Technical Report TR02-017

Department of Computer Science
Univ of North Carolina at Chapel Hill

An Extensible Object Tracking Architecture for Hyperlinking in Real-time and Stored Video Streams

Jason McC. Smith and David Stotts

Dept of Computer Science
Univ. of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175

stotts@cs.unc.edu

April 24, 2002

An Extensible Object Tracking Architecture for Hyperlinking in Real-time and Stored Video Streams

Jason McC. Smith and David Stotts

Dept. of Computer Science
Univ. of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175 USA

{smithja,stotts}@cs.unc.edu

ABSTRACT

Video is not as richly-used a component of hypermedia documents as text, due to the difficulty of automatically treating objects in video frames as identifiable, linkable content. Rather, video tends to be manipulated as frames of pixels with no further subdivisions, or tedious manual markup is used to outline intra-frame link anchors. We describe OvalTine, a system for tracking objects in video streams so that hypermedia link anchors can be associated with objects in the frames without manual markup. We show how OvalTine can be used to provide hyperlinked structure to real-time video streams (such as teleconferences), as well as to mark up stored video (digital libraries) with hyperlinks in automated fashion. We also discuss the software architecture of OvalTine, which is designed to allow multiple tracker algorithms to be composed dynamically, in response to the particulars of the images being tracked.

Keywords

Video, linking, hypermedia, markup, image analysis, digital library, teleconferencing, tracking

1. VIDEO IN HYPERMEDIA

While video is a major component of multimedia documents, it is less common in hypermedia documents due to the inability to treat video easily as a fully first-class data component. Links into video streams are difficult to do with the detailed targeting and semantic resolution of links in, say, text; this comes from the difficulty of treating the objects depicted in video as algorithmically identifiable, linkable content. Rather, video tends to be manipulated as sequences of pixel frames with no further subdivisions. When link markup is done on video streams, it is done manually frame-by-frame. If links are associated with sub-regions of a frame (object outlines, e.g.), the link anchor identification is done manually. This is a tedious process, and causes most hypermedia documents using video to limit video links to starting, stopping, and hopping into the middle of video streams, usually on a time basis.

We use the term hypervideo in this report to refer to a displayed video stream that contains embedded user-clickable anchors [7]. These anchors are logically attached to objects within the video environment, independent of location within the field of view. For instance, a person's face in a video conferencing system may be designated as a hyperlink. As the person moves within the image, the clickable area that activates that hyperlink will move with the image of the face, so that the face itself defines the active region, or *hot-spot*.

There is increasing interest in video data being incorporated into hyperdocuments. Digital libraries are growing in popularity and scope, and video is an important component of such archives. All major news services have vast video archives, valuable "footage" that would be of use in education, historical research, even entertainment. The current best practices for link markup in video require completely, or considerably, manual markup of the video frames with the active, or hot, areas that serve as link anchors in hypervideo data [9]. Broad hypermedia access to the vast stored video "footage" just mentioned will only be possible with automated link markup methods.

We present here an algorithmic (non-manual) approach to marking up video streams with hyperlink information at the sub-frame, or object, level. Our system is called OvalTine, and it allows the embedding of hyperlinks in both real-time video streams (*e.g.,* video teleconferencing) as well as stored video (*e.g.,* digital libraries, video databases). We do not specify to what a link anchor may refer, nor do we limit the type of objects that may be designated as a link anchor. The goal of our work is to explore the creation and maintenance of hyperlinks in video streams, and to automate these procedures as much as possible.

Outline of the Presentation

In the following sections, we first discuss the basic OvalTine system and its hyperlinking facilities for real-time video streams. We then discuss the link storage layer we run on top of the real-time trackers to allow capture of anchor tracking information and link annotation of stored video. We then discuss the software architecture of OvalTine that allows chaining together of different trackers to get improved results for different video environments. We conclude with a discussion of related previous research in hypervideo and video tracking.

MM 2002, Juan les Pin, France, December 1-6, 2002.

ACM 1-58113-477-0/02/0006.

2. REAL-TIME ANCHOR TRACKING

Our work on OvalTine began with the desire to create a hypermedia system in which video streams could be first-class data -- that is, data in which link anchors could be embedded and links followed, just as for text links in a Web page. The goal was to allow a viewer of a video stream to establish a link anchor on an object in the video frame, and as the object moved around the frame, the link anchor moved as well, tracking the object that represents it. The viewer could, at any time after creation of the link, click on the anchor and retrieve the linked information. In this way, an object in a video stream could be linked to text, to sound, even to another video stream or to an object in a video stream. A report on an early version of OvalTine discussed the various distributed system architecture issues involved in storing and serving video hyperlinks in a client/server implementation [1]. In terms of that taxonomy, our current work is a *Server/Archived* scenario, and we have chosen to implement a *Manual* object selection scheme for initiating a tracked object (hyperlink anchor).

Version 1 of OvalTine (OvalTine-RT) does real-time tracking of faces in live video streams, such as you might have in a real-time videoconferencing system. This tracking allows hypermedia link anchors to be associated with objects in the video window, creating a first-class hypermedia capability for video data. A face in the video frame becomes a live link, a selectable target for the user to click on to trigger some action. As the face moves around the video screen, the live target area moves with it, providing the illusion that the face itself is the hyperlink anchor. The target is optionally made visible by means of a simple highlighted oval that moves with the face in the frame. Links are URLs, associated with an object when it is first selected as a link anchor (at the initiation of tracking). A mouse click within the oval counts as a click on the object, and the enveloping hypermedia layer "follows" the link by launching a web browser with the URL loaded.

