Using MMX™ Instructions to Implement Alpha Blending

Information for Developers and ISVs

From Intel® Developer Services
www.intel.com/IDS

March 1996

Information in this document is provided in connection with Intel® products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications. Intel may make changes to specifications and product descriptions at any time, without notice.

Note: Please be aware that the software programming article below is posted as a public service and may no longer be supported by Intel.

Copyright © Intel Corporation 2004
* Other names and brands may be claimed as the property of others.
CONTENTS

1.0. INTRODUCTION
2.0. OVERVIEW
3.0. DATA STORAGE
4.0. IMPLEMENTATION
5.0. PERFORMANCE
6.0. CODE LISTING
1.0. INTRODUCTION

The Intel Architecture (IA) media extensions include single-instruction, multidata (SIMD) instructions. This application note presents an alpha blending filter used for imaging effects. Specifically, the alpha blending function \( \text{alphaB} \) demonstrates how the MMX technology shifting capability can be used to organize and calculate the output array efficiently.

2.0. OVERVIEW

Alpha blending is used for imaging effects to merge two images together, weighting one image more than the other. Alpha blending can be used for fading from one image to another or for creating a translucent effect. For example, it can be used to put a 'ghost' image in a picture, or to place a window in an image. The equation for alpha blending is as follows:

\[
\text{s}[r,g,b] = \alpha \ast \text{p}[r,g,b] + (1 - \alpha) \ast \text{q}[r,g,b]
\]

where \( \alpha \) is the weighting factor, \( 0 \leq \alpha \leq 1 \), \( \text{p} \) and \( \text{q} \) are the input pixels, and \( \text{s} \) is the alpha blended pixel within the final image.

3.0. DATA STORAGE

The \( \text{p} \) array is made up of 32-bit elements. The lower 8 bits are the blue value of the pixel (\( \text{pb} \)), the next 8 bits are the green value of the pixel (\( \text{pg} \)), the next 8 bits are the red value of the pixel (\( \text{pr} \)), and the upper 8 bits are the \( \alpha \) value. This pixel format was chosen for this application note since the \( \text{p} \) array is commonly the result of calculations to determine the proper RGB \( \alpha \) values.

The \( \text{q} \) array is made up of 15-bit values. The lower 5 bits are the blue value of the pixel (\( \text{qb} \)), the next 5 bits are the green value of the pixel (\( \text{qg} \)), and the upper 5 bits are the red value of the pixel (\( \text{qr} \)). The resulting array \( \text{s} \) is made up of 15-bit values set up in the same order as the \( \text{q} \) array of pixels. This format, shown in Figure 1, was chosen for this application note since it is a common format for pixels already stored in a high color display buffer.

\[
\text{Figure 1. Bit Arrangement for p, q, and s Pixels}
\]
4.0. IMPLEMENTATION

The alpha blending equation was optimized for MMX technology as follows:

\[
\begin{align*}
    s &= \alpha \times p + (1 - \alpha) \times q \\
    s &= \alpha \times p + q - \alpha \times q \\
    s &= \alpha \times (p - q) + q
\end{align*}
\]

Figure 2 shows a register diagram of the pixel calculations that are done in the main loop L3. The loop L3 occurs n times where n is the number of rows times the number of columns in the images $p$ and $q$. Before the start of loop L3, register MM0 contains the $p$ pixel value and register MM7 contains the $q$ pixel value for the current calculation. After the L3 loop occurs four times the four new $s$ pixel values are stored over the old $q$ pixel values.

*Figure 2. Program Flow of Main Loop*
Using MMX™ Instructions to Implement Alpha Blending

March 1996

\[ aa = \alpha \]

\[
\begin{array}{cccc}
\text{MM7} & \text{MM0} & \text{MM2, MM3, MM4} & \text{MM1} \\
\text{qvqvb[4]} & 00aa & 00pr & 00pg \\
\text{qvqvb[3]} & 00pr & 00pg & 00pb \\
\text{qvqvb[2]} & 00pg & 00pb & \text{XXXX} \\
\text{qvqvb[1]} & \text{XXXX} & \text{XXXX} & \text{XXXX} \\
\end{array}
\]

\[ \text{PUNPCKHWDD mm1, mm1} \]

\[
\begin{array}{cccc}
\text{MM1} & \text{MM2} & \text{MM4} & \text{MM0} \\
00aa & \text{00aa} & \text{XXXX} & \text{XXXX} \\
\end{array}
\]

\[ \text{PUNPCKHDQ mm1, mm1} \]

\[
\begin{array}{cccc}
\text{MM1} & \text{MM2} & \text{MM4} & \text{MM0} \\
00aa & \text{00aa} & \text{XXXX} & \text{XXXX} \\
\end{array}
\]

\[ \text{PAND mm2, maskr} \]

\[ \text{PSLLQ mm2, 25} \]

\[
\begin{array}{cccc}
\text{MM2} & \text{MM4} & \text{MM3} & \text{MM1} \\
0000 & 00000000qr000 & 0000 & 0000 \\
\end{array}
\]

