Monitorización de vídeo con Perl y una webcam

PESCA DE IMÁGENES

Las webcams son útiles para monitorizar. Usando módulos CPAN y algo de código C a medida, un script Perl puede ajustar automáticamente la exposición y pescar las imágenes más interesantes de un flujo de vídeo. **POR MICHAEL SCHILLI**

La mayoría de las webcams incluye algún tipo de software para Windows, que no vale para nada al usuario de Linux. Afortunadamente, las distribuciones más recientes incluyen Video4Linux, que nos proporciona controles sencillos para muchas cámaras USB. La Creative NX Ultra que he usado para este artículo ofrece salida para vídeo y cuesta entorno a 70$. De hecho, puede parecer un despilfarro de las capacidades de la cámara usarla como una simple webcam, pero es que simplemente estaba abandonada en un cajón en los laboratorios Perlmeister, por lo que, ¿por qué no usarla? La cámara no necesita suministro externo de corriente, y fue detectada inmediatamente por el subsistema de conexión en caliente cuando la enchufé a mi PC. La información de vídeo normalmente se almacena en */dev/video0*. El módulo de Perl *Linux::Capture::V4l* de CPAN se fija al dispositivo de entrada, captura la información de los fotogramas y permite al programador que modifique los parámetros de exposición, como la sensibilidad de la cámara, sobre la marcha.

El Listado 1 muestra una sencilla aplicación que fija en primer lugar la sensibilidad de la cámara a 40000 antes de capturar una imagen del flujo de vídeo, y la guarda en disco como archivo JPEG (véase Figura 2).

El módulo *Camcap* que usa *single* se muestra en el Listado 2. Añade una capa de abstracción para acceder al flujo de vídeo. El constructor de la línea 12 define unos cuantos parámetros, como la anchura y altura de la imagen, así como los valores máximo y mínimo del brillo *(br_min, br_max)*. El código usa entonces el módulo *Video::Capture::V4l* de CPAN para conectarse al dispositivo de vídeo, */dev/video0*. Si alguna otra entidad está ya obteniendo datos del dispositivo, la conexión falla.

El método *cam_bright()* definido en la línea 35 configura la sensibilidad de la cámara. Aguarda un valor entre 0 y 65535, llama al método *picture()* para recuperar la estructura de la imagen de la cámara, llama a *brightness()* para fijar el valor de la sensibilidad definido allí, y finalmente llama al método *set()* para configurar el valor en la capa de *Video4Linux*.

El método *capture()* definido en la línea 117 acepta opcionalmente un valor de brillo antes de capturar el siguiente fotograma del flujo de vídeo. El primer fotograma disponible lleva el número 0, el siguiente estará disponible como 1. Al llamar al método *capture()* de *Video::Capture::V4l* en la línea 128 se hace la petición del fotograma específico del dispositivo de vídeo. La siguiente llamada a *sync()* asegura que la información de la imagen se transfiere al escalar *$frame*.

Algunos tests muestran que el primer fotograma a comprimir puede no ser útil, debido a que el nivel de brillo fijado por la cámara justo antes de capturar el fotograma aún no ha sido aplicado. Para solucionar esto, *capture* siempre captura dos fotogramas y descarta el primero.

Figura 1: Ésta es la cámara usada en los experimentos: la NX Ultra de Creative.

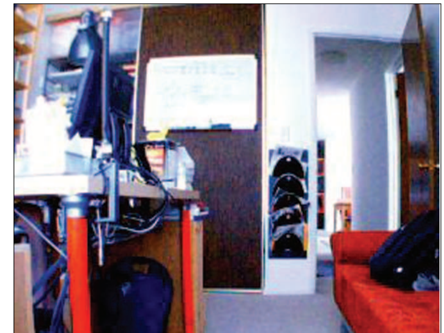

Figura 2: Un pequeño programa en Perl capturó esta imagen a partir del flujo de vídeo.

Cuando el método *sync()* vuelve, la variable *$frame* guarda la información de la imagen en bruto en formato BGR. Cada pixel queda descrito por tres bytes consecutivos que contienen los valores de azul, verde y rojo (entre 0 y 255). Para generar un formato que puedan entender las herramientas de manipulación de imágenes, el comando *reverse* invierte la cadena de bytes, lo que da lugar al formato más común RGB.

