Subject: Porting SystemDraw to Frambuffer
Posted by kohait00 on Thu, 03 Sep 2009 09:18:24 GMT
View Forum Message <> Reply to Message

Hi there,

SystemDraw has been there for a bit of time now, but it's only implemented for the major backends, Win32 and X11 on linux.

beeing on linux embedded, one often has no X11, so bare /dev/fb0 is what we have. no problem though, one might start at that point.

so my intention is here to port the SystemDraw layer (of which i have no pretty clue for now) to the famous framebuffer.

now here is the problem, i might need to look around for quite a time in the code, finding out which are the vital parts of the SystemDraw, and where to settle to write to fb.

If you could simply give a short overview of how SystemDraw is logically set up, meaning what is layer down to device, what is facing U++ layer.. so i know where i am. not filtering unneeded function in the code wasting time.

Or is there a (even short) description of the SystemDraw class..

So far i have understood that the class is splitet itself in 2 parts, beeind the destination draw for the GUI, and using some function to update data in the destinated layer, well, thats quite logical.

any help greatly welcome, or even if anyone has started already, i might join.

saludos
kostah

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Thu, 03 Sep 2009 09:26:43 GMT
View Forum Message <> Reply to Message

an idea came along..

i'd need a bunch of drawing operations, which already are there, in the ImageDraw stuff.

so we might take a step to use an ImageDraw, somehow opening /dev/fb0 and using that one as SystemDraw, so we would have a single window application, nevermind , no problem, thats a start

now any comments of those who already know more of the base part down there in U++, what might be the problems???

Subject: Re: Porting SystemDraw to Frambuffer
Posted by mirek on Fri, 04 Sep 2009 10:06:29 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 03 September 2009 05:26an idea came along..

i'd need a bunch of drawing operations, which already are there, in the ImageDraw stuff.

so we might take a step to use an ImageDraw, somehow opening /dev/fb0 and using that one as SystemDraw, so we would have a single window application, nevermind , no problem, thats a start

Nope, you need Painter, more specifically BufferPainter.

fb support is one of areas I would like to investigate. I have a very stupid question:

Is it possible to somehow develop fb application in X11 environment? I mean, how do I activate fb from X11 so that I can see results?

Mirek

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Fri, 04 Sep 2009 11:55:39 GMT
View Forum Message <> Reply to Message

hi mirek

the point is that we *dont* use X11, so we work on bare /dev/fb0 and drivers (/dev/input/* or dev/mouse and even tslib stuff for touchscreen) to get that working..

now the drivers integration is part of its own, but displaying only, that is sommewaht part of Draw.

so i simply open /dev/fb0 within BufferPainter and have my Draw interface to that?. i think we might need an extra buffer painter here, fb0 is somewhat more complicated

you "simply" open /dev/fb0, but you then mmap it to your userspace and can then start painting on it (some more details on that needed). to find out mode or to set it, to find out strides, color witdth and the like there is this fbinfo struct (donno the name exactly, must look it up also) and according to that, BufferPainter needs to be set up. So maybe a wrapper class?

now the question arising is, where do you really set up the global or static first Ctrl, TopWindow or the like, which is base for all what CtrlCore is doing in the gui, including showing other control? i dont speak of windowing, its part of OS, but as far as i understood (looking up Win32 implementation of CtrlCore stuff) there somewhere mus be the point where a Topwindow or a SystemDraw is instantiated and drawn to GDC and passed all the events to.

Where is that point exactly, i could find it seen thing like CreateWindow and CreateWindowEx, but the rest.... thats kind of specific.. i wanted to know where the start point for Upp actually begins, where OS things switch /hand over to Upp stuff/interface.

as a startpoint we might want to take a look at the nanox project in blackfin uClinux trunk, here is the start point maybe, how to use fb0 (in a simple way, nanox is smalles X11, but we might want to have no X11 at all.
svn: blackfin.uclinux.org, svn of uClinux distro
trunk/user/microwindow/src/nanox/clientfb.c
dig for /dev/fb0

http://blackfin.uclinux.org/gf/project/uclinux-dist/scmsvn/?action=browse&path=%2Ftrunk%2Fuser%2Fmicrowin%2Fsrc%2Fnanox%2Fclientfb.c&view=markup&revision=7547

another Point: SDL

i saw that SDL is sowhat supported, how's that? it can use bare fb0 and also has its layer to the input driver stuff, including touch screen. does Upp support SDL natively?? meaning without any GTK stuff and the like? or even in windowss a native SDL?
native I mean making Upp *GUI* prjects that use no X or Win32 but SDL to display and interact. I've seen that SDL can be compiled, but it actually only using Core stuff, so no Upp CtrlCore things, its simply compiling with the IDE and having benefit of Core, but we want more

that was quite a lot. sorry

Subject: Re: Porting SystemDraw to Frambuffer
Posted by Mindtraveller on Fri, 04 Sep 2009 12:40:34 GMT
View Forum Message <> Reply to Message

