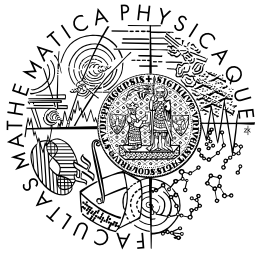


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

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

I would like to express my sincere gratitude to my supervisor doc. Alexander Wilkie, Dr., for his time, guidance and patience over the last year. I would also like to thank my boyfriend, my family and my friends for their constant support.

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 [43], 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. [35] 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 spectral uplifting 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.

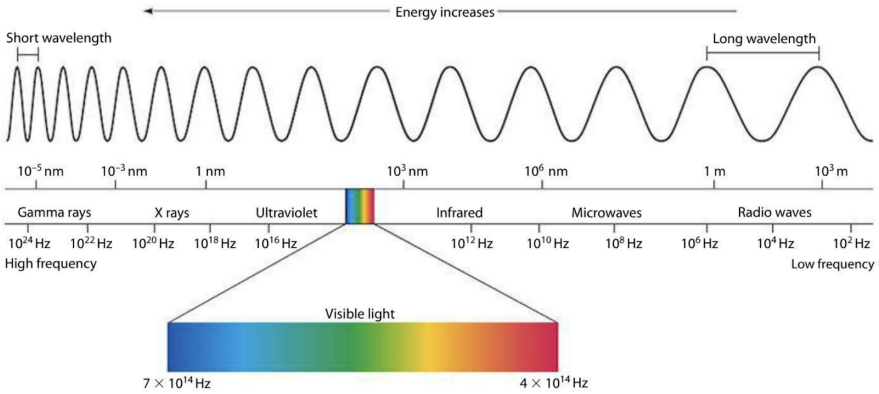


Figure 1.1: An illustration of the electromagnetic spectrum [3]

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. [38], but other sources state that their number could be as high as 125 million [49]). 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 [38] to 6 million [49]). 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

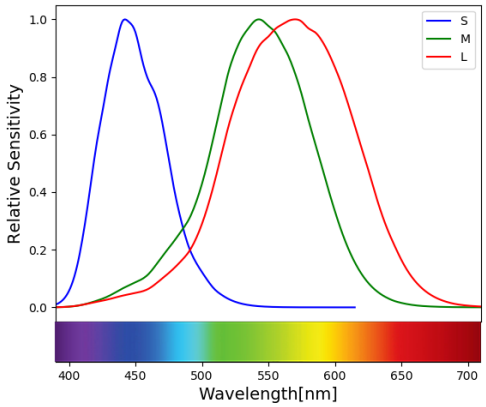


Figure 1.2: Relative sensitivity of S, M and L-cones plotted according to the data measured by Stockman and Sharpe [44].

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

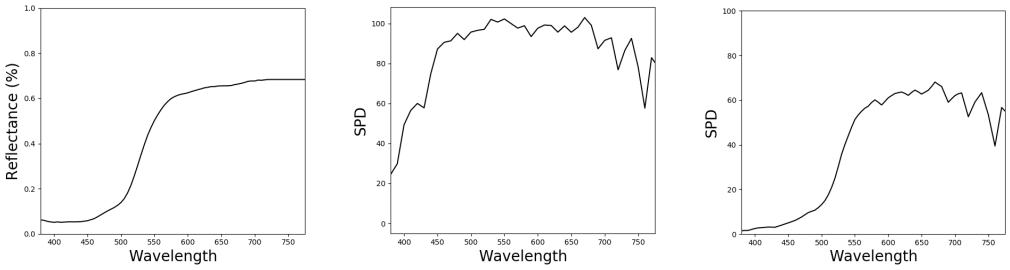
The question of how to discretely represent color has been posed ever since the introduction of the first graphical user interface. For use in computer science, representations are required to be compact, precise, and the operations on colors to be easily executed.

