

SDLBlock Writeup

Tyler Nielsen Ryan Lewis Jeremy Garcia

May 5, 2003

Contents

| | | |
|----------|--------------------------------|----------|
| 1 | Introduction | 2 |
| 1.1 | Scope | 2 |
| 1.2 | Product Idea | 2 |
| 2 | Implementation | 2 |
| 2.1 | Web-cam | 2 |
| 2.2 | Filters | 2 |
| 2.2.1 | Image Filters | 2 |
| 2.2.2 | Block Filters | 3 |
| 2.3 | OCR | 3 |
| 2.4 | Speech Synthesis | 3 |
| 2.5 | Problems | 3 |
| 3 | Educational Value | 3 |
| 3.1 | Child Learning | 4 |
| 3.2 | Open Source Software | 4 |
| 4 | Further Work | 4 |
| 4.1 | Speed and Accuracy | 4 |
| 4.2 | Non-intrusive | 4 |
| 5 | Conclusion | 5 |
| A | Source Code | 6 |

1 Introduction

Here we give a quick introduction to the project we worked on.

1.1 Scope

In this paper we will be going over the second project for the class Things That Think. This project involved adding computation to an existing construction kit. All source code is provided in `Appendix:fefsec:code`.

1.2 Product Idea

In adding computation to a construction kit, we decided that we would work on word recognition on an off-the-shelf set of alphabet blocks. We decided that the best approach was to use a web-cam to get an image, then by use optical character recognition (OCR) software to and display each word. One main consideration when coming up with the product idea was whether to build a small bit of computation into each piece, or build the computation into an intelligent observer. We decided that building an intelligent observer is the more interesting problem. The intelligent observer idea can be used in many cases, and once written well, will work within many preexisting environments.

2 Implementation

In this section we will retrace the steps we went through to get the project implemented.

2.1 Web-cam

The first thing we wanted to make sure of is that we could get control of a web-cam, and get images into our program. After purchasing a web-cam, we proceed with our operating system of choice (Linux). We found this was actually quite easy since each device has a file. We just open the file and read the image in.

2.2 Filters

This was the most complicated and difficult section. We made a number of filters to take the image in and produce an image acceptable for the OCR stage. We have broken this section up into two parts. One for filters we apply to the whole image, and one section for the filters we use on each individual block.

2.2.1 Image Filters

There are three stages in our image filters. The first step is to get a reference for what color is white (by using the top and bottom ten lines) then from there to remove all the gray scale colors from the image. In the second stage, we find

where the blocks actually are. This is fairly easy after all the background has been removed. The last step is the most complicated, this step we try to find the boundaries between each block. To do this we compute the standard deviation of a range of blocks, then find the local minimums. After the last stage, we are left with a series of blocks that we pass onto the block filters.

2.2.2 Block Filters

We originally started out with several different filters here, but now we are only using one. We use a filter to determine what color we think a block is. We then use this information to improve the OCR step.

2.3 OCR

In our initial design we were planning on using a packaged OCR program. The program we were using required us to turn each block into a black and white image. After writing several Block Filters (Section 2.2.2) we found that the OCR program was not accurate enough. To get around this we took pictures of all the blocks for a test case. We now do a correlation with each block, and the block that scores the highest is the one it is identified as. We found this to be much more accurate. We use the color of the block to reduce the number of correlation tests we have to perform.

2.4 Speech Synthesis

To add a little bit to the project we decided that after a valid word has been detected (based on a dictionary) we would have the computer pronounce the word. This turned out to be a simple addition. We found a free text to speech program (called “festival”) that lets us send the word we want to say.

2.5 Problems

We did not encounter many significant problems while completing this project. We did find that the OCR software was not good enough (Section 2.3), and this caused some redesign. In addition, we found that small errors in one of the filters can through off the entire process. If a blocks color is identified wrong, or the splits between a block are identified wrong, this guarantees a mistake in the OCR phase. Further more, there are several accuracy knobs that have been turned down in the interest of speed. Our program is not terribly fast, and without turning down the accuracy, we run too slowly.

3 Educational Value

Here we talk about the educational value of what we have done.

3.1 Child Learning

In its current form, there is little to no benefit for a child playing with alphabet blocks. The software is slow, the filters require special conditions, and there is not very much learning built in. However we think there is a lot of potential for this type of work. Ideally we see that a way this may work is by recognizing what a child is doing, and say appropriate things at appropriate times. This is an open statement, because there is lots of work to be done in this area. You could envision the environment would model what a parent would say in similar circumstances. We may see the computer say the letter when a child picks up a block, help a child sound out a word they spelled, or help them spell it correctly. Ultimately there are many ways this could progress. This subject involves educational psychology as much as computer science.

3.2 Open Source Software

The form of learning that is available currently is through our source code. We will be providing this with the program under the GNU Public License (GPL). This will allow anyone interested in what we did, and how we did it, access to the source. They can then take this, examine it, and hopefully improve upon it.

4 Further Work

Here we take a quick look at where this project can go from here.

4.1 Speed and Accuracy

As we talked about in Section 2.3, there are knobs we used to turn down the accuracy to get better speed. These can be looked at and a way to improve speed and accuracy could be found. Making the filters more robust would help. We were able to get a proof of concept complete, but there are several ways to improve in this area.

4.2 Non-intrusive

To see this in its final form (Section 3.1) it must be much less intrusive. This is quite difficult, because now block letters can be oriented in different ways, camera angle can be changing as you are using it, and you no longer have a nice white background. These things all can be added, it just adds to the complexity of the software. However, this has to be looked and solved if this was to ever be used in an educational capacity.

5 Conclusion

We all feel happy with where we got for this project. We completed a proof of concept, and learned a lot while doing it. We all feel that the intelligent observer is the right way to go and hopefully advances in Artificial Intelligence (specifically pattern recognition) will make this more of a reality for everyday use.