OvalTine-RT is a dynamic, anchor-creating video-conferencing application based on the *mediaConf* example application from Silicon Graphics for their O2 workstations [29]. *mediaConf* is based on SGI's *dmedia* libraries, which provide the basic framework for a server/client video conferencing system. This system is highly configurable, with forking of a video stream possible at any node, regardless of whether it is a video source, receiver, or relay. *mediaConf* includes a simple collaborative drawing environment which has drawing primitives such as lines, boxes, ovals, and a freeform pen.

To implement tracking of video collaborators' heads, we added a real-time video analysis engine developed by Birchfield at Stanford, which uses a mixed-mode analysis algorithm to search for ovals, such as human faces, in a video field [25,26]. This tracker works by combining an intensity gradient edge-detector with a color histogram region-matcher, giving superior performance in acquiring and tracking heads than either algorithm does alone. The combination allows for rotations, tilts, limited occlusions, and noisy backgrounds without losing focus[1]. Our research goal is not necessarily to push the envelope on video segmentation, but rather to make use of proven techniques in creating hyperlinking facilities in video; Birchfield's work was a good starting point and we adapted his implementation.

The found faces are denoted by ellipses sent across the network as drawing elements of *mediaConf*. These elements are further extended with the addition of hyperlink data. In our case, we chose simple ASCII URLs as the base case. When the user activates tracking of a hypervideo anchor, the corresponding URL information is merely captured using a text box within the GUI.

OvalTine-RT was developed on a pair of Silicon Graphics O2 workstations, running IRIX 6.3 OS, connected with 10bT Ethernet over the standard network within the Computer Science department at the UNC. No special configuring of the network was done, in order to simulate more closely a natural user environment. The video input was supplied from the standard O2 video camera through the integrated ICE video hardware grabbing board. The *mediaConf* software was modified to accept the addition of the *HeadTrackerLib* library. *HeadTrackerLib* was a substantial rewrite of the *headtracker* application provided by Birchfield *et al*. on the Windows platform. All Win32 specific code was removed and replaced, resulting in a cross-platform image-tracking library [30]. The *multip* C++ threading package from UNC's Graphics Lab was used to implement our image tracking and analysis computations in a separate thread.

Useage Example

A standard video conferencing session in OvalTine proceeds as follows. The same application binary is started on both machines. Each user registers his or her machine with the other through the UI, which simply asks for a TCP/IP accessible machine name. Each user then selects the other machine to be its video server, so each machine is seeing the other's camera view.

In order to generate anchors, each user selects the image of the other user using an enhanced variant of the supplied drawing tools. This triggers the building of an internal model of the image coming from the server. That model is then used to track the other person's image. The anchor region can be made visible, or not, depending on the user's preference. If it is visible, it appears as an oval overlaying the video image.

Each machine can also act as a server of anchors, by the user selecting his or her own camera's video stream, and operating on it. Only one video stream at a time can actively be seen and have anchors generated. Thus in OvalTine-RT, a machine can act as a server-side anchor generator, or a client-side anchor generator, but not both simultaneously.

Once an anchor is established in the video frame, a link to other data can be made (to text annotations, images, web pages, other video streams). Clicking on an anchor at any time that its video object is visible in the video frame, regardless of how it has moved since anchor creation, will retrieve the linked information as one expects in a hypermedia system.

OvalTine-RT application was designed to exercise several key points in real-time hypervideo: 1) server-side vs. client-side anchor definition, 2) anchor propagation across a network, 3) acceptable performance for real-time execution, 4) flexibility in linked materials, 5) manual and automated anchor creation.

[1] The new system design of OvalTine-SV (see section 4) supports arbitrary combinations of tracking methods, like Birchfield's approach, via it's tracker- chaining software architecture.

