An OpenEXR Layout for Spectral Images

Alban Fichet
Institut d'Optique
Graduate School -
CNRS - Université de Bordeaux
Charles University

Romain Pacanowski
CNRS - INRIA

Alexander Wilkie
Charles University

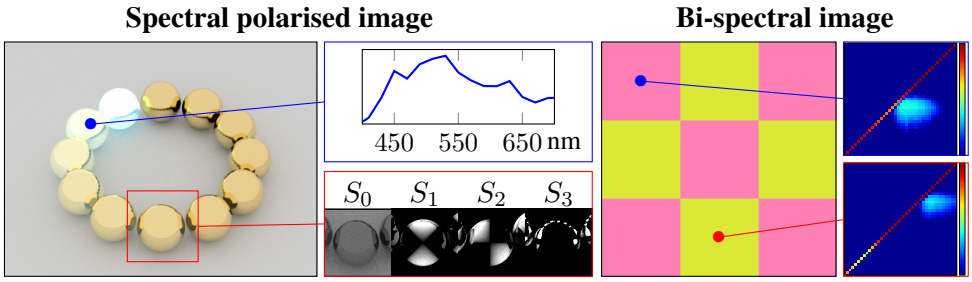


Figure 1. Examples of using proposed OpenEXR layout to store spectral and bi-spectral information along with polarisation state. **Left:** for any given pixel of a radiance image, we can extract the full spectral data (outlined blue). We also store the corresponding Stokes components per wavelength, shown here averaged for all stored wavelengths in a specific area of the image (outlined red). **Right:** a bi-spectral image can store a full re-radiation matrix for each pixel: this is obviously only meaningful as input data for a reflective texture. The sample points are two locations with different bi-spectral reflectances, fluorescent pink (outlined blue) and fluorescent yellow (outlined red). The corresponding RGB image uses a D65 illuminant. These two OpenEXR spectral images are provided in the supplemental materials.

Abstract

We propose a standardised layout to organise spectral data stored in OpenEXR images. We motivate why we chose the OpenEXR format as basis for our work, and we explain our choices with regard to data selection and organisation: our goal is to define a standard for the exchange of measured or simulated spectral and bi-spectral data. We also provide sample code to store spectral images in OpenEXR format.

 ISSN 2331-7418

1. Introduction

In recent years, spectral rendering has become increasingly popular, e.g. for high end VFX work [Fascione et al. 2018]. The main reason for its increasing adoption is that it allows for a far more physically plausible handling of various important visual effects such as skin appearance, at least when compared to tristimulus systems. It also allows proper integration of still rarely found features such as dispersion, polarisation or fluorescence. Spectral renderer output is not tied to any particular colour space: while tristimulus images are intrinsically only valid for a given combination of white point and RGB primaries. However, the output of spectral rendering systems such as Manuka [Fascione et al. 2018] is usually not stored in spectral form: the main reasons why this is rarely done are spectral aliasing issues, storage considerations, and the fact that for most workflows, wide-gamut RGB data in floating point format is sufficient from the point of rendering onwards.

However, there are applications both in research and production environments where storing the actual spectral output of a renderer is useful, so we still need a standardised way of doing this. And there is one additional application area where the ability to use spectral images is arguably far more important anyway: and that is as *input* for the shading subsystem of a spectral renderer. If accurate matches between measured surface appearance and renderings are desired, the ability to handle (bi-)spectral reflectance textures is essential.

Many RGB image formats exist, but – somewhat surprisingly – the same is not true for spectral images. Each pipeline either uses its own proprietary flavour of image format (e.g. ENVI hyperspectral [ENV]), or a collection of monochromatic images stored in set of RGB image format (e.g. METACOW [Fairchild and Johnson 2004]). This makes the exchange, support in software, and long term storage of spectral images difficult.

We propose a spectral image format based on OpenEXR framework: the result is suitable not only for rendering but also for various applications such as acquisition, and as storage format for (bi-)spectral reflectances. In addition, we retain the user friendliness of an RGB image format allowing a quick preview of the spectral image in commonly used graphics software.

2. Related Work

To store spectral images, a simple technique is to provide a collection of standard RGB image files, where each image corresponds to a specific spectral band [Yasuma et al. 2008]. The key advantage of this method is that it allows uncomplicated viewing of all spectral channels with standard software. On the other hand, handling multiple separate files is not convenient, and reconstruction of colour images from the spectral data is tedious.

Specialised formats exist for storing spectral data in specialised fields. For example, the ENVI format [ENV] targets satellite imagery, and separate data blocks are stored for each spectral channel. A header file, in plain text format, describes the information stored in the images, and a binary file is used as dump of the image data. This reduces the amount of separate file types, but still requires two separate files in order to read the image. Moreover, there is no standardised way to store polarisation and bi-spectral information, as especially the latter never occurs in satellite imaging contexts.