A Source Code

This Appendix contains all the source code for SDLBlocks. The code was written in C++ and Perl. As stated above, all code is licensed under the GNU General Public License.

```
RELEASETARGET = SDLBlocks
DEBUGTARGET = SDLBlocks_debug
SOURCES = $(wildcard *.cpp)
RELEASEDIR = release/
DEBUGDIR = debug/
SDLFLAGS = $(shell sdl-config --cflags)
SDLLIBS = $(shell sdl-config --libs)
COMMONLIBS = -L/usr/X11R6/lib -lX11 -lXext -ljpeg -lpng -ltiff -lz -lgif -lm -lImlib -l
pnm
COMMONCFLAGS = -I/usr/X11R6/include -Wall -W -Werror

DEP = g++ $(INCFLAGS) -MM

ifeq ($(MAKECMDGOALS),debug)
BUILDDIR = $(DEBUGDIR)
CFLAGS = -D_DEBUG -ggdb $(SDLFLAGS) $(COMMONCFLAGS)
LIBS = $(SDLLIBS) $(COMMONLIBS)
TARGET = $(DEBUGTARGET)
else
BUILDDIR = $(RELEASEDIR)
CFLAGS = -D_NDEBUG $(SDLFLAGS) $(COMMONCFLAGS)
LIBS = $(SDLLIBS) $(COMMONLIBS)
TARGET = $(RELEASETARGET)
endif

OBJECTS = $(SOURCES:%.cpp=$(BUILDDIR)%.o)
DEPENDS = $(SOURCES:%.cpp=$(BUILDDIR)%.d)
VPATH = $(BUILDDIR)

release: $(TARGET)

debug: $(TARGET)

$(TARGET): $(DEPENDS) $(OBJECTS)
    g++ $(LIBS) $(OBJECTS) -o $(TARGET)

$(OBJECTS): $(BUILDDIR)%.o : %.cpp
    g++ $(CFLAGS) -c $< -o $@

$(DEPENDS): $(BUILDDIR)%.d : %.cpp
    @mkdir -p $(BUILDDIR)
    @echo "Updating dependancies for $< ..."
    @$ (DEP) $(CFLAGS) $< | sed 's,\($*\)\.o[ :]*,$(BUILDDIR)\1.o $@ : ,g' > $@

tidy: force
    @mkdir -p $(DEBUGDIR)
    @mkdir -p $(RELEASEDIR)
    @rm -f *~
    @cd $(DEBUGDIR);rm -f *.bak *.out *.a *~ core
    @cd $(RELEASEDIR);rm -f *.bak *.out *.a *~ core

clean: force tidy
    @cd $(RELEASEDIR);rm -f *.o *.a
    @rm -f $(RELEASETARGET)
    @cd $(DEBUGDIR);rm -f *.o *.a
    @rm -f $(DEBUGTARGET)

cleanall: force clean
    @rm -rf $(RELEASEDIR)
    @rm -rf $(DEBUGDIR)

force:;

ifndef $(MAKECMDGOALS),cleanall)
```

```
ifneq ($(MAKECMDGOALS),clean)
ifneq ($(MAKECMDGOALS),tidy)
include $(DEPENDS)
endif
endif
endif
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#ifdef __CAMERA_H__
#define __CAMERA_H__

#include <string>
#include <linux/videodev.h>
#include "image.h"

using namespace std;

class Camera : public Image
{
private:
    int dev;
    video_capability vid_caps;
    video_window vid_win;
    video_picture vid_pic;
    int greyscale;
    unsigned char *picbuff;
    /*
        int draw;
        int frozen;
        int update_camera;
        int capture;
        int saving;
        int savetype;
        int docked;
        int dump;
        int speed_fastest;
        int on_timer;
        int timeout;
        int swapcolors;
        int autobright;
        quint timeoutid;
        struct video_clip vid_clips[32];
        char devname[256];
        char savefile[256];
        char savefileclean[256];
        GtkWidget *drawing_area;
        GtkWidget *controlcontainer, *controltop, *controlwindow;
        GtkWidget *statusbar;
        GtkWidget *currentsavepage;
        pthread_mutex_t iscam_mutex, pref_mutex, freeze_mutex;
        GdkPixmap *pixmap;
        int fps_avg;
        int fps_current;
        struct Page_Ppm page_ppm;
        struct Page_Jpeg page_jpeg;
    */
};
```

```
    struct Page_Gif page_gif;
    struct Page_Png page_png;
    struct Save_Struct save_struct;
    struct Timer_Struct timer_struct;
    struct Controls controls;
    struct Pref_Dialog pref_dialog;
*/
public:
    Camera(string dev);
    ~Camera();

    void PrintInfo(void);
    bool Grab(void);

    unsigned char *GetPixels(void);
    int GetWidth(void);
    int GetHeight(void);

/*
    void display(struct Camera*);
    void delete_event(GtkWidget*, struct Camera *camera);
    void open_cam(struct Camera*);
    void close_cam(struct Camera*, int force);
    void print_cam_info(struct Camera*);
    void get_cam_info(struct Camera*);
    void dump_locks( struct Camera *camera );
    int dump_pict( struct Camera *camera, char *tofile, int brightness, int contrast, i
nt whiteness );
    void print_usage();
    int next_frame( struct Camera *camera );
*/
};

#endif
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#ifndef __GRAPHICS_H__
#define __GRAPHICS_H__

#include "pixel.h"

#ifdef _DEBUG
#define DEFAULTFLAGS (SDL_SWSURFACE)
#else
#define DEFAULTFLAGS (SDL_HWSURFACE|SDL_FULLSCREEN)
#endif

class Graphics
{
private:
    int Xres;
    int Yres;
    int Bpp;
    int Flags;
    SDL_Surface *Screen;
    SDL_Surface *Back;

public:
    Graphics(int X=640, int Y=480, int bpp=16, int flags=DEFAULTFLAGS);
    ~Graphics();

    bool Init(void);
    void Draw(Surface &image);
};

#endif
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#ifdef __IMAGE_H__
#define __IMAGE_H__

#include <string>
#include <linux/videodev.h>

using namespace std;

class Image
{
private:

public:

    virtual ~Image() { }

    virtual void PrintInfo(void) = 0;
    virtual bool Grab(void) = 0;

    virtual unsigned char *GetPixels(void) = 0;
    virtual int GetWidth(void) = 0;
    virtual int GetHeight(void) = 0;

};

#endif
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#ifdef __JPEG_H__
#define __JPEG_H__

#include <string>
#include <linux/videodev.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/extensions/shape.h>
#include <Imlib.h>
#include "image.h"

using namespace std;

class Jpeg : public Image
{
private:
    Display *disp;
    ImlibData *id;
    ImlibImage *im;
    int width, height;

public:
    Jpeg(string dev);
    ~Jpeg();

    void PrintInfo(void);
    bool Grab(void);

    unsigned char *GetPixels(void);
    int GetWidth(void);
    int GetHeight(void);
};

#endif
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#ifdef __LETTERS_H__
#define __LETTERS_H__

const char *Red      = "AEIMOSW";
const char *Green    = "BFJNRVZ";
const char *Blue     = "DHLPTX";
const char *Yellow   = "CGKQUY";

const char Number[] = {5,4,2,2,5,3,3,2,4,2,2,2,2,5,4,2,2,5,3,3,2,4,2,4,2,2};
const char Colors[] = {0,1,3,2,0,1,3,2,0,1,3,2,0,1,0,2,3,1,0,2,3,1,0,2,3,1};

#endif
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#ifdef _NRC_H_
#define _NRC_H_

const double TINY = 1.0e-20; // Used for some NRC routines

// --NRC--
void moment(double data[], int n, double *ave, double *adev, double *sdev,
            double *var, double *skew, double *curt);
void pearson(double x[], double y[], unsigned long n, double *r, double *prob,
            double *z);
double erfcc(double x);
//void hpsel(unsigned long m, unsigned long n, Pair arr[], Pair heap[]);

#endif
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#ifdef _PAM_H_
#define _PAM_H_

#include "pixel.h"

void writeImage(Pixel *image, int xres, int yres, char *f);
void readImage(Pixel **image, int &xres, int &yres, const char *file);

#endif
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#ifdef __PIXEL_H__
#define __PIXEL_H__

#include <vector>
#include <set>
#include <string>

using namespace std;

struct Pixel {
    unsigned char Red;
    unsigned char Green;
    unsigned char Blue;

    Pixel(unsigned char r = 0, unsigned char g = 0, unsigned char b = 0) {
        Red = r; Green = g; Blue = b;
    }
    Pixel(const Pixel &p) {
        Red = p.Red; Green = p.Green; Blue = p.Blue;
    }

    friend bool operator ==(const Pixel &left, const Pixel &right);
    friend bool operator !=(const Pixel &left, const Pixel &right);

    bool isClose(const Pixel &p) {
        if(abs(Red-p.Red)+abs(Green-p.Green)+abs(Blue-p.Blue)<50)
            return true;
        return false;
    }

    friend ostream& operator << (ostream &left, const Pixel &right);
};

struct Point {
    int x;
    int y;

    Point(int X, int Y) : x(X), y(Y) { }
    Point(const Point &p) {
        x=p.x;y=p.y;
    }

    friend bool operator ==(const Point &left, const Point &right);
    friend bool operator < (const Point &left, const Point &right);
};

inline bool IsInSet(const set<Point> &s, const Point &val) {
```

```
    if(s.find(val)==s.end())
        return false;
    return true;
}

enum BlockColor {
    RED,
    GREEN,
    BLUE,
    YELLOW,
    WHITE
};

const Pixel PixelColor[] = {
    Pixel(255,0,0),
    Pixel(0,255,0),
    Pixel(0,0,255),
    Pixel(255,255,0),
    Pixel(255,255,255),
};

class Block {
private:
    Pixel *data;
    int xres;
    int yres;
    int blockHeight;
    int blockWidth;
    int borderWidth;
    BlockColor color;

    char letter;
    string filename;

public:
    Block(Pixel *image, int xSize, int ySize,
          int xTopLeft, int yTopLeft, int imageXsize, int imageYsize);
    Block(const Block &b);
    Block(string Filename, char Letter, char Color);

    Block &operator =(const Block &right);
    void SaveBlock(char *f);
    void LoadBlock(const char *f);

    const string &getFilename(void){ return filename; }

    void drawBlock(Pixel *buffer, int xPos, int yPos,
                  int xSize, int ySize);
    void BlackAndWhite(void);
    Pixel &getpixel(Pixel *a, int x, int y) {
        return a[y*xres + x];
    }
    void DetermineColor(void);
    void MakeGrayscale(void);
    ~Block();

    friend class Surface;

    friend double BlockCorrelation(Block &a, Block &b, size_t n);
