OpenColorIO / AMF Python Tutorial

Prepared by: Doug Walker
February 22, 2022

Introduction

This is a tutorial designed to accompany the talk *Color Processing with OpenColorIO and the Academy/ASC Common LUT Format* presented at the 2022 HPA Technical Conference. This brief tutorial will walk you through how to install OpenColorIO and use the provided Python script for converting ACES Metadata Files (AMFs) to a color transform file that may be used natively in OpenColorIO (OCIO). The tutorial then proceeds with a brief demo of how to process a color value through the resulting transform. Finally, some links and suggestions are provided for how to explore further.

The tutorial may be run on MacOS, Windows, and Linux.

Installing the Prerequisites

The first step is to install the Python scripting language. There are two main ways to do this. The first approach is to use whatever package manager you may already be using on your computer. For example, `brew` is popular on MacOS, `vcpkg` on Windows. For Linux, the package manager is specific to which type of distro you use. The second approach is to download and run an installer from the Python website <python.org>. (OCIO is compatible with Python 2.7, but Python 3.x is preferred, and necessary for using PyPI below.)

The next step is to install OpenColorIO. The easiest way to do this is to use the Python Package Installer (PyPI). When you installed Python, it should have installed a command-line tool called `pip`. To ask PyPI to install OpenColorIO, the command is as follows:

```
% pip install opencolorio
```

To test that the installation was successful, in a terminal shell type `python` (or `python3` if you have version 2 as well) to launch Python in interactive mode. Then type `import PyOpenColorIO`. If those steps succeed without error, you have completed the installation successfully. On a Mac, it should look something like this (although your Python version may be slightly different):

```
% python
Python 3.8.9 (default, Oct 26 2021, 07:25:54)
[Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import PyOpenColorIO
>>> ocio.GetVersion()
'2.2.0dev'
```

The '2.2.0dev' version is the current version of the main development branch of OCIO as of the time of this writing. An install from a package manager will likely be 2.1.x (2.2.1 is the current stable release). Any version 2.1 or higher will be fine for this tutorial.

If you have trouble with the installation, please reach out on the OpenColorIO Slack instance in the #helpdesk channel and someone will be glad to help you get it sorted out. Note: at the time of this writing, the pip install process will not work on the new M1-chip Macs unless you run it under Rosetta2 (i.e., via Intel emulation).

Downloading the Python Script

The script described during the talk has been submitted as a Git Pull Request (PR) to be included in a future version of OpenColorIO. If you are familiar with how to use GitHub, you may download the script and its associated files directly from the PR here:
<https://github.com/AcademySoftwareFoundation/OpenColorIO/pull/1604>

Alternatively, you may download a zip file with the script and its associated files from here (please be sure to take the latest version, in case there is more than one by the time you are trying this):
<https://drive.google.com/drive/folders/1ZRTIYE6kFb-7sTUtYIYFNJndhY6tmJee?usp=sharing>

The Python script is named `pyocioamf.py` and it requires an OCIO config file: `config-aces-reference.yaml`. The config must be located in the same directory as the Python script.

There are several additional files: `example.amf` and `example_referenced_lut.clf`. The first is the example AMF file that was shown in the presentation and the second file is an external LUT (in the Academy/ASC Common LUT Format) that is referenced by the AMF file. Finally there is a `README.md` file.

Running the Python Script

To run the script, open a terminal/shell and change your directory to where the tutorial script is located. If you list the contents of the directory, you should see the following files:

```
%  ls
README.md                             example.amf                  pyocioamf.py
config-aces-reference.yaml            example_referenced_lut.clf
```

