



In this chapter, we'll take a look at all the tools that come with HDRI. In some cases, I'll need to get very technical, but I'll refer to the practical application wherever possible. The main goal here is to give you an overview of the tools that can be incorporated into an HDRI workflow. Whether you adapt the workflow

proposed later in this book will be a decision only you can make, but this chapter will provide you with the background knowledge you need to make educated choices.

2.1. File Formats

OK, so 8 bits are clearly not enough. You learned in Chapter 1 that HDR images are made of 32-bit floating-point numbers. This is the image data that sits in the computer memory when an HDR image is edited or displayed. We are talking about huge chunks of data here, much bigger and with a different structure than the regular 8-bit image. So we need new image formats that can store that HDR data.

Each image format has its own way of wrapping all this data into an actual file on disk. Important for us are some key elements: How good do they preserve the dynamic range? How widely are they understood? And most important, how much space do they take?

2.1.1. RAW Formats (.raw/.nef/.crw/.orf/.dng/...)

These formats are used as output for digital cameras, and there are as many variants as there are cameras out there. There is no such thing as one standard RAW format; they are all different from each other. Most variants deliver 10, 12, or 14 bits, which qualifies them as prime examples for a medium dynamic range format. It holds more range than in an LDR image, but it's still far from a true HDR image.

The Good and the Bad: Essentially, RAW is the direct digital equivalent of an unprocessed film negative. The data from the image sensor is dumped on the storage card with minimal alteration. Interestingly, most image sensors react to light in a linear fashion, so a RAW file is already set up in a linear color space, just like HDR images.



▲ RAW

Many digital photographers prefer to process this raw image data instead of having the cameras on-board software compress it down to an 8-bit JPEG, just as many traditional photographers prefer to rent out a darkroom and develop the film themselves instead of taking it to the drugstore around the corner. This analogy really holds up because RAW files generally include 1 to 2 EVs more at both ends of the exposure, which simply get clipped by the in-camera JPEG compression. And just like the film negative, a RAW file reveals all the sensor noise that is inherent close to the extremes of light and shadow.

Theoretically, shooting a RAW file would require minimal processing power since it really is just the plain uncompressed sensor data. In real life, the huge file size means it takes longer to save the file to the storage card than it would for the slowest processor to compress it. So the real bottleneck is the storage card—speed and capacity are both maxed out with RAW files.

Camera manufacturers quickly realized these problems, and many of them decided to apply some kind of compression anyway. Some compress in lossless ways; some use

lossy algorithms. Also, the architecture of the sensor chips can be substantially different from camera to camera, so the raw data stream is almost never conforming to the standard format. And just as EXIF data becomes more and more elaborate, there is more and more metadata embedded in RAW files these days. But unlike the JPEG format, there has never been an official standard notation for metadata in a RAW file. So every manufacturer just makes up its own standard.

And the Ugly: The result is a big mess of more than 100 different RAW formats that can drive software developers up the wall. Serious programming effort is needed to support them all; not even Adobe, with an army of coders, can do that. Unless your camera is highly successful in the mass market, like the Canon Digital Rebel line, chances are its RAW files can only be read by the manufacturer's software. This is a serious flaw because it is limiting your software choices, which either limits your creative toolset or sets you up for a complicated multi-conversion workflow. And if your camera manufacturer decides to drop the support for a three-year-old model, or simply happens to go out of business, you are in serious trouble. You might be left with your entire photo collection as unreadable garbage data.

The worst is that most manufacturers treat their own RAW variant as proprietary format, with no public documentation. Software developers have to reverse-engineer these formats, a very time-intensive, expensive, potentially faulty, and entirely boring thing to do. On top of that, some manufacturers even apply encryption to parts of their RAW files—a mean move that can have no other in-

attention than to cripple compatibility to other companies' brands of software.

This is unacceptable from a user's standpoint. That's why a group of concerned photographers came together and formed OpenRAW.org, a public organization whose mission is to convince the industry of the importance of documenting its RAW formats for the public. And it really is in the industry's best interest to listen to them because it has been shown in the past that compatibility and sticking to open standards can get a company much further than proprietary secrets and isolation. What happened to Minolta, whose JPEGs didn't even comply to EXIF standards? What happened to SGI and its proprietary operation system called IRIX? People saw their software choices limited and were voting with their dollars for a competitor that plays well with others.

