2004

Creating a Video Device Application

Kevin Septor
Colby College

Creating a Video Device Application

Kevin Septor

Honors Thesis

Advisor: Dale Skrien

Department of Computer Science

Colby College

Spring 2004

I'd like to thank Dale for being an awesome advisor, Marc for being my reader and LaTeX help, Clare for giving me a good outline and LaTeX help, and everyone else in the department. Also thanks to my Sarah for her amazing English major proofreading skills.

Did you know if you stay up for three days straight, you're certifiably insane? That's what they're telling me, anyway.

Contents

Chapter 1

Introduction and Background

A video device, such as a webcam, has the capability to be used in many ways, including recording video and behaving like a security camera. A TV tuner card can also record video, and in addition be used for watching television. However, the video hardware and device driver the operating system uses to access it are only capable of delivering raw frames of video. A user space application is necessary for exploiting the capabilities of such a device beyond reading sequential frames of video. This research explores the desiderata in a powerful video device application, and introduces a program that aims to fulfill these needs.

1.1 Terminology

To help understand this research, a few terms should be defined in more detail. An **operating system** is the main software platform of a computer. Windows, Linux, and Mac OS X are all examples of modern operating systems.

At a low level, the operating system communicates with the computer's hardware (comprised of various physical hardware devices) through a level of abstraction provided by **device drivers**. These device drivers take simple commands such as "**write a byte of data**" and translate them into a language understood by the device, such as setting various registers. One type of hardware device is a **video device** (sometimes referred to as a video capture device) which can capture frames of video and provide them to the operating system.

At a high level, an operating system provides an **application programmer interface**

(API). Programs (applications) running on the operating system have access to the operating system's API and APIs provided by 3rd party extensions called **libraries**. The combination of operating system and library APIs allows applications to issue complex command such as "draw a rectangle" or "create a zip file." These commands are written in a **programming language**, such as C++, and compiled into an **executable**, which is the binary data an operating system uses to run a program.

1.2 Task Driven Features

The features a video device application should have beyond the bare minimum of reading frames of video from a device are based on user tasks. If a user never needs their video device application to file their taxes, then the program shouldn't have a feature to do so. Let's examine some of these user tasks.

1.2.1 Watch TV

For a device with a video tuner, we want to go beyond tuning channels. We want to really *watch tv*. This means a video device application should have support for all of the following features.

On Screen Display

Display of channel number, audio volume, closed captioning, extended data service (XDS), and program guide.

Deinterlacing and Overlay

Television is transmitted in interlaced frames which must be implicitly deinterlaced (woven together) for use as a single frame of video. Odd numbered scan lines are thus displayed 1/30th of a second behind their even numbered counterparts. The phosphor coating on a television stores the electron charge from the previous scanlines for a seamless effect, but the resulting frame appears to have jagged edges when displayed on a computer monitor.

Various advanced deinterlacing techniques can be used to result in a higher temporal and vertical resolution, and thus a more aesthetically pleasing image. However, these deinterlacing methods require a significant amount of CPU usage. On the other hand, video overlay requires virtually zero processer usage because it copies the current frame of video directly from the TV tuner card's memory to the video card's memory across the PCI bus. Bypassing the system's memory and CPU processing in this fashion is the only acceptable way to use a video device for watching TV on a less powerful system (under ~1.5GHz).

1.2.2 Record Movies

Why not save what you're seeing to preserve the memory?

While You Watch

As you enjoy it in real time. A TV hooked up to a VCR will still receive the current signal as it's being recorded. A software application should be no different and perform two operations with the current frame of video. It should save it to a movie file and draw it on the screen.

Time Elapsed

Sometimes it's interesting to periodically record frames of video to a movie over a long fixed interval. Playing back the resulting film at normal speed can often result in a fun fast-forward through time without the cost of storing and skipping over the middle.

Personal Video Recorder

Digital Video Recorders (i.e. TiVo) are becoming more popular. Being able to automatically schedule which shows to record, pause live TV, and skip commericals are just some of the tasks a user might like to accomplish.

1.2.3 Capture Images

Exhibitionism is part of our culture. Who wouldn't want to take pictures with their webcam?

Labels

It's especially useful if the application that snaps the pictures can also annotate them with the date/time and captions. Otherwise another application has to be manually used and the timestamp would be inaccurate.

Upload

While we're at it, having a built in tool to upload the image to a webserver to be accessed by web surfers fulfills a popular need. Once again, without the built in feature another program must be used. It's tantamount to having one program that will record a movie, and then another program that will save it from RAM to the hard drive so that it can be viewed later. Security conscious users will want a secure method to upload the image.

Timer

Along the lines of time elapsed movies, recording images at regular intervals and storing them to the hard drive or uploading to a remote system is a common task.

Live Streaming

Although a static image is what is usually found on Internet webcam portals, there is often a need for a more frequently updated image. A client with only a web browser should be able to connect and receive streaming updates of the current video input. This is a sort of one way video conferencing without the client needing to install any extra software. Frequently this kind of live streaming is needed by traffic cameras located on busy roads and bridges.

1.2.4 Detect Motion

By comparing frames we can detect differences in the image which translate to motion if noise has been eliminated. A security camera-like functionality is then provided which has the benefit over time elapsed recording of not storing long periods of inactivity. It also stores many more frames when motion occurs.

Exclude Areas

Sometimes a camera's view will include a small object that appears to constantly move, but a user doesn't want to consider as part of the input for motion detection. For example, a mirror in a room that reflects the image of a high traffic area outside. Being able to create a bitmask which excludes that area is very useful and impossible for a user to do with external software.

Various Event Triggers

Naturally we'll want to record a video of the event. Users may also want to deal with the movement in a more advanced way, such as sending an e-mail or uploading the video if the camera in your home sets off a trigger while you're away. It'd be no good to simply record the video to the hard drive if the computer is about to be stolen.

Labels

Knowing the time, date, and location of events is crucial for understanding recorded security footage. The application should have a feature to annotate the frames of video with this information as they are saved.

1.2.5 Video Conference

Often webcams are purchased to keep in touch with friends and family that are far away for a more personal touch than a phone.

Bandwidth Control Options

Depending on a user's connection, they may want to trade off higher image quality for lesser audio quality. Thus a text chat feature should also be available. Antithetically, a user might want higher audio quality and a reduced image quality or frame rate.

Address Book

IPs are long and hard to remember so an address book to store video conference connection addresses is a very desirable feature.

H.323 Standards Compliant

The International Telecommunications Union (ITU) has defined a set of standards which video conferencing software should follow for maximum interoperability. The H.323 standards define how common user tasks with a normal phone such as placing calls on hold should function in a video conferencing application.

1.2.6 Miscellaneous

Along with all of the big tasks, there are the more mundane but necessary tasks of adjusting picture settings (hue, color, contrast, brightness, etc.), resizing the image (TV is commonly 640x480, webcams 320x240), and having the current input be scaled to display in the full screen. Users might also have multiple video devices attached to their system and need all of them to be accessible. Occasionally users might want to apply various post-processing image filters to remove the chrominance errors from black and white TV, mirror an image, or even apply a cool filter.

1.3 The State of V4L Applications

Before designing or writing any code, a bevy of existing video device applications were examined. The purpose of the survey was to determine which programs supported which users tasks. Also, notes were kept on how each application was implemented for possible use as a base program to extend with additional features to satisfy the research's requirements. Because of the low cost specification and intuition that reusing code would be very important in the project, only free open source programs written for Linux were considered. Linux video device applications access the underlying hardware through a simple API called Video4Linux (V4L). It should be noted that during the course of this research a new API

called Video4Linux2 (V4L2) was finalized and is slowly gaining ground. V4L2 became the new standard in the 2.6.x branch of Linux.

1.3.1 xawtv

Written by the primary author of the V4L API in Linux, xawtv[26] is a good reference implementation of a video device application. However, it uses a clunky outdated GUI interface and comes with no easy way to manage the small suite of programs that comes with it, including a webcam program. Further testing revealed that it actually does defy recommended V4L usage techniques and as a result works much better with the smaller set of TV tuner cards than the wide range of webcams on the market. In fact, it only worked with one of the two test webcams (the Logitech - see §B.4 for specific hardware and driver information), and even then only at certain resolutions. On the plus side, xawtv does support recording movies as well as video overlay and can run in full screen mode with close to zero CPU usage.

1.3.2 tvtime

tvtime[14] is one of the most advanced television watching applications in the pack. However, it has a narrow focus on high-quality video output and consequentally doesn't support overlay or recording movies. It does have a nice filter to remove chrominance, but it can't be used without a CPU intensive deinterlacing filter. Empirical testing found performance to be unacceptable on an 800MHz machine.

1.3.3 MythTV

Like tvtime, MythTV[32] is also only focused on only one user task, which is being a poor man's TiVo. It does have some unique features such as picture-in-picture. This feature requires two TV tuner cards, which most V4L users don't have. However, it's not unresonable to assume that a common user might have a TV tuner and a cheap webcam. MythTV is the only application surveyed that can access multiple video devices from a single instance of

the running application. MythTV also supports accessing MPEG-2 encoder cards (modern tuner cards come with one) for full speed recording and playback with low CPU usage.

1.3.4 camE

camE[22] is actually a rewrite of the minimal webcam program that comes with xawtv and adds over 15 new features. But even if it were rebundled with the xawtv package, it would still be difficult to use because you can't get live feedback of what the current video input is.

1.3.5 motion

Also fulfiling a small stand-alone application for a single task is motion[44]. As the name suggets, its main selling point is detecting motion. The author created a video loopback device which allows a user to display the video being processed by motion, but it requires another V4L application, such as xawtv. Unfortunately, the user must manually tell xawtv the dimensions of the video input. These are all poor usability issues.

