CONSTRUCTING A SPECTRAL PHOTOMETER FOR THE STUDY OF LIGHT POLLUTION

A Senior Honors Thesis

by

CASEY PATRICK DEEN

Submitted to the Office of Honors Programs
& Academic Scholarships
Texas A&M University
in partial fulfillment of the requirements of the

UNIVERSITY UNDERGRADUATE
RESEARCH FELLOWS

April 2004

Major: Physics

CONSTRUCTING A SPECTRAL PHOTOMETER FOR THE STUDY OF LIGHT POLLUTION

A Senior Honors Thesis

by

CASEY PATRICK DEEN

Submitted to the Office of Honors Programs
& Academic Scholarships
Texas A&M University
in partial fulfillment of the requirements of the

UNIVERSITY UNDERGRADUATE
RESEARCH FELLOW

Approved as to style and content by:

George Kattawar
(Fellows Advisor)

Edward A. Funkhouser
(Executive Director)

April 2004

Major: Physics

ABSTRACT

Constructing a Spectral Photometer for the Study of

Light Pollution. (April 2004)

Casey Patrick Deen
Department of Physics
Texas A&M University

Fellows Advisor: Dr. George Kattawar
Department of Physics

In this paper, I describe a method and apparatus for carrying out a systematic spectroscopic mapping $I(\lambda, z, \Phi)$ of the night sky, as proposed by Stefano Rosoni. Once completed, this method should prove to be simple and effective, while the SLR film camera spectral photometer constructed and described within should prove to be inexpensive and easily reproducible. The spectral photometer is simply a single slit diffraction apparatus mounted to the rear of a telescope. The resulting interference pattern is recorded as an image projected on the film inside the camera. In order to correlate the image recorded on the film to a spectrum containing information about the intensity and wavelength of the light pollution, the photographs were scanned into digital format and analyzed by a series of computer programs. While a film spectral photometer is in itself, nothing new, the computer algorithm used to extrapolate film response curves was developed by Paul Debeveck and Jitendra Malik for use in computer graphics. I apply their algorithm to the problem of calibration of a spectral photometer and bypass the myriad of tedious and time consuming calibrations which make film cameras almost more trouble than they are worth. Problems from the unsuccessful first prototype are discussed, as well as suggested improvements for further versions

PACS numbers: 95.45. + i, 95.55.Qf, 95.75.Pq

DEDICATION

This work is dedicated to my father, who taught me the value of patience and persistence in scientific pursuits, and to my mother, who has put up with more than her share of my "experiments."

ACKNOWLEDGMENTS

Thanks are due to the following people for their help with the construction and development of this project: Don Carona of the Texas A&M Observatory, Dr. Donald Naugle, Daya Rathnayaka, Maryna Anatska, Saeed Adegbenro, Erin Kueht, and Mandy Deen.

I would like to thank my advisor, Dr. George Kattawar, of the Texas A&M Department of Physics for his suggestions and guidance through this project.

This research was supported by a Grant In Aid of Research, administered by Sigma Xi, the Scientific Research Society.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

METHODOLOGY

Construction of the Spectral Photometer

Table 1: Spectral Photometer Components

Orion SteadyPix SLR Camera Mount
Olympus 2000 SLR camera
Olympus 35-70mm Lens
Close focus lens
Shutter Release Cable
2" x 2" Diffraction Grating – 1000 lines per mm
Kodak T400 CN film
1 3/8" ID aluminum tube with 20 threads per inch on one end.
Brass shim stock
Card stock

The materials required to construct the spectral photometer are listed in Table 1. As shown in Figure 1, the camera is mounted on the Easy Pix camera mount, which is in turn mounted to the aluminum tube extension, which is mounted to the rear of the telescope via an eyepiece holder, as shown in Figure 2. The slit, which was constructed using two pieces of card stock and two pieces of brass shim stock (see Figure 3), has a separation of 2.5 mm. The slit is sandwiched between the eyepiece holder and the

telescope. The diffraction grating is attached to the end of the metal tube, in front of the camera lens with masking tape, sticky tack, or some other non-permanent adhesive. Depending on the size of the camera body/lens, it may be necessary to use a ¼" thick metal plate between the camera mount and the camera to allow the camera to focus properly. The close focus lens attaches to the camera lens and allows the camera to focus on the image of the diffraction patterns, which appear as colored lines on the diffraction grating. If the camera cannot focus on the diffraction pattern, it may be necessary to add the metal plate described above and depicted in Figure 4, or to lengthen the metal tube.

A piece of light-proof fabric (or rubber) can be used in relatively "bright" conditions to shield the camera from ambient light (see figure 5). This is not always necessary in sufficiently dark conditions.