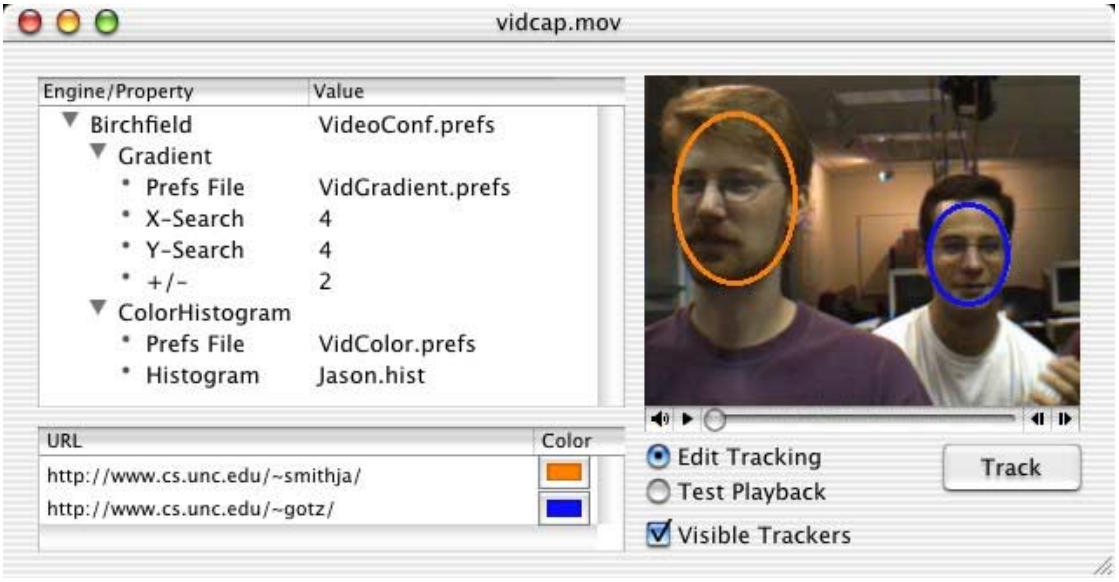


Figure 1. OvalTine user interface showing two tracked objects

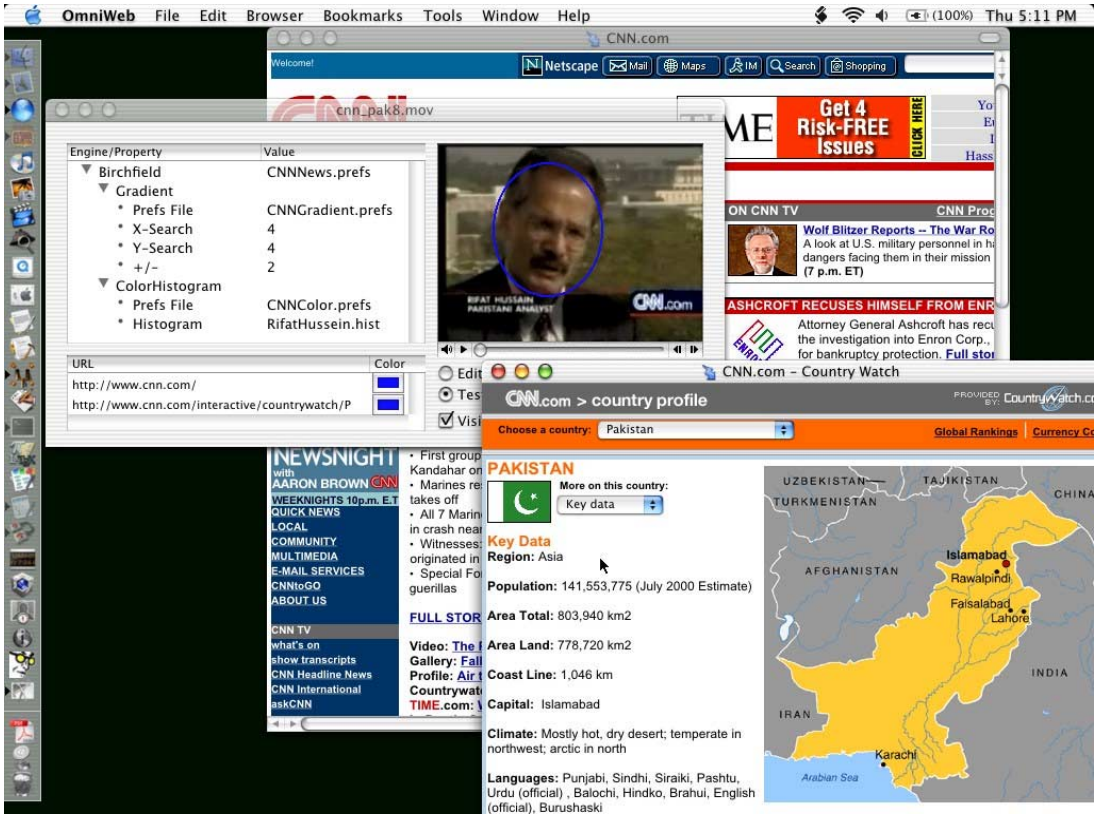


Figure 2. Two links on one tracked object, showing targets displayed

3. STORED VIDEO: SPRITE LINK LAYER

Our initial plan was to use video anchors and automated tracking to allow links to be created in video teleconferencing. This goal required us to find and employ efficient image analysis algorithms so that link anchors could be tracked in real-time, at frame rates of at least several per second for multiple anchors/targets. As the work progressed, it became apparent that the automated tracking methods we employed were useful for stored video streams as well as real-time streams.

OvalTine-RT does tracking of objects in video streams in real-time. While this has utility in creating hypermedia capabilities within previously inaccessible data domains like video-conferencing, it does not have enough infrastructure to capture and save all the calculated link anchors. They are computed frame-by-frame and then lost as the next frame comes along. To mark up stored video for playback, the basic tracking capabilities of OvalTine-RT had to be augmented with a capture and storage layer for link anchor positions.

To this end, we created Version 2 of OvalTine (OvalTine-SV) using the existing tracking structure and COTS software (Apple QuickTime and Sprites) for a link anchor storage layer. Our original plan was to simply extend OvalTine-RT on the SGI IRIX platform, utilizing the existing support for QuickTime saved movies. QuickTime is a container file format and access specification, not a movie *codec*, and allows for any QuickTime capable application to handle any media format for which a *codec* exists in the rather expansive QuickTime library. Multiple types of data can be added to a QuickTime file in *tracks*, such as video of various compression schemes, sound data, text, or any other time-variant information. QuickTime's primary function is to ensure that these tracks are kept synchronized. To reduce confusion with the tracking properties of OvalTine, we will use the term *layer* in this paper to refer to QuickTime *tracks*.

In the QuickTime nomenclature, sprites are object-based animations in a layer that is separate from, but synchronized with, the video layer. Sprites are usually used to produce simple animations, moving a series of static images around a video field much like traditional cartooning. Sprites exist in a QuickTime layer of their own, and can be thought of as an overlay over the video layer. The animations are separate from the video, so can be included in the user presentation, or not, by easily toggling a visibility flag. Each animation can be individually visible or not, or the entire layer can be toggled at once.

Wired sprites [9] are sprites that react to user interaction, registering events such as mouse-over (the cursor has been moved over the sprite by the user), mouse-down (the user has pressed the mouse button while the cursor is over the sprite), and mouse-up (the user has released the mouse button while the cursor is over the sprite). The wired sprites know their own boundaries and automatically track when the user is potentially interacting with them. They also keep track of which actions to perform for which user events, and these actions can include simple changes to the presented image, to moving within the movie to another time code, opening another movie and jumping to any time code within the new movie, and requesting actions of the system of other applications.

