Content-based Retrieval of Flash™ Movies: Research Issues, Generic Framework, and Future Directions*

Jun Yang[1,2*] Qing Li[1] Liu Wenyin[3] and Yueting Zhuang[4]

[1]Dept. of Computer Engineering and Information Technology, City University of Hong Kong, Kowloon, HKSAR, China

[2]Language Technology Institute, School of Computer Science, Carnegie Mellon University, USA

[3]Dept. of Computer Science, City University of Hong Kong, Kowloon, HKSAR, China

[4]Dept. of Computer Science, Zhejiang University, Hangzhou, 310027, China

ABSTRACT

As a prevailing web media format, nowadays Flash™ movies are created, delivered, and viewed by over millions of users in their daily experiences with the Internet. However, issues regarding the indexing and retrieval of Flash movies are unfortunately overlooked by the research community, which severely restrict the utilization of the extremely valuable Flash resource. A close examination reveals that the intrinsic complexity of a Flash movie, including its heterogeneous media components, its dynamic nature, and user interactivity, makes content-based Flash retrieval a host of research issues not thoroughly addressed by the existing techniques. As the first endeavor in this area, we propose a generic framework termed as FLAME (*FLash Access and Management Environment*) embodying a 3-layer structure that addresses the representation, indexing, and retrieval of Flash movies by mining and understanding of the movie content. An experimental prototype for Flash retrieval is implemented to verify the feasibility and effectiveness of FLAME, and future research directions on Flash management and retrieval are discussed in details.

KEYWORDS

Flash animation, content-based retrieval, multimedia information retrieval

1. INTRODUCTION

Flash™ is a new format of vector-based interactive movie proposed by Macromedia Inc. [19], which can be embedded in web pages and delivered over the Web. After its advent dated 1997, Flash has experienced a remarkable growth in the global scale and become nowadays a prevailing format on the Web. Macromedia statistics [20] states that by June 2002, 97% (or 450 million) Internet users are able to view Flash movies using Macromedia Flash Player, the rendering tool of Flash, compared with, 69% for Media Player compatible movies, 51% for Real

* The work described in this paper was substantially supported by a grant from CityU (Project No. 7001457); a substantial amount of work by this author was conducted when he was a research assistant at City University of Hong Kong.

Player movies, and 35% for QuickTime Player movies. 7 out of the top 10 global websites or 62% among the top 50 have adopted Flash content in their pages, including AOL Time Warner Network, Yahoo!, and eBay. Given the observation that most websites use distinct Flash movies (partly due to the copyright reason), the popularity and accessibility of Flash is enormous. Besides their predominant presence as multimedia and interactive components of static, textual websites, Flash movies are being created everyday as cartoons, commercial advertisements, electronic postcards, MTV movies, and computer games, each of which has enormous market potentials. Electronic commerce, interactive media, and various Web applications are among the future applications of Flash.

Table 1 shows a technical comparison between Flash and its competing technologies, such as streaming video, GIF animation, and JavaScript and VBScript, which, like Flash movies, can be used to create interactive and media-rich web pages. Technical advantages contributing to the prevalence of Flash over these competitors mainly include: (1) vector-based Flash movies are usually of small size and therefore can be delivered efficiently even under low-bandwidth network conditions; (2) Flash movies are easy to create even by non-professional users using authoring tools such as Macromedia Flash v4.0/5.0/MX and third-party tools like Flash Animator, SWF Studio (some of which are free wares); (3) last but not the least, Flash supports user interactivity which allows users to control the movie presentation through certain behaviors, e.g., clicking a button in the movie[1]. In comparison, none of the competing technologies of Flash has all the above features. Furthermore, the latest versions of Flash gradually exhibit features of a programming language and the ability of talking to back-end databases. Considering all these unique features, the prevalence of Flash is likely to persist even when the global deployment of broadband offsets its advantage of compactness and MPEG-4 videos are widely adopted.

Table 1: Technical comparison between Flash and its competing technologies

Technology	Format	Size	Ease of composition	Interactivity
Macromedia Flash	Vector-based interactive movie	Small	Easy (with Macromedia Flash 3.0/4.0/MX)	Supported
Streaming video	Compressed pixel-based video	Large	Difficult (movie production skills)	N/A
GIF animation	A stack of pixel-based GIF images	Medium	Medium	N/A
JavaScript &VBScript	Script in HTML source	Very small	Difficult (high programming skills)	Supported

Given the popularity of Flash and its promising future, it becomes an imperative task of the multimedia research community to develop effective and efficient retrieval tools for Flash movies, which are potentially useful to a variety of user groups, ranging from teenagers looking for games (as some games are made by Flash), music fans looking for MTVs, to Flash developers reviewing the designs of existing movies, and customers searching for advertisements. Some online Flash repositories, such as Flash Kit [8], have provided users with simple search

[1] If not indicated explicitly, we use “movie” to refer to “Flash movie” in this paper.

functions, which accept user queries in the form of keywords and retrieve relevant movies by matching the query with the manual (keyword) annotation attached to each movie. This approach, however, overlooks the rich auditory/visual cues contained in the movie content, which provides an indispensable criterion for evaluating user queries orthogonal to descriptive keywords. On the other hand, although previous research has covered the retrieval of various types of media (e.g., text, image, video), there is, to the best of our knowledge, no related work on indexing and retrieval of Flash movies by content, despite the fact that Flash is equally or even more popular than other media on the Web. Therefore, we are motivated to present the first piece of work (again, to our knowledge) on content-based Flash retrieval.

A close examination of Flash reveals its intrinsic complexity on three major aspects: (1) a typical Flash movie usually contains *heterogeneous components*, including texts, graphics, images, QuickTime™ videos, sounds, and even recursively embedded Flash movies; (2) it features *dynamic nature* that is constituted by the spatio-temporal features of its components; and (3) it enables *interactivity* which allows users to interfere with the playout (i.e., presentation) of the movie. Naturally, the media components, dynamic effects, and user interactions in a movie constitute the major types of expressive elements that synthesize the movie semantics. The research issues on the indexing of these movie elements and the analysis of their roles in conveying movie semantics are critical for effective Flash retrieval but are not thoroughly addressed (or even unexplored) in previous research. This paper presents a discussion over these issues and their connections with existing multimedia retrieval techniques. Please note that although a Flash movie is perceptually close to a video clip, Flash retrieval differs significantly from video retrieval since (a) movie components can be readily extracted from vector-based Flash movies, whereas segmenting meaningful objects from pixel-based video clips is extremely difficult (if not impossible) given the state-of-the-art computer vision technology; and (b) user interactivity adds a new dimension to the retrieval of Flash retrieval which is not addressed in video retrieval.

As the main contribution of this paper, a generic framework termed as FLAME (*FLash Access and Management Environment*) is proposed that facilitate users to access Flash movies by content. Specifically, the emphasis of FLAME focuses on the extraction, representation, and retrieval of various movie elements, which are characterized by multi-level features including (1) *object* features describing heterogeneous media components embedded in movies, (2) *event* features depicting the dynamic effects made up by objects and their spatio-temporal features, and (3) *interaction* features capturing the relationships between user behaviors and the events triggered by the behaviors. Given the sophistication of FLAME (which is due to the complexity of Flash), it is unrealistic to address all the components with satisfactory solutions within the scope of this paper. In fact, the objective of this paper is to come up with a comprehensive “skeleton” such that many follow-up works will be devoted to “fill in” the components of this skeleton. To verify the effectiveness of FLAME, we have also implemented a prototypical Flash retrieval system as an “instance” of the framework. Moreover, a number of promising future directions that are not covered by FLAME are discussed.

The rest of the paper is organized as follows. In Section 2, we present an overview of content-based Flash retrieval, addressing the characteristics of Flash, the research issues, and its connection with previous works. We

elaborate on the proposed FLAME framework in Section 3, and describes the prototypical Flash retrieval system in Section 4. The promising future directions are discussed and the concluding remarks given in Section 5.

