Subject: Draw::DrawImageOp optimization bug
Posted by Tom1 on Thu, 27 Nov 2008 12:52:30 GMT
View Forum Message <> Reply to Message

Hi,

There is a problem when printing large images. Disabling the "if((cx > 2000 || cy > 2000) ..." -triggered image printing optimization in Draw.cpp / Draw::DrawImageOp(), makes the problem disappear.

The optimized printout reveals narrow white uncovered gaps (both horizontal and vertical) between rendered blocks:

This can be verified when printing images with a high resolution printer or just with "Microsoft XPS Document Writer". The XPS document, when viewed with Microsoft XPS Viewer shows clearly the problem.

Also, when printing to a color laser printer (Xerox Phaser 6200), the coloring of the printed image becomes uneven along the optimized blocks. This also is fixed when printing without the optimization.

// Tom

File Attachments

1) image_error.png, downloaded 1389 times

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Fri, 28 Nov 2008 07:56:17 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Thu, 27 November 2008 07:52Hi,

There is a problem when printing large images. Disabling the "if((cx > 2000 || cy > 2000) ..." -triggered image printing optimization in Draw.cpp / Draw::DrawImageOp(), makes the problem disappear.

The optimized printout reveals narrow white uncovered gaps (both horizontal and vertical) between rendered blocks:

This can be verified when printing images with a high resolution printer or just with "Microsoft XPS Document Writer". The XPS document, when viewed with Microsoft XPS Viewer shows clearly the problem.

Also, when printing to a color laser printer (Xerox Phaser 6200), the coloring of the printed image becomes uneven along the optimized blocks. This also is fixed when printing without the optimization.

// Tom

Thanks. Looks like we need to work a bit on this.

BTW, can you upload the testing image (and maybe the whole testcase) somewhare?

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Fri, 28 Nov 2008 09:23:32 GMT
View Forum Message <> Reply to Message

I wrote a simple test case that can be used to visualize the problem easily when printing to e.g. Microsoft XPS Document Writer.

```
#include <CtrlLib/CtrlLib.h>

using namespace Upp;

class Testcase2 : public TopWindow{

public:
 typedef Testcase2 CLASSNAME;

 Testcase2(){
  Sizeable();
  MinimizeBox();
  MaximizeBox();
  BackPaint();
 }

 void drawimage(Draw &draw){
  Size sz;
  if(draw.IsPrinter()) sz=draw.GetPagePixels();
  else sz=GetSize();

  ImageBuffer ib(sz.cx,sz.cy);
  for(int y=0;y<sz.cy;y++){
   for(int x=0;x<sz.cx;x++){
    ib[y][x]=(RGBA)Color((16*y/sz.cy)<<4,(16*y/sz.cy)<<4,(16*x/sz.cx)<<4);
```

```
   }
  }

  Rect rect=draw.GetClip();
  draw.DrawRect(rect,Color(255,255,255));
  Image img(ib);
  draw.DrawImage(0,0,img);
  if(!draw.IsPrinter()){
   draw.DrawText(10,10,"Press 'P' to print...",StdFont(),Color(255,255,255));
   draw.DrawText(10,30,"(This really drains your ink/toner cartridge, so try e.g. Microsoft XPS
Document Writer.)",StdFont(),Color(255,255,255));
  }
 }

 void print(){
  PrinterJob *job=new PrinterJob();
  if(job){
   if(job->Execute()){
    Draw &draw=job->GetDraw();
    draw.StartPage();
    drawimage(draw);
    draw.EndPage();
   }
   delete job;
  }
 }

 virtual void Paint(Draw &draw){
  drawimage(draw);
 }

 bool Key(dword key, int count){
  switch(key){
   case K_P:
    print();
    return true;
  }
  return false;
 }

};

GUI_APP_MAIN
{
 Testcase2().Run();
}
```

// Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by rylek on Fri, 28 Nov 2008 12:15:14 GMT
View Forum Message <> Reply to Message

Hi there!

The 2000 x 2000 block limitation is a problematic old hack I used long ago when I had trouble with GDI processing larger areas at once. Some of it was W98-related and is probably long gone, but a few years ago I encountered the same problem when printing a very large raster-based drawing on an A0 cylinder plotter with not much memory where without piecewise upload the plotter wouldn't print anything at all. Some expertise would evidently help here and the blit routine should be if-ed properly perhaps based on some properties of the output device and/or OS version.

