# OpenEXR Image Viewing Software

Florian Kainz,

Industrial Light & Magic

updated 07/26/2007

This document describes two OpenEXR image viewing software programs, `exrdisplay` and `playexr`. It briefly explains real-time playback of moving image sequences and color rendering, and describes how to test if other image viewing software displays OpenEXR images correctly.

## Introduction

The OpenEXR_Viewers package contains two programs for viewing OpenEXR image files on a computer monitor: `exrdisplay` for still images and `playexr` for moving image sequences.

The `exrdisplay` and `playexr` programs are reference implementations. Other OpenEXR image display software can be checked against `exrdisplay` or `playexr` in order to verify that images are correctly placed on the screen and that colors are reproduced as intended. The source code for `exrdisplay` and `playexr` provides a set of programming examples that demonstrate two ways to implement an OpenEXR image viewer.

While `exrdisplay` and `playexr` do display images and are good enough for experimentation and occasional use, the programs are not intended to be industrial-strength tools. The programs lack features that users would expect from software intended to be employed every day, for example, zooming and panning, image comparison or histogram displays. `exrdisplay` and `playexr` are meant to help developers build production-quality software.

The programs demonstrate two different approaches to building an image viewer:

`exrdisplay` performs all image processing and color rendering in software, without relying on specialized graphics hardware to accelerate any of the operations. The program supports a variety of display options, for example, displaying individual image channels, displaying the lower-resolution levels of mipmaps and ripmaps, or cyclically shifting an image to check if it tiles seamlessly. The quality of the displayed images is close to optimal. However, the program is relatively slow.

`playexr` is designed to play back moving image sequences directly from disk, without preprocessing or in-memory caching of the images. The program requires graphics hardware that supports the Cg shading language. The program is multi-threaded. One thread controls file loading from disk while another thread displays the images on the screen. Additional threads, inside the IlmImf library, can accelerate file loading. In order to be as fast as possible, image processing and color rendering take a few shortcuts: color transforms are baked into a 3D lookup table, and a Cg shader applies the lookup table to the image pixels. This Cg shader also converts luminance/chroma images to RGB. The quality of the displayed images is generally very high, although occasionally there may be small but noticeable differences between what `playexr` and `exrdisplay` show on the screen.

## Real-Time Playback

When image sequences are played back with `playexr`, the maximum frame rate that can be maintained for a given image resolution depends on the available hardware and on how the images are stored. In tests at Industrial Light & Magic (November 2006), we were able to play back a 10,000-frame image sequence with a resolution of 2048 by 872 pixels at a controlled rate of 30 fps (frames per second). When `playexr` was allowed to run as fast as possible instead of trying to maintain a specific constant frame rate, images were read from disk and displayed at a variable rate of about 35 to 45 fps. We used a Hewlett-Packard wx9400 workstation equipped with an Nvidia Quadro FX 5500 graphics card and two 10,000 rpm SATA drives in a RAID0 configuration. The operating system was Linux. The images were B44-compressed and stored in luminance/chroma format. The frame rate appeared to be limited by disk access times and I/O bandwidth. When the image files were already present in the operating system's buffer cache instead of being read directly from disk, `playexr` achieved a controlled rate of 40 fps and a maximum rate of 50 to 55 fps. At that point the frame rate appeared to be limited by the throughput of the graphics card; files in the buffer cache were read at about 90 to 100 fps when the images were simply discarded instead of being displayed.

In order to make real-time playback possible, certain constraints must be placed on the image files: since most of OpenEXR's data compression schemes are too slow for real-time decoding on current hardware, the files must be stored either B44-compressed or uncompressed. Additional image channels that will not be displayed should be avoided. The extra channels require additional I/O bandwidth and decoding speed. High-resolution images should be stored in luminance/chroma rather than RGB format. Since luminance/chroma files are only half as big as RGB files, they require less bandwidth and they are decoded faster.

When uncompressed images are played back, the maximum frame rate probably will be limited by the available disk I/O bandwidth. Playback of high-resolution images requires very fast disks.

The B44 compression scheme was designed specifically to enable real-time image playback with lower I/O bandwidth. Decoding of B44-compressed files is very fast. B44 compression has a fixed compression rate of 2.28:1, which does not depend on image content. If B44 compression is combined with luminance/chroma storage the total compression rate is 4.57:1.

