

Re: working with bitmaps in C

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2007-12/msg04015.html

- *From:* Flash Gordon <spam@xxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Thu, 27 Dec 2007 20:22:36 +0000
-

Malcolm McLean wrote, On 27/12/07 17:16:

"Ernie Wright" <erniew@xxxxxxxxxxxxx> wrote in message
news:mIWdnWEh49T1W-7anZ2dnUVZ_gCdnZ2d@xxxxxxxxxxxxxxxxx

Malcolm McLean wrote:

"Ernie Wright" <erniew@xxxxxxxxxxxxx> wrote in message

Broken for 24-bit and 32-bit, and for 4-bit
old-style. See below.

No, it's been tested and basically works. It might not work on
everything, but it will load the vast majority of BMPs OK.

Tested in what way?

If you `loadbmp()`, then `savebmp()` a 24-bit BMP and then examine the
output of `savebmp()`, you'll find that the red and blue channels have
been swapped. After seeing the relevant problem in the source code, I
actually compiled and tested it to confirm that I hadn't missed
something. I wasn't guessing.

Your code to load 24-bit pixels (with indention repaired) is

```
case 24:
for(i=0;i<bmpheader.height;i++)
{
for(ii=0;ii<bmpheader.width;ii++)
{
target = (i * bmpheader.width * 3) + ii * 3;
answer[target] = fgetc(fp);
answer[target+1] = fgetc(fp);
answer[target+2] = fgetc(fp);
```

Re: working with bitmaps in C

You can see that this stores pixels in `answer[]` in the same BGR order as they're written in the BMP.

You're right.

The file is a different version to the one published in the book, which has it the right way round. Obviously something has gone wrong with my versioning.

The specification claims that 4-bit old-style BMP has a 16-color palette. It's been so long since I've encountered one of these that I wouldn't know where to look for one now. Unless you know of users that actually have to deal with these (they haven't been written by Microsoft code since Windows 2.x), I'd be tempted to strip all of that cruft out of your loader.

The version of Paint in Windows Vista claims to be able to save a 16 colour bitmap. I've no idea if the format is the same, but I see no reason why it would not be.

It is a problem testing when you don't have versions of the input.

You create the test data. Creating test images is not difficult seeing as you are using an OS that comes with a program able to save in the relevant format.

Also if degenerate files are in circulation. You get this with IFF files quite a bit – in practise the best thing is to search for the 4-byte tag sequence. The format says field sizes should be included, and you skip to the next tag, but actually you are much more likely to get a corrupt field size than a chance occurrence of the tag.

Actually, the best thing is normally to start off by checking the length is within a valid range, then check the data is valid, then check that you hit a valid tag in the place expected. If any of those things fail you report the file as being corrupt or a format you can't handle.

I've done a moderate amount of work on serial links where there was real risk of data corruption (guaranteed to get some within 1KB of data) so I know what works in the real world.

The other potential problem was your code's handling of header sizes that weren't one of the two it expects. You're most likely to run into this if a user tries to load an OS/2 BMP, or a file that's not a BMP at all but happens to start with the letters 'BM', but it's also a forward compatibility issue, in the event a new version of BMP comes along with a different header size.

Re: working with bitmaps in C

That's a good point. It should be returning -1 instead of zero on an unrecognised header size, at least. Probably we should try to read it as a 40. However to do a really good job we've got to skip to the raster bits, which means a total rewrite and lots of complications.

Don't advertise your code as being reliable then, advertise it as supporting some but not all BMP files.

—

Flash Gordon

.