2. OVERVIEW OF CONTENT-BASED FLASH RETRIEVAL

In this section, we describe the unique properties of Flash and the research issues of Flash retrieval aroused by these properties. We also discuss how Flash retrieval is related to previous works on multimedia retrieval and how it can benefit from them.

2.1 Characteristics of Flash Movies

Through a close examination of the content of several sample movies, we discover that the semantics of a Flash movie is mainly synthesized and conveyed by the following three types of devices:

- *Heterogeneous components*. A typical Flash movie usually consists of component media objects of various types. Texts and graphics (i.e., drawings) of arbitrary complexity can be easily created as components using authoring tools of Flash. Bitmap or JPEG images and QuickTime video clips can be imported into the movie as well. Sounds compressed using ADPCM or MP3 standard are embedded in the movie in one of the two forms: event sound, which is played in response to certain event such as mouse-click, and streaming sound, which is played in synchronization with the advance of the movie. According to the format of Flash [21], all these components are encoded separately such that they can be easily extracted from Flash data files. This differs fundamentally from pixel-level media formats such as image and video. Furthermore, a Flash movie can consist of recursively embedded Flash movies, which are defined as a special type of components. An embedded movie component can also consist of its own components and support dynamic effects of them.

- *Dynamic effect*. A Flash movie is composed of a sequence of frames that are played in an order subject to user interactions. With the progression of frames, components can be placed on the current frame, removed from it, or changed with respect to their positions, sizes, shapes, and angles of rotation. The spatio-temporal features of the components, as well as the spatio-temporal relationships among them, make up of some high-level dynamic effects (such as morphing, motion), which suggest the semantic meaning of a movie.

- *User interactivity*. Rather than a passive media such as streaming video, Flash is an interactive movie in the sense that a user can interfere with the presentation of the movie. As an example, by clicking a button in a movie the user can let the movie “jump” to a frame prior to the current frame, while clicking another button may cause the movie jump to a frame behind the current one. Consequently, an interactive Flash movie usually has multiple presentations, and each of them is the result of a certain series of user behaviors.

2.2 Research Issues

The intrinsic complexity of Flash leads to many issues concerning Flash retrieval that have not been addressed in existing research successfully. The essential issues among them are discussed as follows:

- *Indexing of heterogeneous components, dynamic effects, and user interactions*. A good retrieval approach should conform to the users' subjective criteria regarding the relevance of Flash movies to their needs. From a user point of view, the heterogeneous components, dynamic effects, and interactive feature of Flash are all important clues based on which users are likely to query for movies. For example, user may search for those movies "accompanied by the song 'Yesterday'", movies "containing the text 'Lord of rings'", or movies "describing the scene of sunset". To support such queries, the properties of Flash on each of the three aspects should be indexed with particularly designed features that facilitate effective retrieval of Flash movies. Especially, different features are required to describe component objects of different types (i.e., text, graphic, image, video, sound). Although the indexing of component objects and their dynamic effects can be fully or partially addressed by existing techniques, modeling user interactions poses a brand-new problem. The indexing method should also be able to deal with the case that a movie component is recursively embedded in another Flash movie.

- *Retrieval models for diverse features*. Since the features of a Flash movie are diverse in semantics and representation, multiple retrieval methods are needed for Flash retrieval based on various features. For example, text-based information retrieval (IR) methods can be used for Flash retrieval by the text components embedded in movies; content-based retrieval (CBR) techniques are suitable for multimedia components such as images, videos, and sounds; database-style queries can facilitate retrieval by predefined features such as the shape of graphic components. Ad hoc retrieval methods are needed for the features of dynamic effects and user interactions, depending on their respective representations.

- *Query specification and user interface*. Given the diversity of retrieval methods, user queries also need to be specified in a variety of ways, e.g., keyword query for IR approach, query-by-example for CBR approach, query language for database-style search. All the means of query specification should be provided in an integrated interface that allows users to compose various queries conveniently and efficiently. Furthermore, as the query interface directly addresses user experiences, it should be designed friendly and easy-to-use for average users. In particular, for complex query methods such as using a query language, iconic/graphic query specification techniques need to be investigated.

2.3 Connection with Previous Works

There have been a great number of publications devoted to the retrieval of multimedia data (including text), and for each type of data, some specific retrieval approaches have been proposed. Among them, text-based information retrieval (IR) technique [26] is mainly used for searching large text collections using queries expressed as keywords. Content-based retrieval (CBR) technique [11] is invented by the Computer Vision community to retrieve multimedia objects based on *low-level features* that can be automatically extracted from the objects. In the past decade, a large number of CBR systems have been built for image retrieval (such as QBIC [9], MARS [25], and VisualSEEK[29], as overviewed in [28]), video retrieval (such as Informedia [13], VideoQ [5]), and audio retrieval [10]. The low-level features used in CBR techniques vary from one type of multimedia to another, ranging from keywords for texts, color and texture for images, and pitch and melody for audios. In addition, database access using some declarative

query language (such as SQL) [7] is a method widely used by the Database community to retrieve structured data based on predefined attributes, such as captions of images, titles of documents. This approach is basically applicable to any types of data provided that they conform to certain structures or constraints (i.e., schemas). However, since none of the exiting media formats has all the three characteristics of Flash (e.g., text, image, video, and audio are all non-interactive), the retrieval techniques developed for the existing media data cannot be applied to Flash retrieval without significant modification. On the other hand, since a Flash movie usually contains various types of multimedia objects in it, these existing techniques can serve as the "component techniques" to support Flash retrieval based on the features of its component objects.

Spatial/temporal features have been addressed by the research on multimedia with dynamic nature, usually, video streams. The techniques used for indexing and querying of spatial/temporal features exhibit great variety. For examples, 2-D strings [4], spatial quad-trees [27], and graphs [12] are widely used techniques for flexible and efficient spatial query. The research on temporal databases is surveyed by Ozsoyoglu et al. [23]. In the VideoQ system proposed by Chang et al. [5], videos are retrieved based on the joint spatio-temporal attributes of video objects represented as motion trajectories. In the work of Hjelsvold et al. [14] and Chan et al. [3], specialized query languages augmented with spatial and temporal operators are proposed for video retrieval. However, the complexity of the dynamic effects supported by Flash goes beyond the capability of the current techniques on modeling spatial/temporal features. For example, Flash supports the morphing of a graphic component from a square to a trapezium, which is insufficiently represented by any current techniques. On the other hand, the usefulness of some detailed spatio-temporal attributes such as motion trajectory is arguable, since users are unlikely to query movies by specifying the precise movement of a component, say, "find Flash movies with a circle moving from coordinate (0,0) at frame 1 to (100, 50) at frame 10". In contrast, high-level, semantics-flavored queries are usually more preferred by users.

In addition, some research works are devoted to the modeling and retrieval of generic *multimedia presentation*, defined as a synchronized and possibly interactive delivery of multimedia as a combination of video, audio, graphics, still images, and animations [16]. In this sense, a Flash movie is a typical type of multimedia presentation. Lee et al. [16] adopted an acyclic-graph representation of multimedia presentation and proposed a graphical query language as an effective tool to query and manipulate multimedia presentations based on their content. Adali et al. [1] suggested an algebra for creating and querying interactive multimedia presentation databases based on a tree model, whose branches reflect different "playouts" of the presentation determined by user interactions. Both approaches are not sufficient for Flash retrieval since (1) they do not tailor to the specific characteristics of Flash presentation, and (2) they are not automated. Although the approach of Adali et al. [1] is probably the first piece of work (to our knowledge) addressing the user interactivity in multimedia data, the complexity and diversity of user interactions in Flash are far beyond the capability of the simple tree model used in their approach. For example, the tree model cannot describe the situation that a segment of frames in a Flash movie is played in loops as the result of user interactions.

