7-10-1997

# Multimedia Data Analysis using ImageTcl (Extended version)

Charles B. Owen
*Dartmouth College*

Fillia Makedon
*Dartmouth College*

# Multimedia Data Analysis using ImageTcl (Extended version)[1]

Dartmouth College Computer Science
Technical Report PCS-TR97-310

Charles B. Owen, Fillia Makedon

Dartmouth Experimental Visualization Laboratory
6211 Sudikoff Labs
Dartmouth College, Hanover, NH, 03755, USA
devlab@cs.dartmouth.edu

**Abstract:** IMAGETCL is an new system which provides powerful Tcl/Tk based media scripting capabilities similar to those of the ViewSystem and Rivl in a unique environment that allows rapid prototyping and development of new components in the C++ language. Powerful user tools automate the creation of new components as well as the addition of new data types and file formats. Applications using IMAGETCL at the Dartmouth Experimental Visualization Laboratory (DEVLAB) include multiple stream media data analysis, automatic image annotation, and image sequence motion analysis. IMAGETCL combines the high speed of compiled languages with the testing and parameterization advantages of scripting languages.

## Keywords

TCL/TK, MULTIMEDIA, INFORMATION RETRIEVAL

## 1 Introduction

Multimedia applications require the efficient manipulation of media data (video, images, audio, etc.). Various systems have been developed to assist in this process, each system based on a different philosophy. The IMAGETCL multimedia development environment has been created to facilitate multimedia data analysis research (Owen (1996)). Typically, development of new technologies for multimedia data analysis has been complicated by many factors. Algorithms are usually concerned with processing sequences of image frame data or audio samples. However, prototypes of new algorithms must deal with complicated media file formats, mixed compression techniques, temporal sequencing of data, and pipelining issues required to deal with the large volumes of data. All of these issues ride like baggage on the back of the multimedia algorithm developer.

[1] A shorter version of this paper appears in the Proceedings of Gesellschaft fur Klassifikation e.V., University of Potsdam, Potsdam, Germany, March 12-14, 1997

ImageTcl provides layers of abstraction which mask these issues from the developer while still providing a fast and efficient environment for algorithm testing. It manages implementation layers automatically and provides algorithms under design with data in simple forms (vectors of audio data, matrices of image data). A small, yet flexible, set of standard media data types simplifies this process.

Multimedia algorithm development can be described as a five-step process: (1) algorithms are theoretically devised, (2) a prototype of the algorithm is implemented, (3) test procedures are designed which test the algorithm and give evidence of internal operation and intermediate results, (4) a set of test data is used to gauge algorithm performance and effectiveness, and (5) user interfaces are developed to allow user interaction and demonstration of the algorithm. These steps are normally part of an iterative process of refining and improving and are not necessarily sequential.

While ImageTcl does not theorize new algorithms, it is a great aid to the other four stages of development (2-5). Algorithm implementation is in C++, which provides a clean, easy to use, object oriented environment. The most unique feature of ImageTcl is its support for simplified implementation of algorithms in a modern **compiled** language. Test procedures can be rapidly prototyped and easily altered in the Tcl scripting language. This is a fairly common approach in research software systems due to its inherent flexibility. Test procedures are often highly instrumented and parameterized. Instrumentation provides monitoring of the algorithm operation both for analysis and debugging. Parameter adjustments are performed to tune the algorithm implementation and to test effectiveness and robustness.

Multimedia data is highly stochastic in nature and it is difficult to theorize away the problems of noise and media sampling. Hence, experimental development and demonstration of results is very important. Experimental media can be collected easily in an ImageTcl site and provided using the ImageTcl *Media Database* (ItMD).

The power of the Tcl/Tk development environment is available for user interface design. This environment is popular for user interface design due to its simplicity and rapid development support. Since Tcl/Tk is an interpreted language, user interface design and experiment procedure development can occur rapidly. New compilations of the test program are not required when user interface changes are made.