Regards

Tomas

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Fri, 28 Nov 2008 12:27:23 GMT
View Forum Message <> Reply to Message

rylek wrote on Fri, 28 November 2008 07:15Hi there!

The 2000 x 2000 block limitation is a problematic old hack I used long ago when I had trouble with GDI processing larger areas at once.

Actually, I am afraid you are not up-to-date about this one...

This is not the "old Image code" (although it can fix the same problem as byproduct).

Current DrawImage really does something new, the RLE compression. Also, but splitting the process to bands, it should help with memory consumption - e.g. imagine printing some documentation from RichText, where screenshots are usually rescaled to much bigger size. Banding pushes them in bands, means we do not need the memory for the "big" results.

And more, "RLE compression" (detecting uniform color blocks) has the potential to reduce the memory footprint even in the printer.

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Fri, 28 Nov 2008 12:29:52 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Fri, 28 November 2008 04:23
```
 void print(){
  PrinterJob *job=new PrinterJob();
  if(job){
   if(job->Execute()){
    Draw &draw=job->GetDraw();
    draw.StartPage();
    drawimage(draw);
    draw.EndPage();
   }
   delete job;
  }
 }
```

Off-topic: Why the 'new' here?

In U++, never use 'new' unless you have a VERY good reason to do so - either you are doing some too low-level or you need 'naked' polymorphy. But as for polymorphy, you can also use 'encapsulated' form with One...

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Fri, 28 Nov 2008 12:54:37 GMT
View Forum Message <> Reply to Message

Still off-topic, but even more so: I started writing applications with C++ in 1988 and for years I often ended up with stack overflow whenever I attempted to use local variables from my classes. So, I learned my lesson and started to use pointers and new/delete instead. A lot of time has passed along the way and this problem with local variables and stack overflows may well have disappeared altogether, but I seem to be stuck with my old habits. Well, maybe I will learn over the next decade or so...

// Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Fri, 28 Nov 2008 13:27:46 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Fri, 28 November 2008 07:54Still off-topic, but even more so: I started writing

applications with C++ in 1988 and for years I often ended up with stack overflow whenever I attempted to use local variables from my classes. So, I learned my lesson and started to use pointers and new/delete instead. A lot of time has passed along the way and this problem with local variables and stack overflows may well have disappeared altogether, but I seem to be stuck with my old habits. Well, maybe I will learn over the next decade or so...

// Tom

I can understand that... However, those old habits do separate you from the main advantage of C++/U++ - automated resource management...

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Fri, 28 Nov 2008 20:02:06 GMT
View Forum Message <> Reply to Message

Well, I have spent last 4 hours with this problem and I really do not know what to think....