Wired sprites offer exactly what is needed to make a video markup storage layer – frame-by-frame synchronization, an overlay layer to contain the tracked object regions, with visibility toggling, built-in hit (click) detection, and 'link' traversal to trigger actions on a successful click by the user viewing the movie file. The file format is openly published, and QuickTime is fully supported on several platforms, including the various versions of Windows, and the two MacOS variants.

Re-Architecting the System

The implementation of QuickTime on IRIX, however, does not offer support for *sprites* (or did not at the time we made the implementation decisions). This limitation led us to re-sesign significantly and re-implement the software for OvalTine-SV. Since the object-tracking library has gone through a redesign to be cross-platform implementable, we took the opportunity to move OvalTine to a more commercial OS with stronger QuickTime support, including wired sprites. We also took the opportunity to extend the software architecture in a very useful direction, using several OO design patterns for dynamically composing objects [2]. The new OvalTine-SV architecture provides the ability to chain multiple trackers together to analyze any one anchor (this design and its advantages are discussed further in section 4).

The OvalTine-SV tracker system makes calls to the QuickTime library as it tracks objects in a frame. When the tracker discovers the new location of an object, the coordinates are transmitted to the sprite layer and recorded. When tracking is completed, the sprite layer information is incorporated into the QuickTime playback so the user sees the original video frame as well as the overlaid sprite annotations. In the next section we discuss the association of link targets (URLs) with the wired sprites.

3.1 Example markup session

A sample session might involve a user loading a video stream, perhaps an existing news archive footage file. The goal is to add hyperlinks to the video pointing to biographical information for selected subjects in the video. The user begins by selecting a QuickTime file to work on through use of the *Open... menu* item. The movie appears in the video view, at the first frame. The user uses the standard movie controller interface to move to the segment of the video they wish to begin tracking in. The *editing mode* is chosen from radio button palette, and a rectilinear box is drawn around the face to be tracked. An oval will appear around the face to indicate that the tracker has acquired it If the registration of the oval to the face is not pleasing to the user, they can simply choose the *Undo* menu item, and select again.

Once the object has been selected appropriately, the user can add the necessary tag information to be used when the object is later clicked on during playback. The *Links* window contains bookmarks to items the user may wish to add to an anchor, including http URLs for browser viewing, file URLs for local file viewing, or movie URLs that trigger movement to another time-code in the current movie; they might also open another movie and jump to a particular time-code in it. A link can be dragged and dropped onto the oval desired to be its trigger.

The user can now initiate a frame by frame tracking by selecting

Track from the *File* menu or toggle the *Track* button in the main window. As the tracker acquires the face in each frame, its reported position is displayed as an overlay over the movie. At the same time, the sprite, consisting of the oval, is saved to the movie data in memory. If the user at any time is unhappy with the tracking performance, they can stop the process, scan back to the first frame they feel is incorrect (again, using the standard movie controls), and re-select and re-initiate tracking. When the user is satisfied with the results of the tracking, the movie can be saved back to disk. Any QuickTime capable application will now be able to show the tracked face in the movie, and register hits on the face if clicked upon. The action on link traversal will be dependent on the capabilities of the particular viewer application. A web browser will be able to traverse URLs, for instance.

3.2 User Interface Examples

OvalTine-SV is set up to allow link markup of stored video as an editing task. Figure 1 shows the video display window, the properties window, and the URLs window. The video display window presents the video data and shows the tracked objects as (optionally) outlined ovals. For real-time applications, such as video conferencing, the playback controls are inoperative. For markup of stored video, the user can do the standard *start*, *stop*, *pause*, and *slider* frame selection operations on the video stream.

The tracking properties window allows the user to select the type(s) of tracking algorithms to apply to the video frames. The architecture of OvalTine-SV allows multiple tracking algorithms to be chained together and applied in sequence for each anchor to be tracked. A user can even apply a different chain of trackers to each different oval if desired; OvalTine-SV spawns a separate tracking thread for each ellipse. Some trackers work better than others in varying images; the selection of specific trackers to use depends on image properties such as color variability, background complexity, object motion, texture, etc. A default chain is designated for normal use. As in the Birchfield tracker of OvalTine-RT, this is currently set at an intensity gradient algorithm followed by a color histogram.

Once a link anchor (ellipse) has been established in the video window, the user can associate one or more URLs with that anchor to be targets of the link(s). These URLs show up in the URL window in the lower left. Figure 1 shows one URL for each oval, and the association is made by the color of the tag.

Editing progresses as described in the previous section. The user selects a starting frame with the slider controls, and then designates one or more objects to be tracked. The video is started with the "track" button, and the tracker chain for each oval causes the link anchors to follow the objects as they move in the video frame. The sprite infrastructure captures the layout information needed to maintain the link anchors in association with the video data. At any point the user may pause the video, add or delete ovals, and continue with tracking.

URLs can be added to the tracked objects at any point, either during tracking, or during playback editing of the marked-up video data. Though not shown, ovals can be linked to any first-class Web data, including another OvalTine-SV video clip. There is also a "lost" link palette (not shown) that collects the URLs associated with objects that are being tracked, but move out of the video window. Any URLs associated with such an object are taken out of the URL window and saved in the "lost" list. This is a convenience that makes it easier for the user to re-associate these URLs if the tracked object should reappear in the video window and need to be tracked again. An obvious topic for more research is applying face/object recognition algorithms to automate the re-association of lost links with link anchors when they re-appear in subsequent frames.

Figure 2 shows OvalTine-SV in use while viewing a marked-up video stream. Here we see a CNN clip where the face of the reported has been annotated with 2 different links. One link is to the CNN home page (seen displayed in the background). The other link is to an article and map on Pakistan, which is the topic of the video; this page is shown in the foreground.

