Subject: High Performance Drawing
Posted by cbpporter on Sun, 16 Dec 2007 11:48:52 GMT
View Forum Message <> Reply to Message

Hi!

I need high performance flicker free graphics. The high performance part is relativelly covered, but not the flicker free one.

Code that draws stuff when the mouse is moved is pretty flickery, even when drawn to an Image first (x and y are updated by mouse move. The issue is most apparent when using larger windows and resolutions:

```
void aggtest::Paint(Draw& draw)
{
 if (GetSize().cx == 0 || GetSize().cy == 0)
  return;
 ImageDraw agdraw(GetSize());
 agdraw.DrawRect(GetSize(), White);
 agdraw.DrawLine(x,y,150,150,50,Blue);
 Image img(agdraw);
 draw.DrawImage(0,0,GetSize().cx,GetSize().cy,img);
}
```

I don't think that this is a buffer issue, it is rather and issue with screen refresh synchronization. Does anybody know a way to reduce this flicker under Windows/Linux. I don't want to use OpenGL or DirectX to wait for hardware sync with the monitor.

Also, creating a buffer at every paint seems rather costly for large resolutions. Isn't there a way to keep a buffer pre-allocated and then obtain Image objects out of it with non-destructive copy?

And the documentation about ImageDraw is inaccurate:
Quote:
Image img(100, 100);

ImageDraw idraw(img);

myobject.Paint(idraw);

idraw.Close();

// now the Image has been modified by the myobject's drawing

operations and can be e.g. saved to disk

ImageDraw does not have a ImageDraw(const Image&) constructor or a Close() method.

Furthermore, I couldn't find a way to create an ImageDraw object starting from an existing Image since the above methods are not present.

Subject: Re: High Performance Drawing
Posted by mirek on Sun, 16 Dec 2007 22:15:40 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Sun, 16 December 2007 06:48Hi!

I need high performance flicker free graphics. The high performance part is relativelly covered, but not the flicker free one.

Code that draws stuff when the mouse is moved is pretty flickery, even when drawn to an Image first (x and y are updated by mouse move. The issue is most apparent when using larger windows and resolutions:

```
void aggtest::Paint(Draw& draw)
{
 if (GetSize().cx == 0 || GetSize().cy == 0)
  return;
 ImageDraw agdraw(GetSize());
 agdraw.DrawRect(GetSize(), White);
 agdraw.DrawLine(x,y,150,150,50,Blue);
 Image img(agdraw);
 draw.DrawImage(0,0,GetSize().cx,GetSize().cy,img);
}
```

I don't think that this is a buffer issue, it is rather and issue with screen refresh synchronization. Does anybody know a way to reduce this flicker under Windows/Linux. I don't want to use OpenGL or DirectX to wait for hardware sync with the monitor.

Also, creating a buffer at every paint seems rather costly for large resolutions. Isn't there a way to keep a buffer pre-allocated and then obtain Image objects out of it with non-destructive copy?

And the documentation about ImageDraw is inaccurate:
Quote:
Image img(100, 100);

ImageDraw idraw(img);

myobject.Paint(idraw);

idraw.Close();

// now the Image has been modified by the myobject's drawing

operations and can be e.g. saved to disk

ImageDraw does not have a ImageDraw(const Image&) constructor or a Close() method.

Furthermore, I couldn't find a way to create an ImageDraw object starting from an existing Image since the above methods are not present.

For starters, are you using BackPaint?

Subject: Re: High Performance Drawing
Posted by cbpporter on Mon, 17 Dec 2007 00:47:12 GMT
View Forum Message <> Reply to Message

Quote:For starters, are you using BackPaint?
Yes, but it has little to no effect, since I'm just painting the whole control in one swoop using a manual backbuffer.

Subject: Re: High Performance Drawing
Posted by mirek on Mon, 17 Dec 2007 13:47:48 GMT
View Forum Message <> Reply to Message

OK, in that case, do I need to repeat the mantra? ("testcase!")

I understand that in this case, the problem is perhaps something else than "basic repainting flicker", anyway, testcase is always much better than long description of problem...