In the rendering context, the ARTRAW format was introduced along with the ART renderer [ART 2018]. It supports polarised emissive images, and the spectral sampling options are those of the renderer itself (8, 11, 18 and 46 spectral samples, respectively). The header and data are stored in a single file, in a mixture of plain text (header) and binary (data) formats. This format is mostly used only by this specific renderer, and cannot be displayed without specialised software: it also intentionally eschews all attempts at reducing file size, as it was only ever intended as a lossless reference data storage for raw rendering output. As spectral images tend to be quite large, any reduction in file size would be very welcome: so adopting ARTRAW (which is not very flexible in terms of its internal structure anyway) as a standard would be decidedly sub-optimal.

Finally, there is OpenEXR [Kainz et al. 2004], the de facto standard for storing HDR colourspace images. The official OpenEXR specifications do not cover the use and description of spectral data, but allow very flexible storage of multiple layers of binary data. Some software packages already use layers of the OpenEXR format to store spectral data [Oce 2019]: but, to the best of our knowledge, the specifications used there are limited. These formats generally lack support of polarisation, bi-spectral and capture metadata information with a unified interface.

3. Scope of our spectral image representation

In this section, we describe the range of information we intend to store with our spectral representation.

3.1. Reflective and Emissive spectra

First, we distinguish between *emissive* and *reflective* spectra. Both are used in distinct places within a rendering system, and do not share the same physical units.

An emissive spectrum $L_e(\lambda)$ describes radiant energy within a scene, such as for instance the radiance emitted by light sources. The image computed by a path tracer is a collection of emissive spectral values: it represents the spectral radiance reaching the sensor. The pixels in an environment map are a specialised case of light source, and also contain emissive spectral data. $L_e(\lambda)$ is expressed in $\mathrm{W} \cdot \mathrm{m}^{-2} \cdot \mathrm{sr}^{-1} \cdot \mathrm{nm}^{-1}$

in radiometric units[1].

On the other hand, a reflective spectrum describes the attenuation of energy after reflection, scattering or transmission/volume absorption events. For instance, an albedo texture image describes how a surface attenuates incoming radiant energy upon reflection. The albedo is unitless, and is normally used in the context of a BRDF model (in sr^{-1}) that further describes the directional dependency of surface reflectance (see equation 1).

$$\rho_\lambda(\omega_o) = \int_{\theta_i} \int_{\phi_i} \mathrm{brdf}_\lambda(\omega_o, \theta_i, \phi_i) \cos\theta_i \sin\theta_i \, \mathrm{d}\theta_i \, \mathrm{d}\phi_i \qquad [\mathrm{sr}^{-1}] \tag{1}$$

3.2. Polarisation

Although it is not directly visible to human eyes[2], the polarisation state of electromagnetic radiation can play an important role in light transport for some scenes. Specular inter-reflections are mis-predicted to varying degrees by non-polarising rendering computations, as well as the reflections of blue sky radiance off specular surfaces. For various purposes related to rendering, such as being able to store a polarised environment map of sky dome radiation, and to inspect the polarisation state of reflective highlights in a scene, we want a spectral image format to optionally also store polarisation information.

The mathematical formalism of Stokes vectors and Mueller matrices allows representation of fully, partially or non-polarised light in a compact and intuitive manner that blends in well with non-polarised spectral rendering. Stokes vectors describe light polarisation (see equation 2), and Mueller matrices describe the effect of a scattering event on light polarisation: it encodes the transform of a Stokes vector [Goldstein 2010; Wilkie and Weidlich 2011].

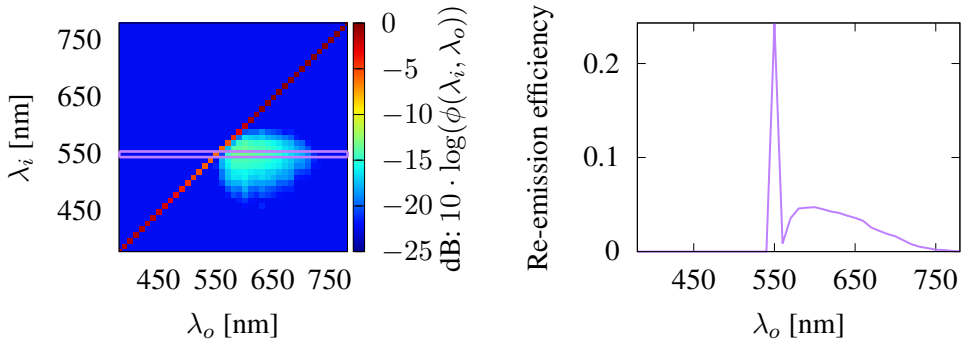
$$S = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} \tag{2}$$

In order to achieve the desired functionality for emissive images, we store the following information, per pixel, and per wavelength:

- all Stokes components S_0, S_1, S_2, S_3 if the image contains polarisation information, or

[1] The corresponding photometric unit is nits, which is used when working with such quantities in a tristimulus pipeline.

[2] At least mostly so: strongly polarised light is weakly visible to some observers, but the effect is not thought to significantly impact how we perceive scenes in general.