1.3.6 GnomeMeeting

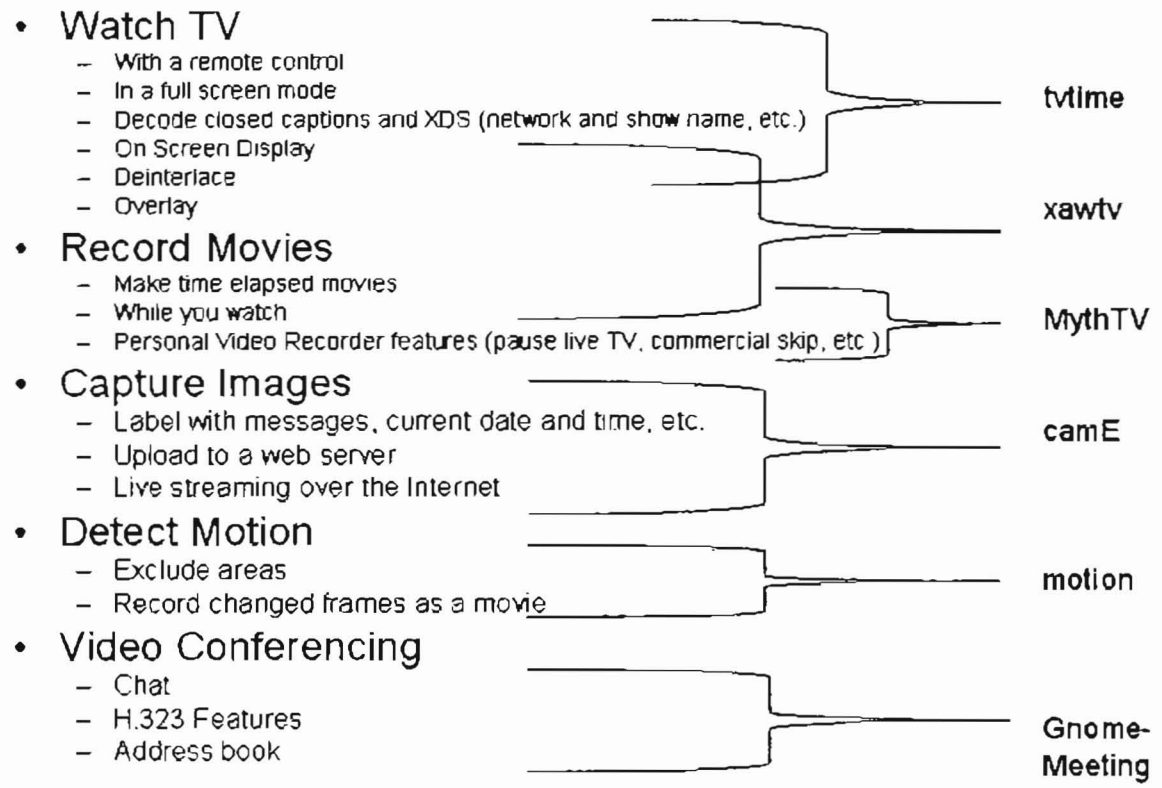
GnomeMeeting[33] is a well-developed application for video conferencing. It has everything a user could possibly want in a video conferencing program, but like the other packages, that's all it's really equipped to do. It has a very nice interface and it's very easy to get started with it.

1.3.7 EffecTV

Apparently written as more of a graphics demo than anything else, EffecTV[15]'s abilities are limited to applying very interesting special-effect plugins to incoming frames of video and displaying them. Despite the live video window, it's a command-line driven program that must be restarted to change image settings and also has a steep learning curve.

1.3.8 Feature Matrix and Other Video Applications

Feature Matrix



The above diagram shows the correlation between the applications and user tasks for which support is provided. Although there is some overlap, each tool corresponds to roughly one general task. As a result, up to six different applications must be used to access all of the capabilities provided by a video device.

Other Video Applications

There are a lot more V4L applications that couldn't be discussed for space reasons. A quick search on freshmeat.net will provide links to most of them. xawtv, tvtime, MythTV, camE, motion, GnomeMeeting, and EffecTV were selected because they appear to be the leaders in their respective target markets.

Windows video device owners usually run the proprietary software bundled with their hardware, such as ATI's Multimedia Center. However it appears that the typical webcam

software program isn't very robust as there are a ton of Windows webcam programs available such as ChillCam, KABCam, VisionGS, and Webcam32.

Because competition with Apple's own iSight camera and iChat AV software is difficult, Mac OS X doesn't have many 3rd party video device applications. However, EvoCam is a very nice webcam program for Mac OS X and GnomeMeeting has been ported as well.

Chapter 2

A New V4L Application is Born: grabber

This survey of the current video device applications reveals that most programs only support a few specialized features, thus a user must juggle several different programs to realize the full potential of their video devices. Managing multiple software programs isn't an easy task. Clearly a unified tool for accessing video devices is needed to rectify this situation.

Because no existing software program contained a majority of the features an advanced video device user would like, a new open source video device application called *grabber* (because it grabs frames of video) was created. Designing and implemeting this new program occupies a large part of this reseach. The techniques used to construct this piece of software are general methods that would benefit the development of similar projects. Failed solutions are also discussed so that others may learn and avoid making similiar mistakes in future work.

2.1 Specifications

A unified tool for accessing video devices needs to support a lot of features, and in this research the author attempted to do so while meeting several specifications.

First, the program's code must be elegant and well designed. It used standard modularization techniques and was implemented in clean, idiomatic code. Modern techniques such

as following design patterns[21] and applying the principals of refactoring[20] were used. As a partial motivator to avoid writing an ill-conceived implementation, the source code was made available to the public for scrutiny.

Second, the application must have excellent usability. Its design is based at the highest level from the needs of a typical user. If the software can't be used by the typical user, it's worthless. Human interface guidelines were followed at all times to yield a product that is friendly, intuitive, and powerful.

Third, budgetary constrains required that development costs be very low. Paying $76 for a few test video devices was acceptable, but paying $1550 for Qt/Windows or even $399 for Intel's C++ compiler for Linux was unacceptable. If technologically feasible, free software was used to minimize overall costs. Research was performed by unpaid volunteers whenever possible.

Finally, the project had an eight month deadline. Consequentially, a working prototype had to be delivered within seven months to provide time for evaluation of the application and recording of the results. If a feature wasn't complete in the initial seven month timeframe, it wasn't included in the research. Core features that got the application up and running had a higher priority than theoretical functionality.

2.2 Requirements

To meet the specifications discussed in §2.1, *grabber* had to almost by definition meet several requirements.

In order to satisfy elegance and deadline specifications, *grabber* had to be implemented in a high level language. Writing *grabber* in a low level language would require a large code base that would undoubtedly lend itself to inelegance and a long development time due to the lack of high level constructs.

Combined with the necessary support for a variety of functions and a usable interface, *grabber* had to have a graphical interface. Matched with the low cost specification, it had to run on Linux under the X Window System. Paired with the high level language and deadline requirements, the author decided that C++ would be used in conjuction with the Qt[40]

library. Qt (pronounced "cute") was selected as the underlying toolkit for *grabber* because of its completely object-oriented design and robust and clean API.

This blend of GNU/Linux/X11/Qt/C++ provided a solid foundation for the development of *grabber*. These core tools provided an affordable, robust, and easy to use general infrastructure suitable for projects with similiar specifications and requirements.

2.3 Goals

The creation of *grabber* yielded a useful non-trival program. In doing so, several skills were developed. The author's knowledge of C++ was increased by virtue of experience and also by way of learning best practices such as leveraging the standard template library (STL) and popular toolkits such as Boost[6]. Additionally, debugging acumen was refined by way of learning the GNU Debugger, gdb. The paradigm for error reporting in a compiled language (segmentation faults) is vastly different than the error reporting (stack traces) provided by interpreted languages such as Java, which made learning gdb a requisite skill. Quickness in understanding and mastering others' software was enhanced because the source code from similar open source programs was used as a model for implementing *grabber*.

Because none of the existing V4L programs were designed with multi-platform support in mind, it would be nice if *grabber* considered this burgeoning need. C++ written with portablity in mind is often well-designed and relies more on correctness than C++ written for a specific platform which relies on system quirks. Qt is also available for free on Linux and OSX, with both operating systems also containing the GNU suite of compilers for free. Coding multi-platform friendly C++ provided a better understanding of issues hidden by Java running on the JVM platform.

Chapter 3

Design and Implementation

3.1 Method

The development of *grabber* was an iterative process. Unlike the classic waterfall model, iterative development is an agile method which allows for mistakes in the design and unforeseen consequences in the implementation. The typical development cycle for *grabber* would last only 1-2 weeks. In each iteration a reasonable set of tasks were selected for completion with the goal of having a new release available at the end of the cycle. This method of project managment allowed frequent progress reports and saved *grabber* from getting bogged down in the "real soon now" state because at least every two weeks a new stable version was available.

In a sample week long iteration the tasks might be to "Implement video overlay and remove the thread from the LIRC class." Knowing that xawtv already implements video overlay, the author would examine the source code to xawtv and see how it's done. Although video overlay requires many steps, part of the process is to refresh any windows that are moved over the overlay window. After finding a method to refresh the root window (the whole screen) in `x11/x11.c:refresh_timer()`, the author would refactor the method to work without the Xt toolkit (only Xlib[4]) and also generalize the solution to refresh any window, not just the root window.

Original source code from xawtv:

```
static void
```

```
refresh_timer(XtPointer clientData, XtIntervalId *id)
{
    Window   win = RootWindowOfScreen(XtScreen(video));
    Display *dpy = XtDisplay(video);
    XSetWindowAttributes xswa;
    unsigned long mask;
    Window   tmp;

    if (!move && wmap && visibility == VisibilityUnobscured) {
    if (debug > 1)
        fprintf(stderr,"video: refresh skipped\n");
    return;
    }

    if (debug > 1)
    fprintf(stderr,"video: refresh\n");
    overlay_refresh = 0;
    if (wmap && visibility != VisibilityFullyObscured)
    did_refresh = 1;

    xswa.override_redirect = True;
    xswa.backing_store = NotUseful;
    xswa.save_under = False;
    mask = (CWSaveUnder | CWBackingStore| CWOverrideRedirect );
    tmp = XCreateWindow(dpy,win, 0,0, swidth,sheight, 0,
            CopyFromParent, InputOutput, CopyFromParent,
            mask, &xswa);
    XMapWindow(dpy, tmp);
    XUnmapWindow(dpy, tmp);
    XDestroyWindow(dpy, tmp);
    move = 0;
}
```

grabber's version of the code (located in `src/x11videosourcewidget.cpp`)

```
void X11VideoSourceWidget::refresh()
{
    Window rootWin = DefaultRootWindow(qt_xdisplay());
    refresh(rootWin);
}

// from xawtv's x11/x11.c
void X11VideoSourceWidget::refresh(Window window)
{
    Display* disp = qt_xdisplay();
```

```
    XWindowAttributes xwa;
    XGetWindowAttributes(disp, window, &xwa);


    unsigned long mask = CWSaveUnder | CWBackingStore | CWOverrideRedirect;
    XSetWindowAttributes xswa;
    xswa.backing_store = NotUseful;
    xswa.save_under = False;
    xswa.override_redirect = True;

    Window tmp = XCreateWindow(disp, window,
                    0, 0, xwa.width, xwa.height,
                    0, CopyFromParent, InputOutput, CopyFromParent,
                    mask, &xswa);
    XMapWindow(disp, tmp);
    XUnmapWindow(disp, tmp);
    XDestroyWindow(disp, tmp);
}
```

At this point it might be discovered that although *grabber* now has the code to refresh a window, it's a very difficult to determine exactly which windows need to be refreshed and when. This problem is discussed thoroughly in §3.6.1. Meanwhile, the author has decided to switch gears and try to complete the other task for this iteration.