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

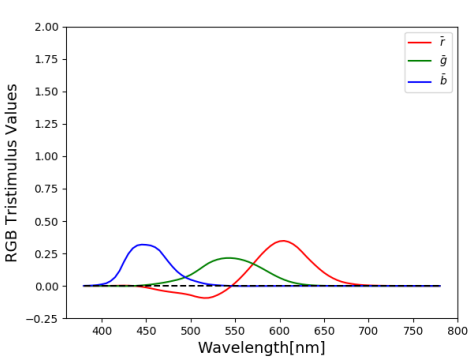
The obvious drawback of spectral representation is the difficulty of its discretization, since there is an infinite number of possible spectral curves, but only a finite number of digital colors that can be displayed by a monitor. For the purposes of color visualization, a distinct, by nature discrete,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

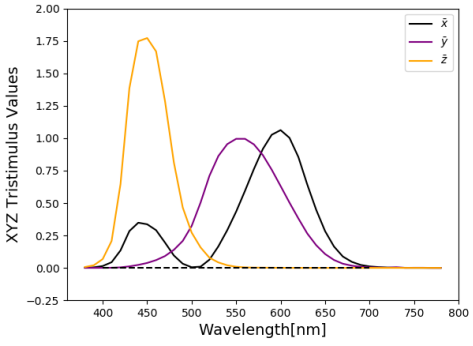
RGB color space

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

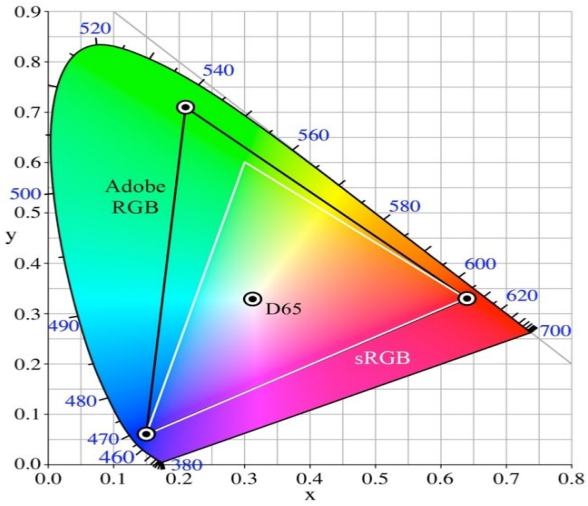


Figure 1.5: An illustrative comparison of the sRGB and Adobe RGB gamut in the chromaticity diagram based on images created by Choi et al. [7]

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 [51] 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 [51].

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 [42]. 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 [41]. 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 [47], 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.

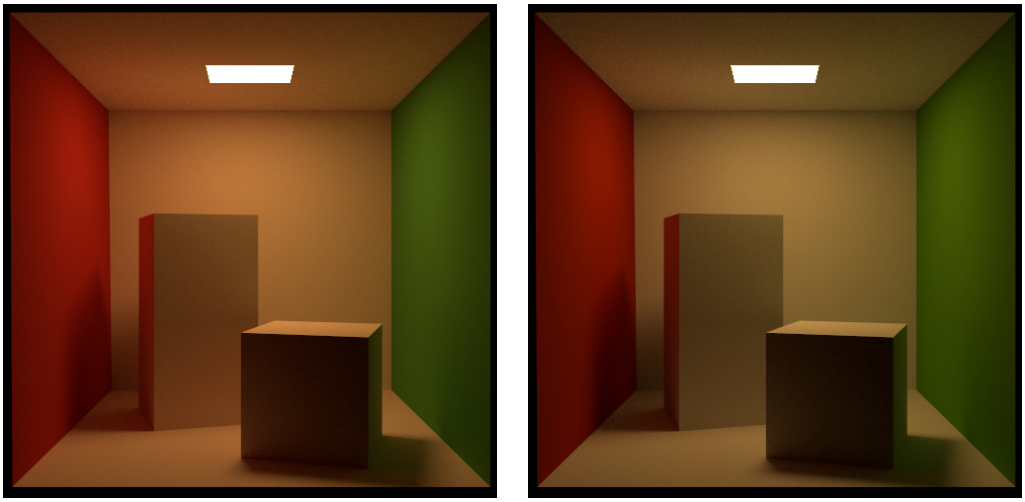


Figure 1.6: Comparison of an RGB-based rendering and spectral-based rendering as presented in the documentation of Mitsuba2 [48]. Left: Spectral reflectance data of all materials is first converted to RGB and the scene is then rendered in the RGB mode, producing an unnaturally saturated image. Right: 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 [48].