(a) *A full re-radiation matrix.* **(b)** *Re-emission for λ_i 550nm.*

Figure 2. **(a)** a full re-radiation matrix like this example completely describes bi-spectral reflectance for the given input (λ_i) and output (λ_o) wavelength ranges. For a given illumination wavelength λ_i, the light re-emitted at various wavelengths is shown as a row in the 2D diagram. **(b)** a plot of the row highlighted in purple in the full matrix at $\lambda_i = 550\,\mathrm{nm}$, including the diagonal part corresponding to the non bi-spectral reflectance.

- only the first Stokes component S_0 if the image does not contain polarisation information.

To use a consistent notation, we always refer to emissive spectral radiance in images as Stokes component S_0, even in unpolarised images.

3.3. Bi-spectral reflectance

Even though this behaviour is not yet commonly simulated in current rendering software, a significant number of man-made and natural materials exhibit wavelength shifting behaviour when reflecting or scattering light. Apart from obviously fluorescent artificial materials such as day-glo warning paint, examples that show at least some degree of such behaviour include wood (lignin is fluorescent), green plants (chlorophyll), and paper and textiles (mostly due to washing powders usually containing optical brighteners that make white cloth look “whiter” by adding a blue fluorescent tinge to it).

Similar to storing “normal” spectral albedo textures that are then used in the context of a BRDF model, complex appearance modelling requires the ability to store extended albedo textures that also describe wavelength shifting. As it can be seen in Figure 2, one needs to store a two dimensional spectrum — the re-radiation matrix — for each pixel in this case: for a given incoming wavelength λ_i, re-emission can occur at various intensities for a wide range of possible output wavelengths λ_o.

3.4. Capture information

For measured spectral images, we also need to store the capture device characteristics as metadata. The Cornell box website [Cor 1998] serves as an example of the sort of data we need to retain: as reference data, they provide a set of monochromatic images of the real Cornell box from $400\ \mathrm{nm}$ to $700\ \mathrm{nm}$ with a sample taken every $50\ \mathrm{nm}$.

In the Cornell dataset, each image was taken with a monochrome camera with a specific narrow-band filter placed in front of the lens: detailed information about the measurement setup is provided along with the raw data on the website. If one wants to be able to interpret the contents of such a measured image in a stand-alone form, directly from the image, this information should be included in the image header. There are two optical elements that alter the energy received by the sensor: first, the camera lenses has a characteristic transmission spectrum. Second, each measurement filter has a transmission spectrum. In addition to this, the camera sensor response is usually not linear, and also needs to be stored. If all this information is retained in the image header, a captured spectral image can be used for long term archival storage without losing the ability to properly interpret its contents.

4. Using OpenEXR for spectral data

OpenEXR supports storage of multiple layers within a single file, but there are some constraints on using this functionality. In this section, we explain our choices made in order to meet our requirements for storing spectral data without violating the OpenEXR format specification.

4.1. Layers and channels organisation

The official OpenEXR specification [Kainz 2013] states that an OpenEXR image can consist of different layers, and that each layer can contain an arbitrary number of channels. In other words, a layer is a group of channels, and from an implementation point of view, layers and channels are indistinguishable. They are merely separated by a naming convention that is enforced by convenient functions in the official OpenEXR library. A layer name is followed by a dot character. Layers can be nested and the tailing string names a channel.

The following outlines the layer naming conventions that we propose.

4.1.1. Layers

Emissive images: layers that store individual Stokes components of a polarised image are named `S0`, `S1`, `S2`, and `S3`. If there is no polarisation information, only the `S0` layer is present.

Reflective images: layers that store reflective or attenuation values are named `T`.

4.1.2. Channels

Each channel is named according to the wavelength or frequency bracket it represents. Since the OpenEXR format reserves dots for separating different layers, we need to use a decimal comma notation for wavelength or frequency values. Wavelength/frequency values are always positive values, and an optional integer power of ten exponent can be also provided with `E` or `e` followed by an optional sign and an integer value.

This value is followed by an optional unit multiplier, and by the used unit. These are either Hertz (`Hz`) or meters (`m`). Table 1 enumerates the different fields that describe a channel name. The channel naming convention is also described in Appendix A.2.

	Information	**Description**	**Example**	**Mandatory**
1	Layer name	Stokes component or absorption	`S0.`	Yes
2	Value	Comma separated	`350,5`	Yes
3	Exponent	Power of ten	`E-3`	No
4	Multiplier	Unit multiplier	`k`	No
5	Units	`Hz` or `m`	`Hz`	Yes

Table 1. Description of channels naming convention.

For instance, the channel "`S0.560,5nm`" corresponds to a S0 component at $560.5\,\mathrm{nm}$.