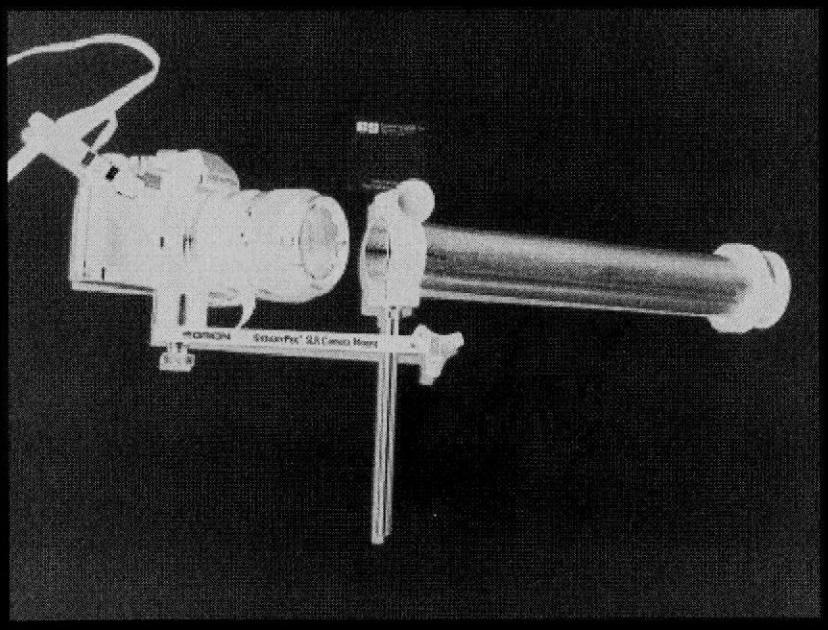

Figure 1. Camera mounted on EasyPix SLR camera adapter. Source: Author

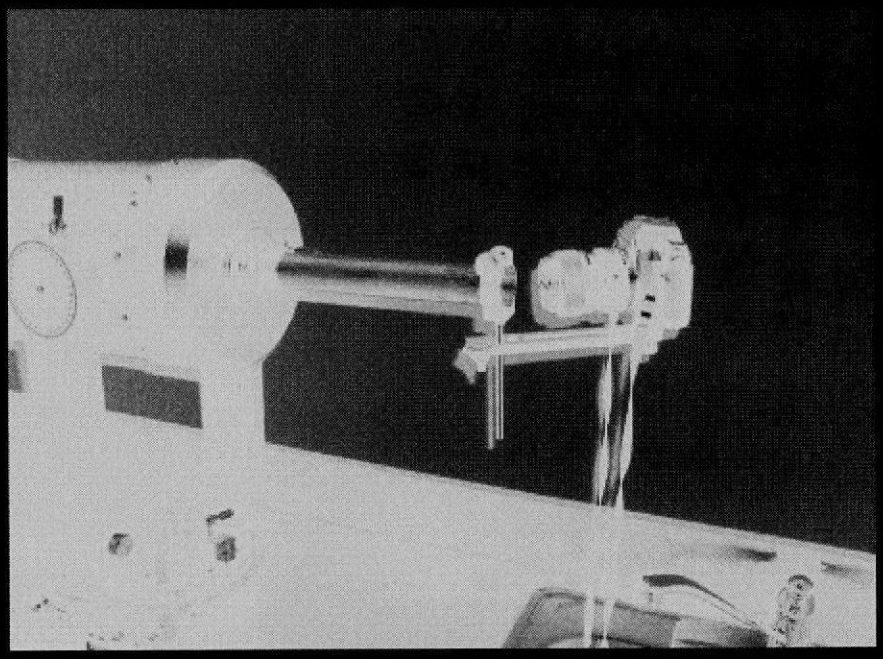

Figure 2. Apparatus mounted on telescope. Source: Author

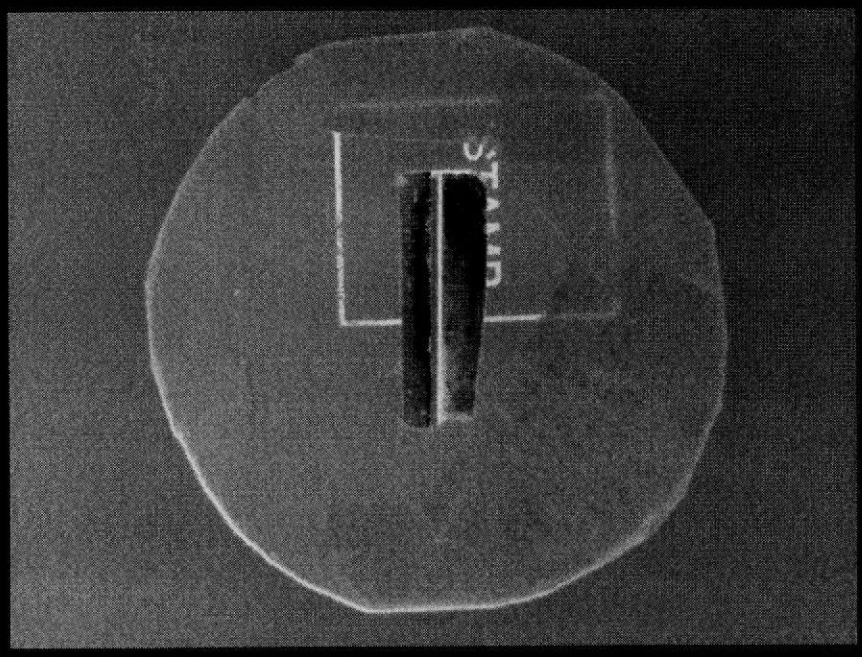

Figure 3. Slit Construction. Source: Author

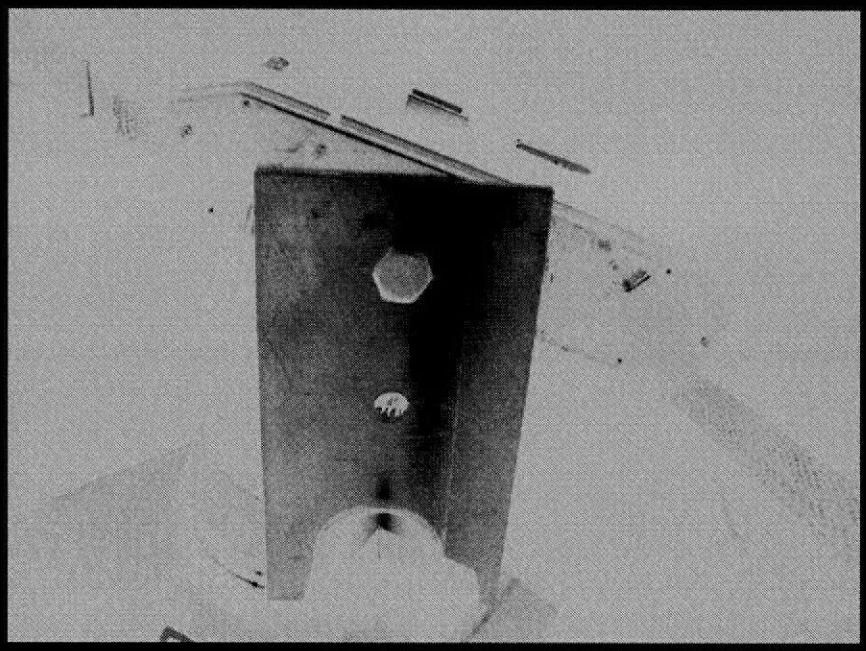

Figure 4. Adapter Plate. Source: Author

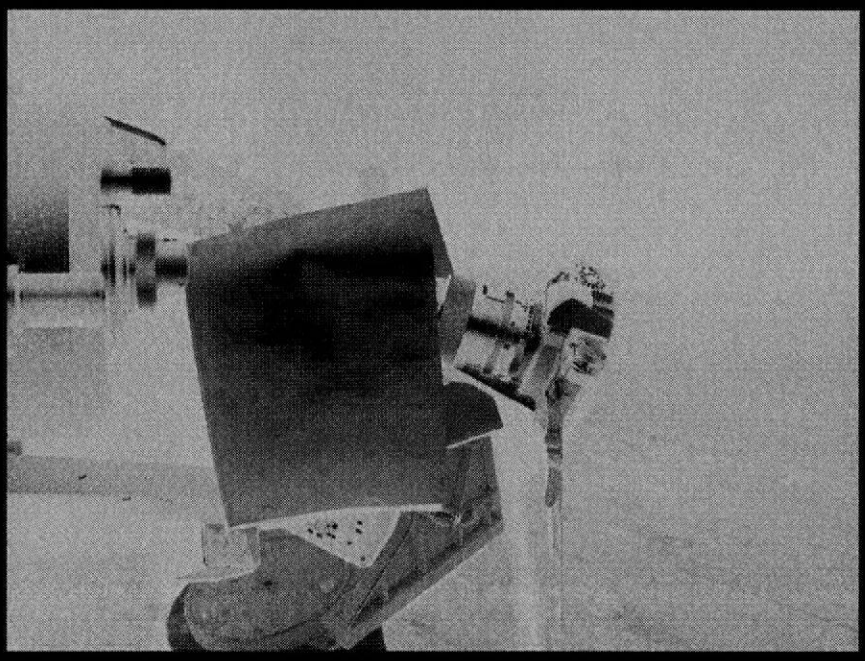

Figure 5. Light Shield. Source: Author

Reciprocity

Film reciprocity is defined as the characteristic of film which produces a given density (opacity) when exposed to a certain amount of light. This relation can be summarized through a Hurter-Driffield curve (See Figure 6), which plots the optical

