〔非公開〕

ＴＲ－Ｍ－００２１

Querying and Indexing Multimedia Data

ティモシー　シェパード
Timothy Shephard

田　中　昭　二　　井　上　誠　喜
Shoji TANAKA　　Seiki INOUE

１９９７．４．１６

ＡＴＲ知能映像通信研究所

Technical Report: Querying and Indexing Multimedia Data

Tim Shephard
Department 3
Media Integration and Communications Laboratories
ATR

April 16th, 1997

Contents

1 Introduction **4**

2 Query Interface **5**

2.1 Introduction 5

2.2 Problem 5

2.3 Other Approaches 5

2.4 This Approach 6

2.4.1 Introduction 6

2.4.2 Expression Model 6

2.4.3 Query Tree 8

2.5 Description of Software 9

2.6 Implementation 9

2.7 Client Files 10

2.7.1 DefinitionList.Java 10

2.7.2 queryInputStream.java 11

2.7.3 attribType.java 11

2.7.4 paintClass.java 12

2.8 Server Files 12

2.8.1 cserver.c, cserver.h 12

2.8.2 SocketHandler.cc, SocketHandler.hh 12

2.8.3 server.cc, server.hh 13

2.8.4 queries.cc 13

2.9 Database Files 13

2.9.1 schema.cc 13

2.9.2 expression.h 13

2.9.3 makeDB.cc 14

3 **Video Indexing** **14**

3.1 Introduction 14

3.2 Problem 14

3.3 Other Approaches 15

3.3.1 Objective versus Subjective, and the Common Sense problem 16

3.3.2 Time information 16

3.4 This Approach 17

3.4.1 N-ary Search 17

3.5 Weighting Characteristics 18

3.5.1 Introduction 18

3.5.2 Motion 19

3.5.3 Luminance 19

3.5.4 Complexity 19

3.5.5 Panning 20

3.6 Description of Software 20

3.7 Implementation 21

3.8 Client Files 22

3.8.1 storyBoardApplet.java 22

3.8.2 util.java 24

3.9 Server Files 24

3.9.1 imageServer.cc 24

3.10 Image Manipulation Files 25

3.10.1 processImage.cc 25

4 **Sound Indexing** **25**

4.1 Introduction 25

4.2 Other Approaches 25

4.3 This Approach . . . 26
4.3.1 Melodic Contour . . . 26
4.3.2 Extracting Melodic Contour . . . 26
4.4 Testing . . . 27
4.4.1 Similarity Metrics . . . 28
4.4.2 Experiment . . . 28
4.4.3 Results . . . 29
4.5 Description of Software . . . 30
4.6 Implementation . . . 31
4.7 Client Files . . . 31
4.7.1 sound.java . . . 31
4.7.2 case.html . . . 32
4.8 Server files . . . 33
4.9 Sound Manipulation Files . . . 33
4.9.1 play.c . . . 33
4.9.2 libaudio.h . . . 33
4.9.3 libaudio.o . . . 33

5 Conclusion **34**
5.1 Integration . . . 34
5.2 Other Applications . . . 34

6 Appendix I: File Locations **35**

1 Introduction

Multimedia Data is heterogenous as compared to data of other kinds, such as financial or business Data. It is complex in nature, often consisting of generalisation-specialization abstractions and whole-part structure. For example, a video sequence is simply an image with a time dimension(inheritance), and a movie is a collection of parts, such as video, sound, and script(whole-part).

One interesting problem when dealing with Multimedia databases is to develop a manner in which to query them efficiently, a method which can deal with the complex nature of such data. In this report I will outline a graphical query method for complex multimedia databases.

Another interesting problem, is indexing multimedia data. This can be approached from two perspectives, indexing for the computer, or indexing from the human. For example, a number ID for students is for the computer, while using the name as an ID is for the human. Often, a combination of both is the ideal indexing method for data.

In this report I will detail indexing methods for two of the most important types of multimedia: video and audio. I will describe a method for extracting representative frames for movies and an algorithm for extracting representative samples from instrumental music. The approach is from the human perspective, as the concern here is mostly to represent larger sets of information by key elements for the human user.

Finally I will conclude with some discussion about how the various results of this technical report can be both integrated into a larger system or used seperately in other applications.

2 Query Interface

2.1 Introduction

In the increasingly computerized world, multimedia is at the fore front of communications. Ideas are spread through rich combinations of graphics, audio, and text on the computer. When the computer-naive user wishes to utilize this media, they are often faced with a difficult choice: re-inventing the wheel, or utilizing media which already exists. While many tools exist for building a multimedia arrangement from scratch, few methods have been developed to solve the latter problem: allowing the user to take complex but vague ideas and represent them with pre-existing multimedia. In this section, I will describe one solution to this problem.

2.2 Problem

When dealing with databases of complex multimedia objects it would be useful to have an intuitive method for the non-specialist to query for desired information. For example, in the multimedia domain a computer-naive user might wish to search for a work which reflects a vague and abstract mental image, such as "Find all horror movies with at least one romantic story unit".

2.3 Other Approaches

Two other approaches I'd like to examine here are Object Oriented SQL and Generalized(see Figure 1) Materilization Results Manager. With Object oriented SQL, we have a formal language which has been developed with a complex syntax. The GMR Manager, on the other hand, is somewhat better in that it provides a more accessible interface, but fails in that it is merely an adapted Query By Example interface made to work with the Object Oriented Method-

$<<f_1, \ldots, f_l>>$							
O_1: t_1	O_2: t_2	...	O_n: t_n	f_1	f_2	...	f_m
id_1	id_2	...	id_n	?	?	...	?
?	?	...	?	$[lb_1, ub_1]$	$[lb_2, ub_2]$	...	$[lb_m, ub_m]$

Figure 1: Generalized Materilization Results Manager

ology. Thus, in the previous case we have a methodology which is complex and not intuitive, and in the latter case we have a more intuitive interface but one which is not a pure extension of the Object Oriented Methodology. In the following, I will describe a graphical interface which is both.