A regex matching this naming convention can be written as follows:

```
layer = "((S[0-3])|T)"
value = "(\d*,?\d*([eE][-+]?\d+)?)"
units = "(Y|Z|E|P|T|G|M|k|h|da|d|c|m|u|n|p|f|a|z|y)?(m|Hz)"
regex = "^" + layer + '\.' + value + units + "$"
```

4.1.3. Bi-spectral data

To store a re-radiation matrix for each pixel a reflective image, we can also use the OpenEXR layering system, although the resulting image is of considerable size. The main diagonal, which corresponds to the case where there is no re-radiation, follows the same convention as described in 4.1.2. Nested layers store the re-radiation part named with the re-radiation wavelength or frequency using the same comma separated format followed by the units. For instance, re-radiation from $560\,\mathrm{nm}$ to $600\,\mathrm{nm}$ would be stored in a channel named "`T.560nm.600nm`".

4.1.4. Nested hierarchy

We also define a layer convention for child layers, so that it is possible to use a spectral layout within an existing layer hierarchy. For instance, a stereo image with a $550\,\mathrm{nm}$

layers will have its "default" (left) spectral layer stored in "`S0.550nm`" and its right view in "`right.S0.550nm`".

4.2. Metadata

We allow storage of additional metadata to retain capture information using OpenEXR custom attributes in the image header. OpenEXR documentation recommends using types provided from `ImfStandardAtrributes.h`. A summary of all metadata we support is provided in Appendix A.1.

4.2.1. Layout version

To allow future evolution of the proposed spectral layout, we add an attribute that specifies the format version. If the layout is exactly as described as in this article, the value shall be set to "1.0" (see Table 2).

Description	**Attribute name**	**Value**
Version	`spectralLayoutVersion`	"`1.0`"

Table 2. Mandatory version information of the layout in OpenEXR header.

4.2.2. Radiometric units

When the image contains emissive spectral layers, the header has to specify the used radiometric units (see Table 3). For instance, the result of a rendering or a hyperspectral image shall be in $\mathrm{W} \cdot \mathrm{m}^{-2} \cdot \mathrm{sr}^{-1}$.

Description	**Attribute name**	**Value**
Emissive layers radiometric units	`emissiveUnits`	"`W`", "`W.m^-2`", "`W.sr^-1`" or "`W.m^-2.sr^-1`"

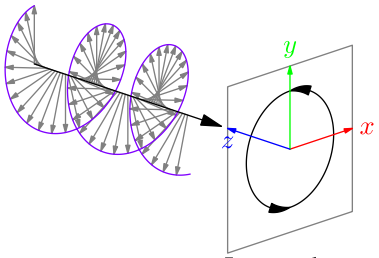
Table 3. Emissive unit information in the OpenEXR header: this is only mandatory for emissive images.

4.2.3. Polarisation

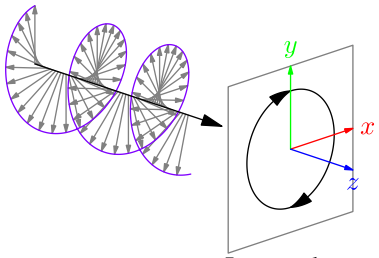
When using emissive polarised image, we need to specify the orientation convention used. Figure 3 illustrates how the same signal can be described as left circular polarised or right circular polarised depending on the convention used.

Description	**Attribute name**	**Value**
Polarisation frame	`polarisationHandedness`	"`left`" or "`right`"

Table 4. Mandatory metadata in OpenEXR header for emissive polarised spectral images.



(a) *When looking toward the lightsource, we observe left circular polarised light in this case. We refer to this as the "*`left`*" handed convention.*



(b) *Looking along the light ray, we observe right circular polarised light in the same scenario as before. This is what we call the "*`right`*" handed convention.*

Figure 3. To avoid confusion when interpreting Stokes parameters, a polarisation orientation convention must be specified. Notice **(a)** and **(b)** have opposite circular polarisation orientation for the same signal depending on the convention used. All extant polarisation renderers we know of (Mitsuba 2 and ART) use the right-handed convention.

4.2.4. *Capture information*

We use optional custom attributes to store properties of the acquisition device:

- The camera sensitivity curve.
- The lens system transmission spectrum.
- A response curve for each `S0` channel.

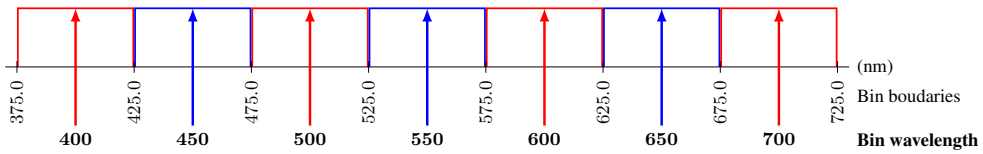
Description	**Attribute name**	**Type**
Lens system transmission spectrum	`lensTransmission`	Spectrum
Camera sensitivity spectrum	`cameraResponse`	Spectrum
Filter / Channel bandwidth	Same as `S0` channel names	Spectrum

Table 5. Optional metadata in OpenEXR header for capture setup information.