3. FLAME: A GENERIC FRAMEWORK FOR FLASH RETRIEVAL

FLAME (*FLash Access and Management Environment*) is proposed as a generic framework for indexing and retrieval of Flash movies by semantic content. As illustrated in Figure 1, it has a 3-layer structure constituted by representation layer, indexing layer, and retrieval layer from bottom to top. The details of each layer are described in this section, and some sample queries are given to demonstrate the usefulness of FLAME.

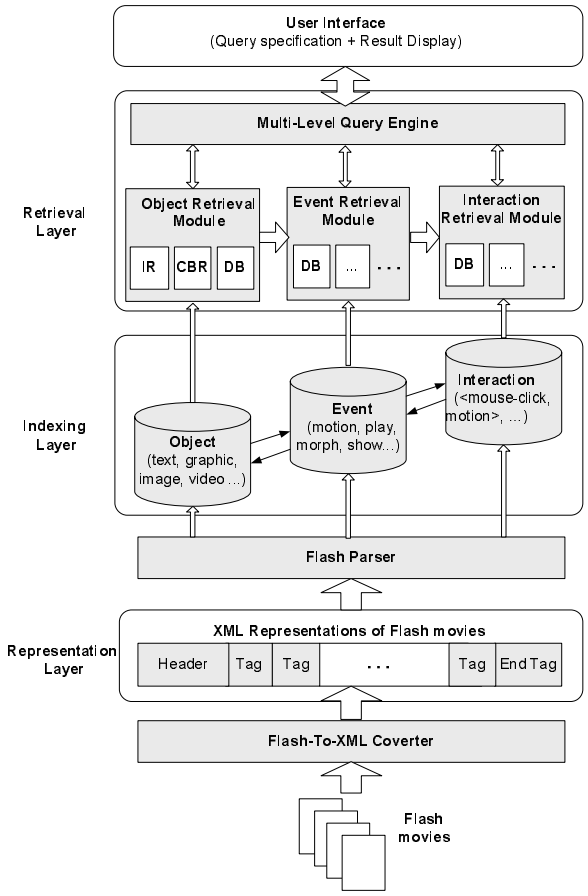

Figure 1: The 3-layer structure of FLAME

3.1 XML-based Movie Representation (Representation Layer)

Flash movies are delivered over the Internet in the form of Macromedia Flash (SWF) file. Each Flash file is composed of a series of tagged data blocks, which belong to different types with each type having its own structure. In essence, a SWF file can be regarded as an encoded XML [15] file (a Flash file is binary while a XML file is ASCII text file), and it can be converted into a XML file using tools such as JavaSWF [15]. Each tagged data block in a SWF file is mapped to a XML tag, which usually has attributes and embedded tags representing the structured data inside the block. Data blocks of the same type are mapped to XML tags with the same name. In the representation layer, we convert SWF files into XML formats using Flash-To-XML Converter for two reasons: (a) XML files are readable and thus convenient for us to understand the internal structure of Flash; (b) being a global

standard, XML format facilitates interoperability with other applications. A sample segment of the XML encoding of a Flash file is shown in Figure 3. We do not choose the source files of Flash (.fla files) as the basis of our content analysis, since they are not available in the published Flash movies and their format is unknown to us, though they may contain more (but redundant) information.

```
<movie version="4" width="550" height="400" rate="12">
    <background-color red="255" green="255" blue="255" />
                                        ......
                                        ......

  <shape id="1" has-alpha="no" min-x="-280.45" min-y="-206.95" max-x="280" max-y="207.0">
        <color-fill>
              <color red="255" green="255" blue="255" />
        </color-fill>
        <move x="-280.45" y="-206.95" />
        <set-secondary-fill index="1" />
        <line dx="560.95" dy="0.0" />
        <line dx="0.0" dy="413.95" />
        <line dx="-560.95" dy="0.0" />
        <line dx="0.0" dy="-413.95" />
  </shape>
                                        ......
                                        ......
```

Figure 2: A segment of the XML representation of a SWF file

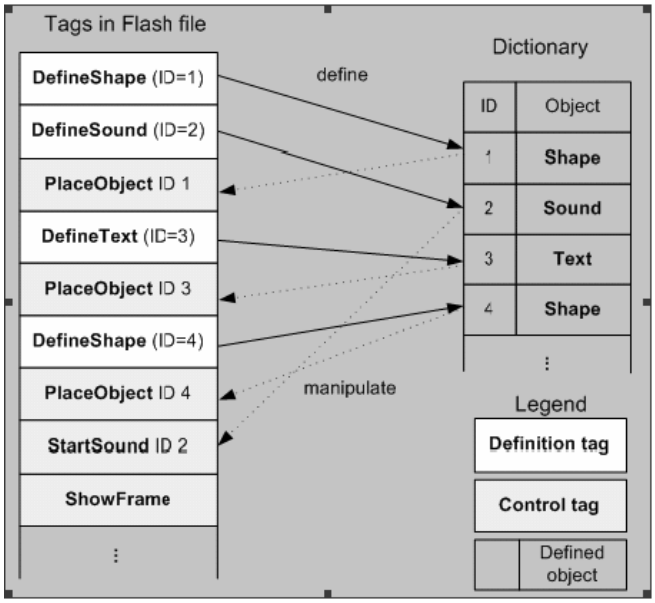

Figure 3: Structure of Macromedia Flash (SWF) file

There are two categories of tags in Flash files: *definition tags*, which are used to define various components in a movie, and *control tags*, which are used to manipulate these components to create the dynamic and interactive effect of the movie. For example, *DefineShape* and *DefineText* are typical definition tags, while *PlaceObject* (placing a component on the frame) and *ShowFrame* (showing the current frame) are typical control tags. All the components defined by definition tags are maintained in a repository called *dictionary*, from which control tags can access these

components for manipulation. The diagram shown in Figure 3 illustrates the interaction between definition tags, control tags, and the dictionary.

3.2 Multi-level Movie Indexing (Indexing Layer)

According to Section2, a Flash movie is mainly characterized by it heterogeneous media components, dynamic effects, and user interactions. In the indexing layer of FLAME, these three types of movie elements are modeled using the concepts of object, event, and interaction respectively. Specifically, *object* represents movie components as texts, videos, images, graphics, and sounds; *event* describes the dynamic effect of an object or multiple objects with certain spatio-temporal features; *interaction* models the relationships between user behaviors and events resulted from the behaviors. These three concepts are naturally at different levels: an event involves object(s) as the "role(s)" playing the event, and an interaction includes event(s) as the consequence of user behaviors. The features describing the objects, events, and interactions in a Flash movie are extracted by the Flash Parser from the XML representation of the movie (see Figure 1). The formal description of each concept and its features is presented as follows:

- **Objects**. A component object in Flash is represented by a tuple, given as:

$$object = <oid, o\text{-}type, o\text{-}feature>$$

where *oid* is a unique identifier of the object, *o-type* $\in$ {*Text, Graphic, Image, Video, Sound, Button*} denotes the type of the object, and *o-feature* represents its features. Obviously, the particular types of feature used to describe an object depend on the type of the object. Table 2 summarizes the most commonly used features for each type of object, which are extracted from the corresponding definition tags in Flash files either directly or through some calculations. For instance, keywords and font size (indicative of the importance of text) can be directly obtained from a text object, whereas the shape of a graphic object has to be deduced from the coordinates of the lines and curves constituting the contour of the graphic object, as long as the shape is simple enough. In Table 2, *Button* is a type of media component similar to a button control in common graphic interfaces, which can respond to mouse and keyword operation by executing certain movie actions.