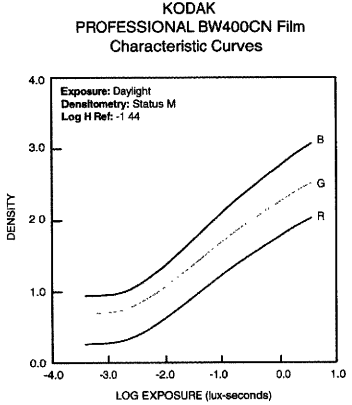

Figure 6. Hurter-Driffield Curve for Kodak 400CN film. Source: (Kodak a)

density of film versus the logarithm of the exposure ($E\Delta t$, where E is the irradiance and Δt is the shutter speed). Under normal photographic conditions, where exposures are between 1/10,000 of a second, and a few seconds, reciprocity holds, i.e. the density of the film is proportional to the amount of light striking the film. However, if the

exposure time is lengthened to the order of tens of seconds or more, reciprocity begins to fail, meaning that the density of the film is no longer proportional to the amount of light which struck the film during the exposure. Reciprocity failure was not thought to be an issue for this experiment, because Kodak CN400 film, which does not suffer reciprocity failure until exposures longer than 120 seconds, was the film of choice (Kodak a).

Data Acquisition

In order to take data, the apparatus was connected to the telescope by way of the eyepiece mount, as detailed in the Construction section. The telescope acts as simply a light gathering device. Without the telescope, the diffracted light, which is orders of magnitude dimmer than the incident light, is just simply too dim to overcome the reciprocity failure of the film.

Once the telescope and apparatus are mounted on the telescope stand, the telescope should be trained on an area of interest, recording the zenith and azimuthal angles. Using a cable release, the shutter on the camera should be opened for a predetermined amount of time, usually on the order of several minutes. The tracking system on the telescope mount, if available should not be enabled. The zenith / azimuthal dependence of the light pollution will not change with the rotation of the earth, as the sources of light pollution are terrestrial and rotate with the earth.

Once a roll of film is exposed, it should be taken to a photo lab to have the negatives developed. The negatives are then taken to the darkroom, where positives are developed according to the procedures described in Appendix A. It is important to note that the procedures and chemicals in Appendix A are not the only procedures or chemicals that will work. However, it is vital that the SAME procedures used in the

calibration be followed for any spectra to be analyzed with that particular calibration scheme. Once the positives have been developed, they are then scanned into digital format and stored on a computer in .PCX format using an ordinary scanner. The .PCX file format was chosen because of the simplicity of the file format, although any file format which does not trade image quality for compactness is acceptable (i.e. .BMP, .PCX, not .JPG).

The areas of the photographs containing "interesting" data (i.e. the spectrum) was then cropped and rotated so that the slit image was vertical, spreading the different wavelengths horizontally across the image (Figure 7)

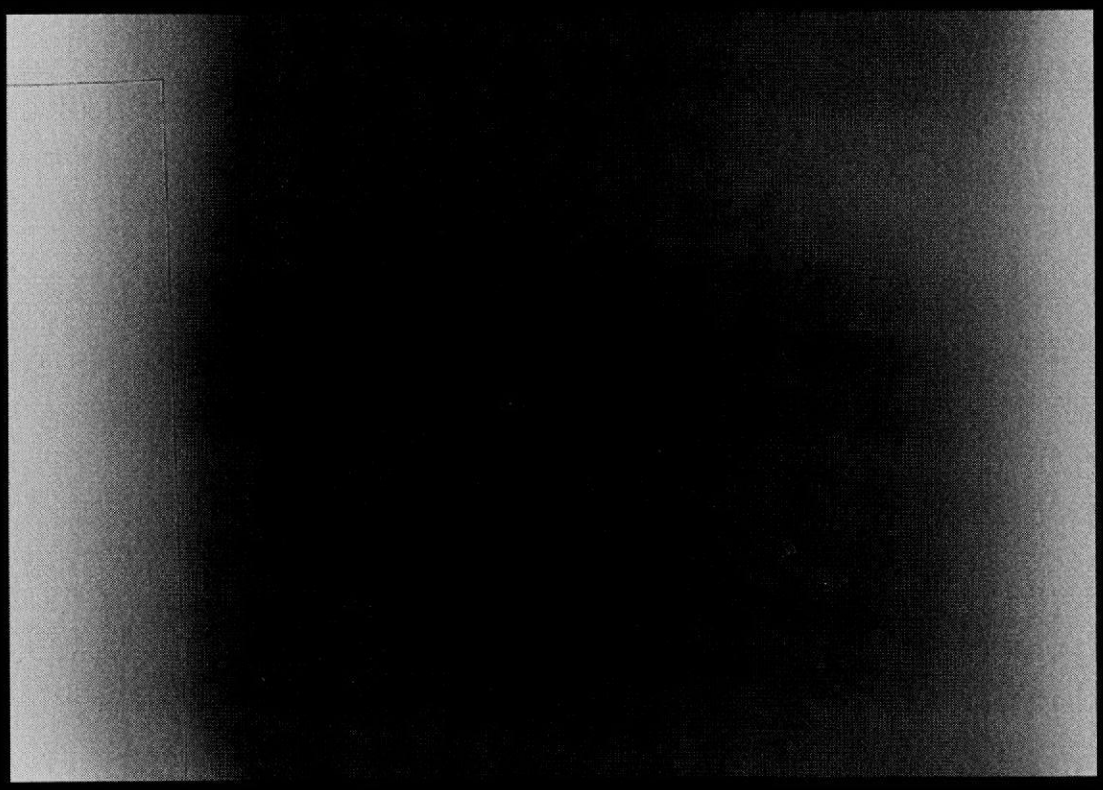

Figure 7. Spectrum of an incandescent light. Shorter wavelengths are on the right of the image, while longer wavelengths are on the left of the image. Source: Author

Calibration of Photographic System

Any photographic system incorporates many non-linear mappings in the process of recording an image to film, photograph paper, or digital storage. Said another way, given two pixels, one twice as "bright" as the other, this does not imply that the light represented by the first pixel is twice as intense as the light represented by the second pixel (Debevec 1997).

