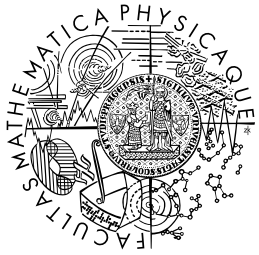


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Lucia Tódová

Constrained Spectral Uplifting

Department of Software and Computer Science Education

Supervisor of the master thesis: doc. Alexander Wilkie, Dr.

Study programme: Computer Science

Study branch: Computer Graphics and Game Development

Prague 2020

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Dedication.

Title: Constrained Spectral Uplifting

Author: Bc. Lucia Tódová

Department: Department of Software and Computer Science Education

Supervisor: doc. Alexander Wilkie, Dr., Department of Software and Computer Science Education

Abstract: Abstract.

Keywords: key words

Contents

Introduction

Over the last few years, the demand for physical accuracy of the rendering process has grown substantially. By providing the renderer with the capability to physically simulate light transport, we can recreate natural phenomena such as metamerism, fluorescence, polarization, etc... To do so, the internal color representation is required to be as close to the visible light's spectral power distribution as possible. However, the advantages of this approach are often not worth the complexity of its implementation.

Even though the most correct physically-based approach to color representation is representing it as it is in nature, which is a spectral distribution of wavelength, its advantages are, in many cases, not worth the complexity of its implementation.

Therefore, vast majority of the conventional renderers internally represent color as an RGB tristimulus value, mainly due to the simplicity and the robustness of such systems. Additionally, almost all of the already existing assets, such as input textures and materials, are defined in RGB, as their creation and usage is fairly simple. Spectral assets require a real-life model whose reflectance must be measured with a spectrometer. Such a process is both tedious and, in many cases, even impossible.

To preserve the physical correctness of the spectral rendering process while utilizing RGB textures as its input, a conversion from the color's RGB representation to its spectral variant is needed. Such process is called *spectral uplifting*, also known as *spectral upsampling*.

However, the conversion process is not straightforward. As the RGB color space is intrinsically smaller than the gamut of visible light, there exist multiple (infinitely many) spectral representations of the same RGB color. Such spectral variations are called *metamers*, and can cause inconsistencies under different lighting conditions. Various uplifting techniques might therefore provide different results, none of which necessarily have to match the real measured spectra.

The goal of this thesis is to extend an already existing uplifting system with the capability to constrain itself with pre-defined spectra:RGB mappings, which must be preserved during the uplifting process. All the other RGB input values should return plausible synthetic data which smoothly interpolates the measured spectra.

Related work

Currently, there exist several approaches to spectral uplifting. They differ in the type of spectra they are capable of reconstructing, the type of gamut they can uplift, the color error caused by round trips, etc...

Unfortunately, many of them have issues. The technique by MacAdam [27] is capable of creating only blocky spectra, which are unsuitable for smooth reflectances usually found in nature. The proposal by Smits [42], although more widely used, is prone to slight round-trip errors, which arise from out-of-range spectra. One of the first approaches that produced smooth spectra was proposed by Meng et al. [30]. However, they did not take energy conservation into account,

which resulted in colors with no real physical counterpart, i.e. no real material could produce such color. Otsu et al. [34] introduce a technique that is capable of outperforming most of the existing approaches under specific conditions. Its drawback is its inability to satisfy the spectral range restrictions, which again causes color errors upon round trips.

In this thesis, we focus mainly on the technique proposed by Jakob and Hanika [20], which employs a pre-built model that is based on spectral representation using sigmoids. This algorithm produces smooth spectra satisfying spectral range restrictions with negligible error.

Jung et al. [21] further improve this technique for wide gamut spectral uplifting by introducing new parameters for fluorescence. Currently, this approach can be considered as state of the art.

None of the existing techniques propose a way in which to constrain the uplifting process.

Layout of this thesis

This thesis is structured as follows:

In Chapter 1, we introduce the reader to light transport and color science. We explain the basic principles of human color perception, overview the existing color representations and focus on their importance in rendering.

Chapter 2 summarizes the already existing spectral uplifting algorithms, placing special emphasis on the technique by Jakob and Hanika [20]. Furthermore, it discusses the theory behind representation of spectra using moments.

Chapter 3 details the implementation of our extension to an already existing model and its utilization in a rendering software. It often refers to chapter 4, where, in addition to discussing our results and comparing them to the non-constrained spectral uplifting, we provide multiple tests and experiments that assess the correctness of various possible implementations.

1. Color Science

Color science, or colorimetry, is a branch of science that concerns itself with human perception of color. It researches the relations between human vision and physical properties of color, and analyzes options for both its capturing and reconstruction.

We begin this chapter by describing the physical properties of light and their subsequent meaning in terms of color. We then provide multiple options for quantifying said color for further possible reconstruction in the digital world. Lastly, we show the importance of color representation in modern-day renderers, and its effects of physically based phenomena.

1.1 Light and Color

Human visual perception refers the ability of the human eye to interpret the surrounding environment. It is based on our capability to detect *electromagnetic radiation*, which is a form of energy that consists of waves which propagate through space and transmit radiant energy.

An *electromagnetic wave* is characterized by its *amplitude* and *frequency*. Amplitude is defined as the distance between the central axis and either the *crest* (the highest point of the wave) or the *trough* (the lowest point of the wave), while frequency specifies how many wave cycles occur in a second. Together, these properties give rise to the term *wavelength*, denoted λ, which measures the length of the wave — the distance between either two subsequent crests, troughs or any two following spots with the same height.

Every electromagnetic wave can be unambiguously defined by its wavelength. Arranging them according to this criterion creates a classification known as *electromagnetic spectrum* (see fig. 1.1). As the electromagnetic spectrum contains all existing types of electromagnetic radiation, it covers wavelengths in the range from fractions of nanometers to thousands of kilometers. This range can be divided into bands to distinguish known types of electromagnetic waves, from low frequency gamma or X-rays to high frequency radio waves.

In this thesis, we focus on *visible light*, which covers only a mere fraction of the electromagnetic spectrum. Its waves are roughly in the 380-780nm range.

To sum up, electromagnetic waves specify the way in which light travels. To, however, describe the interaction between light and matter, the term *photon* is used.

Photons are elementary particles of light moving in a manner specified by their wavelengths. They make up electromagnetic radiation and can be emitted or absorbed by atoms and molecules. During this process, they transfer energy either from the object that emitted them or to the object that absorbed them. This change in energy (denoted E) is proportional to the wave frequency of the absorbed/emitted photon and can be computed as follows [12]:

$$E = hf = \frac{hc}{\lambda} \tag{1.1}$$

where h is Planck's constant, f is the frequency and c is the speed of light.

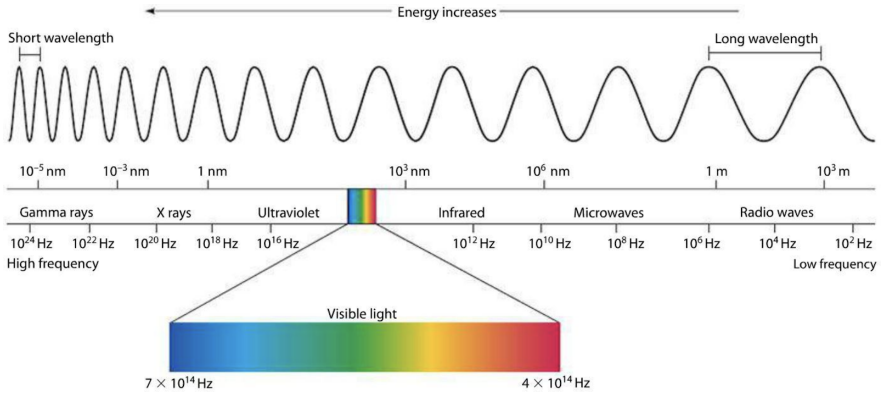


Figure 1.1: An illustration of the electromagnetic spectrum [3]

Therefore, generally speaking, the human eye identifies light when atoms and molecules in the retina absorb photons.

To specify this process, we will first describe the retina. The retina consists of millions of light-sensitive cells, also called *photoreceptors*, which pass a visual signal via an optic nerve to the brain, giving the notion of light and color. There are two types of photoreceptors in the human eye — rods and cones.

Rods make up most of the receptor cells (around 91 million according to Purves et al. [37], but other sources state that their number could be as high as 125 million [48]). They are usually located around the boundary of the retina, and are responsible for low light (*scotopic*) vision. However, they possess very little notion of color, which is also the reason why the human eye has trouble recognizing colors during the night.

Cones are located mainly in the center of the retina and their numbers are a lot lower (from around 4.5 million [37] to 6 million [48]). In contrast to rods, they are active at daylight levels (responsible for *photopic* vision) and have the notion of color. To be specific, different types of cones differ in their sensitivity to photon energies at concrete wavelengths. The final color is then composed by the brain from the stimulation signals sent by each cone.

The human eye has three types of cones:

- *L-cones*, which are the most responsive to longer wavelengths at around 560nm. When stimulated, they correspond to the red color
- *M-cones*, which are the most sensitive to medium wavelengths at around 530nm and correspond to green color
- *S-cones*, which respond the most to small wavelengths that peak at around 420nm and correspond to blue color

Their relative response to stimulation can be seen in fig. 1.2.

This type of color perception is called *trichromatic*, as it uses three types of receptors to create the whole color space. The attempts to simulate such perception in the digital world give rise to tristimulus color representations, which have been widely adapted in color science. We discuss these thoroughly in section 1.2.

Up until now, we have been talking about the interaction of light with the human eye. Photons, however, also interact with objects. As established by the relationship defined in eq. (1.1), the energy transferred to an object upon light

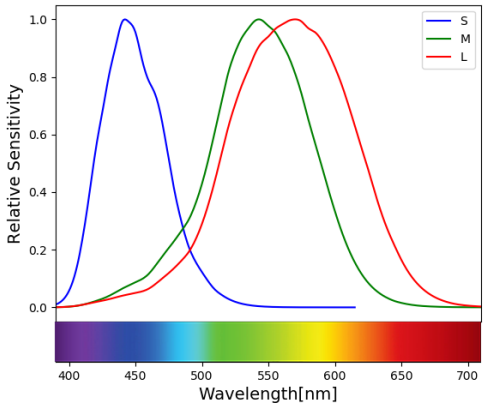


Figure 1.2: Relative sensitivity of S, M and L-cones plotted according to the data measured by Stockman and Sharpe [43].

interaction is dependent on the photon wavelength. This means that objects might absorb some wavelengths and reflect others.

Object color is defined by the wavelengths it *reflects*. For example, if it reflects all the wavelengths, the resulting color is white, while absorbing all the wavelengths would render the object black. Naturally, human perception of object color is not only dependent on its reflective properties, but also on the lighting of the scene. If the only present light is red, wavelengths corresponding to colors other than red never hit the object. Therefore, the object might reflect only a subset of wavelengths than it would under white light, which may subsequently result in a change of the perceived color.

1.2 Color representation

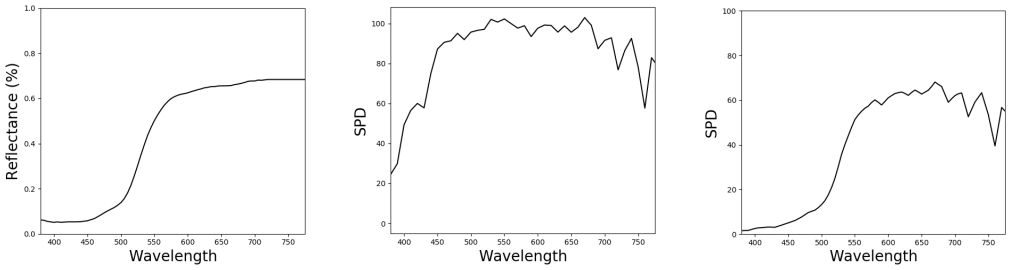
The question of how to discretely represent color has been posed ever since the introduction of the first graphical user interface. For use in computer science, representations are required to be compact, precise, and the operations on colors to be easily executed.

We have already briefly mentioned the tristimulus representation in the previous section. In this section, we will overview its basic properties and describe some of the most popular tristimulus systems. We will also talk about an alternative representation, based primarily on the physical properties of color — the spectral representation.

1.2.1 Spectral representation

When defining the color of an object, we must not only specify the wavelengths it reflects, but also the ratio between the incoming and the outgoing energy at these wavelengths. The dependency of reflectance on the wavelength is called a *reflectance spectrum*, and is usually a smooth, continuous curve (see example in fig. 1.3a).

Although this definition might be sufficient for reflective surfaces, describing the color emitted by a light source requires the knowledge of the source's power



(a) Reflectance spectrum ("orange yellow" patch of Macbeth Color Chart)

(b) Emission spectrum (D50 illuminant)

(c) "orange yellow" patch of Macbeth Color Chart illuminated by D50

Figure 1.3: The behavior of a reflectance of a surface under an illuminant

rather than reflectance. For these purposes, *spectral power distribution* (SPD) is used. Generally, SPD is a function describing the relationship between wavelength and any radiometric or photometric quantity (radiant energy, luminance, luminous flux, irradiance et cetera...). In this thesis, however, we use SPD to describe the emissive properties of light sources, and therefore consider SPD to be a function of wavelength and power. We provide an example of an emission spectrum in fig. 1.3b.

The color of an object illuminated by a light source can be determined by multiplying the light source's SPD curve with the reflectance curve of the object, as shown in an example in fig. 1.3. This way the physical properties of color are preserved and the results are the same as they would be in nature.

1.2.2 Tristimulus representation

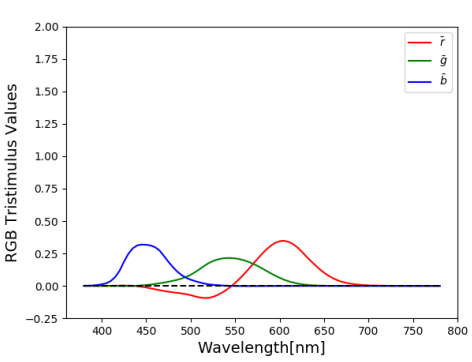
The obvious drawback of spectral representation is the difficulty of its discretization, since there is an infinite number of possible spectral curves, but only a finite number of digital colors that can be displayed by a monitor. For the purposes of color visualization, a distinct, by nature discrete, color representation is required.

Tristimulus representation approaches this issue by saving the color as a set of three values. Although the original idea was to simulate the trichromatic perception of human eye (i.e. save values that specify how much have the red, green and blue cones been stimulated), over time, multiple other tristimulus color spaces have been created. They differ mostly in the range of colors they are capable of representing and in their practical use. Following, we provide an overview of some of the most popular ones.

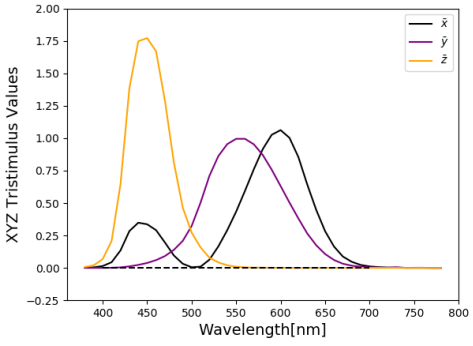
RGB color space

The RGB color space is an additive space employing three primaries — red, green and blue. In other words, if three lights with red, green and blue chromacities are used to illuminate a single point, any color within the RGB color space can be created solely by changing the lights' intensities.

