Automatic Generation of Time-lapse Animations

Stephen B. Jenkins[*]
Institute for Aerospace Research
National Research Council
Ottawa, Canada

Presented at YAPC::Canada
Ottawa, Ontario
May 2003

Abstract

A simple system for the automatic generation of time-lapse animations has been created at the Aerodynamics Laboratory of the Institute for Aerospace Research, National Research Council of Canada (NRC). The system consists of two small Perl programs and a static web page, and relies heavily on two open source software packages: LWP::Simple, and ImageMagick. The first program, tlget.pl, runs for several hours each weekday and is responsible for the periodic acquisition (and storage) of images from a web-based camera. The second program, tlmpeg.pl, is responsible for the conversion of groups of JPEG format images into MPEG format animations. Typically, tlmpeg.pl is executed each weekday night, and generates three animation files, one from the past day's images, one from the past week's images, and one from the past month's images. A static web page has been created to display the image most recently acquired by tlget.pl, and to provide links to each of the three animation files most recently generated by tlmpeg.pl.

1 Introduction

During the construction of a building to house the new NRC Gas Turbine Environmental Research Centre (GTERC), it was suggested that a continuously updated image of the building site be made available to all interested parties via a web page.

Having recently completed another project involving the dynamic manipulation of images using Perl [Jenkins & Jordan], the author consented to undertake this task, and expand upon it. Rather than provide only static images of the construction site, it was decided to add the ability to view time-lapse animations of the activity over the previous day, week and month.

2 Hardware

The hardware for the system consists of only two components: a camera, and an Intel-based PC. The camera selected for this task was an AXIS 2100 Network Camera, manufactured by Axis Communications. It should be noted that this is not a simple "web-cam" that relies on an external PC for its connectivity. Rather, it is a fully networkable device that incorporates an internal computer running the Apache web server on an embedded version of Linux. The camera requires only power and network connections to be completely functional.

The computer responsible for image storage and manipulation, and for serving the web page, is a rather generic PC: 1.6 GHz P4, 1 Gb memory, 80 Gb hard drive. The processor speed and hard drive size constrain, respectively, the time required to generate the MPEG animations, and the number and size of stored images. The memory size imposes a limit on the maximum number of frames in the generated MPEG files (more on this later).

[*] Senior Programmer/Analyst – http://www.nrc.ca/~jenkins

3 Support software

In keeping with a carefully considered development philosophy [Jenkins, 2001], all software used for the creation of the time-lapse animation system is open source. The operating system, Linux, was installed from a RedHat 7.3 distribution on CD. Perl, LWP::Bundle [Burke, 2002] and Mail::Mailer were downloaded from the Comprehensive Perl Archive Network (CPAN) and installed. ImageMagick was downloaded and built with the LWZ and PerlMagick [Wallace, 2002] options enabled. The Apache web server and the mpeg2vidcodec were also downloaded and installed. While the re-installation of Perl, ImageMagick, and Apache may have been, strictly speaking, unnecessary (they are all included in the RedHat distribution), it is the author's opinion that the time required was well spent, as it ensured that these critical components were configured in the expected manner.

4 Proof-of-concept

Conceptually, generating time-lapse animations is a rather simple, two-step process. First, one must periodically acquire images from a camera, storing them to disk. Then, after the desired number of images has been captured, they must be combined into a single animation file. The first task is easily accomplished with the `mirror()` command of the `LWP::Simple` module. Through the use of the ImageMagick image manipulation package and the mpeg2vidcodec MPEG generation program, the second step is also easily performed.

The program shown below (Listing 1) was created as a simple proof-of-concept, while awaiting procurement of the camera. It is made up of three parts: an initialization section, an image capture section, and an MPEG generation section.

```
#!/usr/bin/perl -w
use strict;
use LWP::Simple;

$|++;

my $interval = 60;
my $camURL = 'http://parliamenthill.gc.ca/text/newhillcam.jpg';

$SIG{INT} = \&catch_int;

# capture the images
while(1) {
    my @t = localtime(time);
    my $imgname = sprintf("img%04d%02d%02d-%02d%02d-%02d.jpg",
                          $t[5]+1900,$t[4]+1,$t[3],$t[2],$t[1],$t[0]);
    print "Getting $imgname ... ";
    mirror($camURL, $imgname) || warn "Oops: $!";
    print "done!\n";
    sleep($interval);
}

# generate the animation
sub catch_int{
    print "\n\nDo you want me to create an mpeg? ";
    if( (<STDIN>) =~ m/y/i ){
        my @t = localtime(time);
        my $mpegname = sprintf("%04d%02d%02d-%02d%02d.mpg",
                               $t[5]+1900,$t[4]+1,$t[3],$t[2],$t[1]);
        print `convert -adjoin *.jpg $mpegname`;
    }
    exit(0);
}
```