There are several ways to manage multiple links on each oval. One is simply to left-click the mouse on the ellipse in the video window. This will cause the top link in its URL list (shown in the URL window, lower left) to be followed and displayed. Another method is to right-click on the oval and a pop-up menu will present all the URLs for that oval for selection of one to follow. A third method is to directly click on the desired URL in the URL window; these URLs are only displayed in the window while their associated ovals are in the video window. In the editing mode (as opposed to playback) the order of URLs in the URL window for each oval can be rearranged via click-and-drag operations so that new defaults can be established.

Finally, there is a mode the user can select during editing that causes *all* the associated links on an object to be followed concurrently when an oval is left-clicked. Each link target will be displayed in a separate browser window. This mode is related to the concurrent and synchronized browsing possible with the multi-head/multi-tail links found in MMM [5,6] and first studied in Trellis [3, 4].

4. OVALTINE SYSTEM ARCHITECTURE

After the initial prototype was operating, we spent time re-architecting and re-implementing OvalTine-RT to make the tool more adaptable to new tracking approaches and to adhere to good object-oriented design patterns [2]. The resulting OvalTine-SV prototype is robust and extensible.

We initiated a redesign of the code and functionality in January of 2001, and finished in late March of the same year, converting approximately 15,000 lines of C++ code of varying quality in some rather extreme ways. Design patterns were critical in the redesign and re-implementation phase, and used throughout the system.

OvalTine-SV started as a conglomeration of two very different sets of code. One, called *MediaConf,* is a demonstration video conferencing app from Silicon Graphics that runs on their line of O2 workstations. The other was a half-completed (and quite broken) set of code from Stanford University illustrating a way of tracking heads and faces in real-time video on a specific proprietary video card on the Windows platform. The work involved in taking these two utterly different bodies of code and making them work together was tremendous, not the least of which was involved in just making them work at all independently.

Once we had the code working well enough for demos, we

decided to re-factor and rewrite the system, since it utterly failed to meet our original vision. What we wanted was a flexible research platform that would run on multiple OSs, would handle video feeds from live cameras, stored files, or from a network feed, would allow different tracking algorithms to be experimented with in a freeform manner, and would be extremely simple to use and develop with. Instead what we had was a system that was incredibly fragile, had one hard-coded algorithm for tracking, ran only on the SGI O2 platform (and then only on certain OS versions), and was limited to a live local video feed.

The reworking effort was done from the viewpoint of using design patterns as a springboard for system architecture design, as well as guiding implementation. These patterns gave the architecture the following properties:

- Extensible algorithm system for object tracking in the video stream... algorithms should be easily added to the pool for selection, and should work interchangeably
- Algorithms should be chainable for efficient pre and post processing
- Dynamic selection of algorithms, preferably on the fly during execution, but a simple selection process for configuration at launch is acceptable
- The video tracking library should have a simple external API
- The video tracking library should be completely separated from any GUI, front end app, or IO, except as through the above API
- The video tracking library should be cross-platform

While each requirement created specific patterns in our new architecture, the first two (extensible trackers, chainable trackers) are of particular interest to a hypervideo audience. We therefore examine each property in more detail in the following sections.

4.1 Extensible tracking algorithms

One of the most critical components needed in the redesign was to allow new tracking algorithms to be added to the system as imaging researchers develop them. This exactly corresponds to the *Strategy* pattern, whose intent is given in the patter literature as:

> "Define a family of algorithms, encapsulate each one, and make them interchangeable."

As new concepts in identifying object in an image are developed, OvalTine-SV can be extended without being abandoned or re-written. Extensions can be done by hypervideo designers with OO programming skills; it involves writing new classes with the same basic form as the existing trackers, and adding their compiled for to the class libraries.

Figure 3 shows the full design pattern diagrams for the *Strategy* pattern that give OvalTine-SV extensibility in tracker algorithms. By looking at the structure of the pattern in these diagrams, we can quickly start identifying the basic concept classes to fulfill the roles: *Context* is our Tracker system, *Strategy* is an abstract base class corresponding to our idea of a SearchEngine, and the *ConcreteStrategy* roles will be filled by the actual algorithms. In our initial situation, we have one algorithm, which we called *BirchfieldSearchEngine*, after the author of the describing paper.

4.2 Chainable trackers

The *BirchfieldSearchEngine* is actually a hybrid tracking algorithm that uses two separate approaches: a gradient based edge detection algorithm to find an oval shape corresponding to a head; and a color histogram approach that detects the face itself through a visual field analysis. We wanted to compare the performance of the *BirchfieldSearchEngine* against the individual algorithms comprising it, but unfortunately the conglomerated algorithm was originally written as a monolithic piece of code. We considered writing two completely new engines, to recreate the two algorithms, but decided that this was a poor way of reusing the concepts, since we would be reusing no code.

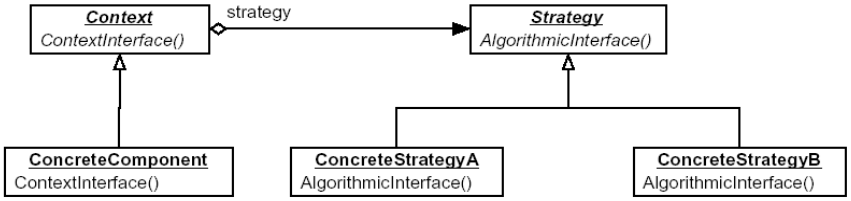
What was needed was a way to have the object *BirchfieldSearchEngine* reuse the code from the other engines, instead of attempting to reuse the code from the hybrid. This would necessitate a *SearchEngine* class calling two other *SearchEngines*, and using the results to create a third set of tracking feedback. This meets the criteria for applying the *Composite* pattern:

> "Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly."

Figure 4 shows the design pattern diagrams for the *Composite* we use to give OvalTine-SV chainable trackers. The structure of the pattern is straightforward, where the *Client* role is naturally again that of *Tracker*, the *Component* is a *SearchEngine*, the class fulfilling the *Composite* role we will call blandly enough *CompositeSearchEngine*, and the *Leaf* role will be our various *SearchEngines* that do the actual tracking.

