Using Intel Streaming SIMD Extensions for 3D Geometry Processing

Wan-Chun Ma and Chia-Lin Yang

Dept. of Computer Science and Information Engineering
National Taiwan University
firebird@cmlab.csie.ntu.edu.tw,
yangc@csie.ntu.edu.tw

Abstract. Three dimensional (3D) graphics applications is an important workload running on today's computer system. A cost-effective graphics solution is to use a general processor for 3D geometry processing and a specialized hardware for rasterization. 3D geometry processing is an inherently parallel task. Therefore, many CPU vendors add SIMD (Single Instruction Multiple Data) instruction extensions to accelerate 3D geometry processing. In this paper, we evaluate the performance impact of using the Intel Streaming SIMD Extensions (SSE) for 3D geometry processing. We use SIMD-FP to improve the computational throughput by processing four vertices in parallel. We find that the layout of vertices in memory is important for the effectiveness of SIMD-FP. We also study the effect of using prefetch instructions to improve the memory performance. The experimental results show that using Intel SSE can achieve close to 4x speedup for geometry processing.

1 Introduction

Multimedia applications (e.g. speech, audio/video, image and graphics applications) have become important workloads running on general processors. This type of applications often presents data parallelisms. Therefore, one important architectural enhancement to accelerate multimedia applications is the SIMD (Single Instruction Multiple Data) instruction extensions. In 1996, Intel introduced the MMX technology [3], which packs 8-bit or 16-bit fixed-point data into a 64-bit register and performs arithmetic or logical operations on the packed data in parallel. The MMX works well for applications with integer data type, such as image and video processing. However, several visual and 3D graphics applications are floating-point intensive. To accelerate floating-point computation Intel develops the Streaming SIMD Extensions (SSE) [1] [4]. The key component of the SSE is the SIMD-FP extensions, which can process four single-precision floating-point values in parallel. Another important feature of the SSE is the memory streaming instruction extensions, which allow programmers to prefetch data into a specified level of the cache hierarchy. Most multimedia applications present the streaming data access pattern; that is, data are accessed sequentially

Y.-C. Chen, L.-W. Chang, and C.-T. Hsu (Eds.): PCM 2002, LNCS 2532, pp. 1080–1087, 2002.

and seldom reused. Therefore, prefetching this type of data into the L2 cache is an effective way to improve the memory system performance.

3D graphics is an important workload of nowadays multimedia applications. 3D graphics pipeline contains three stages: 1) database traversal, 2) geometry processing, and 3) rasterization. The first stage reads in the scene models and the second stage transforms 3D coordinates into 2D coordinates. Finally, the rasterization stage converts transformed primitives into pixel values and stores them in the frame buffer for display. For cost consideration, a commodity system usually uses the host processor for geometry processing and a custom hardware to accelerate rasterization. 3D geometry processing has streaming data access pattern and floating-point intensive computation. The vertex information (e.g. coordinate and color) are stored in the floating-point format and read sequentially from the storage. Geometry processing is an inherently parallel task since each vertex can be processed independently. Therefore, 3D geometry processing is one of the targeted applications for the SSE.

Previous studies on the SSE focused primarily on the usage and only analyze the performance effect for application kernels [2]. In this paper, we perform detailed performance analysis of using the SSE on the complete 3D geometry pipeline. We first evaluate the performance impact of using SIMD-FP and the effect of different data layout. We then analyze how much memory stall time can be eliminated through prefetching. Experimental results show that using SIMD-FP along can achieve close to 3x speedup, and arranging the vertices favorable to SIMD computation can further improve performance. We also find that prefetching vertices into the L2 cache one iteration ahead can eliminate most of the L2 cache misses. The overall speedup of using SSE in 3D geometry processing is up to 4x. The paper is organized as follows. Section 2 provides background information on the SSE and 3D geometry processing. Section 3 describes our experimental methodology. Section 4 presents the performance analysis. Section 5 discusses related work. Section 6 concludes this paper.