Listing 1: Proof-of-concept program

In the first part of the initialization section, strict naming is enabled, the `LWP::Simple` module is loaded, and output buffering is turned off. The next two lines initialize variables with the desired wait time between image captures, and with the URL of the web camera's image file. Because the camera being purchased for the project was not yet available, it was necessary to find a publicly accessible camera. This was not as simple as it first appeared, since the terms-of-use of many web-cams specifically prohibit both continuous sampling and the storing of images. In the end, it was decided to use a camera owned and operated by another Government of Canada department. The final line of the initialization section sets a signal handler for `SIGINT`. This causes the program to execute the `catch_int` subroutine when the user types a `<CTRL-C>`.

The image capture section begins by setting up an infinite loop. Inside this loop, the current date and time are read and are used to create a filename of the format `img`*`YYYYMMDD-HHmm-SS`*`.jpg`. The `mirror()` routine of the `LWP::Simple` module is invoked with the camera URL and the newly created filename. It acquires the image via the Web, and saves it to the local hard drive. The program then waits for the interval time to pass, and repeats the process.

The final section – the signal handler subroutine – begins by prompting the user to determine if an animation file is to be created. If yes, a filename with a `.mpg` extension is constructed using the current date and time. Then, the `convert` command from the ImageMagick package is executed with the `adjoin` option, a list of the image files to be included in the animation, and the previously created MPEG filename. The `convert` command determines the input and output file formats (JPEG and MPEG respectively) from the filename extensions and automatically makes use of the mpeg2vidcodec, creating a default configuration file for it. After the generation of the MPEG animation has been completed, the subroutine causes the program to exit.