To run the script type `python pyocioamf.py example.amf`. This will produce the following output to the shell:

```
% python pyocioamf.py example.amf

Processing file:  example.amf

Adding Input Transform from Input - ARRI - LogC EI800 AWG to ACES2065-1
Adding Look Transform: LMT - ACES 1.3 Reference Gamut Compression
  Loading To-CDL-Working-Space Transform from ACES2065-1 to ACES - ACEScct
  Loading From-CDL-Working-Space Transform from ACES - ACEScct to ACES2065-1
Adding To-CDL-Working-Space Transform
Adding CDL Transform
Adding From-CDL-Working-Space Transform
Adding Look Transform file: example_referenced_lut.clf
Adding Output Transform from ACES2065-1 to display: Display - Rec.1886 / Rec.709 Video and view:
Output - SDR Video - ACES 1.0

 <GroupTransform direction=forward, transforms=
        <ColorSpaceTransform direction=forward, src=Input - ARRI - LogC EI800 AWG, dst=ACES - 
ACES2065-1>
        <LookTransform direction=forward, src=ACES - ACES2065-1, dst=ACES - ACES2065-1, looks=LMT - 
ACES 1.3 Reference Gamut Compression>
        <ColorSpaceTransform direction=forward, src=ACES - ACES2065-1, dst=ACES - ACEScct>
        <CDLTransform direction=forward, sop=1.1 1 0.9 -0.01 0.02 0 1 1 1, sat=1.1, style=noClamp>
        <ColorSpaceTransform direction=forward, src=ACES - ACEScct, dst=ACES - ACES2065-1>
        <FileTransform direction=forward, interpolation=best, src=example_referenced_lut.clf>
        <DisplayViewTransform direction=forward, src=ACES - ACES2065-1, display=Display - Rec.1886 / 
Rec.709 Video, view=Output - SDR Video - ACES 1.0, >>

Wrote OCIO CTF file: example.ctf
```

In the directory you should now see a new file named example.ctf. This is a self-contained transform file that implements the color processing specified in the AMF file. It has no external dependencies. For example, it does not rely on the OCIO config file or the example_referenced_lut.clf file.

If you open the example.ctf file in a text editor, you will see that the format is very similar to that of the example_referenced_lut.clf file. This is because the CTF format is based on CLF but extends it with some additional processing operators in order to losslessly serialize any OpenColorIO transform.

Evaluating a Color Through the Generated CTF File

Let's now use OpenColorIO to process an RGB value through the CTF file. Begin by launching an interactive Python session in the terminal shell and import the OpenColorIO library.

```
% python
>>> import PyOpenColorIO as ocio
```

Now use a FileTransform to import the CTF file.

```
>>> ft = ocio.FileTransform('example.ctf')
```

Next create an empty OCIO config object and convert the FileTransform into an OCIO Processor object. The Processor is one of the primary types of objects in OCIO. Creating a Processor from the Transform is a necessary step to evaluating it.

```
>>> config = ocio.Config.CreateRaw()
>>> p = config.getProcessor(ft)
```

OpenColorIO will do color processing on either the CPU or GPU. For simplicity, this tutorial only covers the CPU case. So next, create a CPUProcessor from the Processor object. Then evaluate a single RGB value and print the result.

```
>>> cpu = p.getDefaultCPUProcessor()
>>> cpu.applyRGB([0.35, 0.4, 0.5])
[0.29434067010879517, 0.3787069320678711, 0.5780278444290161]
```

If you want to evaluate a large number of colors at once in Python, the best approach is to install the package Numeric Python (aka "NumPy"). The applyRGB command will accept an entire array of NumPy values to process at once. However, that is beyond the scope of this introductory tutorial.

In the HPA presentation, you saw how the example AMF file included an ACES Input Transform, three Look Transforms, and an ACES Output Transform. So it is actually quite a complicated Transform with many component parts called "ops". The presentation showed a graphical layout of these ops but let's take a look at them in Python.

