Subject: Painter refactored/optimized
Posted by mirek on Sun, 11 Nov 2018 12:47:59 GMT
View Forum Message <> Reply to Message

After upgrading to 8C/16T machine, I was ashamed by pitiful gains (and terrible losses in some cases) of Painter performance in MT mode.

Which lead me to spending about 80 hours trying to optimize it. The result is about 10% improvement in ST performance and substantial improvements in MT - usually about 50%, but in some cases 200%. In some cases MT is now 6x faster than ST (on 8C/16T AMD 2700X).

Enjoy :)

Mirek

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Sun, 11 Nov 2018 17:06:55 GMT
View Forum Message <> Reply to Message

Hi Mirek!

Absolutely great news! I just can't wait to get to the office tomorrow morning to test this!

Thanks and best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by koldo on Sun, 11 Nov 2018 17:15:23 GMT
View Forum Message <> Reply to Message

Thank you Mirek.

Subject: Re: Painter refactored/optimized
Posted by Novo on Sun, 11 Nov 2018 18:55:14 GMT
View Forum Message <> Reply to Message

Thank you!
Could you please fix this:
uppsrc/Painter/BufferPainter.h:325:82: warning: control reaches end of non-void function [-Wreturn-type]

```
    BufferPainter&     NoImageCache(bool b = true)          { ImageCache(false); }
                                                                                  ^
```

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Mon, 12 Nov 2018 08:25:25 GMT
View Forum Message <> Reply to Message

Hi Mirek,

It is faster indeed, but now some segments of strokes drop out on the right edge of the view if they are partially clipped by the right edge of the view. I think the 'blanking distance' from the right edge is erroneously dependent on the scaling in the transformation being used. (I use the scaling and transformation to implement zooming, rotation and panning of vector maps.)

Another issue is that Filled text drops out in the top edge when the text touches or crosses the edge of the view. The outline (Stroked) text gets drawn until half of it is clipped from the top. (They should obviously both be drawn as long as there are any pixels visible.)

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Mon, 12 Nov 2018 08:35:30 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Mon, 12 November 2018 09:25Hi Mirek,

It is faster indeed, but now some segments of strokes drop out on the right edge of the view if they are partially clipped by the right edge of the view. I think the 'blanking distance' from the right edge is erroneously dependent on the scaling in the transformation being used. (I use the scaling and transformation to implement zooming, rotation and panning of vector maps.)

Another issue is that Filled text drops out in the top edge when the text touches or crosses the edge of the view. The outline (Stroked) text gets drawn until half of it is clipped from the top. (They should obviously both be drawn as long as there are any pixels visible.)

Best regards,

Tom

Well, thanks for testing, in fact I was hoping you will test this and half expected that there will be issues.

Do you think it would be possible to provide screenshots and/or testcase?

Are this issue apparent both in ST and MT?

Also: In Render.cpp, line 142, there is

if(pathattr.mtx_serial != preclip_mtx_serial) {

try to change that to

if(pathattr.mtx_serial != preclip_mtx_serial || 1) {

Thanks, I will try hard to resolve this as soon as possible... (if you provide testcase, it should be really soon :)

Mirek

Subject: Re: Painter refactored/optimized
Posted by mirek on Mon, 12 Nov 2018 08:39:55 GMT
View Forum Message <> Reply to Message

One last thing, does it show these problems without preclip?

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Mon, 12 Nov 2018 08:55:10 GMT
View Forum Message <> Reply to Message

Mirek,

MT/ST does not have any effect on this.

You are absolutely right: Not using painter.PreClipDashed(); fixes both issues. (I do not use the old PreClip anymore after PreClipDashed was introduced.)

Using:
if(pathattr.mtx_serial != preclip_mtx_serial || 1) {
Fixes both issues too. :)

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Mon, 12 Nov 2018 08:59:06 GMT
View Forum Message <> Reply to Message

As for a testcase, my (commercial) code is complex and figuring out a testcase will take a while. I'll see what I can do. Anyway, please let me know if the above already helped you on the track...

Thanks and best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Mon, 12 Nov 2018 09:08:51 GMT
View Forum Message <> Reply to Message

Hi,

After all, the test case was easy: Just use PainterExamples and add sw.PreClipDashed(); right after constructing the BufferPainter in App::Paint() in main.cpp.

Then compile and run PainterExamples, select Stroke example and drag the right edge of the window to gradually cover the the contents of the example. Both the line and the stroked text will disappear before they are fully covered by the window edge.