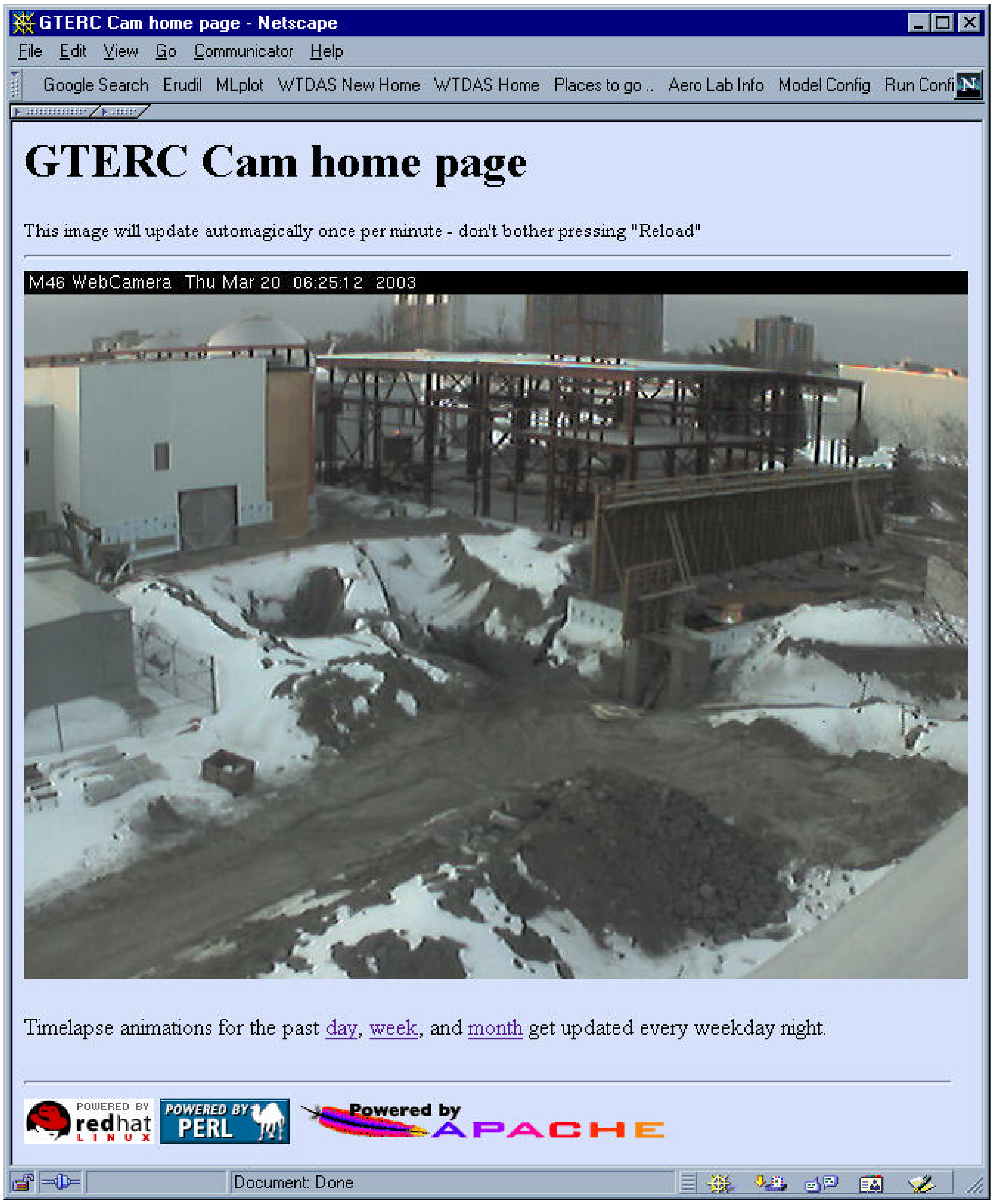
While simple, the test program not only provided proof that the method was feasible, it also allowed the author to experiment with various image capture rates and animation file sizes. Although a 30-second pause between captures generated an esthetically pleasing animation file, it was deemed impractical because of the large number of images to be stored. Each day would require, on average, 12 hours X 60 minutes X 2 images per minute, or 1440 files. Since each image file is approximately 50Kb in size, that would mean 72Mb of disk storage per day for a project that was expected to run continuously for 10 to 12 months. In addition, when converted into an MPEG at one image per frame (25 frames per second), each animation file would run for ~58 seconds and would require an additional 8 Mb of disk space. It was thought that this size of animation file was unnecessarily large. After trying several combinations of capture rate and animation file sizes, it was decided to use a 60 second inter-image delay, and to construct each MPEG from 400 images, yielding a 2.2 Mb, 15 second animation. If it was later decided that these animations were not long enough, they could be regenerated from each day's complete set of images (approximately 700).

Although the test program worked well for its intended purpose, several things about it are inappropriate for an automated system expected to run unattended for many months, and to generate well over 100,000 files. The three most obvious shortcomings are: human intervention required for starting and stopping, a flat file-storage system, and inadequate error reporting.

5 Final system

For the final system, it was decided that the best solution would be to create two small programs, one called `tlget.pl` to capture and store the images, and the other called `tlmpeg.pl` to combine groups of images into time-lapse animations. Since the system was to operate unattended, the `cron` utility would be used to schedule the execution times of both programs. This also provided a simple mechanism to have the program acquire images only when construction activity was likely to occur (i.e. weekdays during daylight hours). Additionally, since both programs were to run unmonitored for long periods, notification of any errors requiring human intervention would need to be carried out by creating and sending email to an appropriate address.

When executed, `tlget.pl` first constructs a directory name of the form `TL`*`yyyymmdd`* using the current date. If that directory does not yet exist, it is created. The program then begins acquiring images, naming them in a fashion similar to that


GTERC Cam home page - Netscape
File Edit View Go Communicator Help
Google Search Erudil MLplot WTDAS New Home WTDAS Home Places to go .. Aero Lab Info Model Config Run Confi
M46 WebCamera Thu Mar 20 06:25:12 2003
POWERED BY redhat LINUX
POWERED BY PERL
Powered by APACHE
Document: Done

GTERC Cam home page

This image will update automagically once per minute - don't bother pressing "Reload"

Timelapse animations for the past day, week, and month get updated every weekday night.

