Subject: Re: BufferPainter::Clear() optimization
Posted by mirek on Sun, 17 May 2020 16:05:52 GMT
View Forum Message <> Reply to Message

What about this:

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

using namespace Upp;

void Fill0(RGBA *t, RGBA c, int len)
{
 while(len >= 16) {
  t[0] = c; t[1] = c; t[2] = c; t[3] = c;
  t[4] = c; t[5] = c; t[6] = c; t[7] = c;
  t[8] = c; t[9] = c; t[10] = c; t[11] = c;
  t[12] = c; t[13] = c; t[14] = c; t[15] = c;
  t += 16;
  len -= 16;
 }
 switch(len & 15) {
 case 15: t[14] = c;
 case 14: t[13] = c;
 case 13: t[12] = c;
 case 12: t[11] = c;
 case 11: t[10] = c;
 case 10: t[9] = c;
 case 9: t[8] = c;
 case 8: t[7] = c;
 case 7: t[6] = c;
 case 6: t[5] = c;
 case 5: t[4] = c;
 case 4: t[3] = c;
 case 3: t[2] = c;
 case 2: t[1] = c;
 case 1: t[0] = c;
 }
}

void Fill2(RGBA *t, RGBA c, int len)
{
 while(len >= 16) {
  t[0] = c; t[1] = c; t[2] = c; t[3] = c;
  t[4] = c; t[5] = c; t[6] = c; t[7] = c;
  t[8] = c; t[9] = c; t[10] = c; t[11] = c;
  t[12] = c; t[13] = c; t[14] = c; t[15] = c;
```

```
  t += 16;
  len -= 16;
 }
 if(len & 8) {
  t[0] = t[1] = t[2] = t[3] = t[4] = t[5] = t[6] = t[7] = c;
  t += 8;
 }
 if(len & 4) {
  t[0] = t[1] = t[2] = t[3] = c;
  t += 4;
 }
 if(len & 2) {
  t[0] = t[1] = c;
  t += 2;
 }
 if(len & 1)
  t[0] = c;
}

void Fill3(RGBA *t, RGBA c, int len)
{
 dword m[4];
 m[0] = m[1] = m[2] = m[3] = *(dword*)&(c);
 __m128d val = _mm_loadu_pd((double *)m);
 if(len >= 16) {
  if(len > 1024*1024 / 16 && ((uintptr_t)t & 3) == 0) { // for really huge data, bypass the cache
   while((uintptr_t)t & 15) { // align to 16 bytes for SSE
    *t++ = c;
    len--;
   }
   do {
    _mm_stream_pd((double *)t, val);
    _mm_stream_pd((double *)(t + 4), val);
    _mm_stream_pd((double *)(t + 8), val);
    _mm_stream_pd((double *)(t + 12), val);
    t += 16;
    len -= 16;
   }
   while(len >= 16);
   _mm_sfence();
  }
  else
   do {
    _mm_storeu_pd((double *)t, val);
    _mm_storeu_pd((double *)(t + 4), val);
    _mm_storeu_pd((double *)(t + 8), val);
    _mm_storeu_pd((double *)(t + 12), val);
    t += 16;
```

```
    len -= 16;
   }
   while(len >= 16);
 }
 if(len & 8) {
  _mm_storeu_pd((double *)t, val);
  _mm_storeu_pd((double *)(t + 4), val);
  t += 8;
 }
 if(len & 4) {
  _mm_storeu_pd((double *)t, val);
  t += 4;
 }
 if(len & 2) {
  t[0] = t[1] = c;
  t += 2;
 }
 if(len & 1)
  t[0] = c;
}

int len = 2000 * 4000;

GUI_APP_MAIN
{
 Color c = Red();

 Buffer<RGBA> b(2000);

 Vector<int> rnd;
 for(int i = 0; i < 200; i++)
  rnd << Random(100);

 for(int i = 0; i < 1000000; i++) {
  {
   RTIMING("memsetd");
   for(int i = 0; i < rnd.GetCount(); i += 2)
    memsetd(b + rnd[i], *(dword*)&(c), rnd[i + 1]);
  }
  {
   RTIMING("Fill");
   for(int i = 0; i < rnd.GetCount(); i += 2)
    Fill(b + rnd[i], c, rnd[i + 1]);
  }
  {
   RTIMING("Fill0");
   for(int i = 0; i < rnd.GetCount(); i += 2)
    Fill0(b + rnd[i], c, rnd[i + 1]);
```

```
  }
  {
   RTIMING("Fill2");
   for(int i = 0; i < rnd.GetCount(); i += 2)
    Fill2(b + rnd[i], c, rnd[i + 1]);
  }
  {
   RTIMING("Fill3");
   for(int i = 0; i < rnd.GetCount(); i += 2)
    Fill3(b + rnd[i], c, rnd[i + 1]);
  }
  {
   RTIMING("memset");
   for(int i = 0; i < rnd.GetCount(); i += 2)
    memset(b + 4 * rnd[i], 255, 4 * rnd[i + 1]);
  }
 }

 b.Alloc(len);

 for(int i = 0; i < 20; i++) {
  memsetd(b, *(dword*)&(c), len);
  {
   RTIMING("HUGE memsetd");
   memsetd(b, *(dword*)&(c), len);
  }
  {
   RTIMING("HUGE Fill");
   Fill(b, c, len);
  }
  {
   RTIMING("HUGE Fill3");
   Fill3(b, c, len);
  }
  {
   RTIMING("HUGE memset");
   memset(b, c, len * 4);
  }
 }

 BeepExclamation();
}
```

I believe Fill3 does not have any weakness here... Actually, CLANG produced almost exactly the same code for Fill2 and memsetd for small fills, but I guess providing SSE2 implementation directly does not hurt anything. Plus we still like to have that cache bypass...

So I would go for Fill3 for X86 and Fill2 for non-X86 (in the hope it gets optimized for neon on ARM...)

Mirek