To reliably correlate pixel brightness with intensity, the photographic system must be calibrated. In order to calibrate the system, an incandescent lamp with a dimmer dial should be used to create several different spectra of varying intensity. For each spectrum, several photographs should be taken with the spectral photometer, each with a different shutter speed. These images should then be fed into the computer algorithm described in the next section. The algorithm will extrapolate the film response curve for each wavelength of light, and relate the exposure time of an image and a pixel value to an intensity value. Only then can the intensity of different wavelengths be reliably compared. The algorithm used in this project was derived from the algorithm presented in Debevec and Malik and is described in detail in the next section.

DISCUSSION OF ALGORITHM

As stated above, the main algorithm on which this research is based was presented by Debevec and Malik. The original intent for the algorithm was for use in computer

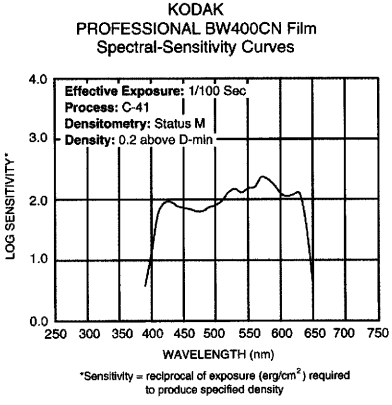

Figure 8. Spectral Sensitivity for Kodak BW400CN film. Source: (Kodak a)

graphics, to improve virtual renderings of physical objects. By measuring the change in brightness for several different pixels in the picture, each through several different shutter speeds, the algorithm extracts a film response curve, which relates the amount of light passing through the shutter, to a digital brightness value. The algorithm, while quite powerful, is limited by the spectral response of the photographic system. As their paper states, the film response curve is accurate to a factor of the spectral sensitivity of

the camera/film. For the purpose of scanning images for virtual renderings, it is sufficient to ignore this artifact. Unfortunately, the aim of a spectral photometer is to measure relative/absolute intensities of different wavelengths of light. Depending mostly on the film used, the spectral sensitivity varies widely with wavelength, as shown in Figure 8, so any response curve calculated from the entire spectrum would average all the variations in spectral sensitivity, which is undesirable for use in a spectral photometer. However, all is not lost, as a different response curve can be calculated for each individual wavelength of light. The only drawback to this is that it requires a great deal more pictures to be taken for calibration of the apparatus.

The implementation of the algorithm as described in Debevec and Malik's 1997 paper calls for a scene with a wide range of intensity values to be photographed several times with each exposure at a different shutter speed. Once the images are digitized, the algorithm picks out a number of pixels throughout the image and examines how the intensity values of those pixels change throughout the different exposures.

Translating the Algorithm: Goblin.c++

Before setting about modifying Debevec and Malik's algorithm, it was necessary to first ensure that the original algorithm could be made to work. To this end, a C++ version of the Matlab algorithm was written using the JAMA (Java Matrix Library) and TNT (Template Numerical Toolkit) mathematics packages for C++ devloped by the National Institute for Standards and Technology on their website at http://math.nist.gov. To test it, pictures were taken of an indoor scene with a digital camera (Figure 9). Since this was not the final product, the digital camera was chosen for ease of transfer to the digital format, although a film camera could have been used just as well. Fifty pixels were then selected from throughout the scene and the

brightness value for each one (0 being completely black, 255 being completely white) was fed into the algorithm stated in Debevec and Malik's paper. The algorithm returned a film response curve, plotting the natural log of the exposure against pixel value (Figure 10), mirroring the response curve presented in Debevec and Malik's paper. The code for goblin.c++ can be found in Appendix B.1

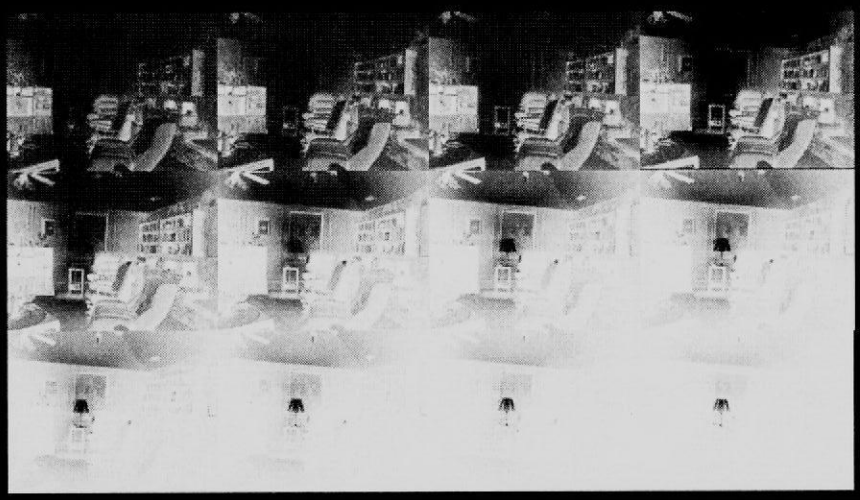

Figure 9. Set of 12 images of varying exposure used to test Goblin.c++. Shutter speeds range from 16 seconds (top left) to 1/20 of a second (bottom right). Source: Author

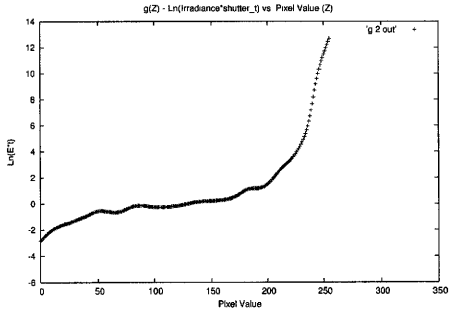

Figure 10. Output of Goblin.c++. g(z) relates pixel values the natural logarithm of the exposure. Source: Author

Modifying the Algorithm: Gandalf.c++

After successfully translating the original algorithm from Matlab to C++, the algorithm was then modified to attack the problem of calibration of a spectral photometer. In order to do this, a slight paradigm shift was necessary in the way individual photographs were used. The original algorithm required N pictures of unique exposure times and selected P pixels from throughout the scene to extrapolate a film response curve for the entire photograph. As stated above, a film response curve for the entire spectrum will not work for this application because of the wide variance in spectral response. Instead, each wavelength must be treated as its own separate photographic system, each with its own separate response curve. This means that several sets of images must be taken. Within each set, images must vary the shutter

speed, and each entire set must be taken of a different spectrum of light. Each column in entire set of photographs of a certain spectrum of light corresponds to a "pixel" in an overall "picture" of that particular wavelength. This can be accomplished by using an incandescent lamp with a dimmer switch. The dimmer switch can be used to vary the intensity of the light

The drawback to this approach is the length of the calibration process. In order to sufficiently over-determine the system of system of linear equations employed by the algorithm, for a set of N photographs and P pixels, N*(P-1) must be greater than (Z_{max} – Z_{min}) where Z_{max} and Z_{min} are the brightest (255) and darkest (0) pixel values, respectively. This means 256 different photographs are required to calibrate the spectrometer. For this implementation, the number of pixels P will be the number of different intensity settings for incandescent light. The number of photographs N will be the number of different shutter speeds in each set of photographs of an intensity setting.