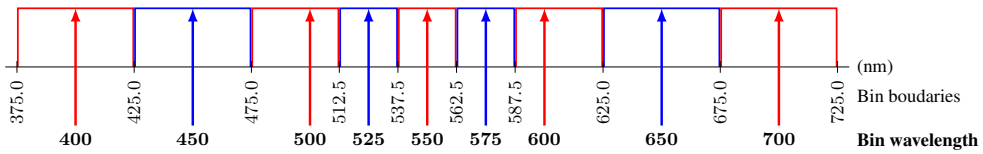
In case that the lens transmission spectrum or the camera sensitivity spectrum metadata field are not present, we assume that this optical element is perfect. When no filter is specified in the header for a spectral bin, the filter is assumed to be a gate function. The filter boundaries $[B_n(\lambda^-), B_n(\lambda^+)]$ for a given bin wavelength B_n are:

$$B_n(\lambda^-) = B_n - \frac{B_n - B_{n-1}}{2} \tag{3}$$

$$B_n(\lambda^+) = B_n + \frac{B_{n+1} - B_n}{2} \tag{4}$$



(a) *Regularly sampled spectrum*



(b) *Varying sampling rate over the spectrum*

Figure 4. Default spectral bin filtering behaviour when no filter is specified in the header.

And the special cases for the first and the last bins are:

$$B_1(\lambda^-) = B_1 - \frac{B_2 - B_1}{2} \tag{5}$$

$$B_N(\lambda^+) = B_N + \frac{B_N - B_{N-1}}{2} \tag{6}$$

Figure 4 illustrates this default filtering behaviour with regular and varying sampling rate.

4.2.5. *Spectrum types metadata*

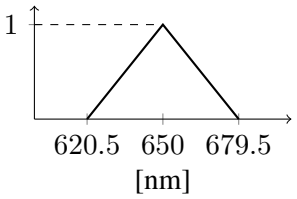
To store spectrum types, we use an `Imf::StringAttribute` packing spectrum in a string. Each pair of wavelength or frequency and value are separated by a colon and followed by a semicolon. We enforce specification of the units of the wavelength or frequency and allow a unit multiplier. Contrary to channel naming, we do not have constraint on character set so we use dot decimals.

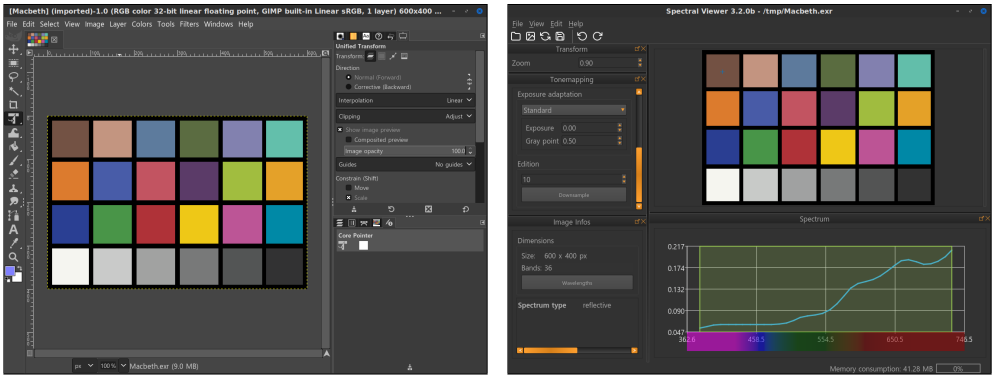
A regex matching this naming convention can be written as follows:

```
value = "(\d*\.?\d*([eE][-+]?\d+)?)"
units = "(Y|Z|E|P|T|G|M|k|h|da|d|c|m|u|n|p|f|a|z|y)?(m|Hz)"
regex =  value + units + ":" + value + ";"
```

For instance, a triangular filter response starting from $620.5\,\mathrm{nm}$ to $679.5\,\mathrm{nm}$ with its maximum at $650\,\mathrm{nm}$ is described as follows:

```
620.5nm:0;650nm:1;679.5nm:0;
```





(a) *GIMP* **(b)** *Spectral Viewer*

Figure 5. The same spectral OpenEXR image opened in two different programs: first, a standard graphic software, GIMP [GIM 1996], which is only capable of handling RGB images. Second, our specialised software for spectral images, Spectral Viewer [Dufay et al. 2018]. **(a)** Gimp cannot access the spectral information but still can render the RGB version thanks to the additional R, G, B channels provided in the spectral OpenEXR image. **(b)** Our specialised software can read spectral information stored in each pixel.

4.3. Data compression

The OpenEXR format offers a wide variety of lossless and loosely compression algorithms, which is interesting for spectral and bi-spectral data that have a significant memory footprint. An overview of the various compression formats available are discussed in the technical introduction to OpenEXR [Kainz et al. 2013]: choosing any particular one of them depends on the application one needs spectral image storage for (that is, whether lossy compression is acceptable, or not). When in doubt, lossless compression should be chosen.

5. Enforcing backward compatibility

OpenEXR is widely supported in graphic packages (GIMP, Photoshop...) in its RGB variant. Although these software packages only support normal colour spaces (RGB, CMYK...), we can provide additional layers in spectral OpenEXR files which allow opening and previewing an RGB version of the spectral image.