2 Background

In this section, we describe two main kernels of 3D geometry processing and illustrate how to apply the SSE to speed up the process.

2.1 3D Geometry Pipeline

The 3D geometry pipeline consists of two main kernels:

1. **Transformation:** 3D geometry processing contains three stages of coordinates transformation: viewing, modeling and projection. Each transformation requires a multiplication of a 1x4 vector and a 4x4 matrix. Hence, each transformation needs 12 multiplications and 16 additions.
2. **Lighting:** the lighting stage of the 3D geometry pipeline determines the color of each vertex. For each light source in the scene, the following illumination model is used to calculate the light intensity of a vertex [14]:

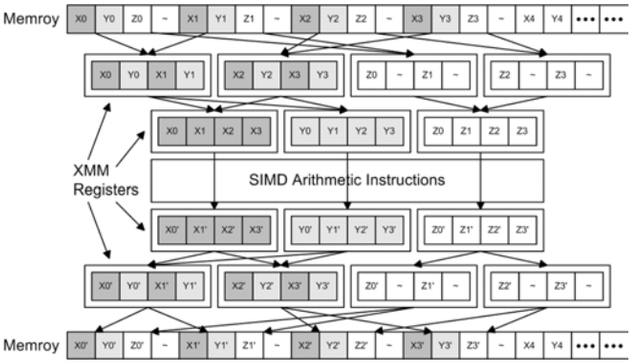

Fig. 1(a). Illustration of using SIMD-FP to process four vertices in parallel using AOS data structure .

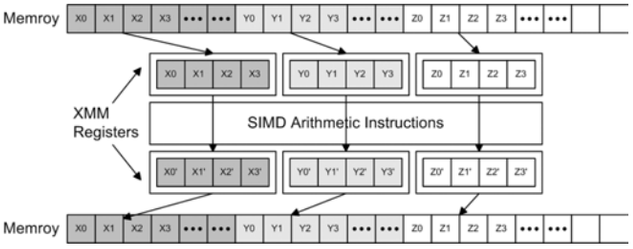

Fig. 1(b). Illustration of using SIMD-FP to process four vertices in parallel using SOA data structure .

$$I = k_a \times I_V + (\frac{1}{d}) \times (k_d \times I_L \times (N \cdot L) + k_s \times I_L \times V \cdot R^{n_s})$$

In this paper, we simplify the illumination model by discarding the specular component of the formula. Therefore, lighting calculation requires one division, 16 multiplications, 6 additions, 3 subtractions and one square root operation assuming single light source.

2.2 Using the SSE

To support the SSE Intel adds eight new 128-bit registers (XMM registers). Thus we can pack four single-precision floating-point operands into a register and use SIMD-FP to operate in parallel on all packed data operands. The most intuitive way to apply the SIMD-FP to 3D geometry processing is to exploit the parallelism between vertices as shown in Figure 1(a). The x (y and z) coordinates of four vertices are first packed into one XMM register. We then apply SIMD-FP arithmetic instructions to the packed data. The computed data is unpacked before stored back to the memory.

```
struct coordinate
{
  float x,y,z,w;
}

coordinate vertex[10000];
```

Fig. 2(a). AOS declaration

```
struct coordinate
{
  float x[10000]; float y[10000];
  float z[10000]; float w[10000];
}
coordinate vertex;
```

Fig. 2(b). SOA declaration

As we can see from the illustration, organizing data into the SIMD format incurs significant overhead. To avoid this overhead Intel proposes to transpose the data layout. The conventional approach stores vertices in memory using AOS (array of structures) format (see Figure 2(a)). Intel suggests to store vertices in the SOA (structure of arrays) format (see Figure 2(b)) such that the x (y and z) coordinate of different vertices are stored contiguously in the memory. Therefore, we can reduce the data packing/unpacking overhead for realizing SIMD computation in this new data layout as shown in Figure 1(b). In Section 4, we evaluate the effect of SIMD-FP using both data layout.