While 256 is quite a large number of photographs to develop in a darkroom at one time, it only need be done once. Also, for calibration purposes, sheets of contact prints of many negatives can be made by placing the negatives directly on top of the photographic paper and exposing all the negatives at once. While the prints will be very small, they can be enlarged digitally, and the time trade off is more than worth it. Once the calibration is complete, it can be applied to any spectra taken by the photographic system.

Once the calibration photographs are stored in the computer, they are fed into the computer program by way of a data file, containing the names of the images as well as the shutter speeds for each image. The program then computes the response curve for each wavelength and the writes that particular response curve to a computer file

titled by lambdaXXX.dat where XXX is the horizontal location of the wavelength in question. These files can later be used to apply the calibration to a spectrum of light taken as data. The source code for Gandalf.c++ is available in Appendix B.2

Auxiliary Programs: Hobbit.c

In order to prepare the scanned images for use in Gandalf.c++, the information encoded in the .PCX file must first be extracted from the images. Because the spectra are oriented horizontally, each column of pixels corresponds to a certain wavelength of light. By averaging over pixel brightness value for the entire column, the program obtains the average brightness value for that wavelength. Also calculated is the standard deviation of the columnar distribution corresponding to that particular wavelength. These averages and standard deviations, and the corresponding x-pixel positions are then output to a text file for use by Gandalf.c++. The source code for hobbit.c is shown in Appendix B.3

OBSTACLES

As the project progressed, numerous obstacles became apparent. Originally, film was to be processed completely and digitally scanned at a 1-hour photo lab, reducing the amount of time and effort required to obtain digital images for processing with the algorithms. Unfortunately, the photo lab used to develop the film automatically individually processed each of the exposures to obtain 18% brightness in each print, meaning that each image was processed under a different method. Unfortunately the algorithm assumes each image has undergone the same development procedure. The negatives, however, were processed uniformly, and therefore correctly showed the variations. Because the negatives had been processed uniformly, they could be used to generate prints in a darkroom and still ensure that each exposure had undergone the same development process. If a photo lab could have been found which did not individually process each print, the darkroom procedures would have been unnecessary.

In the original plans, Kodak TMZ 3200 black and white film was to be used in the camera, due to its extremely fast ASA speed. This was abandoned, however, due to the extreme reciprocity failure of the film. At exposures of 100 seconds or more, the film had lost over 2 stops of reciprocity; in order to expose film for a corrected exposure time of 100 seconds, the shutter must be open for at least 400 seconds (Kodak b). In other words, if there was not enough light to record the image on the film in the first few seconds, it most likely was not going to get recorded. After realizing the severity of the reciprocity failure, the film was changed to a lower-speed film which did not require black & white chemistry to process, Kodak CN400, namely. The CN400 film does not suffer reciprocity failure until approximately 120 seconds, according to Kodak's

technical document on the film (Kodak a)

Unfortunately, even with the telescope and the low-speed film, the spectra turned out to be too dim to be recorded on the film. Even after a 17 minute exposure, no discernible image of a spectrum could be seen on the film. The failure of the film to record any spectrum whatsoever is testament to the fact that while light pollution can be extremely disrupting to astronomical observation, it is exceedingly difficult to record. In the next section, I make some suggestions about how to improve the spectrometer to obtain usable data.

RECOMMENDATIONS

The first and foremost recommendation is a change in recording media. While the original intent of the project was to create a low-budget spectral photometer out of an ordinary manual camera, it may be necessary to use a digital camera, even a CCD camera to finally be able to pick up the faint spectrum produced by the light pollution. While CCD cameras would be idea for the task, they are also quite expensive, running upwards of a thousand dollars, well out of range for the average amateur astronomer.

One method that was not attempted, but is worthy of consideration is dry-nitrogen film hypering. When film is "hypered," it is placed in an environment of hydrogen gas and baked. This increases the sensitivity of the film and helps combat reciprocity failure. This would add quite a bit of complexity, however.

Digital cameras, while not as sensitive as CCDs, do hold promise in the fact that they completely remove film from the process of transferring the image to the computer, thus eliminating a great deal of headache. Also, one must be careful in buying a digital camera for this purpose, to ensure that the camera gives the user enough control over the exposure. For example, one needs to be able to control the length of the exposure almost indefinitely. Another aspect important to have control over is the focus. Instead of focusing on the diffraction grating, it is necessary to focus on the image of the slit, which appears some distance behind the grating.

Another possible way to improve results is to increase the number of photons entering the camera. This can be achieved by either placing the apparatus on the back of a larger telescope, or by changing the dispersive element to a blazed holographic diffraction grating, which has a higher transmission ratio than does a normal diffraction grating.

One practical item which should be addressed in further versions of the spectral photometer is the problem of locating the beginning of the spectrum in the photographs. Most of the cropping of the images was haphazardly done by estimation and guess work, introducing unnecessary error into the algorithm. Perhaps a small LED located in one of the corners of the diffraction grating could be used to demarcate a reference point in each image.

Also, while the computer programs are functional and provide useful output, they are a bit clunky and hard to use. Perhaps a more unified user interface can be developed, which would allow much easier manipulation of the images.

CONCLUSIONS

A method for constructing, calibrating, and operating a low-budget spectral photometer has been discussed. Although this particular implementation of the spectral photometer has not proved successful in that the photometer was unable to pick up any spectrum at all. However, I believe that the methods outlined for calibration and analysis are useful and applicable. In this respect, the project has been a success. If the problem of the film reciprocity can be solved, the algorithms outlined in the appendices will serve to calibrate the spectral photometer and compare the relative intensities of different wavelengths of light pollution.

By pursuing the goal of a low-budget yet effective spectral photometer, amateur astronomers around the world will be able to join in the fight against light pollution by gathering data as to the intensity, wavelength, and angular position in the night sky. This will help professional astronomers, theorists, and policy makers come up with better strategies to combat the light pollution which is slowly encroaching on our night sky.

REFERENCES

Debevec, P. and Malik. J. 1997, Recovering High Dynamic Range Radiance Maps from Photographs, SIGGRAPH 97.

Eastman Kodak. 2004b, Tech Pub F-4036.

Eastman Kodak. 2004a, Tech Pub F-4016.

Reeves, R. 2004, Films compared for Astronomy, http://www.robertreeves.com/filmtest.htm

Rossoni, S. 2000, Proposal of a Spectroscopic Map of Astronomical Sites, Journal of the Italian Astronomical Society, Vol. 71 No. 1, 235-238.

