Subject: Optimized storage of 1BPP images
Posted by mdelfede on Thu, 07 Feb 2008 12:30:52 GMT
View Forum Message <> Reply to Message

I made a Tiff viewer control, with thumbnails and contiuous page scolling, in order to view Tiff images that are all 1 bit per pixel ones (black and whte fax).

I tried in 2 ways :

1- As explained here :

http://www.ultimatepp.org/forum/index.php?t=msg&th=3131& amp;start=0&

the image is rescaled on the fly zooming, panning and/or resizing the widget. The performance is not bad at all, but being the image very big the user feels like the scroll is a bit coarse.

2- The same but caching rescaled images when first displaying them. Of course, I must re-cache them on zooming. That works great, the scroll is very smooth but the cached images get *very* big in memory, with high zoom factors (I must cache the whole image because of pan/scroll requirements). In addition, when the zoom factor is high, there's a noticeable delay when rescaling/buffering the whole image.

As the images are just 1-bit-per-pixel, I think is nonsense to store them in RGBA format; it would be much better to have a packed 1BPP format... that would solve all problems.
Is there an (easy!) way to get image through ImageEncoder that are so packed, and of course a way to draw those packed images
directly on a Draw object ?

Ciao

Max

Subject: Re: Optimized storage of 1BPP images
Posted by cbpporter on Thu, 07 Feb 2008 12:47:49 GMT
View Forum Message <> Reply to Message

mdelfede wrote on Thu, 07 February 2008 14:30
1- As explained here :

http://www.ultimatepp.org/forum/index.php?t=msg&th=3131& amp; amp;start=0&

the image is rescaled on the fly zooming, panning and/or resizing the widget. The performance is not bad at all, but being the image very big the user feels like the scroll is a bit coarse.

I had a similar problem with my image scrolling control. Performance was reasonablly good, but when scrolling, it felt like some weight was holding you back and it felt plain wrong to use.

I fixed this by creating a permanent preallocated image, the size of the screen, and using it as a back buffer. When zooming for example, you only draw a zone zoomed on the backbuffer, and simply draw a portion as large as the parent control from the backbuffer. Not having to create Image objects on each redraw gives great performance boost. I don't know if this applies to you needs.

About number 2, I don't really know. U++ Images are in RGBA format, and it would be quite some effort to add support for other formats too. With WinAPI (if under win), you could easily put together a 1 bit bitmap, and on Paint, get the HDC and blit it, but of course this is not platform independent. Under X I think that working with image buffers is a little more natural, but I have no details.

Subject: Re: Optimized storage of 1BPP images
Posted by mr_ped on Thu, 07 Feb 2008 12:54:35 GMT
View Forum Message <> Reply to Message

There's third way too.

With zooms in range 0..100% the new scaled image is always smaller/equal to original picture, so the memory consumptions is under O(2). I don't think this case needs further improvements, and I would keep it as it is. (actually this method is probably optimal even for zooms like 100-200%)

When the zoom is > 100%, you can try different approach, not to scale whole image, but only the current viewport of it.
In case even this is too slow, you may during panning use the previous viewport from cache, and rescale only the parts which are newly on screen.

I'm not sure how large your viewport is, but anything under 1000x1000 pixels should be well manageable by GHz CPUs, if you have the original picture in RAM.

In this way your rescaled image for >100% zooms will have fixed size, and will not grow further with bigger zooms, also with bigger zooms you will use smaller and smaller part of original picture to create the rescaled one, so the performance will grow.

Of course this is pure theory, maybe it would not perform as well as I think in real world, but I have some experience with real time graphics, and unless your view has like 2000x1000 pixels, the modern CPU + RAM should easily rescale such amount of data on the fly.

edit: cbpporter suggested pretty much the same thing I see.

Subject: Re: Optimized storage of 1BPP images
Posted by mdelfede on Thu, 07 Feb 2008 13:22:59 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Thu, 07 February 2008 13:47
I had a similar problem with my image scrolling control. Performance was reasonablly good, but when scrolling, it felt like some weight was holding you back and it felt plain wrong to use.

well, it's not so strong as you tell here bu, yes, the feeling is of something holding back

Quote:
I fixed this by creating a permanent preallocated image, the size of the screen, and using it as a back buffer. When zooming for example, you only draw a zone zoomed on the backbuffer, and simply draw a portion as large as the parent control from the backbuffer.