3D geometry processing has poor cache performance because of the large working set and streaming access patterns. Therefore, to improve the cache performance we can use the prefetching instructions provided in the SSE to reduce memory stall time. Prefetching hides memory latency by bringing data close to the CPU earlier than demand fetches. The following pseudo code segment shows the usage of prefetch instructions:

```
for i = 0 to # of vertices
   prefetch vertex[i+x];
   process vertex[i]; /* computation on a vertex*/
end loop
```

The variable x controls the prefetching distance; that is how far ahead we need to prefetch data in order to completely hide memory latency. The amount of computation on each vertex and memory latency determines the value of x.

3 Experimental Methodology

We evaluate the SSE on a Pentium 4 processor running Window 2000. The processor and memory configurations are summarized in Table 1. We first implement the 3D geometry pipeline in C and then modify it to use the SSE assembly

Table 1. System configuration

CPU	Intel Pentium 4 1.5GHz	
	L1 Data Cache	4-way, 8K byte
	L2 Cache	8-way, 256K byte
Memory	256M byte 440MHz RAMBUS	

Fig. 3. 3D models used in the experiments.

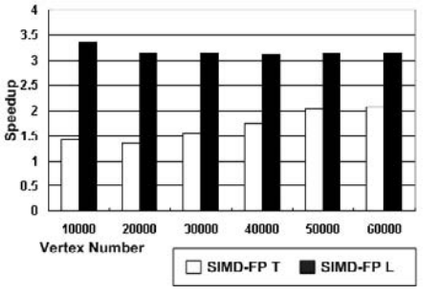

Fig. 4. Speedup from the SIMD-FP implementation for 3D transformation and lighting.

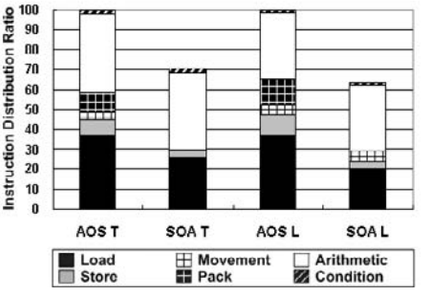

Fig. 5. Instruction distribution in AOS and SOA implementations.

codes. The program codes are compiled by Microsoft Visual C++ 6.0 with Processor Pack 5. The 3D model used in this study is shown in Figure 3. We use mesh re-sampling technique to change the number of vertices in the model. To evaluate the SSE we use two performance profiling tools:

1. **TrueTime** [6]: TrueTime is a performance profiler developed by NuMega. It automatically pinpoints slow codes and accurately reports application and component performance. We use TrueTime to obtain the execution time of different geometry pipeline implementations.
2. **VTune Performance Analyzer** [5]: VTune is a system performance profiling tool created by Intel. This tool is able to monitor several important events, such as mis-predicted branches, cache misses, etc. We use VTune to evaluate the memory system performance.

4 Analysis of Results

In this section, we analyze the performance impact of the SSE instructions on 3D geometry processing. In order to get more insight into the performance increase, we first present the results for transformation and lighting kernels, respectively. The effect of using the SSE on complete geometry pipeline is presented last. We measure the speedup from using only SIMD-FP instructions, evaluate the benefit from using the SOA structure, and finally use prefetch instructions to improve the memory system performance.

Effect of SIMD-FP

The SIMD-FP implementation could achieve the speedup of 4x potentially since we can process four vertices simultaneously. The experimental result shows about 2x and 3x speedup for transformation and lighting as shown in Figure 4. Note that we obtain the speedup using the following formula:

$$\frac{\text{Execution-Time(without SIMD-FP)}}{\text{Execution-Time(with SIMD-FP)}}$$

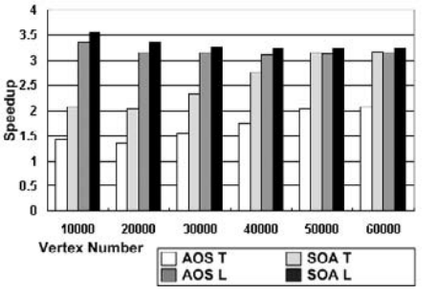

Fig. 6. Speedup from the SOA implementation.