APPENDIX A

Darkroom Procedures

To minimize headaches and the need for non-standard equipment and a truly dark-room, all negatives in this experiment were processed at a photo lab and then positives were developed in a quasi-dark room using the procedure described below. Prints were not obtained from the photo lab because the computer program used to process the prints tried to over or under-expose each exposure to achieve a certain percentage of light to dark. This results in a non-uniform process, and will not give the proper film response curve when fed into the algorithm. In the event that a photo lab can be found which does not auto-correct each exposure, it would be acceptable, even suggested, to skip this entire process of development, as it is extremely tedious and difficult.

Materials:

- Darkroom set (Enlarger, 2 tongs, 3 developing trays, red safe-light, stopwatch)
- Small square of transparent plastic
- Overhead marker
- Two lightproof containers for undeveloped photo paper
- Kodak Dektol Developer
- Kodak Stopbath
- Kodak Fixer

With the room lights out and the safe-light on, the film was inserted facing upside down into the enlarger and the spectrum was centered on the paper stage. Then, an arrow and number were written on the transparent plastic corresponding to the

forward direction and exposure number of the film. With the safe-light off, a piece of photographic paper was removed from the lightproof container and placed on the paper stage. The small square of transparent plastic was then placed in the upper left corner of the picture, so that it did not obscure the image projected on the paper. The enlarging lamp was then turned on. After 8 seconds, the enlarging lamp was turned off, and the paper removed from the stage and placed in another lightproof container. This process was repeated for however many exposures needed to be developed.

After all the prints had been exposed, the enlarger was turned off, and the safe-light turned on. Single sheets of exposed photographic paper were then taken out of the second lightproof container and developed one by one. They were first placed in a tray containing a working solution of Kodak developer for 45 seconds. Then, using the developer tongs, the print was taken from the developer tray and placed in the stop bath tray for 30 seconds, exercising care that the developer tongs never touch the stop bath. Then, using the stop bath tongs, the print was transferred to the fixer tray for a period of at least a minute. After the fixer, the print was moved using the stop bath tongs to the sink where it was washed for another two minutes. After the wash, it was hung to dry. This process was repeated until all the prints had been processed. Only then, could the normal room lights be turned back on.

APPENDIX B

Computer Code

1. Goblin.c++

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tnt.h"
#include "jama_qr.h"

using namespace TNT;
using namespace JAMA;

struct pcx_header  // Header format for PCX images
{
   char signature;
   char version;
   char encoding;
   char bytes_per_pixel;
   unsigned short int xmin;
   unsigned short int ymin;
   unsigned short int xmax;
   unsigned short int ymax;
   unsigned short int vres;
   unsigned short int hres;
   char palette[48];
   char reserved;
   char color_layers;
   unsigned short int bytes_per_line;
   unsigned short int palette_type;
   char unused[58];
};

struct pcx_struct
{
   pcx_header header;
   unsigned char * image;
   unsigned char pallete[768];
   unsigned short int length, height;
};

const int QTY_PIX = 11;   /* Number of pictures to be processed */
const float L = 35.0;      /* Smoothness factor */
const float ZMIN = 0.0;   /* Minimum Pixel Value */
const float ZMAX = 255.0; /* Maximum Pixel Value */

unsigned char bytes_per_pixel;

void readpcximage(FILE * file,void * target,int size)
{
  unsigned char buf;
  unsigned int counter;
  int i=0;

  while(i<=size)  /* Image not entirely read? */
    {
      /* Get one byte */
      fread(&buf,1,1,file);
      /* Check the 2 most significant bits */
      if ((buf&192)==192)
       {
         /* We have 11xxxxxx */
         counter=(buf&63);         /* Number of times to repeat next byte */
         fread(&buf,1,1,file);     /* Get next byte */
```

```
        for(;counter>0;counter--) /* and copy it counter times */
          {
            ((char*)target)[i]=buf;
            //printf("%d\n", buf);
            i++;   /* increase the number of bytes written */
          }
      }
      else
       {
         /* Just copy the byte */
         ((char*)target)[i]=buf;
         //printf("%d\n", buf);
         i++,   /* Increase the number of bytes written */
       }
    }
}

pcx_struct readpcx(FILE *file)
    /* return_struct.image=NULL if failed, otherwise a pointer to the loaded image */
{
   pcx_struct temp;
   fseek(file, 0, SEEK_SET);
   fread(&temp.header, sizeof(pcx_header), 1, file);  /* Reads the PCX header */
   if((temp.header.signature!=0x0a)||(temp.header.version!=5))  /* Checks if file is PCX
format */
   {
      temp.image = NULL;
      return (temp);
   }
   else
   {/* it is! */
      /* Return height and length */
      temp.length=temp.header.xmax+1-temp.header.xmin;
      temp.height=temp.header.ymax+1-temp.header.ymin;
      /* Allocate the sprite buffer */
      temp.image=(unsigned char *)malloc((temp.length)*(temp.height));
      /* Read the image */
      /*printf("length :%d height :%d",temp.length, temp.height);
         Useful for debugging purposes */
      readpcximage(file,temp.image,(temp.length)*(temp.height));
      /* PCX succesfully read! */
      return(temp);
   }
}

void getPixs(int rands[50])  /* Gets xy coords of the chosen pixels */
{
   FILE * pixelFile;
   int x, y, length, height;

   if ( (pixelFile=fopen("pixeldata.txt", "r"))==NULL)
   {
      printf("Error!  Could not open Pixel Data File\n");
   } else
   {
      fscanf(pixelFile, "%d %d", &length, &height);
      for(int i = 0; i < 50; i++)
      {
         fscanf(pixelFile, "%d %d", &x, &y);
         rands[i] = (int)(((float)x/4.0) + ((float)y/4.0)*(float)length);
      }
      fclose(pixelFile);
   }
}

void grabPixels(FILE *file, float pixelArray[][50], float exposureArray[QTY_PIX])
{
   char imageName[20];
   FILE * imageFile;
```

