The ImageTcl Multimedia Algorithm Development System

Charles B. Owen
Dartmouth Experimental Visualization Laboratory
Dartmouth College
Hanover, NH

The ImageTcl Multimedia Algorithm Development System

Charles B. Owen
Dartmouth Experimental Visualization Laboratory
Dartmouth College
6211 Sudikoff Laboratory, Hanover, NH 03755
cowen@cs.dartmouth.edu
http://devlab.dartmouth.edu/imagetcl/

Abstract

The IMAGETCL *multimedia development system is a new Tcl/Tk-based development environment specifically targeting development of high-performance multimedia data analysis algorithms. Multimedia algorithm development is complicated by large volumes of data, complex file formats, compression and decompression, and temporal synchronization. Testing algorithms requires elaborate user interfaces which can display intermediate and result images, play audio, and adjust parameters.* IMAGETCL *uses the features of the Tcl/Tk environment as a base on which to build a system that significantly aids this process. This paper describes the* IMAGETCL *approach to algorithm development and describes several applications of* IMAGETCL *in the Dartmouth Experimental Visualization Laboratory (DEVLAB).*

1 Introduction

IMAGETCL is a new system designed to support multimedia algorithm development. 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 devised to test and debug the algorithm, (4) algorithm performance and effectiveness are tested using standard and custom data sets, and (5) user interfaces are developed to support user interaction and performance demonstration. These steps are a process and are not necessarily sequential. Media data (images, video, text, etc.) are noisy, imprecise, and ambiguous. The one perfect procedure for motion analysis, media alignment, or speech recognition is not known. Hence, proposed algorithms require extensive testing and adjustment.

The primary design goal of IMAGETCL is simplifying this process. IMAGETCL is based on the Tool Command Language and the Tk Toolkit and has been designed to provide the algorithm developer with a rapid prototyping environment which greatly aids steps 2 through 5 [7]. Automatic tools, a powerful function and application library, a cohesive media manipulation structure, and a compiled high-performance environment support the algorithm development process. Test procedures can be quickly and easily devised and modified using scripts written in Tcl. A central database for a site can be built which contains test data for algorithm evaluation, and high quality user interfaces can be quickly devised using Tk and Tix.

IMAGETCL is unique in its emphasis on **algorithm development in C++**. A large number of components are available at the script level and can be used to build and test applications. However, experience in the DEVLAB has shown that interpreted scripting languages do not lend themselves well to algorithms that directly manipulate samples, pixels, or voxels. Constraining the developer to matrix manipulations and avoiding low level loops as is common in systems such as Matlab limits flexibility in the design process [14]. IMAGETCL uses Tcl/Tk for what is does best, overall control and configuration, and leaves the low level processing in C++.

1.1 Related Work

Most multimedia development environments have focused on providing high-level scripting tools in an attempt to eliminate compiled language development. This approach assumes that such a complete set of tools can be devised and that the interpreted scripting code can use such powerful primitives that performance will not be compromised. A major element of the IMAGETCL design philosophy is that such a goal is difficult, if not impossible, to achieve. Systems often provide some facility for extending

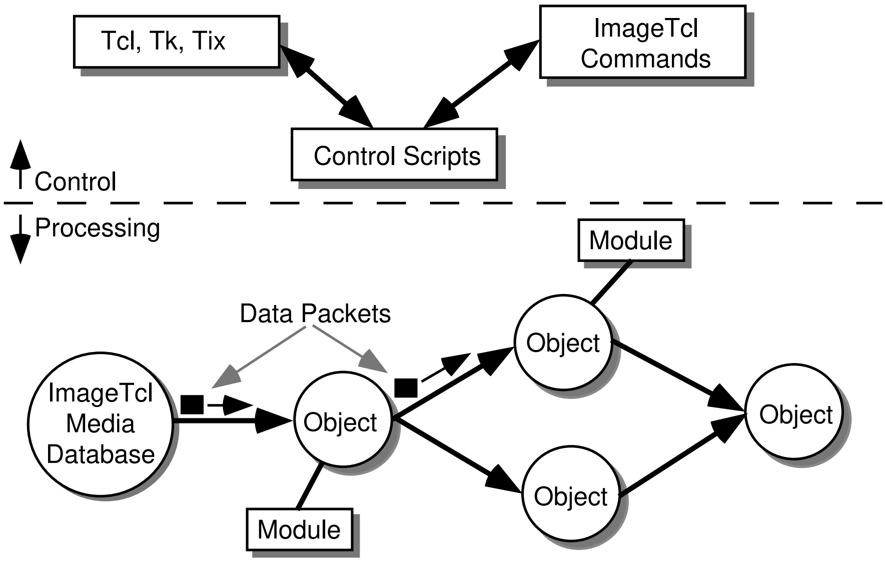

Figure 1: ImageTcl Execution Structure

the environment using a compiled language (usually C). In ImageTcl, however, easy extension of the environment using C++ is a basic development element.