Mirek

Subject: Re: High Performance Drawing
Posted by cbpporter on Tue, 25 Dec 2007 12:02:16 GMT
View Forum Message <> Reply to Message

Sorry it took so long, but it's the Holydays and I did not have access a computer.

Here is a little example with the problem. Also, I moved it to another computer, and here there is no refresh problem, but the performance is awful. I just tested a similar example (written in something else), which does not draw the same line, but still, it paint a bitmap to the screen and there is no performance issue.

File Attachments
1) aggtest.rar, downloaded 586 times

Subject: Re: High Performance Drawing
Posted by mirek on Wed, 26 Dec 2007 13:16:52 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Tue, 25 December 2007 07:02Sorry it took so long, but it's the Holydays and I did not have access a computer.

Here is a little example with the problem. Also, I moved it to another computer, and here there is no refresh problem, but the performance is awful. I just tested a similar example (written in something else), which does not draw the same line, but still, it paint a bitmap to the screen and there is no performance issue.

Hm, can you check the colors setting on "slow" machine? If it is set to anything else than TrueColor, this example can be much slower.

Note that this way of "antiflicker" (I mean, ImageDraw etc..) is principally much slower than simple BackPaint. Escpecially if machine is not in TrueColor.

Mirek

Subject: Re: High Performance Drawing
Posted by cbpporter on Wed, 26 Dec 2007 16:01:15 GMT
View Forum Message <> Reply to Message

It is set to 32 bit (AFAIK TrueColor is 24). Why is drawing on anything else than TrueColor slower. Are the buffers 24 bit and if you paint them to other bit depths, there is a hidden conversion or something. And 32 bit should be the fastest, because of both align issues and register size on a x86 machine.

I can't use just backpaint, because I need a buffer, draw something with agg, draw something with my functions, run it through some postprocessing and then display. I'm not doing this each time, so I need a persistent "buffer" which I can update and which is as fast as possible to paint on the screen.

Subject: Re: High Performance Drawing
Posted by mirek on Tue, 01 Jan 2008 09:54:16 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Wed, 26 December 2007 11:01It is set to 32 bit (AFAIK TrueColor is 24). Why is drawing on anything else than TrueColor slower. Are the buffers 24 bit and if you paint them to other bit depths, there is a hidden conversion or something. And 32 bit should be the fastest, because of both align issues and register size on a x86 machine.

I can't use just backpaint, because I need a buffer, draw something with agg, draw something with

my functions, run it through some postprocessing and then display. I'm not doing this each time, so I need a persistent "buffer" which I can update and which is as fast as possible to paint on the screen.

By truecolor I mean either 24 or 32.

Buffers really are 32-bit (RGBAx8). Anyway, if display is set to 256 colors, everything is much slower because Win32 has to perform the palette conversion....

Mirek

Subject: Re: High Performance Drawing
Posted by phirox on Tue, 01 Jan 2008 20:21:27 GMT
View Forum Message <> Reply to Message

Since I'm having a bit of the same problem, I tried out your testcase.

With some testing I figured out the problem lies in the draw.DrawImage part. I was wondering if there is some way to copy the imagebuffer from ImageDraw into the one of Draw. I've been spitting through the source a bit, but couldn't really find it.

This would really be cool as a feature though.

p.s.:

Quote:Also, creating a buffer at every paint seems rather costly for large resolutions. Isn't there a way to keep a buffer pre-allocated and then obtain Image objects out of it with non-destructive copy?

With the virtual function Layout() you can make it so the ImageDraw only updates when your window resizes. I tried this out and it helps. But really not that much and far from noticable.

Subject: Re: High Performance Drawing
Posted by phirox on Tue, 01 Jan 2008 22:54:21 GMT
View Forum Message <> Reply to Message

Ok I was further playing and found out that you can use DrawingDraw for much faster performance. Basically this solves the problem imo.

Use the following piece of code in your Paint routine:

```
DrawingDraw dd(GetSize());

dd.DrawRect(GetSize(), White);
dd.DrawLine(x,y,150,150,50,Blue);

draw.DrawDrawing(0, 0, GetSize().cx, GetSize().cy, dd);
```

You can turn on backpainting for a flicker-free version, with a small performance hit.