Si añadimos ahora una cabecera P6, lo que sucede en la línea 144, y especificamos el ancho y alto de la imagen, el método *read()* del módulo Imager de CPAN puede proporcionarnos una imagen en formato PNM. El truco anterior de invertir el orden de los bytes ha invertido también el orden de los píxeles, dejando la imagen colocada al principio. Pero esto puede solucionarse fácilmente llamando al método *flip*, que usa el parámetro *dir = > "hv"* para girar la imagen 180 grados. Por último, el método *capture()* devuelve la imagen como un objeto de la clase *Imager*, que ya se puede procesar llamando a la función.

Locura a Bajo Nivel

Para ajustar la cámara a la luz ambiente, el método *calibrate* investiga una imagen de prueba, determina su brillo y ajusta la sensibilidad de la cámara. Si la nueva imagen aún no es demasiado buena, el proceso continúa. Pero, ¿qué hacemos si la imagen es demasiado oscura o demasiado clara?

Las Figuras 3 y 4 muestran la distribución de todos los valores RGB de los píxeles de dos imágenes diferentes. El histograma de la Figura 3 está tomado de una imagen fuertemente subexpuesta, razón por la cual tiene algunos valores RGB en los rangos inferiores, y luego cae abruptamente y no da más tonos brillantes. Por contra, la Figura 4 muestra el histograma de una imagen expuesta normalmente, donde podemos ver la distribución simétrica de los valores entre 0 y 255.

Para evaluar el brillo de la imagen de prueba, vamos a usar un algoritmo algo primitivo: sumamos todos los valores y dividimos entre tres el número de píxeles. Si esto nos da en torno a la mitad de 256, entonces el nivel de lectura de color está en mitad de rango, y suponemos la imagen más o menos bien balanceada.

Sin embargo, Perl no está pensado para hacer esto muy rápidamente. Con una imagen de 320 por 240, donde cada pixel contiene un valor para los canales rojo, azul y verde, esto hace 230.400 puntos de información. Hacer el cálculo en este caso lleva bastante tiempo, lo cual puede ser un problema, a menos que eliminemos toda la carga de cómputo extra.

Afortunadamente, podemos extender el módulo de Imager usando C, por lo que podemos volar a través de las estructuras de datos a alta velocidad y devolver el resultado al script Perl. Para ello, necesitamos crear un directorio *Imager-Misc* en la estructura de directorios del módulo Imager de CPAN (lo necesitamos para usar algunas de sus cabeceras en C), cosa que hacemos tecleando:

```
$ cd Imager-0.51
$ h2xs -Axn Imager::Misc
```

En el *Makefile.PL* que aparece en el directorio *Imager-Misc* tenemos que cambiar la línea *INC = > -I.* a *INC = > -I..* para permitir las llamadas a *perl Makefile.PL* y *make* para encontrar los archivos Include de la distribución de Imager. *h2xs* crea también un archivo *Misc.xs* para el

Listado 1: single

```
01 #!/usr/bin/perl
02 ################################
03 use strict;
04 use warnings;
05 use Camcap;
06
07 my $cam = Camcap->new(width
   => 640,
08                        height
   => 480);
09 $cam->cam_bright(42_000);
10 my $img = $cam->capture();
11 $img->write(file =>
   'buero.jpg')
12   or die "Can't write: $!";
```

nuevo y veloz código en C y la capa de conexión (véase el Listado 3).

La línea 18 del Listado 3 muestra el código en C de la función *brightness()*. El vudú de Perl XS necesario para ligar esto al script Perl comienza en la línea 80. El ancho de la imagen que se pasa se da en píxeles con *im->xsize*, la altura con *im->ysize*. Dos bucles *for* iteran a través de los píxeles, y la macro *i_gpix* extrae el valor del color del pixel. Devuelve