Several systems have influenced ImageTcl development. VideoScheme, also developed at the DEVLAB, utilized the Scheme programming environment as a rapid prototyping system for digital video editing and analysis [6]. MIT's ViewStation is a powerful multimedia environment which is also Tcl/Tk based [3]. ImageTcl uses a data-flow structure very similar to that of the ViewStation. Cornell's Rivl system abstracts away most of the media parameters, including resolution and timing, and provides very powerful media manipulation tools [13]. ImageTcl uses an interpreter level data representation similar to that of Rivl. Multimedia *application* development is the major goal of these systems. ImageTcl is designed primarily for *algorithm development*, though it has many application level features.

2 System Overview

The system design of ImageTcl can be divided into an execution structure and a development structure. Figure 1 illustrates the execution structure, and Figure 2 illustrates the development structure. This section describes the components in these figures. Note the clear delineation between *control* and *processing*. ImageTcl performs media processing at the compiled code level and control and user interfaces at the Tcl scripting level.

2.1 Media Processing Model

ImageTcl is based on a dataflow media processing model. This is illustrated in the processing section of Figure 1. A *directed multigraph* (henceforth referred to simply as a *graph*) models the flow of data in the system. The nodes of the graph are *objects*, which provide general media processing. Objects are connected with directional edges representing the flow of *data packets* containing media data. *Modules* attach to objects and provide a specific algorithm implementation. The use of modules allows the separation of generic algorithm code from specific algorithm code. As an example, *optical flow* is a generic class of algorithm which attempts to derive flow fields describing the motion in an image sequence. Horn and Shunck devised a particular algorithm for computation of optical flow [2]. In the ImageTcl model, optical flow is an object and the Horn and Shunck algorithm is a module.

Composite objects are constructed using Tcl scripts. The ImageTcl *Media Database* (ItMDB) in Figure 1 is an example composite object. A library of these scripts is available to the application developer. Components in that library appear as objects in the graph. Composite objects are used in ImageTcl to hide issues of file formats and data compression and decompression from the developer. Media data does not flow through interpreted code

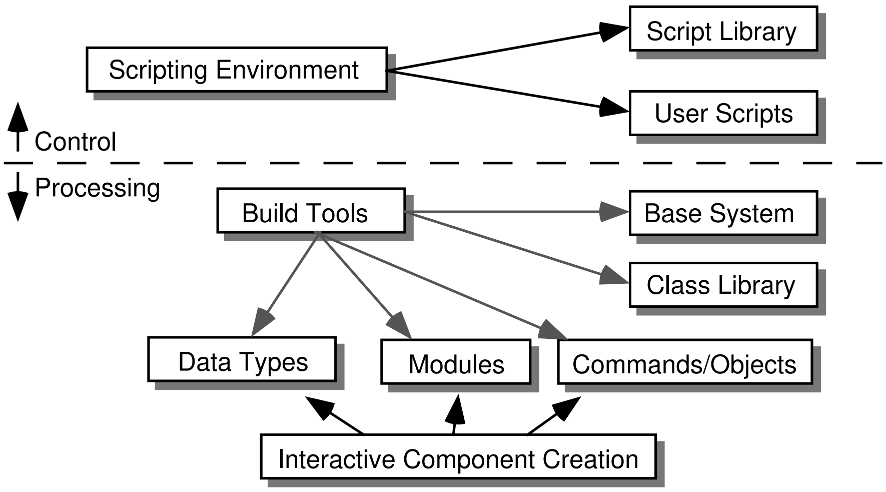

Figure 2: ImageTcl Development Structure

by default in this model[1]. Construction of the graph and all control is implemented in Tcl. This approach allows for very fast development of test scripts that provide a node with data and collate and present results. The scripts do not process the actual data and control is not embedded in compiled code, balancing the flexibility of Tcl with the performance of compiled languages.