Subject: Re: High Performance Drawing
Posted by cbpporter on Wed, 02 Jan 2008 17:11:04 GMT
View Forum Message <> Reply to Message

OK, so I made some tests in Delphi and it seems that the current computer I'm using has a very poor graphics card (which theoretically should be good enough for simple bliting, being an ATI 9250) and this is the main reason the performance is so bad.

But still, U++ bliting speed is lower than the one in VCL. I still need to do some benchmarks, but this is certainly an area which could theoretically be improved. I looked over Image, and I saw nothing which looks slower, except the fact that the image is always 32 bits and this could be potentially slow on other bit depths (not the case here).

I also ran an older app of mine which had the same issue and in which I used a a combination of smart bliting and xor updates, which is very fast, but I don't have the sources right now and I don't remember how I done it exactly, but I'm sure I will be able to reproduce it and I hope I can apply it to U++.

Quote:Ok I was further playing and found out that you can use DrawingDraw for much faster performance. Basically this solves the problem imo.
Yes, you a right. This was actually to be expected, seeing how DrawingDraw handles stuff, but I need an image buffer, where I can apply custom effects, so DrawingDraw is not good for my current needs.

On a note to Mirek, ImageDraw is NoCopy, it can not be initialized with an Image and it does not even retrieve the size. This pretty much limits it's use and it is not possible to create a persistent backbuffer.

Anyway, I'll continue to reproduce that drawing method from that Delphi app, which manages to draw a bitmap almost as large as the screen resolution and still do mouse based updates in real time even on a Celeron 900Mhz. I really don't want to introduce a DirectX dependency only for this task.

Subject: Re: High Performance Drawing
Posted by mirek on Wed, 02 Jan 2008 21:00:26 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Wed, 02 January 2008 12:11
On a note to Mirek, ImageDraw is NoCopy, it can not be initialized with an Image and it does not even retrieve the size. This pretty much limits it's use and it is not possible to create a persistent backbuffer.

Yes. It is because Image is always rather considered a "client-side" matrix of pixels. Therefore sort of planned solution is to introduce software rendering into this matrix.

Maybe "BackDraw" represents what you need better for now.

Mirek

Subject: Re: High Performance Drawing
Posted by phirox on Sat, 05 Jan 2008 18:55:01 GMT
View Forum Message <> Reply to Message

Edit: Ok problem was actually quite simply solved
As a last resort I tried to compile it without debugging, it worked well then. The solution: Turn off XSyncronization by disabling the flag SYNCHRONIZE in X11App.cpp

---OLD---
I've actually come across another issue regarding high speed drawing. My program had no real speed problems until I upgraded to the latest development release.

The lay down: It looks like it doesn't effect Windows just X11. No problems with 701-dev1 and 2007.1, it starts to go really slow with 708-dev2b(I have not tried versions between those).

I've added a test case, the difference in paint update speed is huge.

I've tried for 2 days to find the core of the problem myself, but I'm at a loss. It doesn't seem to be the palette changes, or the drawlock.

Only workaround I found to make it just as fast as 2007.1 is to comment out this line in the function Draw::DrawRectOp
XFillRectangle(Xdisplay, dw, gc, x + actual_offset.x, y + actual_offset.y, cx, cy);
Which is really weird, I'm guessing something changed for the Xdisplay; dw; and gc variable or something.

File Attachments
1) drawtest.zip, downloaded 518 times

Subject: Re: High Performance Drawing
Posted by mirek on Sun, 06 Jan 2008 07:54:02 GMT
View Forum Message <> Reply to Message

phirox wrote on Sat, 05 January 2008 13:55Edit: Ok problem was actually quite simply solved
As a last resort I tried to compile it without debugging, it worked well then. The solution: Turn off XSyncronization by disabling the flag SYNCHRONIZE in X11App.cpp

Well, perhaps I should comment that out even in debug mode....