Several systems have been developed for multimedia software design and prototyping. VideoScheme, also developed at the Dartmouth Experimental Visualization Laboratory (DEVLAB), utilized the Scheme programming environment as a rapid prototyping system for digital video editing and analysis (Matthews (1993)). VideoScheme heavily influenced the genesis of ImageTcl . MIT's ViewStation is another powerful multimedia environment which is also Tcl/Tk based (Lindblad (1994)). ImageTcl uses a data-flow structure very similar to that of the ViewStation. Cornell's Rivl system abstracts away most of the media pa-

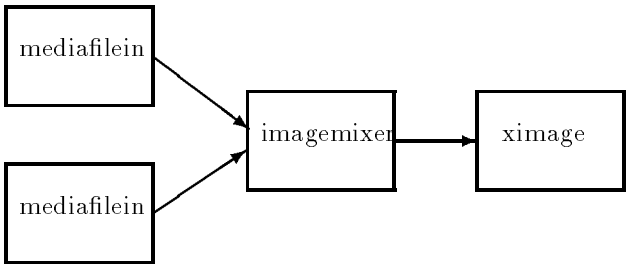


Figure 1: Example of an ImageTcl application as a graph.

rameters, including resolution and timing and provides very powerful media manipulation tools (Swartz (1995)).

## 2 ImageTcl Components

The most basic element in ImageTcl is the media manipulation *object*. These objects are created by ImageTcl commands and are *connected* into a directed graph, (which may be cyclical) where the nodes are objects and the edges are connections. Objects are the data processing elements. Figure 1 illustrates an image combination application, where two video files are combined (possibly with a cross fade) and displayed. In this figure the `mediafilein` objects are sources of data from files and `ximage` is a data sink which is a display window on the screen. The `imagemixer` object is both a sink and a source, accepting two channels of media data as input and producing a mixed media stream as output.

This structure is similar to the ViewStation design, except that a more general graph structure avoids any "T" data structures (Lindblad (1994)). Objects are typically general-purpose elements in the graph which perform a generic function. A *module* is associated with an object to provide detailed algorithmic functionality. In addition to objects and modules, ImageTcl has *data types* which are described below, and *system commands*, commands which provide control functionality in the system, but do not create new graph objects.

The flow of data packets within the graph is managed entirely by ImageTcl . Data pipe-lining and packet queuing is transparent to the algorithm designer. A source object program simply transmits packets to a source channel. Sink object programs provide a virtual function which is called when a new packet becomes available. A process scheduler balances execution among the graph nodes.

## 2.1 Media File and Live Media Support

Most application development involves processing media from files since such tests are easily repeatable. For this reason ImageTcl supports a wide variety of multimedia file formats, including WAVE, audio/video interleaved (AVI), Sun rasterimage, PPM, PGM, AIFF, and even some specialty medical image formats such as IMGF. ImageTcl uses a plug-in module system for media file support, making it easy to extend the system with new media file formats.

Real-time video and audio I/O is also supported. Video and audio provided in real time by computer interfaces is often useful for demonstrations of multimedia applications. Live media also provides a limitless data source; it is not uncommon for workstations to have a permanent connection to cable television for raw test data.

## 2.2 ImageTcl Data Types

ImageTcl supports a set of data types for media data. The *media data format* is the form in which the actual media data is stored. A set of basic media types are explicitly selected by the algorithm developer. This approach promotes efficiency and accuracy, avoiding computationally intensive and potentially error contributory implicit data conversion. *Data type conversion* components are standard in ImageTcl and can easily be applied to data not in the desired format. Some example ImageTcl data types include fixed and floating point color and monochrome images, digital audio, generic matrices and vectors, and special data types such as the YV data type which supports an image sequence as a three dimensional matrix. A powerful vector and matrix class library provides superclasses and manipulation tools for many of these data types. Matrix and vector manipulation can be directly applied to media data. There is no need to copy data out of data objects into intermediate storage.

## 2.3 Image and Media Processing Operations

There are many standard image and audio processing operations that algorithm developers call upon in development and don't wish to write anew for each project. ImageTcl has a wide variety of such operations as built-in components. These can be used at both the script level and in C++ code. A small sampling of these components includes image cropping, warping, edge enhancement, and edge detection. Several optical flow algorithms are provided as well as FFT and arbitrary flow field generation.

## 2.4 Automatic System Tools

A powerful feature of ImageTcl is automatic component creation. Supplying a C++ class library and an integrated environment does not necessarily

lead to rapid algorithm development. A developer must create new classes for the components derived from any of several *base classes*, the ancestors in an object-oriented environment with inheritance. Selecting the correct base class and knowing which functions to supply would be a burden on the prototyping process and result in a lot of useless code copying and deleting. The ImageTcl *Interactive Component Creation utility* allows for the simple creation of a new component by simply filling out a form. All files necessary to build and include this component are then generated automatically. Simple information about the new command, module, or data type is provided by the user before the component files are generated. The component is initially non-functional, but shells of all necessary procedures including demonstration code are provided.

