# Optimization Techniques for Mobile Graphics and Gaming Applications using Intel ® Wireless MMX ™ Technology


Moinul H. Khan
Intel Corporation
Moinul.H.Khan@intel.com

Bradley.C. Aldrich
Intel Corporation
Bradley.C.Aldrich@intel.com

Allen Hux
Intel Corporation
Allen.Hux@intel.com

Nigel C. Paver
Intel Corporation
Nigel.Paver@intel.com



## Abstract

Mobile multimedia is growing at a startling rate, fueling the trend toward rich gaming and graphics on mobile devices. The demand for improved performance is driving new products and architectures such as the Intel® PXA27x processor family with Intel® Wireless MMX™ Technology. These architectures are designed to accelerate graphics and gaming applications. In this paper we discuss a set of techniques for optimizing graphics and gaming application for Wireless MMX™ Technology and PXA27x processor family in general. These techniques are based on features of Wireless MMX Technology and characteristics of the graphics and gaming applications in wireless domains. We demonstrate average performance improvement seen by the optimization and programming techniques. We use vertex transformation, raster operation, shading, blending and Z-buffering as typical graphics algorithms.


## Introduction

Both Video gaming and graphics are emerging to be key applications for wireless platforms. Algorithms used in graphics and gaming applications demonstrate homogeneity of operations and sub-word level parallelism. These characteristics make them suitable for SIMD acceleration. However, acceleration of the graphics and gaming applications also requires a system oriented view. In this paper we outline and demonstrate different optimization techniques which can be applied in Wireless MMX technology and also the system level techniques which can be applied on Intel® PXA27x family of processors [1] , which are highly integrated wireless systems on a chip.

## Intel® PXA27x Processor Family

In wireless platforms area, performance, power and cost and key metrics for product success. This is driving increasing levels of on-chip integration in state of the art application processors. The Intel® PXA27x processor family is a highly integrated System-on-a-Chip (SoC) targeting wireless and handheld platform. Figure 1 shows an overall block diagram of the Intel® PXA270 processor and shows the Intel® XScale® microarchitecture and Intel® Wireless MMX™ technology as key features. It also includes 256KB of SRAM memories which is useful in video and graphics applications as a frame buffer. The PXA27x processor also provides multimedia components such as an LCD controller, camera interface, and an extensive set of peripheral interfaces such as UART, USB, AC97, SSP and I2S. The memory subsystem of the XScale core contains 32 KB caches for both instruction and data. The PXA27x also supports a wide range of flash memory card interfaces for program and data storage.

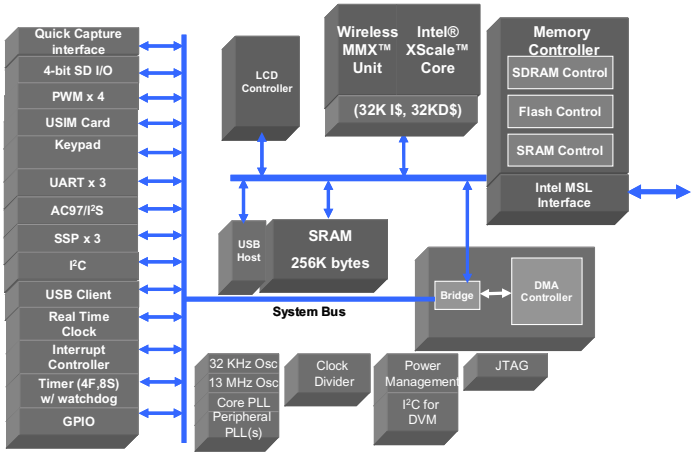


**Figure 1. PXA27x Processor Microarchitecture**

## Intel Wireless MMX™ Technology

Significant research effort and desktop processor development has been under-taken related to SIMD processing for media applications [2][3][4][5]. Wireless MMX Technology [6][7][8] is one of the newest architectures. It integrates equivalent functionality of all of Intel® MMX™ Technology [9] and the integer functions from SSE [3] to the Intel XScale microarchitecture [10].

