Subject: How does CtrlCore Image::Data::PaintImp work?
Posted by fudadmin on Mon, 24 Jan 2011 15:32:13 GMT
View Forum Message <> Reply to Message

sorry for my loose thoughts and sorry if some documentation does exist. If that's the case, please refer me to it. But I couldn't find anything useful.

I am trying to write ImageMac.cpp and I need some help.
As the topic name says I can't grasp and I need a general idea of "How does CtrlCore Image::Data::PaintImp work?"

1. Firstly, what are ResData?
my guesses (involves some optimisation - reusing?) :
 mouse pointer = cursor cheat, parts of the image being painted, or other resources like e.g brushes, masks, all of them combined (but how, in which cases?) ?

2. Why then ResCount>512 for X11?
And why for win32 - int max = IsWinNT() ? 250 : 100; ?
(see my comments below in the code.)

3. Of course, one of the main questions what would be the ResCount limit for Mac?

for X11:
```
int  Image::Data::GetResCountImp() const
{
 return !!Sys().picture + !!Sys().picture8;
}
```

4. image converted to => picture + mask?

for win32:
```
int  Image::Data::GetResCountImp() const
{
 SystemData& sd = Sys();
 return !!sd.hbmp + !!sd.hmask + !!sd.himg;
}
```

5. image converted to => bitmap + mask + ??? ? (is this related somehow to DIB used somewhere? that's why the difference with X11?)

6. Also, what does
```
 Unlink();
 LinkAfter(ResData);
```
do?

7. How does ImageDraw::Section relate to Image::Data::PaintImp? Or what is ImageDraw::Section?

Just a code reference below.

From ImageX11.cpp:

```
void Image::Data::PaintImp(SystemDraw& w, int x, int y, const Rect& src, Color c)
{
 GuiLock __;
 SystemData& sd = Sys();  //Sys holds the image data?
 while(ResCount > 512) {  //what are ResData? - mouse pointer = cursor cheat ? why 512
  Image::Data *l = ResData->GetPrev();
  l->SysRelease();
  l->Unlink();
 }
 x += w.GetOffset().x;
 y += w.GetOffset().y;
 Size sz = buffer.GetSize();
 int  len = sz.cx * sz.cy;
 Rect sr = src & sz;
 Size ssz = sr.Size();
 if(sr.IsEmpty())
  return;
 if(GetKind() == IMAGE_EMPTY)
  return;
 if(GetKind() == IMAGE_OPAQUE && !IsNull(c)) {
  w.DrawRect(x, y, sz.cx, sz.cy, c);
  return;
 }
 if(GetKind() == IMAGE_OPAQUE && paintcount == 0 && sr == Rect(sz)) {
  SetSurface(w, x, y, sz.cx, sz.cy, buffer);