An RGB value can therefore be thought of as a point in a 3-dimensional Euclidean space with each of the coordinate axes representing one of the primaries. As the lights' intensities must be bounded, this space is narrowed down to a cube



(a) $\overline{r}(\lambda)$, $\overline{g}(\lambda)$ and $\overline{b}(\lambda)$ functions plotted with data by Broadbent [5]



(b) $\overline{x}(\lambda)$, $\overline{y}(\lambda)$ and $\overline{z}(\lambda)$ functions according to their spectral data from Choudhury [8]

Figure 1.4: Color matching functions

starting at the base of the coordinate system. Usually, the range for each value is defined within 0 and 255, but a normalized (0,1) range is also used.

Various implementations of the RGB color space exist. They differ in the specifications of the RGB primaries, and therefore in their *color gamut*, which is the subset of colors they are capable of representing. Some examples (named in ascending order with respect to their color gamut) include ISO RGB, sRGB, Adobe RGB, Adobe Wide Gamut RGB and ProPhoto RGB. An illustrative comparison of the sRGB and Adobe RGB gamut in the chromaticity diagram (described thoroughly in section 1.2.2) can be seen in fig. 1.5.

RGB color spaces are commonly used in everyday world, e.g. in LCD and LED displays, digital cameras, scanners and even in computer graphics rendering. Their main downside has, however, been discovered when designing color matching functions [11].

A *color matching function* is a function designed to simulate the response of a certain type of cone in the human eye. In 1931, CIE designed a set of three color matching functions that could be used for spectral to RGB conversion [11]. Denoted $\overline{r}(\lambda)$, $\overline{g}(\lambda)$ and $\overline{b}(\lambda)$, they approximate the response of the L, M and S cones respectively. However, as seen in figure fig. 1.4a, the functions may also acquire negative values. At the time, this posed a problem due to calculation errors. Therefore, to eliminate these negative portions of functions, CIE designed a new, imaginary color space — the XYZ color space.

XYZ color space

The XYZ color space is a hypothetical color space capable of encompassing all colors perceptible by the human eye. Its color matching functions, $\overline{x}(\lambda)$, $\overline{y}(\lambda)$ and $\overline{z}(\lambda)$, were specifically designed for the purposes of SPD to tristimulus conversion,

which is computed using the following equations:

$$\begin{aligned} X &= \int P(\lambda)\overline{x}(\lambda)d\lambda, \\ Y &= \int P(\lambda)\overline{y}(\lambda)d\lambda, \\ Z &= \int P(\lambda)\overline{z}(\lambda)d\lambda, \end{aligned} \tag{1.2}$$

where X, Y and Z are the resulting tristimulus values and $P(\lambda)$ is the spectral power distribution.

Although the X, Y and Z primaries were designed so that the Y primary closely matches luminance and X and Z primaries give color information, they are only imaginary, i.e. they do not correspond to any spectral distribution of wavelengths. This property renders the whole XYZ space imaginary, which means that it cannot be used for visualization purposes. Its main function is to serve as a "middle step" when performing a conversion from SPD to an arbitrary tristimulus space, which eliminates the need for creating color matching functions for other spaces. The conversion from XYZ into a tristimulus space can then be performed by a simple space-specific $3x3$ matrix transformation.

xyY color space

In addition the impossible visualization process, another downside of the XYZ color space is that its values are practically unbounded and do not have any real meaning (such as the RGB triplets have). Therefore, a more intuitive color space has been created, which considers the relative proportions of the X, Y and Z values rather than their unbounded versions — the xyY color space [23]. It is based on the assumption that color can be regarded as a quantity with two properties: *luminance* and *chromaticity*.

The conversion from the XYZ to the xyY color space is performed as follows — first, the X, Y and Z values are converted to their bounded versions (also called *chromaticity coordinates*) as defined in eq. (1.3) [13].

$$\begin{aligned} x &= \frac{X}{X+Y+Z} \\ y &= \frac{Y}{X+Y+Z} \\ z &= \frac{Z}{X+Y+Z} \end{aligned} \tag{1.3}$$

Since $x+y+z = 1$, the z term can be expressed as $z = 1-x-y$. This means that z does not give us any additional information about the current color and therefore can be dropped from the representation. It also implies that some information has been lost during the conversion, i.e. we cannot reconstruct the original XYZ triplet using only the x and y values and therefore cannot obtain the initial color. At least one of the original values is needed for this purpose — CIE [9] decided to use the Y component, as it already specifies luminance.

Plotting the values of the x and y chromaticity coordinates creates a *chromaticity diagram*, shown in fig. 1.5. Each point of the curved boundary line (which is also called the *spectral locus*) corresponds to a XYZ value that is the

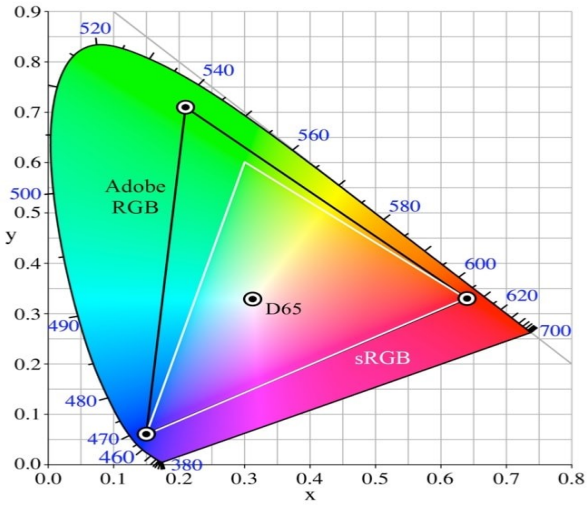


Figure 1.5: An illustrative comparison of the sRGB and Adobe RGB gamut in the chromaticity diagram based on images created by Choi et al. [7]

result of a monochromatic radiation (i.e. a single-wavelength stimulus). All other chromaticities visible to the standard observer lie within a region bounded by the spectral locus.

L*a*b*

Although some of the color spaces we have mentioned so far are already a lot more intuitive in terms of human color perception (e.g. xyY), neither of them regards for the human perception of color differences. The human eye is, for example, more prone to spotting differences when comparing lighter pastel colors and neglecting them upon interaction with darker color (e.g. dark blue). If we wish to accurately describe color differences in a color space, we must regard for this factor and aim for *perceptual uniformity*, i.e. for the difference between two colors (as perceived by the human eye) to be proportional to their Euclidean distance within the color space.

The Hunter's Lab color space [50] addressed this issue and was designed so that the distance between its two triplets characterized roughly how different they are in chromaticity and luminance. It is based on the Opponent color theory [24], which suggests that the cones in the human eye are linked together in opposing pairs and that the visual system records the *difference* between the stimulation of the pairs rather than the cones' individual responses.

As the Hunter's Lab color space does not achieve perfect uniform spacing of values, CIE $L^*a^*b^*$ color space (CIELAB) has been proposed in an attempt to improve some of its shortcomings and is now more widely used. However, neither of the systems are completely accurate in terms of perceptual uniformity [50].

The three opponent channels used to specify color in the CIE L*a*b* color space are defined as follows [17]:

- L^* — indicates lightness, i.e. the difference between *light* and *dark*. Its values range from 0 (yielding black color) to 100 (indicating diffuse white color) [17].

- a^* — defines the difference between *green* and *red*. Positives values of this component indicate the object's color to be more green, while negative values signify the domination of red.

- b^* — defines the difference between *yellow* and *blue*. Positive values indicate the object to be more yellow, while negative values indicate the domination of blue.

Neither the range of the a* nor the b* component has any specific numerical limits [17].

The L*a*b* color space is a *reference system* — an abstract, non-intuitive space encompassing all the human perceptible colors. Due to its perceptual uniformity, it can be used for color balance corrections by modifying the a* and b* components, and for lightness adjustments by modifying the L* component. However, as mentioned earlier, its main purpose is the determining of *color differences*.

In 1976, CIE introduced the concept of *Delta E*, which is the measure of change in visual perception of two colors [16]. Denoted ΔE^*_{ab}, it is computed as an Euclidean distance between the two sample points, i.e.:

$$\Delta E^*_{ab} = \sqrt{(L^*_2 - L^*_1)^2 + (a^*_2 - a^*_1)^2 + (b^*_2 - b^*_1)^2}, \tag{1.4}$$

where (L^*_1, a^*_1, b^*_1) and (L^*_2, a^*_2, b^*_2) are the L*a*b* coordinates of the sample points.

However, the ΔE^*_{ab} error is prone to exaggerate the differences occurring when comparing two highly saturated colors of similar hue. This is due to the fact that the L*a*b* color space is not perfectly perceptually uniform, which can be perceived upon observation of these regions. To improve upon these shortcomings, other measuring techniques for computing Delta E, such as Delta94 and Delta2000, have been proposed over the years.

Delta94 is computed by first modifying the original L*a*b* values of both colors to compensate for perceptual distortions in the color space and then by computing the Euclidean distance from the modified values. Although the results match the human color difference perception more closely, the Delta94 error metric still lacks some accuracy in the blue-violet region [16].

Delta2000 attempts to remove these inaccuracies. Including the corrections added to Delta94, Delta2000 overall adds five correctional factors to the original ΔE^*_{ab} — compensation factors for lightness, hue and chroma, compensation for neutral colors and, lastly, a hue rotation term for the problematic blue-violet region.

From the listed Delta E equations, the Delta2000 error measurements are the most accurate in terms of human color difference perception [16]. However, in addition to being computationally intensive, the Delta2000 equation is, by its nature, discontinous [41]. Therefore, it is not a preferred measure to use in gradient-based optimizers. As we utilize such an optimizer in the practical part of this thesis, we opt for using the simple ΔE^*_{ab} error.

Other color spaces

In addition to the already mentioned tristimulus color spaces, there exist many more used for various purposes. Following, we briefly overview some of them:

- *L*u*v** — Similarly to the CIELAB system, L*u*v* (or CIELUV) aims for perceptual uniformity. As a matter of fact, the $L*$ value is defined in the same manner as in the CIELAB system, while u and v values are evaluated by certain projections of the x and y coordinates of the chromaticity diagram. When comparing their Euclidean error measure, the most important distinction between the two spaces is that while the CIELAB generally improves CIELUV in terms of color difference [29], CIELUV does not have as many inaccuracies in the dark regions [40]. Therefore, it is often recommended to use the CIELUV color space for characterization of color displays and the CIELAB color space for the characterization of colored surfaces and dyes.

- *HSL* and *HSI* color spaces define color by its *hue*, *saturation* and *lightness* (or *intensity*). They are an alternative representation of the RGB color space and must therefore be defined purely with reference to an RGB space [18]. As their components correlate better with human perception of color than those of the RGB system, they are often used in image processing applications, e.g. for processes such as feature detection (edge detection [46], object recognition) or image segmentation (which can be performed solely by using the hue component) [18].

- *CMYK* model is a subtractive color model commonly used in color printing. It is based on RGB's complementary colors — *cyan*, *magenta* and *yellow* (respectively). This means that assigning zero values to all components renders white light, and increasing the value of a component specifies how much of the respective color is *subtracted* from the white light. Although the premise is that maximizing CMY values should render perfect black, in reality, the printing inks are not 100% pure CMY and their combinations therefore cannot produce rich black. For this purpose, a fourth component, *black* (K), is often added, giving rise to the CMYK model.

Other color spaces include the Munsell color system, RAL, Natural Color System, Pantone Matching System, $\text{CIELCH}_{\text{ab}}$, $\text{CIELCH}_{\text{uv}}$, etc...

1.2.3 Color representation in rendering

Accurate color representation is the core of rendering softwares. Although most of today's renderers support multiple color spaces, we can still divide them into two main categories according to the space used during evaluation of light transfer equations — *tristimulus* and *spectral* renderers.

Tristimulus renderers are usually based on the RGB color space, although they often offer conversions to other tristimulus spaces. Due to the ease of use and simplicity of representation, RGB renderers are more common in commercial rendering software. They provide realistically looking images, often indistinguishable from a photograph, and are more robust, memory efficient and easy to implement.

However, light in real world does not travel as a tristimulus value, but rather as a distribution of wavelengths. Therefore, RGB renderers cannot properly simulate the physical properties of color during e.g. reflections or refractions when ray tracing.

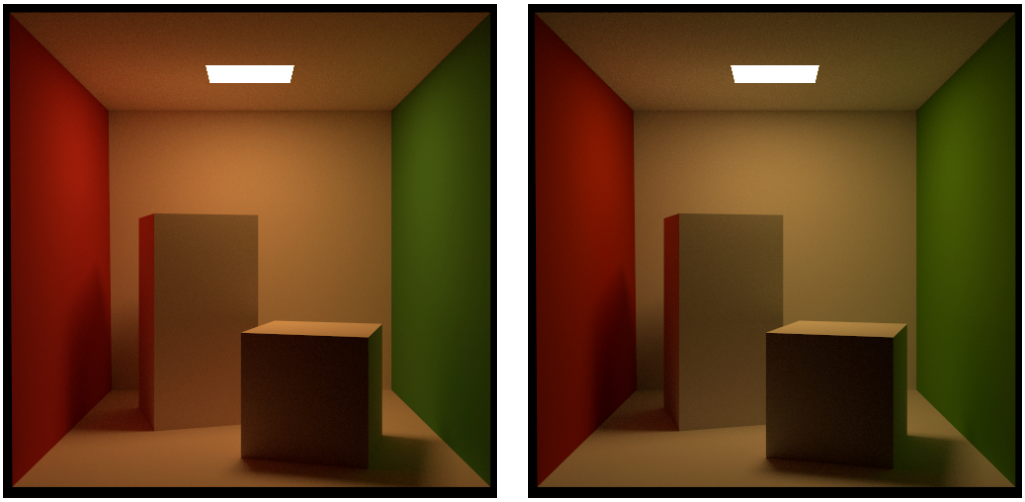


Figure 1.6: Comparison of an RGB-based rendering and spectral-based rendering as presented in the documentation of Mitsuba2 [47]. Left: Spectral reflectance data of all materials is first converted to RGB and the scene is then rendered in the RGB mode, producing an unnaturally saturated image. Right: Scene is rendered directly in the spectral mode, resulting in more realistic colors.

Spectral rendering, on the other hand, uses full-spectral information of all materials and lights in the scene throughout the whole rendering process. Obviously, before visualization occurs, spectral information must be converted into tristimulus (usually RGB) values, but this does not pose a problem as, at the moment of conversion, all the physically based simulations have already taken place. Therefore, the rendered scene appears more realistic. We demonstrate this difference in fig. 1.6, on a scene already rendered by Mitsuba2 [47].

In addition to a more convincing rendering of reflections and refractions, another reason for using spectral rendering is its capability to simulate physically based phenomena that arise due to the interaction of color with light. Following, we overview some of the most common ones:

- *Metamerism*

  As already mentioned in section 1.2.2, the human tristimulus perception has a significantly lower domain than the (practically infinite) spectral domain. Therefore, two different spectra can trigger the same cone response in the human eye and appear to have the same color (and, subsequently, to have the same RGB values), giving rise to a phenomenon called *metamerism*. The two spectra evaluating to the same tristimulus values are called *metamers*.

