Subject: MMX - no gain again...
Posted by mirek on Wed, 21 Jan 2009 11:57:09 GMT
View Forum Message <> Reply to Message

Well, I sort of knew it already, but gave it a try anyway - I have implemented MMX pixel blending path for Painter today.

The result is the same as before - MMX is no faster than plain C++.

It is really strange, as I would expect some difference There is virtually none.

These are my blending routines for the reference:

```
//#define USE_MMX

inline RGBA Mul8(const RGBA& s, int mul)
{
 RGBA t;
 t.r = (mul * s.r) >> 8;
 t.g = (mul * s.g) >> 8;
 t.b = (mul * s.b) >> 8;
 t.a = (mul * s.a) >> 8;
 return t;
}

inline RGBA MulA(const RGBA& s, int alpha)
{
 return Mul8(s, alpha + (alpha >> 7));
}

#ifndef USE_MMX

inline void AlphaBlend(RGBA& t, const RGBA& c)
{
 int alpha = 256 - (c.a + (c.a >> 7));
 t.r = c.r + (alpha * t.r >> 8);
 t.g = c.g + (alpha * t.g >> 8);
 t.b = c.b + (alpha * t.b >> 8);
 t.a = c.a + (alpha * t.a >> 8);
}

inline void AlphaBlendCover(RGBA& t, const RGBA& c, byte cover)
{
 AlphaBlend(t, MulA(c, cover));
}

inline void FinishBlend() {}
```

```
#else

#include <xmmintrin.h>

inline __m64 RGBAtoMMX(const RGBA& rgba)
{
 return _mm_unpacklo_pi8(_mm_cvtsi32_si64(*(dword *)&rgba), _mm_setzero_si64());
}

inline RGBA MMXtoRGBA(__m64 c)
{
 int x = _mm_cvtsi64_si32(_mm_packs_pu16(c, _mm_setzero_si64()));
 return *(RGBA *)&x;
}

inline void AlphaBlend(RGBA& t, const RGBA& c)
{
 __m64 alpha = _mm_set1_pi16(256 - (c.a + (c.a >> 7)));
 __m64 tx = RGBAtoMMX(t);
 tx = _mm_mullo_pi16(alpha, tx);
 tx = _mm_srli_pi16(tx, 8);
 tx = _mm_add_pi16(tx, RGBAtoMMX(c));
 t = MMXtoRGBA(tx);
}

inline void AlphaBlendCover(RGBA& t, const RGBA& c, byte cover)
{
 __m64 cx = _mm_srli_pi16(_mm_mullo_pi16(RGBAtoMMX(c), _mm_set1_pi16(cover + (cover
>> 7))), 8);
 __m64 tx = RGBAtoMMX(t);
 byte a = MMXtoRGBA(cx).a;
 __m64 alpha = _mm_set1_pi16(256 - (a + (a >> 7)));
 tx = _mm_mullo_pi16(alpha, tx);
 tx = _mm_srli_pi16(tx, 8);
 tx = _mm_add_pi16(tx, cx);
 t = MMXtoRGBA(tx);
}

inline void AlphaBlendCover(dword *t, dword s, const byte *c, int len)
{
 dword *e = t + len;
 __m64 zero = _mm_setzero_si64();
 __m64 src = _mm_unpacklo_pi8(_mm_cvtsi32_si64(s), zero);
 while(t < e) {
  byte cover = *c;
  __m64 cx = _mm_srli_pi16(_mm_mullo_pi16(src, _mm_set1_pi16(cover + (cover >> 7))), 8);
  __m64 tx = _mm_unpacklo_pi8(_mm_cvtsi32_si64(*t), zero);
```

```
  byte a = (dword)_mm_cvtsi64_si32(_mm_packs_pu16(cx, zero)) >> 24;
  __m64 alpha = _mm_set1_pi16(256 - (a + (a >> 7)));
  tx = _mm_mullo_pi16(alpha, tx);
  tx = _mm_srli_pi16(tx, 8);
  tx = _mm_add_pi16(tx, cx);
  *t = _mm_cvtsi64_si32(_mm_packs_pu16(tx, zero));
  t++;
  c++;
 }
 _mm_empty();
}

inline void FinishBlend()
{
 _mm_empty();
}

#endif
```

Later I might try SSE2, there is some small chance that Intel optimized only these, not MMX.

Mirek
