In Support of Similarity Measures

Deborah Swanberg Ramesh Jain

deborah@cs.ucsd.edu jain@cs.ucsd.edu
Department of Computer Science and Engineering
University of California at San Diego
La Jolla, CA 92093-0114

Abstract

Research is well underway to investigate the appropriate use of similarity measures across complex objects such as faces and color histograms. This research is crucial in producing systems that support content-based retrieval. To support the successful integration of the these techniques into database systems, support tools and object definitions are needed that can assist with the data modeling process that segments, stores and accesses the data. Here, we define our first efforts to defining the classes and operations inherent in any segmentation process. We define a generic processor structures these essential processes. Finally, we define a set of video objects we have defined to support the segmentation of broadcast video streams.

1 Introduction

Current systems and research into content-base multimedia databases is focusing on questions that ask "find me the art images that look like this", or "find me the people who look like this". When an image query is submitted to the database, images that match the specified description are returned for the user. Future systems will not only locate a specific piece of data, they will also analyze it, with questions such as "where is the ice moving quickly?", or "what is the tumor growth rate?". Crucial in these systems, then, is the need to derive, manage, and compare complex objects such as art pictures, faces, and temporal real-world objects such as ice floe or tumors.

Today, many research techniques are under investigation to support these systems. Appropriate data representations are essential, and research ranges for domain specific models [1, 2] to domain independent models using image features [3]. Fuzzy logic and similarity measures are under investigation for categorizing and comparing data.

To support the integration of this work into the computer systems, systems are under development to build databases the utilize image features, similarity measures, and indexing techniques [4]. This work is necessary to provide an architecture that can use these techniques, and to simplify the required complex data modeling and analysis. The work presented here discusses the data modeling process, and presents a base system that can be used to segment any data stream. On top of this, several models for segmenting video data have been defined, some of them are presented here.

2 Previous Work

In a previous paper [5], I described a hierarchical model for broadcast news video streams. Steve Smoliar [6] has developed a system to segment a news stream based on these models. We have developed similar techniques, and won't report them here. Rather, here we continue the discussion about the database segmentation problem, but describe some tools that we have developed to support our efforts in constructing models for video objects.

3 The Designing Process

The process of designing models for segmenting image and video data has multiple steps, and is a complex problem. Perhaps only the QBIC project [4] has begun to address the question of how systems for content-based retrieval are designed. The steps are presented here to outline the operations that must be developed by any supporting system. We define these routines, and incorporate them in a general model of the parsing process, described below. Due to the complexity of this problem, tools are needed to support the development of each of these items.

- Identify the desired objects

1058-6393/95 $4.00

- Determine the representation of the objects in the database
- Determine the segmentation process to locate the objects in the database.
- Determine the segmentation process to derive the representation of the objects from the data.
- Determine the similarity measures to apply across the representation

4 The Parsing Process

The parsing process incorporates the methods developed in the design process to segment an image or video stream for important information. Generally, the parsing process will step through the following components.

- Segment the Data: Layers of processing
- Identify, and locate the desired objects
- Calculate the unique features of the data and objects
- Store the objects, with relative pointers

If the segmentation routines, the representations, and the similarity routines are known, models can be constructed to support the data parsing process.

5 Generic Parsing Process

In [5], I presented a base model for a generic object functioning in the segmentation/retrieval environment. The models have four components. Let $M = \{D, O, S, T\}$ where

- D is the data model of the object, i.ethe representation of the the object to located in the input data being analyzed.
- O is the operator(s) that derive the object representation from the raw data.
- S is the similarity measure that compares the model to the object instance.
- T is the threshold or range of acceptable values from the similarity measure of the object.

6 Modeling Language

To assist in the construction of these models to support video segmentation, we have defined a language that developers can use to structure their models.

The primary parsing model defnes a finite automaton engine, that compares a finite automaton model to an input stream. It is generic, in that it can be used as a base engine to develop any model over any datastream that can be represented with a finite automaton. The parser identifies a *base element* to be mached in the automaton, and compares the base element to the objects associated with the different arcs. It *matches* the object with the best fit, and traverses that arc to the associated state. In addition, we extended the parser algorithm so that when an arc is *traversed*, the parser invokes a function of the arc that signals that its object matched; we use this to store the object instance in the database.