  In real world, metamerism is often perceived when the lighting conditions under which we observe metamers change. An example of this can be seen in fig. 1.7, where the color of the presented spheres is similar under the D65 illuminant, but clearly differs under the fluorescent F11 illuminant.

  As an RGB renderer does not possess spectral information, it cannot replicate the behavior of spectral reflectance under an illuminant, and is therefore unable to reproduce metamerism.

- *Fluorescence*

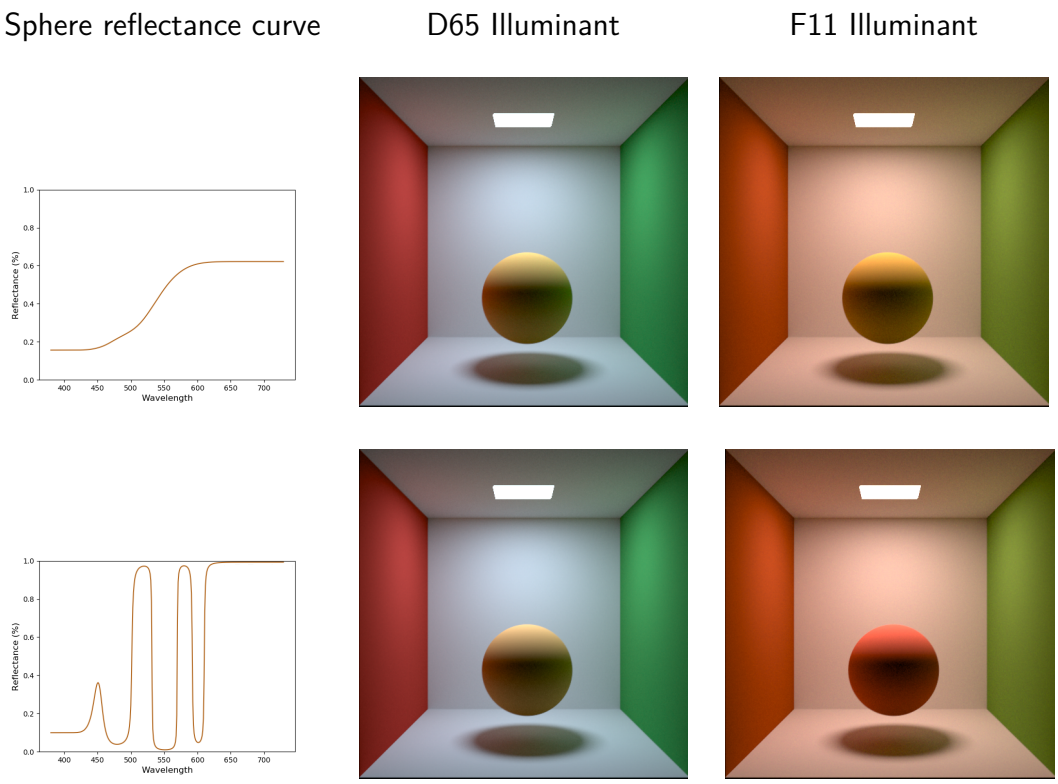


Figure 1.7: The effects of metamerism. Left: Two different spectral reflectance curves, both evaluating to roughly $RGB = (220, 175, 105)$ under D65. Middle and right: Renderings of a sphere with assigned reflectance curves under D65 and F11 illuminants respectively.

By definition, fluorescence occurs when light from one excitation wavelength λ_0 is absorbed by an object and is almost immediately re-emitted at a different, usually longer, wavelength λ_1 [15]. Specifically interesting is the fact that the absorbed light can come from outside of the visible spectrum and be re-emitted inside it, which results in an unrealistically bright appearance of materials. This can be perceived in real world when, for example, fish, corals, jellyfish or even minerals are illuminated by a UV light.

RGB renderers attempt to fake this kind of behavior through custom shaders [52]. As they often produce satisfactory results and, in comparison to physical simulation, are immensely easier to implement, physically based fluorescence has received small amount of work. Its support can be found in spectral renderers, added for example to ART by Mojzík [31].

- *Iridescence*

  Iridescence, or goniochromism, is a phenomenon occurring when certain surfaces change color according to the current viewing angle. It arises when the object's physical structure causes interferences between light waves (e.g. inside extremely thin dielectric layers), yielding rich color variations [4]. It can be perceived in nature in certain plants, specific minerals, butterfly wings, peacock's feathers, snakes, but also in man-made products such as oil leaks, soap bubbles or car paints.

Similarly to fluorescence, iridescent behavior can be "faked" in an RGB renderer [44]. However, research based on physical properties of iridescence has also been conducted. For further information about the current development, we refer the interested reader to the articles by Belcour and Barla [4], Sadeghi and Jensen [39], or Werner et al. [49].

- *Dispersion*

  When light travels from one medium to another (e.g. when light coming from air hits glass or water), its direction of travel is changed. This phenomenon is called *refraction* and is closely described by Snell's law, which specifies how the angle of refraction can be computed from the angle of incidence and the *refraction indices* of the two media [10]. However, the refraction index depends not only on the *type* of media, but also on the current *wavelength* [45] — which implies that the resulting direction of photons may vary according to their wavelength.

  Probably the most known scenario displaying this phenomenon is white light hitting a dispersive prism. Upon interaction, light is split into a spectrum, creating a "rainbow" effect.

  There have been multiple attempts to simulate physically based dispersion. We refer the interested reader to articles by Sun et al. [45] or Wilkie et al. [51].

- *Polarization*

  Electromagnetic waves traveling through space are *transverse waves* — their oscillation is perpendicular to their path of propagation. By default, the directions of oscillations are arbitrary for each photon — this type of light is called an *unpolarized light*. Restrictions to the directions of oscillations (also called *polarization*) render *polarized light*. Such phenomenon usually occurs upon light's interaction with certain materials.

  The polarization process contributes to the overall color only in special cases (e.g. when using polarization filters) [52]. Therefore, it receives little attention in implementation of rendering softwares. However, for physical consistencies (and due to the possibility of unusual scenes) both ART [32] and Mitsuba [47] follow the direction of oscillation during the rendering process.

Other researched phenomena (some of it closely linked to the already mentioned ones) include *phosphorescence*, *bioluminescence*, *dichroism*, *opalescence*, *aventurescence* and many more.

2. Spectral Uplifting

While spectral rendering offers many advantages in terms of physical correctness, it is often neglected and traded off for the RGB color representation due to its ease of use and memory efficiency. Even the physically-based phenomena can be "faked", and, therefore, many conventional rendering systems are RGB-based.

Another reason for the disinclination towards spectral renderers is the deficiency of spectral textures. The process behind their creation is, in comparison to RGB-based textures, a lot more complicated. It usually requires a real-life model whose reflectance spectra must be measured with a spectrometer, which is, in many cases, virtually impossible.

Spectral renderers are, nonetheless, used both in the research and in the commercial sphere (e.g. ART, Mitsuba, Manuka). To eliminate the need for a repeated creation of the textures from scratch, a technique for utilizing the already existing RGB models by converting them to their spectral variants has been proposed. We refer to this process as *spectral uplifting*, however, other sources also use the term *spectral upsampling* [19].

Converting an RGB value into its respective spectrum poses multiple difficulties. As the relationship between the spectral and the RGB domain is not bijective (specifically, infinitely many spectral distributions render the same RGB values), distinctive approaches to the conversion process may render different spectral distributions. Although all of them might be correct in terms of the resulting RGB value, it is possible that none of them would be identical to spectral distribution measured with a spectrometer.

This does not cause a problem under standard illuminant with regard to which the RGB values were uplifted. However, as already mentioned in section 1.2.3, changing the illuminant may result in a rather significant color change due to metamerism. Therefore, our uplifted spectra might behave differently than they would in real world. sem dam obrazok ak ho budem mat

We begin this chapter by reviewing the already existing approaches to spectral uplifting. We then talk about a novel technique, *constrained spectral uplifting*, which provides means for eliminating metamerism during uplifting and the implementation of which is the goal of this thesis.

2.1 Uplifting methods

Although there have been multiple attempts at spectral uplifting, not many meet all the conditions required for a successful and complete conversion. Some of the methods, for example, output reflectance spectra with values outside the (0,1) range, others work for saturated colors only, others cause noticeable round-trip errors (i.e. the difference between the original RGB and the RGB of the uplifted spectrum is not negligible), etc.

Following, we overview the most popular approaches to spectral uplifting and call attention to their advantages and disadvantages. We base most of this section on an article by Jung et al. [22], which, in addition to providing a survey of existing techniques, proposes a new one that is considered to be the current state of the art.

One of the first techniques was proposed by MacAdam [27]. The main goal of his research was to achieve the highest possible brightness for a given color saturation in printing. The uplifting process was only a byproduct of proof of limits to the brightness of colors, created especially for representing the reflectance of the researched colors (i.e colors of maximum brightness for any given saturation). Although this method is not limited to a specific input, it produces spectra that are box-shaped and only consist of rising and falling edges. This type of representation is unsuitable for reflectances usually found in nature, which tend to have a smoother curve.

Another technique was proposed by Smits [42]. In this case, the uplifting is based on a box basis split into 10 discrete bins, which are derived using an optimization algorithm that accounts for energy conservation and aims for overall smoothness of the spectra. This approach is practically implemented and widely used, as it provides satisfactory results in the sRGB gamut [20]. However, in some cases, the uplifted spectra acquire values above 1, which does not satisfy the (0,1) range criterion. Furthermore, round-trips consisting of uplifting an RGB value and converting the resulting spectrum back to RGB produce slight color differences, which are only amplified in scenes with multiple reflections. Lastly, this approach becomes unstable when used with wider gamuts, as it was not designed for this purpose.

The goal of the method proposed by Meng et al. [30] is wide-gamut uplifting. It also concentrates on optimizing the uplifting algorithm for spectral smoothness. However, it does not take energy conservation into account, which results in images with colors that have no physical counterpart (i.e. no real material could produce such colors). Meng et al. [30] try to solve this by introducing a set of scaling methods for mapping the uplifted spectra to valid reflectances. These, however, fail upon uplifting bright colors.

One of the most recent uplifting techniques has been proposed by Otsu et al. [34]. It is based on the observation that a typical measured reflectance spectrum can be represented with only a few principle components. The method uses clustered principal component analysis (PCA) and, unlike many other approaches, does not presume that spectra must necessarily be smooth. Such a simplification both eliminates the requirement of having a smoothness heuristic and enables the reconstructed spectra to match the actual measured spectra pretty well. This approach, however, has its downsides. Firstly, the method does not satisfy the (0,1) range criterion. Therefore, the values must be clamped, which results in color reproduction errors. Moreover, since there is no interpolation across clusters, similar RGB values might produce very different spectra, which might lead to discontinuities in rendering. However, in multiple cases, this method has been shown to outperform all of the already mentioned ones [20].

A large part of this thesis is based on the work of Jakob and Hanika [20]. We will therefore describe their approach in more detail.

In their article, Jakob and Hanika [20] describe a parametric function space for efficient representation of spectral reflectance curves. They also show how to utilize such a space for the purposes of spectral uplifting.

The main goal of their research was to create a spectral representation that would be both energy-conserving and would have a successful round-trip, i.e. the Delta E difference between the original RGB and the RGB obtained by conver-

sion to spectra and back would be as small as possible. Based on the equation specifying the Delta E error, a simple analytical model has been created. Spectra in accordance with this model are represented as following:

$$f(\lambda) = S(c_0\lambda^2 + c_1\lambda + c_2), \tag{2.1}$$

where $f(\lambda)$ is the resulting spectrum, S is a simple sigmoid function and c_i are coefficients of a second-order polynomial. Therefore, all spectra in this space are represented by three parameters.

In addition to energy conservation, the resulting spectra do not violate the (0,1) range constraint. They are smooth and simple, which corresponds to many spectra typically found in nature. Another great advantage is memory efficiency, as storing one spectrum requires only three values. However, representing spectra as such also has its drawbacks. For example, there is currently no straightforward, well-defined computation of the RGB $\rightarrow$ spectrum conversion in such a domain. To uplift an RGB value, one must keep "guessing" the coefficients until the spectrum evaluates to the desired RGB.

In this specific implementation, the "guessing" process is performed mostly by the CERES solver [1]. It requires only an initial guess and a metric according to which it improves the guess (i.e. the Delta E error originating from round-trips) and requires only a few iterations to converge to 0.

The uplifting process itself works by pre-computing RGB:spectra mappings and storing them in a texture. During rendering, only the required spectra are looked up.

Obviously, it is impossible to store mappings for every RGB triplet — the RGB space needs to be discretized as efficiently as possible. Jakob and Hanika [20] propose a specific discretization method of the sRGB space, which divides the space into three quadrilateral regions in which the coefficients are very smooth. For satisfactory results, only three 3D cubes of size 64^3 are required. Another approach is to store all mappings in a table based on a 3D regular grid inside an RGB cube [21]. As this approach is already used in ART and is very similar to the one in this thesis, we describe the pseudo-algorithm used to create such an uplifting model in algorithm 1.

Discretization of the RGB space does not, however, eliminate the need for uplifting RGB values that have no mapping in the spectral uplifting model. In such case, the unknown spectral reflectance values must be computed from the already existing mappings. Without much elaboration, three straightforward methods of how to create a mapping from an arbitrary point in the RGB cube come to mind:

1. copying the coefficients of the closest lattice point in the RGB cube
2. interpolation of spectra of the neighbor lattice points
3. interpolation of coefficients of the neighbor lattice points

The original paper suggests that interpolating coefficients should, within limits, produce reasonable spectra without unexpected artifacts. However, even despite the longer rendering time, the spectral uplifting tool in ART interpolates spectra instead [33]. The reasoning behind is that it provides higher round-trip accuracy,

Algorithm 1 Spectral uplifting by Jakob and Hanika [20]

create $RGBCube$ with empty RGB:spectra mappings
$unfittedPoints \leftarrow$ a list of all points in $RGBCube$
$centerPoint \leftarrow$ index of the middle of $RGBCube$
▷ $RGBCube[centerPoint].rgb \simeq (0.5, 0.5, 0.5)$
$centerPoint.coefficients \leftarrow (0, 0, 0)$
▷ "guess" the coefficients at $centerPoint$
run the CERES optimizer for $RGBCube[centerPoint]$
remove $RGBCube[centerPoint]$ from $unfittedPoints$
while $unfittedPoints$ is not empty **do**
    for all $point \in unfittedPoints$ **do**
        if $point$ has a neighbor v with defined coefficients **then**
            $point.coefficients \leftarrow v.coefficients$
            run the CERES optimizer for $point$
            if optimization was successfull **then**
                remove $point$ from $unfittedPoints$

especially in a case such as this — when the spectra are smooth and similar to those of their neighbors. The approach of copying the coefficients is not usually used, as the output image is slightly darker and less saturated even compared to the interpolation of spectra or coefficients.

We show the differences between these three approaches in fig. 2.1. We include the original texture and its three renderings with the above mentioned methods. The discontinuity in the dark blue region, which arises especially with the nearest neighbor approach, can already be perceived by the human eye in the images. However, as most of the differences are barely noticeable, we also include the most interesting difference images.