Listado 2: Camcap.pm

```
001 ################################
002 package Camcap;
003 ################################
004 use strict;
005 use warnings;
006 use Video::Capture::V4l;
007 use Imager;
008 use Imager::Misc;
009 use Log::Log4perl qw(:easy);
010
011 ################################
012 sub new {
013 ################################
014     my($class, @options) =
    @_;
015
016     my $self = {
017         width   => 320,
018         height  => 240,
019         avg_opt => 128,
020         avg_acc => 20,
021         br_min  => 0,
022         br_max  => 65535,
023         @options,
024     };
025
026     $self->{video} =
027
    Video::Capture::V4l->new() or
028             LOGDIE "Open
    video failed: $!";
029
030     bless $self, $class;
031 }
032
033 ################################
034 sub cam_bright {
035 ################################
036     my($self, $brightness) =
    @_;
037
038     my $pic =
    $self->{video}->picture();
039
    $pic->brightness($brightness)
    ;
040     $pic->set();
041 }
042
043 ################################
044 sub img_avg {
045 ################################
046   my($img) = @_;
047
048   my $br =
    Imager::Misc::brightness($img
    );
049   DEBUG "Brightness: $br";
050   return $br;
051 }
052
053 ################################
054 sub calibrate {
055 ################################
056   my($self) = @_;
057
058   DEBUG "Calibrating";
059
060   return if
061
    img_avg($self->capture($self-
    >{br_min}))
062     > $self->{avg_opt};
063
064   return if
065
    img_avg($self->capture($self-
    >{br_max}))
066     < $self->{avg_opt};
067
068       # Binary search
069   my($low, $high) =
    ($self->{br_min},
070
    $self->{br_max});
071
072   for(my $max = 5;
073       $low <= $high && $max;
074       $max—) {
075     my $try = int( ($low +
    $high) / 2);
076
077     my $i  =
    $self->capture($try);
078     my $br = img_avg($i);
079
080     DEBUG "br=$try got
    avg=$br";
081     return if
    abs($br-$self->{avg_opt}) <=
082
    $self->{avg_acc};
083
084     if($br <
    $self->{avg_opt}) {
085         $low = $try + 1;
086     } else {
087         $high = $try - 1;
088     }
089   }
090   # Nothing found, use last
    setting
091 }
092
093 ################################
094 sub capture {
095 ################################
096     my($self, $br) = @_;
097
098     $self->cam_bright($br) if
    defined $br;
099
100     my $frame;
101     for my $frameno (0, 1) {
102        $frame =
    $self->{video}->capture(
103               $frameno,
    $self->{width},
104
    $self->{height});
105
106
    $self->{video}->sync($frameno
    ) or
107                LOGDIE "Unable
    to sync";
108     }
109
110     my $i = Imager->new();
111     $frame = reverse $frame;
112     $i->read(
113       type => "pnm",
114       data =>
    "P6\n$self->{width} " .
115
    "$self->{height}\n255\n" .
116               $frame
117     );
118     $i->flip(dir => "hv");
119     return $i;
120 }
121
122 1;
```

una estructura *i_color* en la variable *val*. El valor rojo, por ejemplo, de un pixel se puede referenciar entonces como *val.channel[0]*.

Para compilar e instalar el nuevo módulo *Imager::Misc* sólo tenemos que seguir los pasos normales: *perl Makefile.PL*, *make*, *make install*. Con *use Imager::Misc* se carga el módulo. Podemos acceder ahora a la nueva función *Imager::Misc::brightness*, que acepta una imagen Imager y devuelve un valor entero para expresar el brillo de la imagen.

El sencillo algoritmo nos da un valor de 7 para *brightness()* en caso de la imagen oscura de la Figura 3, y un valor de 125 para la imagen con exposición normal mostrada en el histograma de la Figura 4.

Para configurar la cámara para que se adecúe a la luz ambiente, el método *calibrate()* de *Camcap.pm* lanza primero una imagen de prueba, determina el valor de retorno pasado por la veloz función *brightness* y compara este valor de retorno con el valor ideal de 128. Si el valor medido está por debajo, *calibrate()* llama a *cam_bright()* para ajustar la sensibilidad de la cámara. Si el valor medido está bajo el valor ideal, es decir, si la imagen está subexpuesta, *cam_bright()* configura un valor mayor para la siguiente imagen de prueba.

calibrate() comienza disparando dos imágenes con una sensibilidad máxima y mínima de la cámara. Si la imagen es demasiado oscura con exposición máxima (o demasiado clara con la exposición mínima), la función mantiene el valor actual, ya que no hay nada que se pueda hacer para conseguir una mejor imagen.

En los demás casos, *Capcam.pm* inicia una búsqueda binaria para hallar la exposición óptima entre 0 y 65535. Si la imagen es demasiado oscura, el algoritmo continúa su búsqueda en la parte superior del intervalo, y si la imagen es dema-

Figura 3: Los valores altos de brillo no aparecen en este histograma de una foto fuertemente subexpuesta.

