Subject: Re: MILESTONE: gtk3 replaces gtk2 as default linux backend
Posted by mirek on Tue, 21 Apr 2020 14:20:18 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Mon, 20 April 2020 20:37

I think this has now been taken pretty much as far as it goes with these backends, unless GTK3 still has some optimization available for opaque images.

Actually, it has cairo_surface_create_similar_image.

I have just finished experimenting with this feature. It is very confusing: It works really well, bringing DrawImage to microseconds time (probably using XRender path), however it has really nasty feature that it kills the drawing performance after DrawImage.

My guess is that it flips cairo into "GPU" mode and then when you want to draw e.g. rectangles, it starts copying memory between GPU and CPU for each element. Or something like that....

For future reference, before I revert them for now, here are changes I have tried:

```
struct ImageSysData {
 Image           img;
 cairo_surface_t *surface = NULL;

 void Init(const Image& m, cairo_surface_t *other);
 ~ImageSysData();
};

cairo_surface_t *CreateCairoSurface(const Image& img, cairo_surface_t *other)
{
 Size isz = img.GetSize();
 cairo_format_t fmt = CAIRO_FORMAT_ARGB32;
 cairo_surface_t *surface = other && 0 ? cairo_surface_create_similar_image(other, fmt, isz.cx,
isz.cy)
                                       : cairo_image_surface_create(fmt, isz.cx, isz.cy);
 cairo_surface_flush(surface);
 byte *a = (byte *)cairo_image_surface_get_data(surface);
 int stride = cairo_format_stride_for_width(fmt, isz.cx);
 for(int yy = 0; yy < isz.cy; yy++) {
  Copy((RGBA *)a, img[yy], isz.cx);
  a += stride;
 }
 cairo_surface_mark_dirty(surface);
 return surface;
}
```

```
cairo_surface_t *CreateCairoSurface(const Image& img)
{
 return CreateCairoSurface(img, NULL);
}

void ImageSysData::Init(const Image& m, cairo_surface_t *other)
{
 img = m;
 surface = CreateCairoSurface(m, other);
 SysImageRealized(img);
}

ImageSysData::~ImageSysData()
{
 SysImageReleased(img);
 cairo_surface_destroy(surface);
}

struct ImageSysDataMaker : LRUCache<ImageSysData, int64>::Maker {
 Image img;
 cairo_surface_t *other;

 virtual int64  Key() const                    { return img.GetSerialId(); }
 virtual int    Make(ImageSysData& object) const { object.Init(img, other); return img.GetLength(); }
};

void SystemDraw::SysDrawImageOp(int x, int y, const Image& img, Color color)
{
 GuiLock __;
 FlushText();
 if(img.GetLength() == 0)
  return;
 LLOG("SysDrawImageOp " << img.GetSerialId() << ' ' << x << ", " << y << ", "<< img.GetSize());
 ImageSysDataMaker m;
 static LRUCache<ImageSysData, int64> cache;
 static int Rsz;
 Rsz += img.GetLength();
 if(Rsz > 200 * 200) { // we do not want to do this for each small image painted...
  Rsz = 0;
  cache.Remove([](const ImageSysData& object) {
   return object.img.GetRefCount() == 1;
  });
 }
 LLOG("SysImage cache pixels " << cache.GetSize() << ", count " << cache.GetCount());
 m.img = img;
 m.other = cairo_get_target(cr);
 ImageSysData& sd = cache.Get(m);
```

```
 if(!IsNull(color)) {
  SetColor(color);
  cairo_mask_surface(cr, sd.surface, x, y);
 }
 else {
  RTIMESTOP("cairo_paint");
  if(img.GetKindNoScan() == IMAGE_OPAQUE)
   cairo_set_operator(cr, CAIRO_OPERATOR_SOURCE);
  cairo_set_source_surface(cr, sd.surface, x, y);
  cairo_paint(cr);
  cairo_set_operator(cr, CAIRO_OPERATOR_OVER);
 }
 static Size ssz;
 if(ssz.cx == 0)
  ssz = Ctrl::GetVirtualScreenArea().GetSize();
 cache.Shrink(4 * ssz.cx * ssz.cy, 1000); // Cache must be after Paint because of PaintOnly!
}
```