Where is our savior? Adobe's answer was the introduction of the DNG file format in late 2004. DNG stands for "digital negative", and it's supposed to unify all possible RAW configurations into one format. It still contains the original data stream from the image sensor, which is different each time. But DNG adds some standardized metadata that precisely describes how this data stream is supposed to be deciphered. The file structure itself is very similar to TIFF and is publicly documented. Obviously, it is a format designed to replace all the RAWs on the camera level. But that hasn't happened yet. Only a handful of high-end professional cameras capture DNG directly now, and the support in the software world is just as sparse. Sadly, for a photographer, DNG is yet another RAW variant right now.

And just when it looked hopeless, a man named David Coffin came into the game, saw

the mess, rolled up his sleeves, and created an open source freeware RAW decoder that supports them all. He called it dcrw, and it looks like he really made it. At last count, it supports 266 different variants, and most applications that support RAW these days do it through dcrw. One man took on the burden of an entire industry by untangling that knot. Thank you, David, for this generous gift to the community!

OK, so does it make sense to shoot in RAW format? Absolutely yes, when you want to get the most out of your camera in a single shot. You will see later on that you can get way more out of multiple shots, and in that case, you might be fine with JPEGs.

But does it make sense to archive my RAW pictures? Maybe, but you should not rely solely on it. You must be aware that your camera-specific RAW format will have a life span that does not exceed the life span of your camera. But the purpose of an archive is to give you a chance to safely restore all data at any point in the future. A standard HDR format is much better suited for archiving because these standards are here to stay. The quality of the preserved data would be the same because an HDR image has even more precision and dynamic range built in than any of today's cameras' RAW files.

However, given the complexity of such a task with today's software, DNG would at least be a future-proof alternative. It's quite unlikely that Adobe will disappear anytime soon, and Lightroom, Photoshop, and Bridge already offer convenient ways to do exactly this conversion in a more convenient manner.



▲ Cineon and DPX

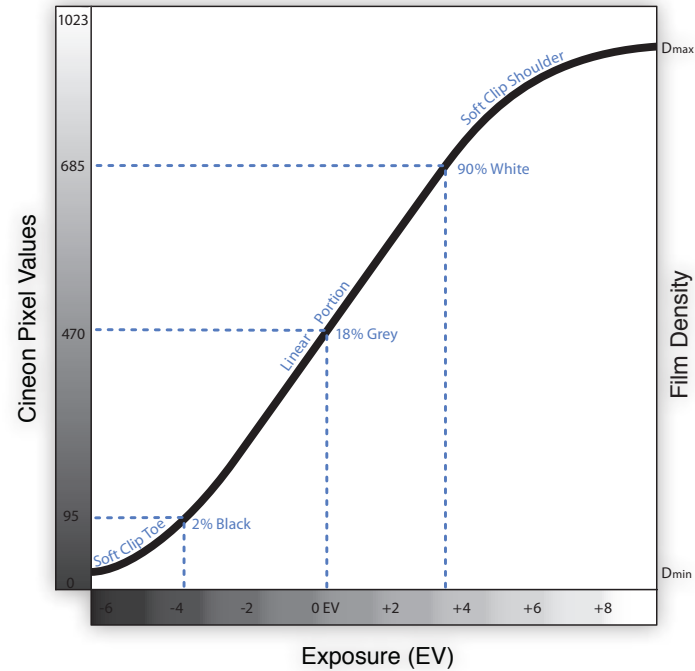
2.1.2. Cineon and DPX (.cin/.dpx)

Cineon was developed in 1993 by Kodak as the native output format of its film scanners. It quickly became the standard for all visual effects production because it is designed to be a complete digital representation of a film negative. Essentially, it's the RAW format of the movie industry. What's not in the Cineon file hasn't been shot.

One important difference is that Cineon files are meant to be worked with. They start their life on film, then they get digitally sweetened with all kinds of visual effects and color corrections, and then they are written back onto film material. There is no room for anything to slip. Cineon is a so-called Digital Intermediate that is between the film reel that was shot on set and the film reel you see in the theatre.