See this:

```
#include <CtrlLib/CtrlLib.h>

using namespace Upp;

class Testcase2 : public TopWindow{

public:
 typedef Testcase2 CLASSNAME;

 Testcase2(){
  Sizeable();
  MinimizeBox();
  MaximizeBox();
  BackPaint();
 }

 Image MakeImage2(Size sz) const {
  ImageBuffer ib(sz.cx / 3, sz.cy / 3);
  Fill(ib, LtBlue(), ib.GetLength());
  const Color c[] = { LtBlue() };
  for(int y = 0; y < ib.GetWidth(); y++) {
   for(int x = 0; x < ib.GetHeight(); x++)
    ib[x][y] = c[2 * (x > ib.GetHeight() / 2) + (y > ib.GetWidth() / 2)];
  }
  return ib;
```

```
}

Image MakeImage(Size sz) const {
 ImageDraw iw(sz / 3);
 iw.DrawRect(sz / 3, Blue);
 iw.DrawText(0, 0, "O", Roman(sz.cy / 3).Italic(), Red());
 return iw;
}

Image MakeImage1(Size sz) const {
 ImageBuffer ib(sz);
 for(int y=0;y<sz.cy;y++){
  for(int x=0;x<sz.cx;x++){
   ib[y][x]=(RGBA)Color((16*y/sz.cy)<<4,(16*y/sz.cy)<<4,(16*x/sz.cx)<<4);
  }
 }
 return ib;
}

void print(){
}

virtual void Paint(Draw &w){
 Size sz = GetSize();
 w.DrawRect(GetSize(), White);
 w.DrawImage(0, 0, MakeImage(GetSize()));
 w.DrawText(10,500,"Press 'P' to print image, 'R' to print rectangle test, 'C' to another test...");
 w.DrawText(10,530,"(This really drains your ink/toner cartridge, so try e.g. Microsoft XPS
Document Writer.)");
}

bool Key(dword key, int count){
 PrinterJob job;
 switch(key){
  case K_P:
   if(job.Execute()){
    Draw &draw = job;
    draw.StartPage();
    draw.DrawImage(0, 0, MakeImage(draw.GetPagePixels()));
    draw.EndPage();
   }
   return true;
  case K_R:
   if(job.Execute()){
    Draw &draw = job;
    draw.StartPage();
    bool flag = false;
    for(int y = 0; y < 100; y++)
```

```
      for(int x = 0; x < 100; x++) {
       draw.DrawRect(16 * x, 16 * y, 16, 16, flag ? Blue : Red);
       if((Random() & 31) == 0)
        flag = !flag;
      }
     draw.EndPage();
    }
    return true;
   case K_C:
    if(job.Execute()){
     Draw &draw = job;
     draw.StartPage();
     for(int y = 0; y < 100; y++)
      for(int x = 0; x < 100; x++)
       draw.DrawRect(16 * x, 16 * y, 16, 16, Blue);
     draw.EndPage();
    }
    return true;
  }
  return false;
 }

};

GUI_APP_MAIN
{
 Testcase2().Run();
}
```

Now interesting things happen when you press R or C.

These try to emulate the compression issue. When you do R, you have those line artifacts, but C seems without the issue - and only thing that has changed is the color.

Is not it really weird? (Note that it is not even related to Image).

It can also be seen with P... The areas that are "full line" are OK.

Well, I have also tried to mitigate the issue by "overpainting" rectangle (adding to height/width), but interestigly that seems to have produced other issues.

OK, enough for today, next tomorrow. We really need this working BTW.

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Sat, 29 Nov 2008 11:54:28 GMT
View Forum Message <> Reply to Message

luzr wrote on Fri, 28 November 2008 15:02
Well, I have also tried to mitigate the issue by "overpainting" rectangle (adding to height/width), but interestigly that seems to have produced other issues.

- just try to extend values in DrawRect

```
draw.DrawRect(16 * x, 16 * y, 19, 19, flag ? Blue : Red);
```

Frankly, I am in dead end. I believe that the whole issue is just a driver bug, and I do not see any reasonable workaround.

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Mon, 01 Dec 2008 10:25:49 GMT
View Forum Message <> Reply to Message

I looked into this problem from another aspect: I shrinked the rectangle by one pixel from each edge. This proved that in the RLE code the edges of the rectangles and sub-images are exactly accurate. What happens in XPS, is that the edges show sub-pixel level inaccuracies that are internal to the XPS. This can be very accurately visualized in XPS Viewer using 5000% zooming. So, it is not U++ fault. XPS Viewer also internally softens the rasters when zooming in, and then another set of problems arise with the rectangles and sub-images: They do not quite fit together with soft and sharp edges side-by-side.

On the other hand, the unrelated problem with uneven colors in Xerox Phaser 6200, might have something to do with color spaces being different for images and vector elements causing weird color effects. I have no proof of this, but still, the printout gets corrected when this optimization is commented out.

I suggest an option flag -- maybe even enabled by default -- for skipping the optimization for large images.

// Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Mon, 01 Dec 2008 11:12:31 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Mon, 01 December 2008 05:25I looked into this problem from another aspect: I shrinked the rectangle by one pixel from each edge. This proved that in the RLE code the edges of the rectangles and sub-images are exactly accurate. What happens in XPS, is that the edges show sub-pixel level inaccuracies that are internal to the XPS. This can be very accurately visualized in XPS Viewer using 5000% zooming. So, it is not U++ fault. XPS Viewer also internally softens the rasters when zooming in, and then another set of problems arise with the rectangles and sub-images: They do not quite fit together with soft and sharp edges side-by-side.

OK, that would mean we should attempt for "overpaint" solution and ignore those weird artifacts?

