated.

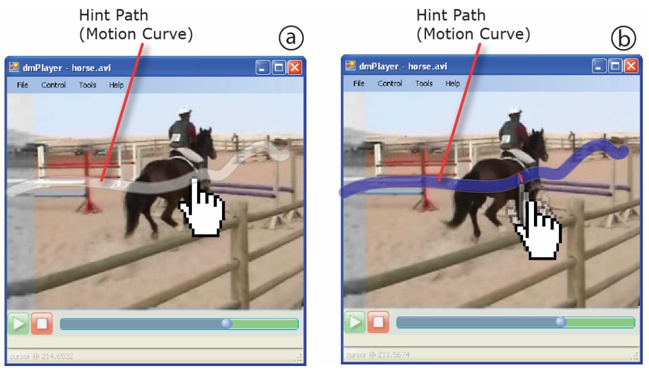

Figure 5: As the pointer moves away from the motion curve, the visual representation of the curve changes from (a) subtle to (b) emphasized.

Position Feedback

We display a red cursor over the hint path that indicates the position over the motion 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). The cursor additionally facilitates video navigation with sub-frame accuracy, which is useful in editing tasks involving videos with soundtracks.

Background Stabilization

Because of the phenomenon of induced motion, it is necessary to provide control over relative (perceived) motions rather than over real motions. Following a relative dragging approach, DimP computes and extracts background motion information and subtracts it from all the trajectories.

One approach to supporting relative dragging is to disengage the pointer's motion from the input device motion. For example, dragging an object might not move the object, yet it appears to move because of a moving background. In this case, moving the input device does not move the cursor, only the video. Such an approach is only possible with relative pointing devices such as a mouse.

Instead of the previous solution, we use background stabilization, an approach that works for both relative and absolute pointing devices. Figure 4 illustrates an example where background stabilization helps to match a motion curve to an object's perceived motion. People watching a video of cars going up a street will see cars going up the street even under the presence of upward camera panning motions. Figure 4a shows how an upwards camera pan can make a car's motion in the video a downwards one, even though the car moves up the street. 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.

Background stabilization shifts video frames within the player's window as a user drags a particular object. These shifts are intended to match an object's actual motion with the user's perceived motion. As video frames shift inside the player's window as a result of the stabilization process, they leave a trail of blurred grayscale images which aid in putting the currently displayed video frame in context with the video scene (Figure 4c). To a certain degree, this trail of images forms a structure close to a stitched panorama or video mosaic [10, 14]. Once the direct manipulation ends, the current frame springs back with a smooth animation 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 the motion information:

Manual Annotation: in order to collect initial user reactions, we first prototyped the idea of relative flow dragging in a Java prototype requiring manual annotation of videos. Points of interests were positioned and moved across frames. Manual annotation is a fairly reasonable approach in the context of multimedia authoring and can be complemented with automatic techniques. However, fully automatic solutions are needed for video consumers to be able to experience direct manipulation on arbitrary video files.

Metadata Extraction: video formats such as MPEG already contain motion information used for the purpose of decoding [30, 34]. We did not investigate this approach because we wanted to test relative flow dragging with reliable motions first, as a proof of concept. Still, this strategy is worth exploring in future work.

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 image sequences is a classical problem in computer vision with a number of applications. Given the considerable amount of research in this area [3, 15, 26, 28, 30, 34, 35], our goal is not to propose a new method, but rather to use well-established vision techniques in order to demonstrate the feasibility of built-in support for the direct manipulation of video.

We can arbitrarily group motion estimation problems into two categories: object tracking and optical flow.

Object Tracking: tracking involves following the motion of one or several objects of interest over a video sequence. Such objects are supposed to have stable salient features, which are sometimes known in advance. Applications include motion capture and video surveillance [35].

Optical Flow: optical flow computation consists of estimating pixel velocities between adjacent video frames, without enforcing temporal consistencies. Applications include video compression, 3D reconstruction and robot servo control [3].

Some problems, like ours, can be tackled using either of these approaches. Tracking algorithms are efficient at following objects over long periods of time, but they assume that such objects are known a priori. Automatically extracting them is a complex process [15]. Optical flow methods do not require such knowledge and are able to keep track of non-rigid motions. However, optical flow methods can have serious problems with temporal discontinuities such as occlusion between objects and scene cuts.