After reading the technical documentation for LIRC[34] online the author would see that a `QSocketNotifier` might be the appropriate solution. A little time reading the online Qt documentation[41] would help clarify how the API should be used and now the author can implement a solution.

From src/lirc.cpp:

```
...
int flags = fcntl(lircfd, F_GETFL);
fcntl(lircfd, F_SETFL, flags | O_NONBLOCK);
QSocketNotifier* qsn = new QSocketNotifier(lircfd, QSocketNotifier::Read,
    this, "lircreader");
connect(qsn, SIGNAL(activated(int)), SLOT(readCode()));
...
```

Now instead of having an entire thread devoted to a blocking read operation on the LIRC file descriptor (the `lircfd` variable), the device is polled by Qt and when new data is available the `readCode` method is called.

By now the sample week long development cycle is over. Although video overlay wasn't fully implemented, the author has a better understanding of the scope of the problem and can choose to either tackle it later or devote all of a two week iteration to solving the problem. The good news is that the new stable release now has one less multi-threading issue to worry about. This means the program is less likely to crash due to improper locking and might even be faster because of reduced operating system task switching.

3.2 Framework

To avoid re-inventing the wheel, code from other open source software projects was used whenever possible. The GUI elements of *grabber* supported directly by Qt had to be coded from scratch, but the basic logic and lessons learned from existing programs such as xawtv were incorporated to fit *grabber*'s framework. Some sections of code, such as the highly researched list of various television station frequencies around the world, were copied verbatim under the terms of the GNU General Public License version 2[18].

3.2.1 Full Hardware Support

A video device application that would provide the best end user experience is one that supports all of the functionality provided by the system's hardware. For example, if a TV tuner card can read closed captioning data, then a video device application should support displaying the captions. Ideally, every piece of functionality supported by the hardware should be supported by a feature in the software.

In general, video device applications should be able to do more than just read frames of video from a hardware device. For example, if a query of the device's capabilties reveals that the video's brightness and size can be adjusted, then the application should present the user with a method to set these properties. Most devices support modifying video settings, but many devices do not have tuners. For example, a webcam won't have a tuner, but a TV card will. Thus, sometimes special hardware capabilties are available, and these should always be supported with a corresponding feature in the application. Unlike most other V4L applications, *grabber* supports multiple video devices and infrared remote controls.

3.2.2 Responsive GUI

Full frame 640x480 NTSC video has a frame rate of approximately 30 frames per second, and at 24 bits of color per pixel, that adds up to over 26 MB of data a second. Consequentially, *grabber* has to be fast. To alleviate some of the high data rate problems, the YUV color space, which only requires 16 bits per pixel, is used whenever possible. In addition, frames of video are shared throughout the program via thread-safe reference counted smart pointers. These techniques minimize memory access, thus providing for a higher overall throughput by reducing traffic over the system's bus.

In order to balance *grabber*'s multi-platform abilities with speed requirements, several abstract base classes are used with concrete platform specific implementations. This design allows *grabber* to take advantage of important optimized functions, such as the X Window System's XVideo and shared memory extensions. A speedy and elegant high-level interface is thereby presented to the programmer. When testing or porting to a new platform, a default implementation of the interface, which only relies on Qt, is also available.

3.2.3 Color Spaces

Computers typically operate with data in the RGB[A] color space. 8 bits of red, 8 bits of green, 8 bits of blue, and an optional 8 bits of alpha channel or transparency. On the other hand, video typically operates with data in the YUV (also called YCbCr) color space. Y represents the luminance or brightness, U represents the blue chrominance component, and V represents the red chrominance component. The YUV format is a result of forwards compatibility for black and white television sets with color TV transmissions. A black and white TV simply reads the Y (luminance) as a grayscale value and ignores the red and blue components. A color TV uses U and V for red and blue and the Y component for green. There are several YUV formats, and they all favor the Y component with the most resolution because that's the color the human eye sees best[3].

grabber needed a method to convert between the RGB and YUV color spaces. TV tuner cards internally use YUV and might need to have their data converted to RGB for display. Most webcams seem to internally use RGB, yet they might also need to have their data

converted to YUV format to take advantage of special fast YUV drawing routines that the video card might have. Eventually I ended up using libjpeg[24] to convert between the two colorspaces.

3.2.4 Internationalization (i18n)

Qt version 3.0 and higher makes internationalization easy. Although *grabber* was written for Qt 2.3, the i18n tools in version Qt 3.0+ are backwards compatible. By borrowing the `linguist`, `lupdate`, and `lrelease` tools from 3.0, a framework for making translated versions of *grabber* was established. User-visible text is wrapped in a `QString`, which internally uses unicode. Running the `lupdate` tool generates source translation files which can then easily be edited by the `lingust` tool. Afterwards `lrelease` is executed to create the translation objects which *grabber* loads during initialization based on the language specified in the configuration file.

3.2.5 Web Server

Because a simple webserver was needed, one was written from scratch. Using an existing webserver, like Apache[12] would require a lot of work to extract only the meaningful parts.

3.2.6 Remote Control

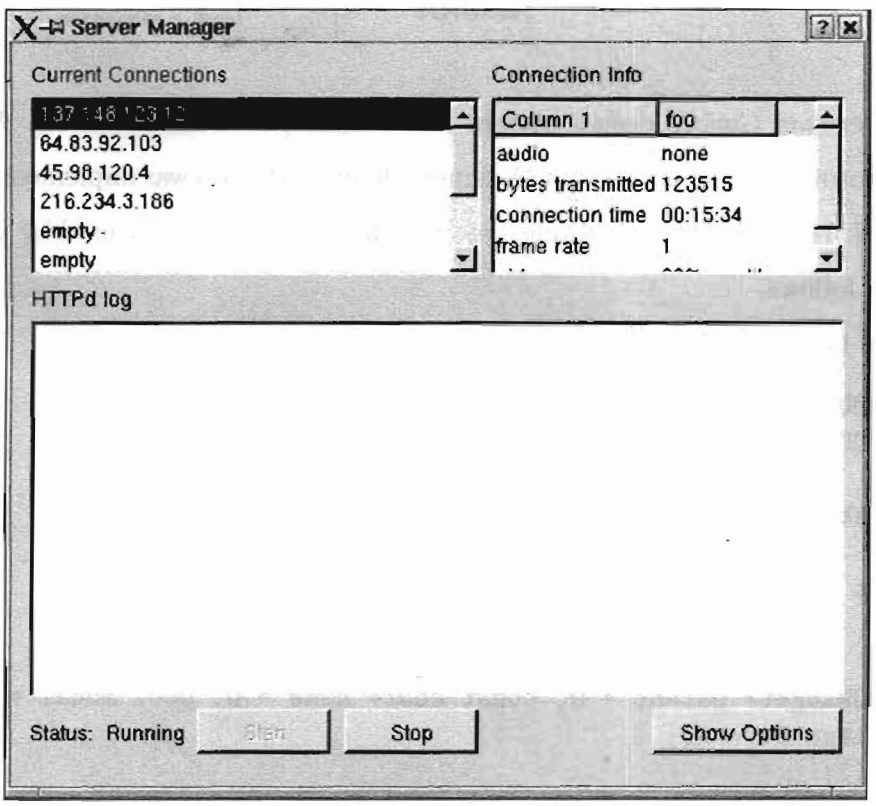
We want to sit back and relax while watching TV, so the LIRC[8] project was used for reading data from an infrared remote control.

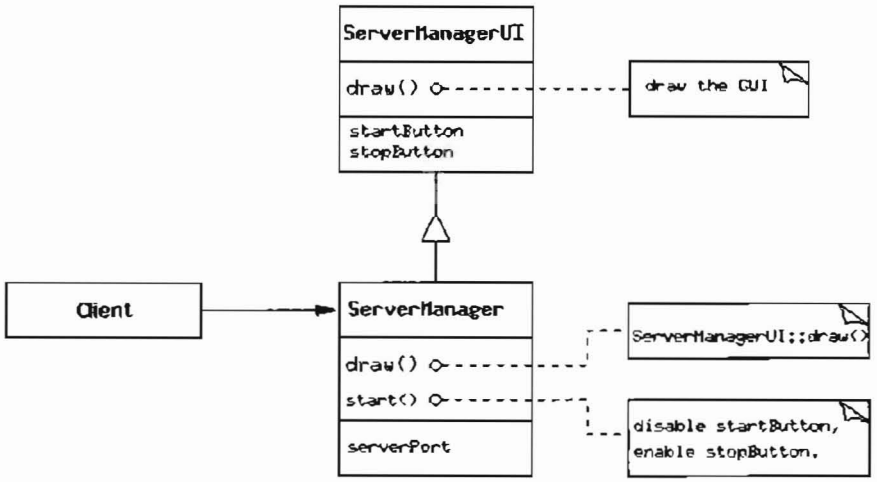
3.2.7 Streaming Applet

The client for the live streaming webserver is actually written in Java[25]. What else would run in a web browser? Reflection was used to determine the audio capabilities of the applet. If Java 1.2+ is found, the API audio is used. Otherwise, an attempt to use the forbidden `sun.audio.AudioPlayer` package is used, which seems to be present under all Windows implementations. Double-buffering is used for a flicker-free image.

3.3 User Interface

When writing a GUI program, usability is paramount. However, a program's usability doesn't have to sacrifice easy programmability. Consider the poor user interface presented below. It hides information from the user because one can't see at a glance statistics for all of the current connections. As Jef Raskin said "If people weren't good at finding tiny things in long lists, the *Wall Street Journal*, would have gone out of business years ago."[31] Normally this would be a pain to reprogram, but not in *grabber*.





The *Generation Gap*[43] design pattern (UML above) is used extensively. After visually creating a prototype interface with Qt Designer, all one had to do was implement the methods that accessed the buttons. A minimal C++ implementation (actually used by *grabber* in the `About` class) follows.

