Subject: Displaying raw image data
Posted by keltor on Mon, 22 Apr 2013 15:46:13 GMT
View Forum Message <> Reply to Message

Hey everyone,

Silly me, I started to think that I was getting more fluent with U++, yet here I am in the newbie subforum once again.

I have a library that draws some stuff in memory. The drawing can be accessed using something like obj->GetRGBA(); however it is raw data, namely an unsigned char * consisting of 4 bytes for each pixel.

The library does its magic and creates the raw data. I would like to draw it into a bitmap, ideally as a picture in an ImageCtrl. However, I am a little lost with Buffer, Raster and the like. I couldn't find any suitable documentation or examples, so here I am.

My goal would be to reuse the raw buffer as the data to plug directly into ImageCtrl, so that if eventually the data in the buffer changes, one can force a redraw without having to copy data. But I don't know it that can be achieved, much less how. So if someone with more experience with this kind of things can help me out, he would certainly have my gratitude.

To clarify a bit more the kind of data I'm dealing with: bytes 0 to 3 contain the RGBA info for point (0,0); bytes 4-7 for (1,0); and point (i,j) is at 4*(i+width*j). The buffer itself has no picture height/width information, but that can be obtained elsewhere.

Thanks for reading

Keltor

Subject: Re: Displaying raw image data
Posted by dolik.rce on Mon, 22 Apr 2013 16:59:02 GMT
View Forum Message <> Reply to Message

Hi Keltor

keltor wrote on Mon, 22 April 2013 17:46Silly me, I started to think that I was getting more fluent with U++, yet here I am in the newbie subforum once again. Happens to me all the time

keltor wrote on Mon, 22 April 2013 17:46I have a library that draws some stuff in memory. The drawing can be accessed using something like obj->GetRGBA(); however it is raw data, namely an unsigned char * consisting of 4 bytes for each pixel.

The library does its magic and creates the raw data. I would like to draw it into a bitmap, ideally as a picture in an ImageCtrl. However, I am a little lost with Buffer, Raster and the like. I couldn't find any suitable documentation or examples, so here I am.

My goal would be to reuse the raw buffer as the data to plug directly into ImageCtrl, so that if eventually the data in the buffer changes, one can force a redraw without having to copy data. But I don't know it that can be achieved, much less how. So if someone with more experience with this kind of things can help me out, he would certainly have my gratitude.
Some libraries let you do the opposite, that is to tell them in which part of memory should they draw. If that is your case, than you can simply do (assuming it uses same byte order) something like Image img;
ImageBuffer ib(width, height);
obj->setTargetMem(ib.Begin());
obj->doSomething();
img = ib;

If not, then the easiest way is AFAIK to do the copy, it should be very quick if you just memcpy the entire thing (again, same byte order expected):
Image img;
ImageBuffer ib(width, height);
memcpy(ib.Begin(), obj->getRGBA(), 4*width*height);
img = ib;

Creating ImageBuffer or Image using your pre-allocated piece of memory is not possible, IIRC. I think it would be possible to add such feature, but I'm not sure if it really corresponds with the "U++ way" of handling memory

Best regards,
Honza

Subject: Re: Displaying raw image data
Posted by keltor on Mon, 22 Apr 2013 22:03:20 GMT
View Forum Message <> Reply to Message

Much obliged, Honza. I will try to create a derived class to override the memory location of the raw data. If that doesn't work, I guess I will be stuck with the memcpy that I was trying to avoid, but no big deal. Thank you for the clear examples!

Kel

Subject: Re: Displaying raw image data
Posted by dolik.rce on Tue, 23 Apr 2013 05:27:01 GMT
View Forum Message <> Reply to Message

keltor wrote on Tue, 23 April 2013 00:03Much obliged, Honza. I will try to create a derived class to override the memory location of the raw data. If that doesn't work, I guess I will be stuck with the memcpy that I was trying to avoid, but no big deal. Thank you for the clear examples!