Figura 4: El histograma de una imagen con una exposición normal muestra una distribución uniforme de los valores de brillo.

Listado 3: Misc.xs

```
01 #ifdef __cplusplus
02 extern "C" {
03 #endif
04 #include "EXTERN.h"
05 #include "perl.h"
06 #include "XSUB.h"
07 #include "ppport.h"
08 #ifdef __cplusplus
09 }
10 #endif
11
12 #include "imext.h"
13 #include "imperl.h"
14
15 DEFINE_IMAGER_CALLBACKS;
16
17 /* ====================== */
18 int
19 brightness(i_img *im) {
20   int      x, y;
21   i_color val;
22   double   sum;
23   int      br;
24   int      avg;
25
26   for(x = 0; x < im->xsize;
   x++) {
27     for(y = 0; y < im->ysize;
   y++) {
28       i_gpix(im, x, y, &val);
29       br = (val.channel[0] +
val.channel[1]
30         + val.channel[2]) / 3;
31         sum += br;
32     }
33   }
34
35   avg = sum / ((int)
   (im->xsize) *
36                (int)
   (im->ysize));
37   return avg;
38 }
39
40 /* ======================== */
41 int
42 changed(i_img *im1, i_img
   *im2, int diff) {
43   int      x, y, z, chan;
44   i_color val1, val2;
45   int      diffcount = 0;
46
47   for(x = 0; x < im1->xsize;
   x++) {
48     for(y = 0; y < im1->ysize;
   y++) {
49
50         i_gpix(im1, x, y,
   &val1);
51         i_gpix(im2, x, y,
   &val2);
52
53         for(z = 0; z < 3; z++) {
54           if(abs(val1.channel[z]
   -
55
   val2.channel[z]) > diff)
56             diffcount++;
57         }
58     }
59   }
60
61   return diffcount;
62 }
63
64 /* ======================== */
65 MODULE=Imager::Misc
   PACKAGE=Imager::Misc
66
67 PROTOTYPES: ENABLE
68
69 int
70 brightness(im)
71         Imager::ImgRaw im
72
73 int
74 changed(im1, im2, diff)
75         Imager::ImgRaw im1
76         Imager::ImgRaw im2
77         int diff
78
79 BOOT:
80       PERL_INITIALIZE_IMAGER_
   CALLBACKS;
```

siado clara, usa la mitad inferior. Si todo va bien, la cámara debería darnos una imagen con un valor de brillo de 128 +/- 20 (confuso parámetro *avg_acc*). Para evitar demoras, la búsqueda se suspende tras cinco intentos.

Diferencias

Si dejamos la cámara funcionando 24 horas al día, 7 días a la semana, acabaremos con una gigantesca cantidad de imágenes. Pero para propósitos de monitorización, sólo vamos a querer almacenar las imágenes que cambian considerablemente en comparación con la imagen anterior.

Para filtrar el material carente de interés, *Misc.xs* define una función que devuelve el número de valores RGB que ha cambiado entre dos imágenes. Ademas de dos punteros a las estructuras *i_img* (objetos Imager::ImgRaw a nivel Perl), se aguarda un parámetro *diff* para las diferencias entre los valores de los canales. Si el valor de rojo de un pixel de la primera imagen es 15, y el valor rojo para el mismo pixel en la segunda imagen es 30, el contador *diffcount* se incrementa, en caso de que se fije un valor de 15 o más para el parámetro *diff*. Esto ayuda a compensar la desviación estadística que ocurre debido al ruido del chip CCD o a cambios naturales en la luz ambiente.

El script *tracker* del Listado 4 ejecuta un bucle infinito, disparando una imagen tras otra, pero sólo almacenándolas en caso de que *Imager::Misc::changed()* señale que hubo un cambio importante, debido a que más de 2000 píxeles excedieron el umbral. Si el contador es inferior, se sobrescribe la última imagen en la caché para mantener la cuenta de los cambios graduales.