In summary, then, there are three operations which must be defined for working with any automaton parser, and specifically, in working with our parser implementation. These are: a method to identify the *base elements* from the input stream; *matching* routines to compare the modeled objects to the input stream; and *traversal* operations when the arc is traversed.

If these operations are available, the developer can use the parser objects to construct models for segmentation. The model can either be constructed explicitly using the parser language, or the language can be used to define a higher level language more appropriate for the data. We developed a higher level language to support the segmentation of video objects from a video stream. The parser language is described below, and the video language is described in the following section.

6.1 Parser

The language defines generic *object*, *arc*, and *state* terms, which collectively define the automaton to segment the input data.

An *object* serves as the alphabet in the automaton. It must be able to compare itself to the input stream and return a measure of similarity to some component of the input stream. At the lowest level of the parser, a generic object model is implemented to have no data, and only one fuction defined - a *matching* function.

The *arcs* contain both the objects to be compared to the input stream, and the next state to be entered, should the object match. The arcs also point to the *traversal* routine that is invoked by the parser when the the arc's object matches, and the arc is traversed to the next state.

The *states* are defined to be contain list of arcs. In our development, we found that it was neccessary to extend the traditional automaton definition of state so that it provided a default state. This was necessary because there were cases where an explicit model could not be developed to describe an object, but the type could be assigned because it didn't match any other objects that could be modeled. This occurred in modeling CNN newreel shots and commercials.

The parser engine begins segmentation at an initial state, specified by the developer. The parser steps through each of the arcs, invoking the match function on the objects specified in the arcs. Currently, the parser determines which arc to traverse by stepping through the arcs one by one, looking for a "match", and it accepts the first match, as defined by the objects. When a match is found, the parser does not compare the input stream to any other arcs; the parser traverses the arc to the next state. If no arcs match, the default arc is chosen. If no default is present, and no match is found, and error is returned to the invoking level. Once the matching arc is found, the arc is "traversed": the arc operation is executed, and the move is made to the new state.

An alternative parsing algorithm could define matching to return a measure of similarity from every object. Then, the parser would compare all the arc objects to the input stream, and select the object with the highest similarity. While this incurs more processing, it may be a better algorithm when definitions of similarity are not precise. When no matching arc is found, the default arc is selected. Yet another alternative might be to devolop a system that supports parallel processing of the data, to avoid redundant data access and computations, such as those used by blackboard architectures.

The matching function of the object performs two of the necessary functions described in section 5: it segments the data from the input stream, supplied by the parser; and then it compares that segmented data to the data it is trying to match. The object returns a value of either true or false that specifies whether or not the object matched.

New objects, with different similarity routines and matches are incorporated in an object-oriented manner by defining object classes based on the existing and defined objects. It is the responsibility of each object to know how to segment the input data to derive the representative information for comparison to the data in it's model.

7 Parsing the Video Objects

- Segmenting: Breaking a video stream into segments of contiguous frame, based on some criterion, such as a shot from a video, or an object being present, such as a football player.
- Labelling or Typing: Give a video stream or segment of a stream, label it, or type it according to some prespesified categorization.
- Grouping: Given several segments, group them into larger segments. This is different than forming sets that have the same label, it is recombining contiguous segments to form a larger segment.

The system must be able to capture the same object being represented from different streams at the same time. In this case, an object may appear in several different streams, such as a story being reported over several different news casts, or a football game being captured from several different cameras, where a player is captured in several different video streams. In a similar manner, it is possible for the object to reappear over time.

The system must be able to capture the object being represented at different times, existing through disjointed times.

7.1 Video Objects

Representing the video stream and it's segmentation process with a generic parser representation would be quite tedious, and often redundant. So, using the parser object class, we defined new subclasses that supports the segmentation of video streams. These classes are episodes, shots, frames, and regions. The classes, and their relationships, are described in detail in [5]. In brief, the video stream is composed of individual frames. A shot is defined by a camera change. Shots can be typed according to different features, such as motion, or semantic labels that identify the type of shot, such as anchorperson. An episode is defined over a sequence of contiguous shots.