Table 2: Features for various types of objects

Object	Features
Text	keywords, font size
Graphic	shape, color, number of borders
Image	size, color, texture
Sound	MFCCs (mel-frequency cepstral coefficients)
Video	features of a set of key-frames, motion vectors
Button	keywords, size

- **Events**. An event is a high-level summarization of the spatio-temporal features of object(s), denoted as:

$$event = < eid, \{action\}_N >$$
$$action = <object, a\text{-}type, a\text{-}feature>$$

where *eid* is a unique identifier of the event, followed by a set of *N* actions. Each action is a tuple consisting of an *object* involved as the "role" of the action, *a-type* as the type of the action, and *a-feature* as the attributes of the action. Each type of action is described by a particular set of features and can be applied to certain type(s) of objects (e.g., only graphic objects can be morphed). The relationships among action type, its applicable objects, and its features (which are derived from control tags) are summarized in Table 3. Two action types require more explanations: (1) "trace" is the action of an object following the movement of the mouse cursor in the movie frame; (2) "navigate" is an action of a Web browser being launched and directed to a specified URL, and therefore it does not involve any object.

Table 3: Features and applicable objects of actions

Action	Applicable objects	Features
show	all but sound	position, start/end frame
motion	all but sound	trail, start/end frame
rotate	all but sound	angle of rotation, location, start/end frame
resize	all but sound	start/end size, location, start/end frame
morph	graphic	start/end shape, number of frame
play	sound, video	current frame
trace	all but sound	closeness to mouse
navigate	N/A	target URL

Compared with the existing models for spatio-temporal features, such as spatial/temporal operators [3], and motion trajectory [5], the concept of event provides a more compact, semantics-flavored representation of these features because the predefined action types directly address the human perception on the dynamic effects of a movie. On the other hand, it is also powerful in terms of expressiveness, mostly because an event can have multiple actions. For example, a graphic object that is moving and resizing simultaneously over frames can be modeled by an event consisting of two actions describing the motion and resizing of the object respectively. Such a multi-action event can be also used to model the recursively embedded movies in a main Flash movie (cf. Section 2.1).

- **Interactions**. The concept of interaction describes the relationship between user behavior and the event caused by the behavior. Its formal definition is given as:

$$interaction = <iid, i\text{-}type, \{event\}_N, i\text{-}feature>$$

where *iid*, *i-type*, and *i-feature* represent the identifier, type, and features of the interaction respectively, and $\{event\}_N$ is a set of *N* events triggered in the interaction. The type of interaction indicates possible user actions, such as mouse click, button release, and key press. The features for each type of interaction are summarized in Table 4. For a button interaction, for example, an important attribute is the button event, such as button-release, button-click.

Table 4: Features for different interactions

Interaction Type	Features
button_click	button
button_press	button
button_release	button
key_down	key code
key_up	key code
key_press	key code
mouse_click	position, single/double, right/left button
mouse_press	position, right/left button
mouse_up	position, right/left button
mouse_drag	start position, end position, right/left button
mouse_move	start position, end position, right/left button

So far we have described the concepts of object, event, and interaction. The index of a Flash movie can be represented as the collections of objects, events, and interactions that are embodied in it, given as:

$$movie = <\{object\}_M, \{event\}_N, \{interaction\}_T>$$

where M represents the number of objects embedded in the movie, N is the number of events, and T is the number of interactions. The retrieval of Flash movies is conducted based on such multi-level features, as described in the next subsection. Please note that the features defined in Table 2, 3, and 4 are not necessarily exhaustive. For example, Table 3 includes only a set of basic events, and more (probably higher-level) events such as "collide" of two objects can be deduced from these basic events. However, we argue that completeness is not a strict requirement for the features used for retrieval (e.g., color histogram feature frequently used in image retrieval does not even model the spatial distribution of pixels), and defining an exhaustive set of (high-level) features is out of the scope of this "framework" paper, although it could be a good topic for the follow-up papers.

3.3 Multi-level Query Processing (Retrieval Layer)

As shown in Figure 1, the retrieval layer of FLAME consists of three *individual retrieval modules* for matching objects, events, and interactions based on their respective features. Note that these three modules do not work independently; rather, since interactions involve events which in turn involve objects, the retrieval modules for higher-level concepts need to "call" these modules for lower-level concepts. For example, to find movies containing, say, a rotating rectangle graphic object, the event retrieval module, which matches the "rotate" action, needs to cooperate with the object retrieval module, which matches the "rectangle" graphic object. On the other hand, as user queries usually target at movies and may involve features at multiple levels, a *multi-level query engine* is

designed to decompose user queries into a series of sub-queries for objects, events, and interactions processed by underlying retrieval modules, and then integrate and translate the results returned from these modules into a list of relevant movies. In the following, we describe each retrieval module (cf. Figure 1) and the multi-level query engine in details. In particular, we define some high-level functions as the summarizations of their functionalities.

- **Object retrieval module**. This module accepts the type and features of object as input, and returns a list of objects of the specified type that are ranked by their similarity to the given features. The retrieval process is summarized by the following function:

object-list: **SearchObject** (*o-type, o-feature*)

where *object-list* is a list of <*oid*, *score*> pairs with *score* indicating the similarity of each object to the feature specified by parameter *o-feature*. If *o-feature* is not specified, all objects of the given type are returned. Note that the "search space" of this operator covers all objects of every movie in the database. Thus, the returned objects may belong to different movies. The type of features used as search conditions varies with the object type, and naturally, various retrieval techniques are needed to cope with different features. Typically, IR approach is used for the keyword feature of text objects, CBR approach is used for the low-level features of video, image, and sound objects, and DB-style retrieval is suited for structured features such as the shape of graphic objects. Since the IR and CBR approaches are similarity-based, we set the similarity score of each returned object to the (normalized) similarity computed from these approaches. In case of a DB-style search, all similarity scores are set equally.

- **Event retrieval module**. To search events, we need to specify search conditions for not only actions but also objects as the "roles" of the actions.

event-list: **SearchEvent** (*a-type, a-feature, object-list*)

This function returns a list of events, each of which has at least one action that satisfies all the following three conditions: (1) the type of the action is equal to *a-type*, (2) the feature of the action is similar to *a-feature*, and (3) the object involved in the action is within *object-list*. If either *a-feature* or *object-list* or both of them are not given, the returned events are those with at least one action satisfying conditions (1) and (3), (1) and (2), or only condition (1). The *event-list* is of the same structure as *object-list*, except that the elements in it are events. Note that the ranking of events in *event-list* is fully determined by the similarity of their action features with *a-feature*, and is not subject to the ranking in *object-list*. The similarity of action features is computed based on the closeness of feature values to the specified values, but the technical details are omitted in this paper. Moreover, since only one action can be specified in **SearchEvent** (which is mainly for the simplicity of the function), the query for multi-action events is handled by firstly performing **SearchEvent** based on each desired action and then finding the events containing all the desired actions by intersecting multiple *event-list* returned from **SearchEvent**.

- **Interaction retrieval module**. The retrieval of interactions is conducted by the following function:

interaction-list: **SearchInteraction** (*i-type, i-feature, event-list*)

The semantics of this function, its parameters, and its return value are similar to those of **SearchEvent**. Each of

the interactions returned by this function must trigger at least one of the events specified by *event-list* (instead of all the events in *event-list*). Thus, to search for an interaction that causes multiple events simultaneously, we need to perform this function for each desired event, in which case *event-list* contains only one event, and intersect the results to find the interactions causing all the desired events.