Given the tradeoffs between the two approaches, we believe that optical flow is worth considering as a first step, because it spares us from making any assumptions about the structure of the objects present in the video scenes. To achieve a minimal robustness, we opted for a solution in which we generate a feature flow between pairs of video frames.

Extraction of Feature Flows
Traditional optical flow estimation methods that use variations in pixel intensities to generate dense motion fields suffer from a number of difficulties [3]. When coarse motion estimation is sufficient, one can obtain more reliable results by using a feature-based optical flow estimation approach [17]. In a feature-based optical flow, one tracks the variations of feature points between frames in order to estimate pixel motion.

Feature detection is an image operation that detects optically interesting parts of an image, such as edges or corners. Robust features can be used to reliably match pairs of images, even when they differ in aspects such as illumination or point of view. We chose the SIFT approach (Scale-Invariant Feature Transform) [17]. SIFT is a robust feature extraction method already available in several libraries and has been used successfully in applications ranging from panorama creation tools [20], reconstruction of 3D scenes [29], and video tracking [28]. In addition to the above, SIFT libraries also provide methods for feature matching.

In DimP, we use Nowozin's panorama stitching library [20], which includes a C# implementation of the SIFT detection and matching algorithms [17]. We made minor modifications. For example, we increased the $|D(\hat{x})|$ threshold from 0.03 to 0.5 in order to reduce the detector's sensitivity to low-contrast video compression artifacts.

The feature flow between each pair of adjacent frames is computed by detecting and matching SIFT feature points between these two frames. Each pair of matched feature points gives a motion vector. Unmatched features are discarded. A feature flow is marked "unknown" if the total number of matched features is zero. Once feature flows are computed for the whole video, motion trajectories can be generated on-the-fly.

Construction of Motion trajectories
Optical flow is deduced from feature flow using nearest-neighbor interpolation. In other words, we assume that the displacement of a pixel is the displacement of the nearest feature point. This method is conservative, since local motions are typically tagged with few features (sometimes one or two) and their immediate surroundings are devoid of features.

The motion trajectory going through a given pixel of a given video frame is then built by adding up flows forward and backwards in time. There is very little cumulative error due to rounding because the location of SIFT feature points has sub-pixel accuracy. The process of building the motion trajectory stops as soon as an unknown flow is encountered. Apart from the first and last frame, this happens almost exclusively during scene cuts.

Finally, we tag each of the vertices on a trajectory curve with the frame number it corresponds to. Given any point on the curve, the corresponding floating-point video frame number is obtained by linear interpolation.

Extraction of Background Motion
The task of detecting the background or dominant motions of any video sequence can be formulated as a clustering problem. The goal is to segment feature motion vectors into clusters that map, preferably in a one-to-one manner, to coherent areas in screen-space and also potentially in time.

We implemented a simple, 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 has shown acceptable results from the user perspective and has the advantage of being computationally cheap. 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 [8].

Limits of the Method
The quality of our method, i.e., how well the generated trajectories match users' expectations, has been assessed through informal testing sessions. Our method yields very good results on high-quality videos involving large continuous motions. However, there remains some limitations:

Speed: a standard, non-optimized feature extraction and matching can be time consuming. While our implementation can take between five to ten seconds of processing time per frame on a regular computer, research suggests that feature detection and matching can be done faster, if not in real-time [17, 28]. We are planning to use such accelerated implementations in the future.

Sensitivity: in order to keep computation times acceptable and bounded, we sub-sample video frames to 128 x 128 grayscale images before processing. As a result, motions of small objects are not detected. Faster feature extraction implementations could operate on higher resolution images, thus improving sensitivity.

Discontinuous Motions: our implementation does not "remember" objects which have been briefly occluded or moved outside the camera field. The motion of a given object can get merged with another object's motion. Such issues are intrinsic to the feature flow paradigm, and can be partly addressed using tracking techniques [35].

Induced Motion: in addition to the imperfections of its background motion detection, our method makes several simplifying assumptions about induced motion: it does not detect multiple coherent motions, nor does it account for non-translational induced motions [21].

The motion 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 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 IMPLEMENTATION

Supporting flow dragging mostly requires supporting curvilinear dragging on multiple curves. One minor issue is the dynamic transition between trajectories; we will not address it and only consider the case where trajectories remain fixed once they have been invoked by a mouse press. When a relative flow dragging approach is necessary, we will additionally suppose that background motion has already been subtracted from trajectory curves.