I'll stay tuned for the fix to test. :)

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Mon, 12 Nov 2018 10:15:29 GMT
View Forum Message <> Reply to Message

Found and fixed 3 issues, can you test please?

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Mon, 12 Nov 2018 10:53:04 GMT
View Forum Message <> Reply to Message

Mirek,

As far as I can see, everything renders perfectly now.

Thank you very much for your excellent work on Painter again!!

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Mon, 12 Nov 2018 10:59:23 GMT
View Forum Message <> Reply to Message

BTW, preclipping is optimized as well. As long as you do not change transformation matrix too often, it should be significantly faster.

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Mon, 12 Nov 2018 11:39:12 GMT
View Forum Message <> Reply to Message

Hi,

I cannot confirm any change in PreClip() or PreClipDashed() performance with my usage profile. But it is true that my program tries really hard to not pass Painter anything that would not be visible at least partially. So, this may be the reason I'm not seeing the improvement here.

Importantly, both PreClip() and PreClipDashed() still improve rendering speed of a _very_ long partially visible horizontal dashed line from e.g. 10 seconds to about 0.9 seconds. I guess the only way to dramatically improve this is to clip the line before dashing it.

Is there any difference between the two preclip functions anymore?

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Mon, 12 Nov 2018 12:25:59 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Mon, 12 November 2018 12:39Hi,

I cannot confirm any change in PreClip() or PreClipDashed() performance with my usage profile. But it is true that my program tries really hard to not pass Painter anything that would not be visible at least partially. So, this may be the reason I'm not seeing the improvement here.

Importantly, both PreClip() and PreClipDashed() still improve rendering speed of a _very_ long partially visible horizontal dashed line from e.g. 10 seconds to about 0.9 seconds. I guess the only way to dramatically improve this is to clip the line before dashing it.

Is there any difference between the two preclip functions anymore?

Yes, they basically behave the same. What is new is that inverse matrix is now calculated only if transformation matrix changes.

Mirek

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Mon, 12 Nov 2018 15:15:57 GMT
View Forum Message <> Reply to Message

OK, that sounds logical. :)

Thanks and best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Mon, 12 Nov 2018 22:42:27 GMT
View Forum Message <> Reply to Message

After integrating new Painter with the target application, I have initially noticed that performance is not so great.

Investigation revealed that the problem was the app was creating BufferPainter several times per 'frame'. So in order to achive good perfromance, it is advisable to limit the number of BufferPainter destructors called. I have ended with single BufferPainter as member variable that exists for the whole lifetime of the application.

There is now new method "BufferPainter::Create" that allow it to "rebind" to another ImageBuffer, keeping as much initialized internal data as possible.

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Tue, 13 Nov 2018 08:14:11 GMT
View Forum Message <> Reply to Message

Hi Mirek,

Here's the test result from this morning.

If I replace the following in my Paint() method:

BufferPainter painter(ib);

With the following:

painter.Create(ib);

And add a class variable 'BufferPainter painter;', The rendering will seemingly randomly drop various elements, especially texts, from the result when using MT. It does not matter if I have PreClip()/PreClipDashed() enabled or not. When using ST, everything works fine even with this new Create() mechanism. Also, if I use the traditional 'BufferPainter painter(ib);' on each Paint(), everything works fine with both MT and ST.

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Tue, 13 Nov 2018 08:18:37 GMT
View Forum Message <> Reply to Message

You have to call "Finish" at the end of rendering (that is also called by destructor). It waits for thread "pipeline" to finish the work.

Anyway, if you create single BufferPainter per render and you have thousands of polygons to render, you are probably fine. My problem was that my original code was creating like 30 BufferPainters...

Mirek

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Tue, 13 Nov 2018 08:35:59 GMT
View Forum Message <> Reply to Message

Hi,

OK, I added 'sw.Finish();' in the end. There is still something strange with it.

