Subject: Painter future
Posted by Tom1 on Tue, 24 Feb 2009 09:05:23 GMT
View Forum Message <> Reply to Message

Hi everyone,

I wish to open a discussion about the future of Painter and related technologies.

The exact point I wish to address is that the Painter introduced not only a versatile software rendering engine, but also a powerful API for doing so. When writing applications producing graphical content, there are almost always multiple target devices for the content produced: Screen display, images (in various formats) and printers.

Most targets would benefit from implementing the new Painter API more directly on their native APIs for various reasons.

1. Screen display would get more speed from using the capabilities of the modern GPUs.

2. Printing on large high resolution ink-jet plotters may not even be possible without sending the content in vector format. At above 2 GB per A0 sized sheet, the ImageBuffer is just too large to handle. Sending the vector content as vectors and raster content as scaled rasters to the printer, would make a lot of sense to me -- also with smaller paper sizes.

3. Producing vector based PDF and SVG -files directly using Painter interface would be nice.

Is there a plan how the new Painter API will be efficiently mapped to the various targets?

I'm not a Painter specialist, but I guess there should be separate "backends" for GDI, X11, PDF, SVG... in addition to the current ImageBuffer rendering to get the most efficient result.

Please throw in your thoughts.

Regards,

Tom

Subject: Re: Painter future
Posted by mirek on Tue, 24 Feb 2009 12:22:14 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Tue, 24 February 2009 04:05
2. Printing on large high resolution ink-jet plotters may not even be possible without sending the content in vector format. At above 2 GB per A0 sized sheet, the ImageBuffer is just too large to handle.

Actually, that is no problem. Banding of Painting is already implemented.

Quote:
 Sending the vector content as vectors and raster content as scaled rasters to the printer, would make a lot of sense to me -- also with smaller paper sizes.

If only usual printers would be capable of this....

Quote:
3. Producing vector based PDF and SVG -files directly using Painter interface would be nice.

Planned.

Quote:
Is there a plan how the new Painter API will be efficiently mapped to the various targets?

Well, all it needs is work... Painter itself is abstract class, you can implement it in any way...

Personally, I am quite sceptical about GPU acceleration. But rest makes sense.

In any case, PDF and SVG are almost requirements.

IMO, the work still needed on graphical front is:

- separation of abstract draw (and Painter) from host OS
- better font management (glyph replacements etc..)
- Painter -> PDF export
- Painter SVG import/export (import might require implementation of SVG filters...)

Mirek

Subject: Re: Painter future
Posted by Mindtraveller on Tue, 24 Feb 2009 13:58:18 GMT
View Forum Message <> Reply to Message

luzr wrote on Tue, 24 February 2009 15:22Personally, I am quite sceptical about GPU acceleration.Mirek Could you please explain why?

Subject: Re: Painter future
Posted by mirek on Tue, 24 Feb 2009 15:50:21 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Tue, 24 February 2009 08:58luzr wrote on Tue, 24 February 2009 15:22Personally, I am quite sceptical about GPU acceleration.Mirek Could you please explain why?

Well, for example imagine the difference between GPU accelerated "true subpixel precision" text rendering.

Each text glyph consists of about 50 line segments (after curve approximations), usually very small.

To rasterizer them in software, I have to do very little CPU steps. Rasterizer only goes through "real" pixels, subpixel precision is "collected" on the way; what I want to say here is that drawing line from [0, 1.23] to [1.21, 2.91] is basically done in 2 steps.

I can rasterize the whole glyph polygon quite quickly this way and then just draw resulting scanlines.

Compare to GPU solution. AFAIK, I would have to tesselate these 50 line segments into ~50 triangles, not an easy task, then push a lot of data through GPU API. I think that CPU would win here.

(And do not let me start to examine how diffucult would be to implement even-odd rule there

Mirek

Subject: Re: Painter future
Posted by Tom1 on Wed, 25 Feb 2009 13:09:09 GMT
View Forum Message <> Reply to Message

Putting aside true subpixel precision and anti-aliasing, comparing just raw simple polygon rendering under Upp, I made a simple test app:

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

using namespace Upp;

struct PainterExample : Ctrl {
public:
 double delta;
 virtual void Paint(Draw &draw) {
  delta=0;

  ImageBuffer ib(draw.GetPagePixels());
  BufferPainter pntr(ib);
  dword begin=GetTickCount();

  int reps=0;
  for(reps=0;reps<5;reps++) for(int i=0;i<500;i++){
   //pntr.Move(i,0).Line(499,i).Line(499-i,499).Line(0,499-i).Close().Stroke(1,Red());
   pntr.Move(i,0).Line(499,i).Line(499-i,499).Line(0,499-i).Close().Fill(Red());
  }
  draw.DrawImage(0,0,ib);
  dword end=GetTickCount();
  delta=end-begin;
  delta/=reps;
 }
};

struct DrawExample : Ctrl {
public:
 double delta;
 virtual void Paint(Draw &draw) {
  delta=0;

  dword begin=GetTickCount();
  Rect rect(0,0,500,500);

  int reps=0;
  for(reps=0;reps<5;reps++) for(int i=0;i<500;i++){
   Point v[5]={Point(i,0),Point(499,i),Point(499-i,499),Point(0,499-i),Point(i,0)};
   //draw.DrawPolyline(v,5,0,Green());
   draw.DrawPolygon(v,5,Green());
  }
  dword end=GetTickCount();
  delta=end-begin;
  delta/=reps;
 }

