Subject: Re: Optimized storage of 1BPP images
Posted by mdelfede on Thu, 07 Feb 2008 20:19:32 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Thu, 07 February 2008 17:131728x2210x50 pages in RGBA is whopping 700MB, that's bad.

very bad

Quote:
But, the tiff itself is only 1BPP? So in such case the file has approximately 23MB only! (no problem to cache it)

If you manage to produce (rescale) final viewport from these raw data, you need less than 10MB for viewport (2000x1200 pixels) content in RGBA. (that's 23 + 10 = ~33MB so far .. that's nothing for modern PC, but of course there will be additional memory consumption caused by OS itself and underlaying technologies like U++)

No, I agree... the solution would be to work directly on 1BPP image, without having to convert it in RGBA. Then buffering would not be a problem.

Quote:
So if you prefer to rescale the viewport out from RGBA data (may save you some coding of your own scale routines for 1BPP -> RGBA scaling), you may still unpack+cache only 2-3 pages from the raw tiff, and work with those. Predict when new page will be approximately needed, and unpack in background thread soon enough into cache the next page.

My problem is that rescaling stuff are provided (AFAIK) only for RGBA images. So, if I don't want to write brand-new rescaling routines aimed to 1BPP images, I have to go through RGBA.
Maybe some sort of 'banding' would solve the problem, so rescale the image in strips and buffer them as 1BPP ones... the conversion from RGBA to 1BPP would be trivial in this case.
So, load a strip, convert to RGBA, rescale, reconvert to 1BPP and store it ob buffer. When displaying, revonvert stripwise to RGBA and draw it. Of course that means double conversion, one of which must be done realtime, but it should be trival and fast conversion.

Quote:
....
But you will get somewhat complex caching/unpacking/rescaling code. I don't think the complexity is beyond single focused developer, but depends how much time you can spend on this application development.

Not too much time, indeed
But the problem is intriguing.... Maybe I'll spend a bit more time trying to optimize it....

Quote:
The only performance-wise problematic place IMO is the on-the-fly production of viewport data. This can choke a bit on slower CPUs with very large window area. Than again if you will cache

even this during scrolling, so you will scale during scrolling only newly visible area, the slow few lines per sec scrolling can be done already on 1GHz CPU.
If somebody want to go trough pages fast with PageDown, you will have to decode the 1BPP pages on the fly ... still if I would get 0.1 - 0.3s response on (many) PageDown hits, I would be not happy, nor sad. Depends how your users use this viewer most often.
If they go all around the page slowly in high zoom, it should work very smoothly. If they go very fast down, it will hurt.

I think the real bottleneck is the rescaling on the fly of solution 1, or the huge buffering on solution 2.
Rescaling once the full image and buffering it as 1BPP packed stuff should do the trick... The program is quite usable now rescaling the viewport, so having "only" to unpack it would be much fast.
Quote:
Than again, the KPDF on my old 1GHz Athlon XP is pain in *ss to go trough 30-50 pages PDF with page down, often it takes several seconds to show content of current page, if I go down too fast beyond the pre-cached pages. (than again, PDF is much more complex thing than raw 1BPP image)

Hmmm... PDF is complex, but is (AFAIK) mostly vectorial format (besides of some scanned-pdf-ized stuffs), so repainting it can be a lot faster. Problem with images is not complexity but the memory footprint...

Ciao

Max