Like MMX technology and SSE, Wireless MMX technology utilizes 64-bit wide SIMD instructions, which allow it to concurrently process up to eight data elements in a single cycle. Wireless MMX technology defines three packed data types (8-bit byte, 16-bit half-word and 32-bit word) and the 64-bit double word. The elements in these packed data types may be signed or unsigned.

The Wireless MMX™ unit is a tightly coupled coprocessor of the XScale® microarchitecture. The programmer's model is a straightforward extension of the XScale microarchitecture programming model. Thus, a multimedia application can maintain a single thread of control while taking advantage of the SIMD acceleration only on the critical section of the algorithm. As Wireless MMX is an extension of the XScale microarchitecture, it

utilizes the existing memory subsystem. This eliminates the need for dedicated memory and the corresponding increased power consumption.

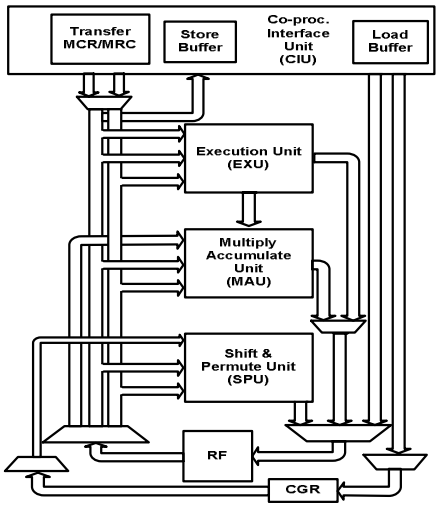


**Figure 2: Intel® Wireless MMX™ Microarchitecture**

The Wireless MMX functional units are illustrated in Figure 2. The shift and permute unit is responsible for performing shift and permute operations. These operations include; alignment, shift, rotation, packing and shuffling. The execute unit is responsible for performing arithmetic and logic operations and it also provides a saturation capability. The multiply and accumulator unit is responsible for performing all multiply and multiply-accumulate operations. The coprocessor interface unit transfers data between the Wireless MMX unit registers and the Intel XScale microarchitecture. In addition to supporting coprocessor data transfer, the XScale core is also responsible for storing and loading data to and from the memory. The main Register File (RF) is organized as sixteen 64-bit registers, located in the coprocessor 0 space (CP0) and the Control Register File (CGR) is organized as four 32-bit registers in coprocessor 1 space (CP1) [10]. The main register file is used to hold SIMD operands and results and the control register file is used to hold alignment and shift values.

## Mobile Graphics and Gaming Workload

Majority of the gaming workloads are comprised of graphics functionality. Graphics workload has two major components, they are Geometry processing and Rasterization, typically occupying 20% and 75% of the graphics workloads respectively [15]. A typical graphics processing pipeline is shown in Figure 4 [12].

The main steps of geometry processing comprises of performing model view transformation, per-vertex lighting and texturing. It also assembles the primitives by clipping and perspective transformation. Most of these operations are done using floating-point or high precision fixed-point operations. The biggest contributor to the graphics chain workload is the Rasterization processing. At this stage, the transformed triangles are drawn out on the screen one line at a time. While working on each line, a series of operations are performed on a per-pixel basis. Rasterization can be further divided into a set of dominant subcomponents, as follows[11]:

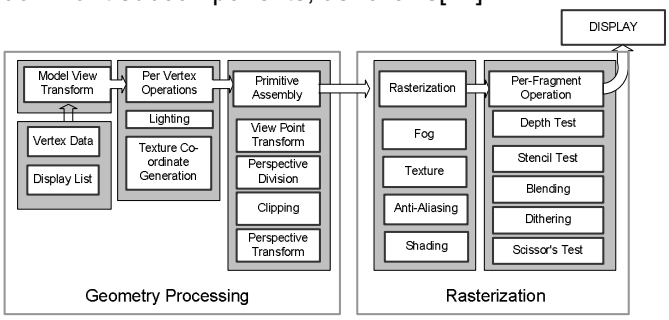


**Figure 3. Typical fixed function graphics pipeline**

**Pixel Depth:** During writing or rendering each pixel, depth (or distance from the eye) is compared to ensure that the nearest pixel overwrites pixels that are farther away (called Depth test).