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*

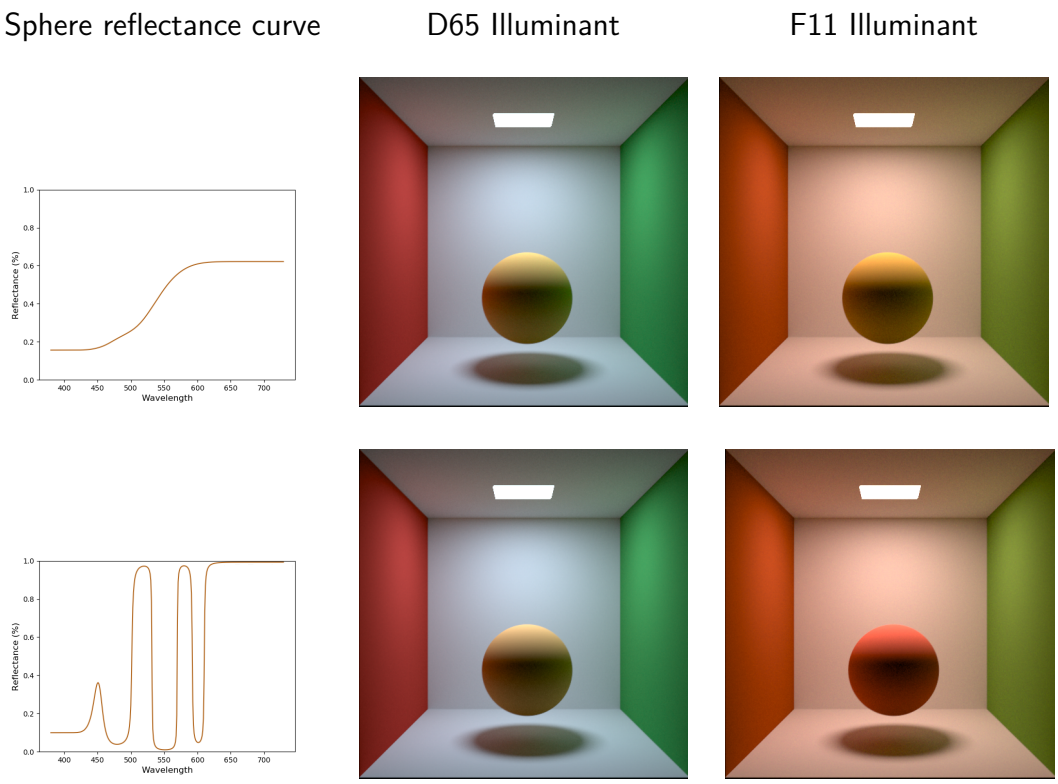


Figure 1.7: The effects of metamerism. Left: Two different spectral reflectance curves, both evaluating to roughly $RGB = (220, 175, 105)$ under D65. Middle and right: Renderings of a sphere with assigned reflectance curves under D65 and F11 illuminants respectively.

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 [53]. 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 [45]. 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 [40], or Werner et al. [50].

- *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* [46] — 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. [46] or Wilkie et al. [52].