};

struct OpenGLExample : GLCtrl {
public:
 double delta;
 virtual void GLPaint() {
  delta=0;
  Size sz=GetSize();
  dword begin=GetTickCount();

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glViewport(0,0,sz.cx,sz.cy);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glPushMatrix();
  glOrtho(0,sz.cx,sz.cy,0,-1,1);


  int reps=0;
  for(reps=0;reps<10;reps++) for(int i=0;i<500;i++){
   //glBegin(GL_LINE_STRIP); // Polyline
   glBegin(GL_POLYGON); // Polygon
   glColor3f(0.0f,0.0f,1.0f); // Blue
   glVertex2f((float)i,(float)0);
   glVertex2f((float)499,(float)i);
   glVertex2f((float)499-i,(float)499);
   glVertex2f((float)0,(float)499-i);
   glVertex2f((float)i,(float)0);
   glEnd();
   glFlush();
  }

  glPopMatrix();
  glFlush();

  dword end=GetTickCount();
  delta=end-begin;
  delta/=reps;
 }
};

class ExampleTopWindow: public TopWindow{
public:
 PainterExample pr;
 DrawExample dw;
 OpenGLExample gl;

 ExampleTopWindow(){
  pr.SetRect(0,0,500,500);
  dw.SetRect(500,0,500,500);
  gl.SetRect(1000,0,500,500);

  pr.BackPaint();
  dw.BackPaint();
  gl.BackPaint();

  pr.delta=0;
  dw.delta=0;
  gl.delta=0;
  Add(pr);
  Add(dw);
  Add(gl);
 }

 virtual void LeftDown(Point p,dword keyflags){
  Title(Format("Painter/Draw/OpenGL: %.3f/%.3f/%.3f ms",pr.delta,dw.delta,gl.delta));
  Refresh();
 }

};

GUI_APP_MAIN
{
 ExampleTopWindow win;
 win.Sizeable().Zoomable();
 win.Open();
 win.Run();
}
```

(Changing the limit for repeats on different platforms may be required for sufficient clock resolution. Also, this requires a screen width above 1500 pixels -- changing some values for smaller screens will be necessary.)

By clicking on the area not covered by the controls, you get the results of the previous round and simultaneously start the next one.

The results on my system (AMDx64/2.0GHz single core + NVidia 7600GS, running Vista Business x64) look as follows:

Painter polygons: 450 ms
Draw polygons (using GDI): 300 ms
OpenGL polygons (using GLCtrl): 51 ms

Painter polylines: 210 ms
Draw polylines(using GDI): 5 ms
OpenGL polylines (using GLCtrl): 32 ms

What are your results?

// Tom

Subject: Re: Painter future

Posted by mirek on Wed, 25 Feb 2009 14:33:17 GMT
View Forum Message <> Reply to Message

Nice benchmark. Obviously, painting big rectangles is faster with GPU.

It is however quite interesting to change the line width to e.g. 4...

Mirek

Subject: Re: Painter future
Posted by Tom1 on Wed, 25 Feb 2009 15:09:04 GMT
View Forum Message <> Reply to Message

Changing the line width to 4 pixels on my system yields the following change in results:

Painter polylines: 210 ms -> 243 ms
Draw polylines(using GDI): 5 ms -> 43 ms
OpenGL polylines (using GLCtrl): 32 ms -> 31 ..34 ms

NOTE: Only line widths of 1 are required to be supported by any OpenGL implementation, so drawing polylines wider than 1 pixels will have to be mapped to polygons anyway for OpenGL to ensure compatibility across platforms.

In my opinion, the most notable change is with GDI performance level dropping from the "awesome" level to just "OK".

// Tom

Subject: Re: Painter future
Posted by mirek on Wed, 25 Feb 2009 17:36:28 GMT
View Forum Message <> Reply to Message

You might also want to try Lion test... (PainterExamples/Lion.cpp) Should be quite easy to be adapted.

Well, OpenGL clearly is NOT what I had in mind when I started work on Painter. Anyway, OpenGL backend is likely possible, I am just not going to do it myself.

Thinking about the issue, I guess the main and really big issue is that OpenGL only draws convex polygons. Means you will have to implement some smart algorithm to break polygons, using actual winding rule, into convex ones.

Also, another problem is aliasing. Obviously, you are not going to get the same kind of quality (or even subpixel rendering) using OpenGL.

Mirek

Subject: Re: Painter future
Posted by Mindtraveller on Wed, 25 Feb 2009 22:15:16 GMT
View Forum Message <> Reply to Message

Tom1, what about memory requirements of GUI and OpenGL versions?

Subject: Re: Painter future
Posted by Tom1 on Thu, 26 Feb 2009 10:17:14 GMT
View Forum Message <> Reply to Message

Pavel; Memory footprint of the Windows 32-bit exe file changes by just 5120 bytes, mostly because of inclusion of GLCtrl package.

Mirek; OpenGL anti-aliasing seems platform dependent. On my linux box, I can get nice smooth lines and polygon edges, while on Windows heavy aliasing occurs regardless of "glEnable(GL_LINE_SMOOTH and/or GL_POLYGON_SMOOTH);" or application independent graphics board settings. Aliasing behavior also seems different on Vista/NVidia vs. XP/ATI based systems.

The OpenGL requirement of making polygons convex first is really annoying. Good thing you pointed it out.

To sum it up: It was and still is quite obvious that implementing additional Painter backends for GDI, X11 and/or OpenGL is a clear tradeoff between quality and speed. Maybe some day...

Anyway, thanks for lining up the planned future of Painter -- extensions and more.

// Tom