};

class Surface {
private:
    static Pixel *oldimage;
```

```
Pixel *original;

Pixel *data;

static Pixel white;
static const Pixel refwhite;

static vector<Block> testcase;

int xres;
int yres;

int signaltop;
int signalbottom;
int signalleft;
int signalright;

int numBlocks;
int BlockHeight;

vector<int> BlockDiv;
vector<Block> Blocks;

Pixel &getpixel(Pixel *a, int x, int y) {
    return a[y*xres+x];
}

Pixel &getpixel(Pixel *a, const Point &p) {
    return a[p.y*xres+p.x];
}

void loadtestcase(void);

string word;

public:
Surface(unsigned char *RGB, int x, int y);
~Surface();

const string & getWord(void) { return word; }

int SaveBlocks(void);

//filters
int ColorAmplify(void);
int Gray2Black(void);
int IsolateSignal(void);
int IdentifyBlocks(void);
int BlackAndWhite(void);
int DetermineColors(void);
int DrawBlocks(void);
int Correlate(void);

friend class Graphics;
friend class Block;
};

#endif
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#include "pixel.h"
#include "nrc.h"
#include "pam.h"
#include <iostream>

using namespace std;

Block::Block(Pixel *image, int xSize, int ySize,
             int xTopLeft, int yTopLeft, int imageXsize, int imageYsize) {
(void)imageYsize; //get rid of warning
    data = new Pixel[xSize*ySize];
    for (int y = yTopLeft; y < yTopLeft + ySize; ++y)
        for (int x = xTopLeft; x < xTopLeft + xSize; ++x)
            data[(y - yTopLeft)*xSize + (x - xTopLeft)] =
                image[y*imageXsize + x];
    xres = xSize;
    yres = ySize;
    blockHeight = ySize;
    blockWidth = xSize;
    borderWidth = blockHeight/9;
    color = WHITE;
    letter = ' ';
    filename = "";
}

Block::Block(const Block &b)
{
    xres = blockWidth = b.xres;
    yres = blockHeight = b.yres;
    borderWidth = b.borderWidth;
    data = new Pixel[xres*yres];
    memcpy(data, b.data, xres*yres*sizeof(Pixel));
    color = b.color;
    letter = b.letter;
    filename = b.filename;
}

Block::Block(string Filename, char Letter, char Color)
{
    data = NULL;
    xres = yres = blockWidth = blockHeight = borderWidth = 0;
    filename = Filename;
    letter = Letter;
    color = (BlockColor)Color;

    LoadBlock(filename.c_str());
}
}
```

```
Block &Block::operator =(const Block &b)
{
    delete [] data;
    xres = blockWidth = b.xres;
    yres = blockHeight = b.yres;
    color = b.color;
    letter = b.letter;
    filename = b.filename;
    borderWidth = b.borderWidth;
    data = new Pixel[xres*yres];
    memcpy(data, b.data, xres*yres*sizeof(Pixel));
    return *this;
}

void Block::drawBlock(Pixel *buffer, int xPos, int yPos,
                    int xSize, int ySize) {
    (void)ySize; //get rid of warning
    for (int y = 0; y < yres; ++y)
        for (int x = 0; x < xres; ++x)
            buffer[(y + yPos)*xSize + x + xPos] = data[y*xres + x];
}

Block::~~Block() {
    delete [] data;
}

void Block::DetermineColor(void)
{
    // Look in the middle of the top and bottom for a patch of color. Then
    // try to figure out what that color is.

    // Red1, Red2, Green, Blue, Yellow, White

    double redfact, greenfact, bluefact;
    redfact = double(Surface::refwhite.Red)/Surface::white.Red;
    greenfact = double(Surface::refwhite.Green)/Surface::white.Green;
    bluefact = double(Surface::refwhite.Blue)/Surface::white.Blue;

    Pixel colors[] = {Pixel(155, 51, 94),
                      Pixel(225, 50, 100),
                      Pixel(12, 82, 76),
                      Pixel(8, 49, 77),
                      Pixel(214, 180, 135),
                      Pixel(209, 184, 200)};
    BlockColor bcolors[] = {
        RED,
        RED,
        GREEN,
        BLUE,
        YELLOW,
        WHITE,
    };
    const int numcolors = 6;
    int matches[numcolors];
    int counts[numcolors];
    int color;
    int min;
    int idx;
    for(idx=0;idx<numcolors;idx++) {
        counts[idx]=0;
        colors[idx].Red = int(colors[idx].Red/redfact);
        colors[idx].Green = int(colors[idx].Green/greenfact);
```

```
    colors[idx].Blue = int(colors[idx].Blue/bluefact);
}

for (int y = 0; y < blockHeight; ++y) {
    for (int x = 0; x < blockWidth; ++x) {
        min = 0;
        for (int i = 0; i < numcolors; ++i) {
            matches[i] =
                (colors[i].Red - getpixel(data, x, y).Red) *
                (colors[i].Red - getpixel(data, x, y).Red)
                +
                (colors[i].Green - getpixel(data, x, y).Green) *
                (colors[i].Green - getpixel(data, x, y).Green)
                +
                (colors[i].Blue - getpixel(data, x, y).Blue) *
                (colors[i].Blue - getpixel(data, x, y).Blue);

            // Remember the best match
            if (matches[i] < matches[min]) min = i;
        }
        // getpixel(data, x, y) = colors[min];
        ++counts[min];
    }
}

color = 0;
for (int i = 1; i < numcolors-1; ++i)
    if (counts[i] > counts[color])
        color = i;

/*
for (int y = 0; y < blockHeight; ++y) {
    for (int x = 0; x < blockWidth; ++x) {
        if (getpixel(data, x, y) != colors[color])
            getpixel(data, x, y) = Pixel(0, 0, 0);
    }
}
*/

this->color = bcolors[color];
}

void Block::MakeGrayscale(void)
{
    for (int y = 0; y < yres; ++y)
        for (int x = 0; x < xres; ++x)
            if (getpixel(data, x, y) == Pixel(0, 0, 0))
                getpixel(data, x, y) = Pixel(255, 255, 255);
            else
                getpixel(data, x, y) = Pixel(0, 0, 0);
}

void Block::BlackAndWhite(void)
{
    double Rstd,Gstd,Bstd;
    double Ravg,Gavg,Bavg;
    double Rvar,Gvar,Bvar;
    double dummy;
    int x, y;
    double *channel;

    channel = new double[blockHeight*blockWidth];

    // Compute standard deviation of each color over entire block
```

```
for (int y = 0; y < blockHeight; ++y)
    for (int x = 0; x < blockWidth; ++x)
        channel[y*blockWidth + x] = getpixel(data, x, y).Red;
moment(channel, blockHeight*blockWidth, &Ravg, &dummy, &Rstd,
        &Rvar, &dummy, &dummy);
for (int y = 0; y < blockHeight; ++y)
    for (int x = 0; x < blockWidth; ++x)
        channel[y*blockWidth + x] = getpixel(data, x, y).Green;
moment(channel, blockHeight*blockWidth, &Gavg, &dummy, &Gstd,
        &Gvar, &dummy, &dummy);
for (int y = 0; y < blockHeight; ++y)
    for (int x = 0; x < blockWidth; ++x)
        channel[y*blockWidth + x] = getpixel(data, x, y).Blue;
moment(channel, blockHeight*blockWidth, &Bavg, &dummy, &Bstd,
        &Bvar, &dummy, &dummy);

cout << Ravg << ' ' << Gavg << ' ' << Bavg << endl;

for (x = 0; x < blockWidth; x++) {
    for (y = 0; y < blockHeight; y++) {
        Pixel &current = getpixel(data, x, y);
        if (current == Pixel(0,0,0))
            current = Pixel(255,255,255);
        else if (current.Red + current.Green + current.Blue >
                 (Ravg + Gavg + Bavg))
            current = Pixel(255,255,255);
        else
            current = Pixel(0,0,0);
    }
}
delete [] channel;
}

void Block::SaveBlock(char *f)
{
    // MakeGrayscale();
    writeImage(data, xres, yres, f);
}

void Block::LoadBlock(const char *f)
{
    delete [] data;
    data = NULL;
    xres = yres = 0;
    readImage(&data, xres, yres, f);
    blockWidth = xres;
    blockHeight = yres;
}

double BlockCorrelation(Block &a, Block &b, size_t n)
{
    Block small = b;
    Block large = a;

    /*
    if(a.xres<=b.xres && a.yres<=b.yres) {
        small = a;
        large = b;
    }
    else if(b.xres<=a.xres && b.yres<=a.yres) {
        small = b;
        large = a;
    }
    */
}
```

```
else {
    // Blocks bad size
    return -1000;
}
*/

int x,y, x1,y1;
vector<double> all;

double *smalld;
double *larged;
double r, prob, z;

smalld = new double[small.xres*small.yres*3];
larged = new double[small.xres*small.yres*3];

for(x=0;x<small.xres;x++) {
    for(y=0;y<small.yres;y++) {
        smalld[(y*small.xres+x)*3+0] = (double)small.data[y*small.xres+x].Red;
        smalld[(y*small.xres+x)*3+1] = (double)small.data[y*small.xres+x].Green;
        smalld[(y*small.xres+x)*3+2] = (double)small.data[y*small.xres+x].Blue;
    }
}

int xmin = 0;
int xmax = large.xres-small.xres;
int ymin = 0;
int ymax = large.yres-small.yres;

while(xmax-xmin>=10) {
    xmax--; xmin++;
}

while(ymax-ymin>=10) {
    ymax--; ymin++;
}

for(x=xmin;x<xmax;x++) {
    for(y=ymin;y<ymax;y++) {
        for(x1=0;x1<small.xres;x1++) {
            for(y1=0;y1<small.yres;y1++) {
                larged[(y1*small.xres+x1)*3+0] =
                    (double)large.data[(y+y1)*large.xres+x+x1].Red;
                larged[(y1*small.xres+x1)*3+1] =
                    (double)large.data[(y+y1)*large.xres+x+x1].Green;
                larged[(y1*small.xres+x1)*3+2] =
                    (double)large.data[(y+y1)*large.xres+x+x1].Blue;
            }
        }
        pearsn(smalld, larged, small.xres*small.yres*3, &r, &prob, &z);
        all.push_back(r);
    }
}

sort(all.begin(), all.end());

if(all.empty())
    return -1000;

while(all.size(>n) {