Quote:
On the other hand, the unrelated problem with uneven colors in Xerox Phaser 6200, might have something to do with color spaces being different for images and vector elements causing weird color effects. I have no proof of this, but still, the printout gets corrected when this optimization is commented out.

Ooops.

Quote:
I suggest an option flag -- maybe even enabled by default -- for skipping the optimization for large images.

Well, that is the last resort solution...

BTW, at least, we should not be afraid to optimize white areas, correct?

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Mon, 01 Dec 2008 11:41:17 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Mon, 01 December 2008 05:25
On the other hand, the unrelated problem with uneven colors in Xerox Phaser 6200, might have something to do with color spaces being different for images and vector elements causing weird color effects. I have no proof of this, but still, the printout gets corrected when this optimization is commented out.

BTW, it is interesting that those rects are being painted using PatBlt (basically, a pixel-pushing api).

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Mon, 01 Dec 2008 11:49:22 GMT
View Forum Message <> Reply to Message

After seeing all this happening with basically simple rasters and rectangles, I'm not quite sure anymore what really will happen if even the white areas get optimized... How do they cover the stuff drawn behind those areas to be covered with white?

Of course, if drawing the optimized image starts with a single white DrawRectOp() covering the entire image rect, and then continues by adding the non-white sub-images on top of that should work... but only for OPAQUE pictures.

UPDATE:

Quote:
BTW, it is interesting that those rects are being painted using PatBlt (basically, a pixel-pushing api).

Correct, but: PatBlt uses current brush just like Rectangle and other vector functions with fill capability, so the color mapping mechanism must be the same.

BTW: Is PatBlt more efficient that Rectangle?

// Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Mon, 01 Dec 2008 11:57:27 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Mon, 01 December 2008 06:49After seeing all this happening with basically simple rasters and rectangles, I'm not quite sure anymore what really will happen if even the white areas get optimized... How do they cover the stuff drawn behind those areas to be covered with white?

Of course, if drawing the optimized image starts with a single white DrawRectOp() covering the entire image rect, and then continues by adding the non-white sub-images on top of that should work... but only for OPAQUE pictures.

// Tom

Just a sidenote: I must see stubborn desperately trying to keep this optimization, but:

- some of my existing applications would have troubles without it, this is essential to e.g. printing .jpges directly to the printer, without the need of rescaling them to 500MB.

- this concept plays an important role in the future, when we will need to print svg pictures. I expect that this will work very well with most bussines oriented graphivs.

I now believe that we can handle artifacts issue, at least on printers.

So the problem now is the color mismatch..

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Mon, 01 Dec 2008 12:12:42 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Mon, 01 December 2008 06:49
Of course, if drawing the optimized image starts with a single white DrawRectOp() covering the entire image rect, and then continues by adding the non-white sub-images on top of that should work... but only for OPAQUE pictures.

I would not hope too much that any existing printer driver can really handle non-OPAQUE picture, meanwhile, it should be simple to avoid optimization for them too...

Quote:
BTW, it is interesting that those rects are being painted using PatBlt (basically, a pixel-pushing api).

Correct, but: PatBlt uses current brush just like Rectangle and other vector functions with fill capability, so the color mapping mechanism must be the same.
[/quote]

Yes, likely yes.

Quote:
BTW: Is PatBlt more efficient that Rectangle?

I do not believe so. But AFAIK, they are rounded to physical resolution differently. In fact, that is most likely the source of our problem.

Mirek

Subject: Re: Draw::DrawImageOp optimization bug

Posted by mirek on Mon, 01 Dec 2008 12:14:56 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Mon, 01 December 2008 05:25
On the other hand, the unrelated problem with uneven colors in Xerox Phaser 6200, might have something to do with color spaces being different for images and vector elements causing weird color effects. I have no proof of this, but still, the printout gets corrected when this optimization is commented out.

Something simple to try: Create uniform color Image (e.g. using CreateImage(Size sz, Color color) then print it alongside the Rect with the same color.

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Mon, 01 Dec 2008 12:50:54 GMT
View Forum Message <> Reply to Message

luzr wrote on Mon, 01 December 2008 13:57
Just a sidenote: I must see stubborn desperately trying to keep this optimization, but:

- some of my existing applications would have troubles without it, this is essential to e.g. printing .jpges directly to the printer, without the need of rescaling them to 500MB.