2.4 This Approach

2.4.1 Introduction

In order to cleanly match a complex query-structure from the user, and to provide an efficient indexing and retrieval method, it is necessary to specify restrictions on the data model often used by complex objects. We provide here a brief, but exact definition of a model for describing complex objects based on the work of Wong Kim[1].

2.4.2 Expression Model

An expression model is an ordered pair (A, H), where A is an *attribute graph* (V, EA) and H is a *generalisation-specialization graph* (V, EH). V corresponds to the set of classes in the model. Both graphs are simple and directed, but only H is necessarily connected and acyclic. H has a special vertice r which is

the *root expression*. For example, $\forall\ v \in V$ | there is at least one path from r to v.

An *object* is a set of *object attributes* which are sets of ordered pairs, (l, p). l is an n-tuple, where n is the dimension of the space for the domain of the object referenced by p. Note that the function $P{:}I \rightarrow O$ is one to one. Also note the restriction on an attribute $a \in o$, $(\exists n \in I)(\forall (l,p) \in a)|\ dim\ l = n$. Queries are performed in reference to object sets. Operations which can be used are the normal set operations, as well as projection, selection, and multiplication.

Projection, when performed on a set, specifies the output attributes of objects. Selection is used to perform comparison operations on scalar as well as complex data. Multiplication is the operation which can combine objects together to create new objects and classes.

Evaluating the distance between graphs which describe complex objects is a difficult task, especially with heterogenous data such as multimedia. If there is a reliable function $f(a, b)$ which is the probability that objects a and b are value equal for all $v \in V$, $domain(a)$ is v and $domain(b)$ is v then the problem of structure and data matching becomes one of efficiency. However, this assumption is difficult to make in several cases, and an analysis of Type I and II errors becomes necessary.

We have studied three algorithms, with the strongest matching algorithm requiring a traversal of the entire graph. Error is dependent on the distance functions defined over the entire graph, and so sometimes a weaker but more probably algorithm might be necessary. One such method, would be to calculate distance in the "leaves" where error would be dependent upon the atomic values of the graph. And even more probable, but the most weak, would be to match on likely subgraphs.

The ability to express example information at various levels of the class

hierarchy, the ability to leave information unspecified, and the ability to import external data as example complex information are ome of the important attributes a query by example for multimedia system should have.

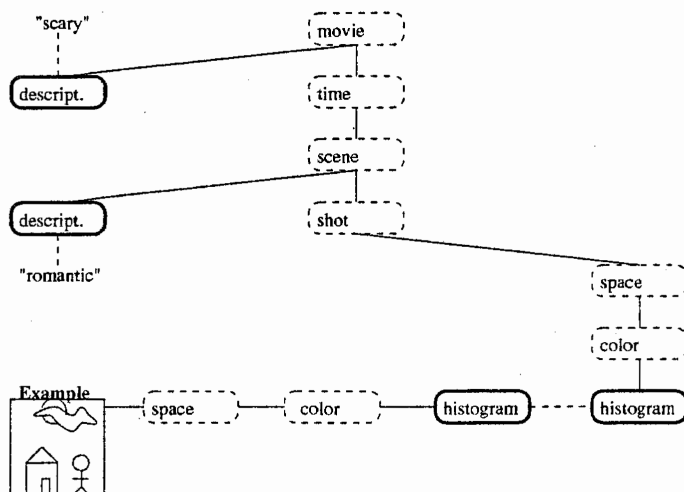


Figure 2: A Query Tree

2.4.3 Query Tree

A useful and visual tool for assisting the user in specifying example query data, is a *query tree*. A query tree is a graphical representation of the attribute graph representing the example multimedia. Nodes with specified information are colored different than nodes where information is left unspecified. For example, in Figure 2, we have an example query on a complex multimedia object(assuming the necessary pre-processing and distance functions). It would translate as: "Select all image with image.kansei like 'scary' and image.space.lines.kansei like 'active' and image.space.color.histogram like Example.histogram".

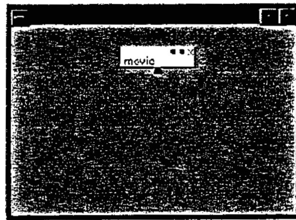
2.5 Description of Software

The system is a client-server application, with the client written in Java with a "thin" description of the interface, and the server written in C++ and C, utilizing the Objectsore MetaObject Protocol. The user constructs the query from information the server provides about the inheritance and whole-part structure of the complex multi-media objects stored in its database. After constructing the query, through a series of selections of parts and connecting scalar values to those parts, the user can send a query to the database which would return an answer in the form of multimedia data of similar structure and content to the query.

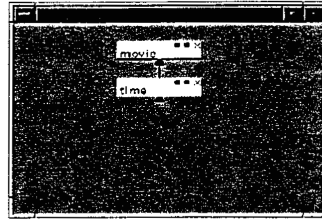
In order to use the software, the server "queries" must be running on miris37. From there, the main file can be run with "java paintClass" on a machine with java enabled. The interface will pop up at this point. It is fairly straightforward- use the mouse to click on the boxes set in each of the query tree nodes. Some example uses of the interface can be found in Figure 3.