    all.erase(all.begin());
}

double total = 0;
```

```
vector<double>::iterator alli;

for(alli=all.begin(); alli!=all.end(); alli++) {
    total+=*alli;
}

total = total/all.size()*n;

delete [] smalld;
delete [] larged;

return total;
}
```

```
/*
  Copyright (C) 2003 Tyler Nielsen

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

#include <SDL.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include "camera.h"

#if 0
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <sys/mman.h>
#include <pthread.h>
#include <semaphore.h>
#include <gtk/gtk.h>
#include <linux/types.h>
#include <signal.h>
#include <png.h>
#include "frontend.h"
#include "save.h"
#include "filters.h"

char version[] = VERSION;

#endif

Camera::Camera(string device)
{
  dev = open(device.c_str(), O_RDWR);
  if (dev < 0) {
    perror(device.c_str());
  }

  int i;
  struct video_clip vid_clips[32];

  ioctl(dev, VIDIOCGCAP, &vid_caps);
  ioctl(dev, VIDIOCGWIN, &vid_win);
  ioctl(dev, VIDIOCGPICT, &vid_pic);

  for (i = 0; i < 32; i++) {
    vid_clips[i].x      = 0;
    vid_clips[i].y      = 0;
  }
}
```

```
    vid_clips[i].width  = 0;
    vid_clips[i].height = 0;
}
vid_win.clips = vid_clips;
vid_win.clipcount = 0;

if (vid_caps.type & VID_TYPE_MONOCHROME) {
    greyscale = 1;
    picbuff = new unsigned char[vid_caps.maxwidth*vid_caps.maxheight];
}
else {
    greyscale = 0;
    picbuff = new unsigned char[vid_caps.maxwidth*vid_caps.maxheight*3];
}
}

Camera::~Camera()
{
    if(dev > 0){
        close(dev);
        dev = 0;
    }
    delete [] picbuff;
}

void Camera::PrintInfo(void)
{
    if(dev<=0)
        return;
    printf("Name: %s\n", vid_caps.name);
    printf("Type: %i\n", vid_caps.type);
    if (vid_caps.type & VID_TYPE_CAPTURE) {
        printf("\tCan capture\n");
    }
    if (vid_caps.type & VID_TYPE_TUNER) {
        printf("\tCan tune\n");
    }
    if (vid_caps.type & VID_TYPE_TELETEXT) {
        printf("\tDoes teletext\n");
    }
    if (vid_caps.type & VID_TYPE_OVERLAY) {
        printf("\tOverlay onto frame buffer\n");
    }
    if (vid_caps.type & VID_TYPE_CHROMAKEY) {
        printf("\tOverlay by chromakey\n");
    }
    if (vid_caps.type & VID_TYPE_CLIPPING) {
        printf("\tCan clip\n");
    }
    if (vid_caps.type & VID_TYPE_FRAMERAM) {
        printf("\tUses the frame buffer memory\n");
    }
    if (vid_caps.type & VID_TYPE_SCALES) {
        printf("\tScalable\n");
    }
    if (vid_caps.type & VID_TYPE_MONOCHROME) {
        printf("\tMonochrome only\n");
    }
    if (vid_caps.type & VID_TYPE_SUBCAPTURE) {
        printf("\tCan capture subareas of the image\n");
    }
}

printf("Channels: %i\n", vid_caps.channels);
printf("Audios: %i\n", vid_caps.audios);
```

```
printf("Maxwidth: %i\n", vid_caps.maxwidth);
printf("Maxheight: %i\n", vid_caps.maxheight);
printf("Minwidth: %i\n", vid_caps.minwidth);
printf("Minheight: %i\n", vid_caps.minheight);

printf("-----\n");

printf("X: %i\n", vid_win.x);
printf("Y: %i\n", vid_win.y);
printf("Width: %i\n", vid_win.width);
printf("Height: %i\n", vid_win.height);
printf("ChromaKey: %i\n", vid_win.chromaKey);
printf("Flags: %i\n", vid_win.flags);
/*
printf("Clips: %i\n", camera.vid_win.clips);
printf("Clipcount: %i\n", camera.vid_win.clipcount);
*/

printf("-----\n");

printf("Brightness:\t%i (%i)\n", vid_pic.brightness, vid_pic.brightness/256);
printf("Hue:\t\t%i (%i)\n", vid_pic.hue, vid_pic.hue/256);
printf("Color:\t\t%i (%i)\n", vid_pic.colour, vid_pic.colour/256);
printf("Contrast:\t%i (%i)\n", vid_pic.contrast, vid_pic.contrast/256);
printf("Whiteness:\t%i (%i)\n", vid_pic.whiteness, vid_pic.whiteness/256);
printf("Depth:\t\t%i\n", vid_pic.depth);
printf("Palette:\t%i\n", vid_pic.palette);
}

bool Camera::Grab(void)
{
    int len;
    unsigned char *temp;
    unsigned char swap;

    if(dev<=0)
        return false;

    len = read (dev, picbuff, vid_caps.maxwidth * vid_caps.maxheight*3);
    if (len <= 0) {
        fprintf(stderr, "Error reading image...\n");
        return false;
    }
    temp = picbuff;

    for(int i=0;i<len/3;i++) {
        swap=*temp;
        *temp=*(temp+2);
        *(temp+2)=swap;
        temp+=3;
    }

    return true;
}

unsigned char * Camera::GetPixels(void)
{
    return picbuff;
}

int Camera::GetWidth(void)
{
    return vid_caps.maxwidth;
}
```

```
}

int Camera::GetHeight(void)
{
    return vid_caps.maxheight;
}

#if 0

void init_cam(struct Camera *camera)
{
    camera->greyscale = 0;
    camera->pic = NULL;
    camera->picbuff = NULL;
    camera->draw = 0;
    camera->pixmap = NULL;
    camera->frozen = 0;
    camera->update_camera = 0;
    camera->saving = 0;
    camera->savetype = PNG;
    camera->capture = 1;
    camera->dev = 0;
    strcpy(camera->devname, "/dev/video");
    camera->docked = 1;
    camera->dump=0;
    camera->speed_fastest = 0;
    camera->currentsavepage = NULL;
    camera->timeout = 100;
    camera->on_timer = 0;
    camera->timer_struct.unit = SECONDS;
    camera->timer_struct.iscommand = 0;
    camera->swapcolors = 0;
    camera->save_struct.isinfo = 0;
    pthread_mutex_init( &camera->pref_mutex, NULL ); /* to modify pref/setting */
    pthread_mutex_init( &camera->freeze_mutex, NULL ); /* to freeze display */
    pthread_mutex_init( &camera->iscam_mutex, NULL ); /* is there an open cam? */
}

sem_t s_draw;
sem_t s_grab1, s_grab2;
int plsquit = 0;
int x_frames = 0;
int y_frames = 0;

void set_cam_info(struct Camera *camera)
{
    if (ioctl (camera->dev, VIDIOCSPICT, &camera->vid_pic) == -1) {
        perror ("ioctl (VIDIOCSPICT)");
    }
    if (ioctl (camera->dev, VIDIOCSWIN, &camera->vid_win) == -1) {
        perror ("ioctl (VIDIOCSWIN)");
    }
}

void display(struct Camera *camera)
{
    GdkDrawable *drawable;
    int xpos=0, ypos=0;
    GdkRectangle update_rec;
    char tbuff[1024];

    while( !plsquit ) {
        if( !sem_wait( &s_draw ) ) {
            unsigned char *tmp;
```

```
tmp = camera->pic;
camera->pic = camera->picbuff;
camera->picbuff = tmp;

sem_post( &s_grab1 );

update_rec.x = 0;
update_rec.y = 0;
update_rec.width = camera->vid_caps.maxwidth;
update_rec.height = camera->vid_caps.maxheight;

xpos = (camera->vid_caps.maxwidth - camera->vid_win.width)/2;
ypos = (camera->vid_caps.maxheight - camera->vid_win.height)/2;

drawable = camera->drawing_area->window;

gdk_threads_enter();
/* Run filters */
if (camera->swapcolors)
    swap_rgb24(camera);
if (camera->autobright)
    auto_bright(camera);

if (camera->greyscale)
    gdk_draw_gray_image (camera->pixmap, camera->drawing_area->style->white_gc, xpos
s, ypos, camera->vid_win.width, camera->vid_win.height, GDK_RGB_DITHER_NORMAL, camera->
pic, camera->vid_win.width);
else
    gdk_draw_rgb_image (camera->pixmap, camera->drawing_area->style->white_gc, xpos
, ypos, camera->vid_win.width, camera->vid_win.height, GDK_RGB_DITHER_NORMAL, camera->p
ic, camera->vid_win.width*3);
gdk_threads_leave();

gdk_threads_enter();
gtk_statusbar_pop( GTK_STATUSBAR( camera->statusbar ), 12 );
sprintf( tbuff, "Speed (fps) - Average: %d Current: %d", camera->fps_avg, camera
->fps_current );
gtk_statusbar_push( GTK_STATUSBAR( camera->statusbar ), 12, tbuff );
gtk_widget_draw (camera->drawing_area, &update_rec);
gdk_threads_leave();
}
x_frames++;
y_frames++;
}
return;
}

void grab_image(struct Camera *camera)
{
    int len;
    GdkRectangle update_rec;
    GdkEventExpose *event;

    get_cam_info(camera);

    update_rec.x = 0;
    update_rec.y = 0;
    update_rec.width = camera->vid_caps.maxheight;
    update_rec.height = camera->vid_caps.maxwidth;

    while ( !plsquit ) {
        // order matters! the sem_waits MUST be before the mutex lock
        if( !sem_wait( &s_grab1 ) && ( camera->speed_fastest || !sem_wait( &s_grab2 ) ) &&
```

```
!pthread_mutex_lock( &camera->freeze_mutex ) && !pthread_mutex_lock( &camera->iscam_mutex ))){

    pthread_mutex_lock( &camera->pref_mutex );
    if (camera->update_camera){
        set_cam_info(camera);
        get_cam_info(camera);

        camera->update_camera = 0;
    }
    pthread_mutex_unlock( &camera->pref_mutex );

    if( camera->dev )
    }
    pthread_mutex_unlock( &camera->freeze_mutex );
    pthread_mutex_unlock( &camera->iscam_mutex );
    if( camera->dump )
        return;
    if(camera->on_timer){
        savefile_append_time(camera);
        if(camera->timer_struct.beep)
            gdk_beep();
        switch (camera->savetype) {
        case PPM:
            ppm_save(camera);
            break;

        case JPEG:
            jpeg_save(camera);
            break;

        case PNG:
            png_save(camera);
            break;
        }
        if(camera->timer_struct.iscommand){
            printf("Commanding...\n");
            system(camera->timer_struct.command);
        }
    }
    sem_post( &s_draw );
}
}

void delete_event(GtkWidget *widget, struct Camera *camera)
{
    FILE *preffile;
    char savefile[255];

    sprintf(savefile, "%s/.gqcamrc", getenv("HOME"));
    preffile = fopen(savefile, "w");