Did I understand correctly that you try to create layer that enables writing POSIX GUI apps wthout need of X11?
(excuse me interfering this topic)

Subject: Re: Porting SystemDraw to Frambuffer
Posted by mr_ped on Fri, 04 Sep 2009 13:53:56 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Fri, 04 September 2009 14:40Did I understand correctly that you try to create layer that enables writing POSIX GUI apps wthout need of X11?
(excuse me interfering this topic)

I would say partially that's what he's trying to do, except he's probably not worried about window manager itself, he needs that part inside window, i.e. drawing widgets + receiving events. Moving/resizing/overlapping windows is maybe out of scope for him right now.

kostah:
The current status of SDL package is add-on, so you can add it to Core application, initialize the screen, and use ordinary SDL functions to draw something, but the CtrlCore is not aware of it at all.

edit: and due to licensing of SDL I'm not sure it's possible to integrate it into base U++ libs so tightly.
I think Mirek would rather opt for custom U++ FrameBufferPaint back-end (under BSD license), but I'm not sure if it fits into his development budget, there's never enough time to do everything you would like to.

Subject: Re: Porting SystemDraw to Frambuffer
Posted by zsolt on Fri, 04 Sep 2009 14:06:03 GMT
View Forum Message <> Reply to Message

Quote:Is it possible to somehow develop fb application in X11 environment? I mean, how do I activate fb from X11 so that I can see results?
I think, it is not possible to you use X11 and fb symultanously on the same hw.

About 10 years ago I created an embedded Linux app with SDL. It was convenient to develop it on my local machine under KDE.
The target device had a framebuffer console without X11.

SDL is a very thin layer, I found it not very resource hungry.

I think, the best solution would be to create an SDL backend to Draw and render ctrls into the SDL surface.
SDL apps can run on fb and in an X11 window also, without change.

And this way you would be able to compile the app to Windows as well (fortunately SDL is not POSIX only).

Subject: Re: Porting SystemDraw to Frambuffer
Posted by Mindtraveller on Fri, 04 Sep 2009 14:51:35 GMT
View Forum Message <> Reply to Message

May be it would be interesting to consider X11 alternative like DirectFB. As far as I know it is much more lightweight than X11 but handles GTK from the other hand.

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Fri, 04 Sep 2009 16:47:27 GMT
View Forum Message <> Reply to Message

hi people

to make me understand:

in embedded systems, you often have no X11 or DirectFb (although there is a small port nxlib for the Blackfin uClinux). all what we have is frambuffer and input drivers.

so the need is only to run *1* single application, without windowmanager, just as a simple control gui for things to do on an embedded system. so frambuffer / input drivers should be enough, so we could spare out all the complex layers, making an embedded system slow and using large memory chunks (in fact thats the biggest concern in ES). using other intermediate layers like DirectFB or SDL, which give you abstract access to fb0 and input drivers is fine and comfortable, but also brings in some costs (regardless of licence, this is another concern as well).

i hope you got the picture..

in any case, we need a layer for the CtrlCore that either can draw on /dev/fb0 and get its input from /dev/input/* or a layer for CtrlCore to interact with SDL or DirectFB..

The graphic stuff is far complex, and the layering especially under linux is of its own, i.e SDL itself can run on DirectFB or bare Framebuffer, DirectFB can run on X or DirectFB or others...but you know all that already.

like you already got the point, SDL is just able to be build and used under Upp, but only for Core (not CtrlCore) base Applications, means not Upp native GUI applications, simple Console applications or so..

so what would be the best / easiest / most flexible solution to drop the X11/Win32 chains ??

Application layer like that is meant to be:

```
Upp GUI application (single window normally, one TopWindow instance)
||
output===================input
||                      ||
framebuffer             inputdrivers
```

or like that

```
Upp GUI App (real CtrlCore)
||
SDL
||
output=======================intput
||                          ||
(frambuffer, X, GL,Dir.FB)   (/dev/input or touchscreen, or X11..)
```

or like that

```
Upp GUI App (real CtrlCore)
||
DirectFB
||
output=====================intput
||                  ||
(frambuffer, X, GL..)   (/dev/input or touchscreen, or X11..)
```

or like that

```
Upp GUI App
||(native X11 layer)
nanox (former MicroWindows)
||
output=====================intput
||                  ||
(frambuffer)   (/dev/input or touchscreen)
```

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Fri, 04 Sep 2009 16:49:30 GMT
View Forum Message <> Reply to Message

Addition:

its even worth it to use Upp here in the point of not to need GTK/KDE/FLTK/wxWidgets and all that stuff

it makes it obsolete as far as i know

Subject: Re: Porting SystemDraw to Frambuffer
Posted by mirek on Fri, 04 Sep 2009 16:53:20 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Fri, 04 September 2009 07:55hi mirek

the point is that we *dont* use X11, so we work on bare /dev/fb0 and drivers (/dev/input/* or dev/mouse and even tslib stuff for touchscreen) to get that working..