You may introspect the contents of a Processor by converting it into a GroupTransform and printing it.

```
>>> gt = p.createGroupTransform()
>>> print(gt)
<GroupTransform direction=forward, transforms=
        <LogCameraTransform direction=inverse, base=10, logSideSlope=0.24719 0.24719 0.24719,
logSideOffset=0.385537 0.385537 0.385537, linSideSlope=5.55556 5.55556 5.55556,
linSideOffset=0.0522723 0.0522723 0.0522723, linSideBreak=0.010591 0.010591 0.010591>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f,
matrix=0.680205505106279 0.236136601606481 0.0836578932872397 0 0.0854149797421403 1.01747087860704
-0.102885858349182 0 0.00205652166929683 -0.06256250038479209 1.0605059787155 0 0 0 0 1, offset=0 0 0
0>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f, matrix=1.45143931614567
-0.23651074689374 -0.214928569251925 0 -0.0765537733960204 1.17622969983357 -0.0996759264375522 0
0.008316148425697721 -0.00603244979102103 0.997716301365323 0 0 0 0 1, offset=0 0 0 0>
        <FixedFunction direction=forward, style=ACES_GamutComp13, params=1.147 1.264 1.312
0.8149999999999999 0.803 0.88 1.2>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f,
matrix=0.695452241357452 0.140678696470294 0.163869062172254 0 0.0447945633720376 0.859671118456422
0.0955343181715404 0 -0.00552588255811354 0.00402521030597866 1.00150067225213 0 0 0 0 1, offset=0 0 0
0>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f, matrix=1.45143931614567
-0.23651074689374 -0.214928569251925 0 -0.0765537733960206 1.17622969983357 -0.0996759264375522 0
0.008316148425697721 -0.00603244979102103 0.997716301365323 0 0 0 0 1, offset=0 0 0 0>
        <LogCameraTransform direction=forward, base=2, logSideSlope=0.0570776255707763
0.0570776255707763 0.0570776255707763, logSideOffset=0.554794520547945 0.554794520547945
0.554794520547945, linSideSlope=1 1 1, linSideOffset=0 0 0, linSideBreak=0.0078125 0.0078125
0.0078125>
        <CDLTransform direction=forward, sop=1.1 1 0.9 -0.01 0.02 0 1 1 1, sat=1.1, style=noClamp>
        <LogCameraTransform direction=inverse, base=2, logSideSlope=0.0570776255707763
0.0570776255707763 0.0570776255707763, logSideOffset=0.554794520547945 0.554794520547945
0.554794520547945, linSideSlope=1 1 1, linSideOffset=0 0 0, linSideBreak=0.0078125 0.0078125
0.0078125>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f,
matrix=0.695452241357452 0.140678696470294 0.163869062172254 0 0.0447945633720377 0.859671118456422
0.0955343181715404 0 -0.00552588255811354 0.00402521030597866 1.00150067225213 0 0 0 0 1, offset=0 0 0
0>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f, matrix=1.45143931614567
-0.23651074689374 -0.214928569251925 0 -0.0765537733960206 1.17622969983357 -0.0996759264375522 0
0.008316148425697721 -0.00603244979102103 0.997716301365323 0 0 0 0 1, offset=0 0 0 0>
        <LogCameraTransform direction=forward, base=2, logSideSlope=0.0570776255707763
0.0570776255707763 0.0570776255707763, logSideOffset=0.554794520547945 0.554794520547945
0.554794520547945, linSideSlope=1 1 1, linSideOffset=0 0 0, linSideBreak=0.0078125 0.0078125
0.0078125>
        <Lut3DTransform direction=forward, fileoutdepth=32f, interpolation=tetrahedral, gridSize=3,
minrgb=[0 0 0], maxrgb=[1 1 1]>
        <LogCameraTransform direction=inverse, base=2, logSideSlope=0.0570776255707763
0.0570776255707763 0.0570776255707763, logSideOffset=0.554794520547945 0.554794520547945
0.554794520547945, linSideSlope=1 1 1, linSideOffset=0 0 0, linSideBreak=0.0078125 0.0078125
0.0078125>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f,