How to file a film negative: What sets Cineon apart is that it stores color values that correspond directly to film densities. As discussed in Chapter 1, film has a logarithmic response to light. That's why color values in a Cineon are stored logarithmically. It even has a mechanism built in to compress the contrasts at the upper and lower knee, just as in the density curve of film. In many ways a Cineon file is more closely related to HDR images

► Cineon pixel values correspond directly to densities on film material, thus sharing film's s-shaped response curve towards captured light intensities. Note that EV is a logarithmic scale, so the linear portion of the curve should be read as "precisely logarithmic".



than traditional 8-bit images, even though it uses only 10 bits. That gives it $2^{10} = 1,023$ different intensity values, which is not much to cover the whole dynamic range of film. Only the logarithmic distribution makes it possible to have enough of those values available for the middle exposures—the meat of the shot. This “sweet spot” exposure range usually is set between the black point at value 95 and the white point at 685. Lower or higher values are the extra dynamic range, values brighter than white or darker than black.

A Cineon file is not made to be displayed as it is. You have to convert the logarithmic intensities to linear values, ideally with a so-called lookup table, or LUT. A LUT is a tone curve that was generated by measuring the light response of film. It's slightly different for each film stock and usually done by the film manufacturer. Often, a generic conversion curve is good enough, but if you know the film

stock that was used, you should use the LUT specific to the film stock. This way, each value in a Cineon file is precisely tuned to match the density of the film negative, and it also points to a linear RGB value that can be displayed on-screen.

These 10 logarithmic bits expand to roughly 14 bits when you stretch them out to linear space, so it makes a lot of sense to convert into a higher bit depth. In 8-bit, you will inevitably lose a lot of details. A 16-bit format is better in precision but still of limited use here because it does not follow the logic of having über-white or super-black values. You can cheat it by setting a white and black point somewhere within the intensity, but either the image you see on-screen will appear to be low contrast or you clip off the extra exposures. Sixteen-bit images are not made for that. Exposure information beyond the visible

range is clearly the domain of HDR imagery, where there is no explicit upper or lower limit.

Unfortunately uncompressed: There is no compression for Cineon files, so they are always wastefully big. Each pixel on disk even includes 2 entirely unused bits. That's because 10 bits in three channels equals only 30 bits; they have to be rounded up to 32 to fit the next magic computer number. Cineons are hard to work with in a compositing environment since it involves dealing with multiple numbered frame sequences, which become huge in this case. A standard 2K resolution film frame takes up 12 MB. Always. There is no such thing as playing this back directly from disk at 24 frames per second on a desktop machine.

Digital Picture eXchange (DPX) is the predecessor of the Cineon format. It is slightly modernized by allowing a larger variety of metadata information. This makes it less dependent on the specific Kodak film scanner devices and makes it more applicable for general use. However, the basic encoding is still the same: uncompressed 10 bits per channel.

DPX and Cineon both only qualify as medium dynamic range formats. Even though they can hold more range than a traditional image, they are limited to represent what can be captured on film. They do catch up with film but don't exceed to fully use the potential of digital imaging. And as digital sensors start to exceed film, Cineon and DPX will fall back.

2.1.3. Portable Float Map (.pfm/.pbm)

Let's move the focus over to the really high dynamic range formats.

The Portable Float Map is a very simple format that is almost identical to the representation of a 32-bit floating-point image in



▲ Portable Float Map

memory. Essentially, it is a fully fledged RAW for HDR images.

A big dumb space eater: We are talking here about full 32 bits per pixel and color in a constant stream of floating-point numbers without any kind of compression. If you add that up, you need 96 bits, or 12 bytes per pixel; that is almost 4 MB for a simple NTSC frame, or 12 MB for a 1-megapixel image. It's huge! This format will eat its way through your disk space in no time!

The advantage of this format is its simplicity. Each PFM file starts with a little header that states in plain text the pixel dimensions of the image. Even a novice programmer can now design software that reads the image data stream. Every programming language on the planet knows the standard IEEE notation for floating-point numbers, so it really is a piece of cake to read one number for red, one for green, one for blue. And voilà—we have an HDR pixel! Do that for all pixels in a row (the header told you how many there are). And then start the next row. No compression, no math, no confusion. Simple and straightforward.



▲ Floating-Point TIFF

Consequently, you are likely to see this format in prototyped software. HDR1 is a very interesting medium for young computer scientists to develop revolutionary imaging technologies. The programs they write are considered a proof of concept, are often bare bones, and sometimes they perform only a single function. It can be very adventurous and a lot of fun to give them a shot, and in that case, the PFM format will be your buddy. Maybe that single function is a killer feature and you have no idea how you could live without it.