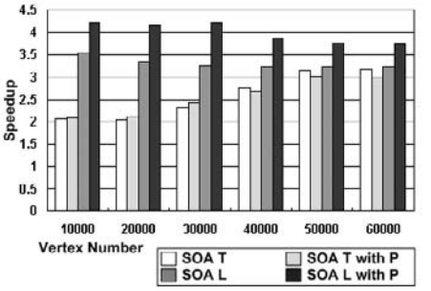

Fig. 7. Speedup from the SOA implementation with prefetching instructions.

The data packing/unpacking overhead undermines the effect of using the SSE. Lighting has higher speedup than transformation because lighting performs more computation on a vertex as described in Section 2. This implies that data manipulation overhead is less significant in lighting compared to transformation. Next, we evaluate how much of the overhead can be eliminated using the SOA structure.

AOS vs. SOA

Figure 5 shows the instruction distribution of transformation and lighting in two different data layouts - AOS vs. SOA. The number of instructions is normalized to the AOS implementation. The results show that SOA reduce 30% of instructions for transformation and 37% for lighting. Note that data packing/unpacking instructions are completely eliminated and the number of load instructions is also reduced significantly. Figure 6 shows the speedup of the kernels using SIMD-FP SOA implementation. The AOS speedup is included for comparison. For transformation, using SOA can achieve the speedup of 3x while AOS can only achieve the speedup of 2x for the largest model. However, SOA shows little performance benefit for lighting even though it reduces 37% of instructions. The computation on lighting requires long latency operations, such as square root and division. Therefore, the number of instructions is not a good indication on the execution time. As mentioned before, the data manipulation overhead is less significant in lighting kernel compared to transformation. So we see less performance gained from using SOA for lighting.

Effect of Prefetching

In this section, we examine the prefetching effect. Because of the streaming access pattern, the vertex data is only prefetched into the L2 cache to avoid the L1 cache pollution. We only prefetch one vertex ahead since it is enough to hide memory latency (in the VTune cache profiling statistics, all the L2 cache misses are eliminated). Figure 7 shows the speed up from prefetching. The results show that prefetching

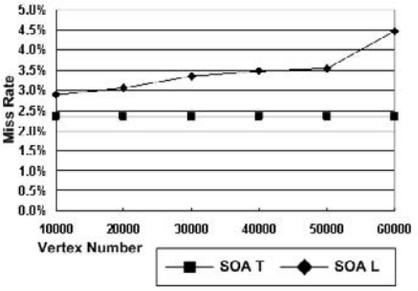

Fig. 8. The L2 cache miss rate of SOA kernels.

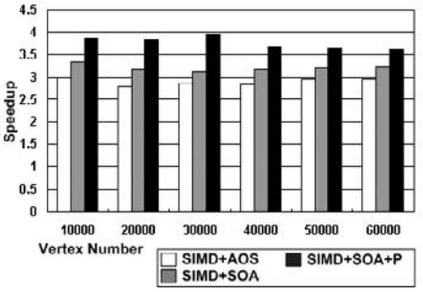

Fig. 9. Speedup of the complete geometry pipeline implementation (AOS, SOA, SOA with prefetching).

achieve significant performance improvement for lighting but little for transformation. From the VTune statistics, we find that lighting has higher L2 miss rate than transformation as shown in Figure 8. It indicates that lighting has more memory stall time than transformation, thus prefetching is more effective for lighting. Note that prefetching could incur overhead, such as wasting memory bandwidth and issuing more instructions. That is why prefetching shows negative performance impact for transformation in some testing cases.