- *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) [53]. Therefore, it receives little attention in implementation of rendering softwares. However, for physical consistencies (and due to the possibility of unusual scenes) both ART [33] and Mitsuba [48] 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 in an article by MacAdam [27], which primarily focused on achieving the highest possible brightness for a given color saturation in printing. The uplifting process was only a byproduct of a proof of the limits to the colors' brightness, 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 [43]. 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. [35]. 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 necessarily 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 may 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 article by Jakob and Hanika [20], which discusses a parametric function space for efficient representation of spectral reflectance curves and its subsequent utilization in spectral uplifting. We therefore describe their approach in more detail.

The goal 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 conversion 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 follows:

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

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, similarly to reflectance spectra of real-life surfaces. Another 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. One of the possible solutions is to integrate an iterative optimizing process to find a coefficient triplet that would evaluate to the desired RGB.

In this specific implementation, this process is performed by the CERES solver [1], which 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). As it requires only a few iterations to converge to 0, the conversion of one RGB triplet is rather fast.

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, which divides the sRGB 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. As the RGB value we wish to uplift falls into a specific voxel in the RGB cube, without much elaboration, three straightforward methods of how to uplift it come to mind:

1. weighted interpolation of spectra in voxel corners
2. weighted interpolation of coefficients in voxel corners
3. using the coefficients of the nearest voxel corner (*nearest neighbor* approach)

We present the outcomes of these approaches when used for uplifting an RGB texture in fig. 2.1. We also compare them to the original texture and provide the difference images.

Algorithm 1 Spectral uplifting by Jakob and Hanika [20]

create $RGBCube$ with empty RGB:spectra mappings
$unmappedPoints \leftarrow$ a list of all lattice 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 $unmappedPoints$
while $unmappedPoints$ is not empty **do**
    for all $point \in unmappedPoints$ **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 $unmappedPoints$

Although the original paper suggests that interpolating coefficients should, within limits, produce reasonable spectra without unexpected artifacts, we observe that the interpolation of spectra provides higher round trip accuracy. Therefore, the spectral uplifting tool in ART opts for interpolating spectra even despite the increased time complexity [34]. However, neither of the two approaches produce significant inaccuracies.

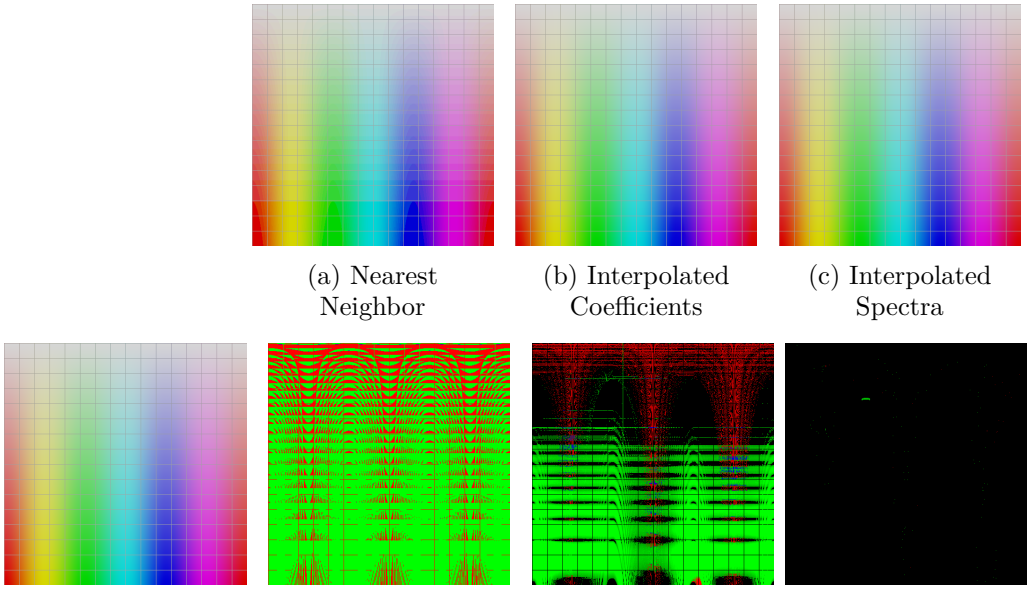
The nearest neighbor approach, on the other hand, is susceptible to visible artifacts, especially in the dark blue region. Therefore, it is not widely used.