- **Multi-level query engine**. The results returned by the individual retrieval modules described before are objects, events, and interactions, while the real target of user queries is Flash movies. The primary function of the multi-level query engine (cf. Figure 1) is therefore to translate the retrieved objects (and events, interactions) into a list of relevant movies, as defined by the following function:

movie-list: **Rank** (*object-list / event-list / interaction-list*)

The movies in *movie-list* are those containing the objects in *object-list*, and their similarity scores (and therefore ranks) are identical to their corresponding objects in *object-list*. Therefore, the order of the movies returned is the same as the order of their constituent objects in *object-list*, and if a movie has more than one objects in *object-list*, its order is decided by the one having a higher order. The semantics of **Rank** taking *event-list* or *interaction-list* as parameters is similar. On the other hand, it is common that a user query may specify multiple search conditions. To deal with such multi-condition queries, we need to merge multiple lists of movies retrieved based on each search condition into a single list giving the final ranking of similar movies. The **Merge** function is introduced for this purpose:

movie-list: **Merge** ($\{movie\text{-}list\}_N$, $\{weight\}_N$)

where $\{movie\text{-}list\}_N$ denotes N movie lists that are obtained based on different search conditions, and $\{weight\}_N$ contains the weight indicting the relative importance of each condition, which is preferably specified by users (if not specified, all the weights are assumed to be 1). Each movie in the returned movie list must appear in at least one input list, and similarity score of the movie (and thus its rank) is determined by the weighted sum of its similarity score in each input list (if it is not in a particular list, its similarity there is assumed to be zero). Please note that weighted sum is only one (and not necessarily the best one) of the many ways for merging multi-modal search results -- a separate research topic which is beyond the scope of this paper.

3.4 Sample Query Processing

The functions defined above, when used in combination, can support rather sophisticated queries of Flash movies. In this subsection, we describe the processing of some sample queries to demonstrate the usage and expressive power of these functions.

- Example 1: (Search by object)

A user trying to find Flash movies about the film “Lion King” through a poster (as an image file ‘lion-king.jpg’) can compose his query as: *Find all Flash movies that contain images similar to a poster of the film “Lion King”***.** This query can be processed as:

Rank (**SearchObject** (image, ‘lion-king.jpg’))

- Example 2: (Search by multiple objects)

A user who wants to search for MTV movie of a specific song from a known singer can probably express his query as: *Find all Flash movies that have keyword "John Lennon" and are accompanied by the song 'Imagine'. (Suppose the audio file of the song is 'imagine.mp3'.)* This query can be handled by combining the results of a search based on the keyword and another search based on the song:

Merge ({**Rank** (**SearchObject**(text, 'John Lennon')), **Rank** (**SearchObject**(sound, 'imagine.mp3'))})

- Example 3: (Search by event)

A query for movies containing the scene of "sunset" can be composed as: *Find all Flash movies that contain a red circle descending from the top of the frame to the bottom.* The processing of this query requires specifying both the desired object and its dynamic effect:

Rank (**SearchEvent** (motion, 'descending', **SearchObject** (graphic, 'red circle')))

- Example 4. (Search by interaction)

An interesting feature shared by many MTV Flash movies is that there is a button such as "Clikc To Play" on their first page, and users can click the button to start playing the music. We can therefore search some MTV movies by a query like: *Find all Flash movies that have a button labeled with keyword "Play" by clicking which a sound will be played.*

Rank (**SearchInteraction** (button-click, **SearchObject**(button, "play"), **SearchEvent**(play, **SearchObject** (sound))})

- Example 5. (Search by interaction and object)

Since some Flash movies as commercials quite often contain the link to the company, a query for Flash advertisements of, say, BMW cars, might be expressed as: *Find all movies that have keyword 'BMW' and a button by clicking which the BMW website will be opened in a Web browser.*This query is processed by a combination of all functions defined in Section 3.3:

Merge ({**Rank** (**SearchObject** (text, 'BMW')),
Rank (**SearchInteraction** (button-click, **SearchObject**(button), **SearchEvent**(navigate,'www.bmw.com'))))})

4. AN EXPERIMENTAL PROTOTYPE AND PERFORMANCE EVALUATION

4.1 The Prototype System

To demonstrate the feasibility of FLAME as a Flash retrieval system, an experimental prototype has been built based on FLAME. The prototype system has implemented the indexing and retrieval functions for most types of objects, events, and interactions supported by FLAME. Some functions (e.g., retrieval and indexing of embedded QuickTime video and MP3 audio) are left out either because they are difficult and time-consuming to implement or because they are not very critical compared with other functions. The prototype consists of a client component as a Web-browser accessible interface in charge of all user-system interactions, and a server component responsible for

the processing, storage, and retrieval of Flash movies. Between them is a Web Server (specifically, MS Internet Information Server), which acts as a proxy between the client and the server by translating user queries into internal commands that are executed by the server, as well as creating the HTML page displaying the retrieved movies in the user interface.

The great variety of queries supported by FLAME poses a challenge on user interface design. A good user interface should allow users to compose various types of queries conveniently and efficiently, as well as to display the retrieved Flash movies in an appropriate layout. In order to achieve good visual experience, we divide various query methods into two separate interfaces. The main interface shown in Figure 4 supports only keyword-based query. In this interface, a user can input the query keywords into the text box in the upper pane of the interface, and a list of Flash movies will be retrieved by matching text (objects) of movies with the query. The “thumbnails” of the retrieved movies ranked in descending order of their similarity (to the query) are displayed in the lower pane of the interface. The main interface contains a hyperlink labeled as “Advanced Search” pointing to the second interface (shown in Figure 5), which allows users to compose more sophisticated queries by specifying the objects, events, and interactions appeared in the desired movies. Such arrangement (of interfaces) is based on the fact that keyword is the most natural tool for users to express their information needs, a “rule of thumb” proved by many commercial search engines. Providing advanced query options in a separate interface allows professional users to conduct more sophisticated queries without confusing non-professional users.