```
   pcx_struct spect;

   unsigned char buf;
   int pix[50];
   int j, i = 0;

   getPixs(pix);  // Gets XY coords of chosen pixels

   fseek( file, 0L, SEEK_SET );
   while (i < QTY_PIX)
   {
      fscanf(file, "%f %s", &exposureArray[i], imageName);
      if ( (imageFile=fopen(imageName, "r"))==NULL)
      {
         printf("Error!  Could not open image: %s\n", imageName);
      } else
      {
         spect=readpcx(imageFile);
         if (spect.image==NULL)
         {
            printf("Error loading file!");
         } else
         {
            for( j = 0; j < 50; j++ )
            {
               buf = spect.image[pix[j]];   // Picks out elected pixels
               pixelArray[i][j] = (float)(int)buf;
            }

            i++;
            delete spect.image;
         }
      }
      fclose(imageFile);
   }
}

float weight(float i)   // A simple Hat function for weighting
{
   if( i <= (ZMIN+ZMAX)/2.0)
   {
      return i - ZMIN;
   } else
   {
      return ZMAX - i;
   }
}

void computeResponseCurve(float pixArr[][50], float expArr[])
{

   int i, j, k, xsize, ysize, n = 256;
   float wij;
   xsize = QTY_PIX*50+n+1;
   ysize = n + 50;
   Array2D< double > A(xsize,ysize);      /* create MxN array; all zeros */
   Array1D< double > b(xsize);
   k = 0;
   for( i = 0; i < 50; i++)
        for( j = 0; j < QTY_PIX; j++)// Fills array with chosen pixel data
        {
           wij = weight(pixArr[j][i]+1.0);
           A[k][(int)(pixArr[j][i]+1.0)] = wij;
           A[k][n+i] = -wij;
           b[k] = wij * expArr[j];
           k++;
        }

   A[k][129] = 1.0;
```

```
    k++;
    for( i = 0; i < n-2; i++)
    {
       A[k][i] = L * weight(i+1);
       A[k][i+1] = -2 * L * weight(i+1);
       A[k][i+2] = L * weight(i+1);
       k++;
    }

    QR< double > x(A);

    Array1D< double > ans = x.solve(b);  /* Solves system of equations */

    for( i = 0, i < n+50; i++)
       printf("%d %f\n", i, ans[i]),
}


int main(int argc, char ** argv)
{
    FILE * file;
    float pixelArray[QTY_PIX][50],
    float exposureArray[QTY_PIX];

    if (argc!=2)      /* Checks for invalid invocation */
    {
       printf("Usage: goblin namefile.txt.\n");
       return(1);
    }
    if ((file=fopen(argv[1],"r"))==NULL)   /* Checks to see if file is readable */
    {
       printf("Cannot open file!\n");
       return(1);
    }
    grabPixels(file, pixelArray, exposureArray);  //Loads data in memory
    fclose(file);     /* Closes file once image is loaded into memeory */
    computeResponseCurve(pixelArray, exposureArray);
              //Computes the spectral response curve.
    return(0);
}
```

The above program must be run with the following syntax:

>goblin namefile.txt

namefile.txt is a text file containing a list of the names of the images and their

corresponding exposure times. e.g.:

name of file Exposure time (in seonds)

one.pcx 1.5
two.pcx 0.5
....
It also must be run in the same director as pixelfile.txt, which lists the X and Y

coordinates of the pixels to be examined in each image. e.g.:

x coordinate y coordinate

356 1254

642 572

....

2. Gandalf.c++

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tnt.h"
#include "jama_qr.h"

using namespace TNT;
using namespace JAMA;

const int QTY_EXP = 4;        /* # of different exposure times to be processed */
const int QTY_PIX = 3;        /* #of diff intensity settings (# of "pixels") */
const int SPECT_WIDTH = 880; /* Width in pixels of the spectrum */
const float L = 100.0;        /* Smoothness factor */
const float ZMIN = 0.0;       /* Minimum Pixel Value */
const float ZMAX = 255.0;     /* Maximum Pixel Value */


/* Reads in the image information from the data file created by hobbit.c */

void readImage(FILE *file, float pixels[SPECT_WIDTH])
{
   int t;
   float trash;
   fseek( file, 0L, SEEK_SET );
   for( int i = 0; i < SPECT_WIDTH; i++)
      fscanf(file, "%d %f %f", &t, &pixels[i], &trash);
}

/* grabPixels fills each wavelength array with the intensity information stored in the
data files */

void grabPixels(FILE *file, float pixelArray[][QTY_EXP][QTY_PIX], float
exposureArray[QTY_EXP])
{
   char imageName[20];
   FILE * imageFile;
   float image[SPECT_WIDTH];

   int j, k, i = 0;
   k = 0;

   fseek( file, 0L, SEEK_SET );
   while (k < QTY_PIX)
   {
      while (i < QTY_EXP)
      {
         fscanf(file, "%s", imageName); /* Reads next file name from datafile */
         if ( (imageFile=fopen(imageName, "r"))==NULL)
         {
            printf("Error!  Could not open image: %s\n", imageName);
         } else
         {
            readImage(imageFile, image);    /* Reads in data file to memory */
            for( j = 0; j < SPECT_WIDTH; j++ )
            {
               pixelArray[j][i][k] = image[j]; /* Fills pixelArray with image */
            }
            i++;
```

```
        }
        fclose(imageFile);
    }
    k++;
  }
  for(i = 0; i < QTY_EXP; i++)          /* Reads exposure times into memory */
    fscanf(file, "%f", &exposureArray[i]);
  i = 0;
}

float weight(float i)    /* A simple hat function */
{
  if( i <= (ZMIN+ZMAX)/2.0)
  {
    return i - ZMIN;
  } else
  {
    return ZMAX - i;
  }
}

/* the computeResponseCurve function computes, using the Debevec algorithm, the response
curve for each particular wavelength and stores it in a data file for use at a later
time. */

void computeResponseCurve(float pixArr[][QTY_EXP][QTY_PIX], float expArr[QTY_EXP], int
lambda)
{

  int i, j, k, xsize, ysize, n = 256;
  FILE * out;
  float wij;
  xsize = QTY_EXP*QTY_PIX+n+1;
  ysize = n + 50;
  Array2D< double > A(xsize,ysize);      /* create MxN array; all zeros */
  Array1D< double > b(xsize);
  k = 0;
  for( i = 0; i < QTY_PIX; i++)          /* Fills arrays with information */
       for( j = 0; j < QTY_EXP; j++)      /* contained in the arrays */
       {
          wij = weight(pixArr[lambda][j][i]+1 0);
          A[k][(int)(pixArr[lambda][j][i]+1.0)] = wij;
          A[k][n+1] = -wij;
          b[k] = wij * expArr[j];
          k++;
       }

  A[k][129] = 1.0;
  k++;
  for( i = 0; i < n-2; i++)
  {
     A[k][i] = L * weight(i+1);
     A[k][i+1] = -2 * L * weight(i+1);
     A[k][i+2] = L * weight(i+1);
     k++;
  }

  QR< double > x(A);

  Array1D< double > ans = x.solve(b);    /* Solves the system of equations */
  char buf[10];
  sprintf(buf, "lambda%d.out", lambda);

  if ( (out=fopen(buf, "w"))==NULL )  // Opens file for output
  {
     printf("Error!  Could not open file(s)!");
  } else
  {
//      for( i = 0; i < QTY_PIX; i++)
```

```
        fprintf(out, "%d %f\n", i, ans[0]);
      fclose(out);
   }
}

int main(int argc, char ** argv)
{
   FILE * file;
   float pixelArray[SPECT_WIDTH][QTY_EXP][QTY_PIX];
   float exposureArray[QTY_EXP];

   if (argc!=2)      /* Checks for invalid invocation */
   {
      printf("Usage: goblin namefile.txt.\n");
      return(1);
   }
   if ((file=fopen(argv[1],"r"))==NULL)   /* Checks if file is readable */
   {
      printf("Cannot open file'\n");
      return(1);
   }

   grabPixels(file, pixelArray, exposureArray);

   fclose(file);    /* Closes file once images are loaded into memeory */
   printf("Checkpoint #2\n");
   for(int i = 0; i < SPECT_WIDTH; i++)
      computeResponseCurve(pixelArray, exposureArray, i);
   return(0);
}
```