I am asking because of development/debugging environment...

Quote:
you "simply" open /dev/fb0, but you then mmap it to your userspace and can then start painting on it (some more details on that needed). to find out mode or to set it, to find out strides, color witdth and the like there is this fbinfo struct (donno the name exactly, must look it up also) and according to that, BufferPainter needs to be set up. So maybe a wrapper class?

I think you are going to do double-buffering anyway, so ImageBuffer is just fine for this (as you can solve the format difference later...)

Quote:
where a SystemDraw is instantiated and drawn to GDC and passed all the events to.

It is during handling of WM_PAINT (or expose).

Quote:
Where is that point exactly, i could find it seen thing like CreateWindow and CreateWindowEx, but the rest.... thats kind of specific.. i wanted to know where the start point for Upp actually begins, where OS things switch /hand over to Upp stuff/interface.

as a startpoint we might want to take a look at the nanox project in blackfin uClinux trunk, here is the start point maybe, how to use fb0 (in a simple way, nanox is smalles X11, but we might want to have no X11 at all.
svn: blackfin.uclinux.org, svn of uClinux distro
trunk/user/microwindow/src/nanox/clientfb.c
dig for /dev/fb0

Well, thinking about it, I guess nice and simple solution is to provide some abstraction level...

I mean, a class that abstractly represents sort of framebuffer and mouse and keyboard input and then build the whole thing above it.

For development purposes, we can use normal single window to emulate this.

Quote:
another Point: SDL

i saw that SDL is sowhat supported, how's that?

Not realy, the only support is that it comes with mingw version.

Quote:
it can use bare fb0 and also has its layer to the input driver stuff, including touch screen. does

Upp support SDL natively?? meaning without any GTK stuff and the like? or even in windowss a native SDL?

No. But I guess we can use SDL as inspiration for above abstraction layer...

Mirek

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Wed, 16 Dec 2009 10:19:59 GMT
View Forum Message <> Reply to Message

Ok, people, this framebuffer topic is about to have a revival.
since we have the Painter stuff now and the headless drawing, and even the Docking facilities in bazaar, we almost have a "window manager".. but I need help

I have started to prepare some porting stubs for the use of framebuffer in the CtrlCore. Well, for now, ist really stubs, just moslty plain copy and paste from X11 code (which lies next), and commenting doubtable or unneeded/unneccessary parts away.

i realized that the underlying interface is quite "distributed" over a lot of sections, without explicit "tagging" what is what. it'd be esier to have the commen interface functions somehow grouped, so one could know, these functions are required. current situation is that its pretty mixed with the helper functions for the apropriate interface as well. both, in headers and code. so this is maybe to be reorganised, especially if we take in account, that this might not be the last porting we face (next important would be to port to apple graphics), and there will be even more i think. so we need to make things clearer here. (at least a short description on what so far is used by both interfaces WIN32/X11 and is required in any other port, and what are their interface specific helpers, and maybe a sentence of description to each, especially when it comes to Clipboard,DragandDrop and the like).

back to topic.

base idea:
using /dev/fb0 to display the one and only (no window manager) TopWindow instance of an application (naturally fullscreen), which is driven directly by user input extracted from /dev/input layer

class
SystemDraw : public BufferPainter

where the BufferPainter can be initialized with an ImageBuffer
which in turn should somehow represent the /dev/fb0.
-->problem: the fb0 can have really weired dimensions, bitwisths per color, strides and the like..is the ImageBuffer capable to cope with that? or do we need to have the double buffering as only solution (where the normal ImageBuffer 24bit is overlayed/calculated on tob of the weired fb0

layout)?

the control input section should be driven directly with /dev/input/, this normally does invoke a repaint in some sort

we need to provide a full implementation of own drag and drop, which can be significantly simpler i think, because we dont drag between multiple windows od Win32/X11, but simply inside the TopWindow (which is between Ctrls)

we need to provide a full replacement for Clipboard stuff, how does this work?

we need to provide a full replacement of the TimerThread (which is used heavily by Ctrls, it also needs to be reentrant (means a new occurance of timer event is handled either by interrupting a possibly currently still working timerthread procedure, or by spawning another thread to handle this, which leads either to application lock (due to repeated reentrance) or to thread explosion if handling callbacks are misdimensioned somehow.

The GUI thread (which is the main thread) would be far the simplest part i think, cause it creates, opens and runs the TopWindow and realizes all the drawing and user input.

is there any description (besides the code itself) available in terms of how Clip, Drag and Drop, the timer thread and the GUI thread are set up (basic ideas should be enough)

the GUI flag, which is used currently can be abandoned maybe, because it is evaluated only in POSIX environment, in WIN32 GUI is somewhat implicit, as soon as one uses the package CtrlCore, one uses gui stuff in Win32. in Linux, one should be able to specify GUI X11 or GUI FB somehow.. but this is small peanuts. backwards compatibility can be done imlicitly by defining a standard flag.

the stubs (NOT implementations) are half way done, as soon as it compiles and links, i will post it here, maybe mirek can take a look over it and clean out what he dooms or claims not neccessary (since i have no much idea of underlying gui facility) and maybe already integrate it in the trunk, to be able to code parallely. first thing woul be, of corse, to open the /dev/fb0 and display the main window once...

ok, till then, cheers

PS: maybe we can really use this effort to restruct code to make porting in future a lot esier.