2.2 Core and Standard System Components

ImageTcl components can be divided into three sets: (1) core components, (2) standard components, and (3) user components. Multimedia development systems are subject to constant expansion and can rapidly become large and cumbersome. Compile and link times can grow unwieldy. ImageTcl is highly modularized with components treated as building blocks that can be chosen at will. Both applications and dynamic link libraries can be built using any desired set of components. Only one line in an ImageTcl build file need be changed to add a component.
Core components are those necessary for system operation. These components are always included. *Standard components* are those which are included in the published system. A standard system containing all of these components is available. A user can also construct a subset system with only desired components. This modularity allows for faster compile and link times and has been particularly useful in system development.

[1] An alternative interpreted path exists for application flexibility.

The `itbuild` utility builds make files for applications and libraries. This utility can combine not only system and user components, but also outside components. `itbuild` automates make file creation, system configuration, and dependency checking.

2.3 User Component Creation

The most important feature of ImageTcl is user component creation. Every effort has been made to simplify this process. The *Interactive Component Creation Utility* is the first step in component creation. A user can easily add new commands, objects, modules, and data types to the system. Figure 3 illustrates the Interactive Component Creation Utility fill-in forms for new commands and data types. Characteristics of the component are entered into the form and all necessary files to create the new component are generated. ImageTcl is C++ based and has superclasses for a variety of components. These classes have many options and creating new elements manually would be cumbersome. The algorithm designer is primarily interested in implementation of the algorithm, not class construction, so automatically generated files provide a template that can be used immediately to create a new component.
Three files are created: a header file (.hxx file), a source file (.cxx), and an ImageTcl build file (.itb). The header and source files implement all classes necessary for the object including template procedures. Example code for the function is included in comments. The ImageTcl build file is used with the system build tools to include the component. The new component can be immediately included in the

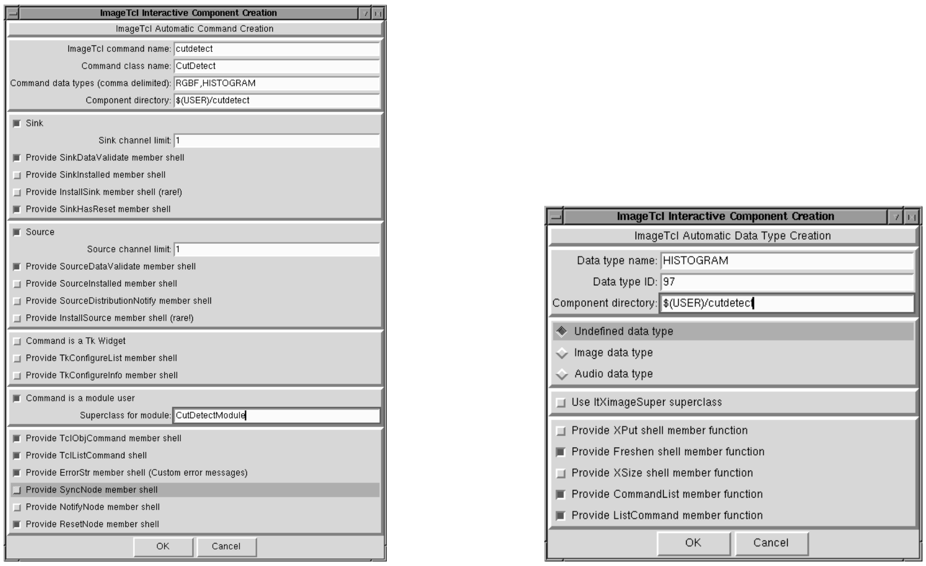

Figure 3: ImageTcl Interactive Component Creation Utility

system, compiled, and linked. The user must still, of course, provide algorithm functionality, but all of the mechanics of providing a new Tcl command, instantiation of objects, and usage will be fully operational.

2.4 Algorithm Support

Many resources are available at the algorithm implementation level. Class functions simplify use of the ImageTcl environment. These functions include command processing, error management, and command dispatching. Support for lists, stacks, directed graphs, trees, and other common containers is provided in easy to use template classes. Matrix and vector classes are provided and include overloaded operators for arithmetic manipulation as well as many standard signal processing and statistical analysis functions. A library of Tcl scripts is provided to simplify test and application level development.

