Glift: Generic Data Structures for the GPU*

Aaron E. Lefohn
University of California, Davis

Shubhabrata Sengupta
University of California, Davis

Joe Kniss
University of Utah

Robert Strzodka
Stanford University

John D. Owens
University of California, Davis

Abstract

Glift is an abstraction and generic template library for parallel, random-access data structures on graphics processing units (GPUs). Glift simplifies the description of new and existing GPU data structures, stimulates development of complex GPU algorithms, performs equivalently to hand-coded implementations, and introduces a parallel iteration model for GPU computation.

1 Introduction

Although designed for interactive rendering, modern GPUs are quickly evolving into general-purpose parallel processors with dozens of processors and high-performance, parallel memory systems. Their peak performance and memory bandwidth substantially outstrip their CPU counterparts; however, the primitive GPU programming model greatly limits the ability of GPU programmers to build complex applications that take full advantage of the hardware.

One of the primary difficulties of GPU programming is the lack of established abstractions and libraries with which to author complex applications. While CPU programmers have access to a wealth of libraries of generic, fundamental data structures, such as those provided by the Standard Template Library or Boost, GPU programmers typically write their programs at a low level. The resulting complexity makes it difficult to create complex GPU applications, and GPU code is often a tangled mess of algorithms and data structures with little to no possible code reuse.

2 Glift: An Abstraction for Random-Access GPU Data Structures

Glift is designed to simplify the creation and use of random-access (i.e., indexable) GPU data structures for both graphics and general-purpose (GPGPU) programming. The abstraction combines common design patterns present in many existing GPU data structures with concepts from CPU-based data structure libraries. Unlike these CPU libraries, however, Glift is designed to express graphics and GPGPU data structures that are efficient on graphics hardware. Glift enables such data structures as n-dimensional arrays, stacks, and dynamic multiresolution adaptive 2D and 3D grids.

The design goals of the Glift abstraction include a simple but powerful abstraction of the GPU programming model; factorization of complex programs into separate and orthogonal data structures and algorithms; and a minimal abstraction penalty.

In order to separate GPU data structures into orthogonal, reusable components and simplify the explanation of complex structures, Glift factors GPU data structures into five components: virtual memory, physical memory, address translator, element iterators, and container adaptors (Figure 1).

*http://www.idav.ucdavis.edu/projects/glift/

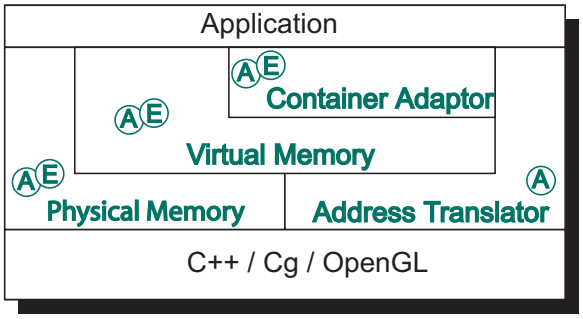

Figure 1: Block diagram of Glift components. Glift factors GPU data structures into a virtual (problem) domain, physical (data) domain, and an address translator that maps between them. Container adaptors are high-level structures that implement their behavior atop an existing structure. Glift structures support CPU and GPU iteration with address iterators (circled A) and element iterators (circled E). Applications can use high-level, complete Glift data structures or use Glift's lower-level `AddrTrans` and `PhysMem` components separately. The Glift library is built on top of C++, Cg, and OpenGL primitives. Figure from Lefohn et al. [2006].

- The `PhysMem` component defines the data storage for the structure. It is a lightweight abstraction around GPU texture memory that supports the 1D, 2D, 3D, and cube-mapped physical memory available on current GPUs as well as mipmapped versions of each.
- The `VirtMem` component defines the programmer's interface to the data structure and is selected based on the algorithm (problem) domain, irrespective of the choice of `PhysMem`.
- A Glift address translator is a mapping between the physical and virtual domains. While conceptually simple, address translators are the core of Glift data structures and define the small amount of code required to virtualize all of GPU memory operations. We also present a taxonomy for address translator characteristics (memory and access complexity, access complexity, etc.) and use them to characterize a wide range of previously presented data structures.
- Glift iterators are the key to allowing computation over a structure's elements, forming a generic interface between algorithms and data structures by abstracting data traversal, access permission, and access patterns. Glift supports both element iterators (whose values are data structure elements) and address iterators (that traverse the virtual or physical addresses of a data structure rather than its elements).
- Finally, Glift container adaptors are higher-level abstractions that implement their behavior on top of an existing container. For instance, a Glift stack is implemented as a container adaptor atop a Glift array.