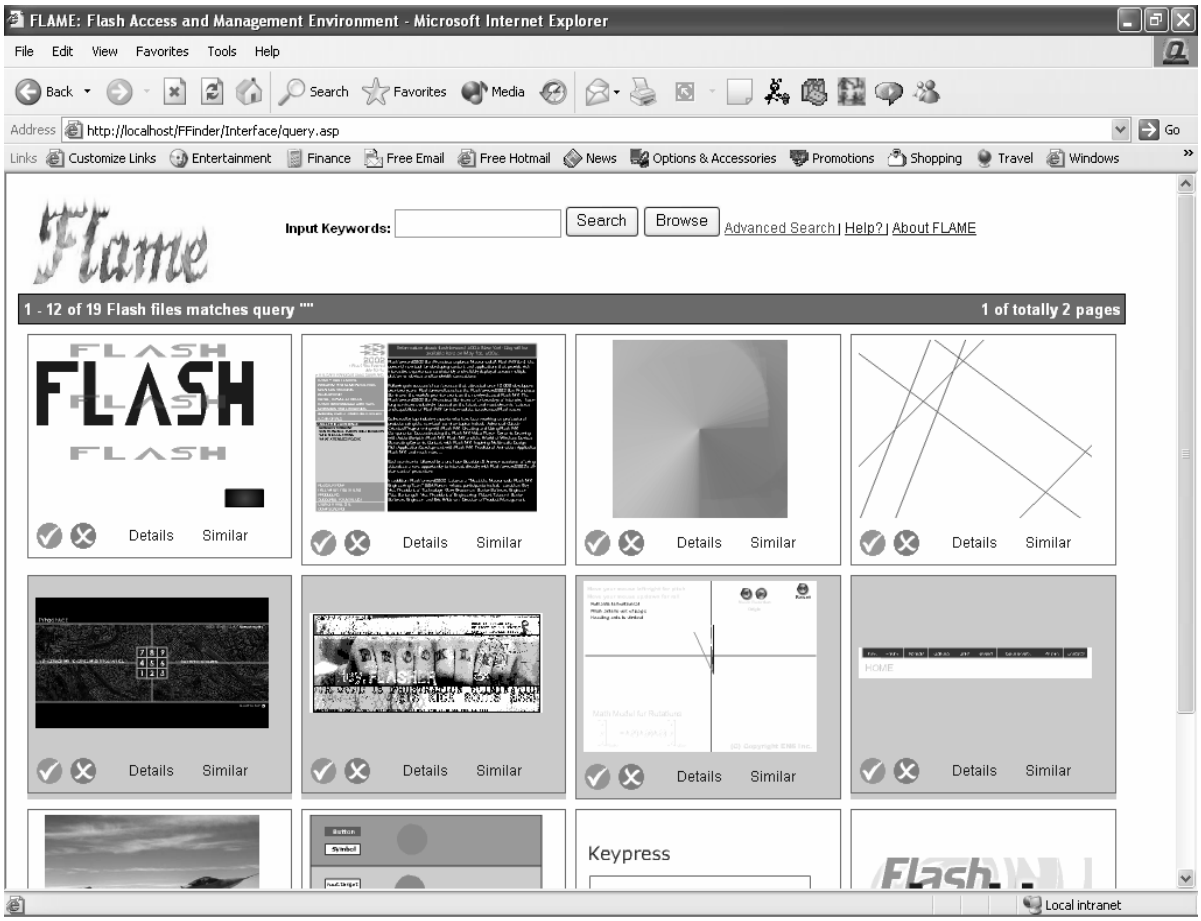

Figure 4: The main interface of the prototype

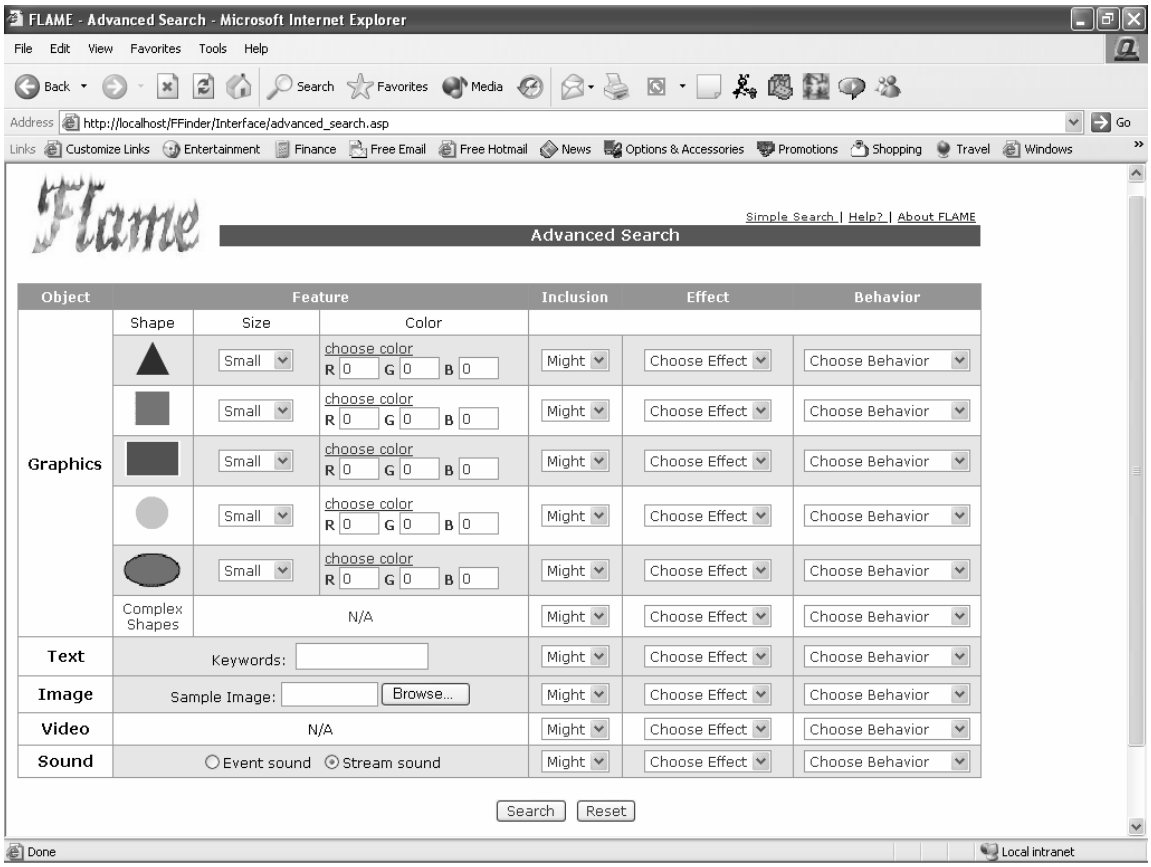

Figure 5: The “advanced search” interface of the prototype

The visualization of query specification methods is essential to the convenience and even productivity of users. In the “advanced search” interface, we adopt an iconic specification of sophisticated queries. As shown in Figure 5, users can specify various components (including text, images, graphics, videos, sounds) appearing in desired movies with respect to their features, dynamic effects, and user behaviors triggering the effects. The features of these components are specified in different ways. For example, graphic objects are specified by their shape (chosen from a list of shape icons), size (chosen from the drag-down box labeled as “Size”), and dominant color (chosen from a color palette), and the features of images are specified by submitting a sample image. Video objects can be any type of video that is supported by Flash animations. Note that "button" is not available as a type of object on the interface, since buttons are primarily for interactions which can be specified by "Behavior". The desired dynamic effects of each component in the movie can be designated using the drag-down box in the column labeled as “Effect”, and the user behavior triggering the event can be specified using the drag-down box in the column labeled as “Behavior”. The items listed in “Effect” are the types of dynamic effect, such as “motion”, “resize”, and the items in “Behavior” are the combinations of interaction type and feature, such as “button-click”, “mouse-drag”.

4.2 Preliminary performance evaluation

In our performance evaluation, we have studied two aspects of the system, namely the *usability* as well as the

retrieval accuracy. The first aspect measures how easily and conveniently an ordinary user can do a search in our system, while the second aspect measures in which degree the results returned from a given query match the user's need. The details are given below.

In the usability test, we have invited a group of 10 students in our department as the pilot users of our system to do random searches using various functions provided. Preliminary feedbacks have been collected from them, which are concluded as follows. First, all users quickly grasp how to use both the "simple" and "advanced" interface without any instructions. Second, most users prefer keyword search to advanced search, since the features used for advanced search, as they agreed on, are too primitive and detailed to compose semantics-rich queries. (The drawback lies in the lack of direct mapping from low-level features to semantics of Flash movies, as further discussed in Section 5.2.) Third, some users suggest adding the "sample-based" search function, which allows users to search for Flash movies similar to a sample movie. This raises an interesting research issue of measuring the similarity between two movies to explore in our future work.

The experiment on retrieval accuracy has to be conducted with several restrictions due to the system design. In particular, unlike most multimedia retrieval systems which work in a fully similarity-based manner, FLAME works in a mixture of declarative (binary) and similarity-based manner. For this reason, our system cannot be evaluated using purely similarity-based measures, such as precision/recall, neither can it be measured like a database system supporting declarative queries. (Though we can define an arbitrary similarity metric, say, as the sun of all the similarities on components, such trivial metrics do not make sense to users who look for semantically relevant Flash movies.) Moreover, since in our system a query has to be expressed with primitive structural features, which do not readily map to semantics, it is hard for us to construct a testing collection with meaningful sample queries and ground truth. Sample queries like "Finding me a Flash movie with a moving, green rectangle" are trivial from a practical point of view.

Despite these restrictions, we manage to conduct a preliminary experiment with the help of the same group of students as users. Specifically, we collect 200 simple Flash movies from an online Flash gallery (specifically, Flash Kit) as our test data. Since our system is currently not apt for semantics-rich queries, our selections are mostly simple Flash movies collected from the category of "Basics" and "Tutorials" with size below 20KB. All the 10 students are trained so that they know how to use the system but with no idea on the underlying retrieval techniques. In the experiment, we randomly divide the 200 testing movies into 10 even groups, with 20 movies in each group, and assign each group to a distinct user. Each user is asked to look at the movies assigned to him in details, and compose a query targeted at each movie and submit it to our system, which has been populated with the 200 testing movies in advance. The query may consist of keywords, but more often it specifies the components that should appear in the desired Flash movies. For each query, the system returns the 200 movies ranked by the descending relevancy to the query, among which only the movie targeted by the user is regarded as relevant. For each query, we record the position of the target movie in the list of the returned movies, and therefore totally 200 positions are recorded for the total of 200 queries. The distribution of the positions of the target movies are plot in Figure 6.