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 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 to create smooth spectra with added fluorescent dyes to compensate for the lack of saturation. Similarly to Jakob and Hanika [20], the uplifting model is based on an RGB cube structure optimized by the CERES solver. As expected, the optimization takes longer, as it has 6 coefficients to consider. However, the method is the first one capable of simulating fluorescent



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

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

Figure 2.1: Comparison of techniques for obtaining spectra for non-mapped RGB triplets in the RGB cube as created by Borgtool. The exposure of the difference images (e), f) and h)) is increased to 10 for the errors to be visible.

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 is solved in a different manner than in the previous approach. 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.

2.2 Constrained spectral uplifting

Achieving identity of uplifted spectra to the real-world spectra is, obviously, impossible. However, uplifting many RGB-based assets does not require us to be able to uplift the whole RGB gamut, but only the spectra used for the creation of said assets. 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 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

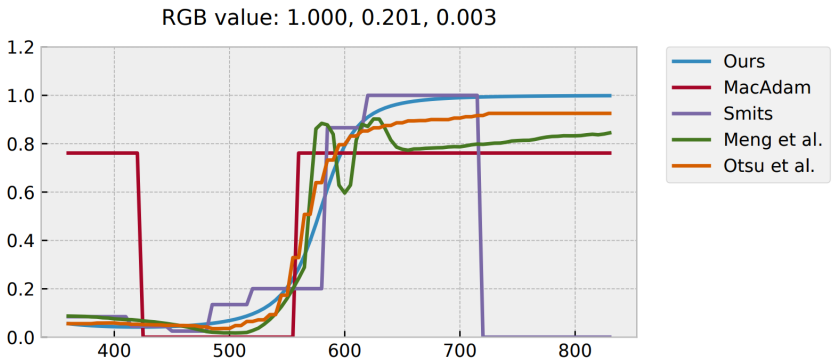


Figure 2.2: Comparison of spectral uplifting techniques as shown by Jakob and Hanika [20]. All spectra were created by uplifting the (1, 0.201, 0.003) RGB value and the results were plotted according to the corresponding techniques. The "Ours" approach, in this case, refers to the approach by Jakob and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

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.

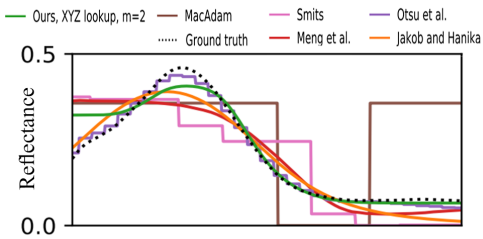
In this thesis, we base the algorithm used to implement constrained spectral uplifting on algorithm 1. We also use an RGB cube as a structure for saving the mappings, and we also create an uplifting model before rendering. We leave the specific details of implementation to ref. In this section, we discuss 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

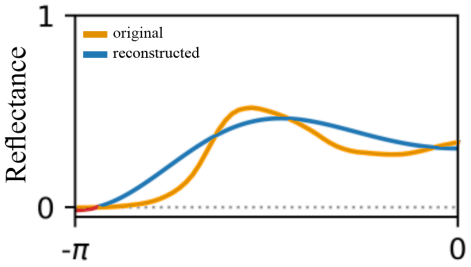
The constraining process starts when the user inserts a set of spectra. Finding an RGB match and creating a mapping is a straightforward task — one must simply convert the spectra to RGB. However, the spectra must then be stored in the structure, which requires its discretization.

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 [37]. 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.



(a) Reconstruction with uplifting models as plotted by Peters et al. [36], where *Ours* represents a newly proposed technique



(b) Reconstruction with truncated Fourier series [37]

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

Although the existing approaches to spectral uplifting, which we have reviewed in section 2.1, propose ways of discretizing some spectral shapes, all of them are restricted to a concrete spectral space and therefore unable to recreate all the possible reflectance curves. 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.