Please add 'BufferPainter csw;' to PainterExamples App and Change the App::Paint() in main.cpp:

```
void App::Paint(Draw& w)
{
 Size sz = GetSize();
 if(ctrl.transparent) {
  for(int y = 0; y < sz.cy; y += 32)
   for(int x = 0; x < sz.cx; x += 32)
    w.DrawRect(x, y, 32, 32, (x ^ y) & 32 ? Color(254, 172, 120) : Color(124, 135, 253));
 }
 ImageBuffer ib(sz);
 {
  //BufferPainter sw(ib, ctrl.quality); // Removed
```

```
  csw.Create(ib, ctrl.quality); // Added

  BufferPainter &sw=csw;

  if(ctrl.transparent)
   sw.Clear(RGBAZero());
  else
   sw.Clear(White());
  sw.Co(ctrl.mt);
  DoPaint(sw);

  sw.Finish(); // Added
 }
 w.DrawImage(0, 0, ib);

}
```

Just scaling causes strange behavior.

Best regards,

Tom

EDIT: My own code worked fine again after adding the 'sw.Finish();'. However, it feels like the transformations do not get reset to default identity transformation in PainterExamples. Maybe this should be part of Create()...?
EDIT2: This appears to solve the issue with PainterExamples. Add the following in the end of BufferPainter::Create():
pathattr.mtx = attr.mtx = Xform2D::Identity();
However, I'm not sure if this breaks something else instead...

Subject: Re: Painter refactored/optimized
Posted by mirek on Tue, 13 Nov 2018 10:37:06 GMT
View Forum Message <> Reply to Message

Probably my bug.

Can you try to place Begin/End around DoPaint?

Mirek

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Tue, 13 Nov 2018 10:52:55 GMT
View Forum Message <> Reply to Message

Yes. Begin()/End() around DoPaint(); equally fixes the issue with PainterExamples. But I think it would be nice to have a clean table after BufferPainter::Create()... Of course, I understand that each re-initialized variable increases the cost towards full constructor/destructor pair.

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Tue, 13 Nov 2018 10:55:27 GMT
View Forum Message <> Reply to Message

Sure, I agree, I justed wanted hints about the problem...

Subject: Re: Painter refactored/optimized
Posted by mirek on Tue, 13 Nov 2018 11:22:50 GMT
View Forum Message <> Reply to Message

Please test (trunk).

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Tue, 13 Nov 2018 11:50:30 GMT
View Forum Message <> Reply to Message

The issue is still there in SVN 12531 if you do my above changes to PainterExamples. (Obviously, adding Begin/End will still remove the issue.)

I can't figure out what exactly you changed in Create though. Or am I working on a completely wrong SVN version?

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Tue, 13 Nov 2018 12:54:50 GMT
View Forum Message <> Reply to Message

rev 12531:

void BufferPainter::Create(ImageBuffer& ib, int mode_)

```
{
 ip = &ib;

 if(mode_ != mode || (Size)size != ib.GetSize()) {
  mode = mode_;

  rasterizer.Create(ib.GetWidth(), ib.GetHeight(), mode == MODE_SUBPIXEL);
  paths.Alloc(BATCH_SIZE);
  path_info = paths;

  ClearPath();

  render_cx = ib.GetWidth();
  if(mode == MODE_SUBPIXEL) {
   render_cx *= 3;
   subpixel.Alloc(render_cx + 30);
  }
  size = ib.GetSize();
 }

 Attr& a = attr;
 a.cap = LINECAP_BUTT;
 a.join = LINEJOIN_MITER;
 a.miter_limit = 4;
 a.evenodd = false;
 a.hasclip = false;
 a.cliplevel = 0;
 a.opacity = 1;
 a.dash = NULL;
 a.mask = false;
 a.invert = false;
 a.mtx_serial = 0;

 gradientn = Null;

 jobcount = fillcount = 0;
 cojob.Clear();
 cofill.Clear();

 attrstack.Clear();
 clip.Clear();
 mask.Clear();
 onpathstack.Clear();
 pathlenstack.Clear();
 onpath.Clear();

 preclip_mtx_serial = -1;
 path_index = 0;
```

}

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Tue, 13 Nov 2018 13:06:38 GMT
View Forum Message <> Reply to Message

OK, the file was correctly updated. (Reverting my changes simultaneously slightly misguided me to believe otherwise.)

Anyway the problem is still there: The transformation matrix does not get reset to identity in Create.

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Tue, 13 Nov 2018 15:23:03 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Tue, 13 November 2018 14:06OK, the file was correctly updated. (Reverting my changes simultaneously slightly misguided me to believe otherwise.)

Anyway the problem is still there: The transformation matrix does not get reset to identity in Create.