  paintcount++;  //what does paintcount do?
  return;
 }
 Unlink();
 LinkAfter(ResData);
 if(IsNull(c)) {
  if(!sd.picture) {
   bool opaque = GetKind() == IMAGE_OPAQUE;
   Pixmap pixmap = XCreatePixmap(Xdisplay, Xroot, sz.cx, sz.cy, opaque ? 24 : 32);
   sd.picture = XRenderCreatePicture(
    Xdisplay, pixmap,
      XRenderFindStandardFormat(Xdisplay, opaque ? PictStandardRGB24
                                          : PictStandardARGB32),
      0, 0
   );
   ResCount++;
   XImage ximg;
   sInitXImage(ximg, sz);
   ximg.bitmap_pad = 32;
```

```
  ximg.bytes_per_line = 4 * sz.cx;
  ximg.bits_per_pixel = 32;
  ximg.blue_mask = 0x00ff0000;
  ximg.green_mask = 0x0000ff00;
  ximg.red_mask = 0x000000ff;
  ximg.bitmap_unit = 32;
  ximg.data = (char *)~buffer;
  ximg.depth = opaque ? 24 : 32;
  XInitImage(&ximg);
  GC gc = XCreateGC(Xdisplay, pixmap, 0, 0);
  XPutImage(Xdisplay, pixmap, gc, &ximg, 0, 0, 0, 0, sz.cx, sz.cy);
  XFreeGC(Xdisplay, gc);
  XFreePixmap(Xdisplay, pixmap);
  PaintOnlyShrink();
 }
 XRenderComposite(Xdisplay, PictOpOver,
                  sd.picture, 0, XftDrawPicture(w.GetXftDraw()),
                  sr.left, sr.top, 0, 0, x, y, ssz.cx, ssz.cy);
}
else {
 ASSERT(!paintonly);
 if(!sd.picture8) {
  Pixmap pixmap = XCreatePixmap(Xdisplay, Xroot, sz.cx, sz.cy, 8);
  sd.picture8 = XRenderCreatePicture(Xdisplay, pixmap,
                                     XRenderFindStandardFormat(Xdisplay, PictStandardA8),
                                     0, 0);
  ResCount++;
  Buffer<byte> ab(len);
  byte *t = ab;
  const RGBA *s = buffer;
  const RGBA *e = s + len;
  while(s < e)
   *t++ = (s++)->a;
  XImage ximg;
  sInitXImage(ximg, sz);
  ximg.data            = (char *)~ab;
  ximg.bitmap_unit     = 8;
  ximg.bitmap_pad      = 8;
  ximg.depth           = 8;
  ximg.bytes_per_line  = sz.cx;
  ximg.bits_per_pixel  = 8;
  XInitImage(&ximg);
  GC gc = XCreateGC(Xdisplay, pixmap, 0, 0);
  XPutImage(Xdisplay, pixmap, gc, &ximg, 0, 0, 0, 0, sz.cx, sz.cy);
  XFreeGC(Xdisplay, gc);
  XFreePixmap(Xdisplay, pixmap);
 }
 XRenderComposite(Xdisplay, PictOpOver,
```

```
                sGetSolidFill(c), sd.picture8, XftDrawPicture(w.GetXftDraw()),
                sr.left, sr.top, 0, 0, x, y, ssz.cx, ssz.cy);
 }
}
```