Mirek

Subject: Re: High Performance Drawing
Posted by cbpporter on Wed, 23 Jan 2008 14:06:35 GMT
View Forum Message <> Reply to Message

I'm still trying to solve this, and it seams that the only way to do it is to find an old computer, install Delphi on it and do some serious debugging on my old drawing lib.

In the mean-time, there are some experiments I would like to try. Is there a way in U++ to paint a component outside of paint method? I remember the old rule, which says: never ever cal Refresh or invalidate rect on you custom drawn controls without backbuffering, because they will flicker like hell. Just get the HDC and do a smart update. I wonder if it hold true for U++ to?

Subject: Re: High Performance Drawing
Posted by cbpporter on Wed, 23 Jan 2008 15:56:41 GMT
View Forum Message <> Reply to Message

So basically I need to update stuff on mouse move, and I came to this three cases that I tested:

1. When drawing a line in paint method, it is fast, but due to the need of clearing the previous position, it is flickery. Not using DrawRect, only updating minimum zone is less flickery, but still bad.

2. When pre-caching the final result in and ImageDraw, it is very slow, but no flicker.

And the strange one:

3. When instead of drawing a line, I'm resizing a control, I get no flicker and good performance. This case doesn't have the Refresh call, and also, the control itself is probably backbuffered, that's why no flicker. But why is the performance still good?

Subject: Re: High Performance Drawing
Posted by mirek on Wed, 23 Jan 2008 16:01:40 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Wed, 23 January 2008 10:56So basically I need to update stuff on mouse move, and I came to this three cases that I tested:

1. When drawing a line in paint method, it is fast, but due to the need of clearing the previous position, it is flickery. Not using DrawRect, only updating minimum zone is less flickery, but still bad.

2. When pre-caching the final result in and ImageDraw, it is very slow, but no flicker.

Maybe I have lost the context (again?), but what is wrong with BackPaint?

Quote:
And the strange one:

3. When instead of drawing a line, I'm resizing a control, I get no flicker and good performance. This case doesn't have the Refresh call, and also, the control itself is probably backbuffered, that's why no flicker. But why is the performance still good?

If widget is transparent, very complex code is used to paint it. Is it transparent?

Mirek

Subject: Re: High Performance Drawing
Posted by cbpporter on Wed, 23 Jan 2008 16:10:18 GMT
View Forum Message <> Reply to Message

luzr wrote on Wed, 23 January 2008 17:01
Maybe I have lost the context (again?), but what is wrong with BackPaint?

Nothing wrong with BackPaint. IMO, it works just great.

But I need a GUI which is used to create quite complex images, one step at a time, including alpha-blending and antialiasing, so I'm basically stucked with using a manual backbuffer, because I need to both read and write raw pixel information. Painting that buffer is too slow.

These shapes were organized an a tree structure, and when I want to modify one of them by mouse interaction, I would deactivate the old element, redraw the image without it, and during drag operations with mouse only use a sketch of that given figure, even something as simple as RectBand (I don't remember how the rubber band rectangle from U++ was called).

Quote:If widget is transparent, very complex code is used to paint it. Is it transparent?
Not transparent, but still excellent performance and no flicker. Like a full screen populated

ArrayCtrl.

Subject: Re: High Performance Drawing
Posted by cbpporter on Fri, 01 Feb 2008 10:02:44 GMT
View Forum Message <> Reply to Message

OK! I found a solution which seems to work pretty well. The component still has an Image blit in it and is still a little slow (scrolling to container of the image is not as fast as before, when I didn't use the Image), but now for time critical refreshes (like on mouse move), I'm using RectTracker.

Two things are left to do:
1. Create other trackers with others shapes.
2. Need to add bidirectional conversions between Image and ImageDraw.

Subject: Re: High Performance Drawing
Posted by cbpporter on Fri, 01 Feb 2008 15:27:57 GMT
View Forum Message <> Reply to Message

I made some test, and using a preallocated buffer greatly improves performance, at the price of a constant memory overhead. But we are talking about specific performance needs in a graphical application, so this is a small price to pay.

There is still a lot of functionality that I need to added Image, especially in-place resizing. I'll try to add these over the weekend.