Subject: Re: 2d filters and large images (continued from ASSOC)
Posted by Peter Mason on Thu, 27 May 2004 00:53:04 GMT
View Forum Message <> Reply to Message

Jonathan Greenberg wrote:
> So I probably should have asked a more general question (thank you
> for the feedback on ASSOC, by the way). I was hoping to get some
> feedback on how to apply 2-d filters of various sizes to a large 2-d
> image (too big to load the entire file into memory) -- which ways
> would work the best? Right now, if my filter is, say, 5 x 5, I grab
> 5 lines of the image and run through the center pixels (row 3 in the
> subset) and apply the filter, then grab the next line and create a
> new 5 line x number of samples chunk.
>
> Are there better/quicker/easier ways of applying filter like this?

Hi again Jonathan,

Sounds nasty. Here are a few ideas. (I haven't been faced with a problem
quite like this so these ideas will be somewhat abstract.)

As you are grinding through the image sequentially, I would say that you'd
get the best performance out of plain old READU (and WRITEU - you are
writing a separate output, right?). A little better than ASSOC (which
always does file-pointer manipulation) and maybe even better than proper
memory mapping (which will page-fault when it feels the need and might end
up bloating your working-set size).

The immediate inclination is to maintain a 5-line buffer for your image
data, shifting lines up by 1 and plastering the new line into the bottom of
the buffer as you go through the image. But that's a lot of memory
manipulation and it's certainly not going to go unnoticed.
A faster approach is to maintain a cyclic image buffer. (No shifting.
The new-line insertion index cycles round and round, and the current line
index tracks behind it accordingly.) Here you do the line-shifting on the
filter - a much quicker task as the filter is small.

Another idea is to use proper memory mapping, mapping only 5 image lines at
a time and bumping the mapping's offset by one line's worth to step through
the image. The beauty of this is that there's no shifting involved at all,
and *hopefully* the OS's caching will be smart enough to avoid repeated
reads. The downside is the time taken by the OS to "bump the mapping's
offset". With IDL's implementation I think that you have to close the old
mapping entirely and open a new one in order to achieve this. I don't have
a feel for the performance consequences here. If you are working on a
Win32 platform though... A year or so ago I wrote a memory-mapping suite
for IDL on Win32. It was made pretty much obsolete by IDL's memory-mapping

calls (which debuted not long afterwards) but it still has a trick or two up its crusty little sleeves. Of interest here is that you can change a mapping (bump the offset in this case) without having to close the thing entirely. This may give you a performance edge here. If you would like to try this suite, it's called "Stoneface" and it's on the IDL user-contrib site under DLMs.