From the file `about.h`:

```
#ifndef ABOUT_H
#define ABOUT_H

#include "aboutui.h"

class About : public AboutUI
{
public:
    About(QWidget* parent = 0, const char* name = 0, bool modal = false,
        WFlags fl = 0);
    ~About();
};

#endif // ABOUT_H
```

From the file `about.cpp`:

```
#include "about.h"

About::About(QWidget* parent, const char* name, bool modal, WFlags fl)
    : AboutUI(parent, name, modal, fl)
{
}
```

```
About::~About()
{
}
```

In a typical implementation that actually uses the widgets in the GUI, `connect()`ions would be created in the constructor with call-backs to class member functions. For example, the Upload button in the WebCam is set up in the constructor with this line

```
connect(uploadButton, SIGNAL(clicked()), SLOT(upload()));
```

and the `WebCam::upload` method contains the code that actually implements uploading an image.

When user testing reveals a problem, a quick edit with `designer` and a recompile are the only steps needed to manipulate widgets into a more user-friendly display. Using this technique, *grabber* was able to rapidly develop a plesant interface and remains easily extensible.

3.4 Testing and Debugging

Testing occured every day during the development of *grabber*. The author "ate his own dog food" and used the most recent version for personal use. This means if watching TV was broken, the author couldn't watch TV until it was fixed in *grabber*. Having the incentive to really test new features seemed to be enough as only a handful of broken releases were made. One testing technique that really helped to debug subtle memory errors was to run Valgrind[36] while using the program. It outputs in real time common errors such as a conditional jump depending on an uninitialized value.

3.5 Using C++

C++ is a big language. Its creator wrote a very large book[38] on all of its features. Ideally a C++ programmer would known everything there is to know about C++, but that might take a decade. There are some other books, like Effective C++ [28] that make the process of learning C++ less painful. However, the "50 Specific Ways to Improve Your Programs

and Designs" don't apply equally well to every project. For example, following item 29 (p.123-128) which suggests avoiding returning handles to internal data would be far too slow if applied to the VideoFrame class. A 640x480 VideoFrame is huge and can't be copied, it must be shared. A data sharing solution is discussed in §3.6.5. On the other hand item 48 (p. 223-224) which recommends paying attention to compiler warnings was taken liberally to mean rewriting the Makefile to produce a less verbose output. This turned out to be a fabulous idea because a warning had become lost in the prolix output which was echoing the current compiler command being issued including the long list of flags and options.

3.5.1 Building a C++ Program

One of the 12 questions on The Joel Test[37] is "Can you make a build in one step?" A one step build of a C++ program is only trivial for a single source file.

```
g++ main.cpp -o app
```

Even a program with only two source files requires three steps to build.

```
g++ -c main.cpp -o main.o
g++ -c util.cpp -o util.o
g++ main.o util.o -o app
```

Add in linking with libraries and generated source files, and you'll have a complicated build process that's far from being one step. A tool, called GNU Make[19], is commonly used to manage the build of C++ programs. Because *grabber* uses Make, it has a one step build process. However, Make is easly abused and used incorrectly. The next subsections cover these issues.

Make

Although compiling a C++ program may seem like a trival task, it's not. Large programs develop complicated dependency graphs and automation is the only feasible way to manage the relationships. For example, every source file that includes a header must be recompiled when the header is modified. Further, code relying on generated files must evaluated only after the generated files are created or updated.

A simple solution to the problem is to pass the `-MM` flag to g++. The resulting output is in the format of a rule GNU Make can use to calculate which objects must be rebuilt. However, g++ `-MM` must be re-run and the Makefile updated every time a header file is modified. Clearly this is not putting us on the path to one touch compilation bliss.

Linking Order

If you're linking against a library and get a warning like "undefined reference to 'XF86VidModeSwitchToMode'", you *need* to switch the order of the command like arguments to g++ so that the list of objects gets passed *first* and the list of libraries comes *last*. For example...

```
g++ -lqt-mt -L/usr/X11R6/lib -lXxf86vm foo.o bar.o -o monkey
```

is incorrect and

```
g++ foo.o bar.o -lqt-mt -L/usr/X11R6/lib -lXxf86vm -o monkey
```

is correct.

Further Reading

Recursive Make Considered Harmful[29] is an excellent paper that not only covers in more depth the pitfalls of recursive Make, but also the proper use of Make.

3.5.2 Broken Compilers

g++ 2.96 is not an official compiler release, just a broken version from RedHat. Avoid using it at all costs; it can't even compile the Linux kernel. RedHat was nice enough to include a real version, 2.95, under the name kgcc.

3.5.3 Allocating on the Stack

The following snippet of C++ code seems innocuous.

```
void sillyCopy(int length, char* data) {
    char tmp[length];
```

```
    for (int i = 0; i < length; i++) {
        tmp[i] = data[i];
    }
}
```

Assuming the input is valid (`length` is equal to or less than the size of the char array pointed to by `data`), this function should work. It's a worthless function because it doesn't do anything with the copy of data it stores in `tmp`, but it's logically correct. Notice how `tmp` is allocated on the stack instead of the free store so we don't have to worry about deleting the pointer.

The problem with the code is that the `length` variable may be larger than the size of the stack, thus causing a segmentation fault for apparently no reason. To determine how large the stack is under UNIX variants, run `ulimit -s`. In testing RedHat's was found to be unlimited (although `/usr/src/linux/include/linux/sched.h` says 8192 KiB), Debian and Solaris report 8192 KiB, IRIX 65536 KiB, and Mac OS X only 512 KiB. Therefore, if you need to allocate over half a MiB of data, you shouldn't do so on the stack. Ideally, only small plain old data (POD) objects should be allocated on the stack for maximum portability.

3.5.4 auto_ptr

To overcome the burden of properly `new`ing and `delete`ing objects from the free store, C++ provides the `auto_ptr` template. When the `auto_ptr` object goes out of scope, it automatically `delete`s the pointer it holds. This is not only useful for when a function returns normally, but also when it returns because of an exception, as it will also clean up the object properly. However, Boost's[6] `shared_ptr` is often a better choice because it can be used in the C++ Standard Library containers. More discussion on the topic can be read in Using auto_ptr Effectively[39].

3.6 Problems Encountered

Despite extensive reuse of existing code, several technical hurdles had to be overcome.

3.6.1 Refreshing the Screen

xawtv's method of refreshing the screen was written for a different toolkit, Xt, and simply can't work under Qt. This feature had to be implemented in *grabber* via accessing a platform-specific hook provided by Qt and is the only example of such a use known by the author.

3.6.2 Calling Interactive External Programs

It wouldn't be wise to statically integrate code from a security related application into *grabber*. What if a vulnerability were found? All *grabber* installations would need to be patched, which is a complicated affair. Generally this issue is avoided by programs accessing other code through dynamic libraries. In this paradigm, only the broken library would need to be updated, and then all of the applications that access it are automatically updated by virtue of no longer dynamically calling the broken code.

When it came time to add secure copy (SCP) support to *grabber* so that it could upload webcam images without sending the user's password in plain-text over the network (as FTP does) the choice was made to simply access the SSH utilties installed on the local machine. However, because there aren't any high-level functions in the SSH library to copy files, *grabber* had to wrap around the scp binary as if the user were directly calling it from the command line.

Usually a child process is `fork()`ed off to call a function in the `exec()` family, but this wouldn't work because scp is an interactive process that outputs important information and requires a user to type a password which can't be specified on the command-line. A private key file can be used (this is the method camE relies on for its scp file copying feature), but typically users don't have a private key setup and simply type their login name and password at the prompt. Normally a set of read and write pipes can be attached to the running process for interacting with it. However, the scp binary tries to defeat a man-in-the-middle attack at this level by writing directly to the terminal device.

The `forkpty()` function can fork open a psuedo-tty which scp will believe is a normal terminal. Data can then be read and written, thus wrapping around the stand-alone binary as if the user were directly accessing it. However, when using pthreads, a call to `exec()` will

stomp on the footprint of the threads, thus causing the program to crash[30].

The solution to this problem is to fork before any threads are created. This child thread becomes a "surrogate parent" which the parent thread communicates with should it need to fork another child. In *grabber* this surrogate parent is the one that forks children to call `forkpty()`.

Finally, a communication link has to be established between the forked process that's calling scp and the parent process that's running the GUI and creating the requests. Named pipes didn't seem to work, nor did shared memory segments. However, Linux's Message Queue system did, therefore the two forks communicate via messages. Much later on it was discovered that the development system didn't have support for the shared memory faux filesystem mounted on `/dev/shm` compiled into the kernel, which may have caused the shared memory method to fail.

At the moment, the message queue system is buggy and often crashes after the first successful upload.

3.6.3 Opening a Server Socket

For a long time, if *grabber* was closed and then re-opened in under sixty seconds the following error message would be printed to the console

```
QServerSocket: failed to bind or listen to the socket
```

and the built-in webserver wouldn't work. Under Linux, sixty seconds is the default timeout for the CLOSE-WAIT state. Even though the program was done, the socket wouldn't be available again for sixty seconds. After much research, it was discovered that by default Qt wasn't setting the address reusable option on the socket. The working code in the `HTTPServer` constructor reads:

```
if (!ok()) {
    // allocate a new socket.  this object won't get deleted because that
    // would call close on the socket.  we're instead going to pass
    // control of the socket to ourself, the QServerSocket
    QSocketDevice* qsd = new QSocketDevice();
    qsd->setAddressReusable(true);
    if (qsd->bind(qsd->address(), port) && qsd->listen(backlog)) {
```

```
        //TODO: how about not printing it in the first place?
        qDebug("Fixed \"QServerSocket: failed to bind or listen to the"
               " socket\" error");
        setSocket(qsd->socket());
    } else {
       return;
    }
}
```

3.6.4 Signals and Slots in a Threaded Program

You can't send a signal from a thread. If the signal you emit happens to cause a change in the GUI the program will crash if Qt's main event thread (the GUI thread) also happens to be modifying the GUI. The solution is to create a CustomEvent and place it on Qt's event queue for processing by the GUI thread.

3.6.5 Data Sharing

§7.13 of Modern C++ Design[1] discusses smart pointer multithreading issues. It's possible that ZooLib[23] has a correct implementation in its ZRef class. Qt 4 will have native support for thread-safe implicit data sharing.

3.7 Releasing

3.7.1 grabber

Making a new release of *grabber* is painless. Simply type `make tarball` from the project's root directory. It will create a tarball release named after the VERSION variable in the Makefile.