This is simply done by adding `R`, `G`, and `B` channels to the layer-set. This behaviour, which is not necessary to have a valid spectral OpenEXR file, is strongly encouraged in the OpenEXR official documentation page [Hillman and Welford 2007].

Multiple spectral to RGB conversions exist. In this section we explain how we keep track of the conversion pipeline.

5.1. Proposed conversion pipeline

5.1.1. Emissive images

For emissive images, we use the `S0` layer and convert the spectrum to RGB using the following pipeline:

1. Emissive spectrum $L_e(\lambda)$ to CIE-XYZ 2° 1931.
2. XYZ to linear sRGB D65.

Steps (1) and (2) correspond to the following equation:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \mathbf{M}_{\mathrm{XYZ}\to\mathrm{RGB}} \cdot \begin{bmatrix} \int_\Lambda L_e(\lambda)\cdot\bar{x}(\lambda)\mathrm{d}\lambda \\ \int_\Lambda L_e(\lambda)\cdot\bar{y}(\lambda)\mathrm{d}\lambda \\ \int_\Lambda L_e(\lambda)\cdot\bar{z}(\lambda)\mathrm{d}\lambda \end{bmatrix} \tag{7}$$

with:

- $\mathbf{M}_{\mathrm{XYZ}\to\mathrm{RGB}} = \begin{bmatrix} 3.2404542 & -1.5371385 & -0.4985314 \\ -0.9692660 & 1.8760108 & 0.0415560 \\ 0.0556434 & -0.2040259 & 1.0572252 \end{bmatrix}$, (8)
- $L_e(\lambda)$ the emissive spectrum,
- $\bar{x}(\lambda), \bar{y}(\lambda), \bar{z}(\lambda)$ the sensitivity curves for the CIE-XYZ 2° 1931 colourspace.

5.1.2. Reflective images

For a reflective image, we use the `T` layer and convert the spectrum to RGB using the following pipeline:

1. Multiplication of the spectrum $S(\lambda)$ with CIE D65 illuminant $L_{\mathrm{D65}}(\lambda)$.
2. Spectrum projection to CIE-XYZ 2° 1931 colourspace.
3. Normalisation of components with CIE D65 Y component (CIE 1931 Y).
4. XYZ to sRGB D65 linear.

Spectral case: The conversion is defined as follows:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \frac{1}{Y_{\mathrm{D65}}} \cdot \mathbf{M}_{\mathrm{XYZ}\to\mathrm{RGB}} \cdot \begin{bmatrix} \int_\Lambda L_{\mathrm{D65}}(\lambda)\cdot S(\lambda)\cdot\bar{x}(\lambda)\mathrm{d}\lambda \\ \int_\Lambda L_{\mathrm{D65}}(\lambda)\cdot S(\lambda)\cdot\bar{y}(\lambda)\mathrm{d}\lambda \\ \int_\Lambda L_{\mathrm{D65}}(\lambda)\cdot S(\lambda)\cdot\bar{z}(\lambda)\mathrm{d}\lambda \end{bmatrix} \tag{9}$$

with:

- $Y_{\mathrm{D65}} = \int_\Lambda L_{\mathrm{D65}}(\lambda)\cdot\bar{y}(\lambda)\mathrm{d}\lambda\,,$ (10)
- $S(\lambda)$ the reflective spectrum.

Bi-spectral case: The integration becomes two dimensional:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \frac{1}{Y_{\mathrm{D65}}} \cdot \mathbf{M}_{\mathrm{XYZ}\to\mathrm{RGB}} \cdot \begin{bmatrix} \int_{\Lambda_o}\int_{\Lambda_i} L_{\mathrm{D65}}(\lambda_i)\cdot S(\lambda_i,\lambda_o)\cdot \bar{x}(\lambda_o)\mathrm{d}\lambda_i\mathrm{d}\lambda_o \\ \int_{\Lambda_o}\int_{\Lambda_i} L_{\mathrm{D65}}(\lambda_i)\cdot S(\lambda_i,\lambda_o)\cdot \bar{y}(\lambda_o)\mathrm{d}\lambda_i\mathrm{d}\lambda_o \\ \int_{\Lambda_o}\int_{\Lambda_i} L_{\mathrm{D65}}(\lambda_i)\cdot S(\lambda_i,\lambda_o)\cdot \bar{z}(\lambda_o)\mathrm{d}\lambda_i\mathrm{d}\lambda_o \end{bmatrix} \quad (11)$$

5.2. Using a different conversion pipeline

One can use one's own conversion pipeline for a specific application. In this case, additional information can be provided as OpenEXR header attributes: the XYZ sensitivity curves and the illuminant spectrum for reflective images and, an additional exposure compensation value (see Table 6). This additional information is especially important for bi-spectral images where the illuminant greatly influences the RGB conversion even for two illuminant with the same white point.

The exposition compensation is defined as follows:

$$\mathbf{RGB}' = \mathbf{RGB} \cdot 2^{\mathrm{EV}} \quad (12)$$

with: EV the exposure compensation value (unitless).