    save_pref_file(preffile, camera);
    fclose(preffile);

    gtk_main_quit ();
}

int next_frame(struct Camera *camera) {
    int val;

    sem_getvalue( &s_grab2, &val );
    if( !val )
        sem_post( &s_grab2 );
}
```

```
    return 1;
}

int increment_second_counter(struct Camera *camera) {
    static int x_seconds = 0;

    x_seconds++;
    // the following just prevents integer wrap
    if( ( x_seconds % 512 ) == 0 ) {
        x_seconds /= 2;
        x_frames /= 2;
    }
    camera->fps_avg = x_frames / x_seconds;
    camera->fps_current = y_frames;
    y_frames = 0;
    return;
}

int main(int argc, char *argv[])
{
    static struct Camera camera;
    pthread_t grab_thread;
    pthread_t draw_thread;
    quint timeoutid;
    int i, brightness=180, contrast=104, whiteness=155;
    unsigned char buff[3];
    char *filename = NULL, readfile[255];
    int done = 0;
    FILE *preffile;

    init_cam(&camera);
/*
    g_thread_init(NULL);
    gtk_init (&argc, &argv);
    gdk_rgb_init();
*/

    sprintf(readfile, "%s/.gqcamrc", getenv("HOME"));
    preffile = fopen(readfile, "r");
    if(preffile != NULL){
        read_pref_file(preffile, &camera);
        fclose(preffile);
    }

    while( !done ) {
        static struct option long_options[] =
        {
            { "help", no_argument, NULL, 'h' },
            { "version", no_argument, NULL, 'V' },
            { "swap", no_argument, NULL, 's' },
            { "autobright", no_argument, NULL, 'a' },
            { "video", required_argument, NULL, 'v' },
            { "dump", required_argument, NULL, 'd' },
            { "type", required_argument, NULL, 't' },
            { "bright", required_argument, NULL, 'b' },
            { "white", required_argument, NULL, 'w' },
            { "contrast", required_argument, NULL, 'c' },
            { "maxfps", required_argument, NULL, 'm' },
            { "fullspeed", no_argument, NULL, 'F' },
            { 0, 0, 0, 0 }
        };
        int c;
```

```
c = getopt_long( argc, argv, "FhVsav:d:b:w:c:m:t:", long_options, NULL );
switch ( c ) {
case 'h' :
    print_usage();
    exit(0);
    break;
case 'V' :
    printf( "gqcam version %s\n", version );
    exit(0);
    break;
case 's' :
    camera.swapcolors = 1;
    break;
case 'a' :
    camera.autobright = 1;
    break;
case 'v' :
    sprintf(camera.devname, "%s", optarg );
    break;
case 'd' :
    camera.dump = 1;
    if( strcmp( optarg, "-" ) != 0 ) /* leave alone if stdout */
        filename = optarg;
    break;
case 't' :
    if(!strcmp(optarg, "PPM"))
        camera.savetype = PPM;
    else if(!strcmp(optarg, "PNG"))
        camera.savetype = PNG;
    else if(!strcmp(optarg, "JPEG"))
        camera.savetype = JPEG;
    break;
case 'b' :
    brightness = atoi( optarg );
    break;
case 'w' :
    whiteness = atoi( optarg );
    break;
case 'c' :
    contrast = atoi( optarg );
    break;
case 'm' :
    if( atoi(optarg) > 0 ) /* Avoid weird values */
        camera.timeout = 1000/atoi(optarg);
    break;
case 'F' :
    camera.speed_fastest = 1;
    camera.timeout = 0;
    break;
case '?' :
    fprintf( stderr, "invalid option, or ambiguous argument\n" );
    print_usage();
    exit(1);
    break;
case EOF :
    done = 1;
    break;
default :
    print_usage();
    exit(1);
};
}
```



```
gdk_threads_leave();  
  
    return 0;  
}  
  
#endif
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#include <SDL.h>
#include "graphics.h"

Graphics::Graphics(int X, int Y, int bpp, int flags)
{
    Xres=X;
    Yres=Y;
    Bpp=bpp;
    Flags=flags;
    Screen=NULL;
}

Graphics::~Graphics()
{
}

bool Graphics::Init(void)
{
    Screen = SDL_SetVideoMode(Xres, Yres, Bpp, Flags);
    if( Screen == NULL) {
        fprintf(stderr, "Error setting the video mode: %s", SDL_GetError());
        return false;
    }

    Back = SDL_CreateRGBSurface(SDL_HWSURFACE, Screen->w, Screen->h,
                               Screen->format->BitsPerPixel,
                               Screen->format->Rmask,
                               Screen->format->Gmask,
                               Screen->format->Bmask,
                               Screen->format->Amask);

    if(Back == NULL) {
        fprintf(stderr, "CreateRGBSurface failed: %s\n", SDL_GetError());
        return false;
    }
    printf("Video Mode Set at %dx%dx%d\n",
           Screen->w, Screen->h, Screen->format->BitsPerPixel);
    return true;
}

void Graphics::Draw(Surface &image)
{
    if ( SDL_MUSTLOCK(Screen) ) {
        if ( SDL_LockSurface(Screen) < 0 ) {
            fprintf(stderr, "Cannot lock screen!\n");
            return;
        }
    }
}
```

```
int x,y;

for(y=0;y<image.yres;y++) {
    Uint16 *line = ((Uint16 *)Screen->pixels) + y*Screen->pitch/2;
    for(x=0;x<image.xres;x++) {
        Uint16 *buff = line + x;
        Pixel temp = image.data[y*image.xres+x];
        *buff = SDL_MapRGB(Screen->format, temp.Red, temp.Green, temp.Blue);
    }
}

if ( SDL_MUSTLOCK(Screen) ) {
    SDL_UnlockSurface(Screen);
}
SDL_UpdateRect(Screen, 0, 0, 0, 0);
}
```

```
/*
  Copyright (C) 2003 Tyler Nielsen

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

#include <SDL.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/extensions/shape.h>
#include <Imlib.h>
#include <iostream>
#include "jpeg.h"

using namespace std;

Jpeg::Jpeg(string device)
{
    char *Buffer;

    disp = XOpenDisplay(NULL);
    if (disp == NULL) {
        cerr << "Jpeg: Could not open X display" << endl;
        exit(-1);
    }

    id = Imlib_init(disp);
    if (id == NULL) {
        cerr << "Jpeg: Could not init Imlib" << endl;
        exit(-1);
    }

    Buffer = new char[device.length()];
    strcpy(Buffer, device.c_str());
    im = Imlib_load_image(id, Buffer);
    delete [] Buffer;
    if (im == NULL) {
        cerr << "Jpeg: Could not open image" << endl;
        exit(-1);
    }

    width = im->rgb_width;
    height = im->rgb_height;
}

Jpeg::~Jpeg()
```

```
{  
}  
  
void Jpeg::PrintInfo(void)  
{  
    cout << "Width: " << width << "\nHeight: " << height << endl;  
}  
  
bool Jpeg::Grab(void)  
{  
    return true;  
}  
  
unsigned char * Jpeg::GetPixels(void)  
{  
    return (unsigned char *)im->rgb_data;  
}  
  
int Jpeg::GetWidth(void)  
{  
    return width;  
}  
  
int Jpeg::GetHeight(void)  
{  
    return height;  
}
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#include <SDL.h>
#include <iostream>
#include <unistd.h>
extern "C" {
#include <pgm.h>
}
#include "camera.h"
#include "graphics.h"
#include "jpeg.h"
#include "pam.h"
using namespace std;

int main(int argc, char *argv[])
{
    Image *input;
    Graphics TheScreen;
    int state = 1;
    bool save = false;
    bool cheat = false;
    FILE *output;
    set <string> possible;
    int maxcount = 0;
    output = popen("./talkey.pl", "w");

    pgm_init(&argc, argv);

    if( SDL_Init(SDL_INIT EVERYTHING) != 0) {
        fprintf(stderr, "Error initializing SDL: %s", SDL_GetError());
        return 1;
    }

    if(!TheScreen.Init()) {
        fprintf(stderr, "Graphic initialization failed\n");
        return 1;
    }

    if(argc==1) {
        input = new Camera("/dev/v4l/video0");
    }
    else {
        input = new Jpeg(argv[1]);
    }

    input->PrintInfo();

    while(input->Grab()) {
        SDL_Event event;
```

```
while ( SDL_PollEvent(&event) ) {
    switch (event.type) {
    case SDL_KEYDOWN:
        switch(*SDL_GetKeyName(event.key.keysym.sym)) {
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                state = (*SDL_GetKeyName(event.key.keysym.sym))- '0';
                break;
            case 's':
            case 'S':
                save = true;
                break;
            case 'c':
            case 'C':
                cheat = true;
                break;
            case 'Q':
            case 'q':
                goto done;
            }
            break;
        case SDL_QUIT:
            goto done;
        }
    }

    Surface image(input->GetPixels(), input->GetWidth(), input->GetHeight());

    if(state == 1) {
        TheScreen.Draw(image);
        continue;
    }

    if(image.Gray2Black())
        continue;

    if(state == 2) {
        TheScreen.Draw(image);
        continue;
    }

    if(image.IsolateSignal())
        continue;

    if(state == 3) {
        TheScreen.Draw(image);
        continue;
    }

    if(image.IdentifyBlocks())
        continue;

    if(state == 4) {
        TheScreen.Draw(image);
        if (save) {image.SaveBlocks(); save = false;}
        continue;
    }
}
```

```
}

if(image.DetermineColors())
    continue;

if(state == 5) {
    TheScreen.Draw(image);
    if (save) {image.SaveBlocks(); save = false;}
    continue;
}

if(image.Correlate())
    continue;

if(!feof(output)) {
    fprintf(output, "%s\n", image.getWord().c_str());
    fflush(output);
}

if(possible.empty() && maxcount == 0) {
    FILE *dictwords;
    char Buffer[1000];
    char Temp[1000];