The Glift implementation is a C++ template library that maps the abstractions described above to a C++/OpenGL/Cg GPU program-

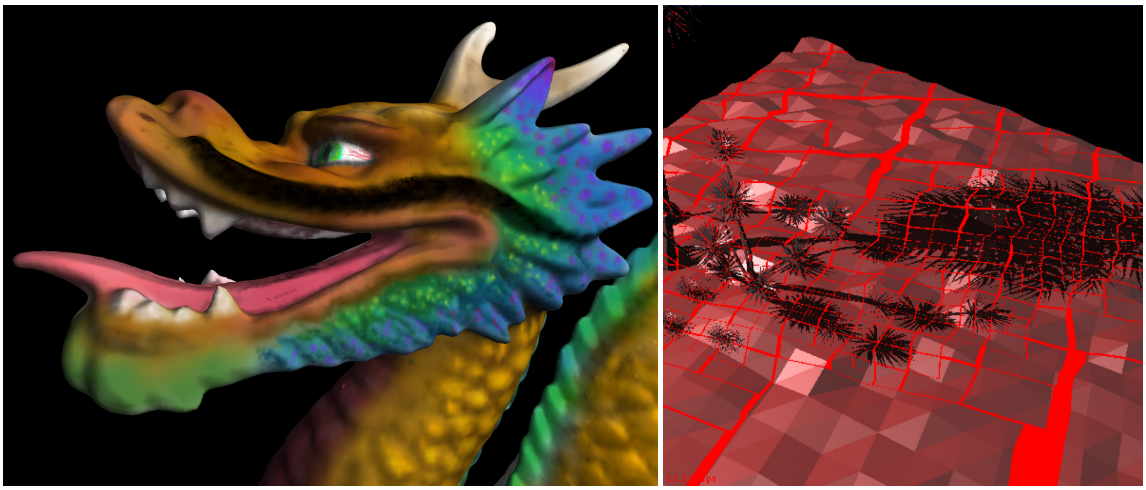

Figure 2: At left, an 817k polygon model with paint stored in an octree with an effective resolution of 2048^3, using 15 MB of GPU memory stored in a Glift 3D adaptive multiresolution data structure that supports quadlinear filtering on the GPU. At right, a visualization of an adaptive shadow map stored in a Glift 2D adaptive multiresolution grid on the GPU.

ming environment. Our implementation goals include incremental adoption, extensibility, efficiency, and CPU/GPU interoperability. Cg is a particularly well-suited target for our implementation because of its support for static polymorphism and program specialization. The result is a library that can easily integrate into existing programming environments or stand alone in new applications.

3 Applications

We used Glift to implement several applications that were previously considered challenging problems on GPUs because of the complexity of their data structures, including the two below. Both were presented in recent Siggraph sketches (Figure 2).

- *Octree 3D Paint* is an interactive 3D paint application that stores paint in a GPU-based, 3D octree-like structure built with the Glift framework [Kniss et al. 2005]. The structure is built using a number of existing Glift components and has an intuitive, texture-like Cg syntax. The implementation supports quadlinear filtering of paint texels and has minimal impact on application performance. We demonstrate interactive painting of an 817k polygon model with effective paint resolutions varying between 64^3 to 2048^3.
- *Adaptive Shadow Maps* address the well-known but difficult problem of interactive rendering of alias-free hard shadows for dynamic scenes [Lefohn et al. 2005]. We use the Glift framework to build a multiresolution, dynamic quadtree of shadow maps, allowing us to implement and improve upon Fernando et al.'s adaptive shadow map algorithm [2001]. Such an application had not been previously demonstrated on GPUs due to their data structure complexity. The required GPU data structures are built using Glift components, exercising nearly every portion of the Glift framework.

4 Conclusion

Our work has three major contributions. First, we characterize a large body of previously published GPU data structures in terms of Glift abstractions, and present novel GPU data structures built with Glift. Second, our example Glift data structures perform comparably to hand-written implementations but require only a fraction of the programming effort. Third, we use the Glift framework to develop novel high-quality interactive rendering algorithms with complex data structure and iteration requirements.

In the same way that efficient implementations of data structure libraries like the Standard Template Library (STL) and Boost have become integral in CPU program development, an efficient GPU data structure abstraction makes it possible for vendors to offer implementations optimized for their architecture and application developers to more easily create complex applications. In addition, Glift's parallel iteration model helps bridge the gap between CPU and GPU programming models and defines a potentially unified approach to expressing computation on disparate commodity parallel architectures.

This work was more completely described in a recent ACM Transactions on Graphics paper by the authors [Lefohn et al. 2006].

References

FERNANDO, R., FERNANDEZ, S., BALA, K., AND GREENBERG, D. P. 2001. Adaptive shadow maps. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 387–390.

KNISS, J., LEFOHN, A., SENGUPTA, S., STRZODKA, R., AND OWENS, J. D. 2005. Octree textures on graphics hardware. In *Technical Sketches Program, ACM SIGGRAPH 2005*.

LEFOHN, A., SENGUPTA, S., KNISS, J., STRZODKA, R., AND OWENS, J. D. 2005. Dynamic adaptive shadow maps on graphics hardware. In *Technical Sketches Program, ACM SIGGRAPH 2005*.

LEFOHN, A. E., KNISS, J., STRZODKA, R., SENGUPTA, S., AND OWENS, J. D. 2006. Glift: Generic, efficient, random-access GPU data structures. *ACM Transactions on Graphics 26*, 1, 60–99.