Comparison of Graphics Processors and General Purpose Microprocessors for Texture Mapping

Ian Romanick *

August 20, 2009

1 Introduction

This project aims to compare implementations of texture mapping algorithms on general purpose microprocessors and graphics processing units (GPUs). There is a notion among graphics hardware professionals that GPUs attain higher texture mapping performance than many-core general purpose microprocessors due to the existence of specialized texture access hardware found in the GPU. As an example, Intel's Larrabee essentially combines a many-core general purpose microprocessor architecture with GPU-style texture access hardware[9].

We seek to analyze two specific aspects of the performance of texture mapping algorithms on general purpose microprocessors. First, we seek to determine what the performance gap is between texture mapping algorithms implemented on general purpose microprocessors and texture mapping on GPUs with comparable memory architectures. Second, we seek to determine what factors limit texture mapping performance on current general purpose microprocessors.

The remainder of this paper is divided in to five sections. Section 2 provides some texture mapping background. Section 3 describes how a set of representative benchmarks will be generated for the purpose of this comparison. Section 4 describes the attributes of GPUs and general purpose microprocessors that will be used in the comparison. Section 5 proposes a subset of common texture mapping operations that will be implemented in software on a many-core general purpose microprocessor. Finally, section 6 describes the analysis that will be performed on the collected data.

2 Texture Mapping Background

Texture mapping is, in the most general terms, the application of an image onto a three-dimensional surface. This approach was originally developed by Edwin Catmull in the early 70's [5]. By the mid-90's texture mapping was considered by many to be one of the fundamental drawing primitives [7]. Today, it is difficult to find an paper published on the topic of real-time 3D graphics that does not involve texture mapping in some way.

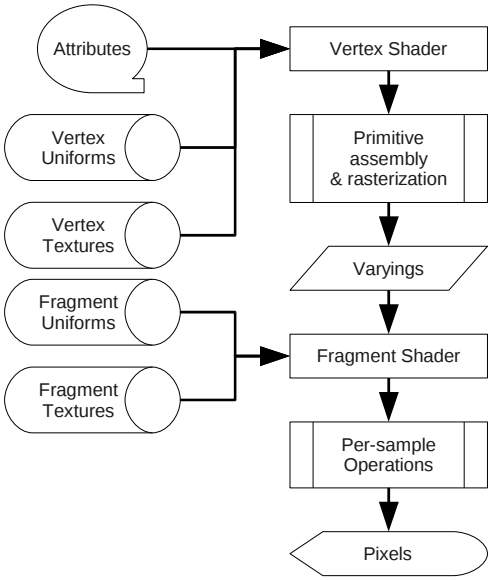

Figure 1: Graphics pipeline

Polygons in 3D are rendered by a variety of algorithms. Applications specify values for several parameters at each vertex of each polygon. The rendering algorithm calculates the value of each parameter at the 3D position corresponding to each display pixel covered by the polygon. These parameters are then supplied as inputs to a program, which typically runs on the GPU, called a fragment shader[1]. The fragment shader can perform a variety of calculations and access texture maps. Figure 1 shows a portion of a typical graphics pipeline.

One common use is to specify coordinates for specific positions in the texture map as per-vertex parameters. These parameters are then used directly in the fragment shader to read values from the texture map. This allows images to be wrapped around a 3D object in much the same way that wrapping paper is wrapped around a present. These types of texture accesses are called direct reads.

*e-mail: idr@cs.pdx.edu

[1] *Fragment shader* is the terminology used in OpenGL. Direct3D calls these programs *pixel shaders*.

Another common use is for applications to sample one texture, perform some calculations on the value read, and use the new value as the location to sample another texture. The second texture access in this scenario is called a dependent read. This can be used, for example, to calculate reflections from bumpy surfaces. The first texture specifies parameters for the bumpiness of the surface, and the second texture contains an image of the environment[8].

In order to limit aliasing artifacts resulting from undersampling the texture image, GPUs implement mipmapping[11]. In mipmapping, successively smaller prefiltered copies of the original texture are stored. Texture coordinates at a particular pixel are compared to the coordinates used at neighboring pixels to determine the area of the texture that covers each pixel. The prefiltered image that contains texture pixels (texels) covering approximately the same area in the original image is used. Since the calculated coverage area may not match any prefiltered image exactly, the two closest prefiltered images are sampled and filtered. Since the specified texture coordinates may not specify the center of a texel, four neighboring texels from both prefiltered images are sampled and filtered together. As a result, eight texels are read for each texture map access. This method of texture access is often referred to as trilinear filtering[10].

3 Benchmarks

Modern graphics applications access texture data through a variety of means. Accesses from typical applications are a mixture of direct reads and dependent reads.

Several modern applications will be analyzed to determine their access patterns for both direct and dependent reads. The analysis will be performed by modifying Mesa, an open-source implementation of an OpenGL-like 3D rendering API, to emit extra data during rendering. In particular, the shaders and textures used will be written to disc during rendering. New shaders and textures will be created that mimic the access patterns of the originals. One goal of the new shaders will be to remove all non-essential elements. The ultimate purpose is to measure the speed of texture access, not to measure the speed of lighting calculations.