2.6 Implementation

The architecture(Figure 4) is client-server in nature, and involves a network server, the Objectstore database system, and the query tree java thin client. After the user has constructed queries in the query client, the query is sent through the network to the server, which returns the results of the query. The network server is also responsible for reading the internal structure of the OBS Database and sending it to the Query Tree Client to work as the Query Graph.

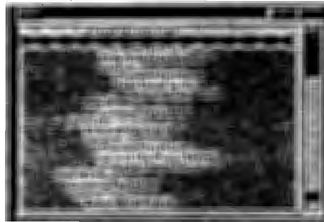


1. The starting screen.



2. Select blue box to add a new attribute, select time from the list of attributes.

4. What happens when you select new box for



5. An "attribute graph"

Figure 3: Example uses of the Query Tree interface

2.7 Client Files

2.7.1 DefinitionList.Java

The DefinitionList file contains all the connection routines to the server. It is what queries the server and creates a "query graph" that the user will utilize when developing the whole-part structure of their query. It defines the classes:

DefinitionList	- Network methods to connect to server
attribSelectionDialog	- Dialog extension for selection data member.
attribList	- member list used in the attrib dialog

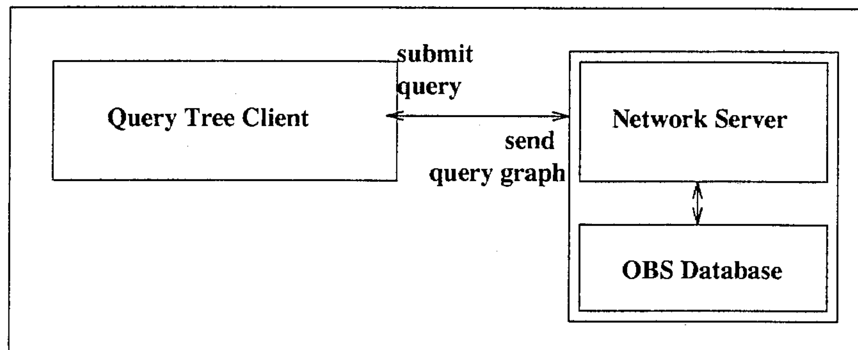


Figure 4: Architecture for Query Tree System

Client Files	Server Files	Database Files
DefinitionList.java	cserver.c, cserver.h	schema.cc
queryInputStream.java	SocketHandler.cc, SockerHandler.hh	expression.h
attribType.java	server.cc, server.hh	makeDB.cc
paintClass.java	queries.cc	

Figure 5: Files for Query Tree

2.7.2 queryInputStream.java

The queryInputStream file contains a description for the protocol for sending attribute descriptions to the definition list. This file only contains the public class, which implements the stream API.

2.7.3 attribType.java

The attribType.java file contains a description of a linked tree structure which is used to contain the attributes of the query graph. It also contains important code for interfacing with the DefinitionList and queryInputStream when filling out the query graph during the initilization process.

2.7.4 paintClass.java

The paintClass.java file is the main file, as it contains the description of the main() routine, as well as a description of the main event handler in the boxCanvas object. This file contains descriptions of methods for manipulating, drawing, deleting, and adding query boxes. The following classes are defined in paintClass.java:

paintClass	- the public class with main() defined
boxCanvas	- the canvas boxes are painted on, this contains the main event handler
box	- deals with box drawing and box info
textBox	- dialog for entering sclar text

2.8 Server Files

2.8.1 cserver.c, cserver.h

These files contains C routines for dealing with socket connections. It starts up the server by forking a process, returning if its the parent, and waiting for a connection if it's a child. If a connection occurs, it then forks a child process to deal with the socket connection.

2.8.2 SocketHandler.cc, SocketHandler.hh

These C++ files contains descriptions of the SocketHandler class. This class is used to deal with individual socket connections, including important routines to read and write data to the socket.

2.8.3 server.cc, server.hh

This is a basic class which was designed to act as the C++ wrapping for the cserver.c routines. By calling those routines, it waits and spawns off socket connections.

2.8.4 queries.cc

This is a complex file which describes the socket_connection routine, do_something. do_something has routines which access the MetaObject protocol of the Objectstore system. Briefly, they can query the objectstore database for the whole-part structure of classes described in the schema file. For example, a movie class which has the data members time, and description is described in an internal database of the Objectstore system. When socket connections are made to the server, the routines in queries.cc traverse the query graph described by the internal representation of the movie object, and return that information through the socket connection.

2.9 Database Files

2.9.1 schema.cc

Contains the objectstore schema description. See Objectstore documentation before modifying.

2.9.2 expression.h

Contains a description of the expression generalisation/specialization hierarchy. When the server returns the query graph, it parses an internal description of this .h file which has been put in the objectstore database..

2.9.3 makeDB.cc

Creates the database required by the server. The database will contain the internal description of the expression generalisation/specialization hierarchy.

3 Video Indexing

3.1 Introduction

On the other side of querying large databases, comes the problem of browsing the results of the querying process. A useful area of research in studying any kind of databases, is the issue of how to represent large sets of information with key elements of these sets.

For example, a viewer might find, as the results of a query, a five minute segment of an open field with a blue sky. Rather than complicating the results with a full motion video of this segment, it would be useful to select one still frame from the shot as a "summary" of the shots contents. In this manner, a large number of segments, even a movie, could be summarized on the screen at once.

3.2 Problem

Given a set of images which, when put in temporal sequence create full motion movie data, we would like to develop some technique which would enable the extraction of representative frames of movie content(Figure 6). Representative plays two roles: one, which is to summarize the movies content, and two, to provide a recognition role that prompts the users memory into remembering the contents of the movie. The former is very subjective, a point which provides the prime motivation for the approach described below.