- this concept plays an important role in the future, when we will need to print svg pictures. I expect that this will work very well with most bussines oriented graphivs.

I now believe that we can handle artifacts issue, at least on printers.

So the problem now is the color mismatch..

Mirek

No need to apologize, you're the chief -- not me.

I faced the same 'oversized printout' problem with images and went around it using ::StretchDIBits() GDI API. This effectively shrinked the amount of data transferred to the printer. I guess there are very few printers really capable of printing 24 or 32 bit colors at the full resolution so the colors visible to the eye are really sums of many adjacent pixels in both horizontal and vertical directions. Therefore, I use lower resolution for color and gray scale images to be transferred to the printer and print only the vector graphics at the full resolution on top of that.

I'm really dependent on this for many of my applications, but I have been able to do it without modifications U++ itself, so I have not been pressuring you to take it in. If it was to be integrated to U++ Draw::, I would prefer an option flag for natively stretched images, instead of U++ based

pre-stretching when expanding of images was needed.

// Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Mon, 01 Dec 2008 13:25:29 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Mon, 01 December 2008 07:50
I faced the same 'oversized printout' problem with images and went around it using ::StretchDIBits() GDI API. This effectively shrinked the amount of data transferred to the printer.

Those JPEGs are 600dpi scans of A4 pages...

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Mon, 01 Dec 2008 13:37:05 GMT
View Forum Message <> Reply to Message

I made a simple test drawing two adjacent 100x100 dots squares filled with the same Color(180,180,0). The first one was an Image and the second was a Rectangle.

For Windows Vista with standard printer driver: I can confirm that the colors do not match on Xerox Phaser 6200. Not even close. The image is much lighter than the rectangle. I tried with ICM (Image color management) enabled and disabled, and with other options: No change.

For Windows XP it first appeared that nothing is different compared to Vista, but after changing a parameter (Color Correction: Press Match = Commercial Press), I got the colors finally to match. (This parameter can not be set for Windows Vista though, so no luck there.)

To sum it up, the color space may indeed be different for raster and vector entities within the same device context. Not a very nice feature from the optimization point of view.

// Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Mon, 01 Dec 2008 13:52:12 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Mon, 01 December 2008 08:37
To sum it up, the color space may indeed be different for raster and vector entities within the same device context. Not a very nice feature from the optimization point of view.

Indeed

There are still "rat in the corner" means to overcome it.

The simple one comes to mind is to reuse small(er) uniform color Image to draw rectangles instead. E.g. something like 16x16 block.

Either tile it or stretch it.

Damn it. And I thought that the issue is solved, years ago

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Tue, 02 Dec 2008 18:15:49 GMT
View Forum Message <> Reply to Message

Update:

I have performed the first test on my brand new Epson SX100 - and the result is perfect, without any changes to the code

Anyway, I plan to do this:

- print in printer resolution, increased to be <1000dpi (that should solve the "division lines" problem)

- only optimize out white areas (non-matching color problem)

I now think that I will add to Draw something like:

```
Pointf BeginNativeResolution();
void   EndNativeResolution();
```

with Pointf containing constants to convert from dots to native pixels.

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Tue, 02 Dec 2008 20:16:21 GMT
View Forum Message <> Reply to Message

Hi Mirek,

I'm not quite sure what you mean by "increased to be <1000dpi". Anyway, I think those changes will increase usability of both old printers and the yet to be published ones. I hope you make the pixels square (horizontal dpi = vertical dpi) even in the native resolution mode when the native pixels are not square. Otherwise, I suggest a flag for requesting square pixels in the call to BeginNativeResolution(bool useSquarePixelsFlag).

I will test it and report the results as soon as I get your solution from the SVN.

(Off the record: I did some testing today to print the RLE rectangles with StretchDIBits() and the results were pretty promising, but not quite perfect. 1x1 pixel source raster does not work correctly. It needs to be at least 1x2 for some reason. There were also still slight coverage problems with XPS when zoomed out -- although not nearly as bad as with rectangles.)

Thanks and regards,

Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Tue, 02 Dec 2008 20:46:24 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Tue, 02 December 2008 15:16Hi Mirek,

I'm not quite sure what you mean by "increased to be <1000dpi". Anyway, I think those changes will increase usability of both old printers and the yet to be published ones. I hope you make the pixels square (horizontal dpi = vertical dpi) even in the native resolution mode when the native pixels are not square.