Documentation

Doxygen [2], which supports JavaDoc-style comments, was used to document *grabber*. Only a few files are documented (EasterEgg, VBISource, Video*) and running `make doc` will generate them yielding a useful main page in `doc/index.html` and `doc/html/index.html`

3.7.2 Writing This Document

Text

LaTeX was used to format this document. The author should have stopped coding *grabber* and started writing this thesis April 2nd instead of April 16th. More topics that couldn't be discussed can be found in the timesheet logs on the enclosed CD-ROM.

Graphics

While adding images to this document with the `graphicx` package a few tricks were learned. A bad way to convert .png files to .eps files would be:

```
pngtopnm tmp.png | pnmtotiff > tmp.tiff
tiff2ps tmp.tiff > tmp.eps
rm -f tmp.tiff
```

A good way would be to simply do:

```
convert tmp.png tmp.eps
```

Whenever you want to include a graphic use the command

```
\includegraphics{graphicname}
```

and `pdflatex` will automatically use the file named `graphicname.png` and `latex` will use automatically use `graphicname.eps`. It's just that easy.

Camera Ready Copy

The BibTeX and MakeIndex packages as well as `pdflatex` were used in creating a final version of the document. A final version has been included on the enclosed CD-ROM.

Chapter 4

Future Work

As always, the order of priority for implementing future work should be based on user needs. A few general items to fix include:

- Porting *grabber* to Mac OS X probably wouldn't be too difficult. There is a version of Qt for OSX and maccam[16] comes with a small sample program that reads frames. Looking at its source code would probably make figuring out the appropriate calls to the QuickTime [2] library pretty easy.

- It'd also be nice to be able to pass in command-line arguments to perform various actions, much like several of the programs (camE and motion) I've incorporated into *grabber*. At the very least a quick little `-grabjpg` flag that would execute code to grab the current frame and save it as a JPEG without opening the GUI would be handy.

- The main Makefile in *grabber* needs to be re-written so that it doesn't recursively call the Makefiles in the subdirectories. Using make recursively doesn't handle the dependencies between files in subdirectories correctly (see §3.5.1).

- When the message queues are full, the command that must be executed to clear them is only printed to the terminal. Someone starting grabber from a shortcut will never see this standard output, so it'd be nice if a little message box popped up telling the user what they should do. Better still, the dialog should ask the user if they want *grabber* to fix the problem and restart itself.

- Use some form of source control, such as Subversion[11] or GNU Arch[27]. CVS is showing signs of age and should probably be avoided.
- Volunteers fluent in a foreign language are needed to produce accurate translations.

And there's also more specific work to be finished in several classes.

4.1 HTTPServer

- This should probably be re-written as a single-threaded multiplexing I/O server. Doing so is probably a good idea for scaling well to high loads as it will increase the likelihood of cache hits[45].
- *grabber* needs to not send out streaming data as fast as possible. The client end (Netscape in particular) will sometimes buffer up the data and fall very far behind the current image.

4.2 Label Editor

- When a highlighted item is deleted, the next one is selected, but not highlighted. It should be highlighted as well.
- It might be neat to have some way to actively read the labels from a text file, like NowPlaying[17].

4.3 Live Streaming

- A splitter widget needs to be inserted between the list of connections and the HTTPd log.
- The Start/Stop buttons should be replaced with a single button.
- This may be a bug in Qt, but text in the HTTPd log is always inserted below two empty lines at the top. There shouldn't be any blank lines at the top.

4.4 Motion Capture

- This feature and the widget and all of the menu links to it should be probably be renamed "Motion Detection."
- The size of the video input should be fixed when recording a movie file.
- Adding an option that would output to numbered JPEGs (1.jpg, 2.jpg, etc.) would be nic .
- The difference algorithm should be updated with the logic from the most recent version of motion.

4.5 Options

- The aspect ratio and full-screen resolution features need to be implemented.
- Perhaps the channel editor from Zapping should be incorporated into *grabber*.

4.6 WebCam

- The mechanism to upload images via SCP needs to be more throughly tested and debugged.
- Suspending the application currently breaks the message queues. This should be fixed.
- Add support for SSH identity files.
- Add support for HTTP PUT and active FTP (Qt only supports passive FTP).
- Allow the "file://" protocol to be used. It should just copy the file to somewhere on the filesystem.

4.7 VBI

- Closed captions should be overlayed on the incoming video instead of merely displayed in the VBI Window widget.
- Zapping[35] may have a more mature VBI decoding library called ZVBI. If so, it would behoove *grabber* to use it instead of the current implementation.

Chapter 5

Conclusion

Even in it's current unfinished state, *grabber* is arguably the most powerful video device application currently available. The features that are already implemented satisfy the majority of user tasks discussed in §1.2 and *grabber*'s flexible design should make adding features to support the remaining tasks easy.

The research, design, and development of *grabber* has been an amazingly educational experience. Learning new techniques such as the Generation Gap, new tools such as Valgrind, and a more in-depth understanding of C++ have already helped the author in other software development projects. The impressive size of *grabber* itself (~16,000 lines) stands as proof to the knowledge that has been obtained in order to manage such a large and complex application. Perhaps even more impressive is the fact that at one point *grabber* contained over 20,000 lines but had less features. The skills involved in refactoring away duplicate code and making the most of the available API can't be learned by reading a textbook, they have to come from experience.

As final proof of this research's success, *grabber* is actively used ever day. The author does indeed eat his own dog food when he wants to use his video devices. From streaming the audio and video of a Colby student's appearance on Jeopardy over the network to a web browser in the computer lab to taking pictures out the window of the previous night's snow to watching the nightly news, *grabber* can and does do it all.

The author recommends every liberal arts student of computer science strive to undertake a similiar self-directed software engineering project. The curriculum at liberal arts colleges

is often limited by a small faculty that only has the time to teach a small number of courses. Reduced course offerings can leave knowledge gaps that the IT industry demands potential employees fill before being hired. An independent study that focuses on solving the real world computer science problems a student hopes to work on after graduation is an excellent solution to this dilemma.

5.1 The Future of grabber

In the near future, *grabber*'s source code will be released to the Internet community as an open source project. As such, *grabber* will have the ability to evolve and fit the needs of developers and users around the world. Although the author would like to see personal video recorder features added to *grabber*, perhaps the user demands for some other feature, such as advanced deinterlacing will take precedence. It's hard to predict exactly how *grabber*'s entry into the V4L appliction landscape will be perceived, but it's sure to be be an interesting and exciting journey.

5.2 Contacting the Author

The author plans to continue working on *grabber*. Please contact him if you would like to develop *grabber* further or have plans to use it in another project. A search on the Internet should reveal a current e-mail address for Kevin Septor, but failing that please try <kjseptor@alum.colby.edu>.

Appendix A

User Manual

A.1 Installing

- Install all of the required libraries (asound, lirc_client, jpeg, qt) from the attached CD.
  - Use Qt version 2.3.2 with the lupdate and lrelease tools from version 3.0+
- Download the tarball, or get it from the attached CD.
- `tar -xzvf grabber-yyyymmdd.tar.gz`
- `cd grabber-yyyymmdd/`
- `make`

  If compilation fails, edit the variables in src/Makefile to match the system's install locations for the required libraries and re-run `make`
- `mkdir ~/.grabber`
- `./src/grabber`

A.2 Basic Use

Above is the main interface presented to the user when *grabber* is started. Using either the top menu bar or the right-click popup menu the following menus are available.

A.2.1 Input

Easy access to functions which modify the input from the current video device.

Size

Contains various useful preset sizes (e.g. 320x240, the defacto webcam image size), dynamically updated minimum and maximum size menu entries, and an entry to toggle full-screen mode.

View

Lists the various methods of displaying the incoming video such as overlayed, deinterlaced, etc.

Control

Allows the user to adjust the volume and channel settings, if the attached device supports these features.

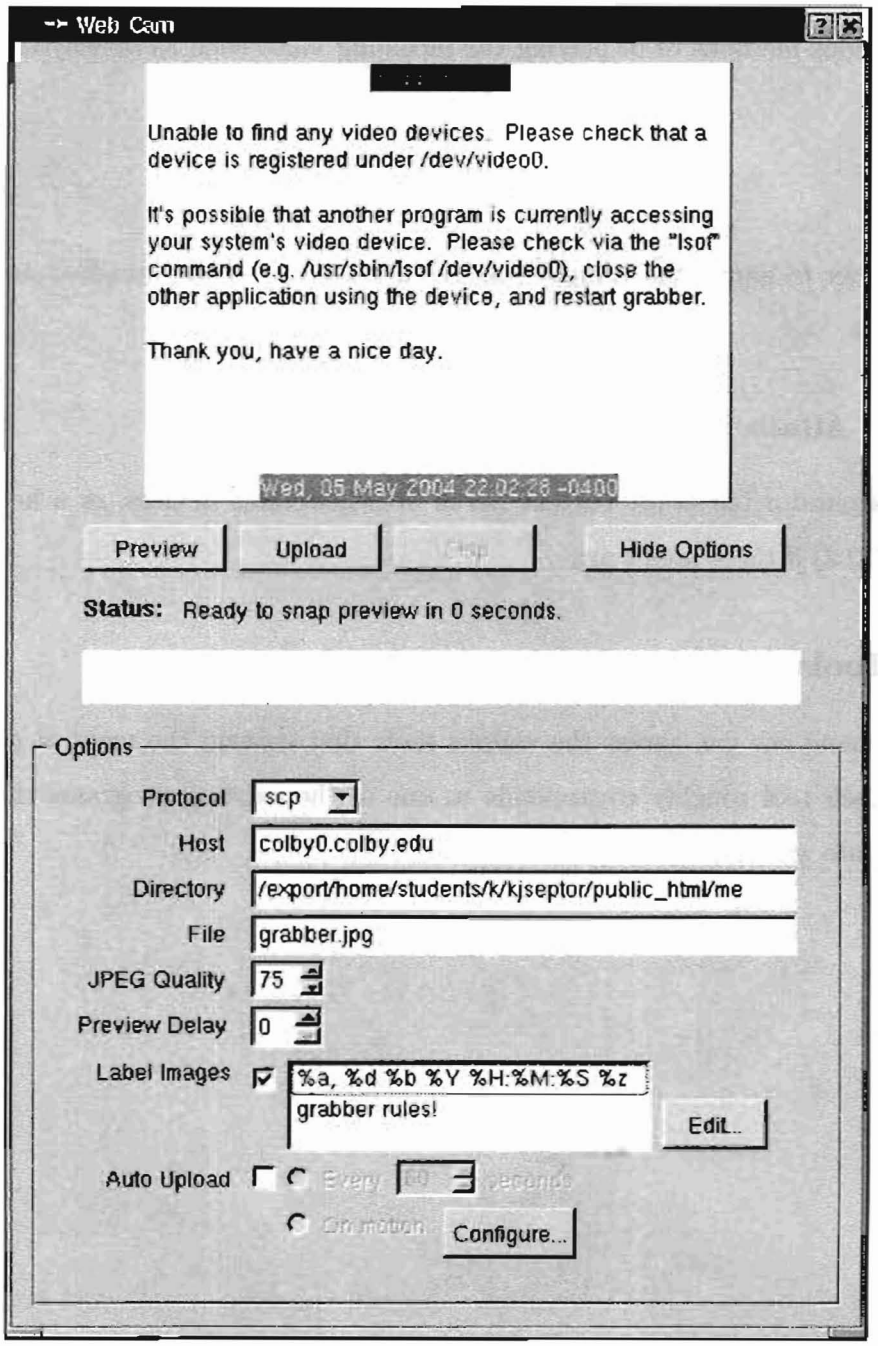
The List of Attached Video Devices

Below the separator bar is the current list of attached video devices, or a helpful message (show in §A.2.2) if there aren't any.