PSS: sorry for the text chunk

Subject: Re: Porting SystemDraw to Frambuffer
Posted by mirek on Wed, 16 Dec 2009 13:37:37 GMT

kohait00 wrote on Wed, 16 December 2009 05:19
where the BufferPainter can be initialized with an ImageBuffer
which in turn should somehow represent the /dev/fb0.
-->problem: the fb0 can have really weired dimensions, bitwisths per color, strides and the like..is the ImageBuffer capable to cope with that?

No.

Quote:
 or do we need to have the double buffering as only solution (where the normal ImageBuffer 24bit is overlayed/calculated on tob of the weired fb0 layout)?

Yes. BTW, ImageBuffer is 32bit RGBA.

Quote:
we need to provide a full implementation of own drag and drop, which can be significantly simpler i think, because we dont drag between multiple windows od Win32/X11, but simply inside the TopWindow (which is between Ctrls)

we need to provide a full replacement for Clipboard stuff, how does this work?

IMO, it is in fact much easier to start from the scratch than implementing Clips in X11/Win32.

There is nothing magic, you have raw binary data and string that represents the type of data. Perhaps use filesystem to store both and you are done... (with clipboard anyway, D&D will need a bit more).

Quote:
we need to provide a full replacement of the TimerThread (which is used heavily by Ctrls, it also needs to be reentrant (means a new occurance of timer event is handled either by interrupting a possibly currently still working timerthread procedure, or by spawning another thread to handle this, which leads either to application lock (due to repeated reentrance) or to thread explosion if handling callbacks are misdimensioned somehow.

There is no TimerThread in CtrlCore And thing is pretty simple generally in fact. All you need to do is limit waiting for input events to some max timeout in message loop and to know current system time (one way or another). Then call TimerProc at the end of event processing or if there is timeout.

Quote:
is there any description (besides the code itself) available in terms of how Clip, Drag and Drop, the timer thread and the GUI thread are set up (basic ideas should be enough)

I hope that above is helpful for starters, ask for more details as you go.

Quote:
the GUI flag, which is used currently can be abandoned maybe, because it is evaluated only in POSIX environment, in WIN32 GUI is somewhat implicit, as soon as one uses the package CtrlCore, one uses gui stuff in Win32.

GUI is because Win32 - console vs GUI apps have different linker options - that is the original problem.

Quote:
in Linux, one should be able to specify GUI X11 or GUI FB somehow.. but this is small peanuts. backwards compatibility can be done imlicitly by defining a standard flag.

Well, that is interesting problem I guess GUI FB and X11 as default or something like that.

Quote:
PS: maybe we can really use this effort to restruct code to make porting in future a lot esier.

Well, it is not so messy, but it is true that over those 10 years of development, thing are getting more and more patched.

Anyway, in reality, I am not huge believer of fresh starts there. If nothing else, it would put all of my apps in jeopardy...

Also, it is questionable that you can gain too much more clarity there. Some of stuff in CtrlCore is pretty nasty, but that is partly because host-platform details are pretty nasty too...

Mirek

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Wed, 16 Dec 2009 14:19:31 GMT
View Forum Message <> Reply to Message

did i understand you right? there is no extra thread for the TimerProc in WIN32 or X11? (WIN32 used WM_TIMER as far as i know, ahh..which is ran by he same thread in spare times, when it sleeps and in X11 is maybe similar, right?
well, i'm learning.

your're right, platfomr stuff is a mess. so it's no wonder. actually i have learned a lot, how to have cleaner code because thanks to upp code. so its generally really clean, even fun to read, logical and short (most times) thank you, i'll give it a try

---

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Wed, 16 Dec 2009 21:08:53 GMT
View Forum Message <> Reply to Message

hi mirek, here comes the stubs for framebuffer environment. it should compile fine. is just the extract from my current svn tree (only the changed stuff left, simply replace)

flags situation:

GUI needs to be specified for FB only, if no backend follows, X11 is choosen automatically. (GUI X11). to compile frambuffer: GUI FB, WIN32 currently does not actively distinguish anything (besides what you mentioned).

i used X11 backend stuff as template, just tweaking as little as possible in first instance.

if possible, you could include this into trunk uppsrc, it comes with full .svn backend, so you can backcheck svn diff, but it mostly is things like extending #ifdef PLATFORM_X11 to #if defined(PLATFORM_X11) || defined(PLATFORM_FB) and the like.

the stubs, which were demanded during compile or linking have been prepared so far commenting code with