This feature allows OvalTine-SV to track objects by passing image information from a frame though several different tracking algorithms in series. Each has the opportunity to decide if an object appears in the image or not, either using the data on its own or by using the conclusions of a prior tracker.

We also spawn a new thread for each object being tracked. This allows a different chain of trackers to be applied to each different object, if such is desired. It might, for example, be easier to track faces with one combination of algorithms, and easier to track rectilinear objects (books, signs, etc.) with a different set of algorithms. OvalTine-SV will support this variance.



ggggg

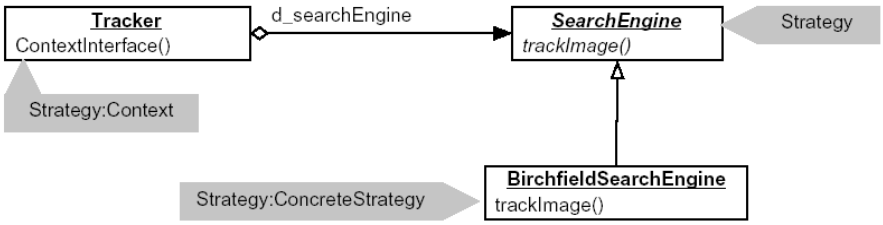


Figure 3: Strategy pattern for extensible tracker

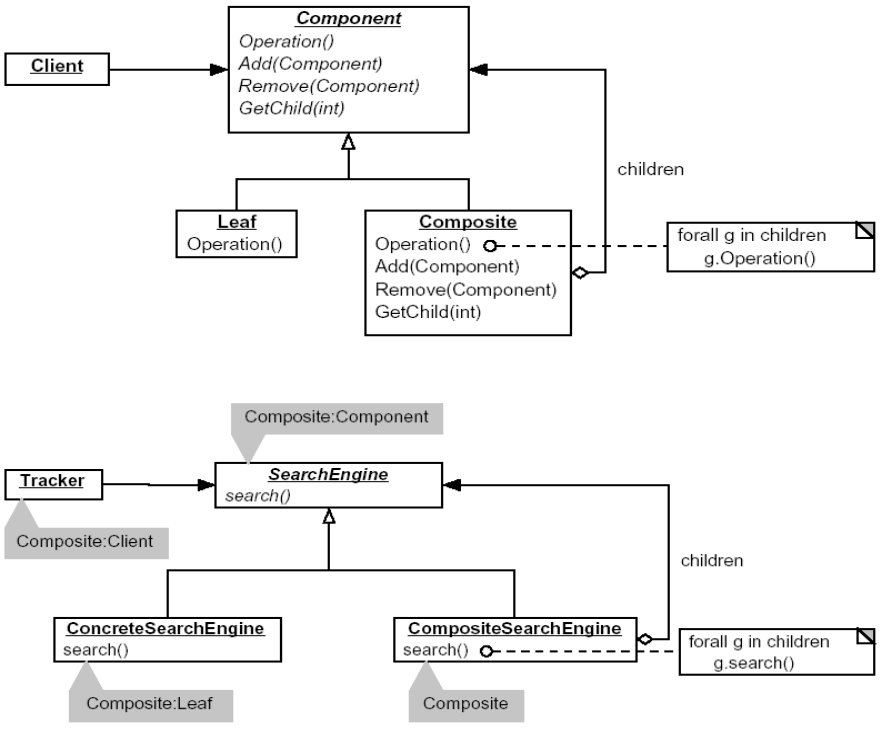


Figure 4: Composite pattern for chainable tracker

5. RELATED WORK

Multimedia has been trending from text to static images to video. On a parallel path, hyperlinking systems, most notably the World Wide Web, have evolved from the use of pure text to the incorporation of static images with embedded anchors (image maps), and now to video streams with active regions [9, 17].

Video Segmentation and Object Tracking

Numerous research projects have concentrated on segmentation of video into regions (objects). The results are too numerous to list thoroughly. Our focus is not on finding better segmentation or location methods, but rather on employing effective ones for creating hyperlinks for hypervideo. Our architecture makes it easy for better identification and tracking methods to be worked into OvalTine-SV as they are found, or to be used in combination as in Birchfield's tracker [25,26] mentioned earlier as our base technique.

Notelook [24], from Xerox, is a system for manipulation of video data during a conference or presentation; it allows a form of video annotation using the live video feed. Frames from the video stream can be captured and incorporated into notes on a pen-based computer; web pages can be generated with links to the captured notes and video clips. It does not, however, seems to allow links in the captured video streams; rather it seems to create links on web pages that launch video clips.

In [28] researchers are tracking faces in a video stream for a teleconference for the purpose of camera control; they use skin color cues to identify objects to track. This method does not distinguish between faces and hands well, however, and the Birchfield combination deals more effectively with environments that have skin-colored regions that are not faces.

Numerous projects are developing ways to identify objects in a video frame more rapidly, more exactly, and more reliably [31,32, 36, 27, 22], including work done in the compressed domain [35]. Our work currently is in the non-compressed domain, but nothing prevents compressed techniques from being incorporated into the software architecture of OvalTine. Techniques like those in [32] scan the entire video frame to segment it into objects; we depend on the user to highlight the initial region of an anchor, eliminating the need for methods that require scanning and segmenting the entire video frame.