ImageTcl is a highly modular system, a fundamental feature of any large software system, particularly rapid prototyping environments. *Automatic System Build Tools* allow an application to be built using any desired combination of components. Inclusion of components naturally creates larger systems, requiring larger compile and link times. Because ImageTcl applications can be built using a subset of available components, the development process can be accelerated. A smaller system can be developed which will compile and link much quicker. In addition, it is very easy to add new components to the system one at a time when there is no interrelationship between them.

# 3 Applications

ImageTcl is a standard development environment in the DEVLAB and has been used in many multimedia projects including video and audio analysis, speech recognition and synchronization, and medical imagery. This section describes a few applications in these diverse areas in which ImageTcl has been a major component.

## 3.1 Analysis of American Sign Language

A major research concern at the DEVLAB has been the study of human communication. Of particular interest is the visual language: American Sign Language (ASL) (Sternberg (1996)). There are three major categories of automation that have been addressed: cut detection, pause detection, and non-manual (facial) component motion analysis. The first two of these categories are concerned with *video segmentation* for efficient manipulation and categorization.

## 3.2 Video Cut Detection

ASL research video has several unique characteristics. Each utterance sequence is produced as a unit with recording starting at the beginning of the first utterance. This results in *cuts* between utterances. If cuts can be detected automat-

ically, the video segmentation required for easy access to individual utterances can be derived in advance, saving manual work.

VideoScheme, an earlier DEVLAB video editing system, was used for implementation and testing of several cut detection algorithms which determine the edit points in the construction of the image sequence (Matthews (1993)). In ImageTcl a new command called `cutdetect` was created using the Interactive Component Creation utility. This command creates "generic" video cut detection objects. Modules for the command can be created in a similar fashion so that several cut detection algorithms can be implemented and compared. Experiments performed on digital video in the DEVLAB media database occur in three basic steps: (1) an algorithm is written in C++, (2) test scripts are written in Tcl, and (3) Tk and Tix are used to write a simple user interface.

This application demonstrates the importance of the ImageTcl approach. The algorithms must manipulate pixels, performing frame comparisons and creating color histograms to detect the changes indicating a cut. Since each algorithm uses a differing approach, no standard command for image histogram or comparison could have performed these operations, and processing all of this data (230,400 bytes of data per frame, 30 frames per second, for nearly 7 megabytes per sequence second of processing) in interpreted code would have been too slow.

A major problem in all of these algorithms is parameterization. Numerous parameters must be "tweaked" during performance comparisons, including histogram bucket counts, detection thresholds, and frame support counts. Had these parameters been compiled in the algorithm source, each change would have entailed a time-consuming re-compilation. Certainly there are alternatives to this approach; for example, the parameters could be supplied by user input for each run. This approach, while useful, will be prone to error and limit reproducibility of experiments. It would have been possible to place these parameters in files and read them, but that would have added additional programming complexity. In ImageTcl , these parameters are supplied by the interpreted script. They can be easily changed and the program immediately restarted.

These experiments have yielded rather surprising results. While the algorithms perform quite well on given video sequences, they must be adapted at the parameter level for different sequences. As an example, the standard multimedia benchmark table tennis sequence requires a high threshold due to fast motion, while a large amount of American Sign Language (ASL) video has required low thresholds to capture cuts between signing examples due to the small amount of scene variation between the segments.

### 3.3 Video Pause Detection in American Sign Language

An alternative approach to segmentation of video sequences is *video pause detection*. Each sign language utterance can be further segmented by detecting the times when the signer paused between elements of the utterance. Indeed,

this is an important issue in the study of ASL. Pauses are *not* a simple segmentation means for terms, just as they are not a segmentation means for spoken sentences. The characterization of pauses is an important linguistic research goal.

Pause detection is important in American Sign Language video and, in particular, in indexing video segments that denote the end of an utterance, the completion of a facial gesture, etc. The DEVLAB has large amounts of ASL video and is performing experiments in classifying the motion sequences. Several algorithms have been implemented using IMAGETCL. A full report on the results is available in Liu (1997).