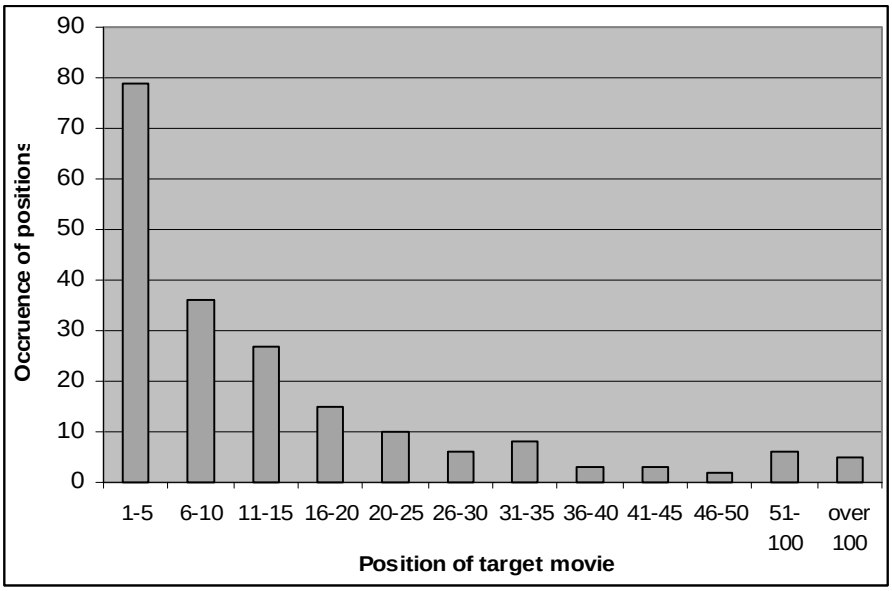

Figure 6: Statistics on the positions of target image over 200 queries

As we can see, in about 40% of the queries the target movies appear among the top 5 results. Moreover, in 147 out of the totally 200 queries, the target movies appear in the first page (each page contains 16 results by default) of the returned results. These statistics show the effectiveness of our system in composing queries and retrieving relevant movies, despite that the queries are quite primitive. By studying some "failure cases" where a target movie is far behind in the result list, we discovered that most failures are because a target movie is too complex to be effectively described by the query. For example, a motion in a Flash movie is driven by some complicated script currently not indexed by our system, so the user has no means to specify this motion in the query.

Admittedly, our current experiment is rather primitive and restrictive, especially in terms of the size and complexity of the testing collection and the level of semantics embodied in the queries. However, given that our testing queries are "target search" (only one relevant result for each query) rather than "category search", 200 movies constitute a decent testing collection. The lack of complexity of the data and queries is due to the intrinsic problem of our current retrieval framework in handling queries of high-level semantics.

5. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

This paper has investigated the problem of content-based Flash retrieval, which is critical to better utilization of the proliferating Flash resource but unfortunately has not been noticed by the research community. In this paper, we have presented a deep discussion of the research issues regarding Flash retrieval and compared it with related works. As our major contribution, a generic framework FLAME has been proposed for semantic retrieval of Flash movies, which is based on the extraction, representation, and query processing of the *expressive movie elements* including heterogeneous components, dynamic effects, and the way of user interactions supported. A prototypical Flash retrieval system implemented based on FLAME has been described, and an initial study involving some pilot

students on user acceptability and retrieval accuracy has been conducted. Although the proposed FLAME framework covers the key aspects of content-based Flash retrieval, it also opens up a number of research issues for further studies, which are itemized below.

5.1 Modeling Action Logics

From version 4, Macromedia Flash starts to support a powerful action model which is capable of expression evaluation, conditional branch (to frames), looping (of frames), and other functions, based on a stack machine and some primitive operators such as pop, push, add, multiply, and branch. In this sense, Flash exhibits features of a logically complete programming language (actually, some part of a Flash file can be very similar to a segment of assembly code). This action model allows a Flash movie to support more sophisticated interactions, such as showing different graphics according to the position of the mouse cursor. Thus, a good retrieval approach should be able to understand complex action logics in user interactions and allow users to query over them effectively. However, understanding action logic is essentially the problem of automatic program comprehension [33], which is itself an unsolved problem in software engineering. Moreover, the action model in Macromedia Flash v.5 is capable of talking to back-end database and thereby creating dynamic movie content, which leads to another host of research issues.

5.2 High-order Semantics

As discussed in Section 1, the success of Flash retrieval largely depends on the degree in which the content description of Flash movies meet the rich semantics intended by user queries. Admittedly, the movie description used in FLAME does not reach to the level of semantics preferred by users, although they are already superior to the low-level audio-visual features used in image and video retrieval systems. In our view, the high-order semantics of Flash movies should be extracted incrementally by a synergy of both “offline” processing and “online” learning. The offline process “guesses” the movie semantics from its components based on some ad-hoc models specified by experts or concluded from a statistical study. A good reference on capturing high-level video semantics from lower level properties is the work of Al-Khatib[2]. However, the semantics obtained by offline processing can be unreliable, incomplete, or even misleading, which must be compensated by the online learning that deduces the movie semantics from the evaluations explicitly given by users or implied by their behaviors.

Since the semantics is encoded into a Flash movie by its developer, the best time for indexing a Flash movie is actually at its authoring stage. If the Flash authoring tools allow developers to add semantic descriptions into the movie files conveniently, the semantic retrieval of Flash movies can become a straightforward task based on the self-descriptive movie files. This vision, however, demands the support of the Flash format and its authoring tools.

5.3 Human in the Loop

Currently, users are involved only in the query stage of the retrieval cycle in FLAME. A more active role played by users, especially on evaluating the relevance of retrieval results, is preferred for the following two reasons. First, given the complexity of Flash and the ambiguity of its semantics, it is unlikely that a user can precisely and

completely specify his need by an initial query, and a series of interactions is desirable for the user to refine his query. Second, useful knowledge such as the semantic meanings of Flash movies can be deduced from user interactions.

User involvement may take the form of short-term interaction, long-term interaction, or an integration of both. Short-term interaction is what we normally describe as "relevance feedback", a technique for improving the retrieval results based on user evaluations on previously retrieved results, which has been successfully used in text-based information retrieval [26] and image retrieval [24]. While short-term interaction concerns with a user's *immediate* request, long-term interaction helps a retrieval system to establish "user profile" characterizing the relatively *stable* preference of a user, e.g., "which type of movies he prefers, cartoon or MTV?" Such user profiles can be investigated and utilized in the retrieval process to adapt the returned movies to better satisfy the preferences of individual users.

5.4 Web-based Flash Search Engine

Unlike other types of media data that are available both "online" and "offline", Flash is born for the Web and circulated almost exclusively on the Web. Thus, a Web-based Flash search engine is naturally a "killer application" of Flash retrieval technology, which, however, may exhibit certain features not addressed in FLAME. First, Flash movies embedded in web pages are usually accompanied by "context text" such as URLs and surrounding text, which can be utilized as semantic descriptions of Flash in the search. Second, like the *PageRank* technique used in Google, hyperlink structure can be exploited to estimate the "authority" or "popularity" of Flash movies to achieve better user satisfaction. Third, given the popularity of browsing operation, a taxonomy is greatly desired to support browsing/navigation of Flash movies by subject categories (e.g., cartoons, commercial, games, MTVs), a tool similar to the "topic directory" on many portal websites such as Yahoo!.