Therefore, discretization of spectra must be approached in a different way.

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 [39], and appear promising also for reflectance spectra. This method is studied in an article and subsequent presentation by Peters et al. [37]. 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 lattice points of 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 observe this drawback even in our example in fig. 2.3b.

In contrast to a linear function space, spectra can also be represented non-linearly. These representations are, however, incompatible with linear prefiltering of textures [36].

To eliminate the shortcomings but utilize the strengths of both linear and non-linear methods, Peters et al. [36] propose a novel approach to spectral representation. The representation is based on the theory of moments (i.e. it is non-linear), but consists of Fourier coefficients, which implies compatibility with linear filtering. Additionally, it 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 [36].

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.

A simple solution would be to linearly map wavelengths to a 2π-periodic signal. Although such an approach can be used (we discuss its performance in ref), it causes distortions and strong artifacts at the boundaries. Moreover, Fourier coefficients computed for such signal are complex instead of real, which requires almost twice the memory for storage.

Therefore, an improvement to the mapping, called *mirroring*, is proposed. It first maps the negative values of the signal as in the following equation:

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

The mapping for the positive part is then defined as the negative part mirrored, i.e. $g(\varphi) = g(-\varphi)$ for all $\varphi \in [0, \pi]$, which results in smooth transitions at the boundaries. The Fourier coefficients are then computed for this mirrored signal only 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.

Another proposed improvement to obtaining the coefficients is by focusing accuracy on important regions of the curve in terms of color reconstruction (i.e. around 550nm), also called *warping*. This is achieved by means of a differentiable, bijective function that maps the wavelength range to the $[-\pi, 0]$ range and is used as a weighting function when computing coefficients. Warping of the signal is useful especially when using only a small number of coefficients, as they are unable to capture more complex curves, and it is highly recommended to always warp the signal if using less than 5 moments.

We explain our choice of the signal mapping method used for the purposes of this thesis in ref, where we also provide results of experiments that affirm our decision.

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 is capable of reconstructing both smooth and spiky spectra, and has been shown to produce impressive results when used for the reconstruction of emission 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*, has been 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:

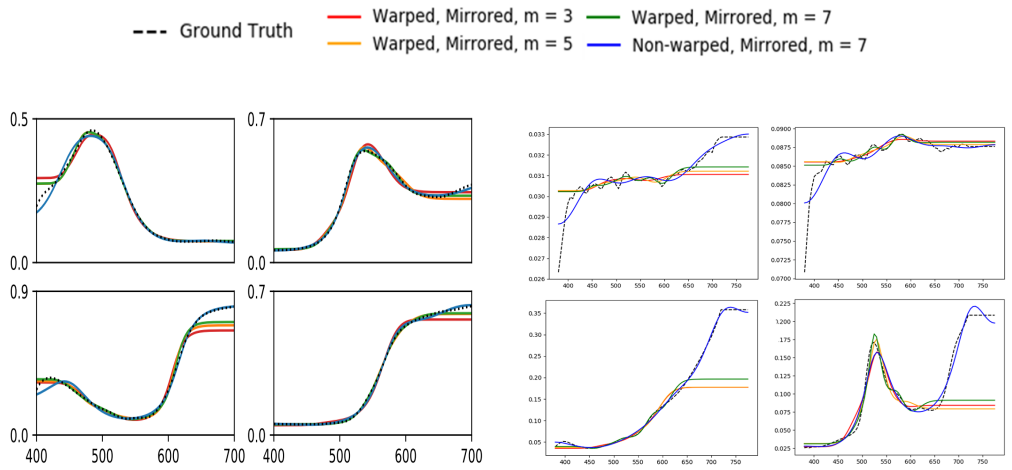
1. compute exponential moments from the trigonometric moments
2. evaluate unbounded MESE for the exponential moments
3. compute bounded MESE by applying duality to the unbounded MESE

We present examples of results obtained by the storage and subsequent reconstruction of reflectance spectra with the trigonometric moment method in fig. 2.4, where we also compare the performance of different number of moments and different signal mapping techniques. For smooth spectra, even a small number of moments($m = 3$) provide a quite accurate reconstruction (see fig. 2.4a). However, more complex spectra with sharper edges and spikes tend to lose detail and their reconstruction is a lot smoother. As seen in fig. 2.4b, especially for the "black" and "neutral35" patches, even 7 moments are not capable of describing the original curve accurately.