Our video classes are used to construct models of the video stream to segment the data into components meaningful to a user, such as CNN segments like Dollars and Sense, half hour episodes, anchorperson shots, etc. Due to limited space, only the method of developing classes to model different shots is explained below. Similar methods are used to model objects such as frames and regions. The episodes are more a more complex object, and are modelled by augmented transition networks.

7.1.1 Typed Shots

Shots are the essential element of our segmentation process, and form the core of our modeling mechanism. They are derived from the input stream, typed according to some criterion, and organized in models of episodes.

The shot classes are subclassed from the generic parser object, and so support the same operation types for segmentation, similarity, and traversal. When the parser calls a shot object to match itself to the input stream, the shot object first invokes a segmentation operation: it parses the shot from the video stream, and then invokes routines to derived the desired data from the shot. Finally, it invokes the similarity routine to compare the derived data to it's internal model, and returns a true or false value indicating the match. When the arc containing the shot object is traversed, the segmented instance is stored.

The routine to segment the shot from the video stream is shared by all shot models. Currently, the data is presegmented into shots manually (we are working with Arun Hampapur to integrate his techniques to automatically segment the shots from a video stream), and stored in a database. The data segmentation operation then simply calls the database for the next segmented shot. Segmenting the shots in advance is justifiable, as the process is independent of higher level analysis, so the results are not changed. In addition, the shots form the essential base for the analysis and will be segmented from the video stream regardless of the model, so no processing is wasted by performing the operation in advance.

Two shot classes have been defined. One shot class models a shot with individual models of frames that must be matched to frames in the shot instance from the video stream. To construct this object, the user defines the frame objects, and assigns them to the shot object. When the parser invokes this object to compare itself to the video stream, the matching routine provided for this class locates the shot, and passes the shot to each frame for matching. All of the frames must match in order for the shot to match. We use this shot class to model shots that contain specific graphics in specific frames, such as the CNN Top-of-the-Hour shot.

The other shot class provides users with a more basic model, rather than specifying the data components, the user specifies the matching routine that will operate over the shot instance from the video stream. This allows the users to build models and comparison routines using other systems, and to integrate those routines into the segmenation process. For example, we use this shot class to integrate shot models that have been developed in Khorors.

8 Architecture

We have used several different application platforms to develop the models of video streams, and support the application. In order to develop models for some of the shots, we have found Khoros to be an essential tool. It allowed us to experiment very easily with different operations and their parameters for developing shot operators. However, khoros did not support the hierarchical development techniques that were need to construct submodels and templates that could be invoked by the higher level routines. Consequently, we developed our own parser and language that integrates operations developed from khoros.

We are working with the UniSQL database, and are developing models of video streams and the object classes to be stored in the database, which correspond to the objects parsed during the segmentation.

We have also developing video stream models that function independent of the devices containing the video. This allows us to access digitized video, computer controlled laserdiscs, and a computer controlled video tape via the same interface. As a result, when the video stream is modeled in the database, a pointer is stored to the location of the data, and device controller, but the stream representation, and the objects derived from it, are modeled independently of the actual video media.

