Programmable Graphics—
The Future of Interactive Rendering

Matt Pharr, Aaron Lefohn, Craig Kolb, Paul Lalonde, Tim Foley, and Geoff Berry

Neoptica Technical Report, March 2007

Neoptica 130 Battery Street, Suite 500, San Francisco CA 94111 **T** 415-513-5175 **www.neoptica.com**

Overview

Recent innovations in computer hardware architecture—the arrival of multi-core CPUs, the generalization of graphics processing units (GPUs), and the imminent increase in bandwidth available between CPU and GPU cores—make a new era of interactive graphics possible. As a result of these changes, game consoles, PCs and laptops will have the potential to provide unprecedented levels of visual richness, realism, and immersiveness, making interactive graphics a compelling killer app for these modern computer systems. However, current graphics programming models and APIs, which were conceived of and developed for the previous generation of GPU-only rendering pipelines, severely hamper the type and quality of imagery that can be produced on these systems. Fulfilling the promise of *programmable graphics*—the new era of cooperatively using the CPU, GPU, and complex, dynamic data structures to efficiently synthesize images—requires new programming models, tools, and rendering systems that are designed to take full advantage of these new parallel heterogeneous architectures.

Neoptica is developing the next-generation interactive graphics programming model for these architectures, as well as new graphics techniques, algorithms, and rendering engines that showcase the unprecedented visual quality that they make possible.

Introduction

Computer system architecture is amidst a revolution. The single-processor computer is being supplanted by parallel heterogeneous systems comprised of processors supporting multiple styles of computation. CPU architects are no longer able to improve computational performance of the traditional heart of the computer system, the CPU, by increasing the clock speed of a single processor; instead, they are now providing a rapidly-increasing number of parallel coarse-grained cores, currently capable of delivering approximately 90 GFLOPS. Simultaneously, graphics processing units have evolved to be efficient fine-grained data-parallel coprocessors that deliver much greater raw floating-point horsepower than today's multi-core CPUs; the latest graphics processors from NVIDIA and AMD provide on the order of 400 GFLOPS of peak performance via hundreds of computational units working in parallel. In addition, although CPUs and GPUs have traditionally been separated by low-bandwidth and high-latency communication pathways (e.g. AGP and PCI-Express), rapidly-improving interconnect technology (e.g. AMD Torrenza and Intel Geneseo) and the promise of integrating CPUs and GPUs on a single chip (e.g. AMD Fusion) allow CPUs and GPUs to share data much more efficiently, thereby enabling graphics applications to intermix computation styles to optimally use the system's computational resources.

The success of these new heterogeneous parallel architectures hinges upon consumer applications taking full advantage of their computational power. In order for this to happen, programmers must be presented with intuitive and efficient parallel programming models for these systems. However, decades of work on parallel programming solutions have shown that low-level primitives such as mutexes, semaphores, threads, and message passing are not amenable to creating reliable, complex software systems. Furthermore, existing higher-level parallel programming abstractions have not proven widely successful; these models typically limit developers to a single type of parallelism (i.e., exclusively data-parallel or exclusively task-

parallel), which unnecessarily constrains developer flexibility and makes poor use of the mixed computational resources in heterogeneous systems. Without a higher-level, easy-to-use parallel programming model that allows developers to take full advantage of the entire system, the new parallel architectures may not deliver compelling benefit to users, thus reducing consumer demand for new PCs.

Interactive 3-D computer graphics is now the most computationally demanding consumer application. The economic force of the computer gaming industry and its appetite for computational power have driven the rapid development of current GPUs. In addition, the GPU programming model represents perhaps the only widely-adopted parallel programming model to date. Unfortunately, this model assumes a GPU-only, unidirectional, fixed graphics pipeline. Creating a new programming model for interactive graphics that fully exposes the computational and communication abilities of these new architectures is necessary to enable a revolution in the quality and efficiency of interactive graphics and to provide a killer app for these new platforms.

Neoptica is developing the next-generation interactive graphics programming model for heterogeneous parallel architectures, as well as a broad suite of new graphics techniques, algorithms, and renderers that showcase the unprecedented visual quality possible with these systems. Neoptica's solution makes possible the new era of *programmable graphics*: parallel CPU and GPU tasks cooperatively executing graphics algorithms while sharing complex, dynamic data structures. With Neoptica's technology, graphics programmers are able to:

- treat all processors in the system as first-class participants in graphics computation;
- easily express concurrent computations that are deadlock-free, composable, and intuitive to debug;
- design custom graphics software pipelines, rather than being limited to the single pipeline exposed by current GPUs and graphics APIs; and
- design rendering algorithms that use dynamic, complex user-defined data structures for sparse and adaptive computations.

By enabling graphics programmers to fully leverage these new architectures and freeing them from the constraints of the predefined, one-way graphics pipeline, Neoptica's system spurs the next generation of graphics algorithm innovation, with potential impact far greater than that of the programmable shading revolution of the last five years.