Description	Attribute name	Type	Default
Illuminant	`illuminant`	Spectrum	CIE D65
X CMF	`X`	Spectrum	CIE X 1931 2°
Y CMF	`Y`	Spectrum	CIE Y 1931 2°
Z CMF	`Z`	Spectrum	CIE Z 1931 2°
Exposure compensation	`EV`	Float	0

Table 6. Optional metadata for spectral to tristimulus conversion pipeline. The spectrum format follows the same convention as the one described in 4.2.5.

Specification of the full conversion pipeline, from spectral to linear RGB values, is necessary to validate one's specific conversion pipeline. With this information on hands, a producer and a consumer of a spectral image with RGB channels can validate the consistency of their spectral conversion pipelines.

6. Conclusion

We introduce a versatile, unambiguous and flexible layout for storing spectral data within the existing framework of the OpenEXR image format. This layout supports emissive and reflective spectral images, and also handles polarisation and bi-spectral reflective spectra. In addition to pixel data, we describe how to retain metadata from capture devices: the camera spectral response, lens transmission and filters properties. It also allows to store all spectral and capture information in a single file, and

common graphic software can display the spectral image in its tristimulus form if the RGB channels are also provided. The format also intrinsically supports various compression schemes on the individual channels, by virtue of being part of the OpenEXR framework. Finally, the format is modular and allows additional metadata and layers for specific applications.

Acknowledgements

We want to thank the reviewers for their meaningful advices, among these, contacting the OpenEXR developers. We also want to thank the OpenEXR developers and mailing list members for their valuable feedback. This work was funded in part by an ANR grant ANR-17-CE23-0017 "VIDA". We also acknowledge funding by Czech Science Foundation grant 19-07626S, EU H2020-MSCA-ITN-2020 grant number 956585 (PRIME), and Charles University grant number SVV-260588.

References

2018. The Advanced Rendering Toolkit. URL: https://cgg.mff.cuni.cz/ART/. 67, 80

1998. The Cornell Box. URL: http://www.graphics.cornell.edu/online/box/. 70

DUFAY, A., PACANOWSKI, R., BARLA, P., AND FICHET, A., 2018. Spectral Viewer. URL: https://adufay.gitlabpages.inria.fr/SpectralViewer/. 75

DUFAY, A., MURRAY, D., PACANOWSKI, R., ET AL., 2019. The Malia Rendering Framework. URL: https://pacanows.gitlabpages.inria.fr/MRF/. 80

ENVI Files. URL: http://www.harrisgeospatial.com/docs/ENVIHeaderFiles.html. 66, 67

FAIRCHILD, M. D., AND JOHNSON, G. M. 2004. METACOW: A Public-Domain, High-Resolution, Fully-Digital, Noise-Free, Metameric, Extended-Dynamic-Range, Spectral Test Target for Imaging System Analysis and Simulation. *Color and Imaging Conference 2004*, 1, 239–245. URL: https://www.ingentaconnect.com/content/ist/cic/2004/00002004/00000001/art00043. 66

FASCIONE, L., HANIKA, J., LEONE, M., DROSKE, M., SCHWARZHAUPT, J., DAVIDOVIC, T., WEIDLICH, A., AND MENG, J. 2018. Manuka: A batch-shading architecture for spectral path tracing in movie production. *ACM Trans. Graph 37*, 3, 31:1–31:18. 66

1996. GNU Image Manipulation Program. URL: https://www.gimp.org. 75

GOLDSTEIN, D. H. 2010. *Polarized Light, Third Edition*. CRC Press, 12. 68

HILLMAN, P., AND WELFORD, M., 2007. Storing Multi-View Images in OpenEXR Files. URL: https://www.openexr.com/documentation/MultiViewOpenEXR.pdf. 75

KAINZ, F., BOGART, R., AND HESS, D. 2004. Openexr image file format. 67

KAINZ, F., BOGART, R., STANCZYK, P., AND HILLMAN, P., 2013. Technical Introduction to OpenEXR. URL: https://www.openexr.com/documentation/TechnicalIntroduction.pdf. 75

KAINZ, F., 2013. Reading and Writing OpenEXR Image Files with the IlmImf Library. URL: https://www.openexr.com/documentation/ReadingAndWritingImageFiles.pdf. 70

2019. Ocean 2019. URL: http://www.eclat-digital.com/ocean2019-docs/reference/fileformat/spectralexr.html. 67

WILKIE, A., AND WEIDLICH, A. 2011. How to Write a Polarisation Ray Tracer. In *SIGGRAPH Asia 2011 Courses*, Association for Computing Machinery, New York, NY, USA, SA '11. URL: https://doi.org/10.1145/2077434.2077442, doi:10.1145/2077434.2077442. 68

YASUMA, F., MITSUNAGA, T., ISO, D., AND NAYAR, S. K. 2008. Generalized Assorted Pixel Camera: Post-Capture Control of Resolution, Dynamic Range and Spectrum. Tech. rep., Nov. 66

Index of Supplemental Materials

Sample code

We provide a C++ sample code for reading and writing spectral and bi-spectral OpenEXR files under BSD License 2.0 terms.

