Subject: Rendering
Posted by Tom1 on Fri, 24 Feb 2012 13:50:25 GMT
View Forum Message <> Reply to Message

Hi,

I decided to continue the rendering conversation outside the RGBA/Color topic, since this clearly does not belong there.

--

My mind goes frequently around graphics rendering, APIs for that and even dreams of hardware acceleration. Not to mention the 'terrifying' changes in the OS platforms.

I really like to think of U++ as the ultimate abstraction layer to all important platforms and technologies which will protect my SW development investment in the years to come and over the platform changes. I'm thinking of Win32/WinRT change and Wayland coming to Linux as examples of this.

Writing and maintaining applications necessitates that we have fixed interfaces that remain available for the years to come. Employing new technologies (behind the scenes from the application's point of view) requires new abstraction layers and potentially new interfaces to gain access to new features while preserving source compatibility for the existing applications.

I admit I do not understand the U++ rendering infrastructure nearly well enough, and therefore my ideas may be simply stupid.

My point with void Paint(Painter &painter); as an option was just to offer access to new Painter grade rendering features in a way that could allow various backends to implement them efficiently later with different backend technologies and on different platforms -- even accelerated some day... perhaps.

The idea is analogous to what Draw does currently with GDI grade graphics.

Maybe I see the big picture all wrong (it has happened to me before ...)

How do you see the future on this subject?

Best regards,

Tom

Subject: Re: Rendering
Posted by mirek on Fri, 24 Feb 2012 14:51:12 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Fri, 24 February 2012 08:50
How do you see the future on this subject?

IMO, Draw is functional subset of Painter and it is easier to use too, yet it has enough capabilities to cover 90% of cases (I have to admit that I have not yet used Painter to paint the surface of widget in any of my production code. Draw is simply enough).

Draw, unlike Painter, can also be implemented with almost all host systems, which then leads to HW acceleration.

If you have to use Painter, it is OK, but it looks like the hard part, the one that needs to be maintained in the furure, is painting algorithm itself, not its binding to Paint, which is relatively simple.

As to the whole subject of WinRT/Wayland etc... I am starting to think that maybe some experimental RAINBOW project to hijack some other toolkit as U++ low-level might not be that bad idea... OTOH, perhaps OpenGL backend is enough too. Hard to say.

Or maybe we should make CtrlCore somewhat optional and build CtrlLib on higher-level stuff... I do not know. But our unability to get MacOSX covered makes me a little bit nervous

---

Subject: Re: Rendering
Posted by Tom1 on Wed, 29 Feb 2012 20:35:47 GMT
View Forum Message <> Reply to Message

Hi Mirek,

Thanks for your views. Using OpenGL sounds much more appealing to me than some heavyweight toolkit. Do I understand it correctly that Uno is working on this as a Rainbow backend?

If U++ would work on MacOSX, I would certainly consider getting one.

--

The original reason I started using Painter was nothing more than lack of wide dashed lines in Draw. Now I'm addicted to anti-aliasing too...

Anyway, could you consider merging wide dashed lines support to Draw if I can supply working code for both GDI and X11, while also keeping it source compatible for applications?

Best regards,

Tom

Subject: Re: Rendering
Posted by mirek on Wed, 29 Feb 2012 21:44:20 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Wed, 29 February 2012 15:35

Anyway, could you consider merging wide dashed lines support to Draw if I can supply working code for both GDI and X11, while also keeping it source compatible for applications?

Best regards,

Tom

Actually, I was rather thinking about removing (well, obsoleting) anything line/polygon related from Draw...

Thing is, most application I have ever done are only using DrawRect, DrawText and DrawImage. It is enough to build a GUI. For polygons, we can use Painter.

Would simplify creating new rainbow backends as well...

Mirek

Subject: Re: Rendering
Posted by Tom1 on Thu, 01 Mar 2012 09:05:29 GMT
View Forum Message <> Reply to Message

Scary!

If Draw would only support Rect, Text and Image, it sounds like a lock-out of HW accelerated vector graphics. Is this really your intent? Although there are many dialogs using standard controls, most of my applications use vector graphics too.

Best regards,

Tom

Subject: Re: Rendering
Posted by mirek on Thu, 01 Mar 2012 09:25:38 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Thu, 01 March 2012 04:05Scary!

If Draw would only support Rect, Text and Image, it sounds like a lock-out of HW accelerated vector graphics. Is this really your intent?

At this moment, it is just opinion, not intent

And, if that happens, it would be more likely "obsoleting", not removing. Or maybe saying that basic Draw is not required to implement those or something like that.

All things considered, I would say that the basic level provided by Draw is simply unsufficient for anything serious and good looking by today's standards - but if you move beyond that, you would find that there is little HW acceleration possible for 'fine' 2D.

Subject: Re: Rendering
Posted by koldo on Thu, 01 Mar 2012 12:08:18 GMT
View Forum Message <> Reply to Message

Hello Mirek

You are right but for now there is a big speed difference between Draw and Painter.

Draw visual quality is sub-standard but key for special applications. I realized it using ScatterCtrl for "real time" graphs: Draw is much faster.

Subject: Re: Rendering
Posted by Tom1 on Sat, 03 Mar 2012 08:33:05 GMT
View Forum Message <> Reply to Message

Hi,

Based on current GDI and X11 implementations, I have been thinking of Draw as a single native vector graphics abstraction that is and will be mapped to all native targets on all supported platforms, no matter if they are window, buffer, printer or PDF. Additionally, some targets (window and buffer) may gain acceleration on the way if such is supported by the target. IMO this kind of one-stop-vector-graphics-shop will be needed in the future too.

If mandatory parts of Draw will be narrowed down, I seriously suggest having mandatory BeginGdi()/EndGdi() -like solutions for all native targets to allow proper target specific extensions to be developed. This way it would be possible to make a 'SuperDraw' that takes Draw as contructor parameter, and then implements wider API calling GDI/X11/OpenGL/Painter or whatever the backend really is. If the backend was not (yet) natively supported by 'SuperDraw',

we could use local BufferPainter to handle the drawing and push the image to Draw.

Best regards,

Tom