In any other scenario, the Portable Float Map is just a waste of space.

Please note that Photoshop prefers to call this format Portable Bitmap Map and uses the extension .pbm. The file itself is identical to a PFM file; it's just the filename extension that limits portability. So in case a program doesn't accept it, you just have to rename it.

2.1.4. Floating-Point TIFF (.tif)

Hold on—aren't we talking about new file formats? Nothing is older than TIFF!

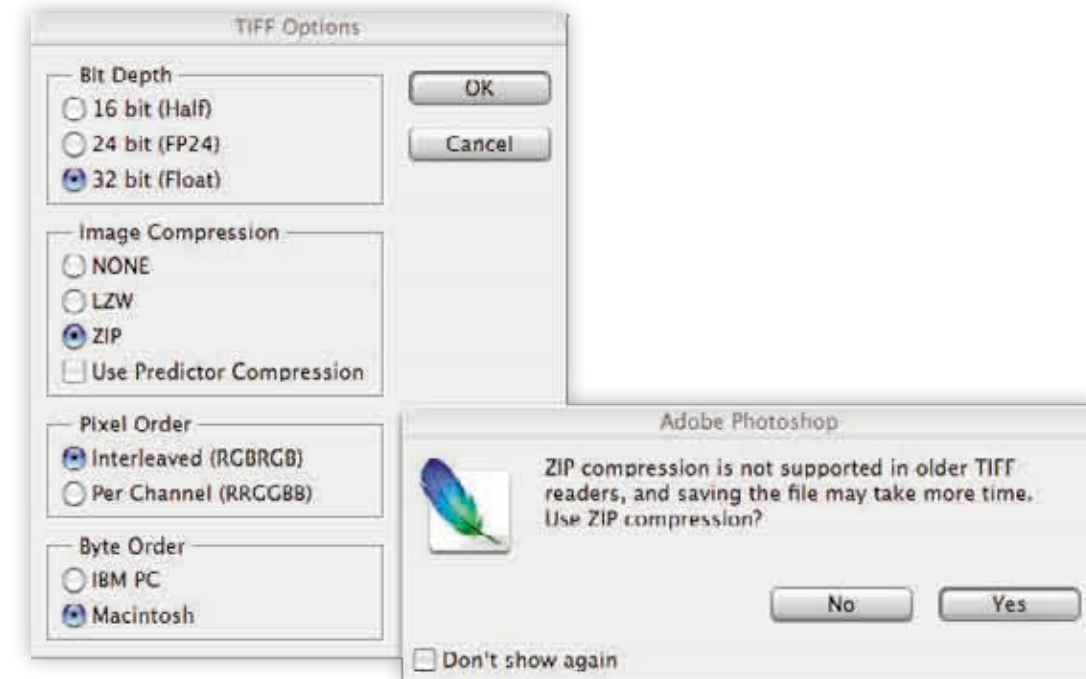
Yes, indeed. Today's TIFF files are still following a specification that dates back to 1992. And that was already revision 6 of the format.

Who knows, it might have been invented before color monitors.

The TIFF format comes in many flavors, and when Adobe bought the format in 1994, it added even more variations. One standard even applies a JPEG compression to the image data, mixing up the whole concept of different file formats having different characteristics.

Whatever you want it to be: In programmers' terms, TIFF is considered a wrapper—because the image data is wrapped in descriptive tags. It's the tagged file structure that makes a TIFF file so versatile because it allows all kinds of different image data to be included. Color space can be RGB or CMYK or something fancy like CIE L*a*b*, compression can be ZIP or RLE, the bit depth can be anything from 1 bit (black and white) to 32 bits per channel. You name it. As long as the data is labeled with the proper tag, the application knows how to deal with it. TIFF is just an acronym for tagged image file format. Well, that sounds good in theory. In reality, not every application supports every compression method or data format, and so it is quite easy to produce an incompatible TIFF. In fact, some features are supported only by the owner, Adobe.

Has all the features, but none of them reliable: However, even the original specification allows a 32-bit floating-point variant, often referred to as FP TIFF, TIFF₃₂, or TIFF Float. This is pretty much the same raw memory dump as the PFM format, and so the file size is just as huge. Compression could technically be used here, but this limits the compatibility again. It doesn't make much sense anyway because old-school compression algorithms like LZW and ZIP are not made for crunching down floating-point data. In fact, the added



◀ Confusing variety of Floating-Point TIFF options, that are only seen in Adobe Photoshop.

overhead can even cause a compressed TIFF₃₂ file to be slightly bigger than an uncompressed one.