4 Representative Hardware

High-end graphics processor systems include a number of features not found in general purpose microprocessor systems. High-end GPUs are traditionally paired with fast, wide buses connected to exotic memories. For example, the Geforce GTX 285 from Nvidia utilizes 8 independent 64-bit data buses connected to 2.4GHz (effective) GDDR3 memory. This results in a theoretical peak memory bandwidth

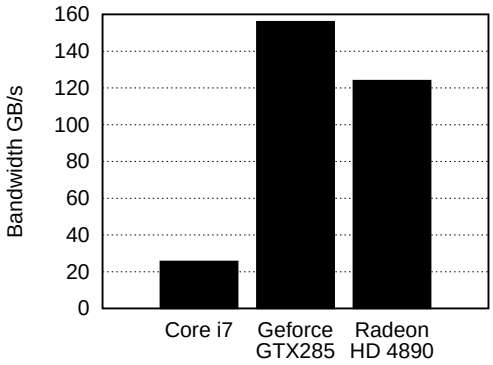

Figure 2: Memory bandwidth comparison

of approximately 156GB/s for the GTX 285[2]. Comparatively, an Intel Core i7 typically utilizes 3 64-bit channels connected to 2.1GHz (effective) DDR3 memory. This results in a theoretical peak memory bandwidth of 25.6GB/s for the Core i7[3]. The exotic, complex memory architecture of the GTX 285 provides it roughly six times the bandwidth of the Core i7. AMD's Radeon HD 4890 has a similar memory bandwidth advantage[1].

In order to make an even comparison between a GPU and a general purpose microprocessor for a memory intensive task such as texture mapping, systems with similar memory architectures must be used. There do not currently exist any general purpose microprocessor systems that utilize 159GB/s memory buses. However, there are GPUs that utilize 25.6GB/s memory buses. At least four vendors ship GPUs integrated into the memory controllers used with general purpose microprocessors. Since these GPUs are integrated into the CPU's memory controller, they necessarily use the same memory architecture as the CPU.

For the purpose of this project, at least one integrated GPU will be selected. The ultimate selection will depend on GPU availability and on the selection of general purpose microprocessors. If time permits and systems are available, additional integrated GPUs may be tested.

5 Texture Operations

Modern GPUs implement a staggering variety of texture operations that applications can utilize. There is variety both in terms of the layout of the textures and the formats of the data. Textures can be stored as 1-dimensional, 2-dimensional, or 3-dimensional arrays of pixels. Textures can also be stored as several varieties of arrays of 2-dimensional arrays of pixels[4][6]. The pixels stored in the textures can be stored using one to four components (nominally red, green, blue, and alpha) using normalized integers, unnormalized integers, and floating-point values in sizes ranging from 8 to 32-bits per component. Textures can also be stored in a variety of compressed formats.

Implementing support for this variety of operations is sig-

nificantly beyond this scope of this project. This project will instead focus on one common variety of 2-dimensional texture. The texture will store four components of 8-bit normalized integers.

6 Analysis

The raw performance, measured in texture accesses per second, will be measured on each of the GPUs and general purpose microprocessor systems. Analysis of the data across the variety of benchmarks will support or refute the notion that custom texture access hardware gives GPUs an advantage over general purpose microprocessors. The analysis will also enable sizing of the performance gap.

During the measurement of the general purpose microprocessor systems, additional data will be collected. Detailed instruction timings for the texture access functions and detailed cache hit-rate measurements will be collected. The data will be analyzed, and comparisons will be made between the instruction timings and the cache accesses. Based on these comparisons, the factors limiting general purpose microprocessor for texture mapping operations will be identified.

References

[1] Comparison of ATI graphics processing units. Internet, `http://en.wikipedia.org/wiki/Comparison_of_ATI_graphics_processing_units`. Accessed on August 19th, 2009.

[2] Geforce GTX 285. Internet, `http://www.nvidia.com/object/product_geforce_gtx_285_us.html`. Accessed on July 31st, 2009.

[3] Intel Core i7-920 Processor (8M Cache, 2.66 GHz, 4.80 GT/s Intel QPI). Internet, `http://ark.intel.com/Product.aspx?id=37147`. Accessed on July 31st, 2009.

[4] OpenGL Cube Map Texturing. Internet, `http://developer.nvidia.com/object/cube_map_ogl_tutorial.html`, 1999. Accessed on August 2nd, 2009.

[5] Edwin Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, University of Utah, 1974.

[6] Bryan Dudash. Texture Arrays Terrain Rendering. Internet, `http://developer.download.nvidia.com/whitepapers/2007/SDK10/TextureArrayTerrain.pdf`, February 2007. Accessed on August 2nd, 2009.

[7] Paul Haeberli and Mark Segal. Texture mapping as a fundamental drawing primitive. In *Proc. Fourth Eurographics Workshop on Rendering*, pages 259 – 266, June, 1993.

[8] Kim Pallister. "Ups and Downs" of Bump Mapping with DirectX 6. Internet, `http://www.gamasutra.com/features/19990604/bump_01.htm`, June 1999. Accessed on July 31st, 2009.

[9] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. Larrabee: a many-core x86 architecture for visual computing. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–15, New York, NY, USA, 2008. ACM.

[10] Wikipedia. Trilinear filtering. Internet, `http://en.wikipedia.org/wiki/Trilinear_filtering`. Accessed on July 31st, 2009.

[11] Lance Williams. Pyramidal parametrics. *SIGGRAPH Computer Graphics*, 17(3):1–11, 1983.