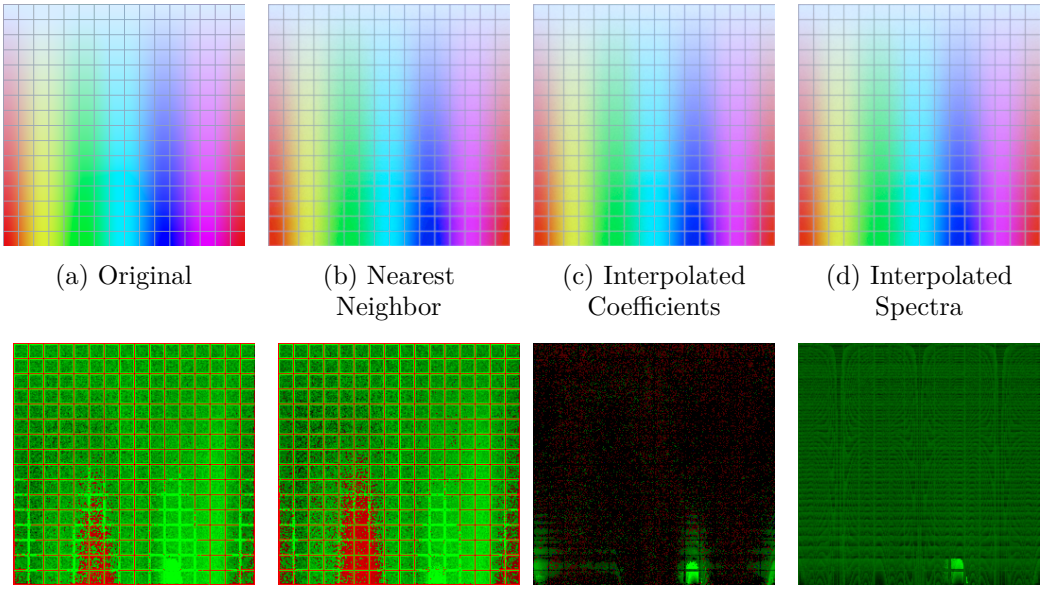
As seen in fig. 2.1g, the approaches of interpolating spectra and coefficients are extremely similar. This and results from both fig. 2.1e and fig. 2.1f imply that all approaches tend to create a bit darker images overall, with an exception of the green region, which is, in contrast, a bit lighter. The nearest neighbor approach is the darkest of the three, which can be seen in fig. 2.1h.

An obvious case in which it is possible to avoid any kind of interpolation or approximation is when the RGB values that will be required during the uplift are known beforehand. They can then be added as lattice points to the RGB cube. This is especially useful when attempting to uplift specific textures or materials, which is, in fact, what this method was originally intended for.

This uplifting model is, in many cases, superior to the ones already mentioned above. First of all, the round-trip error yields 0 in the sRGB gamut. In other gamuts within the spectral locus, it outperforms other models as well. Moreover, the execution speed is by far the best, even with the uplifting process running beforehand.

The smoothness of the spectra can be considered both an advantage and a limitation. Although such spectra closely resemble real-life spectra and are suitable for the interpolation process, they cannot describe extremely bright and saturated spectra, as these tend to be more blocky.

The approach by Jung et al. [21] tries to solve this issue by extending the set



(a) Original (b) Nearest Neighbor (c) Interpolated Coefficients (d) Interpolated Spectra

(e) Original vs Nearest Neighbor (f) Original vs Interp. Spectra (g) Interp. Spectra vs Interp. Coefficients (h) Interp. Spectra vs. Nearest Neighbor

Figure 2.1: Comparison of techniques for obtaining spectra for arbitrary colors in the RGB cube. a) shows the original texture (converted from .exr), while b), c) and d) are rendered with ART by uplifting the texture to its spectral version. All the other figures show differences between some of the pairs. The exposure of the e), f) and h) difference images is increased to 3 for the errors to be visible, and the exposure of the g) image is increased to almost 7.

of three sigmoid coefficients with three additional ones specifically designed for handling fluorescence. Its goal is to avoid creating blocky spectra at gamut boundaries and rather create smooth spectra with added fluorescent dyes to compensate for the lack of saturation. Similarly to Jakob and Hanika [20], the uplifting model is also based on an RGB cube structure optimized by the CERES solver. It is expected for the optimization to take longer, as it has 6 coefficients to consider. However, the method is the first one capable of simulating fluorescent spectra, which is especially useful for wide-gamut input textures.

The problem arising with a requirement to uplift RGB values that do not have a mapping, which we discussed in the previous approach, is solved differently in this case. As both the nearest neighbor and interpolation of coefficients do not produce satisfactory results, *reradiation matrices* of the neighbor lattice points are used. Although this leads to higher memory requirements, the results are smoother and do not produce disruptive artifacts.

In this section, we have presented multiple techniques capable of spectral uplifting. They differ in numerous aspects — the round-trip error, execution time, the gamut they are capable of uplifting, whether all uplifting constraints are satisfied, etc. However, the aspect we focus on most in this thesis is the shape of the uplifted spectra, which further affects the color's behavior during rendering, especially under various illuminants. We show a comparison of spectral shapes created by different techniques by uplifting the same RGB value in fig. 2.2.

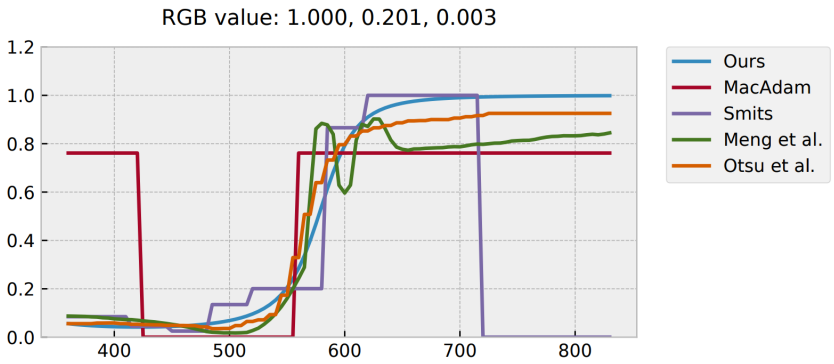


Figure 2.2: Comparison of spectral uplifting techniques as shown by Jakob and Hanika [20]. All spectra were created by uplifting the (1, 0.201, 0.003) RGB value and the results were plotted according to the corresponding techniques. The "Ours" approach, in this case, refers to the approach by Jakob and Hanika [20].

2.2 Constrained spectral uplifting

Achieving identity of our uplifted spectra to the real-world spectra is, obviously, impossible. However, uplifting many RGB-based models does not require us to be able to uplift the whole RGB gamut, but only the spectra used for the creation of said models. As it is pretty common for artists in the VFX modeling industry to use specific color atlases when designing textures and materials, the ability to *constrain* the uplifting system with these atlas colors would be extremely useful in such cases.

In other words, the user would define specific RGB:spectra mappings which would later be used in order to uplift certain RGB triplets. RGB values that would not have a pre-defined mapping would be uplifted by altering the curves of their already-mapped neighbors.

We call this process *constrained spectral uplifting*. The fact that it does not provide as much freedom as other spectral uplifting approaches works for our benefit, as the results are not a subject to high metamerism, which is, after all, the goal of this thesis.

In this thesis, we base the algorithm used to implement constrained spectral uplifting on algorithm 1. We also use an RGB cube as a structure for saving the mappings, and we also create an uplifting model before rendering. We leave the specific details of implementation to ref. In this section, we discuss the theoretical background of the constraining itself. Specifically, we focus on the means of storing the individual spectra in our structure and the problems that arise along with it.

2.2.1 Spectral sampling

The constraining process starts when the user inserts a set of spectra. Finding an RGB match and creating a mapping is a straightforward task — one must simply convert the spectra to RGB. However, the spectra must then be stored in the structure, which requires its discretization.

As the spectra must already be discretized on the input by the user (e.g. each spectrum can be defined in a form of an array which specifies reflectance values in 1nm intervals, starting from 380nm), the first approach that comes to mind is to simply store its every ith value. However, for a good color reproduction, around 30 samples are needed for each spectrum [36]. Storing so many values is extremely memory inefficient, especially when taking into account all the other mappings that must be created later in the system.

Another issue with such storage arises during the approximation stage of the uplifting process. Trying to approximate so many values is infeasible for any optimizer. We must therefore create a more compact representation, which recreates the spectra as efficiently as possible.

We have already talked about approaches to spectral uplifting in section 2.1. However, all of them were concerned with the opposite problem — how to create *any* kind of spectra that evaluates to a specific RGB value. They are therefore restricted to a concrete spectral space and unable to recreate every possible curve. We see an example of this in fig. 2.3a, where the resulting spectra of the uplifting process of multiple methods are compared to the actual measured spectrum of the real-life material.

The simple and smooth shape of the spectra indicate that using a lower-dimensional linear function space, such as Fourier series, could be the key to their storage. Techniques based on this observation have been studied for the storage of emission spectra [38], and appear promising also for reflectance spectra. This method is studied in an article and subsequent presentation by Peters et al. [36]. As the reflectance spectra are aperiodic, it is reasonable for the Fourier basis to consist of cosine transforms only. Eventually, a truncated Fourier series is used for the reconstruction, which is computed according to the following equation:

$$f = \sum_{i=0}^{m} c_j cos(j\varphi) \tag{2.2}$$

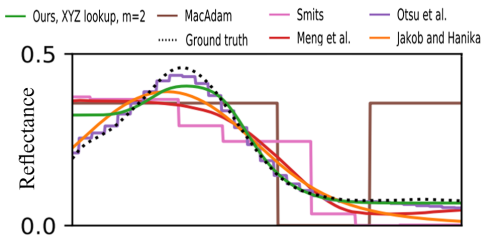
where c_j are the Fourier coefficients eventually stored in the RGB cube.

We show an example of a result obtained by this method in fig. 2.3b. Although the reconstruction is not far off, the resulting spectra do not always have a physical counterpart, as the reconstruction does not obey the (0,1) reflectance range constrain. We can see this behavior even in our example in fig. 2.3b.

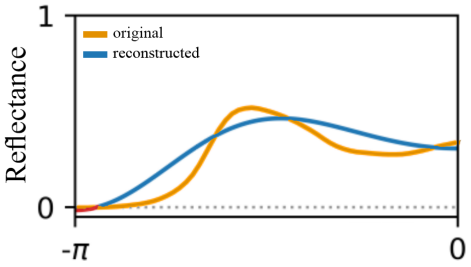
In contrast to linear function space, spectra can also be represented non-linearly. These representations are, however, incompatible with linear prefiltering of textures [35].

Therefore, a novel approach has been proposed by Peters et al. [35] in order to eliminate the flaws of both linear and non-linear approaches. In contrast, it utilizes the strengths of these approaches — the representation consists of Fourier coefficients (which implies compatibility with linear filtering), and the reconstruction is non-linear, based on the theory of moments, and aims for the (0,1) range constraint satisfaction.

Following, we provide a brief overview of both the algorithm for obtaining coefficients and the reconstruction process. For more details, we refer the interested reader to the original article [35].



(a) Reconstruction with uplifting models as plotted by Peters et al. [35], where *Ours* represents a newly proposed technique



(b) Reconstruction with truncated Fourier series [36]

Figure 2.3: Comparison of real-life measured spectra and a few techniques that aim for reconstruction of this spectra.

Obtaining coefficients The first problem with obtaining the coefficients is caused by the shape of the spectra. In contrast to the Fourier basis, they are aperiodic. Their storage with Fourier coefficients therefore requires their conversion to a periodic signal.

Wavelengths can be mapped linearly to a 2π-periodic signal. This, however, causes distortions and strong artifacts at the boundaries. Moreover, Fourier coefficients computed for such signal are complex, which requires almost twice the memory for storage.

These problems can be solved by mapping only the negative values of the signal as in the following equation:

$$\varphi = \pi \frac{\lambda - \lambda_{min}}{\lambda_{max} - \lambda_{min}} - \pi \in [-\pi, 0] \tag{2.3}$$

By mirroring the signal for the positive part, i.e. defining the resulting mapping as $g(\varphi) = g(-\varphi)$ for all $\varphi \in [0, \pi]$, we get smooth transitions at boundaries. The Fourier coefficients are then computed only from this mirrored signal and the reconstruction also uses only that part of the signal. Although this might seem wasteful, the signal created by this approach is even and therefore requires only real Fourier coefficients for its representation, which benefits the storage requirements. We call this approach to the mapping *mirroring*.

Another proposed improvement to obtaining the coefficients is focusing accuracy on important regions, also called *warping*. This is achieved by means of a differentiable, bijective function that maps the wavelength range to the $[-\pi, 0]$ and is used as a weighting function when computing coefficients. This is useful especially when using only small number of coefficients that are unable to capture more complex curves.

Note that using m complex Fourier coefficients for storing a spectrum implies that that $m + 1$ coefficients are actually saved. The $+1$ factor stands for the zeroth moment c_0, which is real in both the mirrored and the non-mirrored case. Therefore, overall, mirroring requires storing $m + 1$ scalars, while non-mirroring requires $2m + 1$ scalars.

Reconstruction The default Fourier coefficients (without improvements such as mirroring and warping) are stored for a 2π-periodic signal $d(\varphi)$, where $d(\varphi) \geq 0$ is a density for all phases $\varphi \in \mathbb{R}$. Therefore, they satisfy the definition of trigonometric coefficients for the *trigonometric moment problem* [26]. Specifically, the coefficients γ can be expressed as

$$\gamma = \int_{-\pi}^{\pi} d(\varphi)c(\varphi)d_{\varphi} \in C^{m+1}, \tag{2.4}$$

where $d(\varphi)$ is the finite measure that they represent, and $c(\varphi)$ is the Fourier basis.

By building upon this observation, the reconstruction of spectra is based on the theory of moments, specifically on Maximum Entropy Spectral Estimate (MESE) [6]. The MESE has been shown to produce impressive results when used for the reconstruction of emission spectra, as it is capable of reconstructing both smooth and spiky spectra.

However, the problem with this approach is that it is not bounded, i.e. not suitable for reflectance spectra. Therefore, a novel, *bounded MESE*, is introduced. It is based on the research by Markoff [28] and, subsequently, Kreĭ [25], who developed a duality between bounded and unbounded moment problems formulated in terms of Herglotz transform. This duality is used for transforming trigonometric moments to *exponential moments* so that the bounded problem represented by the trigonometric moments has a solution if and only if the dual unbounded problem represented by the exponential moments has a solution.