Requirements and Goals

Curvilinear dragging consists of moving a point along a 2D curve using a 2D input device. While there are different ways of mapping dragging actions to curvilinear motions, and the "correct" behavior for a curvilinear dragging method is subjective based on users' opinions and expectations, it is possible to postulate a set of basic requirements:

Multi-Scale: users should be able to perform both slow/fine dragging as well as fast/coarse dragging. If a curve has high and 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. For example, if the user follows a loop and goes through a part in which the curve is intersecting itself, 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. For example, if the user follows a cusp, the algorithm should not go back to the previously explored region (Figure 6c). This allows for navigating through a whole curve with a single gesture, even when the curve involves U-turns.

Proximity: the method should follow direct manipulation principles by minimizing spatial indirection [4]. For example, when the pointer is still, the offset between its current position and the current point location on the curve should be minimized.

Responsiveness: according to direct manipulation principles, the method should also minimize temporal indirection [4]. This means that pointer motions should rapidly reflect on the curve, without noticeable delays in the interaction.

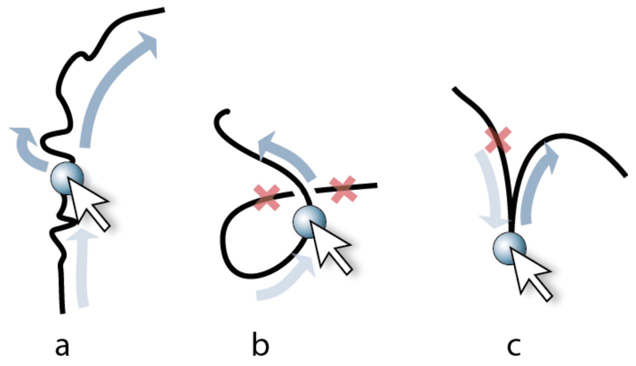

Figure 6: Three requirements for a curvilinear dragging technique: multi-scale navigation (a), arc-length continuity (b) and directional continuity (c).

Curvilinear Dragging Solutions

Current applications use simple solutions to the problem of curvilinear dragging. While algorithmically simple, these solutions do not fully satisfy the requirements we set for curvilinear dragging. Projecting the pointer's location onto a linear trajectory, for example, is an effective method of dragging a point on a line, but it can be only applied to straight motions, e.g., sliders.

One could build a dragging solution based on steering [1] by constraining the pointer's motion to a tunnel built around the curve. This steering method would support *fine dragging* and, to some extent, *arc-length continuity* and *responsiveness*. However, because users cannot deviate from the curve, support for *proximity* and *coarse exploration* is limited. *Directional continuity* is also unachievable.

Computing the closest point to the pointer's location is another solution for mapping pointer locations onto non-straight curves. The closest point approach meets the *proximity* and *responsiveness* criteria, however it does not meet the *arc-length continuity* and *directional continuity* criteria. While this method supports *coarse* dragging, it does not allow for smooth *fine* dragging on complex curves with high-frequency components. However, it is possible to slightly modify this solution and obtain a dragging method that meets our proposed requirements reasonably well.

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 its support for *coarse exploration*. We enforce continuity by taking arc-length distance into account when searching for the closest point.

The Method

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 coordinates 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 is a scale factor. The curve's x and y coordinates are left unchanged.

The pointer's coordinates are also expressed in a 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:

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

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 the 3D distance algorithm preserves continuity at intersection neighborhoods, a jump will occur if the user "insists" 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.

Because the 3D distance method tries to preserve continuity, jumps occur less frequently than when using a closest point solution. However, jumps that occur are discernibly larger. This is a natural 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". Very large jumps can be smoothed 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 introducing a small discontinuity in the distance function, by adding a term $k_D > 0$ in 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 go back (i.e., change 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.

Maintaining Interactive Rates

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

Although we could have used a data structure to optimize our search [18], we found a simpler approach based on our 3D formulation. Notice that the absolute value of the *z*-component of our distance metric $d_z = \left|k \cdot \overline{C_a C}\right|$ increases monotonically as the candidate point *C* moves away from C_a. Additionally, since it is only one component of the distance metric, d_z can be used as a lower bound on the total distance between *C* and C_a. Therefore, given a minimum candidate distance *D* to the curve, we can discard all points on the curve with $d_z > D$ from our closest point search.