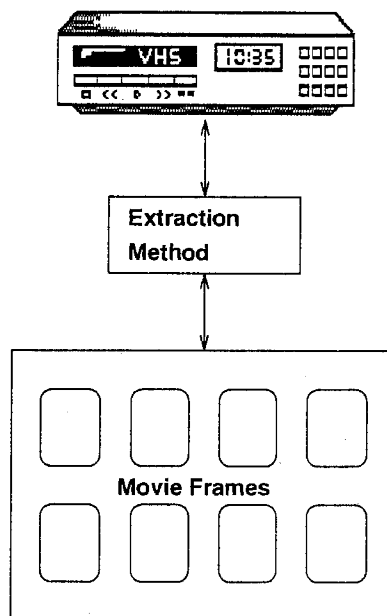


Figure 6: Frame Extraction Problem

3.3 Other Approaches

Other approaches have been designed which are primarily computer directed. They have relied mostly on computer vision techniques, concentrating on detecting shot boundaries in film[2], detecting panning and selecting key frames across the duration of the pan[3], selecting multiple key frames per shot by identifying significantly difference frames per shot[4], and finally, Wayne Wolf's method using motion analyses to detect dramatic moments within a shot[5]. Perhaps of all of them, the last is the most successful, however it, like the others, lacks in two important ways:

3.3.1 Objective versus Subjective, and the Common Sense problem

The processing of all the above methods is generally computer directed, allowing for no user involvement in decision making. For example, images taken from the local minima of motion analysis, the technique used in the last method mentioned above, is useful because it finds pauses in momentary action, which are often signs of dramatic emphasis. However, this is clearly not the case in all situations, and neither is dramatic emphasis by pausing the overriding factor in representing a shot or segment of film.

It's fairly clear that these other factors would often require human intelligence to pinpoint. A clear example of this would be scenes where the director is not attempting to attract any special attention to, but which are relevant to the plot, pacing, mood, etc of the story. This occurs in suspense movies, which has details the director wants to include for completeness, but doesn't want tooovertly give away before the suspense scenes.

It's also clear that these representative frames might be subjective. What is representative of a movie for one person, may not necessarily be representative for another.

It is for these two reasons, that a proper extraction scheme for representative frames from video should be human directed, rather than computer directed. The user should guide an algorithm according to his or her desires.

3.3.2 Time information

Another problem that most of these methods surprisingly lack, is proper use of temporal information. Some algorithms, for example, will remove duplicate frames simply because they assume they do not represent any extra information. However, pacing is a very important factor in directing, and sometimes the repetition of a scene is used purposefully to slow the pacing of a movie down.

Therefore, duplicating these frames in the set of representative images would be appropriate.

3.4 This Approach

This approach entails providing the user with a Storyboard interface, and allowing for an N-ary type search for representative frames, using weighted values for selecting the search boundaries. It is human directed in that the user selects the frames as well as provides the guiding characteristics of the search boundaries. It contains time information in that the search boundaries are fixed on temporal partitioning, in other words, a sequence is broken up into N partitions of equal length in time and the boundary frames are selected from those partitions.

3.4.1 N-ary Search

Like a binary search, an N-ary search performs a recursive search by partitioning a set and selects a particular partition to further search upon. So, given a set of frames F with elements $f_0..f_N$, and a fixed number of partitions p, we open the storyboard with p images(representing the search boundaries at the highest level). Rather than worrying about weighting issues before the user has a chance to interact with the process, we simply select the middle frame of each partition as a boundary frame. For example, we have the partitions length N/p, then we have the partitions:

$$(0, 1N/p), (1N/p, 2N/p), (2N/p, 3N/p)..((p-1)N/p, N))$$

And the display on the storyboard would be one image selected from each partition according to the weighted measure of the characteristics selected by the user. The two end images, selected from the first and last partitions, are the first and last images of that segment, respectively.

From here, the user can either select key frames from those provided, or continue to search for frames between two boundary images display. For example, if two images display from the previous segment of time t_1 and time t_2, then the Storyboard will clear, and provide p images, with the first and pth image representing the temporal boundary, and the $p-2$ images in between, representing the maximum waited images according to the respective partitions and characteristics selected. See Figure 7.

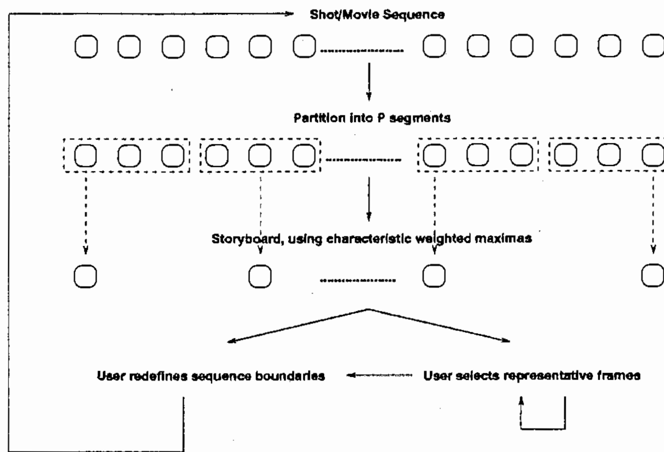


Figure 7: Frame Extraction Algorithm

3.5 Weighting Characteristics

3.5.1 Introduction

A key aspect of this particular method are the weighting characteristics. These are, but are not limited to: motion, luminance, complexity, and panning. They represent extracted information from the frames of the storyboard and are useful for constraining the search the user performs. For example, the user might want to find frames from the motion with maximum luminance. This would be useful

for finding representative frames from a movie segment where the majority of the frames are completely dark. Or motion, to find action scenes with lots of explosions, or images with minimum motion - to represent dramatic emphasis.