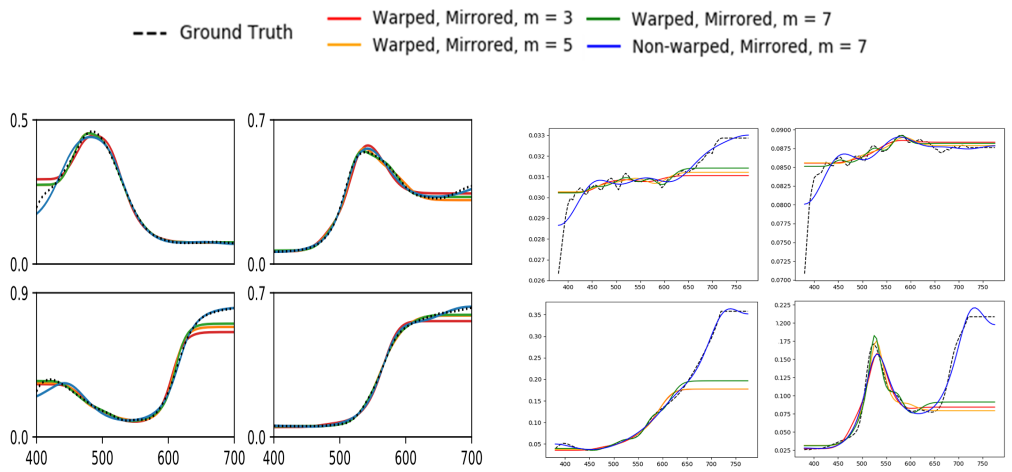
The summary of the reconstruction process is as follows:

1. compute exponential moments from the trigonometric moments
2. evaluate unbounded MESE for the exponential moments
3. compute bounded MESE by applying duality to the unbounded MESE

The results of the spectral reconstruction itself are impressive, especially when applied to smooth reflectance spectra. We show some examples of this in fig. 2.4a. Even with small number of parameters ($m = 3$), the reconstruction describes the original curve quite accurately. Obviously, increasing the number of moments implies higher accuracy, however, it is not recommended to use over 15 moments, as that is roughly the boundary where the mean error stabilizes and does not improve much from then on. Moreover, it is recommended to always use warp for $m \leq 5$.

This technique can also be used for the storage of emission spectra. As these tend to be more spiky and sharp, a lot more moments is required than for the reflectance spectra. To give an example, even a mirrored approached with $m = 15$ does not produce satisfactory results for some emission spectra and the testing was performed for as many as real 32 moments ($m = 31$). Even then, the approach was unable to reconstruct some details.

In this thesis, we focus on storing reflectance spectra. However, in contrast to the spectra shown in fig. 2.4a, we expect some of our to not be so smooth and to have sharper edges. We show examples of such spectra in fig. 2.4b, where we also attempt to save and reconstruct them in the same manner as shown in fig. 2.4a. However, the results are not as accurate, which implies the need for more moments.



(a) Examples of reconstruction of smooth spectra as provided by Peters et al. [35].

(b) Reconstruction of spectra of the Macbeth chart as plotted by us. Top left: "black" patch; top right "neutral35" patch; bottom left: "dark skin" patch; bottom right: "foliage" patch.

Figure 2.4: Examples of reconstruction with the trigonometric moment method.

The number of coefficients that needs to be used depends on many factors, such as the shape of our spectra, available memory and the accuracy for which we aim. We discuss this thoroughly in ref, where we determine the optimal number of coefficients and the method of storing them for our specific problem.

Interpolation of coefficients. Peters hovoria ze by sa mali dat interpolovat, treba spravit experimenty.

3. Implementation

We approach the problem of spectral uplifting similarly to Jakob and Hanika [20], where an uplifting model is created prior to rendering. Our implementation therefore consists of two parts — *model creation* and its subsequent *utilization* in a rendering software.

For the first part, we extend an already existing uplifting tool, the *Borgtool*, which is currently used for creating sigmoid-based RGB cubes as in algorithm 1. We add the possibility for creating trigonometric moment-based cubes, i.e. for the spectra to be stored with trigonometric moments rather than sigmoid coefficients. We also add an option for constraining such a cube with a user-specified atlas.

We then show the performance of such a model by integrating it in ART, which, up until now, used only one built-in sigmoid-based cube for all its uplifting processes.

3.1 Uplifting model

The core of this section is the already mentioned Borgtool. It is a stand-alone, non-open source thing that was created by Weta and is .

The output of the Borgtool is an RGB cube structure which contains multiple entries in form of lattice points. Following, we name the main parameters of a single cube entry of the already supported sigmoid-based cube:

- *target RGB* — the actual RGB that the point has in the cube.
- *coefficients* — the sigmoid coefficients used to reconstruct a spectrum so it matches the target RGB.
- *lattice RGB* — the actual RGB that the reconstructed spectrum evaluates to. Ideally, this should match the target RGB.

Along with its entries, the resulting cube structure also stores a few other properties, both *static*, such as the illuminant according to which the RGB cube is uplifted, and *user-adjustable*, such as the cube dimension or the fitting threshold (i.e. the maximum allowed difference between the target and the lattice RGB).

Our trigonometric moment-based cube can be viewed as an extension of the sigmoid cube — in addition to the already existing parameters, we add a user-adjustable `coefficientCount` variable which specifies the number of coefficients that are to be used for most of the entries. Furthermore, for the purposes of atlas constraining, we extend the cube entry with an optional `entryThisIsBasedOn` pointer to an atlas entry. This is where the main difference between our and the sigmoid cube lies — while the sigmoid cube regards all of its points as equal, we distinguish between *atlas lattice points*, i.e. the lattice points that correspond to specific atlas entries; and *regular points*, which do not. We place special emphasis on the atlas lattice points, as we require their spectra to be as precise and close to the original spectra as possible. As a result, we choose to always store such spectra with the maximum available coefficients (currently, 9). Therefore, the `coefficientCount` variable applies to the regular points only. We explain the

reasoning behind this decision and its impact on rendering and overall performance more thoroughly as we continue with this section.

Our uplifting process is also similar to the one already implemented in the Borgtool, which closely follows algorithm 1. Following are the individual steps of the process:

1. *Initialization*
2. *Fitting of starting points*
3. *Cube fitting*
4. *Cube improvement*
5. *Cube storage*

3.1.1 Initialization

This part of the run is responsible for three things:

- parsing of the parameters
- initialization of the cube and its entries with default values
- loading of the required color atlases

The initialization of the cube is pretty straightforward, as all of its properties are either user-defined or set to default (note: the default illuminant is always D65). The number of cube entries is directly proportional to the cube's `dimension` parameter, which specifies the number of entries per one axis. This renders the total number of entries to $dimension^3$. As the lattice points are positioned evenly, their target RGB values are then equivalent to their coordinates in the RGB cube.

The loading of the atlases is a bit more complicated. Firstly, a single color atlas is inputted in a form of a simple .txt file, which contains merely a list of entries in a textual form as shown in fig. 3.1. Therefore, it requires parsing.

Moreover, the spectral data obtained from the atlases cannot be stored in the Borgtool directly, so as to avoid extreme memory requirements arising with large atlases. To solve this problem, we take advantage of the trigonometric moments.

We store the spectral curves of the individual atlas entries by using the Fourier coefficients as described in section 2.2.1. We use the maximum number of available moments (currently 9, see explanation in section 3.1.2), and we mirror but do not warp the signal prior to coefficient computation. We explain the reasoning behind this in section 4.1.1, where we run experiments to decide on the most efficient and precise method for storing coefficients.

3.1.2 Fitting of starting points

In order to uplift the whole cube as described in algorithm 1, we must first fit one or more *starting points* whose coefficients can then be used as prior for the fitting of other lattice points. For these purposes, we utilize the user-specified color atlas.

```
Entry ID:   orange
-------------------------------------------------------------------------------
Description            :  "orange" patch of the Macbeth colour checker
Type                   :  reflectance spectrum
Fluorescence data      :  no
Measurement device     :
Measured by            :
Measurement date       :

Sampling information
--------------------

Type                   :  regular
Start                  :  380.0 nm
Increment              :  5.0 nm
Maximum sample value   :  100.0

ASCII sample data
-----------------

{6.143748,  5.192119,  4.867970,  5.092529,  4.717562,  4.663087,  4.455331,
4.562958,  4.517197,  4.536289,  4.454180,  4.543101,  4.491708, ... }
```

Figure 3.1: A sample entry from the Macbeth Color Checker atlas.

Ideally, we would assign each entry of the color atlas one lattice point and denote these points as starting points. However, as the RGB value of an atlas entry can be virtually any triplet within the (0,1) range, it is most likely that the atlas entries do not evaluate to any specific lattice RGB but rather to an RGB of a point somewhere in the middle of a cube voxel.

One way to solve such a problem would be to create a complete injective mapping between the atlas entries and their closest lattice points. However, although such an approach is correct in theory, in practice, it is prone to fail during the uplifting process. This is due to the core principle of the uplifting processof a single RGB point, which, as mentioned in ref, is a weighted interpolation of either spectra or coefficients of its neighbors in the RGB cube. If we were to assume a single-color image map obtained by spectrally rendering one atlas entry and we were to uplift a pixel of this image map, it would be crucial for its position in the RGB cube to be the closest to a lattice point seeded by the original atlas entry, so as to utilize it during uplifting. Even a slight change in the RGB value of the image map may cause the uplifting to consider different point as primary during interpolation, which could lead to undesired behavior. Additionaly, especially in the cases of a low voxel size (i.e. high cube `dimension` parameter), such a change could even cause the pixel of the image map to be uplifted in a different voxel than the original atlas entry, which might possibly not utilize the atlas entry at all. As the variance in the RGB value of the resulting image map is inevitable due to the stochastic nature of the spectral rendering process, we opt for utilizing all 8 points on the corners of the voxel instead of just the closest point. We assign the coefficients of the atlas entry to each of them. We call this process the *seeding* of the cube, and we classify the seeded points as *atlas lattice points*.

Seeding of the cube as described gives rise to multiple implementation issues. Firstly, as the number of points that are required to be seeded is 8 times larger than the atlas size, the cube dimension that might have been sufficient for seeding only one lattice point per atlas entry may not suffice now. This could be solved by relaxing the requirement of seeding all 8 entries and just seeding the ones that have not yet been seeded. That would, however, cause the atlas entries that are

fitted first to have an advantage over the ones at the end of the atlas, which could even result in the latter entries to not be utilized at all. Additionally, we wish to prioritize the seeding of the primary point of each atlas entry, i.e. the lattice point that is the closest.

Following, we descibe a step-by-step method we use for seeding:

1. For each atlas entry, we iterate over its neighbors and select the closest one, which we term as its *primary lattice point*.

2. We seed the primary lattice points. If a lattice point is primary for two atlas entries, we seed it with only one of them and issue a warning that advises the user to use a higher cube dimension.

3. For each atlas entry, we iterate over its non-seeded neighbors and seed only the first four, after which we proceed to the next atlas entry. In this way, we attempt to guarantee the representation of each atlas entry in our cube.

4. For each atlas entry, we seed the remaining non-seeded neighbors.

A better approach, without regard to the time complexity, would to seed only one neighbor per atlas entry per iteration, and repeat as many iterations as it would take to seed all the atlas lattice points. An additional improvement would be to seed in the ascending order with regard to the atlas entry's distance to the voxel's corners.

However, the current algorithm solves the issue with non-utilized atlas entries sufficiently. Even if such an error occurs, it is highly likely to be accompanied with the already mentioned primary lattice point error, i.e. two atlas entries having the same primary lattice point, in which case, we issue a warning. This is because such an error evidently points out the insufficient cube size for the given atlas, either due to the atlas's large size or due to its concentration around certain colors (e.g. the Page 14 from the Munsell Book of Colors as in ukazka).

By seeding the cube, we have appointed coefficients to some of the lattice points. These coefficients reconstruct a spectrum that evaluates to an RGB value which we denote as the *lattice RGB*. The difference between the lattice and the target RGB is therefore equal to the distance between the lattice point and its assigned atlas entry in the cube. It is apparent that the distance may be higher than the defined fitting threshold. In such cases, we must "improve" upon the coefficients so that the resulting color difference is as low as possible.

Our problem of improving the coefficients satisfies the definition of the *Non-linear Least Squares* problem [14]. Non-linear Least Squares is an unconstrained minimization problem in the following form:

$$\underset{x}{\text{minimize}} \;\; f(x) = \sum_i f_i(x)^2, \tag{3.1}$$

where $x = \{x_0, x_1, x_2, ...\}$ is a parameter block that we are trying to improve (i.e. our coefficients) and f_i are so-called *cost functions*. The definition of cost functions is dependant solely on the current problem. In our case, we primarily require to minimize the difference between the lattice and the target RGB. Our secondary requirement is the shape similarity of the original atlas entry curve and the resulting curve of the atlas lattice point. This gives rise to multiple choices

for cost functions, such as using the difference between curves along with the Delta E error, using one or multiple cost functions for the RGB error etc... After implementing some of them and testing their performance, the results of which we provide in ref, we decide on neglecting the shape similarity requirement and using three cost functions, each specifying the absolute difference in one of the three axes of the cube.

To solve an optimization problem defined in such a way, we use the CERES solver.

CERES solver

As already mentioned in section 2.1, the CERES solver is an open-source library for solving large optimization problems such as our Non-linear Least Squares problem. It consists of two parts — a *modeling API* which provides tools for the construction of optimization problems, allowing us to set parameters such as maximum number of iterations of the optimizer and maximum number of consecutive nonmononotic steps; and a *solver API* that controls the minimization algorithm.

To solve a Non-linear Least Squares problem, the solver requires us to specify only a so-called *residual block*, which is a structure defined by the prior coefficients and the cost functions. During the execution, the solver tries to minimize the values of the cost functions (or *residuals*) in the residual block. The execution is aborted and the current best parameter block returned when the solver achieves either the specified number of iterations or nonmonotonic steps. For more information on the specifics of the CERES solver, we refer the interested reader to its documentation by Agarwal et al. [2].

There are two main downsides to using the CERES solver. Firstly, the maximum allowed size for a parameter block when using a numeric cost function is 9. This means that we are not able to use more than 9 coefficients for storing a spectrum. This issue could be resolved by adding more residual blocks and combining their results. However, as the runtime of both fitting and rendering with 9 coefficients is already substantially high, we decide not to add such an option as it would most likely be unused.

Another, greater issue is the possibility of CERES getting stuck in local minima and therefore produce unsatisfactory results. This may happen due to the following reasons:

- the dimension of the cube is too low, i.e. the prior coefficients are extremely distinct from the ideal coefficients, or
- the number of coefficients is too high, i.e. the optimizer fails in improving them as a whole

In such cases, the optimizer is not capable of leaving the local minima on its own. We therefore apply a simple heuristics, which consists of only slightly altering the first coefficient (or the first and the second) and running the optimizer again. We chose to alter the first coefficients because they influence the shape of the curve the most.

The need for a heuristic suggests considerable difference between the prior and the resulting coefficients, which implies distinct spectral curves. Such a behavior

is undesired, as it may result in strong metameric artifacts. Fortunately, this heuristic is rarely triggered. We examine this in ref, where we analyze the success rate of the fitting and also demonstrate the curve differences by showing both the spectral curves of the atlas entries and of the fitted lattice points.

The sigmoid-based method approaches the starting points problem by selecting the point in the center of the cube and initializing its coefficients to zero. As these values are extremely close to the real values of coefficients, the optimizer does not have a problem with the fitting.

We inspire ourselves by this approach and provide an option for starting in the middle as well. This eliminates the obligation of the user to specify an atlas, i.e. if no atlas is specified, the cube is fitted from the center. However, providing such an option requires us to define a set of prior coefficients for the center point. We determine it by iterating over multiple existing color atlases and searching for spectral curves that roughly evaluate to an RGB of $(0.5, 0.5, 0.5)$. The coefficients of such curves are roughly $\{0.5, 0, 0, 0, ...\}$, i.e. all zeroes except for the first coefficient. We use these prior coefficients for all available moments and cube dimensions.