3 ImageTcl Algorithm Development Process

Section 1 listed the five steps in the multimedia algorithm development process: (1) algorithm design, (2) prototype implementation, (3) implementation testing, (4) performance and effectiveness testing, and (5) user interface development. This section illustrates the ImageTcl approach to each step.

3.1 Algorithms are theoretically devised

Algorithm design is not directly supported in ImageTcl. The user is expected to design the algorithm. However, a common algorithm design aid is the analysis of data for statistical and visual characteristics. ImageTcl tools can be applied to this process.

3.2 Prototype implementation

An algorithm is implemented as new ImageTcl components, typically objects and modules. The new components must be derived from one of several C++ superclasses, must register themselves in the system, and must implement various virtual functions. While all of this information is documented, it would slow implementation considerably if this detail was the responsibility of the programmer. Instead, the programmer uses the ImageTcl Interactive Component Creation utility described in section 2.3. Necessary features, class and command naming, and function selection for the new component are described in simple fill-in forms. A complete set of template files for the new component is generated automatically.
The new component must then be added to the system. The user creates an ImageTcl build file for a system–this is a simple file which lists the components to be used and some build options such as optimization and build type. Adding a new component to the system requires a single line in the build

file. The `itbuild` utility reads the build file and creates a makefile and an application initialization file. All that is necessary are `make depend` and `make` to use the new component. Users never modify make files directly in ImageTcl.

The created components are ready to compile and link, though they are non-functional. The user "fills-in" the algorithm. Example code included in comments simplifies this process. The default behavior is to create an application which will utilize the standard ImageTcl shared library.

3.3 Test procedures

Once the new components are created, test scripts are required which will aid in the debugging of the components and ensure they are functioning correctly. A common structure for components is enforced by the system, simplifying script development. A *command* creates a named object. As an example, the statement `imagerotate rot` will create a new object named `rot`. The command (`imagerotate`) is equivalent to a C++ class and the object an instantiation of the class (in fact, this is the underlying implementation). More details on the command interface are included in section 4.2. Various test procedures can be devised for the same algorithm (and executable), and test procedures are high-level and can be developed quickly.

A common element in multimedia algorithm testing is parameterization. Many algorithms require thresholds, iteration counts, and other execution parameters, and often the adjustment of these parameters is a major part of the research. Parameters in ImageTcl development are defined at the scripting level. Support for parameter input from the scripting level can be implemented using as few as two lines of C++ code in the algorithm implementation. Because parameterization is implemented in interpreted Tcl scripts, parameters can be adjusted quickly, either by editing the script and re-executing, or by adding Tk user interface components. This rapid development of test procedures is a significant advantage of using the Tcl environment.

3.4 Performance and effectiveness testing

Multimedia algorithms (and information retrieval algorithms in general) require validation on large sets of data. An algorithm can be tuned to work well on a small data set, but must demonstrate effectiveness and performance on larger data sets. The ImageTcl *Media Database* allows accumulation of standard test data for a development site. This data is then available to all ImageTcl users. The database contains pointers to media files which can be located on different file systems.

3.5 User interface development

Though user interface development is listed last, it is commonly concurrent with most other steps. A project's user interface typically moves from minimal functionality to extensive functionality as the project proceeds from small tests to large scale validation. The Tcl/Tk/Tix environment, in combination with composite objects in the ImageTcl library, tremendously simplifies user interface development. User interface design is often highly iterative, with components adjusted and rearranged. The placement of this process at the scripting level has been highly advantageous.

4 System Structure

ImageTcl is highly modular with pieces designed as building blocks which can be added and removed at will for simpler system debugging, porting, and expansion. These blocks build upon the *core system* and the ImageTcl *environment*. A standard interface to components is enforced by the system in order to simplify component development and user scripting. A segregated set of *machine specific components* is included to support features specific to a single hardware architecture. A set of standard libraries, both Tcl and C++, as well as standard user algorithms are integral elements of the system.

4.1 Directory layout

An ImageTcl system is located in a directory available to all users and has a structure as illustrated in Figure 4. Machine and operating system specific directories allows segregation of system binaries and platform specific components. An example machine specific component would be the interface to audio and video machine hardware. This structure provides all ImageTcl source and binaries to all users.