Kel

You're welcome. If you really want to try modifying the U++ classes, you'll probably need to create your own variant of ImageBuffer::Create() that skips the pixels Buffer allocation. Note that you'll probably not be able to use derived classes, Image and ImageBuffer are quite tied together and were not designed with extensibility in mind. So you'll probably have to modify the methods directly in the U++ sources, meaning you'll have to maintain it when new versions come out, which can be quite a burden sometimes Consider yourself warned

But... Are you sure memcpy is a problem here? It usually takes much less time than generating the image. Don't forget what Donald Knuth said Quote:We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.

Honza

Subject: Re: Displaying raw image data
Posted by keltor on Tue, 23 Apr 2013 09:04:04 GMT
View Forum Message <> Reply to Message

Oh, no, I meant to create a class derived from my image library. I planned to make some kind of wrapper that contains both the class that deals with the image and a pointer to the memory that can be accessed by Upp::Image, as per your first example.

However, you are probably right, I may be too paranoid. Images, even in uncompressed format, tend to be rather small for today's standards. Copying 5, 10 or 30 MB is next to nothing. My original motivation was that the user may interact with the image in real time, rotate it in 3D and such. So I thought that it would reduce the refresh time if I'd do it without memcpy. But I will make it as you suggest first, and see how fast it is. The more I think about it, the more I agree that there is basically no overhead.

Thanks again, I'll get to it now.

Kel

Subject: Re: Displaying raw image data
Posted by keltor on Fri, 26 Apr 2013 06:37:04 GMT
View Forum Message <> Reply to Message

Well, it seems that this solution does not work completely. Image expects a RGBA*, while I have an unsigned char*. Force-casting does not work (naturally), the image just displays some garbage.

Does anyone know an efficient way to solve this? Perhaps how to create a RawEncoder, similar to the existing JPEGEncoder?

Thank you,

Keltor

Subject: Re: Displaying raw image data
Posted by Zbych on Fri, 26 Apr 2013 08:55:54 GMT
View Forum Message <> Reply to Message

Why don't you create converter that takes your raw image and returns Image object?

Subject: Re: Displaying raw image data
Posted by keltor on Fri, 26 Apr 2013 09:06:37 GMT
View Forum Message <> Reply to Message

Since I don't know Image too well (hence the newbie section), the only way I know how to solve the problem so far is to take 4 bytes, create an RGBA from them, and store it in an RGBA* sequentially. I was hoping for something a bit more optimised. It seems a bit of a waste of space and time...

Subject: Re: Displaying raw image data
Posted by Zbych on Fri, 26 Apr 2013 09:22:55 GMT
View Forum Message <> Reply to Message

It is quite easy. You just have to create ImageBuffer:

ImageBuffer ib(image_width, image_height);

And fill it with new pixels:

```
for (int y = 0; y < ib.GetHeight(); y++) {
 RGBA * dst = ib[y];
 for (int x = 0; x < ib.GetWidth(); x++) {
  dst->b = conversion from source - blue;
  dst->g = conversion from source - green;
  dst->r = conversion from source - red;
  dst->a = 255;
  dst++;
 }
}
```

And after that you can convert it to Image.
Image img = ib;

Subject: Re: Displaying raw image data
Posted by keltor on Fri, 26 Apr 2013 10:16:39 GMT
View Forum Message <> Reply to Message

Ah, yes, that does the trick. I just didn't realise that one can fill up ImageBuffer directly. Pretty basic c++ pointer stuff, to be honest, but I failed to see it at first. So your code is basically my intended solution, but optimised to avoid unnecessary copying.

Thank you Zbych, and also Honza.

Kel

Subject: Re: Displaying raw image data
Posted by dolik.rce on Fri, 26 Apr 2013 11:31:53 GMT
View Forum Message <> Reply to Message

keltor wrote on Fri, 26 April 2013 08:37Image expects a RGBA*, while I have an unsigned char*. Force-casting does not work (naturally), the image just displays some garbage.Casting should work, I do it quite often The only restriction that your library must return data in same order as they are sorted in RGBA, that is {B,G,R,A} if I remember it correctly. Otherwise the color channels would be switched, which might result in garbled data as you describe.

Anyway, I'm glad to hear you found a working solution

Honza