Best regards,

Tom

What now?

```
void BufferPainter::Create(ImageBuffer& ib, int mode_)
{
 ip = &ib;

 if(mode_ != mode || (Size)size != ib.GetSize()) {
  mode = mode_;

  rasterizer.Create(ib.GetWidth(), ib.GetHeight(), mode == MODE_SUBPIXEL);

  render_cx = ib.GetWidth();
```

```
  if(mode == MODE_SUBPIXEL) {
   render_cx *= 3;
   subpixel.Alloc(render_cx + 30);
  }
  size = ib.GetSize();
 }

 if(!paths)
  paths.Alloc(BATCH_SIZE);

 path_info = paths;

 ClearPath();

 Attr& a = attr;
 a.cap = LINECAP_BUTT;
 a.join = LINEJOIN_MITER;
 a.miter_limit = 4;
 a.evenodd = false;
 a.hasclip = false;
 a.cliplevel = 0;
 a.opacity = 1;
 a.dash = NULL;
 a.mask = false;
 a.invert = false;
 a.mtx_serial = 0;

 gradientn = Null;

 jobcount = fillcount = 0;
 cojob.Clear();
 cofill.Clear();

 attrstack.Clear();
 clip.Clear();
 mask.Clear();
 onpathstack.Clear();
 pathlenstack.Clear();
 onpath.Clear();

 preclip_mtx_serial = -1;
 path_index = 0;
}
```

Subject: Re: Painter refactored/optimized

Posted by Tom1 on Tue, 13 Nov 2018 16:23:59 GMT
View Forum Message <> Reply to Message

Hi,

EDIT: Sorry, I did not pick up the changes yet. Please stand by. I will test these latest changes in the morning at the office.

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Wed, 14 Nov 2018 09:57:25 GMT
View Forum Message <> Reply to Message

Hi Mirek,

As of r12533 Create now works as expected :)

However, there seems to be a severe performance issue with MT. In some cases MT can be three times slower than MT before this optimization round. E.g. a vector map rendering in 20 ms with previous MT and in 40 ms with ST now takes 60 ms with new MT.

This is somehow related to changing transformations (of course within Begin/End pairs) which is now extremely expensive, especially when using MT.

Thanks and best regards,

Tom

EDIT: I created a transformation intensive view that shows a matrix of just 90 symbols. Each of the symbols are drawn with strokes and fills using a different translation for each within a Begin/End pair. Rendering of this same view takes only 2.2 ms with ST but a whopping 45 ms with MT! Using PreClip or not does not have any observable effect on the result.

Subject: Re: Painter refactored/optimized
Posted by mirek on Wed, 14 Nov 2018 12:17:19 GMT
View Forum Message <> Reply to Message

Well, thats disappointing.

I really would like to have some example so I can optimize for this case.

Mirek

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Wed, 14 Nov 2018 12:39:24 GMT
View Forum Message <> Reply to Message

Hi,

You can test with PainterExamples by enabling MT and running Benchmark with OnPath and OnTextPath examples.

Best regards,

Tom

EDIT: 'Pythagoras Tree Image' example portrays this slowdown too. Every other PainterExamples example running MT is on par or faster compared to ST. With my 4C8T Intel Core i7 the best MT gain is about 4x compared to ST. This is common with images and fills. Narrow geometries do not gain so much boost from MT landing at 1x-2x speed improvement.

Subject: Re: Painter refactored/optimized
Posted by mirek on Wed, 14 Nov 2018 13:38:03 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Wed, 14 November 2018 13:39Hi,

You can test with PainterExamples by enabling MT and running Benchmark with OnPath and OnTextPath examples.

Best regards,

Tom

EDIT: 'Pythagoras Tree Image' example portrays this slowdown too. Every other PainterExamples example running MT is on par or faster compared to ST. With my 4C8T Intel Core i7 the best MT gain is about 4x compared to ST. This is common with images and fills. Narrow geometries do not gain so much boost from MT landing at 1x-2x speed improvement.

I have found that BeginOnPath was conservatively flushing rendering pipeline for no good reason, so that is now optimized out. TextOnPath is still slower if you fill the letters, that will have to wait till next batch of optimization I am afraid.

In fact, what is slow is alternating solid color / non-solid color fills - that is the case for both Pythagoras Tree Image and TextOnPath... Will have to think if there is anything I can do there...

Subject: Re: Painter refactored/optimized
Posted by mirek on Wed, 14 Nov 2018 13:39:49 GMT

(P.S. that OnPath optimization is commited, I would be glad if you test it)

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Wed, 14 Nov 2018 13:56:00 GMT

Hi,

Absolutely! I'm extremely motivated to help you squeeze every single extra millisecond out of the rendering times in Painter!

