Crom - Massively Parallel, CPU/GPU Hybrid Computation Platform for Visual Effects

Nathan Cournia
Rhythm & Hues
cournia@rhythm.com

Bradley Smith
Rhythm & Hues
basmith@rhythm.com

Bill Spitzak
Rhythm & Hues
spitzak@rhythm.com

Casey Vanover
Rhythm & Hues
cvanover@rhythm.com

Hans Rijpkema
Rhythm & Hues
hans@rhythm.com

Josh Tomlinson
Rhythm & Hues
josht@rhythm.com

Nathan Litke
Rhythm & Hues
nlitke@rhythm.com

1 Introduction

Crom (Core Rhythm Operating Machine) was designed to provide a flexible, extensible platform on which to develop our next generation of visual effects tools. At its heart were three key design decisions: utilize a strongly-typed dependency graph where the functional pieces are responsible for producing/manipulating a single value type, and the connective tissue handles necessary type conversions; separate functionality and value storage so a single functional piece can operate on disparate tasks without losing the advantage of a persistent cache; and make the application extensible in both functionality and interface through all levels of production development.

2 Dependency Graph

The core of Crom is a dependency graph divided into three primary pieces: *nodes*, *plugs*, and *contexts*. Contexts describe the domain of the computation (frame number, tile number, stereo eye, etc), nodes compute property values, and plugs provide transportation of said values. This division provides for two of our primary conceits: The contexts allow for a stateless design where nodes are passed, then promptly forget, the scope of their computation; and plugs provide a translation mechanism which both simplifies node design and reduces the required number of nodes by eliminating permutations of similar functions.

3 Interface

Like the dependency graph, the user interface is designed for flexibility. Rather than provide customizations, or a selection of layouts, Crom provides a library of panels which the user may dock, resize, and float as he/she deems necessary. This flexibility even extends to the manipulation of individual parameters where users may often toggle between a variety of widgets (or, if they're technically savvy enough, write their own).

The interface is also decoupled from the dependency graph. Computations are performed asynchronously with the values displayed only when ready. This allows the interface to display expensive calculations without the loss of responsiveness. It also has the capacity to provide easy transition to a cloud computing paradigm in situations where that might prove useful.

4 Hybrid GPU/CPU Compositor

Although Crom was not written for a particular application, its first production use is as a compositor. Where possible, nodes leverage on the power of the GPU passing down not image data, but shader functions that can be compiled to process individual pixels. This allows computation to be deferred until actually required, and allows the resulting node tree to be effectively optimized prior to computation. The result is a node graph that effectively performs as a visual computing language.

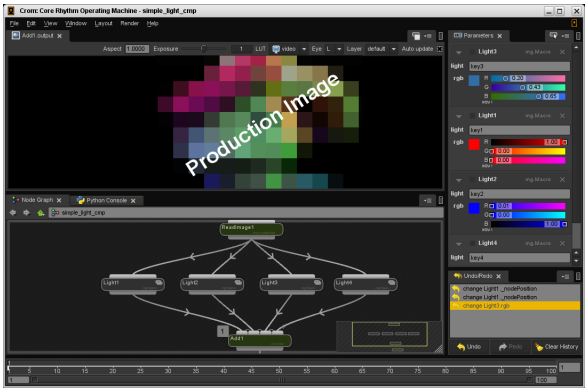

Figure 1: *Using Crom as a compositor*

Where operation on the CPU is necessary (or preferable), plugs handle texture to buffer conversions so data is passed between the two devices seamlessly. Where no GPU is present, the application uses OpenCL to shift all functions to CPU-only processing.

5 Extensibility

Crom is designed to be an open platform. Panels, and node and plug interfaces can all be written in Python as can the nodes themselves. Users can generate *macros* which bundle together a network of nodes into a single tool, with the power to create and customize the interface. Those tools can then be uploaded to our app store, Anvil, and installed on a per-person or per-show basis.

The user-level of development has found particular usage in the compositor. Because Crom optimizes the result of the node graph, there's very little performance difference between a compiled node and a macro providing the same functionality. As a result, we're able to provide a relatively small selection of core nodes, and have users create more specific functionality out of them. This allows the compositor to be easier to maintain and operate in much the same way reduced instruction set computing does.

6 Conclusion

Crom is an application informed by many years of visual effects experience. At its core, it's fast, highly parallelized, and memory efficient. On its surface, it's extremely flexible. It's a tool designed to evolve through usage and leverage on every resource available both in terms of computation and development.

SIGGRAPH 2012, Los Angeles, California, August 5 – 9, 2012.
ISBN 978-1-4503-1435-0/12/0008