- **Spectral**
  The spectral classes (`[EXR]?SpectralImage.*`) are a simpler version for one interested only on spectral part of the OpenEXR. This reads as well bi-spectral files but ignores the re-radiation information.
- **Bi-spectral**
  The bi-spectral classes (`*BiSpectralImage.*`) support both spectral and bi-spectral OpenEXR images.

Sample images

We provide several sample images:

Emissive images

- `D65.exr`
  SPD of CIE D65 illuminant. A 1x1 pixel image giving an example of application for storing spectra.
- `CornellBox.exr`
  A single file spectral EXR version compiled from the official Cornell box data: http://www.graphics.cornell.edu/online/box/data.html. It contains the aggregation of the monochromatic OpenEXR images and the filter responses for each band.

- `PolarisedRendering.exr`
  A spectral EXR file with polarisation information from ART renderer (Advanced Rendering Toolkit) [ART 2018].

- `MatProbe.exr`
  A spectral stereo non polarised EXR file from MRF renderer (Malia Rendering Framework) [Dufay et al. 2019].

Reflective images

- `Macbeth.exr`
  A reflective spectral image. It represents a Macbeth colour chart. The spectral data comes from measurements of several Macbeth colour chart and was taken from `http://www.babelcolor.com/colorchecker-2.htm`.

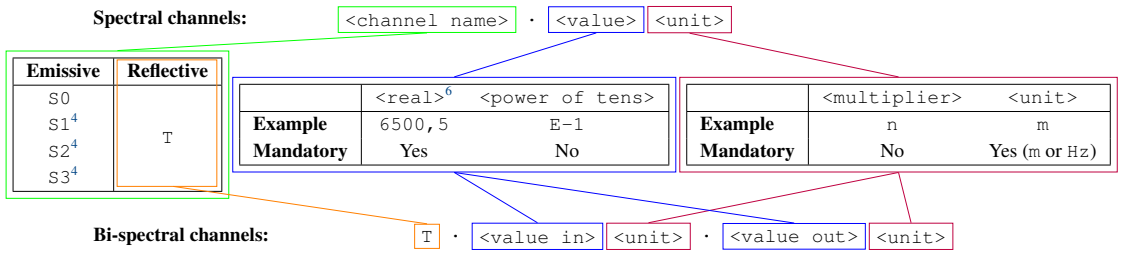
- `BiSpectral.exr`
  A bi-spectral texture made of two different fluorescent reflectance.

A. Appendix: summary of spectral OpenEXR layout

A.1. Header metadata

Attribute name	Description	Type	Mandatory	Default	Table
`spectralLayoutVersion`	Version of the format	String	Yes[3]	-	2
`polarisationHandedness`	Handedness of polarisation observation frame	String (“`left`” or “`right`”)	Yes[4]	-	4
`emissiveUnits`	Radiometric units for emissive Layers	String	Yes[5]	-	3
`lensTransmission`	Lens transmission spectrum	Spectrum	No	flat (perfect optical system)	5
`cameraResponse`	Camera sensitivity spectrum	Spectrum	No	flat (perfect optical system)	5
Same as `S0` channel names	Filter / Channel bandwidth	Spectrum	No	flat (perfect optical system)	5
`illuminant`	Illuminant	Spectrum in a string	No	CIE D65	6
`X`	X CMF	Spectrum in a string	No	CIE X 1931 2°	6
`Y`	Y CMF	Spectrum in a string	No	CIE Y 1931 2°	6
`Z`	Z CMF	Spectrum in a string	No	CIE Z 1931 2°	6
`EV`	Exposure compensation (see equation 12)	Float	No	0	6

A.2. Layer naming



[3] Value must be set to 1.0 for the layout described in this article.
[4] Only for polarised images.
[5] Only for emissive images. Value must be “`W`”, “`W.m^-2`”, “`W.sr^-1`” or “`W.m^-2.sr^-1`”.
[6] Comma separated integer and decimal parts.

Author Contact Information

Alban Fichet
Institut d'Optique Graduate School - CNRS - Université de Bordeaux, Charles University
alban.fichet@gmx.fr
https://afichet.github.io

Romain Pacanowski
CNRS - INRIA
romain.pacanowski@institutoptique.fr
http://manao.inria.fr/perso/~pac

Alexander Wilkie
Charles University
wilkie@cgg.mff.cuni.cz
https://cgg.mff.cuni.cz/ ~wilkie

A. Fichet and R. Pacanowski and A. Wilkie, OpenEXR for Spectral Images, *Journal of Computer Graphics Techniques (JCGT)*, vol. 10, no. 3, 65–82, 2021
http://jcgt.org/published/0003/01/01/

Received:	2021-01-07		
Recommended:	2021-04-20	Corresponding Editor:	Marc Stamminger
Published:	2021-07-15	Editor-in-Chief:	Marc Olano