OnPath show now 1.4x improvement over ST and OnTextPath about 2.3x improvement... And these were over ST not over previous MT where improvement is 5x that. Well done Mirek!

A question --- removed ---

Best regards,

Tom

EDIT: Removing question. Something else is now slowing down my code. It may be related to translations (Xform2D).

Subject: Re: Painter refactored/optimized
Posted by mirek on Wed, 14 Nov 2018 14:19:26 GMT

Tom1 wrote on Wed, 14 November 2018 14:56Hi,

Absolutely! I'm extremely motivated to help you squeeze every single extra millisecond out of the rendering times in Painter!

OnPath show now 1.4x improvement over ST and OnTextPath about 2.3x improvement... And these were over ST not over previous MT where improvement is 5x that. Well done Mirek!

A question --- removed ---

Best regards,

Tom

EDIT: Removing question. Something else is now slowing down my code. It may be related to translations (Xform2D).

Well, what is definitely slow now if you are mixing solid fills and gradient or image fills. Can that be the cause?

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Wed, 14 Nov 2018 14:55:05 GMT
View Forum Message <> Reply to Message

Hi,

My translated symbols are simply solid black over white background. The ST vs. MT speed ratio is about 15..20x in favor of ST. They were about equally fast when compiled against r.11960.

I need to find a pattern here. There is no clear Begin/End dependency as far as I can see. I'm trying to narrow down the code to find a test case.

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Wed, 14 Nov 2018 16:57:13 GMT
View Forum Message <> Reply to Message

Screenshot / piece of code would be helpful (can be PM).

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Thu, 15 Nov 2018 09:14:23 GMT
View Forum Message <> Reply to Message

Hi,

I finally figured out a way to share this test case. I send you code and a Serialized Painting to test. (Actually, this can be quite handy for any Painter performance issue testing in general.) Here's the code:

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

using namespace Upp;

class PainterBench : public TopWindow {
```

```
public:
 Painting p;
 FileSel fs;

 void Open(){
  if(fs.ExecuteOpen("Select a painting to view")){
   p.Clear();
   p.Serialize(FileIn(fs.Get()));
  }
 }
 virtual bool Key(dword key, int count){
  switch(key){
   case K_CTRL_O:
    Open();
    return true;
  }
  return false;
 }

 typedef PainterBench CLASSNAME;

 PainterBench(){
  Sizeable();
 }

 virtual void Paint(Draw &draw){
  int64 STtiming=0;
  int64 MTtiming=0;

  ImageBuffer ib(GetSize());
  {
   BufferPainter bpainter(ib);
   bpainter.Co(true);
   bpainter.PreClipDashed();
   bpainter.Clear(White());
   bpainter.EvenOdd();

   int64 t0=usecs();
   bpainter.Paint(p);
   int64 t1=usecs();
   MTtiming=t1-t0;

  }
  {
   BufferPainter bpainter(ib);
   bpainter.Co(false);
   bpainter.PreClipDashed();
   bpainter.Clear(White());
```

```
   bpainter.EvenOdd();

   int64 t0=usecs();
   bpainter.Paint(p);
   int64 t1=usecs();
   STtiming=t1-t0;

  }

  SetSurface(draw,Rect(ib.GetSize()),ib,ib.GetSize(),Point(0,0));

  double gain=(double)STtiming/(double)(0.1+MTtiming); // Avoid div by zero
  Title(Format("Rendering MT took %lld us, ST took %lld us, MT gain is
%.2f",MTtiming,STtiming,gain));
 }
};

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

There are two Serialized painting files to test with: SomeRocks.painting exhibits the MT slowdown issue dramatically. The other file is just for checking how fast a typical map view renders.

Best regards,

Tom

File Attachments

1) SamplePaintingsSerialized.7z, downloaded 243 times

Subject: Re: Painter refactored/optimized
Posted by mirek on Thu, 15 Nov 2018 09:48:46 GMT
View Forum Message <> Reply to Message

OK, so the short answer: "It is too simple"

Less short answer: Most of time is spend allocating and cleaning per-thread dynamic data. MT is using up to 128 rasterizers, each rasterizer has to be allocated (~ 256KB) and after being used, it has to be "reset". I guess there is a lot of cache misses in the process, meanwhile ST runs in L1/L2 easily.

