Subject: Re: MILESTONE: gtk3 replaces gtk2 as default linux backend
Posted by Tom1 on Thu, 16 Apr 2020 19:08:51 GMT
View Forum Message <> Reply to Message

Hi Mirek,

Thank you very much for solving this issue for me! :)

I did some further tuning of the render/paint structure using the new ViewDraw and ended up with a nice target update time of 60-150 micro seconds on both Windows and Linux GTK3 with the following code:

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

using namespace Upp;

class Testcase5 : public TopWindow {
public:
 typedef Testcase5 CLASSNAME;

 Rect loc;

 ImageBuffer map;

 Testcase5(){
  loc.SetNull();
 }

 void Paint(Draw &w){
  int64 t0=usecs();

  SetSurface(w,Rect(map.GetSize()),map,map.GetSize(),Point(0,0));
  loc.SetNull();

  int64 t1=usecs();
  Title(Format("Paint took %lld us",t1-t0));
 }

 void Layout(){
  Size sz=GetSize();
  map.Create(sz);
  BufferPainter bp(map);
  bp.RectPath(Rect(sz));
  bp.Fill(Pointf(0,0),Green(),Pointf(sz.cx-1,sz.cy-1),Yellow());
  bp.Move(0,0).Line(Pointf(sz.cx-1,sz.cy-1)).Stroke(1,Black());
  bp.Move(0,sz.cy-1).Line(Pointf(sz.cx-1,0)).Stroke(1,Black());
 }
```

```
 void MouseMove(Point p, dword keyflags){
  int64 t0=usecs();

  Rect area;
  area.SetNull();
  if(!loc.IsNullInstance()) area.Union(loc);
  loc.Set(p.x-16,p.y-16,p.x+16,p.y+16);
  area.Union(loc);
  area.Inflate(2);
  area.Intersect(Rect(GetSize()));

  ImageBuffer sb(area.GetSize());
  for(int y=0;y<area.Height();y++)
memcpy(sb[y],&map[y+area.top][area.left],sizeof(RGBA)*area.Width());

  BufferPainter sbp(sb);
  sbp.Translate(loc.left-area.left,loc.top-area.top);
  sbp.Move(0,0).Line(31,31);
  sbp.Move(0,31).Line(31,0);
  sbp.Stroke(5,Black());

  ViewDraw draw(this,area);
  SetSurface(draw,area.GetSize(),sb,sb.GetSize(),Point(0,0));

  int64 t1=usecs();
  Title(Format("Move took %lld us",t1-t0));

 }
};


GUI_APP_MAIN
{
 Testcase5().Sizeable().MaximizeBox().MinimizeBox().Run();
}
```

I do not need support for this feature in Mac, but what worries me is the potential deprecation of ViewDraw... Is there a real risk that you would drop ViewDraw entirely? As you can see from running the example above, an optimized Paint (as above, drawing a readily rendered ImageBuffer with SetSurface()) of the entire view on a 4K screen takes something like 6-12 ms, the small update with ViewDraw runs at around just 1 % of that time. (That's the difference between success and failure for my task.)

How would similar code and its performance look with what you refer to as 'Refresh and optimized Paint'?

Thanks and best regards,

Tom