4.2 Components interface

ImageTcl enforces a common component interface. Commands are grouped into two categories: object commands and non-object commands. *Non-object commands* provide control features in the system

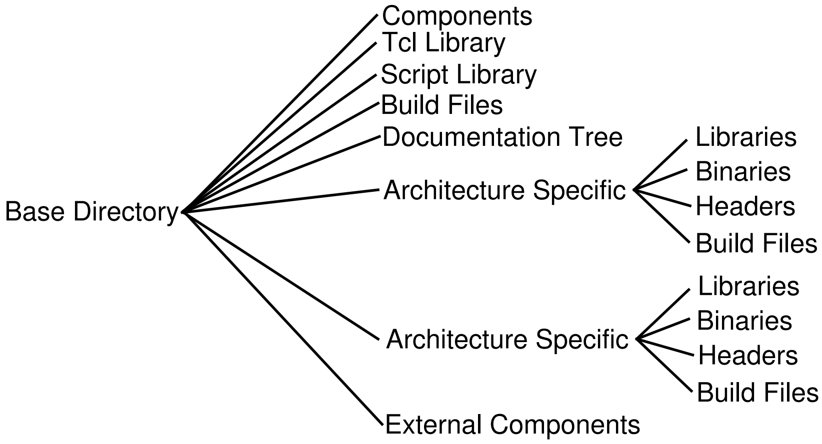

Figure 4: IMAGETCL Directory Structure

which do not entail the instantiation of an object. These are elements which have a single persistent object associated with them. An example is the `connect` command, which is used to connect object outputs to inputs in the graph. An example usage of this command is: `connect video 0 display 0`. In this example output channel 0 of a video input device is connected to input channel 0 of a display device. Note that *video* and *display* in this example are object names, not commands[2].

The system is highly object-oriented and the majority of commands are *object commands*. An object command creates an object. As an example, the `audiooutput` command can create an object named `audio` using the statement `audiooutput audio`. The object is then available as a Tcl command in the system. Communication with the object is through use of this command. This is identical to the widget structure in Tk.

The structure for parameters on commands is also similar to Tk. Parameters which begin with a "-" are considered to be *object list commands*, usually shorted to just *list commands*. As an example, in the Tcl statement `audio -sink 0 PCMW`, the `-sink` is a list command followed by parameters on that command. Multiple list commands can be used on the same line; IMAGETCL automatically scans this list and provides the command and options to the component. An example of a list command processing member function is included in Figure 5. This example processes the list command `-angle` with argument count checking.

List commands are used when no result is required. *Object commands* are used when a response is required. Only one object command is allowed per line and object commands do not start with "-". An example object command usage might be `set height [videocapture height]`. Figure 6 illustrates the member function code to support this command. `TclResult` is a member function of the `ImageTclEnvUser` superclass.

Commands are dispatched to modules attached to objects, then to the objects, then to generic handlers in superclasses. This structure allows generic object and standard system commands with minimal programmer intervention.

[2] As in Tk, they are now Tcl commands, though they are referred to in this system unambiguously as objects.

```
int ImageRotate::TclListCommand(char **argv,
                                int argcnt)
{
   // Process the command
   if(strcmp(argv[0], "-angle") == 0)
   {
      if(argcnt != 1)
          return ER_ARGCNT;

      angle = atof(argv[1]);
   }
   else return ER_BADLISTCOMMAND;

   return ER_NONE;
}
```

Figure 5: TclListCommand Member Function

4.3 Data types

IMAGETCL uses a data passing structure very similar to the ViewStation's [3]. A *data packet* is passed along the edges of the directed graph. Each edge has an associated packet queue. A *data buffer* is as-

```
int VideoInput::TclObjCommand(int argcnt,
                              char **argv)
{
   if(strcmp(argv[1], "height") == 0)
   {
      sprintf(TclResult(), "%d", VideoHeight());
      return ER_NONE;
   }

   return ER_BADCOMMAND;
}
```

Figure 6: TclObjCommand Member Function

sociated with a data packet and contains the actual media data. As in the ViewStation, a data packet is a general control object. Temporal sequencing information is contained in the data packet. Data buffers are specific to a data type (derived, of course, from a common superclass) and serve as a payload. Modifying sequencing information (reordering frames, changing frame duration, etc.) can be done by creating a new data packet and associating the same data buffer with that packet, avoiding duplication of large media data. Some example data types in ImageTcl include monochrome and color images, PCM audio, JPEG compressed images, and arbitrary vectors and matrices.