matrix=0.695452241357452 0.140678696470294 0.163869062172254 0 0.0447945633720377 0.859671118456422
```

```
0.0955343181715404 0 -0.00552588255811354 0.00402521030597866 1.00150067225213 0 0 0 0 1, offset=0 0 0
0>
        <FixedFunction direction=forward, style=ACES_Glow10>
        <FixedFunction direction=forward, style=ACES_RedMod10>
        <RangeTransform direction=forward, fileindepth=32f, fileoutdepth=32f, minInValue=0,
minOutValue=0>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f, matrix=1.45143931614567
-0.23651074689374 -0.214928569251925 0 -0.0765537733960204 1.17622969983357 -0.0996759264375522 0
0.008316148425697721 -0.00603244979102103 0.997716301365323 0 0 0 0 1, offset=0 0 0 0>
        <RangeTransform direction=forward, fileindepth=32f, fileoutdepth=32f, minInValue=0,
minOutValue=0>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f, matrix=0.970889148671
0.026963270632 0.002147580696 0 0.010889148671 0.986963270632 0.002147580696 0 0.010889148671
0.026963270632 0.962147580696 0 0 0 0 1, offset=0 0 0 0>
        <LogTransform direction=forward, base=10>
        <GradingRGBCurveTransform direction=forward, style=log, values=<red=<control_points=[<x=0,
y=0><x=1, y=1>]>, green=<control_points=[<x=0, y=0><x=1, y=1>]>, blue=<control_points=[<x=0, y=0><x=1,
y=1>]>, master=<control_points=[<x=-5.260177612304688, y=-4><x=-3.755027532577515,
y=-3.578688383102417><x=-2.249877452850342, y=-1.821313261985779><x=-0.7447274923324585,
y=0.6812412142753601><x=1.061452507972717, y=2.874577522277832><x=2.867632389068604,
y=3.834062099456787><x=4.673812389373779, y=4>]>>>
        <GradingRGBCurveTransform direction=forward, style=log, values=<red=<control_points=[<x=0,
y=0><x=1, y=1>]>, green=<control_points=[<x=0, y=0><x=1, y=1>]>, blue=<control_points=[<x=0, y=0><x=1,
y=1>]>, master=<control_points=[<x=-2.540623664855957, y=-1.698969960212708><x=-2.080357313156128,
y=-1.588435053825378><x=-1.620090842247009, y=-1.353500008583069><x=-1.159824371337891,
y=-1.046949982643127><x=-0.6995580196380615, y=-0.6564000248908997><x=-0.2392915785312653,
y=-0.2214100062847137><x=0.2209748327732086, y=0.2281440198421478><x=0.6812412142753601,
y=0.6812412142753601><x=1.012846350669861, y=0.9914218783378601><x=1.344451427459717,
y=1.258000016212463><x=1.676056504249573, y=1.449949979782104><x=2.007661581039429,
y=1.559100031852722><x=2.339266538619995, y=1.622599959373474><x=2.670871734619141,
y=1.660654544830322><x=3.002476692199707, y=1.681241273880005>]>>>
        <LogTransform direction=inverse, base=10>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f,
matrix=0.0208420175072947 0 0 0 0 0.0208420175072947 0 0 0 0 0.0208420175072947 0 0 0 0 1,
offset=-0.000416840350145894 -0.000416840350145894 -0.000416840350145894 0>
        <FixedFunction direction=forward, style=ACES_DarkToDim10>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f, matrix=0.949056010175
0.047185723607 0.003758266219 0 0.019056010175 0.977185723607 0.003758266219 0 0.019056010175
0.047185723607 0.933758266219 0 0 0 0 1, offset=0 0 0 0>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f,
matrix=0.652237541886287 0.128236135999712 0.169982249165671 0 0.267672180125336 0.674339988801551
0.0579878310731125 0 -0.00538181576638765 0.00136906020909583 1.09307050631717 0 0 0 0 1, offset=0 0 0
0>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f, matrix=3.24096994190452
-1.53738317757009 -0.498610760293003 0 -0.9692436362808799 1.87596750150772 0.0415550574071756 0
0.0556300796969936 -0.203976958888977 1.05697151424288 0 0 0 0 1, offset=0 0 0 0>
        <ExponentTransform direction=inverse, value=2.4 2.4 2.4 1, style=clamp>>
```