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 local features that will likely never be followed by the user during curvilinear dragging. For example, it might contain small frequency components that are above the display capabilities and/or the pointing accuracy of the input device. In the case of video motion curves, noise can result from motion estimation artifacts, such as feature mismatches. Removing these components will likely improve curvilinear manipulations.

Due to the proximity criteria, filtering should not significantly modify the curves. We thus opted for a shrinkage-free smoothing algorithm, which behaves like a low-pass filter [31]. This efficiently removes variations too small to be followed, while preserving most of the curve's features.

PREVIOUS WORK ON VIDEO BROWSING

Video browsing by direct manipulation is related to several research areas. We previously discussed related work on direct manipulation and computer vision. We now compare our method with prior work on video browsing.

Despite increasing computing power, watching a video using regular software does not significantly differ from the way we used to watch videos on analog video cassette recorders (VCR) [16]. Video player's mostly use traditional VCR controls, enabling playback at different rates. The only significant difference is the seeker bar, allowing random access and continuous navigation in the video timeline. Other innovations in browsing include *non-linear navigation, visual summaries,* and *content-based* video retrieval.

Non-Linear Navigation

Video streams often contain meaningful events, some of which can be extracted automatically, e.g., scene cuts, to support intelligent skip mechanisms [16]. Videos also have segments of different importance, and static scenes can be detected and used to speed-up video playback [16, 33]. Different levels of "interestingness" 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" [22, 33].

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

Visual Summaries

Combining video content analysis with visualization has been a popular method for addressing the problem of searching in long videos. The most common approach consists of extracting a set of relevant key-frames and organizing them into mosaics or interactive storyboards [33]. Video key-frames can be also laid out on the seeker widget in order to provide an overview of the video [24]. This system also supports multi-scale navigation on the timeline with a pressure-sensitive stylus.

Recently, Goldman et al. [10] proposed a method for generating schematic storyboards, and described a technique that can be considered as a precursor of our system. Schematic arrows were generated from the motion of objects of interest, and included a slider that allows navigation into the video. However, the video was shown in a different window and was indirectly manipulated through the storyboard. Also, the system was targeted towards video editing tasks and required user assistance for generating the storyboard.

Content-Based Video Retrieval

A number of contributions in the field of multimedia databases explored the use of images and motion trajectories for indexing and searching video content [26, 30]. Although some of these tools include sketch-based GUIs, they all use a conversational interaction paradigm: the user first makes a query, then waits for the results. This is significantly different from our direct manipulation approach, which affords browsing and local exploration.

CONCLUSION AND FUTURE WORK

We presented an original method for browsing videos, which brings the benefits of direct manipulation to an activity which has previously typically been experienced through indirect interaction. In addition to the potential improvements to the user experience, we believe our approach can significantly facilitate targeting tasks in the image space. We plan to conduct user studies to confirm that hypothesis, and quantify the advantages and the limits of direct manipulation in a variety of video navigation tasks.

Browsing video by direct manipulation presents several research challenges that we addressed in this paper. One challenge was the automatic extraction of motion trajectories from videos. We showed how well-established computer vision tools allow extracting such information with enough accuracy for the purposes of interaction, at least for videos involving large and continuous motion.

We believe that commercial video players should soon be able to support direct manipulation by exploiting motion information already present in video files [30, 34]. Since the image view-port is currently unused for interaction, there is no real cost associated with such a transition. Bringing touch-sensitivity and direct manipulation to the home cinema might also change how people experience movies.

Browsing video by direct manipulation presents a number of challenges from the HCI point of view. Dragging video content is fundamentally different from dragging rigid GUI entities. We introduced the concept of *relative flow dragging* as a general solution to the manipulation of arbitrary motions constrained to one degree of freedom.

Supporting relative flow dragging requires solving the particular case of *curvilinear dragging*, for which no satisfactory method has been proposed so far. We described an algorithm that has a number of desirable properties. This algorithm has been tested on video navigation, but can be applied to any task involving objects dragged on predefined paths. Examples include geometry learning environments [12], 3D modeling software [2] and GUI widgets [1, 11].