By default, `playexr` loads image files using only a single thread. The `-t` command line flag enables multi-threaded file loading and specifies the number of additional file reading threads. The optimum number of threads depends on the available hardware. If a machine supports very fast file I/O, the number of additional threads should be equal to the number of available processors. If playback speed is limited by I/O bandwidth, a smaller number of threads may yield the same frame rate.

## Color Rendering

Conceptually, most OpenEXR image files are scene referred or focal-plane referred. The values stored in the pixels are proportional to the relative amount of light coming from the corresponding objects in the depicted scene. The pixels in the file do not directly represent the colors that should appear on the screen when the image is displayed. Color rendering converts the pixel values in the file into pixel colors for the screen.

In `exrdisplay` and `playexr` color rendering is performed by applying a series of color transforms. Color transforms are represented as functions written in a Color Transformation Language (CTL). Color rendering as implemented in `exrdisplay` and `playexr` follows the model that is currently (as of November 2006) being developed by the Image Interchange Framework Committee of the Academy of Motion Picture Arts and Sciences.

In this model, an optional look modification transform alters the original pixels to achieve particular look for the image. A rendering transform then converts the pixels into output-referred pixels for display on an ideal device. The rendering transform is followed by a display transform that converts pixels for the ideal device into pixels for an actual display device, such as a video monitor.

Before applying any color transforms, `exrdisplay` and `playexr` must load the corresponding CTL functions. The name of the function that represents the look modification transform is taken from the `lookModTransform` string attribute in the header of the image file. If the file header does not contain a `lookModTransform` attribute, then no look modification transform will be applied to the image. The name of the function that represents the rendering transform is taken from the `renderingTransform` string attribute in the header of the image file. If the file header does not contain such an attribute, a default name, "transform_RRT", is used. (RRT stands for Reference Rendering Transform.) The name of the display transform is taken from the environment variable `CTL_DISPLAY_TRANSFORM`. If this environment variable is not set, a default name, "transform_display_video", is used.

Once the names of the transforms are known, the corresponding CTL source files must be loaded. Each transform is assumed to live in a file with the same name the transform, but with `.ctl` appended to the file name. For example, CTL function `transform_RRT()` lives in file `transform_RRT.ctl`. The actual source files are located via a search path, which is specified by the `CTL_MODULE_PATH` environment variable.

Notes:

- Color rendering as described above requires that `exrdisplay` and `playexr` are linked with a CTL interpreter. The CTL interpreter is not included in OpenEXR. The source code for the interpreter is available at http://ampasctl.sourceforge.net/. Both `exrdisplay` and `playexr` can be built without a CTL interpreter; color rendering will be disabled, but otherwise the programs will work.

- At a Birds-of-a-Feather meeting during the ACM Siggraph 2004 conference, ILM proposed a model for OpenEXR color management that differs significantly from the Academy's model. ILM's proposed color management model is not implemented in the current versions of `exrdisplay` and `playexr`.

## En ironment Variable!

Several environment variables affect how `exrdisplay` and `playexr` reproduce color:

| name | description | default value |
|---|---|---|
| CTL_DISPLAY_TRANSFORM | The name of the CTL display transform. | "transform_display_video" |
| CTL_DISPLAY_CHROMATICITIES | The CIE x,y coordinates of the primaries and white point of the display. | "red 0.6400 0.3300 green 0.3000 0.6000 blue 0.1500 0.0600 white 0.3127 0.3290" (primaries and white point according to Rec. ITU-R BT.709-3) |
| CTL_DISPLAY_WHITE_LUMINANCE | The maximum luminance, in cd/m$^2$, of the display. | 120.0 |
| CTL_SURROUND_LUMINANCE | The luminance, in cd/m$^2$, of the background that surrounds the display. | 0.1 times the maximum display luminance. |
| CTL_MODULE_PATH | The search path that is used to locate CTL modules. The path consists of a colon-separated list of directory names, for example, ".:../ctl:/usr/local/ctl" | "." (the search path contains only the current working directory) |
| EXR_DISPLAY_VIDEO_GAMMA | The video gamma of the display, used by `exrdisplay` and `playexr` to convert the output of the display transform into hardware frame buffer values. | 2.2 |

Only the values of the `CTL_MODULE_PATH` and `EXR_DISPLAY_VIDEO_GAMMA` environment variables are used directly by `exrdisplay` and `playexr`. The values of the other environment variables in the table above are made available to the CTL transforms.