That's a lot of ops! But although it may seem like a lot of steps, most of the individual Transforms (e.g., such as MatrixTransform) are quite computationally inexpensive. However, as discussed in the presentation, OCIO is able to detect redundant conversions and remove them. For example, several of the Look Transforms use ACEScct as the working space and so OCIO is able to convert once into ACEScct, apply the Looks, and then convert out of ACEScct. This is faster than doing the conversion to ACEScct and back around each of the Looks. In addition, OCIO is able to do optimizations such as combining two adjacent matrices into a single matrix.

OCIO is capable of doing many types of these optimizations on transform pipelines and it gives the user fine-grained control to turn various optimizations on or off. This tutorial will just show the result of the default optimization level, but you may consult the documentation for more options. Here is how to create an optimized Processor, convert it to a GroupTransform, and print the result.

```
>>> optgt = p.getOptimizedProcessor(ocio.OPTIMIZATION_DEFAULT).createGroupTransform()
>>> print(optgt)
<GroupTransform direction=forward, transforms=
        <LogCameraTransform direction=inverse, base=10, logSideSlope=0.24719 0.24719 0.24719,
logSideOffset=0.385537 0.385537 0.385537, linSideSlope=5.55556 5.55556 5.55556,
linSideOffset=0.0522723 0.0522723 0.0522723, linSideBreak=0.010591 0.010591 0.010591>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f,
matrix=0.9666334472552328 0.1155416188072188 -0.08217506606244765 0 0.04819035218013039
1.184938293429572 -0.2331286456097078 0 0.007193253557313438 -0.06659372144940504 1.059400467892096 0
0 0 0 1, offset=0 0 0 0>
        <FixedFunction direction=forward, style=ACES_GamutComp13, params=1.147 1.264 1.312
0.8149999999999999 0.803 0.88 1.2>
        <LogCameraTransform direction=forward, base=2, logSideSlope=0.0570776255707763
0.0570776255707763 0.0570776255707763, logSideOffset=0.554794520547945 0.554794520547945
0.554794520547945, linSideSlope=1 1 1, linSideOffset=0 0 0, linSideBreak=0.0078125 0.0078125
0.0078125>
        <MatrixTransform direction=forward, fileindepth=unknown, fileoutdepth=unknown, matrix=1.186614
-0.07152000000000006 -0.006498000000000006 0 -0.02338600000000002 1.02848 -0.006498000000000006 0
-0.02338600000000002 -0.07152000000000006 0.9835020000000001 0 0 0 0 1, offset=-0.0122178 0.0207822
-0.001217800000000001 0>
        <Lut3DTransform direction=forward, fileoutdepth=32f, interpolation=tetrahedral, gridSize=3,
minrgb=[0 0 0], maxrgb=[1 1 1]>
        <LogCameraTransform direction=inverse, base=2, logSideSlope=0.0570776255707763
0.0570776255707763 0.0570776255707763, logSideOffset=0.554794520547945 0.554794520547945
0.554794520547945, linSideSlope=1 1 1, linSideOffset=0 0 0, linSideBreak=0.0078125 0.0078125
0.0078125>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f,
matrix=0.695452241357452 0.140678696470294 0.163869062172254 0 0.0447945633720377 0.859671118456422
0.0955343181715404 0 -0.00552588255811354 0.00402521030597866 1.00150067225213 0 0 0 0 1, offset=0 0 0
0>
        <FixedFunction direction=forward, style=ACES_Glow10>
        <FixedFunction direction=forward, style=ACES_RedMod10>
        <RangeTransform direction=forward, fileindepth=32f, fileoutdepth=32f, minInValue=0,
minOutValue=0>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f, matrix=1.45143931614567
-0.23651074689374 -0.214928569251925 0 -0.0765537733960204 1.17622969983357 -0.0996759264375522 0
0.00831614842569772 -0.00603244979102103 0.997716301365323 0 0 0 0 1, offset=0 0 0 0>
        <RangeTransform direction=forward, fileindepth=32f, fileoutdepth=32f, minInValue=0,
minOutValue=0>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f, matrix=0.970889148671
0.026963270632 0.002147580696 0 0.010889148671 0.986963270632 0.002147580696 0 0.010889148671
0.026963270632 0.962147580696 0 0 0 0 1, offset=0 0 0 0>
        <LogTransform direction=forward, base=10>
        <GradingRGBCurveTransform direction=forward, style=log, values=<red=<control_points=[<x=0,
y=0><x=1, y=1>]>, green=<control_points=[<x=0, y=0><x=1, y=1>]>, blue=<control_points=[<x=0, y=0><x=1,
y=1>]>, master=<control_points=[<x=-5.260177612304688, y=-4><x=-3.755027532577515,
y=-3.578688383102417><x=-2.249877452850342, y=-1.821313261985779><x=-0.7447274923324585,
y=0.6812412142753601><x=1.061452507972717, y=2.874577522277832><x=2.867632389068604,
y=3.834062099456787><x=4.673812389373779, y=4>]>>>
        <GradingRGBCurveTransform direction=forward, style=log, values=<red=<control_points=[<x=0,
y=0><x=1, y=1>]>, green=<control_points=[<x=0, y=0><x=1, y=1>]>, blue=<control_points=[<x=0, y=0><x=1,
y=1>]>, master=<control_points=[<x=-2.540623664855957, y=-1.698969960212708><x=-2.080357313156128,
y=-1.588435053825378><x=-1.620090842247009, y=-1.353500008583069><x=-1.159824371337891,
y=-1.046949982643127><x=-0.6995580196380615, y=-0.6564000248908997><x=-0.2392915785312653,
y=-0.2214100062847137><x=0.2209748327732086, y=0.2281440198421478><x=0.6812412142753601,
y=0.6812412142753601><x=1.012846350669861, y=0.9914218783378601><x=1.344451427459717,
y=1.258000016212463><x=1.676056504249573, y=1.449949979782104><x=2.007661581039429,
y=1.559100031852722><x=2.339266538619995, y=1.622599959373474><x=2.670871734619141,
y=1.660654544830322><x=3.002476692199707, y=1.681241273880005>]>>>
        <LogTransform direction=inverse, base=10>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f,
matrix=0.0208420175072947 0 0 0 0 0.0208420175072947 0 0 0 0 0.0208420175072947 0 0 0 0 1,
offset=-0.000416840350145894 -0.000416840350145894 -0.000416840350145894 0>
        <FixedFunction direction=forward, style=ACES_DarkToDim10>
        <MatrixTransform direction=forward, fileindepth=32f, fileoutdepth=32f,
matrix=1.604753433346921 -0.5310809486040146 -0.07367248474190846 0 -0.1020824581065506
1.108134128622124 -0.006051670514573358 0 -0.003267111653294835 -0.07275542413342301 1.076022535787719
0 0 0 0 1, offset=0 0 0 0>
        <ExponentTransform direction=inverse, value=2.4 2.4 2.4 1, style=clamp>>
```