**Shading** is performed to incorporate such variation in intensity while filling the faces of the graphics primitive. This step fills in the line with a gradual intensity of color considering lighting condition and relative orientation between the eye and the graphics object.

**Fogging** is used on the graphics primitives to make it appear distant.

**Texture mapping** is performed on the object's surface to reflect the material type – which makes the scene look more realistic.

**Anti-aliasing** is performed to reduce visual artifacts introduced by the discrete nature of the digital display. This helps improve the perceptual quality of smaller fonts on the smaller display of the hand-held device.

Most of the rasterization operations are performed in fixed-point format in 8-bit, 16-bit and 32-bit precision. These pixel operations are uniform across different pixels and also across different color components of each pixel. Thus, SIMD processing architecture such as Intel® Wireless MMX™ technology can be used effectively to gain performance improvements on these algorithms.

## Architectural Features for Graphics

Intel® Wireless MMX™ Technology has a several architectural features which improve performance and power efficiency for graphics applications. These features include:

**Instruction support**: Specialized instruction support such as WMADD – Multiply and add assists bilinear interpolation etc. Similarly, WALIGN (Align instructions) handle pixels if they are not aligned conveniently. Following the ARM architectural concept, all the Intel Wireless MMX Technology instructions are conditionally

executed. Conditional execution assists efficient execution of tight loops.

**Register cache:** Wireless MMX Technology has large register file (16 64-bit registers). Register file can be effectively used as L0 memory and this feature reduces number of memory load and stores.

**Data prefetch**: Wireless MMX technology supports data prefetch into the 32KByte data caches of the XScale® microarchitecture and allows multiple outstanding loads operations to mitigate the effects of memory latencies

At the PXA27x processor system level, architectural features such as the on-chip SRAM can be used for applications and also as LCD frame-buffer, enhances power and performance for gaming and graphics.

## Performance Optimization Methods

The three steps of optimizing graphics workload for mobile clients are:

- § Algorithmic optimization
- § Computational optimization
- § Memory based optimization

The following sections will offer a set of optimization approaches suitable for Wireless MMX Technology and the PXA27x processor family.

### *Algorithmic Optimization*

In many hand-held multimedia devices the display sizes are small and the user may also be viewing the games or the graphics content while in motion or outdoors. This may imply that the user may not be extremely discerning of the perceptual quality. Thus, choosing the correct level of detail is important. Similarly, there can be trade-offs to be made between the number of triangles and the amount of texturing done per triangle. Using more textures may increase the perceived quality more than increasing the number of triangles [12].

### *Computational Optimization*

Computational optimization techniques can be subdivided into a number of distinct considerations, including; precision, data format, alignment, resource usage and pipelining that an application developer can use to accelerate graphics applications for the PXA27x processor family.

**Precision Management**: The developer ensures that computational inaccuracy due to reduced bit-width and rounding does not introduce any visual artifacts. There is a trade-off in using higher precision and accuracy versus cost and visual impact. For example, geometry pipelines in wireless platforms today use fixed point arithmetic rather than floating-point operations found in desktop systems. Geometry can be processed in 32-bit precision fixed point format. On the other hand, for a rasterization pipe, 8-bit precision for color (RGB component) is not sufficient for most operations, but on the other extreme 32-bit precision for each color does not deliver significantly greater perceptual quality compared to 16-bit color component precision. In addition, using 16-bit precision as opposed to 32-bit exposes more data parallelism which can be accelerated with Wireless MMX technology. For these reasons 16-bit precision is the recommended precision for handheld graphics as it balances quality with performance. Similarly, for Z-buffering algorithm, ~15% improvement can be achieved by using lower precision (16-bit instead of 32-bit) alone. However, a precision error may lead to displaying some occluded primitives leading to an unpleasant viewing experience. Thus, depth-buffer or Z-buffer management may be better handled in 32-bit format despite the potential gain otherwise.