Some methods are promising in their exactness [33, 34] but are too computationally demanding for use in high-frame-rate real-time tracking and are better suited for off-line annotations. OvalTine-RT currently uses trackers that are simple enough to be computed for multiple objects at a high-frame-rate. We have not altered the tracker selections for OvalTine-SV, but we could possibly use more expensive (and hence more accurate) techniques for stored video annotation. However, we would have to alter OvalTine-SV somewhat to slow down the replay rate so each frame would be annotated. In our current implementation, tracking is done by analyzing whatever the current frame happens to be when the previous tracker analysis is done. Hence, expensive trackers would not work acceptably in real-time streams; they would also cause too many intervening frames to be skipped in stored video annotation if the replay rate were not slowed.

Video in Hypermedia Documents

A good overview of the issues and technologies in current hypervideo systems can be found in class notes at Texas A&M [8]. The Hypercafe system from GA Tech is often cited in hypervideo discussions [7]; however, it is a presentation system mostly and does not support the dynamic and automatic link anchor creation we are exploring in OvalTine. Automation of content creation in hypertext is a well-researched area for non-video media [10,11,13]. While much work has been done in the realm of context-assisted anchor creation in video, particularly in news coverage videos, these rely on a blend of modal data, including much that is manually entered by human operators [12, 15, 20, 21].

We are more interested in nearly-context-free object tracking within image streams. The image analysis engine selected for use on the video, or the user manually initiating links, can supply the context. A facial recognition system designed to attribute a link to a database record consists of an entirely different inherent context than an engine that identifies and tracks types of automobiles on a roadway, but they both can operate equally well on the same raw video stream. This is a complementary concept to content-oriented navigation [14], where the context is provided during modal analysis. Since the context of the link data has been removed from the dimensions we define, the resultant contextual link generation engines are also close analogues to *Sprocs* as defined by Nürnberg, et al [18].

Links in Hypervideo

Every current popular method for adding link hotspots to video requires manual selection of video objects, on a frame-by-frame basis. No research efforts in automation have yet made it into common practice in a widely used system such as those from Apple (QuickTime [9]). By contrast, an automatable object tracking system is much more desirable, both for real-time applications, and for the automated addition of hyperlinks to the vast amount of archived video currently in existence.

Two notable exceptions are the Himotoki system from NEC [14] and MAVIS [23]. Himotoki seems to do automated extension of link hotspots from initial frame into succeeding frames, during playback, without manual interventions, but no details of how this is done are given in their report. The MAVIS hypermedia system provides links in video by computing color histograms and other image analysis methods on selected regions of a video frame; the report on MAVIS indicates that videos are halted for link following, at which time the analysis is done and a database search initiated for matches. This "late binding" approach has the advantage of allowing the items linked to a video hotspot to grow in number as the target database is expanded, since matches are computed rather than hardwired to a hotspot.

Our approach allows link following without stopping the video stream, since the link anchors are tracked from frame-to-frame automatically. Note that this dynamic capability is what made OvalTine-RT techniques useful for real-time streams.

5. CONCLUSIONS

OvalTine-RT is a system for tracking objects in real-time video streams (video conferences) and allowing hyperlink anchors to be associated with these tracked objects. We have described here the further use of OvalTine-SV for applying automated object tracking to stored video streams, thereby allowing automated markup of archived video data with hyperlinks.

Extending OvalTine from real-time to archived video requires a link layer to capture and maintain the link anchors as they are tracked from frame to frame; in the real-time mode, this information is available at each instant, but lost as each frame progresses to the next. We demonstrated such a link layer on top of the basic OvalTine tracker using COTS software – namely, the *Wired Sprites* of Apple's *QuickTime* standard. We designed the software architecture of OvalTine-SV using OO design patterns to allow multimedia researchers to extend the basic object tracking algorithms with their own work, or to conveniently investigate combination of trackers for effectiveness in varying video environments.

Use of OvalTine-SV for hyperlink markup of stored video in not fully automated, as it requires user selection of initial anchor regions, as well as user designation of the URLs for the link targets. However, it can progress as rapidly as viewing the videos, once link anchors are designated. This is a significant improvement on how video are marked up for hyperlinks currently.

7. REFERENCES

[1] Smith, J., D. Stotts, and S.-U. Kum, "An Orthogonal Taxonomy for Hyperlink Anchor Generation in Video Streams using OvalTine," *Proc. of Hypertext 2000 (ACM)*, May, 2000, San Antonio, Texas, pp. 11-18.

[2] Gamma, Helm, Johnson, and Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, 1995.

[3] Stotts, P.D., and R. Furuta, "Petri Net Based Hypertext: Document Structure with Browsing Semantics," *ACM Trans. on Information Systems* (ACM), vol. 7, no. 1, Jan. 1989, pp. 3-29.

[4] Furuta, R., and P. D. Stotts, "Programmable Browsing Semantics in Trellis," *Proc. of Hypertext '89 (ACM)*, Pittsburgh, Nov. 1989, pp. 27-42.

[5] Capps, M., B. Ladd, D. Stotts, "Enhanced Graph Models in the Web: Multi-client, Multi-head, Multi-tail Browsing," Computer Networks and ISDN Systems, vol. 28 (Proc. of the 5th WWW Conf., May 6-10, 1996, Paris), pp. 1105-1112.

[6] Ladd, B., M. Capps, D. Stotts, and R. Furuta, "Multi-head/Multi-tail Mosaic: Adding Parallel Automata Semantics to the Web," *World Wide Web Journal*, O'Reilly and Associates Inc., vol. 1 (*Proc. of the 4th International WWW Conference*, Boston, December 11-14, 1995), pp. 433-440.

[7] Sawhney, N., D. Balcom, and I. Smith, "HyperCafe: Narrative and Aesthetic Properties of Hypervideo", Hypertext '96 Proceedings, ACM, Washington, D.C., 1996, pp. 1-10.

[8] Francisco-Revilla, L., "A Picture of Hypervideo Today",

http://www.csdl.tamu.edu/~l0f0954/academic/cpsc610/p-1.htm, 1998.