A control panel on the screen is shown to display the characteristics, and the user is allowed to select how little or how much they weigh when determining the representative frames. Here, I provide a brief description of each and how they're calculated:

3.5.2 Motion

Motion is calculated using image difference, with the change in intensity values giving a rough estimate of how much the image is changing. This is then normalised over the image. To solve the problem of camera panning, directional vectors are necessary. The effect of camera panning can then be reduced by reducing the weight of similar direction vectors. Motion is a useful cue and can be used to detect dramatic emphasis as well as action, such as fights, car chases, and explosions.

3.5.3 Luminance

Luminance is calculated using the simple equation:

$$Luminance = 0.299R + 0.587G + 0.114B$$

Luminance has many uses, most noteably to detect change in lighting, eliminating dark scenes when lighting conditions are too low, or to select dark scenes when lighting conditions are too high.

3.5.4 Complexity

Complexity can be calculated in two different ways. Firstly, it can be calculated using the Standard deviation of intensity values. This works roughly, but can

have difficulty with many complex objects in a scene which have similar color composition. Another way which also uses the semantic information from the image(but loses out in that objects with same color composition represent a non-complex scene) is to measure the number of edges in the scene. An edge detector is applied to the frame, and the number of on pixels, normalized over the image, can represent the complexity metric.

Complexity is a subtle characteristic which is difficult to use unless the user has an idea of what they're looking for. It would be especially useful for locating scenes with particular semantic composition, such as a country image with an open field or a city scene, with lots of buildings.

3.5.5 Panning

Panning is calculated by measuring motion vectors which are uniformally directional, divided by the standard deviation.

$$\sum v_i / \sigma^2$$

By minimizing Panning and maximizing motion, motion-intense scenes which may have camera panning can be eliminated.

3.6 Description of Software

Here are the uses of the four components which make up the storyBoard thin client.

Control Panel: With this, the user can select the weighted characteristics to constrain the N-ary search(Figure 8).

Partition: The partition window or "StoryBoard Search" window, contains the images which represent their various partitions. Click on down to re-define the partition boundaries(Figure 9).

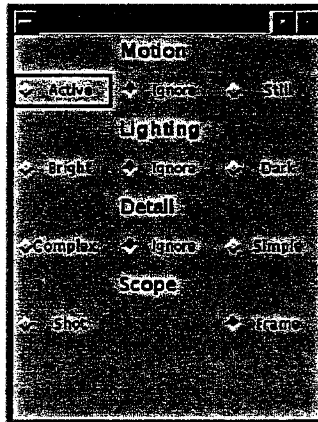


Figure 8: Control Panel Window

History: This has a history of partitions which have been examined before. Note this is especially useful as images in the history have been cached(Figure 10).

Storyboard: This window contains all the extracted images so far(Figure 11).

3.7 Implementation

The storyBoard architecture(Figure 12) is also client/server in nature. The client is made up of four windows: the control panel, the storyBoard(for storing extracted frames), the partitions(where the representative frames from each partition is displayed) and the history(which has a history of the partitions examined).

The server is made up of a Network Server which handles boundary definitions from the thin client and returns images from the image database, which is organised handled by a series of directory names.

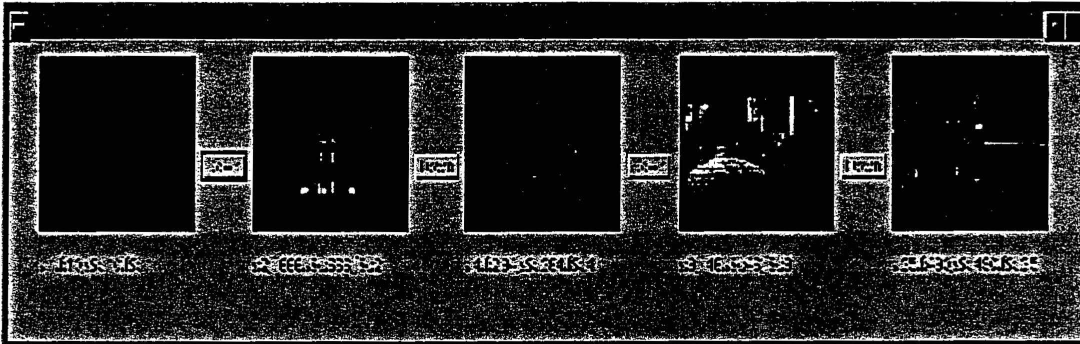


Figure 9: Partition Window

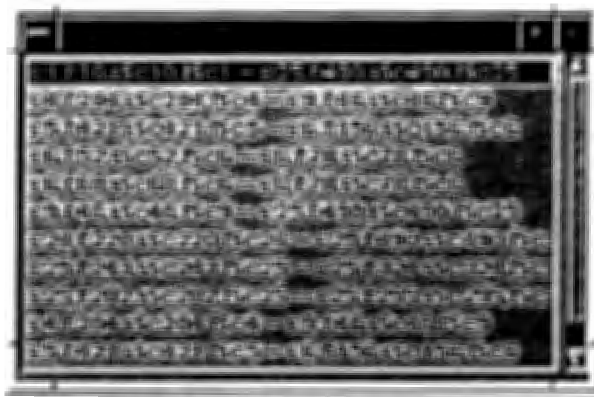


Figure 10: History Window

3.8 Client Files

3.8.1 storyBoardApplet.java

The main java file. storyBoardPanel is the most important class, handling the startup, layout of the user interface, and handling events. The other two important classes are the sideWindowFrame, used for controlling which segment to view, and the buttonMenu, the control panel listing the characteristics. storyBoardApplet contains definitions for the following classes:

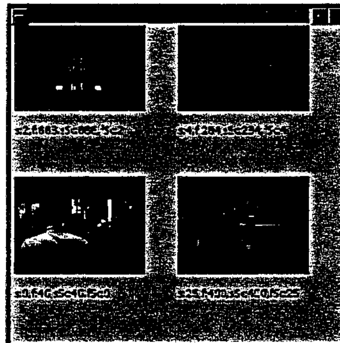


Figure 11: storyBoard Window

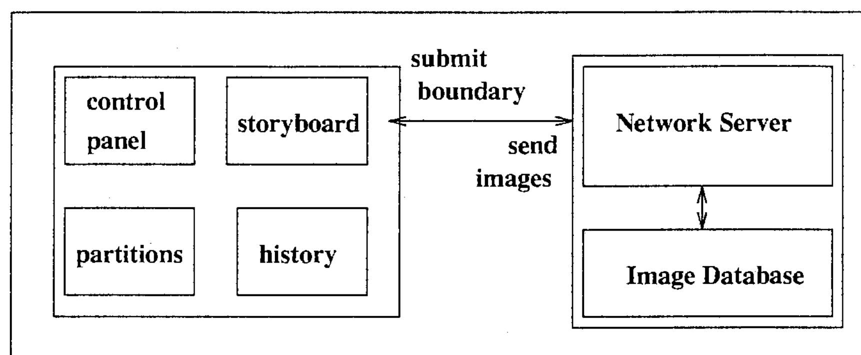


Figure 12: storyBoard Architecture

storyBoard	- top window, contains storyBoardPanel
storyBoardApplet	- applet, called by html file
framePoint	- small data structure for frame data
storyBoardPanel	- main class, lays out interface and images also handles client/server connection and event handling
storyBoardCanvas	- canvas, currently empty class
sideWindowFrame	- handles list of segments viewed
ImageCanvas	- used for painting images onto frame

Client Files	Server Files	Image manipulation files
storyBoardApplet.java	cserver.c, cserver.h	processImage.c
util.java	SocketHandler.cc, SockerHandler.hh	
	server.cc, server.hh	
	imageServer.cc	

Figure 13: storyBoard Files

downButton	- button between images
buttonMenu	- control panel with characteristics

3.8.2 util.java

Utility file, contains a routine called tokenize for splitting up strings in tokens.

3.9 Server Files

The files cserver.c, cserver.h, SocketHandler.cc, SocketHandler.hh, server.cc, server.hh are used from the previous implementations.

3.9.1 imageServer.cc

The imageServer.cc file contains definitions for functions which score frames and shots according to the control panel selections and returns representative frame names to the partition window.

3.10 Image Manipulation Files

3.10.1 processImage.cc

This file defines functions which, given an imagename, processes using various algorithms and writes to the imagename.sts file the results. Used by the image-Server to constrain searches according to selections from the control panel.

4 Sound Indexing

4.1 Introduction

As with video indexing, it would be useful to represent a large amount of sound information with a smaller, extracted sample. An practical example of this is the indexing feature on the CD players which allows the user to select songs to play after hearing the first 15 seconds. Another possible method is to hat first 15 seconds may sound like the same fifteen seconds in every song Problem

4.2 Other Approaches

At least two other approaches to this problem exist, one of which has been implemented on typical CD track programming devices. In the first case, the first N seconds of a song are extracted and played. This approach has the problem of being rather simple-minded in that no particularly representative audio information exists in the first N seconds. The second approach involves scene analysis, where the score for the instrumental song is extracted. While this is interesting key information, it has the problem of being difficult to understand by the average individual.

4.3 This Approach

4.3.1 Melodic Contour

This particular approach entails something similar to extracting the first N seconds of audio, but rather than simply the first N seconds, it finds the optimum contigous subset of N seconds in the audio file. This optimum subset will be the measured melodic contour of the audio file.

Melodic contour is a feature of music that stands out distinctively for the listener on first hearing. It conists of the overall pattern of intervals that make up a melody.

An interval consists of two things: size and quality. Quality is the change between two notes, and size is the length in half-steps before the next note takes over the soundspace.

For example, we have Bach's *The Well Tempered Clavier*[6], where the usage of Melodic Contour is very striking. At three points in the piece, Bach has a set of notes which have similar intervals. It is interesting to note, however, that at the second point in the piece the notes are not the same: merely the size and quality are the same.

We also notice from this piece two qualities of melodic contour: first, that it is overlapping. That is, two occurences of melodic contour can overlap one another. The second quality, is that it is shift-invariant. Though a set of notes may be shifted up and down or forward and back on the scale, the change in pitch or the length between notes does not change.

4.3.2 Extracting Melodic Contour

Melodic contour is extracted as follows. First, the following equation

$$\sum \sum \sum |a_{i+k} - b_{j+k}|$$

i=0...N, where N is 10% of the audio length

k=0...N

j=0...L, where L is the audio length

a, b are vectors describing the intervals which exist in the audio file.

is used to describe the score for a particular N length window in an audio file as being the most representative of the melodic contour. Basically, it means that every possible contigous window of length N in the audio file is compared against all the other contigous windows of length N. The window which has the lowest score(in other words, the least distance bewtween it's vectors and the vectors of other windows) is considered the window which represents the melodic contour.

Interval vectors are extracted by taking the average mu-law encoded amplitude of an 8Khz file over a period of 1/8th of a second or a 1000 samples. We then take the inflection points to be the position of each interval. The length of an interval is the number of samples between two inflection points, the change in pitch is merely the change between two inflection points(or interval positions).