from ImageWin32.cpp:

```
void Image::Data::PaintImp(SystemDraw& w, int x, int y, const Rect& src, Color c)
{
 GuiLock __;
 SystemData& sd = Sys();
 ASSERT(!paintonly || IsNull(c));
 int max = IsWinNT() ? 250 : 100;
 while(ResCount > max) {
  Image::Data *l = ResData->GetPrev();
  l->SysRelease();
  l->Unlink();
 }
 HDC dc = w.GetHandle();
 Size sz = buffer.GetSize();
 int  len = sz.cx * sz.cy;
 Rect sr = src & sz;
 Size ssz = sr.Size();
 if(sr.IsEmpty())
  return;
 if(GetKind() == IMAGE_EMPTY)
  return;
 if(GetKind() == IMAGE_OPAQUE && !IsNull(c)) {
  w.DrawRect(x, y, sz.cx, sz.cy, c);
  return;
 }
 if(GetKind() == IMAGE_OPAQUE && paintcount == 0 && sr == Rect(sz) && IsWinNT() &&
w.IsGui()) {
  LTIMING("Image Opaque direct set");
  SetSurface(w, x, y, sz.cx, sz.cy, buffer);
  paintcount++;
  return;
 }
 Unlink();
 LinkAfter(ResData);
 if(GetKind() == IMAGE_OPAQUE) {
  if(!sd.hbmp) {
   LTIMING("Image Opaque create");
   CreateHBMP(dc, buffer);
  }
  LTIMING("Image Opaque blit");
  HDC dcMem = ::CreateCompatibleDC(dc);
  HBITMAP o = (HBITMAP)::SelectObject(dcMem, sd.hbmp);
```

```
  ::BitBlt(dc, x, y, ssz.cx, ssz.cy, dcMem, sr.left, sr.top, SRCCOPY);
  ::SelectObject(dcMem, o);
  ::DeleteDC(dcMem);
  PaintOnlyShrink();
  return;
 }
 if(GetKind() == IMAGE_MASK/* || GetKind() == IMAGE_OPAQUE*/) {
  HDC dcMem = ::CreateCompatibleDC(dc);
  if(!sd.hmask) {
   LTIMING("Image Mask create");
   Buffer<RGBA> bmp(len);
   sd.hmask = CreateBitMask(buffer, sz, sz, sz, bmp);
   ResCount++;
   if(!sd.hbmp)
    CreateHBMP(dc, bmp);
  }
  LTIMING("Image Mask blt");
  HBITMAP o = (HBITMAP)::SelectObject(dcMem, ::CreateCompatibleBitmap(dc, sz.cx, sz.cy));
  ::BitBlt(dcMem, 0, 0, ssz.cx, ssz.cy, dc, x, y, SRCCOPY);
  HDC dcMem2 = ::CreateCompatibleDC(dc);
  ::SelectObject(dcMem2, sd.hmask);
  ::BitBlt(dcMem, 0, 0, ssz.cx, ssz.cy, dcMem2, sr.left, sr.top, SRCAND);
  if(IsNull(c)) {
   ::SelectObject(dcMem2, sd.hbmp);
   ::BitBlt(dcMem, 0, 0, ssz.cx, ssz.cy, dcMem2, sr.left, sr.top, SRCPAINT);
  }
  else {
   HBRUSH ho = (HBRUSH) SelectObject(dcMem, CreateSolidBrush(c));
   ::BitBlt(dcMem, 0, 0, ssz.cx, ssz.cy, dcMem2, sr.left, sr.top, 0xba0b09);
   ::DeleteObject(::SelectObject(dcMem, ho));
  }
  ::BitBlt(dc, x, y, ssz.cx, ssz.cy, dcMem, 0, 0, SRCCOPY);
  ::DeleteObject(::SelectObject(dcMem, o));
  ::DeleteDC(dcMem2);
  ::DeleteDC(dcMem);
  PaintOnlyShrink();
  return;
 }
#ifndef PLATFORM_WINCE
 if(fnAlphaBlend() && IsNull(c) && !ImageFallBack) {
  if(!sd.himg) {
   LTIMING("Image Alpha create");
   BitmapInfo32__ bi(sz.cx, sz.cy);
   sd.himg = CreateDIBSection(ScreenHDC(), bi, DIB_RGB_COLORS, (void **)&sd.section,
NULL, 0);
   ResCount++;
   memcpy(sd.section, ~buffer, buffer.GetLength() * sizeof(RGBA));
  }
```

```
  LTIMING("Image Alpha blit");
  BLENDFUNCTION bf;
  bf.BlendOp = AC_SRC_OVER;
  bf.BlendFlags = 0;
  bf.SourceConstantAlpha = 255;
  bf.AlphaFormat = AC_SRC_ALPHA;
  HDC dcMem = ::CreateCompatibleDC(dc);
  ::SelectObject(dcMem, sd.himg);
  fnAlphaBlend()(dc, x, y, ssz.cx, ssz.cy, dcMem, sr.left, sr.top, ssz.cx, ssz.cy, bf);
  ::DeleteDC(dcMem);
  PaintOnlyShrink();
 }
 else
#endif
 {
  LTIMING("Image Alpha sw");
  DrawSurface sf(w, x, y, ssz.cx, ssz.cy);
  RGBA *t = sf;
  for(int i = sr.top; i < sr.bottom; i++) {
   if(IsNull(c))
    AlphaBlendOpaque(t, buffer[i] + sr.left, ssz.cx);
   else
    AlphaBlendOpaque(t, buffer[i] + sr.left, ssz.cx, c);
   t += ssz.cx;
  }
 }
}
```