#ifdef UNUSED
...
#endif

next step i will try to prepare the TopWindow thing opening the /dev/fb0 and maybe painting a first picture. this could take a bit.

a branch would be good.. so i dont bother you to often, if you prefer

thanks for help

File Attachments

1) uppsrc_svnextract.zip, downloaded 386 times

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Wed, 16 Dec 2009 21:34:19 GMT
View Forum Message <> Reply to Message

BUG: use this one #include needs to go before NAMESPACE_UPP

File Attachments

1) DrawFB.h, downloaded 815 times

Subject: Re: Porting SystemDraw to Frambuffer
Posted by mirek on Thu, 17 Dec 2009 12:30:25 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Wed, 16 December 2009 09:19did i understand you right? there is no extra thread for the TimerProc in WIN32 or X11? (WIN32 used WM_TIMER as far as i know, ahh..which is ran by he same thread in spare times, when it sleeps

WM_TIMER is in fact only used to "wake up" from GetMessage.

In X11, the same thing is realized using timeout parameter in "select". What you really need is to get ProcessEvents run at most after 10ms of inactivity.

Mirek

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Mon, 28 Jun 2010 11:43:02 GMT
View Forum Message <> Reply to Message

hi all,

i've been busy with another project (which isnt done yet anyway, but it beats my nerves already . but now slightly i want to get back to this framebuffer issue.. over the past months a gained some amount of knowledge about upp, mirek and others are constantly providing extremely valuable infos and i start to wrap my mind on it (Painter, Draw, .. still in progress)

so here we go.. lets collect the things that will need to be adresses when porting. please complete what i forgot so the list could be used as a reference when wanting to port to other things (mac i.e.)

/////

* Drawing (open surface, check modes, alignment, color palette, implement Draw interface operations, TopWindow, DHCtrl ..)

for framebuffer this is more or less the base thing. a SystemDraw : BufferPainter which is directly doublebuffering to the /dev/fb0 with help of a converting function (which adresses strides/alignment issues). The Draw interface is completely implemented in BufferPainter.. TopWindow can only be one for framebuffer, in fullscreenmode.

* Input Interface (open input streams, /dev/input*, translate to Upp messaging, prepare for

periodical invocations

* Timing (set up thread to invoke repaint each 10 ms or on demand, Timer Queue, PostCallback adaption

* Drag&Drop (this is probably the most complicated part, setup static infrastructure to mark things that are dragged??)

* Clipboard (also static infrastructure, buffers for text, images, etc..??)

* Fonts (any idea?)

* later, small "WindowManager" replacement, to be able to render / run more that one TopWindow

////

but still got some questions
what is the exact purpose / idea of:

1) TopWindow (create & startup a system dependant top window instance, linking its GDI to upp stuff as well as means to dispatch input messages. all other Ctrl's are placed as upp own children, not system own windows..)

2) DHCtrl (Ctrl with own GDI context??, seperately refreshable without affection other ctrls?? i found some in info in
 http://www.ultimatepp.org/forum/index.php?t=msg&goto=267 38&
but i guess it wont be needed in frambuffer environment at first.

what more issues do we need to adress here? (maybe this could be sort of a porting guide...)

thanks

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Sun, 04 Jul 2010 09:58:33 GMT
View Forum Message <> Reply to Message

another querstion:

where will be the problems (as expected) concerning Chameleon stuff and framebuffer? is there things to account for? how to properly initialize Chameleon stuff when you dont have OS underlayer?

Subject: Re: Porting SystemDraw to Frambuffer

Posted by mirek on Tue, 06 Jul 2010 07:49:14 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Sun, 04 July 2010 05:58another querstion:

where will be the problems (as expected) concerning Chameleon stuff and framebuffer? is there things to account for? how to properly initialize Chameleon stuff when you dont have OS underlayer?

Not much problem there. Basically, you compile as "NOGTK" and get default, non-OS dependent, look&feel.

Anyway, I may repeat myself, but the idea is that there should be something like "FB" flag and the whole framebuffer "backend" should be virtualized, so that you can easily bind U++ to any sort of device (framebuffer, SDL...).

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Tue, 06 Jul 2010 08:13:29 GMT
View Forum Message <> Reply to Message

Quote:
compile as "NOGTK" and get default, non-OS dependent, look&feel.

thats great..., i already have specified NOGTK flag when selecting FB, so its goot to know that was right

Quote:
there should be something like "FB" flag

i already have previsted something like that, flags: GUI X11 for current linux stuff, GUI FB for framebuffer stuff, with fallback option, if nothing else specified, GUI defaults to GUI X11, so no package breaks

Quote:
the whole framebuffer "backend" should be virtualized, so that you can easily bind U++ to any sort of device (framebuffer, SDL...).

dont quite get you here. frame buffer cant be "virtualized", its a backaned like the others as well, with its own SystemDraw, input parsing queue, timer procedure etc.. could you explain?

btw: SDL should be a port of its own. ofcorse, then, you can run upp on framebuffer as well, because SDL runs on framebuffer as well, but upp wont know about framebuffer..