And since all features of TIFF can be combined however you want, a TIFF₃₂ file can also have layers and extra channels, they can be cropped to a specific area, and they can have color profiles embedded. It's just that every program has its own way of reading and writing a TIFF, and so there is no guarantee that a feature will work. In a practical workflow, the plain uncompressed TIFF₃₂ is often the only one that actually works.

For a long time the TIFF₃₂ variant was considered a purely hypothetical, ultimate precision format. In the nineties, you would need a Cray supercomputer to deal with it, and so it was used primarily in scientific applications like astronomical imaging and medical research. Even though our twenty-first century

laptops are able to beat these old refrigerator-like monsters with ease, it would be wasteful to use TIFF₃₂ for all our HDR imagery. Especially in compositing, where we deal with image sequences in multiple layers, we would not get anything done because we would spend all day watching a loading progress bar.

What it's good for.: There are occasions, however, when this format comes in handy. In a workflow in which you have to use multiple programs in a row, you can use uncompressed TIFF₃₂ for the intermediate steps. This way you eliminate the possibility of image degrading when the image is handed back and forth. There are other formats that minimize degrading, but nevertheless, even so-called lossless encoding/decoding can introduce tiny numerical differences. Only the straight memory dump method of uncompressed TIFF₃₂ and



▲ Radiance

PFM makes sure the file on disk is 100.00% identical to what has been in memory. It is good advice, however, to get rid of these intermediate files when you're done—before you spend all your money on new hard disks.

2.1.5. Radiance (.hdr/.pic)

The Radiance format is the dinosaur among the true high dynamic range formats, introduced in 1987 by Greg Ward.

Accidentally founding HDRI: His original intention was to develop an output format for his 3D rendering software called Radiance, which was so revolutionary itself that it practically founded a new class of render engines: the physically based renderers. Unlike any other renderer before it, the Radiance engine calculates an image by simulating true spectral radiance values that are based on real physical luminance scales. For example, the sun would be defined as 50,000 times brighter than a light bulb, and the color of an object would be defined by a reflectance maxima. Chapter 7 will give you a more elaborate insight on physically based render engines.

However, these light simulations require the precision of floating-point numbers, and they take a long time. They are still expensive today, but back in 1985 there must have been an

unbearably long wait for an image. Concerned about retaining all the precious data when the result is finally saved to disk, Greg Ward introduced the Radiance format. Initially, he chose .pic as filename extension, but this extension was already used by at least three entirely different image formats. Paul Debevec later came up with the suffix .hdr, which has been the primary extension ever since. This is how they both laid the foundation for high dynamic range imaging, which turned out to evolve into a field of research that has kept them engaged ever since.

The magical fourth channel: Back to the actual format. Radiance is the most common file format for HDR images, and the filename extension basically leads to the assumption it would be the one and only. And really, it does use a very clever trick to encode floating-point pixel values in a space-saving manner. Instead of saving full 32 bits per channel, it uses only 8 bits for each RGB channel. But it adds an extra 8-bit channel that holds an exponent. In short notation, the Radiance format is called RGBE. What this exponent does is free up the RGB values from the extra burden of carrying the luminance information and multiplies the possible brightness range. The real floating-point value of a pixel is calculated with this formula:

$$\frac{(R, G, B)}{255} * 2^{(E-128)}$$

It's a simple math trick, but very effective. Let's look at an example: A Radiance pixel is saved with the RGBE color values (100,100,130,150). The first three values describe a pale blue. Now, watch the exponent at work:

$$\frac{(100, 100, 130)}{255} * 2^{(150-128)} = (1640000, 1640000, 2140000)$$

Compare this to a pixel with the same RGB values but a different exponent, let's say 115:

$$\frac{(100, 100, 130)}{255} * 2^{(115-128)} = (0.0000479, 0.0000479, 0.0000622)$$

See, the exponent really makes the difference here between blue as a desert sky and blue as the entry of a cave in moonlight. It is a very elegant concept, and a math-minded person can easily see the poetry in its simplicity. Using only 32 bits in total, symmetrically split in RGBE channels, we can store a color value that would have taken 96 bits all together. That's why even an uncompressed Radiance image takes up only a third of the space a floating-point TIFF would take.