Subject: Re: How does CtrlCore Image::Data::PaintImp work?
Posted by fudadmin on Tue, 25 Jan 2011 03:58:37 GMT
View Forum Message <> Reply to Message

ok, it blits images with masks. but the questions still remains...

Subject: Re: How does CtrlCore Image::Data::PaintImp work?
Posted by mirek on Tue, 25 Jan 2011 11:54:13 GMT
View Forum Message <> Reply to Message

fudadmin wrote on Mon, 24 January 2011 10:32sorry for my loose thoughts and sorry if some documentation does exist. If that's the case, please refer me to it. But I couldn't find anything useful.

I am trying to write ImageMac.cpp and I need some help.
As the topic name says I can't grasp and I need a general idea of "How does CtrlCore Image::Data::PaintImp work?"

1. Firstly, what are ResData?
my guesses (involves some optimisation - reusing?) :
 mouse pointer = cursor cheat, parts of the image being painted, or other resources like e.g brushes, masks, all of them combined (but how, in which cases?) ?

Draw is completely platform independent.

In some versions of Windows the number of image handles is limited to some thousands _per system_. It is thus advisable not to have all Images "realized" in GDI as bitmap handles.

So we keep track of number of handles in ResCount and if there is too much of them, we release some using Data...

Quote:
2. Why then ResCount>512 for X11?

Well, while there is no similar limitation for X11, as we already had mechanism available, it seemed to be a good idea to use it in X11 too, just to soften the load on system.

It is worth mentioning that U++ Image is somewhat unique and departing from "classic approach" where you usually have one class for 'client side' Image and another for 'server side'. Therefore we need to keep number of 'server side' images low, as we put them there automagically.

Quote:
And why for win32 - int max = IsWinNT() ? 250 : 100; ?
(see my comments below in the code.)

Because WinNT has unlimited number of handles, unlike Win98, but still, see the comment in 2.

Quote:
3. Of course, one of the main questions what would be the ResCount limit for Mac?

Hard to say, I would use 512 just like for X11. It is important to keep ResCount in sync though.

Having said all that, maybe all this approach is not the best one. Perhaps we should remove all "host platform hooks" from Image and simply provide a sort of cache only in SystemDraw, based on Image serial id.

I believe this is doable even now for MacOSX - all you need is to simply not use any of ResData, ResCount, SysRelease etc..

Quote:
for X11:
```
int  Image::Data::GetResCountImp() const
{
 return !!Sys().picture + !!Sys().picture8;
}
```

4. image converted to => picture + mask?

for win32:
```
int  Image::Data::GetResCountImp() const
{
 SystemData& sd = Sys();
 return !!sd.hbmp + !!sd.hmask + !!sd.himg;
}
```

5. image converted to => bitmap + mask + ??? ? (is this related somehow to DIB used somewhere? that's why the difference with X11?)

Well, these really are just counting how many handles are there for given Image.

Quote:
6. Also, what does
```
 Unlink();
 LinkAfter(ResData);
```
do?

Puts it to the front of the queue, so that least recently used Images are stripped of handles first (from the tail of the queue).

Quote:
7. How does ImageDraw::Section relate to Image::Data::PaintImp? Or what is ImageDraw::Section?

In no way, it is implementation detail for Win32.

Each platform should have its ImageDraw now. I guess that will have to change a bit with rainbow...

Subject: Re: How does CtrlCore Image::Data::PaintImp work?
Posted by fudadmin on Thu, 27 Jan 2011 14:41:01 GMT
View Forum Message <> Reply to Message

Thanks a lot, Mirek, for the answer which put me into deep thinking mode and exploring the possibilities.

Quote:Hard to say, I would use 512 just like for X11. It is important to keep ResCount in sync though.

Having said all that, maybe all this approach is not the best one. Perhaps we should remove all "host platform hooks" from Image and simply provide a sort of cache only in SystemDraw, based on Image serial id.

I believe this is doable even now for MacOSX - all you need is to simply not use any of ResData, ResCount, SysRelease etc..

This was similar to what I was thinking about for the simple reason that the current approach looked too complicated to me
Another problem for us from MacOSX is that CoreGraphics is like Cairo or Upp Painter but just implemented as a layer on top of OpenGL.

Maybe we should put Upp Painter on top of mac's OpenGL or..?
Btw, google's chrome for mac uses OpenGL.

I think, I need more info about the rainbow...
Ideas?

File Attachments
1) gs_graphics3_01.jpg, downloaded 771 times

Subject: Re: How does CtrlCore Image::Data::PaintImp work?
Posted by mirek on Fri, 04 Mar 2011 09:16:33 GMT
View Forum Message <> Reply to Message

fudadmin wrote on Thu, 27 January 2011 09:41
Maybe we should put Upp Painter on top of mac's OpenGL or..?
Btw, google's chrome for mac uses OpenGL.

Basic painter is just a set of virtual methods. You can attempt to create OpenGLPainter any time.

That said, I doubt it is possible. Painting model of Painter is too different from what OpenGL offers. There IMO is really quite little you can accelerate in Painter (which in fact is not that different from Quartz2D in operations) using OpenGL. Maybe some composition things and some image rendering, but that is about it...

Mirek