4.4 Testing

In order to test this hypothesis, it is necessary to see if the melodic contour does indeed help the listener remember the instrumental song upon first hearing. We do this by using two cases, extracting an N sized window using the melodic contour algorithm, and extracting an N size window using a completely random selection algorithm. We then play the songs the samples were extracted from, and then ask a test subject to match up the samples with the songs they were extracted from.

In order to keep it so it isn't obvious on first hearing which songs match up with which samples, it was necessary to find the most similar songs. It would be a trivial task to match up a piano piece with a piano sample, if the other song to be matched up was all distorted guitar- and thus the testing would return no useful information.

4.4.1 Similarity Metrics

These similar songs were selected as follows:

1. 50 aiffc instrumental songs were gathered and converted to sun audio,mu-law encoded, 8 Khz files.

2. each song was matched up with the most "similar" four songs from the set of 50(excluding itself).

3. the top 3 sets of 5 were selected, with the total difference between songs the minimum of the whole set.

The similarity metric was as follows:

$$\sum a_i - b_i + \sum (a_{i+1} - a_i) - (b_{i+1} - b_i)$$

where:

a_i, b_i are samples from two songs.

This particular metric measures two things: the similarity in amplitude(the "loudness" of the instrument) and the similarity in rate of change or the "speed" of the song. These are particularly crude measurements, but served well enough in that it provides a measureable selection procedure rather than a subjective one.

4.4.2 Experiment

The experiment was performed in Netscape, using a Java Applet(see Figure). The following steps took place when the test was performed upon a subject:

1. 3, 4, or 5 songs were played, depending on the choice of the subject.

2. After every song was played once(and only once), 3 samples were played.

3. After a sample was played, the subject was then asked to select a song from the list which it most likely came from.

4. The subject then submitted the data.

Two things to note here: firstly, the songs were an average of 3 minutes each. Secondly, the samples were 10% of the length of the song they were extracted from. As stated earlier, these sample sets were either completely selected by a random extraction method, or they were completed selected by the melodic contour extraction algorithm.

4.4.3 Results

Machine	Songs	Type	Results	#
miris56	4	M	2-2,3-3,0-0,1-1	4
mmac10	3	M	2-2,0-1,1-0	1
hokusai	4	M	0-0,2-2,1-1,3-0	3
miris37	4	M	2-2,1-3,3-2,0-0	2
miris37	3	M	0-0,1-1,2-2	3
miris50	5	M	1-1,4-4,2-2,3-3,0-0	5
miris50	5	M	1-1,4-0,2-0,3-3,0-0	3
miris56	3	R	1-1,2-2,0-0	3
mmac10	4	R	2-0,3-2,0-1,1-3	0
ms29	4	R	1-0,2-2,3-2,0-1	1
ms29	4	R	1-1,2-2,3-2,0-3	2

Figure 14: Sound Extraction Results

Results(Figure 14) are very encouraging, with 82% of the samples extracted by the melodic contour algorithm correctly matched up with the songs from which the samples were extracted from. In the case of randomly selected samples, only

40% of the samples were correctly matched. Note however that the number of subjects was fairly low, and thus this evidence may be mostly anecdotal. However, we believe that it warrants further study.

4.5 Description of Software

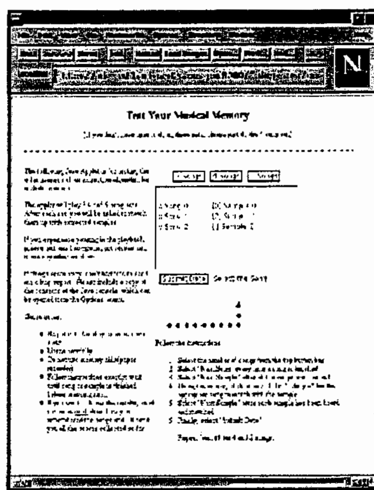


Figure 15: Sound Extraction Test Applet

The applet(Figure 15) can be executed by opening a netscape window at the address: file:/raid5-1/disk1/java/sound/case.html. After it's opened, follow the instructions given here(can also be found in the HTML page):

1. Select the number of songs from the top button bar
2. Select "Next Song" every time a song is finished
3. Select "Next Sample" after all the songs are finished
4. Using the mouse, click on one of the "o Song #" for the
5. Select "Next Sample" after each sample has been heard
6. Finally, select "Submit Data".

Repeat from #1 for 4 and 5 songs.

4.6 Implementation

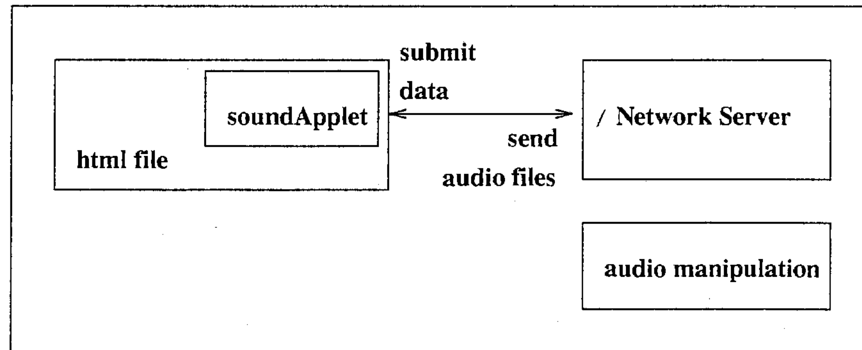


Figure 16: Sound Extraction Architecture

Audio files are manipulated, to find the similar audio files and to extract the melodic contour or a completely random sample. They are also converted from various formats and measured in various ways. See software use for more information. See Figure 16.