## CT" Input! and Output!

When `exrdisplay` or `playexr` calls the CTL transforms, the `R`, `G` and `B` channels of the image files are made available to the rendering transform as three `varying input` parameters, `R`, `G` and `B`, of type `half`. The display transform must make the `R`, `G` and `B` channels of the final image available as three `varying output` parameters, `R_display`, `G_display` and `B_display`. By convention, the pixels produced by the rendering transform are passed to the display transform as three varying parameters, `X_OCES`, `Y_OCES` and `Z_OCES`, of type `half`.

In addition to the input and output image channels, as well as the attributes in the header of the input image file, both the rendering and the display transform have access to the following data:

| name and type | description |
|---|---|
| `Chromaticities chromaticities` | The CIE x,y coordinates of the primaries and white point of the image file, as specified by the chromaticities attribute in the input file header. If the header does not contain a chromaticities attribute, then the primaries and white point are set according to Rec. ITU-R BT.709-3. |
| `Chromaticities displayChromaticities` | The CIE x,y coordinates of the primaries and white point of the display, as specified by the `CTL_DISPLAY_CHROMATICITIES` environment variable. |
| `float displayWhiteLuminance` | The maximum luminance of the display, as specified by the `CTL_DISPLAY_WHITE_LUMINACE` environment variable. |
| `float displaySurrondLuminance` | The luminance of the display surround, as specified by the `CTL_DISPLAY_SURROUND_LUMINANCE` environment variable. |


## Te! ting Ot#er OpenEXR Viewer!

As mentioned above, `exrdisplay` and `playexr` are reference implementations that software developers can use to test if their own OpenEXR image viewers display images correctly. A directory tree with OpenEXR sample images can be downloaded from [http://www.openexr.com/](http://www.openexr.com/). In general, image viewers should display all the sample images the same way as `exrdisplay` and `playexr`. In particular, developers should pay attention to the following:

Directory `DisplayWindow` contains a series of files, `t01.exr`, `t02.exr` … `t16.exr`, that can be used to test if an image viewer properly places, crops and pads the images. All files contain the same grid of 400 by 300 pixels, but the data window, display window and pixel aspect ratio differ. For file `t01.exr`, where the display window and the data window are the same, the pixels should fill the entire on-screen area where the image is displayed. The image should be cropped for files where the data window extends outside the display window, and padded with pixels of an appropriate background color for files where the display window extends outside the data window. The pixel aspect ratio in files `t15.exr` and `t16.exr` is not 1.0; that is, the pixels are not square. The images must be stretched horizontally or vertically when displayed on a screen with square pixels.

Directory `LuminanceChroma` contains several files that are stored in luminance/chroma format, with a Y channel and sub-sampled RY and BY channels, as well as one file image with only a Y channel. The luminance/chroma images should be displayed in color, and the Y-only image, `Garden.exr`, should be displayed in black and white.

Directory `Chromaticities` contains four versions of the same color image: two where the primaries and white point match Rec. ITU-R BT.709-3, and two with CIE 1931 X, Y and Z primaries and an equal-energy white point (CIE illuminant E). One of the Rec. 709 files and one of the XYZ files are stored in luminance/chroma format. Displayed properly, all four versions of the image should look the same.

Directory `MultiResolution` contains several mip-map and rip-map multi-resolution image files. Unless instructed otherwise by the user, OpenEXR viewers should display the highest-resolution version of the image. Some viewing software may want to use the lower-resolution versions to implement fast zoom and pan operations for images whose highest resolution exceeds the resolution of the display screen.

Files `AllHalfValues.exr`, `WideFloatRange.exr` and `WideColorGamut.exr` in directory `TestImages` contain valid but unusual pixel data. The pixels in those files contain very large numbers, negative numbers, infinities and NaNs. None of the files should cause an OpenEXR viewer to fail. Pixel values that are within the capabilities of the display screen should be handled as in other image files; the rest should be clamped to the display's maximum or minimum luminance.