Emission spectra perform even worse. To give an example, even a mirrored approach with $m = 31$ is unable to reconstruct the most spiky details [36]. However, as the focus of this thesis is purely on reflectance spectra, we most likely will not encounter a spectrum that would require such high number of moments.

The optimal number of moments for a spectral reflectance curve depends on many factors, such as the shape of the curve, memory limitations and the accuracy for which we aim, either in terms of curve similarity or color error under various illuminants. We discuss these factors more thoroughly as we progress with this thesis, and explain our method for determining the "sufficient moment count" in section 3.1.1.

Due to memory efficiency, smoother gradients, higher precision and multiple other reasons, thoroughly explained in chapter 3, we decide to use variable number of moments per an input reflectance spectrum. Therefore, although Peters et al. [36] state that the interpolation of coefficients provides satisfactory results, our uplifting process cannot rely on this method for uplifting non-mapped RGB values. The only way in which to achieve this would be to reconstruct all the spectra in the corner voxels and subsequently save them with the same number of coefficients, which is not only less time efficient than interpolation of spectra, but could also result in slight discrepancies.



(a) Examples of reconstruction of smooth spectra as provided by Peters et al. [36]

(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

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 in the same manner as in algorithm 1. We add the possibility for creating trigonometric moment-based cubes (from now on referred to as *trigonometric moment cube*), 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 RGB coordinates that the point has in the cube
- *coefficients* — 3 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 cube can be viewed as an extension of the sigmoid cube, as it keeps most of its parameters and mainly extends the already existing ones. The main difference between them lies in the distinction of the lattice points — while the sigmoid cube regards all of its points as equal, the trigonometric moment cube distinguishes (by means of a `seeded` boolean parameter) between *atlas lattice points*, i.e. the lattice points that store the user-inputted RGB:spectra mappings; and *regular points*, which do not.

Our requirements for the shape of the spectra at atlas lattice points differ from the ones at regular lattice points. While we prefer the regular lattice points to have their spectra as smooth as possible in order to avoid unexpected artifacts under other illuminants, the coefficients of the atlas lattice points must reconstruct spectra almost identical to the input spectra, which might include sharp edges and spikes.

Therefore, it is sufficient for the regular lattice points to be represented with a smaller number of coefficients, while atlas lattice points might require a lot more. Although a smooth spectrum can be represented with a high number of coefficients, such a representation is memory inefficient, its reconstruction is more time consuming, and, most importantly, it does not work well with the optimizer. Based on our experiments in section 4.1.2, we decide to store the spectra of regular lattice points with 3 coefficients and adjust the number of coefficients of the atlas lattice points depending on the nature of its desired spectrum.

In addition to supporting variable number of coefficients (ranging from 3 to 24), the trigonometric moment cube also supports the possibility of a multiple coefficient representations per lattice point. The sole purpose of such extension is to lower the cube size requirements upon constraining, which we address more thoroughly in ref.

In addition to the cube structure, the uplifting process of the trigonometric moment cube is also similar to the one of the sigmoid cube, which closely follows algorithm 1. Its main distinctions are in the constraining process (which the sigmoid cube lacks) and in the first round of optimizing, or, as we refer to from now on, *fitting*.

Following, we name the individual steps of the process, which we then analyze in greater detail.