More range than we'd ever need: Note that we only changed the exponent by 35. But just as with the 8-bit color channels, we have 256 different exponent levels at our disposal. The full numerical range is more than seven times higher than in our example. Actually, this example was chosen carefully to stay within the numerical range that a human mind can grasp. It literally goes beyond the gazillions and the nanos. The dynamic range that this format is able to hold is truly incredible. Speaking in photographic terms, it's 253 exposure values, much higher than any other format can offer. It's even higher than nature itself. From the surface of the sun to the faintest starlight visible to the naked eye, nature spans about 44 EVs. And it is unlikely we will ever take a picture with both in it. A usual, earthly scene

doesn't exceed 20 EVs, even when it's shot in difficult lightning conditions.

Technically, that much available variable space is a waste. Since we will never need that much range for an image, uncompressed HDR files always contain a whole bunch of bits that are just plain zeros. Instead of having 10 times the range that we could possibly use, the format could have 10 times the precision. That way, the colors we actually use would be even better preserved and we would have even less of a risk to get banding and posterization through subsequent editing steps. However, the precision of the Radiance HDR format is still way beyond any 8- or 16-bit format, and since every image source introduces at least some kind of noise, the Radiance format is more than precise enough for general photographic use. Also, there is a run-length encoding (RLE) compression available for this format, which means that the filled-in zeros get crunched down to almost nothing. It's a lossless compression by nature, so you should definitely use it—you can save roughly 30 to 50 percent disk space with it.

But there is another, more serious caveat to the Radiance format. Since the base color values are set in regular RGB color space, it suffers from a limited color gamut, just like many other traditional RGB formats. It's not exactly



▲ TIFF LogLuv

as tight because it can keep color consistent through the whole range of intensities. The available color palette in 8-bit RGB gets very limited close to white and black—simply because all three values are so close to the limit that you have no room left for mixing colors. A Radiance HDR would leave the RGB values alone and make the exponent bigger instead. However, an experienced photographer knows that CMYK is much better in representing printable colors, for example, and if you save an image in RGB, you might have some colors getting clipped off. The same goes for TV production—the NTSC color space is substantially different from sRGB, and you will never know how your final image looks if you don't have a calibrated NTSC control monitor.

Clipping colors is pretty much the opposite of the preservative concept behind HDRI. And even though the Radiance format specifications feature a variant in XYZ color space, that doesn't help us much because this variant is not widely supported.

So when should we use this format, then?

We should use the Radiance format whenever dynamic range is the main concern over

color precision. In image-based lighting, for example, we are aware of the colors getting all scrambled up in our render engine anyway. It's the intensities that we want to feed these algorithms so they can deliver a good lighting solution. We will talk about the details in Chapter 7.

The second major advantage is portability. Radiance HDR is the most common format among the software packages that can work in high dynamic range; in fact, it is the only one that works everywhere. That might be due to its elegant formula and the simplicity of implementation. You would not even need a library or DLL because even a novice can hard-code this conversion. Or it might be due to the historical age of the format or simply the filename extension .hdr that makes it so popular. We don't know. Fact is, Radiance HDR is a well-established standard format, and you are doing good using it when you want to keep all software options open.

2.1.6. TIFF LogLuv (.tif)

Oh no, not another TIFF format!

Yes. TIFF is a 10-headed hydra, and this is yet another head. It's by far the one with the most human-looking face because it stores colors based on human perception.

Greg Ward, once again, developed this format in 1997 when he realized that the popularity of his Radiance format took off in unexpected ways. He was entirely aware of the color and precision limitations that are inherent in the RGBE encoding. So he went back to the drawing board and sketched out the ultimate image format, designed to become the future-safe standard for high dynamic range imaging.

