\ ACADEMY COLOR ENCODING SYSTEM \

Future of CTL

Academy Science and Technology Council

\ ACESCentral.com \

Agenda

Brief history

Review of CTL components

- CTL language
- CTL modules / programs
- CTL interpreter (front end/ back end)
- Applications

Use of CTL as the ACES reference implementation

Discussion of CTL in production

Introductions and Discussion

ACES

Expected outcomes

The ACES community is familiar with the history and technical details of CTL.

The ACES community understands the primary drivers for the use of CTL as the ACES reference implementation language.

The ACES community has a chance to share their thoughts on the potential future of CTL development and the ACES reference implementation.

Brief history

Created by Florian Kainz and Andrew Kunz in ~2004 at Industrial Light and Magic

Transferred ownership to the Academy in 2006

Used as ACES reference implementation language

Review of CTL Components

CTL Language

- C-like language
- Intended to allow complex color transforms to be shared
- Pixel by pixel operations
- Interpreted source
- Domain specific nature increasing security and enhancing portability
- CTL is **not** ...
  - A general programming langage
  - A color management system
  - Capable of performing operations on multiple pixels (e.g. a convolution kernel)

Review of CTL Components

CTL modules and programs

- Modules are source files containing color transforms
- Modules can be imported by other modules
- Namespacing is supported
- Programs are groups of modules that describe a pipeline

Review of CTL Components

CTL Interpreter

- C++ Interface
- Receives a CTL module and image from an application
- Organized into a front-end and back-end
- Front-end
    - parses the module's source text and generates an abstract syntax tree, reporting syntactic and semantic errors.
    - maintains the interpreter's symbol table and performs constant expression evaluation and dead code elimination.
    - constructed in such a way that multiple different back end implementations can be supported.
- Back-end
    - handles converting the abstract syntax tree into executable code and actually running the code
- CTL source includes a single SIMD back-end allowing CTL programs to be run on any platform that supports C++

Review of CTL Components

Applications

- Any application can leverage the CTL interpreter using the C++ interface
- CTL shipped with a handful of example applications (e.g. exrdpx, exr_ctl_exr)
- Academy funded development of a more general image conversion application interfacing with CTL (ctlrender)

Use of CTL as the ACES reference implementation

CTL was chosen as a portable way to communicate ACES reference transforms

Plays nicely with OpenEXR

Portable nature of CTL provided unambiguous results regardless of platform

Ability to see transform source was critical to clear communication

Interpreted nature allowed for easy prototyping of transforms without recompiling source

Introductions and Discussion

How do you use CTL today?

How do you wish you could use CTL?

ACES

Discussion of CTL in production

CTL with SIMD back-end is slow

Production systems do use CTL. Some directly, some indirectly by converting CTL modules into LUTs.

Some have expressed desire to speed up CTL with alternative back-end

Proof of concept cross-compile to C++, GLSL, etc. (e.g. ctlcc)

CTL vs. CLF ... what's the difference?