9 Example Model

Below is a sample code that models the CNN Headline News segment at the top of the hour. It includes a reference to the FlyingLogoClip, which uses an user defined operation developed in Khoros.

```
VARC (SatelliteArc, SatelliteClip, ENDSTATE);
ARCLIST1 (HNEAfterWeatherArcs, SatelliteArc);
STATE (HNEAfterWeatherState, HNEAfterWeatherArcs,
&HNEunknownArc);
VARC (HNEWeatherArc, HNEWeatherClip,
HNEAFTERWEATHERSTATE); VARC (HNENextArc,
HNENextClip, HNEPRENEWSREELSTATE); VARC
(HNEAnchorPersonArc, AnchorPersonClip,
HNEPRENEWSREELSTATE);
ARCLIST2 (HNEAfterNewsReelArcs, HNEAnchorPersonArc,
HNENextArc);
STATE (HNEAfterNewsReelState, HNEAfterNewsReelArcs,
&HNEunknownArc);
VARC (HNENewsReelArc, NewsReelEpisode,
HNEAFTERNEWSREELSTATE);
ARCLIST2 (HNEPreNewsReelArcs, HNENewsReelArc,
HNEWeatherArc); STATE (HNEPreNewsReelState,
HNPreNewsReelHArcs, &HNEunknownArc);
VARC (HNETopOfHourArc, TopOfHourClip,
HNEPRENEWSREELSTATE);
ARCLIST1 (HNEAfterLogoArcs, HNETopOfHourArc);
STATE (HNEAfterLogoState, HNEAfterLogoArcs,
&HNEunknownArc);
```

```
VARC (HNFlyingLogoArc, FlyingLogoClip,
HNEAFTERLOGOSTATE);
VARC (HNFlyingLogoArc, FlyingLogoClip, ENDSTATE);
ARCLIST1 (HNEState1Arcs, HNFlyingLogoArc); STATE
(HNEpisodeState1, HNEState1Arcs, &HNEunknownArc);
STATE (HNEEndState, NULL, NULL); EPISODE
(HeadlineNewsEpisode, INITSTATE, States);
```

10 Future Work

There are many directions to pursue with this work. From the example above, it can be seen that building models of the video stream is messy and difficult to follow, although it is an improvement over an initial implementation that did not define video objects at all. Future work could address user interfaces for defining each of the objects: episodes, shots, frames, regions.

Currently, the existing implementation is written in C, and emulates C++. Classes are instantiated with macro definitions. A proper implementation in C++ would provide an environment more familiar to users. We are currently in the process of porting this code to C++.

In terms of the parsing algorithms, boolean values were used to represent the similarity between a model of an object, and an instance; in the future more sophisticated similarity measures will need to be used.

Finally, there may be more appropriate architectures to support the potential for parallel processing, and for back-tracking in a failed analysis. In the larger sense, there is work to be done to integrate these models closely with the storage models used in the database.

11 Conclusion

Originally, we defined a set of models that could be used to segment broadcast video news. Steve Smoliar implemented these techniques to show their potential. Here, we have described the operations necessary to segment video data for a database, and have described an architecture that we have developed to assist developers in creating models for the segmentation of video streams. Specifically, we have defined a base generic parser that can be used to segment any data stream. Then, as a subclass of the various parser objects, we have defined specific video classes of episodes, typed shots, frames, and regions. These tools are a first step in developing an architecture that integrates the data segmentation environment with the data storage and query environment.

References

[1] A. Gupta, T. Weymouth, and R. Jain, "Semantic queries in image databases," in *2nd Working Conference on Visual Database Systems*, (Budapest, Hungary), pp. 204–218, IFIP WG 2.6, October 1991.

[2] D. Swanberg, T. Weymouth, and R. Jain, "Domain information model: an extended data model for insertions and query," in *Proceedings of the Multimedia Information Systems*, (Pheonix, Arizona), pp. 39–51, Intelligent Information Systems Laboratory, Arizona State University, Feb. 1992.

[3] M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," pp. 586–591, 1991.

[4] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, and C. Faloutsos, "The QBIC project: Querying images by content using color, texture, and shape," Tech. Rep. RJ 9203 (81511), IBM Research Division, Almaden Research Center, 650 Harry Road * San Jose, California 95120, Feb. 1993.

[5] D. Swanberg, C.-F. Shu, and R. Jain, "Knowledge guided parsing in video databases," in *Image and Video Processing Conference; Symposium on Electronic Imaging: Science & Technology*, vol. 1908, (San Jose, California), pp. 13–24, IS&T/SPIE, Feb. 1993.

[6] S. W. Smoliar and H. Zhang, "Content-based video indexing and retrieval," *IEEE MultiMedia*, vol. 1, pp. 62–72, Summer 1994.