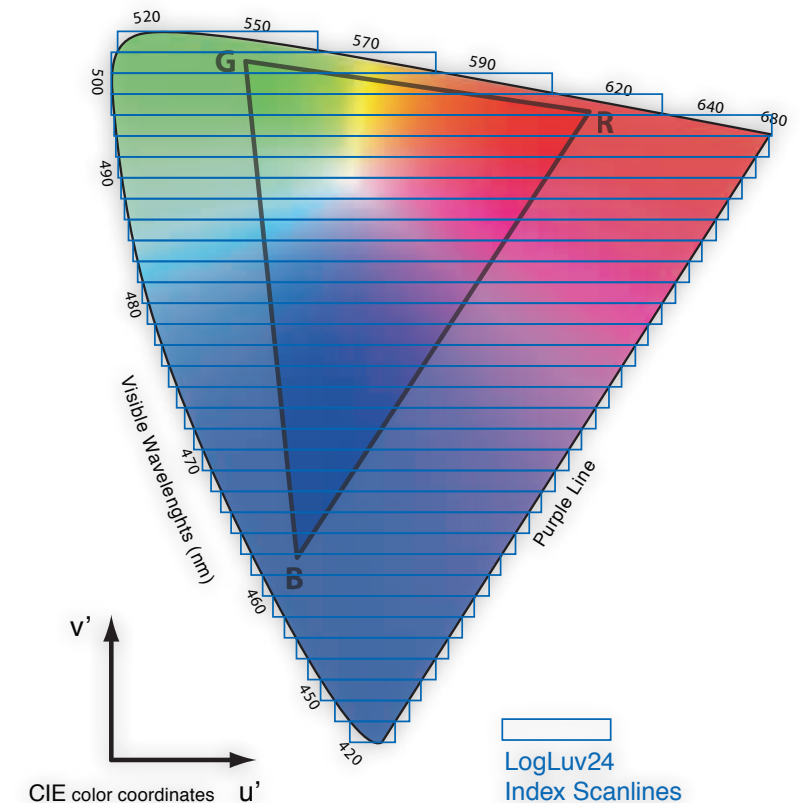
Made to match human perception: And indeed, technically the LogLuv encoding is superior to all others in overall performance: It can represent the entire range of visible colors and stretches over a dynamic range of 16 EV with an accuracy that is just on the borderline of human vision. And it manages to pack all that information in 24 bits per pixel—just the same amount of space that a traditional LDR image takes. It's that efficient.

How is that even possible?

First, instead of the RGB color space, it uses the device-independent LUV color space, the same space that the reference CIE horseshoe diagram is set in. That's how it makes sure that it can really show all visible colors. L represents the luminance, which is logarithmically encoded in 10 bits. That means the smaller the value, the smaller the increments, which is pretty close to the way analog film reacts to light intensities. And as we discussed before, this just happens to be the way the receptors in our eye respond, too.

Now, what it does to the UV color coordinates is so ingenious, it's almost crazy! Instead of picking three primary colors and spanning a triangle of all the representable colors as RGB and CMYK formats do, it simply puts an indexing grid on the entire horseshoe shape of visible colors. It basically assigns each color a number, starting the count from the low purple-bluish corner. The resolution of this grid is set so high that the color differences are just imperceptible to the human eye. Fourteen bits are used for this index, so we are talking about $2^{14} = 16,384$ true colors here.

Sounds good, doesn't it? Everything perfectly aligned to the margin of human vision. The perfect image file for a human beholder, but a perfect nightmare for a programmer. This is not a very convenient way of encoding. It is



▲ TIFF LogLuv's indexed color scheme covers the entire range of visible colors.

super-efficient, but converting RGB to LUV color space first and then using a lookup table to find the color index—all this is somewhat missing the elegance.

There is a variant that uses 32 bits per pixel instead of 24. This one is a little easier, and it's also more precise. Here we have 16 bits instead of 10 for the luminance, and the color values are stored separately in U and V coordinates with 8 bits each. So, it's better suited for images that are not yet in the final stage of editing because there is more room to push pixel values around without having them break through the threshold of human vision.



▲ OpenEXR

Using TIFF LogLuv in practice: Even though the overall performance and efficiency of LogLuv encoding is unmatched, it never became the standard that it was designed to be. Only a few programs fully support it. Some applications can load it but don't offer it as a saving option. Some support the 32-bit variant, but not the indexed 24-bit version. Some programs even think they are loading a floating-point TIFF and load it up as pixel garbage. And from a user standpoint, it is a disaster that it shares the common .tif filename extension, because there is no other way of determining the level of support than the old trial-and-error game. The problem lies within the fact that every program uses a different way of reading and writing TIFFs. If they all would just link to Sam Leffler's library, everything would be fine. But then again, they would be missing other TIFF features that are only available in other libraries.

Bottom line: TIFF LogLuv is just too good to be true. You simply can't rely on it as an exchange format unless you run a test.

