CS 258 Spring 2011
Assignment 2
TIFF Reader/Writer

January 27, 2011

1 Purpose

In this assignment you will constuct a module to the CLI of the last project. You will implement a TIFF reader and writer that you will use to display and create images.

2 Due Date

You must have turned your assignment in by midnight on February 17, 2011.

3 TIFF Reader/Writer

In this part of the assignment you will write a module (or commands) that do the following:

Command	Parameters	Description
TiffRead	f_n	read a tiff file with file f_n
TiffStat	f_n	print the parameters of f_n
TiffWrite	f_n, x_0, y_0, x_c, y_c	write a tiff file from box.

The TiffRead command simply takes a tiff file with filename f_n and displays it in a window. Use the *checkimage* array in the supplied *main.c* to do this. You are not expected to be able to read in *all* tiff files, see below.

The TiffWrite command writes a tiff file with file name f_n and fills it with pixels formed by the box (x_0, y_0, x_c, y_c) of the last image read in by TiffRead. The box is specified by the pixel coordinates of the lower left hand vertex (the origin) and the upper right hand vertex (the corner) of the box. Note that TiffWrite can only work after TiffRead has been called as the file type of what is written must be same as what was read. If you read in an 8 bit grayscale image, you must write an 8 bit grayscale image.

The TiffStat command prints out the tag entries and values of the tiff file f_n. If the file is not a tiff file, it informs you of this. Otherwise it prints out a summary of the tiff tags stored in the file. Of course, don't print out the image or the color map.

3.1 The TIFF Spec

An abbreviated version of the Tiff 6.0 spec has been copied and handed out in class. The complete specification is available as a PDF on Oak and we went over it in class.

The Tiff document is large and intimidating. But it's not my intent to frighten you. Be bold, open the document and start reading through. You'll find that it's not that bad.

The process of reading a spec carefully and writing software to conform to that spec may be new to you. Even if it's the first time you've done this, I guarantee it won't be the last. A surprising large fraction of one's professional duty is taken up by this sort of activity.

3.2 Requirements

The TiffStat command should be able to stat any of the images supported by the baseline Tiff spec and print out their associated Tags and values, if the values are stored in the offset. If the values are not stored in the offset, then you can simply print out the offset.

For reading and writing, however, I only require you to implement the specification for two types of images: 8-bit grayscale uncompressed, and 24-bit RGB uncompressed.

The code already given in class can display your results, if the data is loaded into the appropriate global array. We will supply a test suite of images. You must accomplish the following:

- TiffStat all the images.
- TiffRead all the images which are 8-bit grayscale uncompressed and 24-bit RGB uncompressed.
- TiffWrite an image of your own choosing.
- TiffRead your own written images.

4 What to Turn In

Turn in the source of your program, along with instructions (or a Makefile) to compile it. Also, turn in a few scripts for the CLI which exercise it and the tiff-related commands. Supply a few images along with this: a grayscale image, an rgb image, an image your program won't read but can stat, etc.

4.1 Suggestions

When testing, test with small images. You are not required to support an image greater than 1024x1024 if you don't want to (but your program should gracefully decline to handle a larger image). (0,0) for OpenGL (and Direct3D) is in the lower left hand corner of the window.

Note that the Tiff specification calls for tags to be in ascending numerical order. Also note that your reader should handle the reading in of many different flags (which is may then ignore, depending on their type), but you should only write out the required Tiff fields to make your life easier.

Write the TiffStat command first—the hardest part of this assignment is parsing the IFD. When using TiffWrite, you only need to write out the *required* fields; you can ignore any optional fields that were associated with the file that you read in. Note also that baseline TIFF readers (the kind described in this project) are not required to handle more than one IFD per file, although the Tiff specification requires that the IFD contain the address of the "next IFD" as its last 4 bytes. You can make this a long zero.

The format of TIFF image data is described on pages 37 and 38 of the TIFF spec, under PhotometricInterpretation and PlanarConfiguration. Note that a baseline TIFF reader only has to support PlanarConfiguration=1. If SamplesPerPixel is greater than three, you can throw away the extra bytes or you can generate a diagnostic message and do nothing.

I have provided a set of test images in the 258 distribution in the `images` directory; there is also a `pooh_le.tif` on OAK that is equivalent to pooh.tif but in little-endian format. Some of these images have associated `.tifstat` files that contain the IFD entries for the files, so that you can check your program.

4.1.1 Getting Started

One of the difficulties with this assignment is that it's hard to get started, i.e., to figure out how to get the data from a binary file into a form that you can use. Here is a sample program that parses a simple IFD for one single image. The following program assumes you are running on an Intel-based (little endian) machine, i.e., it assumes that the endian-ness of the underlying architecture and endianness of the TIFF image *match*. So, it does not check the first two bytes of the TIFF file and switch all two and four-byte quantities after that, *as your program should do.* Try this program out, and check its answers with `pooh_le.tifstat`. I think if you understand the code in it then you'll see how to parse an image. Note that code to check the endian-ness of a machine was provided in class.

```
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  char buffer[3];
  short magicno;  // 2 byte quantity
  int   ifdaddress; // 4 byte quantity
  short ifdcount;   // 2 byte quantity

  ifstream imfile;
  imfile.open ("pooh_le.tif", ios::binary );

  imfile.seekg (0, ios::beg); //not strictly necessary, but a good idea

  imfile.read (buffer,2);
  imfile.read ((char *)&magicno,2); //MAKE SURE YOU UNDERSTAND WHY THIS WORKS!
  imfile.read ((char *)&ifdaddress,4);

  imfile.seekg(ifdaddress,ios::beg);
  imfile.read((char *)&ifdcount,2);

  imfile.close();

  buffer[2]='\0'; //Necessary because buffer is a C-style string
  cout << "Endian: " << buffer << "\n";
  cout << "Magic: " << magicno << "\n";
  cout << "IFD Address: " << ifdaddress << "\n";
  cout << "IFD Count: " << ifdcount << "\n";

  return 0;
}
```

Here's how this should work:

```
& g++ -o mytest test.cpp
& ./mytest
Endian: II
Magic: 42
IFD Address: 131936
IFD Count: 17
```