3.1.3 Cube fitting

Once the starting points are successfully fitted, we can use their coefficients as prior for other lattice points. We proceed similarly to the approach in algorithm 1, where the lattice points are fitted in multiple *fitting rounds*, each round attempting to fit the neighbors of the already fitted points. We provide a more detailed description of the principle behind our fitting algorithm in algorithm 2.

Similarly to the cube structure, our algorithm also extends the implementation provided in the sigmoid fitting. Two main features are added — conversion of coefficient representation (i.e. *coefficient recalculation*) and an improvement heuristic.

Coefficient recalculation Up until now, all the uplifting techniques mentioned used the same number of coefficients per lattice point. The reasons for this were both consistency of the representation and the possibility of coefficient interpolation. The latter is especially useful in rendering, as interpolation of coefficients instead of whole spectra substantially increases performance.

We, however, chose to use a different number of coefficients for atlas lattice points and regular points. Although this might sound counter-intuitive, we defend our decision by regarding the performance of the optimizer.

As the time execution of the spectral reconstruction is dependant on the number of coefficients, it is evident that by increasing the number of coefficients, we increase the time requirements of the optimizer. Additonaly, having more coefficients implies more possibilities for the optimizer, which subsequently suggests the need for more iterations. This decreases performance even further.

This does not pose a significant problem when fitting atlas lattice points only, as they usually make up only a small portion of the cube's entries. However, if we were to fit the whole cube with the maximum number of coefficients, we would find a noticeable performance decrease as opposed to fitting with, say, 3 coefficients.

Algorithm 2 Fitting of the cube from starting points

1: $n \leftarrow$ user-defined number of coefficients
2: $fittingRound \leftarrow 0$
3: $unfittedPoints \leftarrow$ a list of all points in $RGBCube \setminus startingPoints$
4: **for all** $point \in unfittedPoints$ **do**
5: $\quad point.fittingDistance = MAX\_DOUBLE$
6: **while** $unfittedPoints$ is not empty **do**
7: $\quad currRoundPts \leftarrow$ points from $unfittedPoints$ that have at least one fitted neighbor
8: $\quad$ **for all** $point \in currRoundPts$ **do**
9: $\quad\quad$ **for all** $fittedNeigbor \in point.neighbors$ **do**
10: $\quad\quad\quad point.coefs \leftarrow fittedNeighbor.coefs$
11: $\quad\quad\quad$ **if** $fittedNeighbor \in atlasLatticePoint$ **then**
12: $\quad\quad\quad\quad spectrum \leftarrow$ reconstruct spectrum from $fittedNeighbor.coefs$
13: $\quad\quad\quad\quad fittedNeighbor.coefs \leftarrow$ save $spectrum$ with n coefficients
14: $\quad\quad\quad currDistance \leftarrow$ CERES.Solve($point.coefs$, $costFunctions$)
15: $\quad\quad\quad$ **if** $currDistance \leq point.fittingDistance$ **then**
16: $\quad\quad\quad\quad point.fittingDistance \leftarrow currDistance$
17: $\quad\quad\quad\quad point.coefs \leftarrow$ coefficients from solver
18: $\quad\quad\quad$ **if** $currDistance \leq fittingThreshold$ **then**
19: $\quad\quad\quad\quad$ break
20: $\quad\quad$ **while** $point.fittingDistance > fittingThreshold$ **do**
21: $\quad\quad\quad$ use heuristics to improve upon the current coefficients
22: $\quad\quad\quad$ run the solver again
23: $\quad\quad\quad$ repeat steps $15 - 19$
24: $\quad\quad\quad$ **if** too many iterations of the while cycle have been performed **then**
25: $\quad\quad\quad\quad$ break
26: $\quad\quad$ **if** $point.fittingDistance > fittingThreshold$ **then**
27: $\quad\quad\quad$ remove $point$ from $unfittedPoints$
28: $\quad\quad$ **if** $point$ has tried the coefficients of all of its neighbors **then**
29: $\quad\quad\quad$ remove $point$ from $unfittedPoints$
30: $\quad fittingRound \leftarrow fittingRound + 1$

If we therefore wish to preserve representation consistency, we need to either accept the high time execution or give up the precision with which we fit atlas entries. The latter is not a realistic option, as the precise atlas fitting is the ultimate goal of this thesis. The increased time execution is similarly unfeasible, as it takes a lot more time. Furthermore, having more coefficients provides no real benefits to the resulting spectra (see ref). We therefore trade it off and allow the user to insert his own desired number of moments.

The problem we solve in steps 11 through 13 of our algorithm is that arising when using atlas lattice points' coefficients as prior to regular points. By supporting two distinct representations of a cube's entry, we simply cannot use the same 9 coefficients as prior to, for example, a point that only requires 4. Luckily, this is easily solved — we just add a *recomputation* function which reconstructs the spectra of the prior point and subsequently saves it with the new, lower num-

ber of coefficients. Obviously, such a conversion causes significant loss of spectral information. However, as we do not need the regular lattice points' coefficients to evaluate to any specific spectra, this does not need to concern us.

Up until now, we have not mentioned the effect of various numbers of coefficients on the interpolation phase of rendering, specifically on the coefficient interpolation. That is because our implementation, similarly to the sigmoid implementation, does not support this and is rather based on the interpolation of spectra. We explain the reasoning behind this in ref DOROBIT. However, if required, adding support for coefficient interpolation does not pose a problem — we would only need to utilize the already existing recomputation function. We would need to proceed in the opposite manner as before, when we converted 9-coefficient representations to a smaller number of coefficients. Instead, we would need to convert the regular lattice points to their 9-coefficient representation, so as to preserve the spectral precision of the atlas lattice points.

Improvement heuristic To minimize the shortcomings of the optimizer mentioned in section 3.1.2, we add a heuristic-based improvement of the coefficients, implemented in steps 20 through 25. Its implementation consists of solely slightly changing the first coefficient and running the optimizer again for a pre-defined number of times. We do not implement any other improvement, so as to not cause significant performance decrease. This is because we do not necessarily require the points to be fitted in the given round. Even if the fitting fails, it may still be successful in the following rounds, where the coefficients of newly fitted neighbors might be used as prior. Additionaly, after the cube fitting is done, we still run an "improvement" step for the unsuccessful points. We discuss the specifics of this step in section 3.1.4.

The number of the fitting rounds depends both on the cube dimension and on the kind of atlas that has been used. If we choose not to input an atlas, the fitted cube "grows" from the middle, while seeding with an atlas makes the cube "grow" from many places, requiring a lot less round. We show the differences in , where we present the

Obviously, neither option is better in terms of performance, as the number of fitted points must still be the same.

3.1.4 Cube improvement

In extreme cases, such as when using a very low fitting threshold, the fitting of some points may be unsuccessful even after applying heuristic improvements. Usually, this happens for the outermost points of the cube, i.e. either (1,1,1), (0,0,0), or points with at least one of their coordinates set to either maximum or minimum. We therefore carry out the following improvements in the order as listed and terminate if any of them succeed:

Step 1 utilize the same improvement method as mentioned in section 3.1.3, but increase the number of iterations

Step 2 proceed in the same manner as in the previous step, but slightly alter all the coefficients instead of only the first

Step 3 pass the current best coefficients to the optimizer and keep improving until the optimizer is no longer able to

Step 4 if the point's distance to the (1,1,1) point in the cube is lower than it is to the (0,0,0) point, try setting the coefficients to $\{1, 0, 0, 0, \ldots\}$, as these are the coefficients for a constant curve obtaining the value 1 at every wavelength. Apply both Step 1 and Step 2 to these coefficients.

Step 5 if, on the other hand, the point is closer to the (0,0,0) point, use the $\{0, 0, 0, \ldots\}$ coefficients as prior and proceed as in Step 4

We call this process *cube improvement*. We once again emphasize that it is aimed only at the most extreme cases. As a matter of fact, there is a rather low chance of triggering it at all, and, even if it is triggered, the number of failed points to be improved is usually only one or two. Therefore, we do not concern ourselves with the performance of this process, as its runtime is negligible in comparison to the cube fitting.

We present the exact statistics on cube improvement in ref.

The issue that could arise with this process is that of the following interpolation. By using

The more important issue is the shape of the resulting curve. Using seemingly "random" coefficients as prior suggests the resulting curve's shape to be unlike those of its neighbors, which may result in metameric artifacts.

The reason behind using improvement heuristics both during and after fitting is both for improved time performance, i.e. a simple change of coefficients is less time-consuming that the process t

multithreaded

We support from 2-8 moments then, however, we will talk about the number of coefficients (although user inserts moments). From now on we talk about coefficients. We do not support 1 or 2 coefs as they only create rovne ciary which does not reconstruct the color we want.

3.1.5 Cube storage

Once the cube is completely fitted, its contents are written to a binary file. As we want the resulting file to be as small as possible, we save only the information crucial for the purposes of rendering. Following, we provide a list of contents of a cube file:

- *version*
- *features*, i.e. whether the cube contains debug information
- *moment flag* — a flag signifying that the cube is based on trigonometric moments. We extend the sigmoid cube structure with a sigmoid flag for an easier cube recognition in rendering software.
- *dimension*
- *illuminant* under which the cube has been fitted
- *fitting threshold*

- for every point, we store:
  - *coefficient count*, which gives us information on whether the point is an atlas lattice point or a regular point
  - *coefficients*
  - *lattice RGB*
  - *fitting distance*, i.e. the distance between lattice and target RGB, should be below fitting threshold
  - in case the debug information is included, the entry also contains:
    - *target RGB*
    - *treated variable*, i.e. in which round was the point fitted. If the variable is equal to -1, it means the fitting has failed for this point.

If we were to use such a cube for the purposes of rendering, the knowledge of both the entry's treated paramter and its target RGB is unnecessary. The only benefit the treated parameter provides is the capability to issue a warning in case the cube is incomplete. This can, however, also be achieved by spectral reconstruction from the coefficients and the comparison of its RGB to that of the entry's lattice point.

Storing the target RGB variable also provides no irreplaceable advantages. The target RGB of all the lattice points can simply be computed from the cube's dimension parameter, rendering such variables useless.

In addition to the storing of the cube, we also provide a function capable of loading such a cube and initializing its parameters. Borgtool utilizes this function in case the user wants to create a texture from an already existing cube or simply wants to check the correctness of the existing cube's parameters. The code of this function is also used during the integration in a rendering software.

3.2 ART integration

The user can choose the size of the cube etc. Borgtool has the following options, and we implement all of them for our method (except for something) The borgtool already has the sigmoid method implemented as in algorithm daco Uses three coefficients, fits from middle, and therefore results are metameric (show picture) We implement the trigonometric moment method by changing up the code provided by peters We aim for allowing the user to add a color atlas, however we also provide an option when the user fits from the middle When fitted from middle, the optimizer works very similarly to the sigmoid method (show pictures)

Also explain the threshold value - it is really important to set it properly and that it affects performance.

as the interpolation of multiple spiky, non-similar spectra often results in a similarly uneven spectrum susceptible to metameric artifacts. Mention here that we NEED to have smooth spectra and that it is very important for interpolation. We therefore set parameters so that this is possible as this is the key.

4. Results

In this chapter, we talk about the results. We start of by talking about the results achieved with different parameters during implementation and which we decided to use in the end. We then proceed to evaluate

4.1 Implementation parameters

4.1.1 Storing moments

The first thing we analyze is the technique used for storage and subsequent reconstruction of the Fourier coefficients. Specifically, we need to decide how to map wavelengths to a signal from which the coefficients are obtained. As already mentioned in section 2.2.1, we have the choice of both *mirroring* and *warping* the signal, which overall creates four options — using only mirroring, using only warping, using both or using neither, i.e. utilizing the original signal.

To decide which of these options suits our problem the best, we run an experiment in which we compare the original color of a spectral curve with the color of a spectrum obtained by reconstruction from the original curve's coefficients. We use entries from multiple color atlases (such as the Pantone atlas, Munsell Book of Colors and Macbeth Color Checker) and we compute the average and maximum color difference. For these purposes, we use the Delta E error specified in eq. (1.4). Although we state that the Delta 2000 is better, it is not(why? MFO) due to discontinuities when using gradients

In appendix B.1, we provide all results obtained from these experiments. Note that using n moments requires storing $n+1$ values in case mirroring is used (i.e. the moments are real) and $2n+1$ values otherwise (i.e. the moments are complex). As we are interested in the number of $coefficients$ needed for storage (and for passing to the optimizer) rather than the number of moments, we surmise the contents of appendix B.1 in table 4.1, where we present the errors according to the number of coefficients.

According to the observation of table 4.1, it is clearly beneficial to use mirroring. Choosing whether to warp the signal is slightly more complicated — warping definitely performs better if the number of coefficients is below six (as a matter of fact, this is exactly the same conclusion that Peters et al. [36] has arrived to), and on average outperforms the non-warping strategy even when using higher number of coefficients. We, however, require successful round-trips only when fitting atlas lattice points, as the regular points do not have a prior atlas entry that their spectra must approximate. We therefore focus exclusively on the performance of these technique under 9 coefficients, as that is the number we use for fitting such entries.

In fig. 4.1, we compare the techniques on a few patches of the Macbeth Color Chart by performing round-trips with 9 coefficients. The difference between the two methods is as mentioned in Peters et al. [35] — since warping focuses on the more important regions of the spectrum in terms of color perception (i.e. around 550nm), it reconstructs the slight waves in this area quite precisely while neglecting the edges. Non-warping, on the other hand, focuses on the spectra

Coefficients	Methods							
	M&W		M&nW		nM&W		nM&nW	
	Avg	Max	Avg	Max	Avg	Max	Avg	Max
1	35.87	114.04	36.2	113.86	35.92	113.97	36.2	113.86
2	21.3	91.19	26.4	99.44	—	—	—	—
3	1.65	15.09	15.43	68.06	6.54	42.93	13.24	60.23
4	0.86	5.6	9.93	55.67	—	—	—	—
5	0.54	2.98	4.19	23.53	2.13	17.51	3.7	17.5
6	0.34	2.22	1.19	5.68	—	—	—	—
7	0.29	2.17	0.77	2.38	1.13	6.87	0.95	5.3
8	0.28	2.03	0.77	1.86	—	—	—	—
9	0.26	1.95	0.62	1.43	0.97	4.35	0.48	1.96

Table 4.1: The average and maximum *Delta E* error originating from round-trips, i.e. from converting spectra to coefficients c and its subsequent reconstruction from c. M represents mirroring, W warping, and the symbol n stands for their negation.