(Please note that in practice there is almost never a need to call getOptimizedProcessor since OCIO will apply the optimization anyway when building a CPUProcessor or GPUProcessor. For example, the ops you see above are what would be evaluated as a result of using the getDefaultCPUProcessor command further above.)

There is much more that you can do in Python with OpenColorIO. For example, you may build color transforms from scratch and write them out in the Academy/ASC Common LUT Format and other formats. You may also create OCIO config files. Those techniques are beyond the scope of this introductory tutorial, but the Python API is covered in the OCIO documentation website. Please reach out on the OpenColorIO Slack instance if you have questions.

Going Further

Once you have converted an AMF file into an OCIO CTF file, you may use that with any of the OCIO command-line tools to do things such as process images. Using `pip` to install OpenColorIO unfortunately does not install these tools. A few of the tools, such as `ociochecklut` are installed if you use a package manager to install OpenColorIO. For example, this may be done using `brew` on MacOS or `vcpkg` on Windows.

The `ociochecklut` tool is useful for quickly sending single RGB values through a LUT in any of the formats supported by OCIO (similar to what you did above in Python). For example, here is what it looks like to do this in a terminal shell.

```
% ociochecklut example.ctf 0.35 0.4 0.5
0.2943407 0.3787069 0.5780278
```

In addition, the `ociochecklut` tool will report on any syntax errors found. And if you run it without an RGB value, it will print the list of component transform operators, similar to what you did in Python above.

Processing Images

OpenColorIO has a command-line tool named `ocioconvert` that may be used to process image frames through a CTF file on either the CPU or GPU. However, the package managers will not install this tool so you would need to build OpenColorIO from source to get this. Note that you must first install another popular open source package called OpenImageIO first. Often the simplest way to do that is via a package manager. So for example, on MacOS you could use `brew install opencolorio`. And then follow the instructions in the OpenColorIO documentation for how to build from source.

If you don't require `ocioconvert`'s ability to process via the GPU, you could avoid the need to build OCIO from source by using a command-line tool that comes with OpenImageIO called `oiiotool`. For example, after doing an install of OpenImageIO, you should be able to process images through a CTF as follows.

```
% oiiotool  inputFrame0001.exr --ociofiletransform example.ctf -o outputFrame0001.exr
```

Conclusion

Hopefully this has been a useful introduction to OpenColorIO. The project has an active community on its Slack instance, so don't hesitate to join and ask questions.

The provided `pyocioamf.py` script, along with the associated OCIO config file, should allow you to convert most AMF files into CTF files for use with OCIO's own tools, or with applications that include OpenColorIO v2 such as Maya, Nuke, and Unreal engine.

One known limitation is that the effort to define ACES Transform IDs for all cameras is an on-going project and its possible that the supplied OCIO config file does not include an ID that may be used in one of your AMF files. And given that this script is a prototype, there may be bugs. If you find any problems please log an issue on the OpenColorIO GitHub site or reach out to me directly at doug.walker@autodesk.com.

Thanks for reading!

Useful Links

OpenColorIO main website:
<https://opencolorio.org/>

OpenColorIO documentation:
<https://opencolorio.readthedocs.io/en/latest/>

OpenColorIO on GitHub:
<https://github.com/AcademySoftwareFoundation/OpenColorIO/>

OpenColorIO on Slack:
<http://slack.opencolorio.org/>

ACES Central
<https://acescentral.com/>