That said, I have some tricks in my mind to be implemented to fix this. The only trouble is that it involves changes to memory allocater, which is difficult...

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Thu, 15 Nov 2018 10:43:24 GMT
View Forum Message <> Reply to Message

Hi,

The difference is so large that it makes me wonder if ST allocates/resets any rasterizers at all on the fly?

Could the number of rendering threads be pre-selected and a sufficient number of rasterizers be pre-allocated for MT so that there would be no extra allocation/reset -penalty for re-using the same BufferPainter -- as was just introduced by BufferPainter::Create?

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Thu, 15 Nov 2018 10:55:08 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Thu, 15 November 2018 11:43Hi,

The difference is so large that it makes me wonder if ST allocates/resets any rasterizers at all on the fly?

The difference is that ST has just one rasterizer :)

Quote:
Could the number of rendering threads be pre-selected and a sufficient number of rasterizers be pre-allocated for MT so that there would be no extra allocation/reset -penalty for re-using the same BufferPainter -- as was just introduced by BufferPainter::Create?

Perhaps, but let me try those optimizations I have in mind first...

(Note that while the ST/MT ratio is horrible, it is still <ms for both mt and st... I guess that if you would add that Clear into time, difference would be much less).

Mirek

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Thu, 15 Nov 2018 11:14:08 GMT

Hi,

You say <ms??? ... you mean below one millisecond for MT??? I get something like 16 ms for MT and 300 us for ST... :? What exactly are your readings?

I bet your hardware is Superb! Mine is Core i7 4790K @ 4 GHz (4C/8T). Windows 10 Professional 64 bit. Compiled with MSBT17x64.

Do you have anything this old to test with?

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Thu, 15 Nov 2018 11:33:11 GMT

Tom1 wrote on Thu, 15 November 2018 12:14Hi,

You say <ms??? ... you mean below one millisecond for MT??? I get something like 16 ms for MT and 300 us for ST... :? What exactly are your readings?

I bet your hardware is Superb! Mine is Core i7 4790K @ 4 GHz (4C/8T). Windows 10 Professional 64 bit. Compiled with MSBT17x64.

Do you have anything this old to test with?

Best regards,

Tom

Nope, that is just difference in testing, sorry, I have adopted it to my development package (which is benchmarks/LionBenchmark). There I am testing by repeatedly doing the paint, with the same BufferPainter, until I spend 1 second, then compute the time based on number of renders achieved.

It is sort of similar to having single global BufferPainter.

My numbers with your example are about the same for ST and half for MT - at least, those 8 cores show up :)

Now if I insert some bechmarking code, it is obvious that those 8 ms in MT are spend by allocating / initializing memory...

Mirek

Subject: Re: Painter refactored/optimized
Posted by mirek on Thu, 15 Nov 2018 11:40:16 GMT
View Forum Message <> Reply to Message

OK, I have just found that I have accidentally deleted that precious initialized memory in Create. So the new version is in the trunk. Changing your example with global BufferPainter now shows some pretty significant gains:

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

using namespace Upp;

class PainterBench : public TopWindow {
public:
 Painting p;
 FileSel fs;
 BufferPainter bpainter;

 void Open(){
  if(fs.ExecuteOpen("Select a painting to view")){
   p.Clear();
   p.Serialize(FileIn(fs.Get()));
  }
 }


 virtual bool Key(dword key, int count){
  Refresh();
  switch(key){
   case K_CTRL_O:
    Open();
    return true;
  }
  return false;
 }

 typedef PainterBench CLASSNAME;

 PainterBench(){
  Sizeable();

  p.Serialize(FileIn("C:/xxx/PainteTest/SomeRocks.painting"));
 }
```

```
 virtual void Paint(Draw &draw){
  int64 STtiming=0;
  int64 MTtiming=0;

  ImageBuffer ib(GetSize());
  {
   bpainter.Create(ib);
   bpainter.Co(true);
   bpainter.PreClipDashed();
   bpainter.Clear(White());
   bpainter.EvenOdd();

   int64 t0=usecs();
   bpainter.Paint(p);
   int64 t1=usecs();
   MTtiming=t1-t0;

   bpainter.Finish();
  }
  {
   bpainter.Create(ib);
   bpainter.Co(false);
   bpainter.PreClipDashed();
   bpainter.Clear(White());
   bpainter.EvenOdd();

   int64 t0=usecs();
   bpainter.Paint(p);
   int64 t1=usecs();
   STtiming=t1-t0;

   bpainter.Finish();
  }

  SetSurface(draw,Rect(ib.GetSize()),ib,ib.GetSize(),Point(0,0));

  double gain=(double)STtiming/(double)(0.1+MTtiming); // Avoid div by zero
  Title(Format("Rendering MT took %lld us, ST took %lld us, MT gain is
%.2f",MTtiming,STtiming,gain));
 }
};

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

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Thu, 15 Nov 2018 12:07:49 GMT
View Forum Message <> Reply to Message

Hi!