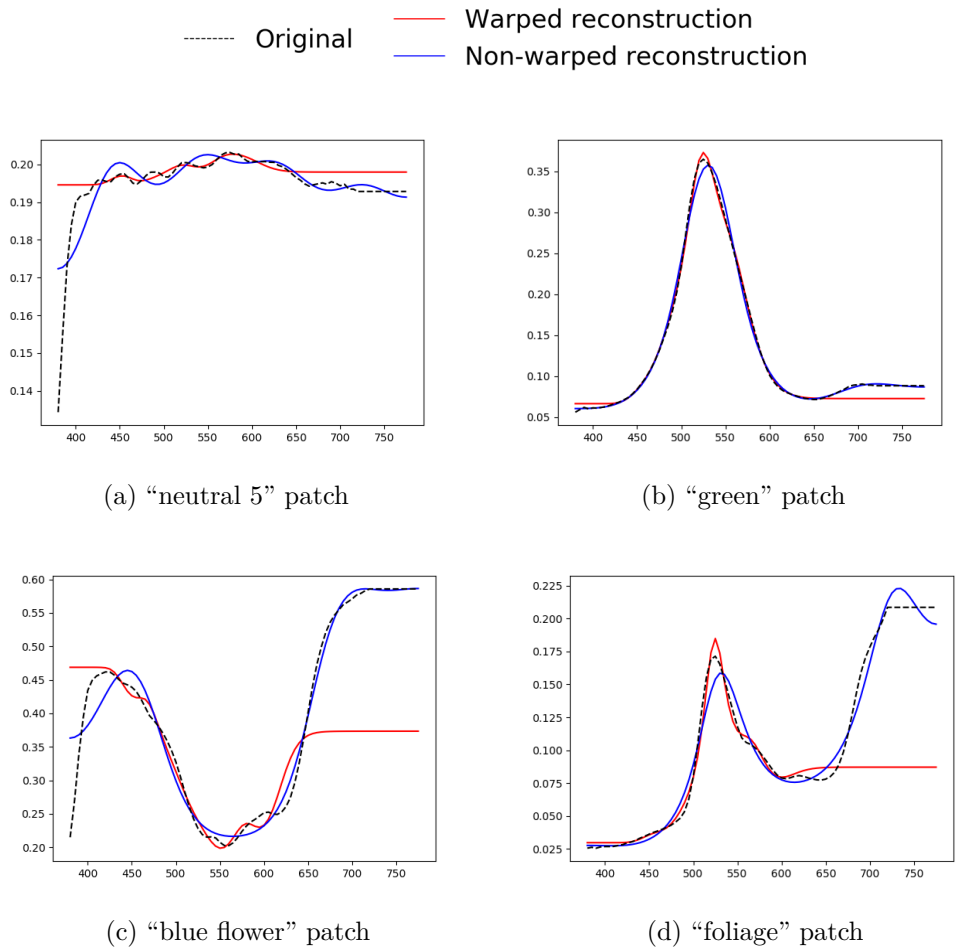
as a whole, which results in approximating the shape over the whole wavelength range but not in an exact replication of any specific spikes.

We cannot definitely determine the more precise method (e.g. warping gives the impression of better performance in case of the "neutral 5" patch in fig. 4.1a, but seems to rather unsuccessful in the case of the "blue flower" patch in fig. 4.1c). However, the last row of table 4.1 suggest that although warping performs better on average, its maximum achieved error is higher than that of the non-warping technique.

The impact of the worst-case scenario on rendering is another important factor to consider in our decision-making process. Having even one incorrect color could cause significant metameric artifacts, which we want to avoid at all costs. Therefore, we specifically analyze the cases in which the maximum error was obtained. For warping, this represents the Munsell 2.5R 2/6 sample of the Munsell Book of Color, while non-warping performs worst for the Pantone 3551 C sample of the Pantone color atlas. We show the round-trip results of both of these cases in fig. 4.2.

Although the shortcomings of warping can already be perceived in e.g. fig. 4.1d or fig. 4.1c, the failure in the reconstruction of the curve's edges does not have a significant effect of the the resulting RGB color, as the source of color is mainly focused around the middle of the curve. However, if the edges are extremely distinct from the rest of the curve (see fig. 4.2a), they tend to provide unanticipated color information. In such cases, warping the signal presents a disadvantage.

Obviously, the Delta E error caused by unnecessary warping can be reduced to almost 0 by passing the computed coefficients to the optimizer, which then alters the curve so that it evaluates to the correct RGB. However, because we lose the notion of the curve's desired shape and because warping does not focus on the edges, the optimizer is apt to amplify the already existing slight bumps in the middle. This behavior may therefore cause the resulting shape to be extremely



(a) “neutral 5” patch

(b) “green” patch

(c) “blue flower” patch

(d) “foliage” patch

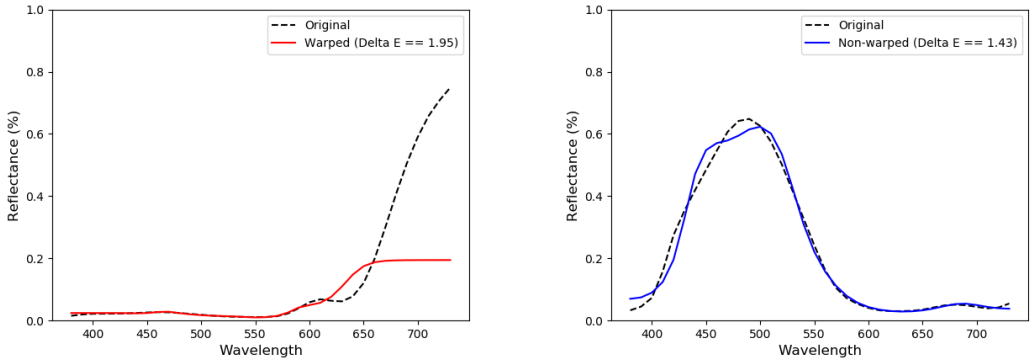
Figure 4.1: Comparison between the warped and non-warped reconstructed signal shown on multiple patches of the Macbeth Color Chart

distinct from the desired one.

The non-warping technique, shown in fig. 4.2b, is not susceptible to this kind of behavior. Although it creates a rather significant Delta E error, we can observe that the shape of the reconstructed spectrum roughly resembles the original shape. Such a behavior is desired in our case, as the reconstructed reflectance is less prone to cause metameric artifacts under different illuminants. Additionally, as the non-warping technique forces the optimizer to not prioritize specific parts of the curve, the optimization is prone to slightly altering the shape as a whole rather than creating irregularities in the middle. Therefore, regardless of the average error, we assume the non-warping technique to outperform warping both when performing simple round-trips, but also if we use this method during the optimization process of fitting the cube.

We put our theory to test. We create a few cubes with varying parameters and we try to fit their entries with both techniques, using the cost functions determined in section 4.1.2. We present some of the results in fig. 4.3, but we examine many more spectra just to confirm our hypothesis.

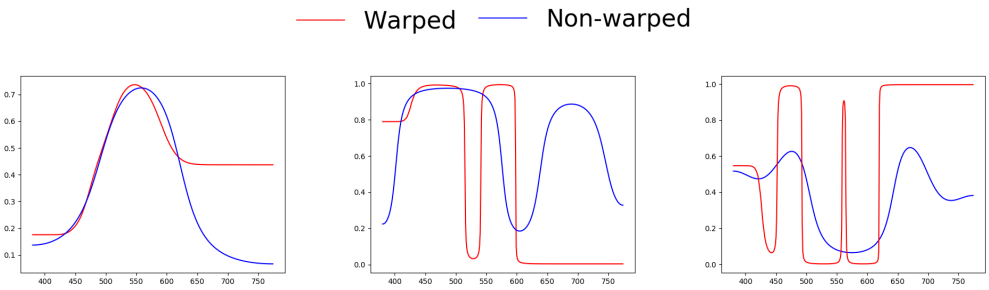
We conclude that our theory is correct — warping indeed amplifies the slight differences around the middle of the curve in order to achieve the correct color, so much that it creates spikier spectra the more the cube grows. Although this



(a) Warped spectral reconstruction of the Munsell 2.5R 2/6 sample from the Munsell Book of Color, Delta E == 1.95

(b) Non-warped spectral reconstruction of the Pantone 3551 C sample of the Pantone color atlas, Delta E == 1.43

Figure 4.2: The worst-case round-trip scenarios of the warped and non-warped technique, 9 coefficients used for spectral reconstruction



(a) $c = 3, cd = 64$, $RGB = (222.6, 230.7, 230.7)$, initial atlas = Macbeth Color Chart

(b) $c = 5, cd = 32$, $RGB = (82.26, 172.74, 255)$, initial atlas = Page 14 from Munsell Book of Color

(c) $c = 7, cd = 16$, $RGB = (102, 17, 153)$, fitted from middle

Figure 4.3: Comparison of warping and non-warping when used for fitting cubes, shown on cube entries from different cubes

does not necessarily render the cubes created with warped signal useless, it is apparent they are prone to creating mematerіc artifacts such as the ones presented in fig. 1.7. Additionally, as we already mentioned in ref, the interpolation phase of the rendering pipeline benefits from smooth, non-spiky spectra, which is a criterion the warped spectra do not satisfy. Therefore, we it is save to discard the option of warping.

Obviously, the ideal solution would be to store the spectra with the method that provides better Delta E error and use non-warping for cube fitting afterwards. However, such an approach is impractical. Firstly, currently, as the first step of the spectral reconstruction is the conversion of wavelength array to a phase signal, using only one method means we can save this signal prior to fitting and reuse it, thus lowering time complexity. Using both methods would require either storing two phase signals, or recomputing them during each reconstruction. Secondly, as we use non-warping for cube fitting anyway, saving only some atlas entries with warping would be both impractical and would not provide too many benefits.

Therefore, we leave the possibility of implementation of the support of both methods as future work.

4.1.2 Cost functions

In addition to the moment storage technique, another thing greatly affecting the performance of the fitting are the cost functions of optimizer. Up until now, Borgtool has used three cost functions, or *residuals*, for fitting with sigmoids, each of them specifying the absolute color difference in one axis of the RGB cube. Such an approach has outperformed both Euclidean color distance and even the Delta E difference — the higher the number of meaningful residuals, the more information about the coefficients' behavior can the optimizer deduce, which, in turn, results in faster and more precise convergence to global optimum.

We therefore copy this approach to specify the color difference. In case of simple cube fitting with the user-specified number of moments (performed in step ref), the results obtained by such cost functions are satisfactory. However, a problem occurs upon fitting with atlas entries.

As already mentioned in ref, the process of seeding the cube with an atlas assigns each atlas entry its closest lattice point (atlas lattice point) in the cube. The distance between the atlas entry and its atlas lattice is dependent on the size of the voxel and, although it can be zero, the chances of that happening for every point of the atlas are slim. This implies that the points' curves cannot be exactly the same — on the contrary, slight differences between them are even necessary.

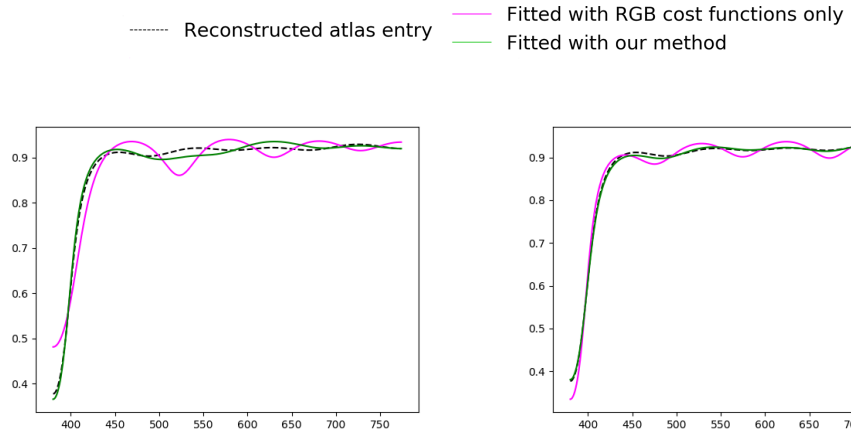
The current cost functions, however, take into account the RGB differences only. They do not try to reconstruct similar shapes in any way, which, in turn, may result in significant shape errors in the resulting spectra. This behavior is not as perceivable in higher-dimensional cubes (e.g. 64), where the starting color difference is so low it does not allow the optimizer to change the reconstructed spectrum a lot before it converges. However, lower-dimensional cubes (e.g. 32) are prone to such behavior. Furthermore, as the specified coefficients are in their nature Fourier coefficients, the optimizer is apt to create spectra in the form of multiple waves with a rather high amplitude, which is only amplified the greater the color difference is.

We show an example of such behavior in fig. 4.4, where we attempt to fit both the "white" and the "blue sky" patches of the Macbeth Color Checker to their main atlas lattice points in both a 32 and a 64-dimensional cube. The magenta plot represents the results of fitting with only the three cost functions, which is definitely undesired.

Note that we compare the optimized spectra not with the original atlas spectra, but with the spectra that is reconstructed from the original's coefficients. That way we can keep track of the optimizer's ability to mimic its input.

When fitting other spectral data, the optimizer does not behave as drastically as shown in fig. 4.4. On the contrary, many fitted altas entries (such as the "blue flower" of the Macbeth Color Checker) resemble their input quite well. However, we must focus on the worst-case scenario as we do not wish to experience any metameric artifacts, not even in one color.

There are two ways of solving the presented problem — either by simply using a higher-dimensional cube, or by adding cost functions that force the optimizer



(a) "white" patch, CD = 32, $d = 5.81$

(b) "white" patch, CD = 64, $d = 1.99$

(c) "blue sky" patch, CD = 32, $d = 4.1$

(d) "blue sky" patch, CD = 64, $d = 0.96$

Figure 4.4: Comparison between the RGB cost functions and our method for fitting main atlas lattice points

to keep the difference between the original and the reconstructed curve as low as possible.

We incline towards using more cost functions, both because the use of a higher-dimensional cube would require greater processing time, but also because the optimizer is prone to creating wave-like spectra nonetheless, even if the waves have much lower amplitude.

Our first idea was to add a fourth residual that would specify the least square error between the curves. Although the results with this method were quite satisfactory, the optimizer was sometimes prone to spikes, as the error was computed over the spectra as a whole rather. We therefore added one residual for each pair of spectral samples, which, in our case (as we sample with an increment of 1) sums up to roughly 400 cost functions. This allows the optimizer to specifically focus on the samples that it wants lowered, which therefore eliminates the possibility of spikes.

Although having this many residuals may seem far-fetched, the optimizer handles it well. It also does not reduce the importance of the three color cost functions. Its only notable issue is that it is prone to declaring optimization failures even if the residuals are satisfactory. This is due to its need to minimize all the residuals so they approach zero. However, we know that the original and

resulting reflectance curves can not be exactly the same.

We solve this problem by adding a simple threshold to the optimizer. In case the least square error between one spectral sample pair is lower, the optimizer assumes it to be zero and is not concerned by it anymore. Initially, we set the threshold to zero, but increase it if the optimizer fails and try again.

The heuristic regarding the threshold does not, on average, need to be invoked more than 2-3 times when fitting an arbitrary atlas to a sufficiently-sized cube (i.e. 64-dimensional). It is clear that by lowering the cube's dimension, the heuristic becomes more utilized. For example, for an 8-dimensional cube, it reaches up to 100 invocations, and even then the fitting may not always be successful.

We, however, do not concern ourselves much with cubes of such low dimension — even for a 32-dimensional cube, the difference between an atlas lattice point and an atlas entry can be as much as $d = 13.86$ (in which d denotes the Euclidean distance), which already suggests a rather significant difference between reflectance spectra and therefore implies loss of atlas information. We therefore strongly recommend the user to avoid cubes of low dimensions (i.e. under 32).

In fig. 4.4, we provide a comparison between the results obtained with our cost functions and the results obtained with the original ones. It is clear that our approach is superior in terms of spectral shape and, as the implementation of the color specifying residuals is identical in both approaches, the resulting RGB values are extremely similar. We therefore use our newly presented residuals for atlas fitting.