Las imágenes guardadas terminan en un disco caché *Cache::File-Cache* y

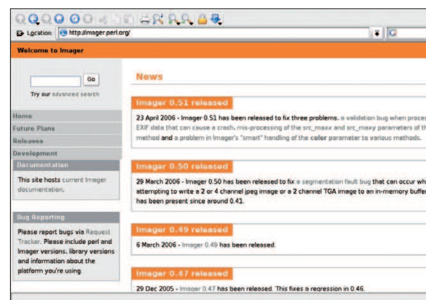

Figura 5: Podemos encontrar información de las versiones de Imager, así como reportes de bugs y una wiki, en la página Web de imager en perl.org.

se borran automáticamente tras 48 horas. Con las imágenes se guarda una clave con la fecha (como *2006/03/28-11:21:22*). Para recuperar la última imagen de la caché, *tracker* simplemente llama a la caché *get_keys()*, que devuelve todas las claves existentes. La función *maxstr* que proporciona el módulo *List::Util* coge

Listado 4: tracker

```
01 #!/usr/bin/perl
02 ##################################
03 use strict;
04 use warnings;
05 use Camcap;
06 use Imager::Misc;
07 use Log::Log4perl qw(:easy);
08 use Cache::SharedMemoryCache;
09 use Time::Piece;
10 use List::Util qw(maxstr);
11
12 my $c =
   Cache::SharedMemoryCache->new(
   {
13     namespace          =>
   “tracker”,
14     default_expires_in =>
   48*3600 });
15
16
   Log::Log4perl->easy_init($DEBU
   G);
17
18 my $cam = Camcap->new();
19
20 while(1) {
21   my $lkey = maxstr grep /\d/,
22
   $c->get_keys();
23
24   if(! $c->get(“calibrated”))
   {
25     $cam->calibrate();
26     $c->set(“calibrated”, 1,
   300);
27     my $img = $cam->capture();
28     saveimg($img, $c, $lkey);
29     next;
30   }
31
32   my $img = $cam->capture();
33
34   if($lkey) {
35     my $limg = Imager->new();
36     $limg->read(type =>
   “jpeg”,
37                 data =>
   $c->get($lkey));
38     my $dpix =
   Imager::Misc::changed($limg,
39
   $img, 80);
40     DEBUG “$dpix pixels
   changed”;
41     if($dpix > 2000) {
42       saveimg($img, $c);
43       next;
44     } else {
45         # minor change,
46         # refresh reference
47       saveimg($img, $c,
   $lkey);
48     }
49   } else {
50       # save first img
51     saveimg($img, $c);
52   }
53
54   sleep(1);
55 }
56
57 ##################################
58 sub saveimg {
59 ##################################
60     my($img, $cache, $date) =
   @_;
61
62     if(! $date) {
63         $date = localtime()->
64
   strftime(“%Y/%m/%d-%H:%M:%S”);
65     }
66
67     DEBUG “Saving image
   $date”;
68     $img->write(type =>
   “jpeg”,
69                 data => \my
   $val) or die;
70     $cache->set($date, $val);
71 }
```