A.2.2 Tools

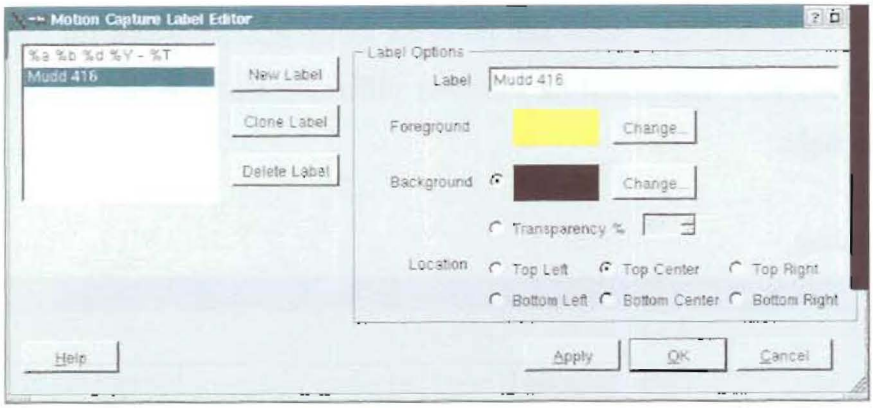
Under this menu one can access the various tools that contain the meat of *grabber*'s functionality. Each tool roughly corresponds to one of the existing programs that have been assimilated into *grabber*.

Web Cam



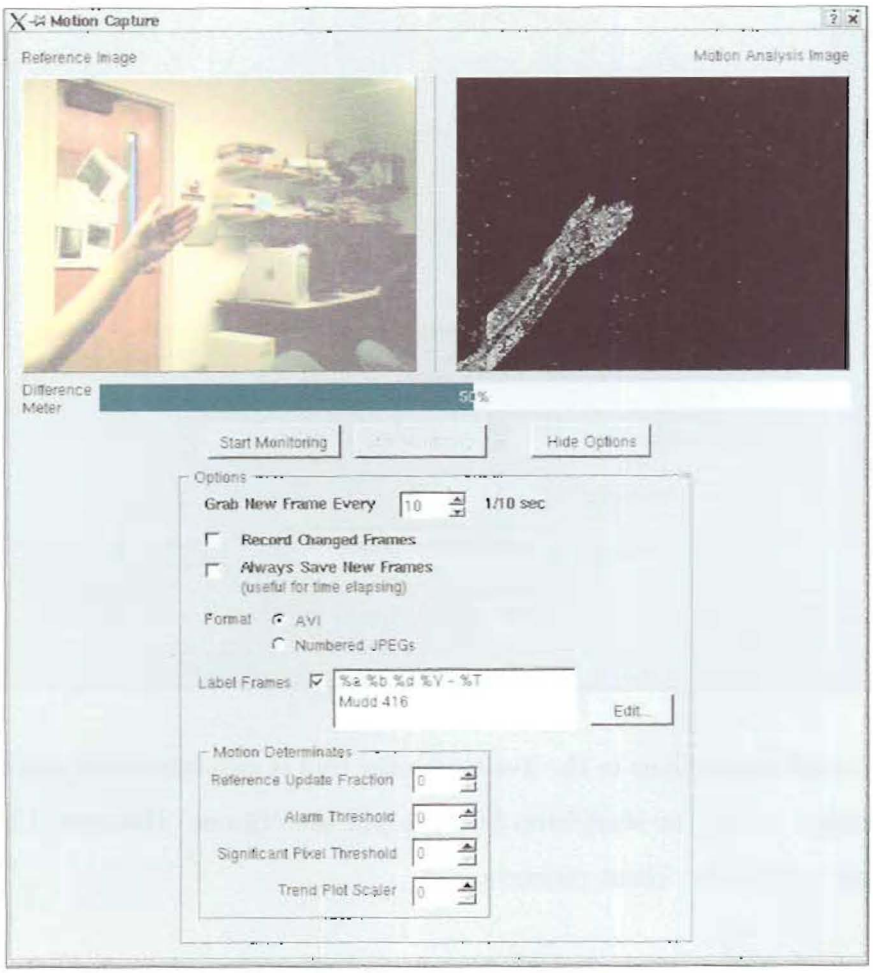
The `Preview` button takes a snapshot based on the parameters in the options. The `Upload` button must be pressed to actually upload the image to the remote site specified by the options.

Label Editor



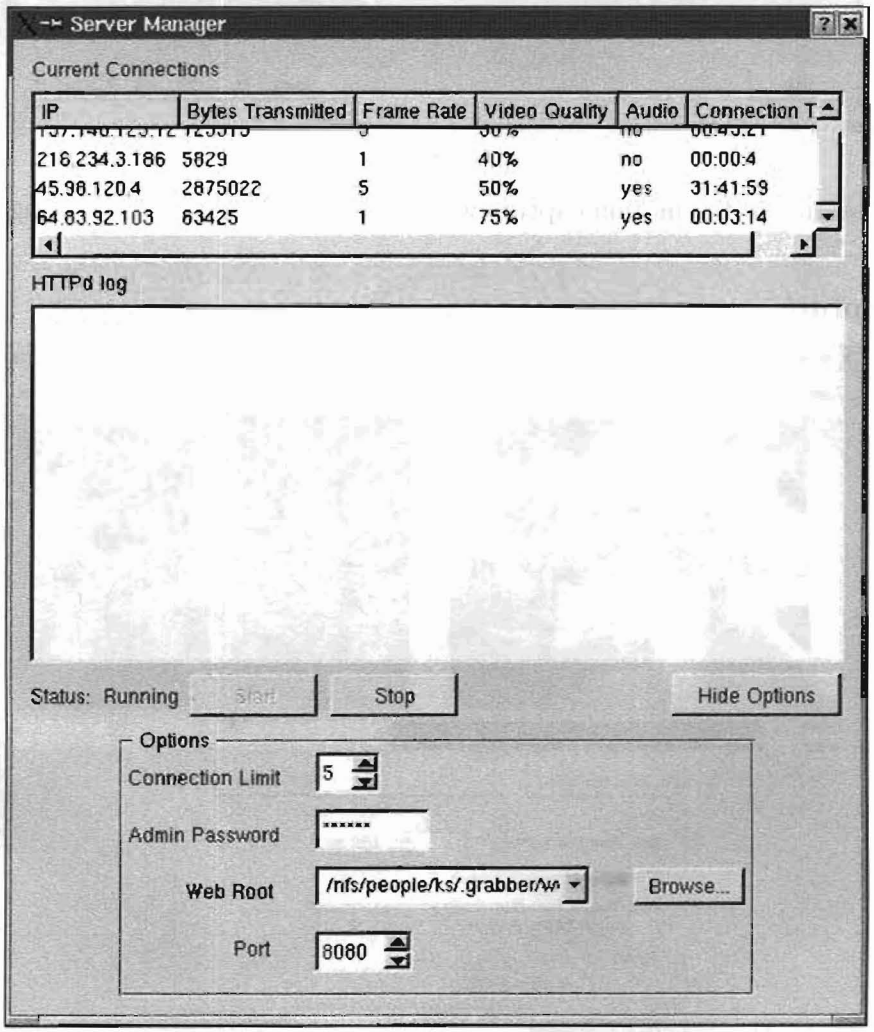
Both the webcam and the motion capture widget have a dialog for easy editing of the labels.

Motion Capture



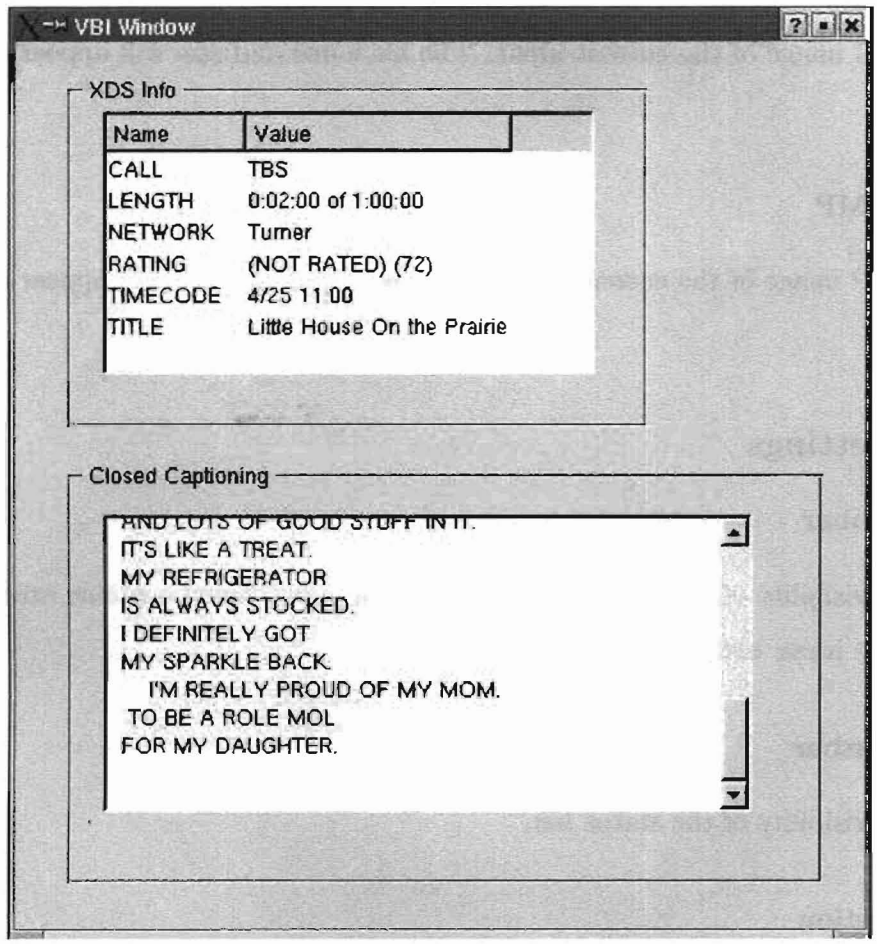
Select the desired options and then press `Start Monitoring` to begin capturing and comparing frames. The left side shows the previous image against which the current image is compared for motion. The resulting intensity difference between the two is shown in the image on the right.