Subject: Re: Porting SystemDraw to Frambuffer
Posted by mirek on Thu, 08 Jul 2010 08:30:23 GMT

View Forum Message <> Reply to Message

kohait00 wrote on Tue, 06 July 2010 04:13Quote:
compile as "NOGTK" and get default, non-OS dependent, look&feel.

thats great..., i already have specified NOGTK flag when selecting FB, so its goot to know that was right

Quote:
there should be something like "FB" flag

i already have previsted something like that, flags: GUI X11 for current linux stuff, GUI FB for framebuffer stuff, with fallback option, if nothing else specified, GUI defaults to GUI X11, so no package breaks

Quote:
the whole framebuffer "backend" should be virtualized, so that you can easily bind U++ to any sort of device (framebuffer, SDL...).

dont quite get you here. frame buffer cant be "virtualized", its a backaned like the others as well, with its own SystemDraw, input parsing queue, timer procedure etc.. could you explain?

All these 'details' can be expressed by a bunch of virtual methods (very simplified said).

Later you take these and bind U++ to SDL or framebuffer or whatever else.

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Thu, 08 Jul 2010 08:38:35 GMT
View Forum Message <> Reply to Message

but they are already present arent they?

otherwise there will be a compile flags dependant source port like current situaion, for X11 and WIN32

Subject: Re: Porting SystemDraw to Frambuffer
Posted by mirek on Thu, 08 Jul 2010 09:00:31 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 08 July 2010 04:38but they are already present arent they?

No.

I am speaking about generic interface to accomodate input event queues and output buffer. This is not to be confused with Ctrl interfaces....

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Thu, 08 Jul 2010 09:36:01 GMT
View Forum Message <> Reply to Message

i'll try to..

Quote:
whole framebuffer "backend" should be virtualized, so that you can easily bind U++ to any sort of device (framebuffer, SDL...).

well, SDL is in fact not quite the same as simply framebuffer and /dev/input, its more sort of X11, you allocate a SDL_Surface, and need to process input events..

as far as got through that whole thing, SystemDraw is the implementation is draw backend and DoMouse & DispatchKey are already backends... to these i tried to connect.

so far i tried to sum up the whole WIN32 thing by runtime tracing in debugger: correct if i am wrong somewhere..

 /// ////
SYSTEM BACKEND WIN32

win32 api current state and important stations, analyzing and tracking the GUI_APP_MAIN macro.

Ctrl::InitWin32(HINSTANCE hInstance)
init/ register der window classes and a seperate timer class, instantiating of timer class with lpfnWndProc = &Ctrl::UtilityProc; the window classes with lpfnWndProc = (WNDPROC)Ctrl::WndProc;, they are the message processors. calls to create windows are done with win function CreateWindow later.

void AppInitEnvironment__()
language and environment specific inits like charsets

GuiMainFn()
applications specific stuff, global intis, here, Run() of a TopWindow class should be called

TopWindow::Run()
basicly forwards to create the window class instances in Open() call, Show()s it and sets up the EventLoop() to run, which only exits when app quits. prepares the ctrl exit.

void TopWindow::Open()
creates the window instance using the Create() subsystem for the main window instance

void Ctrl::Create0(Ctrl::CreateBox *cr)
actually creates and shows the window instance with win32 OS means, sets up the drag and drop stuff and calls a first RefreshDeepLayout(), which triggers a WM_PAINT, which is processed in Ctrl::WndProc

LRESULT CALLBACK Ctrl::WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
takes care of processing WM_CREATE und WM_DESTROY, otherwise forwards to the hwnd specific Ctrl (mapping exists), which is a TopWindow ctrl.. forwards to the WindowProc of the Ctrl, which is virtual

LRESULT TopWindow::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
takes care of some minimal TopWindow specific stuff like close behaviour, then, forwards to Ctrl::WindowProc

LRESULT Ctrl::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
processes the win32 message queue messages, here comes in WM_PAINT, which uses a SystemDraw to paint the main / top Ctrl's area to the win32 GDC surface of the hWnd, here also comes in WM_'mousestuff' which forwards to DoMouse, which takes care of translating and delivering the messages to the ctrls. further, WM_'keystuff' comes in, which forwards to DispatchKey..

void Ctrl::EventLoop0(Ctrl *ctrl)
is called in Run() (via EventLoop) and returns when application is about to finish, basicly calls in a loop ProcessEvents() which calls ProcessEvent() which calls sProcessMSG()

void Ctrl::sProcessMSG(MSG& msg)
runs the win32 API typical TranslateMessage / DispatchMessage duo, which ends up calling WndProc and dispatching / evaluating messages

LRESULT CALLBACK Ctrl::UtilityProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
processes the WM_TIMER message. the timer is used to wake up the ProcessEvents stuff (TranslateMessage / DispatchMessage) which will also process what?? why does it need to wakeup? process the postcallbacks? becuse, user input and repaint stuff comes directly as WM_* messages, which wake up the queue as well

sorry for the long post, just need to clarify i dont miss things.