    sprintf(Buffer, "echo %s | ./pwords.pl", image.getWord().c_str());
    dictwords = popen(Buffer, "r");

    if(dictwords==NULL) {
        perror("popen");
        return 1;
    }

    while(!feof(dictwords)) {
        fgets(Buffer, 1000, dictwords);
        sscanf(Buffer, "%s", Temp);
        possible.insert(string(Temp));
    }
    maxcount = possible.size();
}

set <string>::iterator i;

i = possible.find(image.getWord());
if(i!=possible.end()) {
    possible.erase(i);
}

if(cheat) {
    cheat = false;
    for(i = possible.begin(); i!= possible.end(); i++) {
        cout << *i << endl;
    }
}

cout << image.getWord() << " there are " << possible.size() << " out of "
    << maxcount << " words left." << endl;

if(state == 6) {
    TheScreen.Draw(image);
    continue;
}

}

done:
```

```
fclose(output);  
SDL_Quit();  
return 0;  
}
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
// Many routines in this file are from _Numerical_Recipies_in_C_, 2nd
// ed, Press et. al., 1991. Such routines have a --NRC-- in their header
// comment.

#include <math.h>
#include "nrc.h"

// --NRC--
// Given an array of data[0..n-1], this routine returns its mean in
// *ave, average deviation in *adev, standard deviation in *sdev, variance
// in *var, skewness in *skew, and kurtosis in *curt.
void moment(double data[], int n, double *ave, double *adev, double *sdev,
            double *var, double *skew, double *curt)
{
    int j;
    double ep = 0.0, s, p;

    if (n <= 1) return; // This is an error
    s = 0.0;
    for (j = 0; j < n; ++j)
        s += data[j];
    *ave = s/n;
    *adev = (*var) = (*skew) = (*curt) = 0.0;
    for (j = 0; j < n; ++j) {
        *adev += fabs(s = data[j] - (*ave));
        ep += s;
        *var += (p = s*s);
        *skew += (p *= s);
        *curt += (p *= s);
    }

    *adev /= n;
    *var = (*var - ep*ep/n)/(n - 1);
    *sdev = sqrt(*var);

    if (*var) {
        *skew /= (n*( *var) * (*sdev));
        *curt = (*curt)/(n*( *var)*( *var)) - 3.0;
    } else return; // This is an error
}

// --NRC--
// Given 2 arrays x[0 .. n-1] and y[0 .. n-1], this routine computes their
// correlation coefficient r, the significance level at which the null
// hypothesis of zero correlation is disproved (prob whose small value
// indicates a significant correlation), and Fisher's z, whose value can be
```

```
// used in further statistical tests.
void pearson(double x[], double y[], unsigned long n, double *r, double *prob,
             double *z)
{
    unsigned long j;
    double yt, xt, t, df;
    double syy = 0.0, sxy = 0.0, sxx = 0.0, ay = 0.0, ax = 0.0;

    for (j = 0; j < n; ++j) {
        ax += x[j];
        ay += y[j];
    }
    ax /= n;
    ay /= n;

    for (j = 0; j < n; ++j) {
        xt = x[j] - ax;
        yt = y[j] - ay;
        sxx += xt*xt;
        syy += yt*yt;
        sxy += xt*yt;
    }

    *r = sxy/sqrt(sxx*syy);

    *z = 0.5*log((1.0 + (*r) + TINY)/(1.0 - (*r) + TINY));

    df = n-2;
    t = (*r)*sqrt(df/((1.0 - (*r) + TINY)*(1.0 + (*r) + TINY)));

    // Use the large-n approximation to save time. See pp. 638-639.
    *prob = erfcc(fabs((*z)*sqrt(n - 1.0))/1.4142136);
}

// --NRC--
// Returns the complementary error function erfc(3) with fractional error
// everywhere less than 1.2e-7.
double erfcc(double x)
{
    double t, z, ans;
    z = fabs(x);
    t = 1.0/(1.0 + 0.5*z);
    ans = t*exp(-z*z - 1.26551223 + t*(1.00002368 + t*(0.37409196 +
        t*(0.09678418 + t*(-0.18628806 + t*(0.27886807 +
        t*(-1.13520398 + t*(1.48851587 + t*(-0.82215223 +
        t*0.17087227)))))))));
    return x >= 0.0 ? ans : 2.0-ans;
}

// --NRC--
// Returns in heap[0..m-1] the smallest m elements of the array arr[0..n-1],
// with heap[0] guaranteed to be the mth smallest element. The array arr is
// not altered. For efficiency, this routing should be used only when
// m << n.
//void hpsel(unsigned long m, unsigned long n, Pair arr[], Pair heap[])
//{
//    unsigned long i, j, k;
//    Pair swap;
//
//    // This is an error: if (m > n/2 || m < 1)
//
//    for (i = 0; i < m; ++i)
```

```
//      heap[i] = arr[i];
//      for (i = m; i < n; ++i) {
//          if (arr[i].v < heap[0].v) {
//              heap[0] = arr[i];
//              for (j = 0;;) {
//                  k = j << 1;
//                  if (k > m)
//                      break;
//                  if (m != m && heap[k].v < heap[k+1].v)
//                      ++k;
//                  if (heap[j].v >= heap[k].v)
//                      break;
//                  swap = heap[k];
//                  heap[k] = heap[j];
//                  heap[j] = swap;
//                  j = k;
//              }
//          }
//      }
//  }
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
extern "C" {
#include <pgm.h>
}
#include <stdio.h>
#include "pam.h"
#include "pixel.h"

void writeImage(Pixel *image, int xres, int yres, char *file)
{
    FILE *fp = fopen(file, "w");

    if(fp == NULL)
        return;

    fprintf(fp, "P3\n");
    fprintf(fp, "#SDLBlocks 1.0\n");
    fprintf(fp, "%d %d 255\n", xres, yres);

    for (int i = 0; i < yres; ++i) {
        for (int j = 0; j < xres; ++j)
            fprintf(fp, "%d %d %d\n",
                    image[i*xres + j].Red,
                    image[i*xres + j].Green,
                    image[i*xres + j].Blue);
    }
    fclose(fp);
}

void readImage(Pixel **image, int &xres, int &yres, const char *file)
{
    FILE *input = fopen(file, "r");
    char Buffer[1000];
    int ignore;

    if(input == NULL)
        return;

    fgets(Buffer, sizeof(Buffer)*sizeof(char), input);
    if(Buffer[0]!='P' || Buffer[1]!='3') {
        fprintf(stderr, "'%s' is not a pnm file!\n", file);
        return;
    }

    do {
        fgets(Buffer, sizeof(Buffer)*sizeof(char), input);
    } while(Buffer[0]!='#');
```

```
    sscanf(Buffer, "%d %d", &xres, &yres);

    fscanf(input, "%d", &ignore);

    *image = new Pixel[xres*yres];

    int r,g,b;

    for (int i = 0; i < yres; ++i) {
        for (int j = 0; j < xres; ++j) {
            fscanf(input, "%d %d %d", &r, &g, &b);
            (*image)[i*xres + j].Red=r;
            (*image)[i*xres + j].Green=g;
            (*image)[i*xres + j].Blue=b;
        }
    }
    fclose(input);
}

/*
void writeImage(Pixel *image, int xres, int yres, char *file)
{
    FILE *fp = fopen(file, "w");
    gray * g;

    pgm_writepgm_init(fp, xres, yres, PGM_MAXMAXVAL, 0);
    g = pgm_allocrow(xres);
    for (int i = 0; i < yres; ++i) {
        for (int j = 0; j < xres; ++j)
            g[j] = (image[i*xres + j].Red + image[i*xres + j].Green +
                    image[i*xres + j].Blue)/3;
        pgm_writepgmrow(fp, g, xres, PGM_MAXMAXVAL, 0);
    }
    pgm_freerow(g);
    fclose(fp);
}
*/
```

```
/*
Copyright (C) 2003 Tyler Nielsen

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*/
#include <SDL.h>
#include <string.h>
#include <math.h>
#include <iostream>
#include <vector>
#include <set>
#include <stdlib.h>
#include <assert.h>
#include "graphics.h"
#include "pixel.h"
#include "nrc.h"
#include "pam.h"
#include "letters.h"

using namespace std;

Pixel *Surface::oldimage = NULL;
vector<Block> Surface::testcase;
const Pixel Surface::refwhite = Pixel(180, 172, 180);
Pixel Surface::white;

ostream& operator << (ostream &left, const Pixel &right)
{
    return left << "("
        << int(right.Red) << ", "
        << int(right.Green) << ", "
        << int(right.Blue) << ")";
}

class Pair {
public:
    int n;
    double v;
    Pair(int nt = 0, double vt = 0) {n = nt; v = vt;}
};

class Bins {
private:
    struct DataPoints{
        int num;
        double points;
    };

    int borderWidth;
    int blockWidth;
    vector<DataPoints> Points;
```

```
public:
    Bins(int border, int width) {
        borderWidth = border;
        blockWidth = width;
    }

    void add(int p) {
        vector<DataPoints>::iterator i;
        for (i = Points.begin(); i != Points.end(); ++i) {
            double distance = fabs(p - i->points/i->num);

            // First check to see if this point falls within borderWidth of
            // a previous bin
            if (distance < borderWidth) {
                i->points += p;
                i->num=i->num+1;
                return;
            }

            // Now, check to see if this point is at _least_ 0.75blockWidth
            // away from all previous bins
            else if (distance < 0.7*blockWidth)
                return;
        }

        DataPoints temp;
        temp.points = p;
        temp.num = 1;
        Points.push_back(temp);
    }

    int numBins(void) {
        return Points.size();
    }

    void printBins(void) {
        vector<DataPoints>::iterator i;
        for (i = Points.begin(); i != Points.end(); ++i)
            cout << i->points/i->num << " ";
        cout << endl;
    }

    void getBins(vector<int> &fill) {
        fill.clear();
        vector<DataPoints>::iterator i;
        for (i = Points.begin(); i != Points.end(); ++i) {