5.5 Performance Evaluation Criteria

Research works are driven by good performance evaluation criteria. A widely used criterion in the area of multimedia (including text) retrieval is the "precision/recall" criterion [26], which examines the ratio of *relevant* results returned among all the returned results and among all relevant candidates. Although this criterion is applicable to Flash retrieval, its underlying concept of "relevance" needs to be redefined in a way consistent with the human perceptions of Flash movies. Due to the multifaceted characteristics of Flash (components, dynamic effects, interactivity) and human subjectivity, a fair criterion for Flash "relevance" is as difficult to define as that for images and videos. Another related issue is to establish a standard test dataset of Flash movies as a benchmark based on which various retrieval methods can be evaluated and compared. As discussed in section 4, such a test dataset must be sufficiently large, diverse and balanced in content, and more importantly, it should be associated with a set of sample queries and the ground-truth that tells the correct results for each sample query, like the TREC collection [32] for text retrieval and TREC Video Collection for video retrieval.

5.6 Management of Flash Movies

In addition to pure retrieval, there also exist other research issues of managing Flash resource, such as storage, browsing/navigation, classification, and clustering, which influence the experience and productivity of users in their daily interactions with Flash movies. For example, an online Flash repository may need an automatic tool for classifying Flash movies into generic categories (e.g., MTVs, advertisements, cartoons), so that its users can find their desired movies efficiently. Moreover, a Flash author would like to have a "personal Flash assistant" that memorize all the Flash movies he has ever viewed or created. Later, when he is working on a new Flash movie, the assistant can prompt with some old movies with similar style or content for his reference. An example multimedia management tool is the Media Agent system [17] for organizing personal media files, which serves as a good reference for future works on Flash management.

5.7 Other Multimedia Presentation Formats

Many multimedia presentation formats have emerged in the past decade, such as Microsoft PowerPoint, SMIL (Synchronized Multimedia Integration Language) [31], whose popularity creates the need of specific retrieval tools for them. To our knowledge, however, only limited work has been devoted to the retrieval of these media formats, such as the SliderFinder system [22] designed for retrieving PowerPoint and Lotus Freelance Graphics.

Since these media formats are similar to Flash in one or more aspects, the proposed FLAME framework can be tailored and/or extended for their retrieval tasks. For example, a PowerPoint file also accommodates various media components, but supports limited dynamic features (displaying components with some dynamic visual effects) and interactivity (turning page). Hence, the retrieval of PowerPoint can be fit into a "reduced version" of FLAME with simpler models of events and interactions. It is probably more straightforward for FLAME to be tailored to support the retrieval of SMIL presentation which, like Flash movies, is an interactive and multimedia-rich presentation.

REFERENCES

1. S. Adali, M.L. Sapino, V.S. Subrahmanian, "An algebra for creating and querying multimedia presentations," ACM Multimedia Systems, 8(3): 212-230, 2000.
2. W. Al-Khatib, A. Ghafoor, "An approach for video meta-data modeling and query processing," in Proc. of ACM Multimedia, pp. 215-224, 1999.
3. S.S.M. Chan, and Q. Li, "Developing an object-oriented video database system with spatio-temporal reasoning capabilities," in Proc. 18th Int. Conf. on Conceptual Modeling, LNCS 1728, pp. 47-61, 1999.
4. S.K. Chang, Q.Y. Shi, C.Y. Yan, "Iconic indexing by 2-D strings," IEEE Tran. Pattern Anal. Machine Intell., 9(3): 413-428, 1987.
5. S.F. Chang, W. Chen, H.J. Meng, H. Sundaram, D. Zhong, "VideoQ: An automated content based video search system using visual cues," in Proc. ACM Int. Multimedia Conf., pp. 313-324, 1997.
6. Corel Gallery. http://www.corel.com/galleryonemillion/

7. R. Elmasri and B. Navathe, Fundamentals of Database Systems (2 Edition), The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

8. Flash Kit. http://www.flashkit.com/index.shtml

9. M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, "Query by image and video content: The QBIC system," IEEE Computer, 28(9), pp. 23-32, 1995.

10. J. Foote, "An overview of audio information retrieval," ACM Multimedia Systems, 7: 2-10, 1999.

11. B. Furht. Handbook of Multimedia Computing. CRC Press, Sep. 1998.

12. V.N. Gudivada and Raghavan, "Design and evaluation of algorithms for image retrieval by spatial similarity," ACM Trans. Information Systems, 13(2): 115-144, 1995.

13. A. Hauptmann, T.D. Ng, R. Baron, W. Lin, M.Y. Chen, M. Derthick, M. Christel, R. Jin, R. Yan, "Video Classification and Retrieval with the Informedia Digital Video Library System," Text Retrieval Conference, Gaithersburg, MD, November, 2002

14. R. Hjelsvold and R. Midtstraum, "Modeling and querying video data," in Proc. 20th Int. Conf. Very Large Database, pp. 686-694, 1994.

15. JavaSWF. http://www.anotherbigidea.com/javaswf/

16. T. Lee, L. Sheng, T. Bozkaya, G. Ozsoyoglu, M. Ozsoyoglu, "Querying multimedia presentations based on content," IEEE Trans. Knowledge and Data Engineering, 11(3):361-387, 1999.

17. W.Y. Liu, Z. Chen, F. Lin, R. Yang, M.J. Li, and H.J. Zhang, "Ubiquitous media agents for managing personal multimedia files," in Proc. ACM Multimedia Conf., pp. 519-521, 2001.

18. Y. Lu, C.H. Hu, X.Q. Zhu, H.J. Zhang, and Q. Yang, "A unified framework for semantics and feature based relevance feedback in image retrieval systems," in Proc. of ACM Multimedia Conf., pp. 31- 38, 2000.

19. Macromedia, Inc. www.macromedia.com.

20. Macromedia Flash Player adoption statistics. www.macromedia.com/software/player_census/flashplayer

21. Macromedia Flash File Format (SDK) http://www.macromedia.com/software/flash/open/licensing/fileformat/

22. W. Niblack and W. SlideFinder, "A tool for browsing presentation graphics using content-based retrieval," in IEEE Workshop on Content-Based Access of Image and Video Libraries, June 22 - 22, 1999, Fort Collins, Colorado.

23. G. Ozsoyoglu and R. Snodgrass, "Temporal and real-time databases: a survey," IEEE Trans. on Knowledge and Data Engineering, 7(4): 513-532, 1995.

24. Y. Rui, T.S. Huang, M. Ortega, and S. Mehrotra, "Relevance feedback: a power tool in interactive content-based image retrieval," IEEE Trans. Circuits and Systems for Video Technology, 8(5):644--655, 1998.

25. Y. Rui, T.S. Huang, and S. Mehrotra, "Content-based image retrieval with relevance feedback in MARS," in Proc. IEEE Int. Conf. on Image Processing, pp. 815-818, 1997.

26. G. Salton and M.J. McGill, Introduction to Modern Information Retrieval. McGraw-Hill Book Company, 1983.

27. H. Samet, "The quadtree and related hierarchical data structures," ACM Computing Surveys, 16(2):187-260, 1984.

28. A. Smeulders, et al, "Content-based image retrieval at then end of the early years," IEEE Trans. Pattern Analysis and Machine Intelligence, 22 (12): 1349-1380, 2000.

29. J.R. Smith, S.F. Chang, "VisualSEEk: a fully automated content-based image query system," in Proc. of ACM Multimedia Conf., Boston, MA, pp 87-98, 1996.

30. J.R. Smith, S.F. Chang, "Visually searching the Web for content," IEEE Multimedia Magazine, 4 (3): 12-20, 1997.

31. Synchronized Multimedia Integration Language (SMIL). http://www.w3.org/AudioVideo/

32. Text Retrieval Conference (TREC) Data. http://trec.nist.gov/data.html

33. S. Woods and Q. Yang, "Program understanding as constraint satisfaction," in Proc. 2nd Working Conf. Reverse Engineering, pp. 314-323, Canada, 1995.

34. J. Yang, Q. Li, and Y.T. Zhuang, "Octopus: aggressive search of multi-modality data using multifaceted knowledge base," in Proc. 11th Int. Conf. on World Wide Web, pp. 54-64, Hawaii, USA, May 2002.