Uh, it would not be native mode then?!

Anyway, more information: For our "corpus delicti", Epson, the resolutions returned from GDI are either 360dpi or 720dpi (1440/5xxx is just marketing bluff).

Means I will simply go MM_TEXT mode as native.

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Wed, 03 Dec 2008 11:14:50 GMT
View Forum Message <> Reply to Message

luzr wrote on Tue, 02 December 2008 15:46Tom1 wrote on Tue, 02 December 2008 15:16Hi Mirek,

I'm not quite sure what you mean by "increased to be <1000dpi". Anyway, I think those changes will increase usability of both old printers and the yet to be published ones. I hope you make the pixels square (horizontal dpi = vertical dpi) even in the native resolution mode when the native pixels are not square.

Uh, it would not be native mode then?!

Anyway, more information: For our "corpus delicti", Epson, the resolutions returned from GDI are either 360dpi or 720dpi (1440/5xxx is just marketing bluff).

Means I will simply go MM_TEXT mode as native.

Mirek

OK, done. Seems OK so far (althought in fact, there is no visible difference on my Epson).

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Wed, 03 Dec 2008 14:03:31 GMT
View Forum Message <> Reply to Message

Hi Mirek,

The printing seems to work now correctly for both XPS and also the older Epson printer where we had the scaling and other problems. Further on, the Xerox Phaser 6200 colors are even now within the image. Good work!

One thing popped up when reading the code. The BeginNative() call is followed by a conditional return without corresponding EndNative(). This may cause trouble in some cases. Maybe it should be written the other way around:

```
...
LTIMING("DrawImageOp");
if(IsNull(src))
 return;
BeginNative();
Native(x, y);
```

Native(cx, cy);
...

Thanks and best regards,

Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Wed, 03 Dec 2008 14:21:59 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Wed, 03 December 2008 09:03Hi Mirek,

The printing seems to work now correctly for both XPS and also the older Epson printer where we had the scaling and other problems. Further on, the Xerox Phaser 6200 colors are even now within the image. Good work!

One thing popped up when reading the code. The BeginNative() call is followed by a conditional return without corresponding EndNative(). This may cause trouble in some cases. Maybe it should be written the other way around:

...
LTIMING("DrawImageOp");
if(IsNull(src))
 return;
BeginNative();
Native(x, y);
Native(cx, cy);
...

Thanks and best regards,

Tom

Thanks!

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Wed, 03 Dec 2008 14:36:20 GMT

Mirek,

One more thing: GetPagePixels returns the size as 600 dpi dots now even in the native mode. It would be very helpful if it returned the pixels according to the current addressing mode, i.e. native or dots mode.

Update: Also, would it be possible to return the currently effective DPI by the GetPixelsPerInch() in the same way? This would simplify using the same code for both printing and screen view.

Update 2: The code required for these:

Draw.h, Add/Change:

```
Size   pageDots;

Size  GetPagePixels() const { return native?pagePixels:pageDots; }
Size  GetPixelsPerInch() const { return native?nativeDpi:inchPixels; }
```

DrawWin32.cpp, Change:

```
void Draw::LoadCaps() {
...
pageDots = pagePixels = GetSizeCaps(HORZRES, VERTRES);
...
}

...

void PrintDraw::InitPrinter()
{
...
pageDots.cx = 600 * pagePixels.cx / nativeDpi.cx;
pageDots.cy = 600 * pagePixels.cy / nativeDpi.cy;
...
}
```

// Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Wed, 03 Dec 2008 16:11:25 GMT

Hi,

I ran some more tests on the native mode and discovered that the following should be changed in order to get correct scaling for the images in the native mode too:

1. There should not be switching to native mode and back in the Draw.cpp/Draw::DrawImageOp(). Also, there should not be any Native() translations for the coordinates.

2. The same goes probably for the DrawData.cpp/Draw::DrawDataOp(). No switching to native and back, and no coordinate translations.

Please remove:

```
BeginNative();
Native(x, y);
Native(cx, cy);
..
EndNative();
```

from both of the above functions.

The printing works correctly without these switchings in both native and dots mode. (By selecting the native mode, the user indicates that they wish to use the native coordinate space for all coordinates and create the content with that fact in mind.)