A unique feature of ImageTcl is that data types are plug-in components. Users can create new data types at will. Some example user data types have included speech biphone probabilities and color image histograms.

4.4 System components

System components are standard plug-in elements of ImageTcl. These include audio and video input/output, media conversion, standard media manipulations, arbitrary data packet routing, image display, and compression and decompression. A complete list is omitted here due to space considerations.

4.5 Applications and libraries

A large library is included in ImageTcl. System class libraries include matrix manipulation, strings, lists, and a directed graph template class. At the scripting level numerous scripts have been designed which are available as library components. One example is the *ItViewer* library. Many applications are tested through the application of media in specific formats. ItViewer creates a simple media browser with typical temporal controls. However, this browser is designed as a system component. New controls and menus can be added and data packets can be routed to algorithms under test. Most initial test applications are built on the ItViewer library. Initial test scripts can often be only a few lines long.

In addition to library components, many script applications are available, including development and media manipulation tools. These include applications which view media in different formats such as three-dimensional volume slices, and medical image viewers which support contrast adjustments.

4.6 Why Tcl?

Numerous scripting environments were examined prior to choosing Tcl for this project. A previous DEVLAB research system, VideoScheme, utilized the Scheme programming language, specifically the *Scheme in on defun* (SIOD) implementation [6]. However, if was felt that the functional nature of Scheme was not a natural description of the procedural construction and control of test environments. Perl was considered as an environment as well [15]. The large number of specialized variables and the lack of a standard user interface were reasons for not choosing that environment. Other choices considered included creation of an ad-hoc scripting environment. Tcl/Tk has clearly been an excellent choice.

4.7 Why C++?

ImageTcl development is C++ based. Since the Tcl/Tk environment is written in the C programming language, an obvious question is: why C++? C++ has been chosen as the underlying development language for several reasons [12]. The fundamental basis for this choice is that the object-oriented structure of C++ matches well with the media processing model used in the system. In addition, the strong typing features of the language are felt to be advantageous by the authors in that they decrease errors due to type mismatches and lack of function prototypes. The mix of Tcl and C++ has been seamless and quite successful in this system.

5 Applications

The DEVLAB has applied ImageTcl to numerous application areas in multimedia data analysis [4, 10, 9]. This section briefly describes a few of these projects, illustrating the specific advantages of the Tcl/Tk approach.

5.1 fMRI Data Analysis

Functional Magnetic Resonance Imaging (fMRI) captures volumetric images in a sequence [1]. A complete brain volume can be captured every two seconds. The images can be tuned to accentuate distributions of deoxy- and oxy-hemoglobin, which have been shown to be closely related to neuronal activity, providing a valuable view of the internal workings of the brain. ImageTcl is being used to develop algorithms which detect, localize, and parameterize this activity [8].

fMRI data analysis is a good example of the importance of algorithm development at the compiled level. A three minute fMRI sequence generates over 50 megabytes of image data. The ImageTcl activation localization implementation can process this data set in about two minutes.

5.2 Text-to-speech Alignment

Text-to-speech alignment is the temporal alignment of text to speech audio. The process is described in more detail in other literature [11]. Briefly, text is converted to a biphone directed graph (*Biphones* are sub-phonemic units of pronunciation). Audio is converted to biphone probabilities in ten millisecond time frames. A modified Viterbi algorithm is used to compute the alignment of the two.

This application is unique in that the Oregon Graduate Institute speech tools, another Tcl/Tk based system, were added to ImageTcl. Rather than building these components into the base system, the tools were treated as a plug-in component. A single ImageTcl *build file* was created to include the OGI libraries. This illustrates the flexibility of the ImageTcl automatic build tools: outside components are easy to add to a development project.

5.3 Cut and Pause Detection

The DEVLAB has been very interested in *information retrieval* (IR) in multimedia [5]. One common component of video IR systems is *cut detection* [16]. Cut detection is the location of camera edits in digital video sequences. This information provides a logical segmentation of the content. Each "cut" is a contiguous unit and can often be represented using a single frame called a *key frame* (or a small set of key frames). Several algorithms have been implemented at the DEVLAB.

Cut detection algorithms are often highly parameterized. Nearly all systems have fixed thresholds for detection. Many also have window parameters, histogram bucket counts, etc. Optimization of these parameters and application of test sequences is an on-going process. In ImageTcl the parameters are available at the scripting level and can be easily moved into user interface components.