Live Streaming



The list of current connections in the live streaming tool is non-functional and only intended as a place holder, as are the start/stop buttons and the options. However, the HTTPd log does work and will display client requests.

VBI Window



In this screen capture the closed captioning and XDS data was recorded from a toothpaste commercial during a show on TBS, as the GUI explains. The screen shot is outdated and current versions also have a `Save Buffer As...` button which will bring up a file chooser and save the Closed Captioning data to that file. This feature is very useful for creating movie scripts. Just pop in a DVD, play the whole movie, and save the buffer to a file.

Capture JPEG

Saves a JPEG image of the current input. The file name and size will appear on the status bar.

Capture PNG

Saves a PNG image of the current input. The file name and size will appear on the status bar.

Capture BMP

Saves a BMP image of the current input. The file name and size will appear on the status bar.

A.2.3 Settings

Show Menubar

Toggles the visibility of the menu bar. A right-click popup menu contains exactly the same entries as the menu bar.

Show Statusbar

Toggles the visibility of the status bar.

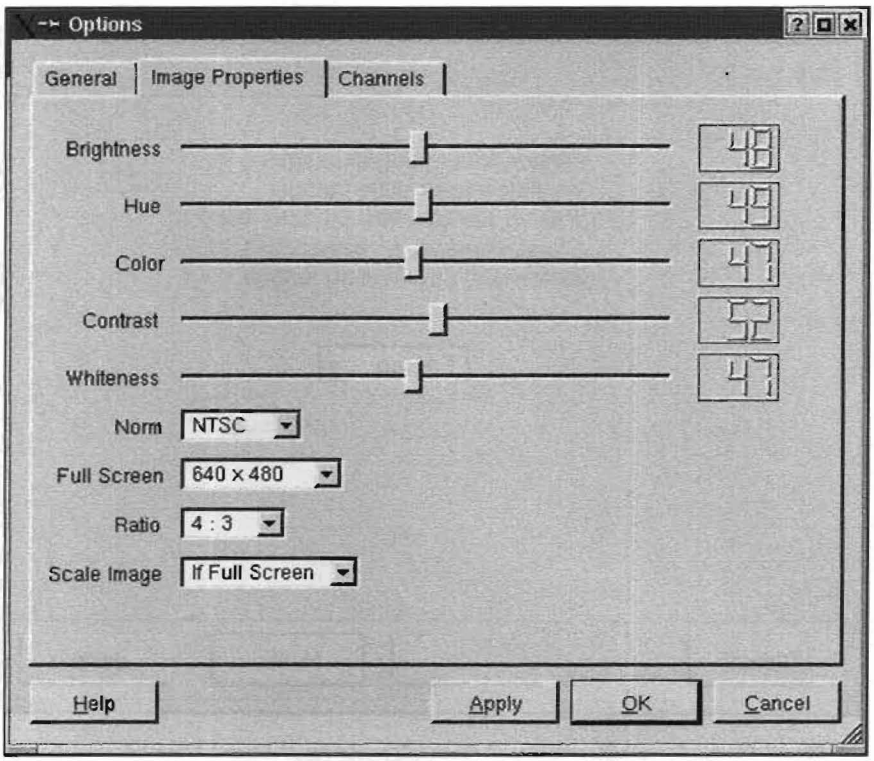
Closed Caption

This button would toggle closed caption display over the video input, but unlike tvtime the feature hasn't been implemented yet.

Some cool option

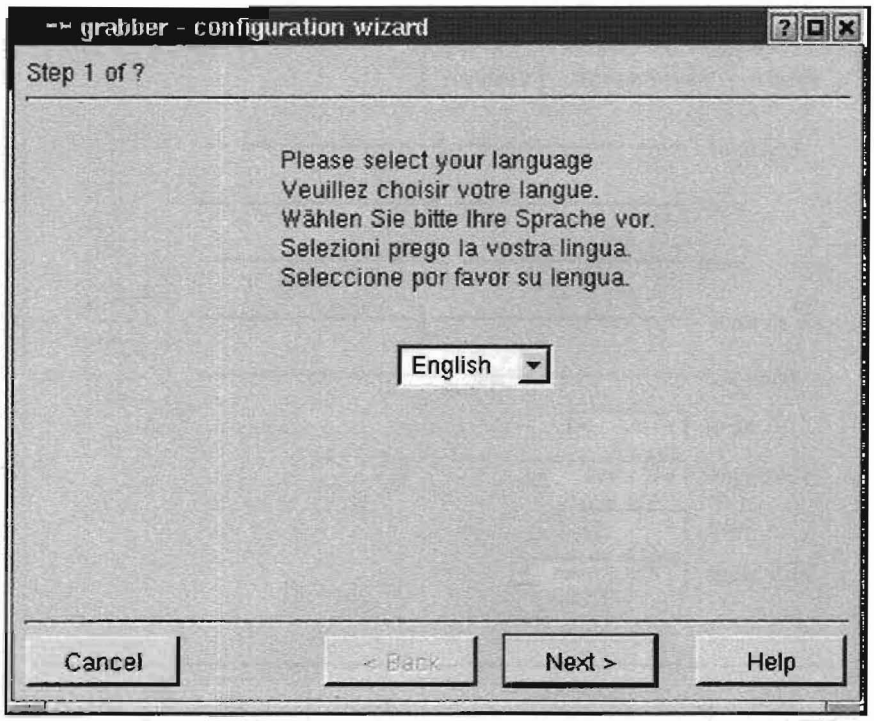
This option doesn't actually do anything, it's just a place holder in an attempt to mimic the KDE User Interface Guidelines[7].

Configure Grabber



This dialog has the application global options that effect all of the individual tools. Only the sliders in the `Image Properties` tab and mixer selection option under the `General` tab are functional.

Setup Wizard

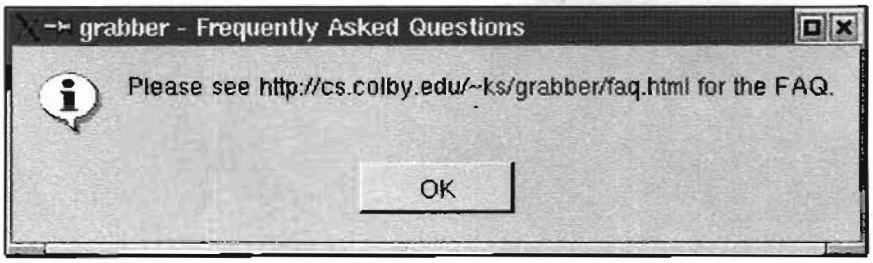


The configuration wizard will only fill in a very minimal config file in the very last step and ignores most of the choices you actually make.

A.2.4 Help

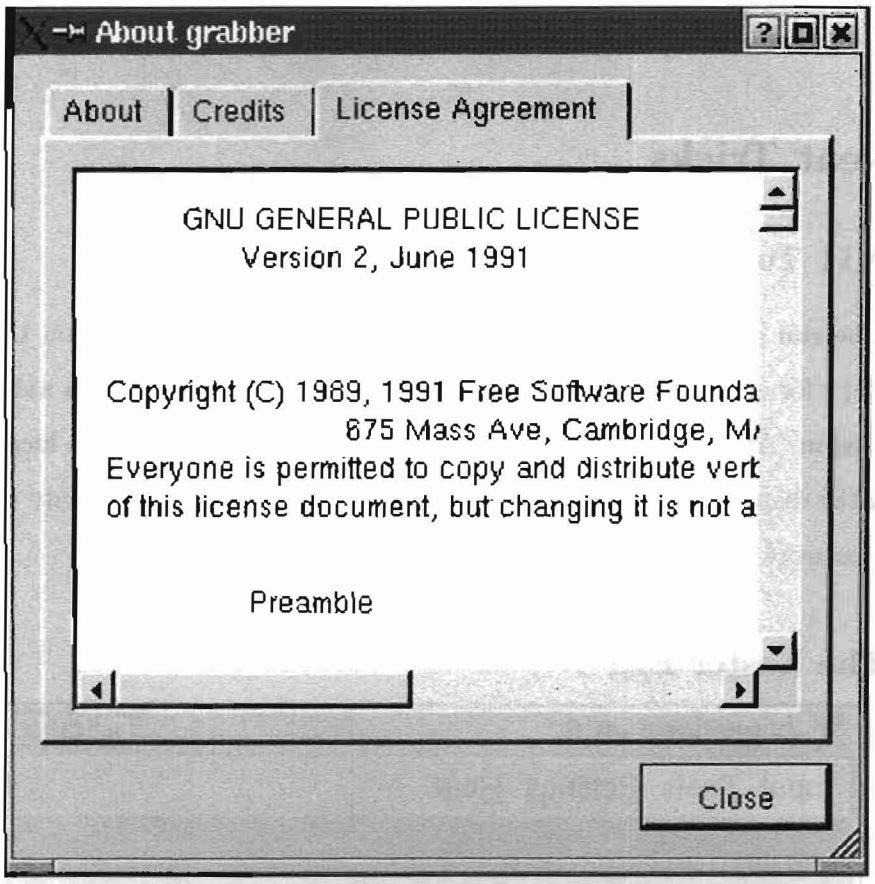
The `Help` menu contains a set of entries that may actually help you with using *grabber*.