Best regards,

Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Thu, 04 Dec 2008 09:19:22 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Wed, 03 December 2008 09:36Mirek,

One more thing: GetPagePixels returns the size as 600 dpi dots now even in the native mode. It would be very helpful if it returned the pixels according to the current addressing mode, i.e. native or dots mode.

Update: Also, would it be possible to return the currently effective DPI by the GetPixelsPerInch() in the same way? This would simplify using the same code for both printing and screen view.

Well, at the time, it was intentional design. You have GetNativeDpi there.

I have to think whether the proposed change is desirable.

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Thu, 04 Dec 2008 09:22:05 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Wed, 03 December 2008 11:11Hi,

I ran some more tests on the native mode and discovered that the following should be changed in order to get correct scaling for the images in the native mode too:

1. There should not be switching to native mode and back in the Draw.cpp/Draw::DrawImageOp(). Also, there should not be any Native() translations for the coordinates.

2. The same goes probably for the DrawData.cpp/Draw::DrawDataOp(). No switching to native and back, and no coordinate translations.

Uh oh. DrawImage was the reason we have started all this stuff anyway.

Why do you suggest to throw all of that now?

Note that it is at least supposed to fix those line artifacts problem in the first place.

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Thu, 04 Dec 2008 10:07:28 GMT
View Forum Message <> Reply to Message

Quote:
Uh oh. DrawImage was the reason we have started all this stuff anyway.

Why do you suggest to throw all of that now?

Note that it is at least supposed to fix those line artifacts problem in the first place.

I do not wish to throw away the BeginNative/EndNative/Native functions. Definitely not. They are really the key to use the native resolution in printing. That's great and they should stay there.

The problem is that DrawImageOp and DrawDataOp use the BeginNative/EndNative pair and the Native translations as if the coordinates input would be dots in all cases. This makes the coordinate space different for vector objects and the image objects.

Based on your code, I assume, you wanted to put the image to the printer on the native resolution. What happened was that if native mode was already on, the Native() calls did a second re-mapping of coordinates causing error in position and scale.

Basically switching the mode and the Native() calls would have been OK, if a check of the native recursion level was done and the Native() calls were conditional (if(native==1)) after switching. Anyway, the correct mapping of pixels to coordinate space was already achieved with the change that skipped ::SetDIBitsToDevice() call and used some other 'bitmap thing'. So I concluded that those calls are not needed in DrawImageOp and DrawDataOp at all. I removed the calls from there, tested it and it worked. Both in native and in dots mode.

Now, if you do these changes I suggest, I guess the situation will be that we should have a Draw:: for printing that by default is mapped with 600 dpi points and works OK with all printers we have tested so far. Additionally, if BeginNative is called, it will completely switch to printers native dpi mode and provides a consistent printing area at the printers native resolution instead of the 600 dpi.

The changes to GetPagePixels and GetPixelsPerInch support this. They will provide the correct values for the drawing area also after switching to native mode. Those changes will not break dots mode in any way.

Regards,

Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Thu, 04 Dec 2008 10:36:49 GMT
View Forum Message <> Reply to Message

Yes, thinking about the issue, I have came to the same conclusion.

The key is to do Native conversion only if BeginNative was called for the first time.

Interesting question is whether we should add such logic directly into "Native" conversion methods.

Something like:

return native == 1 && inchPixels != nativeDpi ? iscale(x, nativeDpi.cx, 600) : x;

What do you think? Maybe it is a little bit too 'automatic', but I do not really see any flaws (yet

The logic of Native methods then could be described as "what was your value before BeginNative now gets converted to the value you need inside BeginNative-EndNative block".

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Thu, 04 Dec 2008 11:09:04 GMT
View Forum Message <> Reply to Message

I think I would not automate it. It will just require more thinking when expanding/debugging the code later -- especially if there are multiple levels of calls involved.

I think for a user of Draw::, there is no need to call the Native() functions at all. They will most probably either use dots by default and not care about any native stuff OR they will call BeginNative() once in the beginning of the page and then use that coordinate system for all of the page content. The Native() calls will probably be left for the uses inside the Draw:: itself. (Unless, of course, someone is doing something quite extraordinary that I can't imagine right now.) I think we should assume the user of Draw:: to know which mode he selected previously and behave accordingly.

// Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Thu, 04 Dec 2008 11:36:31 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Thu, 04 December 2008 06:09I think I would not automate it. It will just require more thinking when expanding/debugging the code later -- especially if there are multiple levels of calls involved.

I think for a user of Draw::, there is no need to call the Native() functions at all. They will most probably either use dots by default and not care about any native stuff OR they will call BeginNative() once in the beginning of the page and then use that coordinate system for all of the page content. The Native() calls will probably be left for the uses inside the Draw:: itself. (Unless, of course, someone is doing something quite extraordinary that I can't imagine right now.) I think we should assume the user of Draw:: to know which mode he selected previously and behave accordingly.

// Tom

Well, that would mean test in DrawImage / DrawData, right?

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Thu, 04 Dec 2008 12:09:23 GMT
View Forum Message <> Reply to Message

Yes, this would mean the 'if(native==1)' testing in DrawImage / DrawData.

I must emphasize that switching to native in DrawImage/DrawData is not required at all unless you wish to put back the SetSurface (::SetDIBitsToDevice) based image printing to Image::Data::Paint().

// Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Thu, 04 Dec 2008 14:53:04 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Thu, 04 December 2008 07:09Yes, this would mean the 'if(native==1)' testing in DrawImage / DrawData.

I must emphasize that switching to native in DrawImage/DrawData is not required at all unless you wish to put back the SetSurface (::SetDIBitsToDevice) based image printing to Image::Data::Paint().

// Tom

I believe we still might need it to overcome horizontal line artifacts.

And the quality should be now better, as Image is rescaled just once, directly to printer resolution.

Anyway, DrawImage/Data fixed.

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Thu, 04 Dec 2008 15:07:10 GMT
View Forum Message <> Reply to Message

luzr wrote on Thu, 04 December 2008 16:53
I believe we still might need it to overcome horizontal line artifacts.
And the quality should be now better, as Image is rescaled just once, directly to printer resolution.

Anyway, DrawImage/Data fixed.

Mirek

Doing image rescaling just once in dots mode is a very good reason indeed for your approach, I agree.

What about GetPagePixels and GetPixelsPerInch updates? Can you throw them in? I assume nobody currently using them would expect to get different values from them than what the currently active coordinate space uses.

Thanks,

Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Thu, 04 Dec 2008 16:46:52 GMT
View Forum Message <> Reply to Message

Done.

I hope our little printing crysis is thus over

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Fri, 05 Dec 2008 10:05:41 GMT
View Forum Message <> Reply to Message

Mirek,

Thank you for the latest fixes. I have tested with all my real printers with success using both dots and native mode. Now, I will also involve my client with the Epson printer to verify the results on that particular hardware. I would not expect any trouble on that, as the preview is OK.

With Microsoft XPS Document Writer + Viewer, there is still horizontal lines if the image gets even partly optimized. Over-painting the image segments does not help. (This can be seen if the Testcase image (O) is drawn with red on white instead of red on blue.) I assume this happens because the XPS Viewer smooths the image and the intensity at the edge of each colored image segment is lower than the pixel color due to smoothing. Therefore, each new stripe paints a narrow line with lighter color.

I suppose, the RLE optimization of images is just impossible with XPS but luckily appears to work well for real printers after the 'optimize-white-only' -change. I guess, there is just the 'disable RLE optimization' -flag left as an option to survive printing to XPS.

Overall, the changes have improved the printing sub-system compatibility and printing quality. Especially the native mode has been an extremely welcome feature.

Thanks and regards,

Tom

Subject: Re: Draw::DrawImageOp optimization bug
Posted by mirek on Fri, 05 Dec 2008 10:20:11 GMT
View Forum Message <> Reply to Message

Well, regarding XPS, we have came to conclusion it is in fact XPS bug. I plan to ignore this problem for now. AFAIK, not many are using XPS anyway.

(It would be however interesting to check how things behave with PDF

Thanks for all of your work. Testing with all these printers is a huge contribution of its own

Mirek

Subject: Re: Draw::DrawImageOp optimization bug
Posted by Tom1 on Fri, 05 Dec 2008 11:13:37 GMT
View Forum Message <> Reply to Message

I agree on dropping XPS for now.

Thank you and have a nice weekend,

Tom