2.1.7. OpenEXR (.exr)

OpenEXR is the new shooting star among the HDR image formats. It was born in visual effects production and is specifically designed for image editing and compositing purposes.

It's also the only image format that won the Oscar Award.

Florian Kainz developed this format at Industrial Light & Magic (ILM) in 2000 because he saw none of the existing formats fit the needs of the company. For those few readers who have never heard of ILM, it's the visual effects house of George Lucas and primarily responsible for the effects in the Star Wars saga. The first movie ever completed entirely in EXR was "Harry Potter and the Sorcerer's Stone", and since then this format is used exclusively with all the in-house tools at ILM—it seems to serve them well.

ILM released this format to the public in 2003 as open source. It prepared a great package for developers that includes detailed documentation, code pieces from working implementations, and even example pictures that can be used for tests. Obviously, that was a move to encourage software vendors to make their products fit into ILM's production pipeline. And the response was overwhelming—everybody wants to have ILM on their client list. The beauty of it is that this made OpenEXR widely available for the rest of us too.

16 is the new 32: OpenEXR is indeed a beautifully crafted format. The core specification is based on the so called "half" data type. That means it rounds each color channel's 32-bit floating-point numbers down to 16 bits. That might sound like a drastic decrease, but in reality it's all the precision you need. It's certainly much more precision than the Radiance HDR and TIFF LogLuv have to offer because it keeps the color channels separate at all times. So all three RGB channels add up to 48 bits per pixel, 16 in each channel. Don't confuse this with a traditional 16-bit format—those are

all regular integer numbers, but we are talking about floating-point numbers here. They are substantially different!

The 16 bits here are split into 1 sign bit, 10 bits mantissa, and 5 bits for an exponent. Effectively this mantissa allows you to save $2^{10} = 1,024$ different color values for each channel. If you multiply all three channels, you get about 1 billion possible colors—independent from the exposure. So these are 1 billion colors per EV. This is more than any other format can offer (except maybe the space hogs PFM and TIFF32). And in terms of dynamic range, this is what's carried in the exponent; each exponent roughly equals one EV. That means, OpenEXR can hold about $2^5 = 32$ EVs, which is very reasonable for photographic image sources.

For true dynamic range freaks, there is also a full 32-bit floating-point variant. In most applications this is just bloat, though, and all it does is slow you down. It's comparable to the floating-point TIFF again. The 16-bit version is significantly faster to work with. First of all, it loads and saves faster, but more interesting is that it is natively supported by modern graphic cards from NVidia and ATI. If you are shopping for a new graphics card, look for the "CG shader language" feature. The next generation of image editors will be able to use this for doing all kinds of heavy-lifting work on the GPU, and then you won't even notice the difference between editing an HDR or an LDR image anymore. Sixteen-bit floating point is on the fast lane to becoming one of the hottest trends right now. Programs like Fusion5 and ChocoFlop already offer 16-bit FP as color depth to work in, and there are surely more to follow.

Compression and features: OpenEXR also offers some very effective compression methods. Most widely used is the PIZ compression, which is a modern wavelet compressor at heart. It is lossless and specifically designed to crunch scanned film images down to something between 35 to 55 percent. Another common lossless compression is ZIP, which is slightly less effective but faster. Where speed is a concern, like for 3D texture mapping or compositing with many layers, this could be a better alternative. PIZ and ZIP are both widely supported, and you can't go wrong with either one. There are many other compression methods possible in OpenEXR, and the format is prepared to have new ones included.

Sounds good? Wait—here is the real killer feature! OpenEXR is the only HDR format that supports arbitrary channels. So besides a red, green, and blue channel, it can also include an alpha channel. And others, too—like depth, shadow, motion vectors, material—you name it. These channels can have different compression, different precision (16- or 32-bit), and even different pixel resolutions. That makes it the ideal output format for 3D renderings because it gives you the freedom to fine-tune every single aspect of a rendered image in a compositing software.

Another, truly unique feature is the separation of pixel size and viewer window. For example, an image can have a 100-pixel margin all around the actual frame size. These out-of-frame pixels are normally invisible. But whenever a large-scale filter is applied to the image, like a Gaussian blur or a simulated camera shake, this extra bit of information will be invaluable. On the flip side, the pixel space can also be smaller than the actual frame size. That way, you have a predefined placement for the pixel image, which can be convenient