            fill.push_back((int)i->points/i->num);
        }
    }

};

bool operator ==(const Pixel &left, const Pixel &right)
{
    if(left.Red == right.Red &&
        left.Green == right.Green &&
        left.Blue == right.Blue)
        return true;
    return false;
}

bool operator !=(const Pixel &left, const Pixel &right)
{
    if (left.Red != right.Red ||
```

```
        left.Green != right.Green ||
        left.Blue  != right.Blue )
    return true;
return false;
}

bool operator ==(const Point &left, const Point &right)
{
    if(left.x == right.x &&
        left.y == right.y)
        return true;
    return false;
}

bool operator < (const Point &left, const Point &right)
{
    if((left.x<<10)+left.y < (right.x<<10)+right.y)
        return true;
    return false;
}

int comp(const void *e1, const void *e2);

Surface::Surface(unsigned char *RGB, int x, int y)
{
    if(oldimage == NULL) {
        oldimage = new Pixel[x*y];
        memset(oldimage, 0, x*y*3);
    }

    unsigned char *temp = (unsigned char *)oldimage;

    for(int i = 0; i<x*y*3; i++) {
        temp[i] = (unsigned char)(RGB[i]*.4 + temp[i]*.6);
    }

    data = new Pixel[x*y];
    xres = x;
    yres = y;
    original = new Pixel[x*y];

    memcpy(data, oldimage, x*y*3);
    memcpy(original, oldimage, x*y*3);
}

Surface::~~Surface()
{
    delete [] data;
    data = NULL;
    delete [] original;
    original = NULL;
}

int Surface::ColorAmplify(void)
{
    //This is based on the fact that we only care about four colors.
    //Make everything else white

    int x,y;
```

```
int minval;
int maxval;
// const int diff = 20;

for(y=0;y<yres;y++) {
  for(x=0;x<xres;x++) {
    Pixel current = data[y*xres+x];
    minval = min(min(current.Red, current.Blue), current.Green);
    maxval = max(max(current.Red, current.Blue), current.Green);
    /*
    if(current.Red>current.Blue+diff*2 && current.Green>current.Blue+diff*2) {
      current.Red=255;
      current.Blue=0;
      current.Green=255;
    }
    */
    if(maxval-minval<20) {
      current.Red=current.Blue=current.Green=0;
    }
    /*
    else if(current.Green>100) { // && current.Green>current.Blue) {
      current.Blue = 0;
      current.Red = 0;
      current.Green = 255;
    }
    else if(current.Blue>current.Red+diff && current.Blue>current.Green+diff) {
      current.Blue = 255;
      current.Red = 0;
      current.Green = 0;
    }
    else if(current.Red>current.Blue+diff && current.Red>current.Green+diff) {
      current.Blue = 0;
      current.Red = 255;
      current.Green = 0;
    }
    else {
      current.Red=255;
      current.Blue=255;
      current.Green=255;
    }
    */
    data[y*xres+x] = current;
  }
}
return 0;
}

int Surface::Gray2Black(void)
{
  // Look for pixels where the standard deviation between the colors is
  // small, which means the pixel is gray, and make them black.

  int x, y;
  double std_dev, sum, sxx;
  double ref_red, ref_green, ref_blue;

  ref_red = ref_green = ref_blue = 0;

  for (y = 0; y < 10; ++y) {
    for (x = 0; x < xres; ++x) {
      Pixel current = getpixel(data, x, y);
      ref_red += current.Red;
      ref_green += current.Green;
    }
  }
}
```

```
        ref_blue += current.Blue;
    }
}
for (y = yres - 10; y < yres; ++y) {
    for (x = 0; x < xres; ++x) {
        Pixel current = getpixel(data, x, y);
        ref_red += current.Red;
        ref_green += current.Green;
        ref_blue += current.Blue;
    }
}

ref_red /= 20*xres;
ref_green /= 20*xres;
ref_blue /= 20*xres;

white = Pixel((unsigned char) ref_red, (unsigned char)ref_green, (unsigned char)ref
_blue);

ref_red = 255 - ref_red;
ref_green = 255 - ref_green;
ref_blue = 255 - ref_blue;

//    cout << ref_red << ' ' << ref_green << ' ' << ref_blue << endl;

for (y = 0; y < yres; ++y) {
    for (x = 0; x < xres; ++x) {
        Pixel current = getpixel(data, x, y);

        current.Red = (unsigned char) ((current.Red + ref_red) > 255 ?
            255 : current.Red + ref_red);
        current.Green = (unsigned char)((current.Green + ref_green) > 255?
            255 : current.Green + ref_green);
        current.Blue = (unsigned char) ((current.Blue + ref_blue) > 255 ?
            255 : current.Blue + ref_blue);

        sum = current.Red + current.Blue + current.Green;
        sxx = current.Red*current.Red + current.Blue*current.Blue
            + current.Green*current.Green - sum*sum/3;
        std_dev = sqrt(sxx/2);

        if (std_dev < 20.0) {
            current.Red = current.Blue = current.Green = 0;
            getpixel(data, x, y) = current;
        }
    }
}
return 0;
}

int Surface::IsolateSignal(void)
{
    // Determine the total signal content of the image, and try to isolate
    // its primary source.  This should be the row of blocks.

    double signal;
    double line;
    int x, y;

    signal = 0;
    signalleft = 0;
    signalright = xres;
```

```
// Just kill the top and bottom quarter of the screen. We don't care
// about them anyway
for (y = 0; y < yres/4; ++y) {
    for (x = 0; x < xres; ++x) {
        Pixel current = getpixel(data, x, y);
        current.Red = current.Green = current.Blue = 0;
        getpixel(data, x, y) = current;
    }
}
for (y = 3*yres/4; y < yres; ++y) {
    for (x = 0; x < xres; ++x) {
        Pixel current = getpixel(data, x, y);
        current.Red = current.Green = current.Blue = 0;
        getpixel(data, x, y) = current;
    }
}

// Find the total signal content of the image
for (y = yres/4; y < 3*yres/4; ++y) {
    for (x = 0; x < xres; ++x) {
        Pixel current = getpixel(data, x, y);
        signal += current.Red + current.Green + current.Blue;
    }
}

// Throw away lines that don't contain a significant fraction of the
// total signal.
for (y = yres/4; y < 3*yres/4; ++y) {
    line = 0;
    for (x = 0; x < xres; ++x) {
        Pixel current = getpixel(data, x, y);

        line += current.Red + current.Green + current.Blue;
    }
    if (line/signal < 0.01) {
        for (x = 0; x < xres; ++x) {
            Pixel current = getpixel(data, x, y);
            current.Red = current.Blue = current.Green = 0;
            getpixel(data, x, y) = current;
        }
    }
    else {
        signaltop = y;
        break;
    }
}
for (y = 3*yres/4; y > yres/4; --y) {
    line = 0;
    for (x = 0; x < xres; ++x) {
        Pixel current = getpixel(data, x, y);

        line += current.Red + current.Green + current.Blue;
    }
    if (line/signal < 0.01) {
        for (x = 0; x < xres; ++x) {
            Pixel current = getpixel(data, x, y);
            current.Red = current.Blue = current.Green = 0;
            getpixel(data, x, y) = current;
        }
    }
    else {
        signalbottom = y;
        break;
    }
}
```

```
}
for (x = 0; x < xres; ++x) {
    line = 0;
    for (y = signaltop; y <= signalbottom; ++y) {
        Pixel current = getpixel(data, x, y);

        line += current.Red + current.Green + current.Blue;
    }
    if (line < 100) {
        for (y = signaltop; y <= signalbottom; ++y) {
            Pixel current = getpixel(data, x, y);
            current.Red = current.Blue = current.Green = 0;
            getpixel(data, x, y) = current;
        }
    }
    else {
        signalleft = x;
        break;
    }
}
for (x = xres-1; x > signalleft; --x) {
    line = 0;
    for (y = signaltop; y <= signalbottom; ++y) {
        Pixel current = getpixel(data, x, y);

        line += current.Red + current.Green + current.Blue;
    }
    if (line < 100) {
        for (y = signaltop; y <= signalbottom; ++y) {
            Pixel current = getpixel(data, x, y);
            current.Red = current.Blue = current.Green = 0;
            getpixel(data, x, y) = current;
        }
    }
    else {
        signalright = x;
        break;
    }
}

for(y=signaltop;y<=signalbottom;y++) {
    memcpy(&getpixel(data, signalleft, y),
           &getpixel(original, signalleft, y),
           (signalright - signalleft)*3);
}

if(signalleft == 0 || signalright == xres)
    return 1;
return 0;
}

int Surface::IdentifyBlocks(void)
{
    // Sweeping across the region that contains the most
    // signal--assumed to have already been identified using
    // IsolateSignal()--look for the region where the std deviation
    // across the colors reaches a minimum across an area the size of
    // a block border . This will be the starting point of the first
    // block.

    double *channel;
    Pair *std_dev;
    double r_avg, r_adev, r_sdev, r_var, r_skew, r_curt;
    double g_avg, g_adev, g_sdev, g_var, g_skew, g_curt;
```

```
double b_avg, b_adev, b_sdev, b_var, b_skew, b_curt;
int blockDim, borderWidth, x, cy, cx;
int numBlocks;

blockDim = signalbottom - signaltop;
borderWidth = blockDim/9;
borderWidth /= 2;

if(blockDim < 10)
    return 1;

std_dev = new Pair[signalright - signalleft];
channel = new double[blockDim*borderWidth];

for (x = signalleft; x < signalright - borderWidth; ++x) {
    // Copy the red channel of this region
    for (cy = 0; cy < blockDim; ++cy)
