Using FreeImage 3.11 for Image Output

Niels Joubert, CS184 TA, UC Berkeley

2008-09-29

FreeImage is an input/output library written in C, compatible with Linux, OSX and Windows. You need to download the latest source from `http://freeimage.sourceforge.net/`.

WARNING: FreeImage has one big caveat for projects at Berkeley. It is not on the instructional machines, thus you need to include the compiled library with your source code when you submit, and make sure it runs on the instructional machines. [1]

1 Getting FreeImage and linking it to your source code.

- Extract the source zip into a folder of your choice.
- Find the library files (an .a and file for Linux or Mac, a .dll file for Windows) in the extracted directory, and the `FreeImage.h` header file in `./Source` directory. You can compile the source manually or just use the supplied library files.
- Copy the `.a` and `.h` files to your root project folder. (For windows, see section 1.1.2)

1.1 Linking against FreeImage (and libraries in general)

1.1.1 With GCC - as on Mac and Linux

Let me preface this by saying that I don't enjoy messing with linker issues. Its painful and incredibly frustrating, and I really hope this document will help everyone avoid such issues! The g++ compiler has the following flags that tells the linker about files of interest:[2] (Note the naming convention for libraries)[3]

```
-I<directory>   Add <directory> to the list of directories searched through for includes
                This directory is searched through for .h files
                eg: -I./

-L<directory>   Add <directory> to the list of directories searched through for libraries.
                This directory is searched through for object code files of type .a, .so, etc
                eg: -L./

-l<library>     Links the given library object code into your executable.
                Takes in the real name (eg. "freeimage" for "libfreeimage.a")
                eg: -lfreeimage
```

[1]As an alternative, ImageMagick is installed on all instructional machines.

[2]For this example, your source code is in "./", your FreeImage.h is in "./" and your libfreeimage.a is in "./", in other words, all relevant files are in the root of your project folder.

[3]For more information on the naming of libraries, see http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html

We compile and link C++ source code using the G++ (or GCC, really the same thing) compiler. Your favorite IDE (eclipse, xcode, kdevelop, textmate, emacs, vim, etc) most probably calls out to g++ to compile your code, so it's worth having some knowledge of what's going on underneath the hood.

Note: In section 2 there is example code you can grab to try out compiling and linking.

To **compile** your source code (and your `FreeImage.h` is not in the same directory as your source code), you need to supply g++ with the -I flag and the path to the directory containing `FreeImage.h`. This is unnecessary in our case, where `FreeImage.h` is in the same folder as your source, but let's show an example for completeness:

```
g++ -c -Wall -I./ test.cpp -o test.o
```

To **link** your compiled object files into one executable file, you need to supply g++ with the -L flag with a path to the directory containing your `FreeImage.a` file, and the -l flag with the name of the library, which is "freeimage" in our case.

```
g++ -L./ -lfreeimage test.o -o test
```

You can combine the linking and compiling steps, which is fine if you don't have several source .cpp and .h files. You can **compile and link** (assuming all your files is in the same directory) using:

```
g++ -Wall -I./ -L./ -lfreeimage test.cpp -o test
```

If you have several source files, the best approach is to do these steps separately, which you can automate with a Makefile, as section 4 will demonstrate.

1.1.2 With Visual Studio C++ on Windows

Thanks to Daniel Ritchie for contributing this section of the notes!

- Make sure you have the following three files: `FreeImage.h`, `FreeImage.lib`, and `FreeImage.dll`. These should all be included in the .zip file you download from the FreeImage site.
- You can put these files anywhere as long as you tell Visual Studio where to look for them, but I like the following setup for it's simplicity:

  1. Create a folder in your Project Directory (you typically have a solution directory, inside of which you have one or more project directories) called `FreeImage`. Put the `.h` and the `.lib` files in here.
  2. Put the `.dll` file directly into your Project Directory (it should be sitting alongside files such as your `.vcproj` file).
  3. In the Visual Studio Solution Explorer (It's a tree-view widget that's typically docked on the far-right or far-left of the screen. If it's not there, you can get to it via `View->Solution Explorer` from the top menu), right click your project (it should be located under your Solution) and select `Properties`.
  4. Expand "Configuration Properties", then "C/C++", and then select "General". Enter `"$(ProjectDir)/FreeImage"` into the field titled "Additional Include Directories". This tells Visual Studio to look in this directory for any `.h` files that it can't find in the standard locations.
  5. Now expand "Linker" under "Configuration Properties". In the General page, enter `"$(ProjectDir)/FreeImage"` into the field titled "Additional Library Directories" (again, this tells Studio to look here for `.lib` files).
  6. In `Linker->Input`, add `"FreeImage.lib"` to the "Additional Dependencies" field.

- You're done! Click OK and build your project.

NOTE: You might have noticed that I didn't say anything about FreeImage.dll beyond putting it in your Project Directory. It's really important that you have it there, though–your project will still build without it, but you won't be able to run the program without it.

2 Including and using FreeImage in your source code.

You can use the following example code to test your linking and compiling stages. Paste this into your editor of choice, and compile it with the above examples.