Gandalf must be run by the following command:

>Gandalf imagedata.txt

Below is a example of imagedata.txt

```
one1.out
one2.out
one3.out
two1.out
two2.out
two3.out
three1.out
three2.out
three3.out
0.125
0.25
0.5
```

The filenames point to the data files created by Hobbit.c and are listed in order of different spectra (i.e. one1.out and one2.out refer to the same spectrum, but have slightly different exposure times.) The three numbers at the end of the file correspond to the shutter speed for

each exposure. (i.e. one1.out and two1.out both have a shutter speed of 0.125 seconds).

3. Hobbit.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
  Hobbit is the first level pass that the raw .PCX image goes through.  Hobbit averages
the brightness value of each column of pixels in the input file.  As the diffraction
grating is oriented vertically, each column of pixels corresponds to a range of
wavelengths.  It also calculates the standard deviation of the brightness values in each
column.

  Input: a .PCX formatted image, an output file name (e.g. 'output.txt')

  Output: the x by y dimensions of the image, an output file ('output.txt') x rows of
data, formatted as follows:

  xvalue average_intensity std_dev \n

  where:

  xvalue - horizontal position in the image
  average_intensity - averaged intensity value through the entire column at horizontal
value xvalue
  std_dev - standard deviation of all intensity values at horizontal value xvalue

*/

typedef struct
{
   char signature;
   char version;
   char encoding;
   char bytes_per_pixel;
   unsigned short int xmin;
   unsigned short int ymin;
   unsigned short int xmax;
   unsigned short int ymax;
   unsigned short int vres;
   unsigned short int hres;
   char palette[48];
   char reserved;
   char color_layers;
   unsigned short int bytes_per_line;
   unsigned short int palette_type;
   char unused[58];
}PCX_Header;

unsigned char bytes_per_pixel;

void readpcximage(FILE * file,void * target,int size)
{
  unsigned char buf;
  unsigned int counter;
  int i=0;
  while(i<=size)  /* Image not entirely read? */
```

```
    {
      /* Get one byte */
      fread(&buf,1,1,file);
      /* Check the 2 most significant bits */
      if ((buf&192)==192)
       {
         /* We have 11xxxxxx */
         counter=(buf&63);         /* Number of times to repeat next byte */
         fread(&buf,1,1,file);     /* Get next byte */
         for(;counter>0;counter--) /* and copy it counter times */
           {
             ((char*)target)[i]=buf;
             //printf("%d\n", buf);
             i++;   /* increase the number of bytes written */
           }
       }
      else
       {
         /* Just copy the byte */
         ((char*)target)[i]=buf,
         //printf("%d\n", buf);
         i++;   /* Increase the number of bytes written */
       }
    }
}

void *readpcx(FILE *file, unsigned short int *length, unsigned short int *height)
     /* Returns NULL if failed, otherwise a pointer to the loaded image */
{
   PCX_Header header;
   void *target;
   fseek(file, 0, SEEK_SET);
   fread(&header, sizeof(PCX_Header), 1, file);  /* Reads the PCX header */
   if((header.signature!=0x0a)||(header.version!=5))  /* Checks if file is in PCX format
*/
      return(NULL);
   else
   {/* it is! */
      /* Return height and length */
      *length=header.xmax+1-header.xmin;
      *height=header.ymax+1-header.ymin;
      /* Allocate the sprite buffer */
      target=(void *)malloc((*length)*(*height));
      /* Read the image */
      readpcximage(file,target,(*length)*(*height));
      /* PCX succesfully read! */
      return(target);
   }
}

void createDataFiles(char * target, unsigned int length, unsigned int height, char *
output)
{
   FILE * stats;
   int x,y;
   float sig, sig_sqrd;

   unsigned char buf;
   if ((stats=fopen(output, "w"))==NULL )  // Opens data file for output
   {
      printf("Error!  Could not open file(s)!");
   } else
   {
      for(x = 0; x < length; x++)
      {
         sig = 0.0;
         sig_sqrd = 0.0;
         for(y=0; y < height; y++)
         {
```

```
            buf=target[(y*length)+x];
            sig+= (float)(int)buf;                      // Calculates the variance
            sig_sqrd += (float)(((int)buf)*((int)buf));  // Calculates the variance^2
         }
         sig/=(float)height;
         sig_sqrd/=(float)height;
         fprintf(stats, "%d %f %f\n", x+1, sig, (sig_sqrd - sig*sig));
      }
      fclose(stats);
   }
}

int main(int argc, char ** argv)
{
   FILE * file;
   void * image;  /* 8 bit image as read from the pcx */
   unsigned short int length, height;

   if (argc!=3)     /* Checks for invalid invocation */
   {
      printf("Usage: hobbit filename.pcx outputname\n");
      return(1);
   }
   if ((file=fopen(argv[1],"r"))==NULL)   /* Checks to see if file is readable */
   {
      printf("Cannot open file!\n");
      return(1);
   }
   if ((image=readpcx(file,&length,&height))==NULL)  /* Checks if file can be loaded */
   {
      printf("Error loading file!");
      return(1);
   }
   fclose(file);    /* Closes file once image is loaded into memeory */
   printf("Image Dimensions: %d by %d.\n",length, height);

   createDataFiles(image, length, height, argv[2]);
   free(image);
   return(0);
}
```

This program acts as the first level processor for any image to be run through Gandalf.c++. It computes an average brightness value for each wavelength (column of pixels) and outputs it in a text file to be used as input by Gandalf.c++.

VITA

Casey Patrick Deen
1637 Merrimac Trail
Garland, TX 75043
casey_deen@excite.com

Casey Deen grew up in Garland, Texas, a suburb of Dallas, where he attended Garland High School. Thanks to a wonderful physics professor, Mr. Richard Lines, Casey became intrigued with physics and decided to pursue it as a major in college. He currently attends Texas A&M University and will graduate in December 2004 with a Bachelors of Science in Physics. Afterwards, he plans to go to graduate school to pursue a master's degree in Astronomy or Astrophysics.

Poster Presentations:

- Pathways to the Doctorate Research Symposium, Texas A&M University System, TAMU at Galveston, November 16th, 2003.
- Student Research Week Symposium, Texas A&M University, TAMU College Station, April 2004.