Pause detection can be likened to *inverted cut detection*. Cut detection searches for changes between frames which cannot be accounted for by normal motion or variance in image content. Such changes are, by definition, gross changes. Pause detection, on the other hand, searches for sections of minimal change. A major distinction between the two techniques is that pauses have duration, but cuts do not.

## 3.4 Motion Analysis in American Sign Language

A significant component of the linguistic study of sign language is the study of *non-manual* components as shown in Bahan (1996). Non-manual components are elements of the language not involving the hands. It is usually assumed that sign language is entirely based on hand movement. However, this is not the case — movement of the eyes and head and facial expressions are also critical components.

DEVLAB researchers are currently exploring motion analysis techniques for tracking the head and facial components in SignStream video. Though this work is too preliminary to be reported in detail here, it is based on a hierarchical derivation of the video motion and construction of affine motion tracks.

## 3.5 Other Example Applications

Some additional IMAGETCL applications include information retrieval in speech-based digital audio and video and analysis of functional magnetic resonance imagery (fMRI).

Information retrieval in digital video and audio is in its infancy. Techniques for locating particular words or phrases in spoken language audio or video are rudimentary at best, while finding these same words in printed text is trivial. There exists a large class of digital video and audio for which printed transcripts already exist. This includes legal depositions, which are always transcribed and commonly video taped, as well as radio and television programs which have been transcribed for cassette services. The transcriptions in these cases are often done in such a way that they are not synchronized with the video or audio.

An IMAGETCL -based information retrieval project currently underway at the DEVLAB is the Speak Alexandria project. This project seeks to design tools for information retrieval in speech-based media databases (including digital audio and video). One important component in this development is the synchronization of printed transcriptions to the corresponding audio stream. Such an alignment allows specific locations in a video or audio sequence to be found using simple text search tools. The DEVLAB text-to-speech alignment project is unique in that it focuses on information retrieval applications.

Speaker-independent speech recognition is very difficult and far from being a useful tool. However, given a transcription, the IMAGETCL based text-to-speech system can synchronize a phonemic translation of the transcription text to the output of a hidden Markov model biphone recognition system, which uses a modified Viterbi algorithm. High probability alignment is supplemented by phonemic timing interpolation over low probability matches (Rabiner (1993)). More detailed publications and a technical report on this work is in progress.

# 4 Summary

IMAGETCL has been designed to provide the multimedia algorithm designer with a high performance rapid prototyping environment. A concise and diverse environment manages the media manipulation elements of the development process. Algorithms can be developed in C++, a high performance programming language, and then tested using simple Tcl/Tk scripts. Projects studying human communications are a major research interest in the DEVLAB and have been developed using IMAGETCL . These projects include video segmentation techniques, analysis of American Sign Language video, information retrieval, and medical imagery analysis. IMAGETCL provides a test-bed for new design and development which can accelerate the state of the art in multimedia data analysis.

## References


BAHAN, B. J. (1996): *Non-manual realization of agreement in American sign language*, Ph.D. Thesis, Boston University.

LINDBLAD, C. J. (1994): A programming system for the dynamic manipulation of temporally sensitive data. MIT/LCS/TR-637, Massachusetts Institute of Technology.

LIU, X. and OWEN, C. B. and MAKEDON, F. S., Automatic video pause detection filter, Technical Report PCS-TR97-307, Dartmouth College, 1997.

MATTHEWS, J., GLOOR, P. and MAKEDON, F. (1993): VideoScheme: A programmable video editing system for automation and media recognition. *Proc. of ACM Multimedia'93*, Anaheim, CA.

OWEN, C. B. (1996): IMAGETCL multimedia development environment, `http://devlab.dartmouth.edu/imagetcl/`.

RABINER, L. and JUANG, B.-H. (1993): *Fundamentals of Speech Recognition*, Signal Processing Series, PTR Prentice Hall, Englewood Cliffs, NJ.

STERNBERG, M. L. A. (1996): *Essential ASL*, HarperPerennial Press, New York.

SWARTZ, J. and SMITH, B. C. (1995): A Resolution Independent Video Language. Proc. ACM Multimedia'95, San Francisco, CA.

TENNENHOUSE, D. et. al. (1993): The ViewStation Collected Papers. Massachusetts Institute of Technology Technical Report MIT/LCS/TR-590.