\[ \text{PAND mm3, maskg} \]

\[ \text{PSLLQ mm3, 14} \]

\[
\begin{array}{cccc}
\text{MM3} & \text{MM4} & \text{MM2} & \text{MM1} \\
0000 & 000000000g000 & 0000 & 0000 \\
\end{array}
\]

\[ \text{PAND mm4, maskb} \]

\[ \text{PSLLQ mm4, 3} \]

\[
\begin{array}{cccc}
\text{MM4} & \text{MM2, MM3, MM4 OR'd together} & \text{MM1} & \text{MM0} \\
0000 & 0000000000000 & 0000 & 0000 \\
\end{array}
\]

\[ \text{PSUBW mm0, mm2} \]

\[ \text{PMULLW mm0, mm1} \]

\[ \text{PSLLW mm2, 8} \]

\[ \text{PADDW mm2, roundf} \]

\[ \text{PADDW mm2, mm0} \]

\[
\begin{array}{cccc}
\text{MM2} & \text{MM4} & \text{MM3} & \text{MM1} & \text{MM0} \\
0000 & \text{aa*(pr-qr)} & \text{aa*(pg-qg)} & \text{aa*(pb-qb)} & \text{aa*(pr-qr)} \\
\end{array}
\]

\[ \text{MOVQ mm3, mm2 and MOVQ mm4, mm3} \]

\[
\begin{array}{cccc}
\text{MM2} & \text{MM4} & \text{MM3} & \text{MM1} & \text{MM0} \\
0000 & \text{aa*(pr-qr)} & \text{aa*(pg-qg)} & \text{aa*(pb-qb)} & \text{aa*(pr-qr)} \\
\end{array}
\]

\[ \text{PSLLQ mm2, 15} \]

\[ \text{PAND mm2, maskrH} \]

\[
\begin{array}{cccc}
\text{MM2} & \text{MM4} & \text{MM3} & \text{MM1} \\
0sr0000000000000 & 0000 & 0000 & 0000 \\
\end{array}
\]

\[ \text{PSLLQ mm3, 26} \]

\[ \text{PAND mm3, maskgH} \]

\[
\begin{array}{cccc}
\text{MM3} & \text{MM4} & \text{MM2} & \text{MM1} \\
000000000000000 & 0000 & 0000 & 0000 \\
\end{array}
\]

\[ \text{PSLLQ mm4, 37} \]

\[ \text{PAND mm4, maskbH} \]

\[
\begin{array}{cccc}
\text{MM4} & \text{MM2, MM3, MM4 OR'd together and} & \text{mm5, mm2, mm3, mm4 OR'd together and} & \text{MM1} \\
000000000000000 & 0000 & 0000 & 0000 \\
\end{array}
\]

\[ \text{MM5} & \text{MM4} & \text{MM3, MM4 OR'd together and} & \text{MM1} \\
0sr0000000000000 & 0000 & 0000 & 0000 \\
\]
Notice how the $\alpha$ value is replicated to perform the transformation. In the equation, $\alpha$ must be multiplied by each component of the result of the $p-q$ calculation, namely $(p-q)_{\text{red}}$, $(p-q)_{\text{green}}$, and $(p-q)_{\text{blue}}$. To do this the $\alpha$ comes from the $p$ pixel and is replicated three times for use with each component. This can be accomplished by using the PUNPCKHWD and PUNPCKHDQ instructions with the same register specified for both inputs, as shown in Figure 3.

**Figure 3. MMX Code Trick for Data Manipulation**

\[
\begin{align*}
\text{MM1} & : 00a & 00pr & 00pg & 00pb \\
\text{PUNPCKHWD} & : \text{mm1, mm1} \\
\text{MM1} & : 00a & 00a & XXXX & XXXX \\
\text{PUNPCKHDQ} & : \text{mm1, mm1} \\
\text{MM1} & : 00a & 00a & 00a & 30a
\end{align*}
\]

Approximately 92% of the instructions in the L1 loop are paired in the code presented in this paper. It might be possible to reduce the number of registers used in this loop to four so that the loop may be unrolled to improve the pairing. Also, the optimized technique used in *Using MMX Instructions to Convert 24-Bit True Color Data to 16-Bit True High Color*, Application Note AP-553 (Order number 243038) for the final packing of the result to 16-bit color format could be used to make the implementation more efficient. For these enhancements, the algorithm presented here would have to be reimplemented.
5.0. PERFORMANCE

The following cycle calculations were done by placing assembler tracing calls around both the C function call and the MMX function call that performs the alpha blending operation. The C code was compiled with a maximum speed compiler switch. The input images are 72x58 pixels. The code was traced, and the data file was used to generate a log file which contains the cycle information for each case.