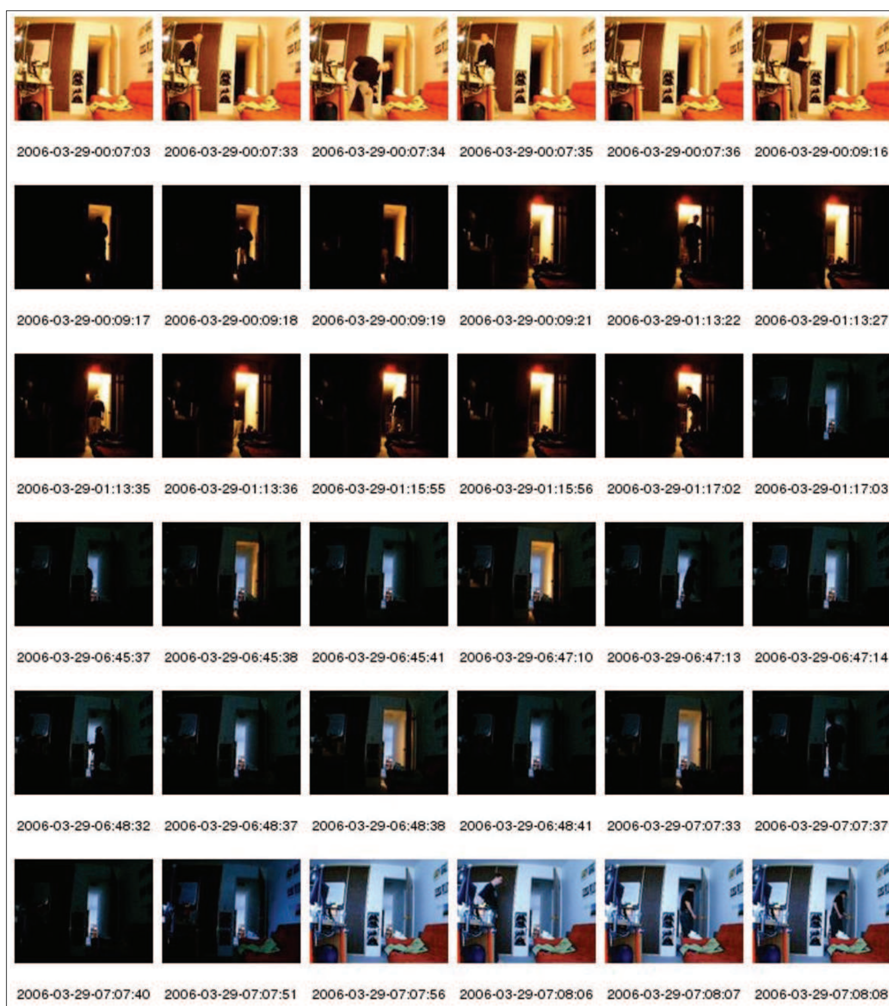

Figura 6: Las imágenes que el tracker ha conservado, vistas tras el script cacheprint.

la fecha más reciente. Y la función *get()* de la caché aguarda la clave como argumento y devuelve la imagen coincidente.

Queremos que la cámara se recalibre cada cinco minutos para tener en cuenta los cambios en la luz ambiente. Para conseguirlo, se guarda en la caché una pseudo entrada denominada *calibrated* que se borra cada 300 segundos. Si el *tracker* falla al tratar de encontrar la entrada, recalibra y escribe el valor de nuevo. El script *cacheprint* coge las imágenes de la caché creada por el tracker. Compone con ellas imágenes JPEG con la información de la caché y las guarda en un nuevo directorio temporal. Entonces *cacheprint* llama al programa *montage* del toolbox *Image-Magick* para crear montajes que parezcan secuencias de imágenes. El visor *xv* muestra las miniaturas y hace coincidir los valores de las fechas (véase Figura 6).

La Figura 6 muestra el resultado de medio día de imágenes que tenía suficientes cambios como para que *tracker* las almacenase. Las imágenes muestran el surrealista mundo de los estudios Perlmeister. A las 0:09, la luz del estudio se apaga, y a eso de la 1:17, todo parece tranquilo en la parte visible de mi apartamento. Los cambios graduales entre la 1:17 y las 6:45 se sobreescriben con la marca de tiempo 01:17:03, hasta que finalmente a las 06:45:37 una persona que surge de entre la oscuridad hace cambiar un número suficiente de píxeles como para que *tracker* registre los movimientos. A las *07:07:56* alguien abre las cortinas y entra por fin la luz de la mañana. ■

RECURSOS

[1] Listados de este artículo: *http://www.linux-magazine.es/Magazine/Downloads/21*

[2] Definición de brillo: *http://en.wikipedia.org/wiki/Brightness*

[3] Marc Lehmann, “Capturing Video in Real Time”, The Perl Journal, 2005/02

[4] Página Web del módulo Imager de Perl: *http://imager.perl.org*

Listado 5: cacheprint

```
01 #!/usr/bin/perl -w
02 use strict;
03 use Imager;
04 use Cache::SharedMemoryCache;
05 use Time::Piece;
06 use List::Util qw(maxstr);
07 use Sysadm::Install qw(rmf mkd
   cd);
08 use File::Temp qw(tempdir);
09
10 my $dir = tempdir(CLEANUP =>
   1);
11
12 my $c =
   Cache::SharedMemoryCache->new(
   {
13     namespace => “tracker”,
14 });
15
16 for my $date (sort
   $c->get_keys()) {
17
18   next unless $date =~ /\d/;
19   my $val = $c->get($date);
20   my $img = Imager->new();
21   $img->read(type => “jpeg”,
22              data => $val);
23   $date =~ s#/#-#g;
24   $img->write(file =>
   “$dir/$date.jpg”) or
25       die “Can’t write $!”;
26 }
27
28 cd $dir;
29 my $str = “”;
30 for (<*.jpg>) {
31     (my $date = $_) =~
   s/\.jpg//g;
32     $str .= “-label $date $_
   “;
33 }
34 `montage -tile 6x6 $str
   sequence.jpg`;
35 `xv $_` for <sequence*>;
```