That can't go for my case I have *very* large images (fax tiffs, thousand's of pixels wide...). With point 2 on previous post I did something like you, just buffering the whole rescaled image. The big problem is that, having a multipage continuous scrolling (if you know evince program in Linux, the same stuff...) I must buffer the whole file in order to have smooth scroll. That works great with small scaled images (but then, it works also without buffering...); the performance on 100% scaled images is not bad at all, but the memory footprint starts to be very high... scaling to 150% it's no more acceptable, being better the performance with rescale-on-the-fly method.

Quote:
About number 2, I don't really know. U++ Images are in RGBA format, and it would be quite some effort to add support for other formats too.

I think so. The best would be bypass completely 'Image' format and to go directly from raster to a packed image format. But then I have to add this format to Rescale() function too, and to Draw() class too.... quite a big work.

Ciao

Max

Subject: Re: Optimized storage of 1BPP images
Posted by mdelfede on Thu, 07 Feb 2008 13:32:03 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Thu, 07 February 2008 13:54There's third way too.

With zooms in range 0..100% the new scaled image is always smaller/equal to original picture, so the memory consumptions is under O(2). I don't think this case needs further improvements, and I would keep it as it is. (actually this method is probably optimal even for zooms like 100-200%)

When the zoom is > 100%, you can try different approach, not to scale whole image, but only the current viewport of it.
In case even this is too slow, you may during panning use the previous viewport from cache, and rescale only the parts which are newly on screen.

I'm not sure how large your viewport is, but anything under 1000x1000 pixels should be well manageable by GHz CPUs, if you have the original picture in RAM.

Images are 1728 x 2210 pixels per page, with often tenths of pages... in RGBA mode they're really too big to buffer.
Quote:

In this way your rescaled image for >100% zooms will have fixed size, and will not grow further with bigger zooms, also with bigger zooms you will use smaller and smaller part of original picture to create the rescaled one, so the performance will grow.

I guess it's not so simple. When I do an on-the-fly rescale, it works like you say, so the performance is limited only by control's width, at whorse. But if I do buffering, I need to scroll/pan over all image, so I must buffer the whole. Even whorse, having continuous scroll between pages, I must buffer whole file.

Quote:
..... and unless your view has like 2000x1000 pixels, the modern CPU + RAM should easily rescale such amount of data on the fly.

that's the problem

I guess I'll take out the buffering stuff and follow as before, keeping the slight feeling of bumpy scroll. All other ways seems to me whorse or too time expensive to code....

Thanx to all for answers !

Ciao

Max

Subject: Re: Optimized storage of 1BPP images
Posted by cbpporter on Thu, 07 Feb 2008 14:16:41 GMT
View Forum Message <> Reply to Message

Quote:
Images are 1728 x 2210 pixels per page, with often tenths of pages... in RGBA mode they're really too big to buffer.

In this case, it would be about 14MB of RAM. Not that much for graphics application. Plus with the method I proposed, you would need another maximum 10 MB for huge resolutions.

Anyway, loading the whole image in RAM is anyway unavoidable IMO, so I would concentrate on performance. In my control, with a zoom level of maximum 800% and huge vectorial images which must be rendered to a buffer, the image size can quickly get out of hand, so I really don't mind 25MB of used RAM.

Subject: Re: Optimized storage of 1BPP images
Posted by mr_ped on Thu, 07 Feb 2008 16:13:50 GMT
View Forum Message <> Reply to Message

1728x2210x50 pages in RGBA is whopping 700MB, that's bad.

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++)

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.
This will be additional 15MB per cached RGBA page. (and I think cache of size 3 or 4 is plenty unless somebody want to scroll really really fast trough all pages) 4*15 = 60; 60 + 33 = 93MB. Still well under 100MB for core application data. Doesn't sound bad to me.

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.
I believe to read the new page from disk is question only couple of tenths of second, the unpack into RGBA original size picture is under 0.2s for sure and all of this could be fitted into background thread to be finished before you will need that page first time.

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.

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)

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

Subject: Re: Optimized storage of 1BPP images
Posted by mirek on Thu, 07 Feb 2008 22:22:01 GMT
View Forum Message <> Reply to Message

The solution to the problem might be "MemoryRaster".

Mirek