For the scalar version of the alpha blending function:

- 936166 total cycles / 4176 pixels 224 cycles / pixel

For the MMX technology version of the alpha blending function:

- 118562 total cycles / 4176 pixels 28 cycles / pixel
6.0. CODE LISTING

;*************************************************************************/
;*              Copyright (c) 1996 Intel Corporation.
;*                      All rights reserved.
;*************************************************************************/
;*************************************************************************/
;*************************************************************************/
;*  File : alphammx.asm                                     Date : 3/4/96
;*;*
;*  Description:
;*  This routine computes the alpha blending of two 8-bit RGB
;*  images as follows :
;*  s[r,g,b] = aa * p[r,g,b] + (1-aa) * q[r,g,b]
;*
;*  IMPORTANT!!!
;*  This function requires that the total number of pixels in
;*  the image to be a multiple of 4, i.e. nx*ny = 4N
;*  Revision History:
;*  Name                    Date        Description
;*  ---------------------------------------------------------------------
;*;*************************************************************************/
title   alphaB
include iammx.inc
.486P
.model  flat,c
.data
zeros dq   0h
roundf  dd  800080h,       80h
maskr dd    7c00h,        0h
maskg dd     3e0h,        0h
maskb dd      1fh,        0h
maskrH dd       0h, 7c000000h
maskgH dd       0h,  3e00000h
maskbH dd       0h,   1f0000h
.code
alphaB proc near C uses esi edi eax ebx ecx edx,
p_ptr : ptr dword,
q_ptr : ptr word,
xn : ptr word,
yn : ptr word,

mov   esi, p_ptr       ; esi = p
xor   eax, eax         ; eax = 0
mov   edi, q_ptr       ; edi = q
xor   ecx, ecx         ; ecx = 0
mov   ax, word ptr nx  ; ax = nx
mov   cx, word ptr ny  ; cx = ny
alphaB endp

alphaB  proc near C uses esi edi eax ebx ecx edx,
Using MMX™ Instructions to Implement Alpha Blending

March 1996

```assembly
    imul   eax, ecx ; eax = nx*ny

L1:    movq mm7, mmword ptr [edi] ; mm7 = qqq4 qqq3 qqq2 qqq1
      pxor mm5, mm5 ; mm5 = 0000 0000 0000 0000
L2:    movq mm6, mmword ptr [esi] ; mm6 = a2r2 g2b2 a1r1 g1b1
      movq mm0, mm6 ; mm0 = xxxx xxxx aarr ggbh
      add esi, 8 ; esi += 8
L3:    punpcklbw mm0, mmword ptr zeros ; mm0 = 00aa 00rr 00gg 00bb
      movq mm2, mm7 ; mm2 = xxxx xxxx xxxx qqqq
      pand mm2, maskr ; mm2 = 0000 0000 0000 qq00
      movq mm3, mm7 ; mm3 = xxxx xxxx xxxx qqqq
      punpckhwd mm1, mm1 ; mm1 = 00aa 00aa xxxx xxxx
      pand mm3, maskg ; mm3 = 0000 0000 0000 0qq0
      psllq mm4, mm7 ; mm4 = 0000 0000 0000 0qq0
      psllq mm3, 14 ; mm3 = 0000 0000 0000 0qq0
      psllq mm4, maskb ; mm4 = 0000 0000 0000 00qq
      punpckhdq mm1, mm1 ; mm1 = 00aa 00aa 00aa 00aa
      por mm2, mm3 ; mm2 = q + round'g factor
      por mm2, mm4 ; mm2 = (p-q)*aa
      psllq mm4, 3 ; mm4 = 0000 0000 0000 0000
      psllq mm2, 8 ; mm2 = 0000 qr00 qg00 qb00
      paddw mm2, roundf ; mm2 = q + round'g factor
      pmullw mm0, mm1 ; mm0 = (p-q)*aa
      psrlq mm5, 16 ; mm5 >>= 16
      psrlq mm7, 16 ; mm7 >>= 16
      nop
      paddw mm2, mm0 ; mm2 = (p-q)*aa+q + round'g factor
      movq mm0, mm6 ; mm0 = xxxx xxxx aarr ggbh
      movq mm3, mm2 ; mm3 = mm2
      psllq mm2, 15 ; mm2 = rrxx xxxx xxxx xxxx
      movq mm4, mm3 ; mm4 = mm3
      psllq mm3, 26 ; mm3 = xggx xxxx xxxx xxxx
      pand mm2, maskrH ; mm2 = rr00 0000 0000 0000
      psllq mm4, 37 ; mm4 = xxbb xxxx xxxx xxxx
      pand mm3, maskgH ; mm3 = 0gg0 0000 0000 0000
      por mm5, mm2 ; mm5 = sss0 0000 0000 0000
      por mm5, mm3 ; mm5 = ssss SSSS SSSS SSSS
      por mm5, mm4
      dec eax ; -- eax
      test eax, 1
      jnz L3
      test eax, 2
      jnz L2
      movq mmword ptr [edi], mm5 ; save 4 alpha blended words in q
      cmp eax, 0
      je L4
      add edi, 8 ; edi += 8
      jmp L1
L4:    emms
      ret
alphaB endp
end
```