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Thu, 08 Jul 2010 13:07:41 GMT
View Forum Message <> Reply to Message

so my base idea would be:

InitFB() / ExitFB()
open and close the framebuffer and input files, init the framebuffer blitting (in case of weired formats) and all other stuff .

Open() / Create()
ensures that only one TopWindow exists, sets fullscreen flag, makes first RefreshDeepLayout()

ProcessEvents()
reads userinput and translate it to DoMouse() / DispatchKey() repaint the TopWindow area using SystemDraw from FB

about the timer thing i need some ideas

Subject: Re: Porting SystemDraw to Frambuffer
Posted by dolik.rce on Thu, 08 Jul 2010 13:41:53 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 08 July 2010 15:07
about the timer thing i need some ideas

Hi Kohait,

What about bazaar/Timer? It is pretty universal and independent package, someone isolated it from CtrlCore for use in console apps (it depends only on core). Only problem might be that it requires MT (Is it really problem?).

kohait00 wrote on Thu, 08 July 2010 15:07
Open() / Create()
ensures that only one TopWindow exists, sets fullscreen flag, makes first RefreshDeepLayout()
It might be easier (and even better) to not control how many TopWindows is opened. Just paint the topmost (the latest opened).

I really like the idea of U++ on framebuffer. Expect my help as far as my knowledge goes

Bye, Honza

EDIT: Oh, now I see that bazaar/Timer is your work, so you probably already considered it So, what about using POSIX timers (timer_create, timer_settime etc.)? I'll have a closer look at it as soon as I get back to my linux computer (which is unfortunatelly not until next week)

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Thu, 08 Jul 2010 15:25:36 GMT
View Forum Message <> Reply to Message

hey dolik, thanks for help

i really considered using Timer, but it uses an extra thread.. well, what about apps compiled with GUI but *without* MT, they wont work..

i basicly thought of using the timer *queue*, executing it myself (means in the MainThread i call TimerProc in a loop)..thus i could PostCallback to it.

but: the TimerProc uses a fixed sleep intervall, 10 ms, when nothing is to be done..now when it sleeps, it sleeps hard. no way to wake it up before that time. this will affect responsiveness.

maybe i will neet to reimplement TimerProc, which will not sleep, but wait for some kind of wakeup signals, maybe select() stuff..

still in modeling phase

Subject: Re: Porting SystemDraw to Frambuffer
Posted by mirek on Thu, 08 Jul 2010 15:55:48 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 08 July 2010 05:36i'll try to..

Quote:
whole framebuffer "backend" should be virtualized, so that you can easily bind U++ to any sort of device (framebuffer, SDL...).

well, SDL is in fact not quite the same as simply framebuffer and /dev/input, its more sort of X11, you allocate a SDL_Surface, and need to process input events..

Which is exactly why I would attempt to make the whole framebuffer variant generic...

Quote:
LRESULT CALLBACK Ctrl::UtilityProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
processes the WM_TIMER message. the timer is used to wake up the ProcessEvents stuff (TranslateMessage / DispatchMessage) which will also process what?? why does it need to wakeup? process the postcallbacks? becuse, user input and repaint stuff comes directly as WM_* messages, which wake up the queue as well

sorry for the long post, just need to clarify i dont miss things.

Utility proc is Win32 implementation hack - we are using one special window (utilityHWND) to handle some Win32 messages that could not be handled otherwise. UtilityProc is its windows proc.

The rest you get right, however, it has little to do with planned framebuffer target.

Mirek

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Thu, 08 Jul 2010 16:46:16 GMT
View Forum Message <> Reply to Message

thanks mirek

Quote:
it has little to do with planned framebuffer target.

ofcorse, but i need to know which stuff will be nessesary and which one is optional, and just wanted to be sure i got the underlaying layer to understand right (to spot and fit the interface).

step by step we go...

just a question aside: is there *anything* concerning format conversion or bitblitting in Painter ? (i asked it already maybe).
or is the one and only supported method: RGBA throughout Upp..

Subject: Re: Porting SystemDraw to Frambuffer
Posted by mr_ped on Fri, 09 Jul 2010 06:37:48 GMT
View Forum Message <> Reply to Message

All the major stuff is RGBA unified (unless Mirek says I'm wrong).

So in case of 16b 5:6:5 FB you have to convert the final RGBA raster during blit to FB. (which may be quite suboptimal on some weak devices, but recently mobile phones have 1GHz processors, so it's probably pointless to worry now ... anyway, forcing Painter&co to work also in some non-RGBA format would be probably as difficult as writing a new variant from scratch, so unless specific device proves to be worth of it, aim for RGBA internally)

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Fri, 09 Jul 2010 07:58:56 GMT
View Forum Message <> Reply to Message

i'm fine with RGBA, if painter is optimized for it, best case anyway, as far as i know, SDL does the same, everything RGBA until it is blited to destination, which may use a weired alignment / format.

the best thing would be to extend the Painter with some converterfunctions, to create i.e a ImageBuffer or sth. with the different format.. which i can write directly to framebuffer.

but first shot is ASSERT(RGBA) anyway.