**Data Format:** In the processing chain, each primitive (i.e. vertex, triangle) is characterized with a set of attributes. A Programmer typically defines all these attributes as a structure. While describing an array of these primitives an array of this record type is used. Algorithms can be designed such that different members from the same structure are operated on concurrently. On the other hand, some algorithms may impose operation on members from different structure concurrently. These two approaches are called, AOS (Array of Structures) and SOA (Structure of Arrays). SOA is traditionally perceived to be SIMD-friendly, whereas AOS is thought of as software-development friendly. However, Intel Wireless MMX Technology can operate easily on both the types of data organization. Larger register file can be used to store multiple of the data points (structures) so that SOA to AOS and vice versa conversion can be done efficiently. However, optimal performance can be obtained by using a hybrid approach (between SOA and AOS). Figure 4 shows different organizations of data.

```
-----------------------------------------------
// AOS Array of Structure
struct {float x, y, z, r, g, b }AoS_xyz_rgb[200];
-----------------------------------------------
// SOA Structure of Array
struct { float x[200], y[200], z[200];
float r[200], g[200], b[200]; } SoA_xyz_rgb;
-----------------------------------------------
// Hybrid SOA
struct { float xx[4], yy[4], zz[4]; } Hybrid_xyz[50];
struct { float rr[4], gg[4], bb[4]; } Hybrid_rgb[50];
```

**Figure 4. Data formatting**

Hybrid structures may be used in both the front end of the graphics-processing pipe as well the back end. For instance, each object is presented as a set of triangles organized as AOS, whereas each triangle may contain three vertices organized as SOA. Similarly at the back

end, the depth buffer can be represented as AOS, where each structure holds an array of depth buffer entries (1D or 2D). Hybrid structures can be helpful for prefetching.

**Alignment Considerations:** Intel® Wireless MMX™ technology offers two-, four-, and eight-way data parallelism, whereas typical color formats are RGB (three elements). Thus, the algorithm implementation should take alignment into consideration. While defining and placing data structure they should be aligned on a 64-bit address (e.g. the commonly used elements RGBA or XYZW are 64-bit wide and should be placed on a 64-bit boundary). This will help reduce the alignment overhead while processing in Intel Wireless MMX technology. For example, assume that an array of pixel's color format is as follows:

```
typedef {
U16  r; U16  g;  U16  b; U16  a;
//U16 represents unsigned shorts.
}COLOR_DATA;
```

If the variable input is not aligned with a 64-bit addressing boundary, loading each of the pixels will require two loads (WLDRD) and an align (WALIGN) operation. Similar arguments can be extended towards aligning data structure to the cache line boundary for memory considerations, which is discussed in a later section.

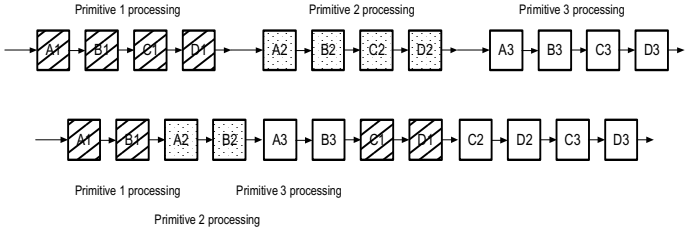


**Figure 5. (a) Fusing multiple algorithmic stages for each primitive (b) Pipelining Between Multiple Primitives**

**Level of Pipelining:** Typical graphics algorithm operates in a pixel-by-pixel or primitive-by-primitive manner. Regular software pipelining should be applied to inner loops to operate on multiple primitives concurrently in a single iteration. e.g. while processing current pixel load next pixel and store previous pixels. Larger register files can hold data for multiple pixels at a time, helping multi-pixel/multi-primitive techniques. Larger register file of wireless MMX helps redesign algorithms to achieve this. Apart from pixel/primitive level pipelining, multiple algorithmic stages can be fused into a single iteration of processing. Figure 6 shows how an algorithmic pipelining approach and primitive-level pipelining may need to work. In this figure, A, B, C, and D are per-pixel or per-primitive algorithmic stages. Both the pipelining approach allows better reuse of data because it increases spatio-temporal locality of the data.