Figure 1: System web page

used in the test program, and placing them in the current day's directory. In order to make the most current image available to users, the image file is also copied to a web-accessible location and renamed `tlcurrent.jpg`.

The animation generating program, `tlmpeg.pl`, is expected to be run each evening after `tlget.pl` has finished acquiring that day's images. Using `File::Find`, a list of all daily directories is created, along with lists of the names of the image files contained within each. The list of directory names is reverse sorted, so that the first one refers to the past day, the first five make up the past week, and the first twenty make up the past month. These three sub-lists are then used to create arrays of all image files captured during each of the three periods. Since the target number of frames to be used for each MPEG is 400, the three lists are culled to produce three arrays of 400 image files each. These final arrays are passed to the convert program to create animations that cover the past day, week, and month. The three newly generated MPEG files are stored in that day's directory, and are also copied to a web-accessible location and renamed `past_day.mpg`, `past_week.mpg`, and `past_month.mpg`.

To make `tlmpeg.pl` more flexible, an option was added to allow the date to be overridden from the command line. This allows the program to be run in a batch mode if there is ever a need to regenerate all of the animation files (if a longer running time were to be needed for example). If this option is exercised, the newly generated animation files are not copied into the web space.

To most users, the output of the entire process appears in the form of a simple web page that shows the most recently acquired image, and also provides links to the most recently generated set of animation files (Figure 1). Since `mlget.pl` and `mlmpeg.pl` change the contents of the files that the page refers to, it can be a static HTML document. This eliminates the requirement to have a CGI program dynamically generate the page.

The four `meta` tags shown in Listing 2 need to be included in the header portion of the system home page to ensure proper behaviour. The first line is required to cause the browser to automatically reload the page every 60 seconds. This guarantees that the user's view is updated for every new image. The next three lines disable the caching normally performed by the web browser, forcing the page and its image to be reloaded regardless of whether the contents appear to have been changed or not. All three of these lines must be added to the header to accomplish this simple task, because of the lack of a single, standard command that disables caching in all browsers.

For a select few interested parties, the Apache web server's `Indexes` option has been enabled in the `httpd.conf` file. This addition allows those users to browse through the daily image directories and view any of the image or animation files.

Only two difficulties of any significance were encountered during the creation and commissioning of this system – one software and one hardware. The first was a problem with the ImageMagick `convert` command discovered while working with the test program. That command appeared to consume a significant amount of memory. With 1Gb of ram, the largest animation files that could be generated were on the order of 450 frames in length. Requests for larger files resulted in significant memory swapping, eventually leading to disk thrashing, and even to OS crashes due to the lack of swap space. It was decided not to pursue the search for a solution, as an animation file size of less than the problem-inducing 450 frames was acceptable for this project. The second area of difficulty was the issue of camera placement. After some discussion, it was agreed that the best view was to be had from a building located just to the North of the construction site. The drawback to this location was the significant glare caused by the sunlight since the camera was pointed roughly southwards. This problem was especially apparent in winter, with the snow covering the ground and the sun low on the horizon. Several alternate camera locations were considered, but it was felt that none would provide significantly better images.

```
<meta http-equiv="Refresh" content="60">
<meta http-equiv="Expires" content="0">
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Cache-Control" content="no-cache">
```

Listing 2: Web page header tags

6 Concluding remarks

The feedback received about this project has been unanimously positive and the knowledge and experience gained is expected to be used in future applications at the Aerodynamics Laboratory. The project has reaped significant benefits for a system that required only about three weeks to complete, and whose cost, due to the use of open source software, was under $1000 CDN.

7 Acknowledgments

The author would like to thank Mr. S. Totolo for procuring and installing the camera.

8 References

Burke, S.M., *Perl & LWP*, O'Reilly & Associates, 2002.

Jenkins, S.B., *Open Source Software at the Aerodynamics Laboratory*, Linux Journal Issue #90, October, 2001

Jenkins, S.B. and Jordan, J., *A Description of the Software for a Cockpit Video Security System*, LTR-FR-199, Institute for Aerospace Research, National Research Council, Feb. 2003.

Wallace, S., *Perl Graphics Programming,* O'Reilly & Associates, 2002.