Subject: Re: Porting SystemDraw to Frambuffer
Posted by mr_ped on Fri, 09 Jul 2010 10:50:49 GMT
View Forum Message <> Reply to Message

Quote:the best thing would be to extend the Painter with some converterfunctions, to create i.e a ImageBuffer or sth. with the different format.. which i can write directly to framebuffer.

You draw into Raster (or ImageBuffer?) with Painter, then with final Raster you can use RasterEncoder and RasterFormat classes to produce final format for output.

It will very likely in most cases require to write the final classes for desired format, but almost everything is ready, you just need to glue it together and init it correctly. See how RasterEncoder base is used trough upp (in various image formats plugins), or you may use RasterFormat only to convert RGBA line by line during blit, it's very simple for 15/16b modes. For palette modes you will have to do some more setup around conversion palette unless you want it dynamically created by upp.

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Fri, 09 Jul 2010 11:02:40 GMT
View Forum Message <> Reply to Message

thats a nice info, really
thanks, i'll take alook on it.

Subject: Re: Porting SystemDraw to Frambuffer
Posted by mirek on Fri, 09 Jul 2010 12:28:44 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Fri, 09 July 2010 06:50Quote:the best thing would be to extend the Painter with some converterfunctions, to create i.e a ImageBuffer or sth. with the different format.. which i can write directly to framebuffer.

You draw into Raster (or ImageBuffer?) with Painter, then with final Raster you can use RasterEncoder and RasterFormat classes to produce final format for output.

Well, I believe that "FB flag" configuration should end right at RGBA and eventual conversion should be performed by framebuffer host virtualization. It is one of tasks I would dedicate for that...

Mirek

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Fri, 09 Jul 2010 12:42:01 GMT
View Forum Message <> Reply to Message

Quote:
eventual conversion should be performed by framebuffer host virtualization

ofcorse, but it would be great to have some 'generic' bit bliting / stride copying functions in Painter / ImageBuffer, imagine all the weired graphics formats that could be exported with it. or is plugin/* the place for that?

Subject: Re: Porting SystemDraw to Frambuffer
Posted by mirek on Fri, 09 Jul 2010 14:44:24 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Fri, 09 July 2010 08:42Quote:
eventual conversion should be performed by framebuffer host virtualization

ofcorse, but it would be great to have some 'generic' bit bliting / stride copying functions in Painter / ImageBuffer, imagine all the weired graphics formats that could be exported with it. or is plugin/* the place for that?

We already have it. See RasterFormat.

Mirek

Subject: Re: Porting SystemDraw to Frambuffer
Posted by tojocky on Sat, 10 Jul 2010 17:13:12 GMT
View Forum Message <> Reply to Message

Very interesting!

In this month I intend to buy a HTC Desire. Android seems to have frame-buffer and can be tested with Android NDK.

Ion Lupascu (tojocky)

Subject: Re: Porting SystemDraw to Frambuffer
Posted by tojocky on Tue, 27 Jul 2010 21:09:32 GMT
View Forum Message <> Reply to Message

OK,

I'm ready to test to HTC Desire u++.
In the next fiew days I will install Android NDK and test to compile some real project like Navit

---

Subject: Re: Porting SystemDraw to Frambuffer
Posted by Novo on Wed, 28 Jul 2010 03:37:59 GMT
View Forum Message <> Reply to Message

tojocky wrote on Tue, 27 July 2010 17:09OK,

I'm ready to test to HTC Desire u++.
In the next fiew days I will install Android NDK and test to compile some real project like Navit

IMHO the Android Emulator should also be handy.

Subject: Re: Porting SystemDraw to Frambuffer
Posted by tojocky on Fri, 15 Oct 2010 11:15:16 GMT
View Forum Message <> Reply to Message

Can I know what news are about framebuffer?

I can test and help in development!

Thank you in advance.

Subject: Re: Porting SystemDraw to Frambuffer
Posted by kohait00 on Thu, 21 Oct 2010 07:25:25 GMT
View Forum Message <> Reply to Message

have to admit that i'havent gone too far since my work burdens have become important again. but the thing is definitely to be done.

mireks idea of an idependant, easy to port interface for upp CtrlCore / SystemDraw is a good idea. in terms of having some kind of function names which are known to be a must to implement,

like the message queue, that generates the input events for upp. so you just know here and there i need to post this and this message.

another must function is the drawing function where either a SystemDraw is instantiated to do drawing or what ever else..

it should also have some possibility to interrupt the waiting for messages in message queue like mirek pointed out, at about 10 ms..to be able to process PostCallbacks and SetTimerCallbacks...

how experienced are you in all the underhood upp interfaces? i'm actually having a hard time to get a grisp of what is what..