### *Memory Related Optimization*

The performance of graphics applications are in general tied very closely to the latency and bandwidth to the attached memories. For embedded systems, memory and access bandwidth are at a premium and they are shared by many system components. Thus, optimizing for memory and access bandwidth utilization is extremely important.

**Memory footprint** is defined by the sum of data and instruction space that the application consumes. Memory footprints are typically reduced by lower precision depth-buffer, reusing texture maps and reducing number of back-buffers used for application and LCD co-ordination.

**Data placement:** PXA27x has a hierarchical memory subsystem. Each type of memories varies in its memory latency and memory throughput. Different data objects of the application can be placed in the memory in a way such that critical objects are in the memory space with lower latency. For example, depth buffer can be stored in the internal memory for faster access. Similar approach can be taken for texture maps (partially or full). Memory accesses during texture processing are sufficiently random to take advantage of the caching. Large texture maps can be divided into two or more segments and relevant segments can be copied into the SRAM as it is needed.

**Memory Latency** The impact of memory latency under cache misses impact directly the performance of the application. Memory latency can be mitigated during application execution by loading data early or by prefetching data into the caches. Performing an aggressive preloading scheme is very effective in hiding memory latency. Specifically for graphics, preloading can be done in different levels, shown in the figure 6.

Preloading can be done at different levels. While processing one pixel ($N^{th}$ ), the data necessary for a pixel down the line ($(N+P)^{th}$) should be preloaded. For pixel level processing, the depth of the pixel is compared with the depth buffer value. Depth buffer values can be pre-fetched easily. Similarly, the preloading schemes can be extended to scanline level preloading and primitive level preloading. Scan line level preloading helps hide latency for accessing alpha-plane values and blend images, depth buffer value and texture maps. On the other hand, primitive level preloading can help vertex processing. Preload distance for each of the cases should be determined the following way: Preload distance

$P_{Distance} = N_{Mem\_Latency} / N_{Process\_Iteration}$

$N_{Mem\_Latency}$ = Num cycles for memory access

$N_{Process\_Iteration}$ = Num cycles /iteration of processing

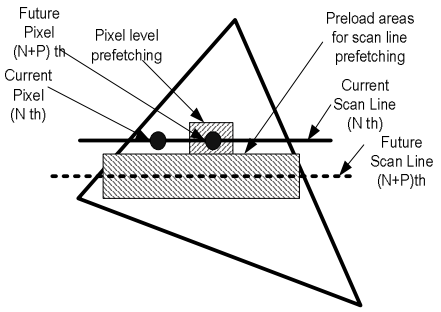


**Figure 6. Pixel Level or Scan Line Level**

The memory latency for PXA27x is dependent on the memory type (on-die SRAM, external SDRAM) and operating frequency. The number of cycles of processing depends on the algorithm being used.

**Cache Usage:** In the graphics processing pipe data structures differ according to the stage in which they are used: vertex representation, scanline representation, or output color representation. The data structure should be designed to be one or a multiple of the cache-line size and should be aligned on a cache-line boundary. This approach helps preloading the data structure. For instance, if a data structure is one cache-line size but is not aligned at the cache-line boundary, accessing the data structure can cause two cache misses. Also, the typical pre-fetching approach might only get one part of the data structure. This concern applies to both the data organization methods shown before (Structure of Array (SOA) and Array of Structure (AOS)).

## Case Study

Consider a case of a rasterization engine shown in figure 7. The engine has the following steps. It performs depth-test for each pixel in the Scan Line at the beginning to avoid any rendering operation for occluded vertices. For each pixel, texture value is looked up from the texture map (with bilinear interpolation for non-integer texture indices). Following the texture mapping, shading is performed. Following the color shading, blending is performed with the back-ground or with a fogging color (or both). This represents a typical rasterization flow. The scene content was comprised of a set of flying tetrahedrons (decomposed into triangles). Small texture map (~16Kbyte) was used for rendering. We focused on each of the algorithms to seek data parallelism and optimization opportunity based on Wireless MMX Technology. Following the inner-loop based optimization strategy, we focused on the memory and system level optimization using the approaches discussed in the previous sections.