FAQ



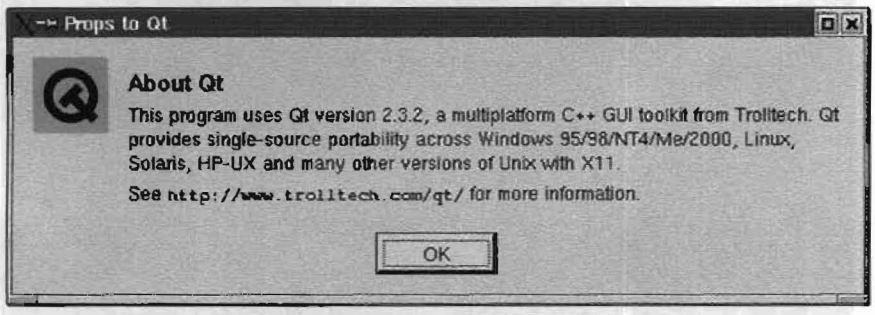
Just points to a URL that has a very sparse list of frequently asked questions because of *grabber*'s awesome usability and low learning curve.

About



Shows the credits and software license.

About Qt



A little prefabricated blurb about Qt. Sadly, this dialog is *modal* and since it's provided by Qt, there's no way to change its behavior. Regardless, Trolltech does provide Qt/X11 for

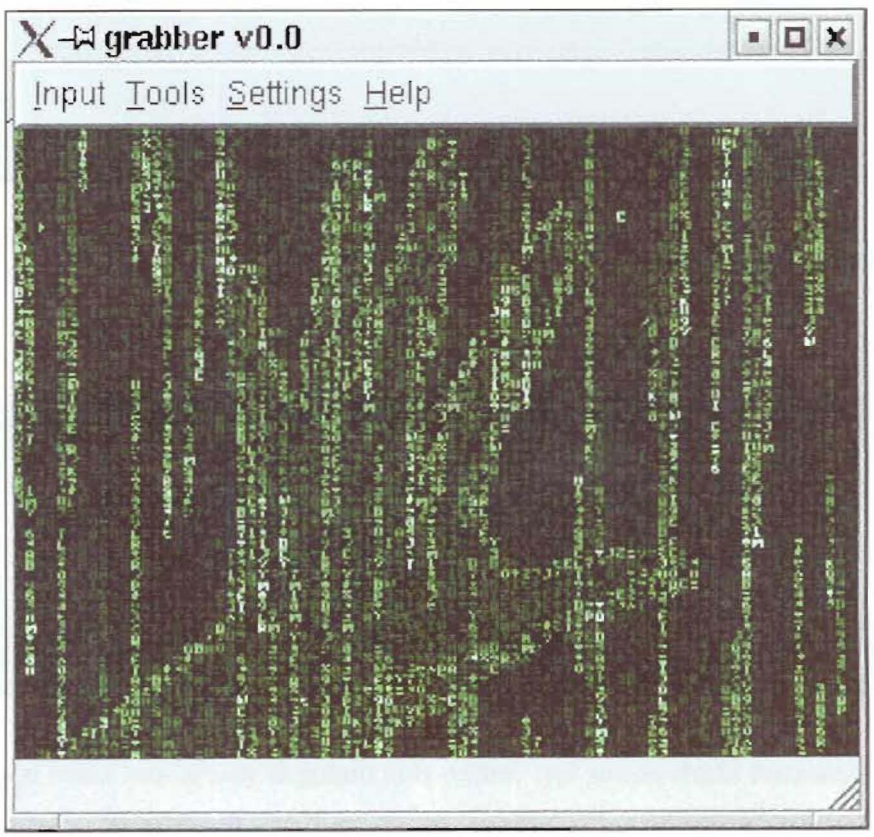
free and at the very least it's appropriate for *grabber* to contain this little plug as users may known nothing about Qt.

A.3 Neat Tricks

A.3.1 X11 Tunneling

grabber can be run on a host remote machine but have its GUI appear on the local client machine if X11 forwarding is used. The simple and secure way is to run `ssh -X <remote server>` to log in. Afterwards, start *grabber* from the command line and a local window will appear with the display. This uses a lot of bandwidth and will probably only work well over a local area network.

A.3.2 The Easter Egg



To toggle the Matrix effect easter egg, simply type **xyzzy**, which is the magic word from Adventure.

A.4 Fixing grabber

A.4.1 Clearing the Message Queues

If *grabber* isn't cleanly shut down, it can't clean up the system message queue it has allocated to transmit data between the main application and the SCPQNetworkProtocol class. To rectify this situation the user must type:

```
ipcs -q | grep 'id -un' | awk '{print $2}' | xargs ipcrm msg
```

at the command line. *grabber* can detect when the queues are filled and prints this message to the terminal.

Appendix B

Programmer's Guide

B.1 Quick Start

`main.cpp` has the `main()` method. It performs some initialization and creates an instance of the `Grabber` class, which in turn creates all of the widgets and sets up the menus.

B.2 UML

An out of date UML diagram has been included on the accompaning CD-ROM. It was created with ArgoUML[5].

B.3 The GAV File Format

GAV stands for grabber audio video. The webserver just streams them out. All blocks are 1 second in duration. The format of the chunks is as follows:

- 4 bytes - block header / magic number, 0x67617630 "gav0" 1,734,440,496
- 1 byte - frames per second, 2's complement
- 2 bytes - length of audio file in bytes, 2's complement, or 0 for no audio
- ? bytes - (optional) 8 bit, u-law (mu-law), 8000 Hz, one-channel, SUN/NeXT ".au" file

- 4 bytes - length of first video frame in bytes, 2's complement
- ? bytes - JPEG image
- ... repeat 4 byte length read / ? byte image read until frames read = FPS

The audio format should be

- encoding = PCM_SIGNED
- sample rate = 8000.0
- sample size in bits = 16
- channels = 1
- frame size = 2
- frame rate = 8000.0
- big endian = true

B.4 Development Hardware

While developing *grabber* an ATI TV Wonder was used with the bttv driver distributed in the 2.4.x stable branch of the kernel, specifically version 2.4.23 of the kernel[13]. A Logitech Quickcam Express USB with the qc-ga 0.40d and qc-usb 0.6.0 drivers[9], and an Intel model number CS330 webcam with the spca50x 0.30 and CVS drivers[10] pulled on March 6th, 2004 were also used.

Bibliography

[1] Andrei Alexandrescu. *Modern C++ Design*, pages 184-187. Addison-Wesley, 2001.

[2] Apple. QuickTime. `http://developer.apple.com/quicktime/`.

[3] Unknown Author. Human Vision Issues. `http://nfg.2y.net/games/ntsc/visual.shtm`.

[4] Unknown Author. Xlib programming manual. `http://www.tronche.com/gui/x/xlib/function-index.html`.

[5] Multiple Authors. ArgoUML. `http://argouml.tigris.org/`.

[6] Multiple Authors. Boost. `http://www.boost.org/`.

[7] Multiple Authors. KDE User Interface Guidelines. `http://developer.kde.org/documentation/standards/kde/style/basics/`.

[8] Multiple Authors. LIRC - Linux Infrared Remote Control. `http://www.lirc.org/`.

[9] Multiple Authors. qc-usb. `http://qce-ga.sourceforge.net/`.

[10] Multiple Authors. SPCA50X USB Camera Linux Driver. `http://spca50x.sourceforge.net/spca50x.php`.

[11] Multiple Authors. Subversion. `http://subversion.tigris.org/`.

[12] Multiple Authors. The Apache HTTP Server Project. `http://httpd.apache.org/`.

[13] Multiple Authors. The Linux Kernel. `http://www.kernel.org/`.

[14] Billy Biggs. tvtime. `http://tvtime.sourceforge.net/`.

[15] EffecTV crew. EffecTV. `http://effectv.sourceforge.net/`.

[16] dirkx and mattik. macam. `http://webcam-osx.sourceforge.net/`.

[17] DrPizza, aka PeterB. NowPlaying. `http://drpizza.arsware.org/nowplaying/`.

[18] Free Software Foundation. GNU General Public License. `http://www.gnu.org/copyleft/gpl.html#SEC1`.

[19] Free Software Foundation. GNU Make. `http://www.gnu.org/software/make/`.

[20] Martin Fowler. *Refactoring*. Addison-Wesley, 2000.

[21] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[22] Tom Gilbert. camE. `http://linuxbrit.co.uk/camE/`.

[23] Andrew Green. ZooLib. `http://zoolib.sourceforge.net/`.

[24] Independent JPEG Group. libjpeg. `http://www.ijg.org/`.

[25] Sun Microsystems Inc. Java. `http://java.sun.com/`.

[26] Gerd Knorr. xawtv. `http://linux.bytesex.org/xawtv/`.

[27] Tom Lord. GNU Arch. `http://www.gnu.org/software/gnu-arch/`.

[28] Scott Meyers. *Effective C++*. Addison-Wesley, 2nd edition, 1998.

[29] Peter Miller. Recursive Make Considered Harmful. `http://www.pcug.org.au/~millerp/rmch/recu-make-cons-harm.html`.

[30] Bradford Nichols, Dick Buttlar, and Jacqueline Proulx Farrel. *Pthreads Programming*, pages 186–190. O'Reilly, 1996.

[31] Jef Raskin. *The Humane Interface*, page 203. Addison-Wesley, 2000.

[32] Isaac Richards. MythTV. `http://www.mythtv.org/`.

[33] Damien Sandras. GnomeMeeting. `http://www.gnomemeeting.org/`.

[34] Karsten Scheibler and Christoph Bartelmus. LIRC - Technical Documentation. `http://www.lirc.org/html/technical.html`.

[35] Michael Schimek. Zapping. `http://zapping.sourceforge.net/`.

[36] Julian Seward. Valgrind. `http://valgrind.kde.org/contact.html`.

[37] Joel Spolsky. The Joel Test: 12 Steps to Better Code. `http://www.joelonsoftware.com/articles/fog0000000043.html`.

[38] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 3rd edition, 1997.

[39] Herb Sutter. Using auto_ptr Effectively. `http://www.gotw.ca/publications/using_auto_ptr_effectively.htm`.

[40] Trolltech. Qt. `http://www.trolltech.com/products/qt/`.

[41] Trolltech. Trolltech documentation. `http://doc.trolltech.com/`.

[42] Dimitri van Heesch. Doxygen. `http://www.doxygen.org/`.

[43] John Vlissides. Generation Gap. `http://www.research.ibm.com/designpatterns/pubs/gg.html`.

[44] Jeroen Vreeken, Kenneth Jahn Lavrsen, and Folkert van Heusden. motion. `http://motion.sourceforge.net/`.

[45] Matthew David Welsh. *An Architecture for Highly Concurrent, Well-Conditioned Internet Servies*. PhD thesis, University of California at Berkeley, 2002. `http://www.eecs.harvard.edu/~mdw/papers/mdw-phdthesis.pdf`.

All URLs were checked on May 12th, 2004.

Index