[9] Apple Computer, "Introduction to Wired Movies, Sprites, and the Sprite Toolbox", http://developer.apple.com/techpubs/quicktime/qtdevdocs/REF/refWiredIntro.htm

[10] Bernstein, M., "An apprentice that discovers hypertext links", Hypertext: Concepts, systems and applications: Proceedings of the European conference on Hypertext, INRIA, France, 1990, pp. 212-223.

[11] Bernstein, M., J. D. Bolter, M. Joyce, and E. Mylonas, "Architectures for Volatile Hypertext", Hypertext '91 Proceedings, ACM, San Antonio, TX, 1991, pp. 243-260.

[12] Boissière, G., "Automatic creation of hypervideo news libraries for the World Wide Web", Hypertext '98 Proceedings, ACM, Pittsburgh. PA, 1998.

[13] Glushko, R.J., "Design issues for multi-document hypertexts", Hypertext '89 Proceedings, ACM, Pittsburgh, PA, 1989, pp. 51-60.

[14] Hirata, K., Y. Hara, H. Takano, and S. Kawasaki, "Content-oriented Integration in Hypermedia Systems", Hypertext '96 Proceedings, ACM, Bethesda, MD, 1996, pp. 11-21.

[15] Ip, H.H., and S. Chan, "Hypertext-Assisted Video Indexing and Content-based Retrieval", Hypertext '97 Proceedings, ACM, Southhampton, UK, 1997, pp. 232-233.

[16] Kendall, R., "Hypertext dynamics in *A Life Set for Two"*, Proc. of Hypertext '96, ACM, Bethesda, MD, 1996, pp. 74-84.

[17] Liestøl, Gunnar, "Aesthetic and Rhetorical Aspects of Linking Video in Hypermedia", Proc. of Hypertext '94, ACM, 1994, pp. 217-223.

[18]. Nürnberg, P.J., J.J. Leggett, and E.R. Schneider, "As We Should Have Thought", Proc. of Hypertext '97, ACM, Southhampton, UK, 1997.

[19] Stotts, P. D., and R. Furuta, "Dynamic adaptation of hypertext structure", Proc. of Hypertext '91, ACM, San Antonio TX, 1991, pp. 219-232.

[20] Zhang, H.J., et al., "Automatic Parsing and Indexing of News Video", *Multimedia Systems*, 2 (6), pp. 256-266, 1995.

[21] Zhang, H.J., C.Y. Low, S.W. Smoliar, and J.H. Wu, "Video Parsing, Retrival and Browsing: An Integrated and Content-Based Solution", Multimedia '95 Proceedings, ACM, 1995, pp. 15-24.

[22] Hua, K.A., and J. Oh, "Detecting video shot boundaries up to 16 times faster," Proc. of ACM Multimedia 2000, Los Angeles, CA, 2000, pp. 385-387.

[23] Lewis, P., H. Davis, S. Griffiths, W. Hall, and R. Wilkins, "Media-based Navigation with Generic Links, " Proc. of ACM Hypertext '96, Washington, DC, March 1996, pp. 215-223.

[24] Chiu, P., A. Kapuskar, S. Reitmeier, and L. Wilcox, "NoteLook: Taking Notes in Meetings with Digital Video and Ink," Proc. of ACM Multimedia '99, Orlando, FL, Nov. 1999, pp. 149-158.

[25] Birchfield, S., "Elliptical head tracking using intensity gradients and color histograms," Proc. of the IEEE Conf. On Computer Vision and Pattern Recognition, Santa Barbara, CA, June 1998, pp. 232-237.

[26] Birchfield, S., "Elliptical Head Tracking Using Intensity Gradients and Color Histograms," implementation code at URL http://vision.stanford.edu/~birch/headtracker/

[27] Fan, L., and K.K. Sung, "Model-based varying pose face detection and facial feature registration in video images," Proc. of ACM Multimedia 2000, Los Angeles, CA, 2000, pp. 295-302.

[28] Stiefelhagen, R., , J. Yang, and A. Waibel, "Modeling Focus of Attention for Meeting Indexing," Proc. of ACM Multimedia '99, Orlando, FL, Oct. 1999, pp. 3-10.

[29] Silicon Graphics, Inc., "mediaConf Videoconferencing," https://toolbox.sgi.com/src/demos/O2/mediaConf/, July 1997.

[30] Smith, J. McC., "HeadTrackerLib: Generalized head tracker," http:://www.cs.unc.edu/~smithja/HeadTrackerLib/, Oct. 1999.

[31] Yang, J., X. Zhu, R. Gross, J. Kominek, Y. Pan, and A. Waibel, "Multimodal people ID for a multimedia meeting browser," Proc. of ACM Multimedia '99, Orlando, FL, Oct. 1999, pp. 159-168.

[32] Kuhne, G., S. Richter, and M. Beier, "Motion-based segmentation and contour-based classification of video objects," Proc. of ACM Multimedia '01, Ottawa, Canada, Oct. 2001, pp. 41-50.

[33] Luo, H., and A. Eleftheriadis, "Designing an interactive tool for video object segmentation and annotation," Proc. of ACM Multimedia '99, Orlando, FL, Oct. 1999, pp. 265-269.

[34] Pass, G., R. Zabih, and J. Miller, "Comapring images using color coherence vectors," Proc. of ACM Multimedia '96, Boston, MA, 1996, pp. 65-73.

[35] Luo, H., and A. Eleftheriadis, "On face detection in the compressed domain," Proc. of ACM Multimedia 2000, Los Angeles, CA, 2000, pp. 285-294.

[36] Kim, C., and, J.-N. Hwang, "An integrated scheme for object-based video abstraction," Proc. of ACM Multimedia 2000, Los Angeles, CA, 2000, pp. 303-311.