The software includes the following files:

Client Files	Server Files	Sound Manipulation Files
sound62.java	cserver.c, cserver.h	play.c
case.html	SocketHandler.cc, SockerHandler.hh	libaudio.h
	server.cc, server.hh	libaudio.o
	soundServer.cc	

Figure 17: Sound Extraction Files

4.7 Client Files

4.7.1 sound.java

This is the main test file, with all classes publicly defined in the java file. The sound class file is the most important, with the user interface and the main event handler being defined. Data is submitted through a network connection to the host computer.

The following classes are defined:

sound	- main classes, sets up interface and main event handler
soundCanvas	- canvas to handle lists of songs and samples played
messageLabel	- message utility to send messages to user
soundList	- extension of hashtable to provide sound dictionary
soundLoader	- loads sounds, can be multithreaded if so desired
songSet	- class to return the names of songs, given a base
randomBox	- class to return a set of unique random numbers between 0 and some limit.

4.7.2 case.html

Simple html file, with instructions and a link to the sound applet. Best editted with Netscape Gold editor. Note the use of tables.

4.8 Server files

The files cserver.c, cserver.h, SocketHandler.cc, SocketHandler.hh, server.cc, server.hh are used from the previous implementations. The only original file is soundServer.cc which defines a simple routine to accept and display submitted results from the sound applet. Can be extended to send out random names

for sound files to increase testing breadth.

4.9 Sound Manipulation Files

4.9.1 play.c

Extensive set of utilities for manipulating and converting sun audio files. Note that Aiffc files should be first converted into sun audio before using these utilities.

4.9.2 libaudio.h

Contains a description of some builtin audio manipulation routines as well as a description of the header attached to sun audio files.

4.9.3 libaudio.o

Include this library when compiling. Currently, compilation will only work on mic2.atr.co.jp, as this is the only machine that this file has been installed on. However, the library is not dynamic and will run on ms74.mic.atr.co.jp after compilation.

5 Conclusion

5.1 Integration

In conclusion, I would like to describe how this project can be used for multimedia databases. Refering to Figure 18 below, we see that the query interface can provide a potential interface to a complex multimedia database. As key index material, for browsing the results of queries, we could use image frames

and audio samples garnered by the algorithms for frame extraction and sound extraction.

The software includes the following files:

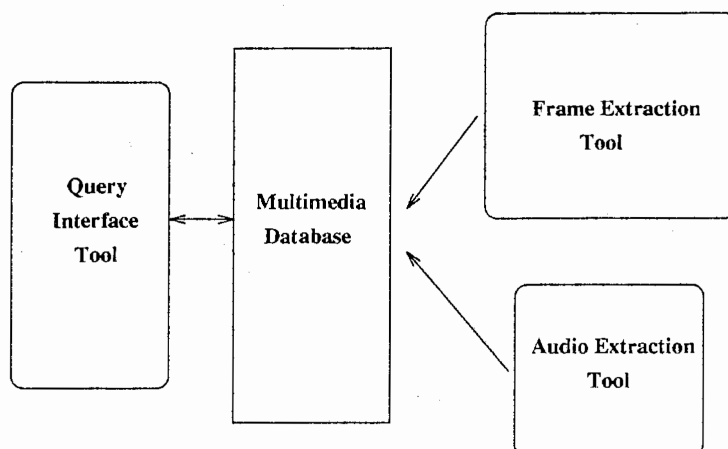


Figure 18: Integration of Software

5.2 Other Applications

There are many other types of applications, but two in particular I'd like to mention are Video browsing and Instrumental Song browsing. By using the aforementioned algorithms it would be possible to browse large collections of audio and visual multimedia, without concerning oneself with the query tree user interface.

6 Appendix I: File Locations

1. They can be found in shephara/software on ms74.mic.atr.co.jp.

Note that these files contain the code for the servers and clients for all three

projects. Data is also included for each project, except for the images required by the imageServer for the Frame Extraction project. These files can be found on:

2. /raid5-1/disk1/shotframes

These files include the segmented shots for the first few scenes of the movie Heat. They have been broken up into various directory names and these names are vital to work with the imageServer which is stored on ms74.

3. /raid5-1/disk1/SOUNDS

These are some useful audio files which have been converted from AIFFC to Sun AUDIO.

4. miris37

No vital files are being stored on miris37. However, please note that since no developmental enviroment exists on ms74, it will be necessary to either install a developmental enviroment on ms74(such as gcc) or change the installation settings on the files on miris37.

References

[1] Won Kim. *Introduction to Object Oriented Databases* The MIT Press, 1990

[2] A. Nagasaka and Y. Tanaka, "Automatic video indexing and full-video search for object appearances," in *Visual Databases Systems II*, Elsevier Scene Publishers, 1992, pp 113-127.

[3] K. Otsuji, Y. Tonomura, and Y.Ohba, "Video browsing using brightness Data," in *Visual Communications and IMage Processing 1991*, Bellingham WA: SPIE, vol. 1606, 1991, pp.980-985.

[4] Behzad Shahraray, "Scene change detection and content based sampling of video sequences, " in R.J. Safranek and A.A. Rodriguez, eds. , *Digital Video Compression: Algorithm and Technologies 1995*, Bellingham WA: SPIE, vol 2419, 1995.

[5] Wayne Wolfe "Key Frame Selection by Motion Analysis" IEEE Multimedia, Fall 1996, pp. 1228-1231

[6] Edited by Rita Aiello with John A. Sloboda, *Musical Perceptions,* Oxford University Press, 1994