Overall Effect on the Complete Geometry Pipeline

The effect of the SSE on the complete geometry pipeline is shown in Figure 9. We assume three light sources in the scene, which is a common setup in 3D applications. The results show that the speedup from using SIMD-FP with the conventional AOS data structure ranges from 2.7x to 3x (the first bar). Using the SOA structure can further improve the performance (3.1x to 3.3x, the second bar). Prefetching shows significant performance improvement for all testing cases. The overall speedup ranges from 3.6x to 3.9x (the third bar).

5 Related Work

Most of papers studying instruction-set extensions for multimedia applications focused on design issues and illustrations of their use instead of performance analysis [7] [8] [10] [11]. Only the speedup of a small code segment (i.e. application kernels) is reported.

Several papers stud the performance aspect of using multimedia instruction extensions. Bharghava et al. [9] evaluated the MMX technology on Pentium-based systems. Daniel Rice [13] and Ranganathan et al. [12] studied the performance of Sun VIS media extensions [8] for image and video workloads. Yang et al. [15] studied the performance impact of using SIMD instructions on 3D geometry processing similar to this work. But their studies were based on simulation and assume a perfect memory system.

6 Conclusion

We evaluate the effectiveness of using the Intel SSE extensions on the 3D geometry pipeline. We observe that:

1. The SSE provides significant speedup for geometry pipeline. The speedup ranges from 3.0x to 3.8x.
2. The layout of vertices in memory is crucial for the effectiveness of SIMD-FP. Using SOA (structure of arrays) can eliminate the overhead of organizing data into SIMD format.
3. Prefetching shows significant performance improvement for lighting. However, for transformation, it shows little performance benefit. Sometimes, the prefetching overhead even outweighs the benefit.

References

1. Intel Pentium 4 and Intel Xeon processor optimization reference manual. *Intel Corporation, order number: 248966-04*
2. Streaming SIMD Extensions -3D transform. *Intel Corporation, order number: 243631-004,* 1999.
3. The IA-32 Intel architecture software developer's manual. *Intel Corporation, order number: 245471,* 1:231–246, 2001.
4. The IA-32 Intel architecture software developer's manual. *Intel Corporation, order number: 245471,* 1: 247–268, 2001.
5. Intel VTune performance analyzer. *Intel Corporation* http://developer.Intel.com/software/products/vtune/vtune60/index.htm.
6. Numega TrueTime,devpartner for visual C++. *Compuware Corporation,* http://www.compuware.com/products/devpartner/visualc/truetimevc.htm.
7. M.P. et al. Altivec technology: Accelerating media processing across the spectrum. *HotChips10*, 1998.
8. M.T. et al. VIS speeds new media processing. *IEEE Micro,* 16(4):10–20, 1996.
9. R.B. et al. Evaluating MMX technology using DSP and multimedia applications. *ACM/IEEE International Symposium on Microarchitecture,* 1998.
10. A. Peleg and U. Weiser. MMX technology extension to the Intel architecture. *IEEE Mirco,* 16(4):42–50, 1996.
11. S.K.Raman,V.Pentkovski,and J.Keshava.Implementing Streaming SIMD Extensions on the Pentium III processor. *IEEE Micro,* 20(4):47–57,2000.
12. P. Ranganathan, S. Adve, and N.P. Jouppi. Performance of image and video processing with general-purpose processors and media ISA extensions. *International Symposium on Computer Architecture,* 1999.
13. D.S. Rice. High-performance image processing using special-purpose CPU instruction set. *Master's thesis, Stanford University,* 1996.
14. J.D. Foley, A.V. Dam, and S.K. Feiner. *Introduction to Computer Graphics Addison Wesley,* 1993.
15. C.-L.Yang, B. Sano, and A.R. Lebeck. Exploiting instruction level parallelism in geometry processing for three dimensional graphics applications. *ACM/IEEE International Symposium on Microarchitecture,* 1998.