An alternative video segmentation method is *pause detection* [4]. Pause detection can be described as "inverse" cut detection. Cut detection searches for frame pairs with large movement which cannot be explained by camera motion. Pause detection searches for periods of minimal motion. The DEVLAB has been studying American Sign Language (ASL) video. In this video, sequences do not have conventional camera cuts, but do have inter-element pauses. Reliable detection of these pauses is ongoing research. In addition, the result is a temporal sequence giving the duration and location of pauses. This sequence must be viewed with the original video, often frame-by-frame, in order to gauge algorithm effectiveness.

6 Summary

ImageTcl, a multimedia algorithm development system based on Tcl/Tk, is described. ImageTcl provides the algorithm developer with a fast and effective prototyping environment for new multimedia algorithms. The focus of the system is on algorithm development in C++ and testing and user interface development in Tcl. Numerous system automation tools and library components assist the developer in this process. Several applications of ImageTcl in multimedia research are also described.

7 Acknowledgments

The author wishes to acknowledge Fillia Makedon for project assistance, development facilities and support, and much wisdom. The American Sign Language research is in association with Carol Neidle from Boston University, Benjamin Bahan of Gallaudet University, and Otmar Foelsche of Dartmouth College. Several graduate students who have contributed to ImageTcldevelopment include James Ford, Jim Shain, and Xiaowen Liu. They have also participated in ImageTcl application projects at the DEVLAB.

8 Availability

Information about ImageTcl is available at `http://devlab.dartmouth.edu/imagetcl/`. At this time the system is considered to be in an alpha state and is not generally available, though specific requests will be considered.

References

[1] Peter A. Bandettini and Eric C. Wong. *Echo Planar Imaging*, chapter in Echo-Planer Magnetic Resonance Imaging of Human Brain Activation. Springer-Verlag, 1997.

[2] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[3] Christopher J. Lindblad. A programming system for the dynamic manipulation of temporally sensitive data. MIT/LCS/TR-637, Massachusetts Institute of Technology, 1994.

[4] Xiaowen Liu, Charles B. Owen, and Fillia S. Makedon. Automatic video pause detection filter. Technical Report PCS-TR97-307, Dartmouth College, 1997.

[5] Fillia Makedon and Charles B. Owen. Multimedia data analysis using ImageTcl and applications in automating the analysis of human communication. In *Proceedings of the 3rd Panhellenic Conference with International Participation: Didactics of Mathematics and Informatics in Education*, Patras, Greece, 1997.

[6] James Matthews, Peter Gloor, and Fillia Makedon. VideoScheme: A programmable video editing system for automation and media recognition. In *ACM Multimedia'93*, Anaheim, CA, 1993. ACM Press.

[7] John K. Ousterhout. *Tcl/Tk Engineering Manual.* Sun Microsystems, 1994.

[8] Charles B. Owen. Application of multiple media stream correlation to functional imaging of the brain. In *Proceedings of the International Conference on Vision, Recognition, Action: Neural Models of Mind and Machine*, Boston, MA, 1997. In submission.

[9] Charles B. Owen and Fillia Makedon. Multimedia data analysis using ImageTcl. In *Gesellschaft für Klassifikation e.V.*, University of Potsdam, Potsdam, Germany, 1997.

[10] Charles B. Owen and Fillia Makedon. Multimedia information retrieval development using ImageTcl. In *20th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Philadelphia, PA, 1997. Rejected.

[11] Charles B. Owen and Fillia Makedon. Multiple media stream data analysis. In *Gesellschaft für Klassifikation e.V.*, University of Potsdam, Potsdam, Germany, 1997.

[12] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley, Reading, Massachusetts, second edition, 1993.

[13] Jonathan Swartz and Brian C. Smith. A resolution independent video language. In *ACM Multimedia'95*, pages 179–188, San Francisco, CA, 1995. ACM Press.

[14] The MathWorks, Inc. The MathWorks web site, http://www.mathworks.com/, 1997.

[15] Larry Wall and Randal L. Schwartz. *Programming Perl.* O'Reilly and Associates, Inc., Sebastopol, CA, 1991.

[16] Hong Jiang Zhang, Atreyi Kankanhalli, and Stephen W. Smoliar. Automatic partitioning of full-motion video. *Multimedia Systems*, 1:10–28, 1993.