As a first step of optimization, the key inner-loops in the rasterization engine were studied and were individually optimized. Optimization was performed on each key kernels based on the techniques shown in this paper. We compared the optimized inner-loops with scalar implementation of the same. For scalar implementation we targeted a typical ARMv5 processor. Table 1 shows the average acceleration gained by applying the optimizations on the inner loops. The measurement does not include loop and function call overhead and assumes that the data is already in the cache and only computational optimizations were applied. It can be noted that typical improvement of the key algorithm kernels reflects the data parallelism available in these algorithms. For depth buffer testing, the parallelism achieved is only 2x since it uses 32-bit precision. In contrast, shading operations (16-bit) can extract 4x data parallelism and rotation example (byte based) can attain 8x data parallelism. Rotation example can be benefited by the large register file, since a block of 8x8 pixels can loaded and rotated before writing back. It can be also be observed that for alpha-blending and fogging, it uses two parallel multiplication provided by the Wireless MMX Technology and the same is reflected in the acceleration. The table also captures a list of optimization schemes, Wireless MMX instructions used to gain the acceleration.

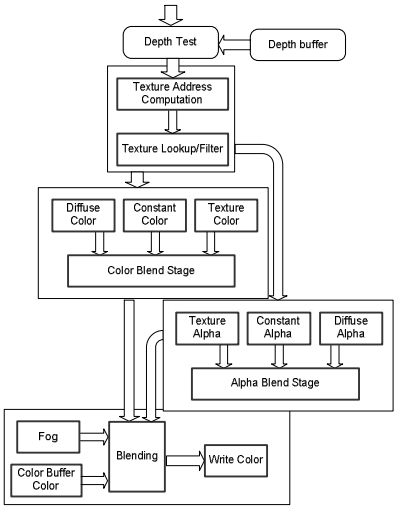


**Figure 7. Simplified Rasterization Engine used for experimentation**

While the inner loop speed up can be significant the impact at the application level is much reduced and dependent on many system level considerations. Acceleration observed in the rasterization case-study depends on the number of polygons used for rasterization, and also on the chosen mode for coloring and alpha-blending. We compared the speed of execution of the rasterizer under different coloring and alpha-blending mechanism. The summary of the application level impact is summarized in Table 2. The measurement was performed on a PXA27x platform running WinCE embedded operating system.

**Table 1. Acceleration for different graphics algorithms inner loops with proposed optimizations**

| Target Algorithm Kernel | Optimization Techniques applied | Key Wireless MMX Instruction Used | Improvement |
|---|---|---|---|
| Gouraud Shading | § Pipelining,<br>§ computational optimization (avoiding divide)<br>§ using larger register file | wadd, wshuf, walign | 2.4x |
| Texture Mapping | § Preloading (pld),<br>§ 16-bit-parallel multiply and accumulate (wmac),<br>§ cache-optimization,<br>§ using larger register file | wmac | 2.2x |
| Trilinear Interpolation | § 32-bit mac (tmia),<br>§ cache-optimization | tmia | 2.2x |
| Z-buffering | § group conditional execution<br>§ preloading (pld),<br>§ cache-friendly structure definition | wstrne, wmax | 1.1x |
| Alpha Blending and Fogging | § parallel multiply and add (wmadd),<br>§ larger register file | wmadd | 1.7x |
| Anti-Aliasing (with a fixed 3x3 Kernel) | § simplifying multiplication into parallel additions (wadd) accumulation (wacc) and shifts, preloading (pld) | wadd, wacc, We | 1.8x |
| Raster Operations | § preloading<br>§ SIMD logical operations | wxor, wand, wor | 2x |
| Vertex Transformation | § 32-bit mac (tmia),<br>§ cache-friendly structure definition<br>§ Using different registers as running accumulators | tmia | 2x |
| Screen Rotation | § Using large register file, memory preloading (pld) | wshuf | 4x |

It can be noted that the application level impacts are lower compared to the kernel or inner-loop level impacts, which is because of the un-avoidable cache misses, the corresponding memory latency and also the control overhead etc. However, the control overhead can be optimized further while working with a larger image or more polygons compared to the case study here. The acceleration observed for these application does translate to power savings, since, for rate constrained application such as gaming, higher performance means, the device can be operated at a lower clock frequency and thus saving on power and voltage.