Trends in Interactive Graphics

The last five years have seen significant innovation in interactive graphics software and hardware. GPUs have progressed from being configurable fixed-function processors to highly-programmable data-parallel coprocessors, while CPUs have evolved from single-core to task-parallel multi-core processors. These changes have brought about three stages of interactive graphics programming:

- *Fixed function*: the GPU was configurable, but not programmable. Certain specialized states could be set to achieve simple visual effects (e.g. bump mapping) using multi-pass rendering techniques. The life-span of this stage was short due to its lack of flexibility and relatively high bandwidth demands.
- *Programmable shading*: the vertex and fragment processing stages of the GPU rendering pipeline could be programmed using small data-parallel programs called *shaders*. Using shaders, procedural techniques such as vertex skinning, complex texturing techniques, and advanced lighting models could be implemented efficiently on the GPU. This approach spurred a great deal of graphics algorithm innovation. However, existing graphics APIs and programming models limit developers to a fixed rendering pipeline and a small set of predefined data structures. Implementing custom rendering techniques that exploited more complex data structures was possible only with heroic developer effort, greatly increased code complexity, and high development costs.
- *Programmable graphics*: today, developers are on the threshold of being able to define custom interactive graphics pipelines, using a heterogeneous mix of task- and data-parallel computations to define the renderer. This approach enables complex data structures and adaptive algorithms for techniques such as dynamic ambient occlusion, displacement mapping, complex volumetric effects, and real-time global illumination that were previously only possible in offline rendering. However, current graphics programming models and tools are preventing the widespread transition to this era.

The promise of programmable graphics illustrates the fact that GPU programmability has implications for computer graphics far beyond simple programmable shaders. User-defined data structures and algorithms bring tremendous flexibility, efficiency, and image quality improvements to interactive rendering. Indeed, programmable graphics can be seen as completing the circle of GPGPU (general purpose computing on GPUs). Much of the recent innovation in using data structures and algorithms on GPUs has been driven by the application of GPUs to computational problems outside of graphics. In the era of programmable graphics, techniques developed for GPGPU are applied to the computational problems of advanced interactive graphics. By giving graphics programmers the ability to define their own rendering pipelines with custom data structures, programmable graphics brings far greater flexibility to interactive graphics programmers than is afforded even to users of today's offline rendering systems.

A renderer's ability to efficiently build and use dynamic, complex data structures relies on a mix of task- and data-parallel computation. The GPU's data-parallel computation model, where the same operation is performed on a large number of data elements using many hardware-managed threads, is ideal for *using* data structures and for generating large amounts of new data. In contrast, the task-parallel compute model used by CPU-like processors provides an ideal environment in which to *build* data structures, perform global data analysis, and perform other more irregular computations. While it is possible to use only one processor type for all rendering computations, heterogeneous renderers that leverage the strengths of each use available hardware resources much more efficiently, and make interactive many techniques that would otherwise be limited to use in offline rendering alone.

The transition to programmable graphics is hampered by current graphics programming models and tools. Seemingly simple operations such as sharing data between the CPU and GPU, building pointer-based data structures on one processor for use on the other, and using complex application data in graphics computation currently require esoteric expertise. The specialized knowledge required severely limits the

number of developers who are able to creatively explore the capabilities of hardware systems, which has historically been the key driver of advancing the state of the art in interactive graphics.

The New Era Of Programmable Graphics

Neoptica has built a new system that moves beyond current GPU-only graphics APIs like OpenGL and Direct3D and presents a new programming model designed for programmable graphics. The system enables graphics programmers to build their own heterogeneous rendering pipelines and algorithms, making efficient use of all CPU and GPU computational resources for interactive rendering.

With Neoptica's technology and a mixture of heterogeneous processing styles available for graphics, software developers have the opportunity to reinvent interactive graphics. Many rendering algorithms are currently intractable for interactive rendering with GPUs alone because they require sophisticated per-frame analysis and dynamic data structures. The advent of programmable graphics makes many of these approaches possible in real-time. New opportunities from the era of programmable graphics include:

- Feedback loops between GPU and CPU cores: with the ability to perform many round-trip per-pixel communications per frame, users can implement per-frame, global scene analyses that guide adaptive geometry, shading, and lighting calculations to substantially reduce unnecessary GPU computation.
- Complex user-defined data structures that are built and used during rendering: these data structures enable demand-driven adaptive algorithms that deliver higher-quality images more efficiently than today's brute-force, one-way graphics pipeline.
- Custom, heterogeneous rendering pipelines that span all processor resources: for example, neither a pure ray-tracing approach nor a pure rasterization approach is the most efficient way to render complex visual effects like shadows, reflections, and global lighting effects; heterogeneous systems and programmable graphics will make it possible to easily select the most appropriate algorithm for various parts of the graphics rendering computation.

During the past year, Neoptica has built a suite of high-level programming tools that enable programmable graphics by making it easy for developers to write applications that perform sophisticated graphics computation across multiple CPUs and GPUs, while insulating them from the difficult problems of parallel programming. The system:

- uses a C-derived language for coordinating rendering tasks while using languages such as Cg and HLSL for GPU programming and C and C++ for CPU programming, integrating seamlessly with existing development practices and environments and providing for easy adoption;
- treats all processors in the system as first-class participants in graphics computation and enables users to easily share data structures between processors;
- presents a deadlock-free, composable parallel programming abstraction that embraces both data-parallel and task-parallel workloads;

- provides intuitive, source-level debugging and integrated performance measurement tools.

This system has enabled in the rapid development of new programmable graphics rendering algorithms and pipelines. Developers are able to design custom rendering algorithms and systems that deliver imagery that is impossible using the traditional hardware rendering pipeline, and deliver 10x to 50x speedups of existing GPU-only approaches.

Summary

We are at the threshold of a new era of interactive computer graphics. No longer limited to today's brute-force, unidirectional rendering pipeline, developers will soon be able to design adaptive, demand-driven renderers that efficiently and easily leverage all processors in new heterogeneous parallel systems. New rendering algorithms that tightly couple the distinct capabilities of the CPU and the GPU will generate far richer and more realistic imagery, use processor resources more efficiently, and scale to hundreds of both CPU and GPU cores. Neoptica's technology ushers in this new era of interactive graphics and makes it accessible to a large number of developers.