        for (cx = 0; cx < borderWidth; ++cx)
            channel[cy*borderWidth + cx] =
                getpixel(data, cx + x, cy + signaltop).Red;
    // Compute the std deviation of the red values
    moment(channel, blockDim*borderWidth, &r_avg, &r_adev, &r_sdev,
        &r_var, &r_skew, &r_curt);

    // Copy the green channel of this region
    for (cy = 0; cy < blockDim; ++cy)
        for (cx = 0; cx < borderWidth; ++cx)
            channel[cy*borderWidth + cx] =
                getpixel(data, cx + x, cy + signaltop).Green;
    // Compute the std deviation of the green values
    moment(channel, blockDim*borderWidth, &g_avg, &g_adev, &g_sdev,
        &g_var, &g_skew, &g_curt);

    // Copy the blue channel of this region
    for (cy = 0; cy < blockDim; ++cy)
        for (cx = 0; cx < borderWidth; ++cx)
            channel[cy*borderWidth + cx] =
                getpixel(data, cx + x, cy + signaltop).Blue;
    // Compute the std deviation of the blue values
    moment(channel, blockDim*borderWidth, &b_avg, &b_adev, &b_sdev,
        &b_var, &b_skew, &b_curt);

    // Now average the std deviations.
    // FIXME: Is this OK, math-wise?
    std_dev[x - signalleft] =
        Pair(x - signalleft, max(r_sdev, max(g_sdev,b_sdev)));
}

qsort(std_dev, signalright - signalleft - borderWidth, sizeof(Pair), comp);
//for (int i = 0; i < signalright - signalleft - borderWidth; ++i)
//cout << std_dev[i].v << " " << std_dev[i].n << endl;
//exit(0);

Bins bins(borderWidth, blockDim);

for (int i = 0; i < signalright - signalleft - borderWidth; i++) {
    bins.add(std_dev[i].n);
}

bins.getBins(BlockDiv);

vector<int>::iterator i, next;

//Find the number of Blocks (should be BlockDiv.size()+1)
```

```
//And the max block width. Also remove any divisions
//that are too close to either signalleft of signalright
for (i=BlockDiv.begin(); i != BlockDiv.end(); i = next) {
    if (*i < blockDim*.5 ||
        *i > (signalright-signalleft) - blockDim*.5) {
        next = BlockDiv.erase(i);
    }
    else {
        *i += signalleft;
        next = i + 1;
    }
}

// Force a division at the beginning and end of the row of blocks
BlockDiv.push_back(signalleft);
BlockDiv.push_back(signalright);

// Now sort all the divisions
sort(BlockDiv.begin(), BlockDiv.end());

// Copy the block images, drawing markers where we divided the blocks
Blocks.clear();
for (i = BlockDiv.begin(); i != BlockDiv.end() - 1; ++i) {
    Blocks.push_back(Block(data, *(i+1) - *i, blockDim,
                          *i, signaltop, xres, yres));
    for (int y = 0; y < yres; ++y) {
        Pixel current;
        current.Red = current.Green = current.Blue = 255;
        getpixel(data, *i, y) = current;
    }
}

BlockHeight = blockDim;
numBlocks = Blocks.size();

delete [] channel;
delete [] std_dev;
return 0;
}

int Surface::BlackAndWhite(void)
{
    vector<Block>::iterator i;
    for (i = Blocks.begin(); i != Blocks.end(); ++i)
        i->BlackAndWhite();
    DrawBlocks();
    return 0;
}

int Surface::DetermineColors(void)
{
    vector<Block>::iterator i;
    for (i = Blocks.begin(); i != Blocks.end(); ++i)
        i->DetermineColor();
    DrawBlocks();
    return 0;
}

int Surface::DrawBlocks(void)
{
    vector<Block>::iterator i;
    int xPos = 0, yPos = 0;
```

```
// Clear the image
memset(data, 0, xres*yres*3);

//Move the blocks so they are BlockWidth+1 appart
for (i=Blocks.begin(); i != Blocks.end(); i++) {
    if (xPos + i->blockWidth >= xres) {
        xPos = 0;
        yPos += BlockHeight + 20;
    }
    i->drawBlock(data, xPos, yPos, xres, yres);
    for(int x=0;x<i->blockWidth;x++) {
        for(int y=BlockHeight;y < BlockHeight+5; y++) {
            getpixel(data, xPos+x, yPos+y) = PixelColor[(int)i->color];
        }
    }
    xPos += i->blockWidth + 20;
}
return 0;
}

int Surface::SaveBlocks(void)
{
    char buff[100];
    int i = 0;
    vector<Block>::iterator j;
    for (j = Blocks.begin(); j != Blocks.end(); ++j, ++i) {
        sprintf(buff, "blocks/block%02d.pnm", i);
        j->SaveBlock(buff);
    }
    return 0;
}

int comp(const void *e1, const void *e2)
{
    if (((Pair *)e1)->v < ((Pair *)e2)->v)
        return -1;
    if (((Pair *)e1)->v > ((Pair *)e2)->v)
        return 1;
    return 0;
}

void Surface::loadtestcase(void)
{
    testcase.clear();
    int idx;
    int numletters;
    char Buffer[1000];

    for(idx = 0; idx < 26; idx++) {
        for(numletters = 0; numletters < Number[idx]; numletters++) {
            sprintf(Buffer, "blocks/%c%d.pnm", idx+'a', numletters);
            testcase.push_back(Block(Buffer, idx+'A', Colors[idx]));
        }
    }
    printf("Size of testcase = %d\n", testcase.size());
}

int Surface::Correlate(void)
{
    if(testcase.empty())
        loadtestcase();

    vector<Block>::iterator b;
```

```
vector<Block>::iterator tc;

double alllet[26];
char count[26];

double max,best;

word = "";

for(b =Blocks.begin(); b!=Blocks.end(); b++) {
    max = -100;
    best = 0;
    for(int i=0;i<26;i++) {
        alllet[i] = 0;
        count[i] = 0;
    }
    for (tc=testcase.begin(); tc != testcase.end(); tc++) {
        if(!((b->color==BLUE || b->color==GREEN )&& (tc->color==BLUE || tc->color==GREEN)
) && (b->color!=tc->color))
            // if((b->color!=tc->color))
                continue;
        double curcor = BlockCorrelation(*b, *tc, 1);
        alllet[tc->letter-'A']+=curcor;
        count[tc->letter-'A']++;
        // cout << tc->letter << ":" << curcor << " ";
    }
    // cout << endl;
    for(int i=0;i<26;i++) {
        alllet[i]=alllet[i]/count[i];
        if(alllet[i]>max) {
            // cout << char(i+'A') << ":" << alllet[i] << " ";
            max=alllet[i];
            best = i;
        }
    }
    if(max>-100) {
        word += char(best+'A');
    }
    // cout << endl << endl;
}

DrawBlocks();
return 0;
}
```

```
#!/usr/bin/perl

#Copyright (C) 2003 Tyler Nielsen
#
#This program is free software; you can redistribute it and/or
#modify it under the terms of the GNU General Public License
#as published by the Free Software Foundation; either version 2
#of the License, or (at your option) any later version.
#
#This program is distributed in the hope that it will be useful,
#but WITHOUT ANY WARRANTY; without even the implied warranty of
#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#GNU General Public License for more details.
#
#You should have received a copy of the GNU General Public License
#along with this program; if not, write to the Free Software
#Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

sub load_dictionary;

$dictionary = "/usr/share/dict/words";
load_dictionary();

while(<>) {
    chop;
    $word = uc $_;
    %letters = ();
    for ($i = 0; $i < length($word); ++$i) {
        $letter = substr($word, $i, 1);
        ++$letters{$letter};
    }

    WORD: foreach $dictword (keys(%dict)) {
        foreach $dictletter (keys %{ $dict{$dictword} }) {
            #print "$dictword $dictletter $letters{$dictletter} $dict{$dictword}{$dictl
etter}\n";
            next WORD if $letters{$dictletter} < $dict{$dictword}{$dictletter};
        }
        print "$dictword\n";
    }
}

sub load_dictionary
{
    open(FILE, "< $dictionary") or
        die "Couldn't open dictionary file $dictionary: $!\n";
    while(<FILE>) {
        chop;
        $word = uc $_;
        $dict{$word} = ();      # Just to be safe

        for ($i = 0; $i < length($word); ++$i) {
            $letter = substr($word, $i, 1);
            ++$dict{$word}{$letter};
        }
    }
    close(FILE);
}
```

```
#!/usr/bin/perl

#Copyright (C) 2003 Tyler Nielsen
#
#This program is free software; you can redistribute it and/or
#modify it under the terms of the GNU General Public License
#as published by the Free Software Foundation; either version 2
#of the License, or (at your option) any later version.
#
#This program is distributed in the hope that it will be useful,
#but WITHOUT ANY WARRANTY; without even the implied warranty of
#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#GNU General Public License for more details.
#
#You should have received a copy of the GNU General Public License
#along with this program; if not, write to the Free Software
#Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

use IO::Socket::INET;

sub load_dictionary;

$host = "localhost";
$port = 1314;
$dictionary = "/usr/share/dict/words";

$skt = IO::Socket::INET->new(PeerAddr => $host,
                             PeerPort => $port,
                             Proto    => "tcp",
                             Type     => SOCK_STREAM)
    or die "Couldn't connect to $host:$port: $!\n";

load_dictionary();

while(<>) {
    chop;
    $_ = uc $_;
    next unless $dict{$_};
    print $skt "(SayText ", "'", $_, "'", ')\n';
}

sub load_dictionary
{
    open(FILE, "< $dictionary") or
        die "Couldn't open dictionary file $dictionary: $!\n";
    while(<FILE>) {
        chop;
        $dict{uc($_)} = 1;
    }
    close(FILE);
}
```