Yes it indeed does. But even better: Now my real application rendering vector maps shows for the first time in MT Painter history consistent and significant MT/ST rendering speed gains of about 2.5x on the average with real data! :)

Thank you Mirek very much! You really Rock!

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Thu, 15 Nov 2018 12:23:41 GMT
View Forum Message <> Reply to Message

Hi,

One minor issue: When in Paint with global BufferPainter and only calling bufferpainter.Create(ib); the rendered area does not change to current ib size. (E.g. After maximizing the window the bufferpainter will only render on the initial initial ib area leaving the rest white.) I need to additionally call bufferpainter.Co(true or false); to get the bufferpainter work on the current ib size.

This is not a problem for me, but maybe it would be more appropriate to handle the resizing in Create somehow.

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Thu, 15 Nov 2018 12:33:42 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Thu, 15 November 2018 13:23Hi,

One minor issue: When in Paint with global BufferPainter and only calling bufferpainter.Create(ib); the rendered area does not change to current ib size. (E.g. After maximizing the window the bufferpainter will only render on the initial initial ib area leaving the rest white.) I need to additionally call bufferpainter.Co(true or false); to get the bufferpainter work on the current ib size.

This is not a problem for me, but maybe it would be more appropriate to handle the resizing in

Create somehow.

Best regards,

Tom

Ops, thats a bug. Fix in trunk, hopefully...

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Thu, 15 Nov 2018 12:44:40 GMT
View Forum Message <> Reply to Message

And Yes! It works!

Thanks and best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Fri, 16 Nov 2018 09:23:09 GMT
View Forum Message <> Reply to Message

Hi Mirek,

While on the subject, I decided to do some testing of thread count for MT Painter. What I found was interesting: My typical map renders at roughly 250 ms with ST and 100 ms with default 10 thread MT. (On my hardware CPU_Cores() returns 8 and CoWork initializes a thread pool of 10 threads.)

So I tampered a little bit with CoWork.cpp, trying with different thread counts:

```
int CoWork::GetPoolSize()
{
 int n = GetPool().threads.GetCount();
// return n ? n : CPU_Cores() + 2;
 return n ? n : 4;
}

CoWork::Pool::Pool()
{
 ASSERT(!IsWorker());

// InitThreads(CPU_Cores() + 2);
 InitThreads(4);
```

```
 free = NULL;
 for(int i = 0; i < SCHEDULED_MAX; i++)
  Free(slot[i]);

 quit = false;
}
```

In this test I ended up with four threads which yield about same performance as 10 threads. When dropping to three threads or below, the MT gain started to fade away.

I think the optimal thread count for CoWork depends on the job's balance of CPU load and memory bandwidth. Also, the CPU and memory bus design changes this balance. As the new CPUs tend to offer a lot of cores (and concurrent threads), a simple or well optimized algorithm will easily saturate the memory channels with a reasonably small subset of cores being used. I'm not sure though, if there is much point in reducing threads (and therefore freeing cores for other tasks), if the memory bus will remain saturated anyway.

Best regards,

Tom

Subject: Re: Painter refactored/optimized
Posted by mirek on Fri, 16 Nov 2018 10:20:23 GMT
View Forum Message <> Reply to Message

IMO, that is to be expected, as it is really 4C CPU...

10 threads default number for CoWork takes into account that perhaps some threads will do blocking operations (e.g. files). And in some workloads, hyperthreading has benefits.

That said, it is true that it would be nice to detect that threads are "wasted", but I am not sure how to do that...

Mirek

Subject: Re: Painter refactored/optimized
Posted by Tom1 on Fri, 16 Nov 2018 11:57:49 GMT
View Forum Message <> Reply to Message

Hi,

IMO your default "CPU logical cores + 2" is a well considered compromise to keep the CPU working full time without wasting much resources. No worries.

Best regards,

Tom