The optimization techniques demonstrate impressive performance gain of Wireless MMX Technology for 3D graphics inner-loops and positive improvement for graphics application. We believe that, application level acceleration can be improved further by choosing a scene more number of polygons such that some of the control overhead in the application could be further optimized.

**Table 2. Comparing acceleration in PXA27x compared to Scalar implementation under different coloring and alpha blending scheme.**

| Coloring Scheme | Alpha Scheme | Preliminary Acceleration |
|---|---|---|
| Constant | Smooth | 1.23x |
| Smooth | Smooth | 1.18x |
| Texture | Texture | 1.17x |
| Smooth Texture | Smooth Texture | 1.22x |

## Conclusion

The paper discussed algorithmic, computational and memory optimization techniques for graphics applications. Different features of Wireless MMX Technology and PXA27x were described and applicability to optimize graphics pipeline were discussed. Initial acceleration is presented for a set of key graphics processing algorithms.

## Acknowledgements

We acknowledge the significant contribution of the entire Wireless MMX technology development team in Austin, TX and Chandler, AZ and the PCG systems engineering in Hudson, MA.

## References

[1] Intel® PXA270 Processor For Embedded Computing Product Brief, http://developer.intel.com/design/embeddedpca/prodbref/302302pb.htm
[2] Alex Peleg and Uri Weiser, "MMX Technology Extension to Intel Architecture", IEEE Micro, 16(4):42-50, Aug. 1996.
[3] Keith Diendroff, "Pentium III = Pentium II+ SSE", Micro Processors Report, 13(3):6-11, March, 1999.
[4] Lee, Ruby, "Subword Parallelism in MAX-2", IEEE Micro, 16(4), 51-59, Aug. 1996
[5] Tremblay, Marc, et. al. "VIS Speeds Media Processing", IEEE Micro, 16(4): 10-20, Aug. 1996.
[6] N. C. Paver et. al. "Accelerating Mobile Video with Intel® Wireless MMX™ Technology", IEEE Workshop on Signal Processing Systems (SIPS), Aug 27-29, 2003.
[7] N. C. Paver et. al. "Intel® Wireless MMX(TM) Technology: A 64-Bit SIMD Architecture for Mobile Multimedia", International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2003.
[8] Paver, Nigel C., Bradley C. Aldrich, Moinul H. Khan, Programming with Intel® Wireless MMX™ Technology: A Developer's Guide to Mobile Multimedia Applications, Hillsboro, OR: Intel Press, 2004.
[9] Weiser, Uri, et al, "The Complete Guide to MMXTM Technology", Mcgraw-Hill, 1997, ISBN 0-07-006192-0.
[10] Intel XScale(R) Core Developer's Manual, http://www.intel.com/design/intelxscale/273473.htm
[11] James Foley, Andries van Dam, Steven Feiner, John Hughes, "Computer Graphics: Principles and Practice (second edition)", Addison-Wesley Publishing Company, Reading MA (1991), ISBN 0-201-12110-7
[12] David Eberly, "3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics", Morgan Kaufmann Publishers, San Francisco CA (2000), ISBN 1-55860-593-2.
[13] Furber, S.B., "ARM System-on-Chip Architecture", Addison Wesley, 2000. ISBN 0-201-67519-6.
[14] http://www.arm.com/aboutarm/55CE4Z/$File/ARM_Architecture.pdf
[15] Gopi K. Kolli et. Al. " 3D Graphics Optimizations for Intel® PCA Applications Processors with Intel® XScale™ Technology", www.intel.com/pca/developernetwork, Vol 3, Spring 2002.