Listing 1: test.cpp : Simple FreeImage test code

```
#include <iostream>
#include "FreeImage.h"

using namespace std;

int main(int argc, char *argv[]) {
        FreeImage_Initialise();
        cout << "FreeImage " << FreeImage_GetVersion() << "\n";
        cout << FreeImage_GetCopyrightMessage() << "\n\n";
        FreeImage_DeInitialise();
}
```

If everything went well, you should see the version and copyright message of FreeImage when you try to run the created test executable. Notice how each FreeImage function starts with `FreeImage_` to make it easily identifiable. Neato!

2.1 Creating a bitmap and setting its pixels

We will use FreeImage to store bitmap data - color data for each pixel. FreeImage bitmaps have their origin `(0,0)` at the bottom left corner of the image, which you need to take into account when you're iterating over your viewport.

To create a bitmap, use the `FreeImage_Allocate` API call:

```
    FIBITMAP * bitmap = FreeImage_Allocate(WIDTH, HEIGHT, BitsPerPixel);
```

Where Width and Height is in pixels, and we use 24 bits per pixel. This function call returns a pointer to an FIBITMAP struct. You don't need to worry if you're not comfortable with pointers though, since all you have to do it pass the "bitmap" object to other functions.

2.2 Saving image

We now convert the internal bitmap image representation into a compact image format (TIFF, PNG or JPEG) on disk. I recommend using a lossless format (PNG or TIFF can both be lossless) since you would rather not have the quality of your raytraced images be degraded by lossy compression.

To save an image use the `FreeImage_Save` API call:

```
    FreeImage_Save(FIF_PNG, bitmap, "filename.png", 0)
```

For other formats and specific flags, see page 19 of the API documentation mentioned in section 2.3.

2.3 API documentation

Get the full API documentation in PDF format from `http://freeimage.sourceforge.net/download.html` or read the header file.

2.4 Complete Example Code

Listing 2: Everything you need for pixel-level PNG output

```
#include <iostream>
#include "FreeImage.h"

#define WIDTH 800
#define HEIGHT 600
#define BPP 24  //Since we're outputting three 8 bit RGB values

using namespace std;

int main(int argc, char *argv[]) {
        FreeImage_Initialise();

        FIBITMAP* bitmap = FreeImage_Allocate(WIDTH, HEIGHT, BPP);
        RGBQUAD color;

        if (!bitmap)
                exit(1); //WTF?! We can't even allocate images? Die!

        //Draws a gradient from blue to green:
        for (int i=0; i<WIDTH; i++) {
                for (int j=0; j<HEIGHT; j++) {
                        color.rgbRed = 0;
                        color.rgbGreen = (double)i / WIDTH * 255.0;
                        color.rgbBlue = (double)j / HEIGHT * 255.0;
                        FreeImage_SetPixelColor(bitmap,i,j,&color);
                        //Notice how we're calling the & operator on "color"
                        //so that we can pass a pointer to the color struct.
                }
        }

        if (FreeImage_Save(FIF_PNG, bitmap, "test.png", 0))
                cout << "Image successfully saved!" << endl;

        FreeImage_DeInitialise(); //Cleanup!
}
```

3 Submitting a project that uses FreeImage

You need to include FreeImage's library files in your submission since it is not installed on the Instructional Machines.

Windows: Include the .dll and .h files in your Visual Studio Solution submission, and confirm that it compiles on one of the instructional machines.

Linux and Mac: Make sure you include the library files (specifically, the `.a` file) relevant to the platform you're submitting on. You need to compile your own library on the Solaris machines to generate a `libfreeimage.a` file for your submission, which you can easily do using `make`. Confirm that your submission runs on the instructional Solaris machines (for Linux) or instructional Mac (although it should be just as easy to confirm that it works on Solaris as well). Feel free to ask me for help!

4 ADDENDUM: Example Makefile

Makefiles are great, and it is unfortunate that Berkeley students aren't exposed to them more. We use Makefiles to build source code from many separate .h and .cpp files, to reference libraries such as GLUT and FreeImage that our code depends on, and to speed up compiling by doing *partial compiles*[4].

Listing 3: Sample Makefile

```
#Basic Stuff ------------------------------------
CC          = g++ -g -Wall -O2 -fmessage-length=0

#Libraries --------------------------------------
CCOPTS = -c  #I don't need -I since everything is in the current directory
LDOPTS = -L./ -lfreeimage

#Final Files and Intermediate .o files-----
SOURCES = raytracer.cpp #ADD YOUR SOURCES
OBJECTS = raytracer.o   #ADD YOUR .o FILES
TARGET = raytracer

#-----------------------------------------------
raytracer: $(OBJECTS)
	$(CC) $(LDOPTS)  $(OBJECTS) -o $(TARGET)

raytracer.o:
	$(CC) $(CCOPTS) $(SOURCES)

default: $(TARGET)

clean:
	/bin/rm -f *.o $(TARGETS)
```

Run it by typing "`make`" at the command line in the root of your project directory.

[4]"partial compiles" recompiles only the files that changed since the last compilation, thus saving compile time.