Our cost functions give us the ability to control the shape of the resulting spectra. By further utilizing them, we could fit the neighbors of atlas lattice points so that their spectra is extremely similar. Applying this method to the whole cube-fitting process could therefore create an uplifting model with a rather uniform spectra. Such an uplifting model is especially desired for the interpolation phase in rendering.

However, two problems already arise with this approach. Firstly, by growing the cube from multiple atlas entries at the same time, there is bound to be a point in which neighbors are fitted from different prior atlas entries. By our premise, this would mean that the spectra of these two points could be vastly different.

Another issue is the significant increase in time complexity. Even excluding the complex calculations the optimizer must perform during minimization, the computation of around 400 residuals takes a lot more time than just computing the original 3. The process of fitting a 32-dimensional cube, regardless of the number of its moments and the allowed optimizer's threshold, then takes hours instead of mere minutes when executed on an ordinary desktop PC. This renders our cost functions unusable for cube fitting and we must therefore settle for using the RGB cost functions only.

The increased time complexity gives rise to the question of whether it is even feasible to use our cost functions for atlas fitting. It is true that in some cases, for example when seeding with the Munsell Color Atlas, in which the atlas lattice points make up a significant portion of the cube, the time difference is noticeable. However, if we wish to seed the cube with all the atlas entries, we often require a cube of a much higher dimension, the fitting of which diminishes the time importance of the initial atlas fit. Although this may not be true in all cases (e.g. we could create an atlas the size of the cube with each entry mapping to a

different lattice point), it is still beneficial to trade off the possibility of a higher time complexity for much precise spectra.

Therefore, we use the original RGB difference cost functions for fitting regular point and our new cost functions for fitting atlas lattice point.

4.1.3 Number of moments

Maybe add a section about unfittable cubes?

However, correct round-trips are not the only factor we need to take into consideration when fitting the cube. We also need to focus on both the *smoothness of the resulting spectra* and the *behavior of the optimizer* under the current technique.

The smoothness of the spectra is especially important for the interpolation process that takes place during the rendering. Interpolating multiple spiky, non-similar spectra would result in similarly uneven spectra, which, in addition to incorrect color, may be susceptible to extreme metameric artifacts.

The behavior of the optimizer also plays a big role. During the optimization, it takes into account only the resulting RGB of the curve rather than the shape of the curve itself. Therefore, it does not aim for a curve with a similar shape than its neighbor, which may likewise cause issues during the interpolation.

When we fit from the middle, we only aim for the smoothness of the resulting spectra and for the behavior of the optimizer. We therefore want as less coefficients as possible and, as we do not care about round trips, we can use any of the techniques available.

We already said that using 9 coefficients is unecessary, as 3 already create rather smooth spectra that is way better for interpolation. However, we here find out that it is not only unnecessary but extremely discouraged. We can see in the image how the fitting proceeds. It amplifies the already existing wave-like patterns, exhibiting the same behavior as it did during atlas fitting, which is caused by the cost functions not considering the shape of the resulting spectra. By allowing only three coefficients, we limit the optimizer so it must create smooth spectra and it cannot create crazy shapes. Additionaly, it is better for performance - fiting of a 9-coefficients takes around blabla, while fitting 3 takes blabla. We provide measurements here? of performance

Although we cannot control shape, we can control smoothness. We want smooth spectra both for interpolation purposes and because smooth spectra is less prone to metameric artifacts, see chapter 1.

The behavior we talked about before of the optimizer is a problem It would be extremely beneficial for the

For middle fitting, we always recommend 2 moments, 3 coeffs due to the runtime. Moreover, they are very smooth but not straight which is ideal for our purposes. Obviously, we can fit with higher but that takes a lot of time and does not provide any advantage. The default setting is therefore 3 coefs, 2 moments.

Obviously, fitting with these causes metameric artifacts, similar to the ones created by sigmoid, we show these in PICTURE. We can also show metameric artifacts when fitting with more? (try this maybe)

Another slight trick is to use a lower number of coefficients - we therefore do not get as big artifacts because we cannot possibly reconstruct such crazy spectra

with low number of coefficients.

The optimizer is always faster for lower number of coefficients as it does not need to change them up as much. Also faster but does not change them so much SO it results in spectrum that is similar to the first one. However, we cannot possibly simulate the curve with only the limited number of coefficients, the table says so. Therefore we must find something in the middle. We try not to focus on performance here because obviously, fitting anything that is higher than 32 takes hours and we want to have it look the best way we can.

4.2 Performace

heuristics performance, overall runtime of the cube

4.3 Rendering

- which technique gives the best results (metamerism results)

Also mention that it is multi-threaded and performance is not really a priority - the cube has to be created only once and then can be reused as much as the artists need.

Conclusion

Bibliography

[1] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

[2] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. 2012.

[3] Anikethan Bekal, Ajit M Hebbale, and MS Srinath. Review on material processing through microwave energy. In *IOP Conference Series: Materials Science and Engineering*, 2018.

[4] Laurent Belcour and Pascal Barla. A practical extension to microfacet theory for the modeling of varying iridescence. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017.

[5] Arthur D Broadbent. A critical review of the development of the cie1931 rgb color-matching functions. *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the Study of Color, The Swedish Colour Centre Foundation, Colour Society of Australia, Centre Français de la Couleur*, 29(4):267–272, 2004.

[6] John Parker Burg. Maximum entropy spectral analysis. *Astronomy and Astrophysics Supplement*, 15:383, 1974.

[7] Kyungah Choi, Jeongmin Lee, and Hyeon-Jeong Suk. Context-based presets for lighting setup in residential space. *Applied Ergonomics*, 52:222–231, 01 2016. doi: 10.1016/j.apergo.2015.07.023.

[8] Asim Kumar Roy Choudhury. *Principles of colour and appearance measurement: Object appearance, colour perception and instrumental measurement*. Elsevier, 2014.

[9] CIE. Commission internationale de l'éclairage, 1913. URL `http://cie.co.at/`.

[10] D Drosdoff and A Widom. Snell's law from an elementary particle viewpoint. *American journal of physics*, 73(10):973–975, 2005.

[11] Hugh S Fairman, Michael H Brill, and Henry Hemmendinger. How the cie 1931 color-matching functions were derived from wright-guild data. *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the Study of Color, The Swedish Colour Centre Foundation, Colour Society of Australia, Centre Français de la Couleur*, 22(1):11–23, 1997.

[12] Lori Gardi. Planck's constant and the nature of light, 05 2018.

[13] TM Goodman. International standards for colour. In *Colour Design*, pages 177–218. Elsevier, 2012.

[14] Igor Griva, S Nash, and A Sofer. Nonlinear least squares data fitting. *Linear and Nonlinear Optimization*, pages 743–758, 2009.

[15] George G Guilbault. *Practical fluorescence*. CRC Press, 2020.

[16] Martin Habekost. Which color differencing equation should be used. *International Circular of Graphic Education and Research*, 6:20–33, 2013.

[17] CIE HunterLab. L* a* b* color scale. *Applications note, Virginia, USA*, 1996.

[18] Noor A Ibraheem, Mokhtar M Hasan, Rafiqul Z Khan, and Pramod K Mishra. Understanding color models: a review. *ARPN Journal of science and technology*, 2(3):265–275, 2012.

[19] Wenzel Jakob and Johannes Hanika. A low-dimensional function space for efficient spectral upsampling. In *Computer Graphics Forum*, volume 38, pages 147–155. Wiley Online Library, 2019.

[20] Wenzel Jakob and Johannes Hanika. A low-dimensional function space for efficient spectral upsampling. In *Computer Graphics Forum*, volume 38, pages 147–155. Wiley Online Library, 2019.

[21] Alisa Jung, Alexander Wilkie, Johannes Hanika, Wenzel Jakob, and Carsten Dachsbacher. Wide gamut spectral upsampling with fluorescence. In *Computer Graphics Forum*, volume 38, pages 87–96. Wiley Online Library, 2019.

[22] Alisa Jung, Alexander Wilkie, Johannes Hanika, Wenzel Jakob, and Carsten Dachsbacher. Wide gamut spectral upsampling with fluorescence. In *Computer Graphics Forum*, volume 38, pages 87–96. Wiley Online Library, 2019.

[23] Douglas A Kerr. The cie xyz and xyy color spaces. *Colorimetry*, 1(1):1–16, 2010.

[24] David H Krantz. Color measurement and color theory: Ii. opponent-colors theory. *Journal of Mathematical Psychology*, 12(3):304–327, 1975.

[25] Kreĭ. *The Markov moment problem and extremal problems*.

[26] Henry J Landau. Maximum entropy and the moment problem. *Bulletin of the American Mathematical Society*, 16(1):47–77, 1987.

[27] David L MacAdam. The theory of the maximum visual efficiency of colored materials. *JOSA*, 25(8):249–252, 1935.

[28] André Markoff. Nouvelles applications des fractions continues. *Mathematische Annalen*, 47(4):579–597, 1896.

[29] Manuel Melgosa. Testing cielab-based color-difference formulas. *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the Study of Color, The Swedish Colour Centre Foundation, Colour Society of Australia, Centre Français de la Couleur*, 25(1):49–55, 2000.

[30] Johannes Meng, Florian Simon, Johannes Hanika, and Carsten Dachsbacher. Physically meaningful rendering using tristimulus colours. In *Computer Graphics Forum*, volume 34, pages 31–40. Wiley Online Library, 2015.

[31] Michal Mojzík. Fluorescence computations in a hero wavelength renderer. 2018.

[32] Computer Graphics Group of Charles University in Prague. Art, . URL `https://cgg.mff.cuni.cz/ART/gallery/`.

[33] Computer Graphics Group of Charles University in Prague. Art sigmoids, . URL `https://cgg.mff.cuni.cz/ART/archivers/art_2_0_3.html`.

[34] Hisanari Otsu, Masafumi Yamamoto, and Toshiya Hachisuka. Reproducing spectral reflectances from tristimulus colours. In *Computer Graphics Forum*, volume 37, pages 370–381. Wiley Online Library, 2018.

[35] Christoph Peters, Sebastian Merzbach, Johannes Hanika, and Carsten Dachsbacher. Using moments to represent bounded signals for spectral rendering. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019.

[36] Christoph Peters, Sebastian Merzbach, Johannes Hanika, and Carsten Dachsbacher. Spectral rendering with the bounded mese and srgb data. In *Workshop on Material Appearance Modeling*, volume 2019, pages 07–09, 2019.

[37] Dale Purves, G Augustine, D Fitzpatrick, L Katz, A LaMantia, J McNamara, and S Williams. Neuroscience 2nd edition. sunderland (ma) sinauer associates, 2001.

[38] Javier Romero, E. Valero, Javier Hernández-Andrés, and Juan Nieves. Color-signal filtering in the fourier-frequency domain. *Journal of the Optical Society of America. A, Optics, image science, and vision*, 20:1714–24, 10 2003. doi: 10.1364/JOSAA.20.001714.

[39] Iman Sadeghi and HW Jensen. A physically based anisotropic iridescence model for rendering morpho butterflies photo-realistically. *Proc. of Iridescence: More Than Meets the Eye (Tempe, Arizona, 2008)*, page 38, 2008.

[40] Gaurav Sharma and Carlos Eduardo Rodríguez-Pardo. The dark side of cielab. In *Color Imaging XVII: Displaying, Processing, Hardcopy, and Applications*, volume 8292, page 82920D. International Society for Optics and Photonics, 2012.

[41] Gaurav Sharma, Wencheng Wu, Edul N Dalal, and Mehmet U Celik. Mathematical discontinuities in ciede2000 color difference computations. In *Color and Imaging Conference*, volume 2004, pages 334–339. Society for Imaging Science and Technology, 2004.

[42] Brian Smits. An rgb-to-spectrum conversion for reflectances. *Journal of Graphics Tools*, 4(4):11–22, 1999.

[43] Andrew Stockman and Lindsay T Sharpe. Cone spectral sensitivities and color matching. *Color vision: From genes to perception*, pages 53–88, 1999.

[44] Yinlong Sun. Rendering biological iridescences with rgb-based renderers. *ACM Transactions on Graphics (TOG)*, 25(1):100–129, 2006.

[45] Yinlong Sun, F David Fracchia, and Mark S Drew. Rendering light dispersion with a composite spectral model. *Diamond*, 2(37.17):0–044, 2000.

[46] Arthur Robert Weeks, Carlos E Felix, and Harley R Myler. Edge detection of color images using the hsl color space. In *Nonlinear Image Processing VI*, volume 2424, pages 291–301. International Society for Optics and Photonics, 1995.

[47] Guillaume Loubet Sébastien Speierer Benoît Ruiz Delio Vicini Wenzel Jakob, Merlin Nimier-David and Tizian Zeltner. Mitsuba2. URL `https://mitsuba2.readthedocs.io/en/latest/src/getting_started/variants.html`.

[48] John Werner. Human colour vision: 1. colour mixture and retino-geniculate processing. 10 2001. doi: 10.1142/9789812811899_0003.

[49] Sebastian Werner, Zdravko Velinov, Wenzel Jakob, and Matthias B Hullin. Scratch iridescence: Wave-optical rendering of diffractive surface structure. *ACM Transactions on Graphics (TOG)*, 36(6):1–14, 2017.

[50] N Whetzel. Measuring color using hunter l, a, b versus cie 1976 l* a* b*. *Application notes). Retrieved from Hunterlab website: https://support. hunterlab. com/hc/enus/articles/204137825-Measuring-Color-using-Hunter-Lab-versus-CIE-1976-Lab-AN-1005b*, 2016.

[51] Alexander Wilkie, Robert F Tobler, and Werner Purgathofer. Raytracing of dispersion effects in transparent materials. 2000.

[52] Alexander Wilkie, Robert F Tobler, and Werner Purgathofer. Combined rendering of polarization and fluorescence effects. In *Rendering Techniques 2001*, pages 197–204. Springer, 2001.

A. Software user guide

B. Attachments

B.1 Delta E error caused by moment sampling

Moments	Methods							
	M&W		M&nonW		nMW		nMnW	
	Avg	Max	Avg	Max	Avg	Max	Avg	Max
0	35.87	114.04	36.2	113.86	35.92	113.97	36.2	113.86
1	21.3	91.19	26.4	99.44	6.54	42.93	13.24	60.23
2	1.65	15.09	15.43	68.06	2.13	17.51	3.7	17.5
3	0.86	5.6	9.93	55.67	1.13	6.87	0.95	5.3
4	0.54	2.98	4.19	23.53	0.97	4.35	0.48	1.96
5	0.34	2.22	1.19	5.68	0.75	3.37	0.31	0.97
6	0.29	2.17	0.77	2.38	0.55	2.46	0.22	0.99
7	0.28	2.03	0.77	1.86	0.51	1.83	0.22	1.0
8	0.26	1.95	0.62	1.43	0.47	1.67	0.21	1.05
9	0.24	1.81	0.4	1.28	0.43	1.68	0.22	1.04
10	0.24	1.75	0.24	0.96	0.42	1.69	0.22	1.02
11	0.23	1.7	0.22	1.11	0.42	1.69	0.21	1.03
12	0.22	1.64	0.24	1.05	0.4	1.53	0.21	1.05
13	0.22	1.6	0.24	1.0	0.38	1.4	0.22	1.04
14	0.22	1.54	0.22	0.99	0.38	1.37	0.22	1.05
15	0.22	1.46	0.21	1.04	0.34	2.4	0.22	1.05