Applications of the more general relative flow dragging paradigm are less commonplace, but video browsing is not the only scenario one can think about. Relative flow dragging can bring interactivity to a variety of applications, including information visualization systems, in which interaction is traditionally mediated by widgets. It can also be useful in computer animation [19], including within authoring software like Flash or Maya, where being able to click and drag objects might be more compelling than scrubbing a timeline when working with a partially authored or evolving 2D or 3D animation. Another application might be as an application agnostic GUI revisitation system [6].

They are several challenges related to relative flow dragging that we would like to address in the future. One of them is providing a better empirically-based support for relative dragging that accounts for non-translational induced motions [21]. Another is the generalization of flow dragging to multi-dimensional motions.

ACKNOWLEDGMENTS

Thanks to Patricio Simari, Digby Elliott, Tomer Moscovich, Shahzad Malik, Nigel Morris and members of the Dynamic Project Laboratory (www.dgp.toronto.edu) at the University of Toronto for their helpful comments.

REFERENCES

1. Accot, J. and Zhai, S. (1997). Beyond Fitts' law: models for trajectory-based HCI tasks. *CHI*. p. 295-302.
2. Autodesk Maya. http://www.autodesk.com/
3. Beauchemin, S.S. and Barron, J.L. (1995). The computation of optical flow. *ACM Computing Surveys*, 27(3). p. 433-467.
4. Beaudouin-Lafon, M. (2000). Instrumental Interaction: An interaction model for designing post-WIMP user interfaces. *CHI.* p. 446-453.
5. Beaudouin-Lafon, M. (2001). Novel interaction techniques for overlapping windows. *UIST. p.* 153-154.
6. 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.
7. 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.
8. Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8). p. 790-799.

9. Dragicevic, P., Huot, S. and Huot, S. (2002). SpiraClock: a continuous and non-intrusive display for upcoming events. *CHI Extended Abstracts*. p. 604-605.

10. Goldman, D.B., Curless, B., Salesin, D. and Seitz, S.M. (2006). Schematic storyboarding for video visualization and editing. *SIGGRAPH*. p. 862-871.

11. Guimbretière, F. (2000). FlowMenu: combining command, text, and data entry. *UIST.* p. 213-216.

12. Hölzl, R. (1996). How does 'dragging' affect the learning of geometry? *International Journal of Computers for Mathematical Learning*, 1(2). p. 169-187.

13. 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.

14. 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.

15. 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.

16. Li, F.C., Gupta, A., Sanocki, E., He, L. and Rui, Y. (2000). Browsing digital video. *CHI.* p. 169-176.

17. Lowe, D.G. (2004), Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2). p. 91-110.

18. 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.

19. 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.

20. Nowozin, S. autopano-sift − Automatic panorama stitching package. http://user.cs.tu-berlin.de/~nowozin/autopano-sift/

21. Pack, C. and Mingolla E. (1998). Global induced motion and visual stability in an optic flow illusion. *Vision Research,* 38. p. 3083-3093.

22. 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.

23. 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.

24. Ramos, G. and Balakrishnan, R. (2003). Fluid interaction techniques for the control and annotation of digital video. *UIST.* p. 105-114.

25. Schneiderman, B. (1992). Designing the user interface: Effective strategies for effective human-computer interaction. Addison-Wesley.

26. Shim, C. and Chang, J. (2004). Trajectory-based video retrieval for multimedia information systems. *Proc. ADVIS, LNCS 3261*. p. 372-382.

27. Shoemake, K. (1992). ARCBALL: a user interface for specifying three-dimensional orientation using a mouse. *Graphics Interface*. p. 151-156.

28. 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.*

29. 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.

30. 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.

31. Taubin, G. (1995). Curve and surface smoothing without shrinkage. *Intl. Conference on Computer Vision.* p. 852.

32. Thorne, M., Burke, D. and van de Panne, M. (2004). Motion doodles: an interface for sketching character motion. *SIGGRAPH.* p. 424-431.

33. 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.

34. Wei, J. (2003). An efficient motion estimation,method for MPEG-4 video encoder. *IEEE Transactions on Consumer Electronics,* 49(2). p. 441-446.

35. Yilmaz, A., Javed, O. and Shah, M. (2006). Object tracking: A survey. *ACM Computing Surveys,* 38(4). p. 1-45.

36. Zivotofsky, A.Z. (2004). The Duncker illusion: intersubject variability, brief exposure, and the role of eye movements in its generation. I*nvestigative Ophthalmology and Visual Science*, 45. p. 2867–2872.