1. *Initialization*
2. *Cube seeding (optional)*
3. *Fitting of starting points*
4. *Cube fitting*
5. *Cube improvement*
6. *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 equivalent to their coordinates in the RGB cube.

```
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

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 with Fourier coefficients as described in section 2.2.1. We mirror but do not warp the signal prior to coefficient computation (see section 4.1.1).

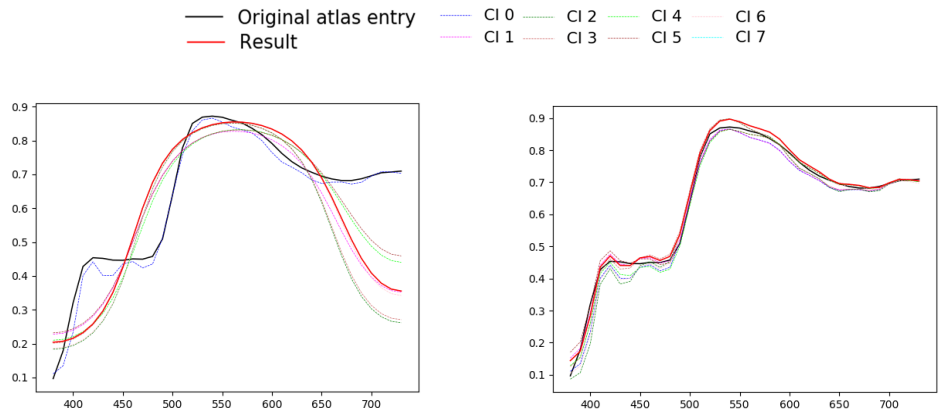
The number of coefficients per atlas entry is variable, ranging from 4 to 24. We explain our method for determining the sufficiency of coefficient representation, and therefore the coefficient count for each atlas entry, in section 4.1.2.

3.1.2 Cube seeding

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. The general idea behind this process is to copy the coefficients of atlas entries to specific lattice points, and then use these coefficients as prior for fitting said lattice points. We refer to this process as *seeding*, and term the seeded lattice points *atlas lattice points*.

The ideal scenario would be if the RGB of the atlas entries were to perfectly match the coordinates of lattice 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 evaluate to a point inside the cube voxel.

A realistic approach would be to create a complete injective mapping between the atlas entries and their closest lattice points. However, such a mapping would provide satisfactory results only if we were to use the nearest-neighbor method for uplifting non-mapped RGB triplets. If we were to use either the interpolation of coefficients or spectra, which both employ all 8 voxel corners during the uplift,



(a) Only 1 voxel corner seeded

(b) All 8 voxel corners seeded

Figure 3.2: Comparison of the performance of two seeding methods

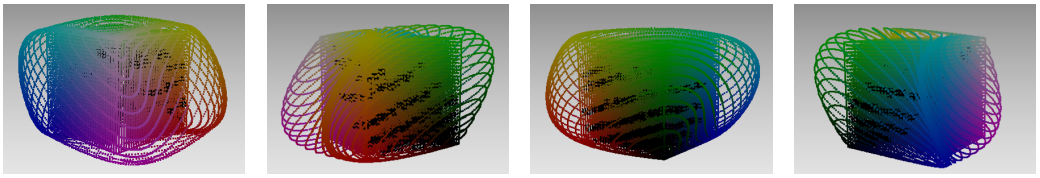


Figure 3.3: A 32-dimensional cube fitted with the Munsell Book of Colour

the curves of uplifted RGB values of atlas entries would most likely be considerably distinct from the desired result (i.e. the original atlas entry spectrum), subsequently causing color artifacts. We can observe this behavior in fig. 3.2a. However, as seen in fig. 3.2b, propagating the information about the original reflectance of the entry to all voxel corners improves the result remarkably. We therefore opt for seeding all 8 voxel corners per an atlas entry.

During the seeding process, it may occur that two atlas entries would fall into neighboring voxels, i.e. that they would share some of the voxel corners. We solve such collisions by utilizing the possibility of one lattice point having multiple coefficient representations. In addition to coefficients and their count, we also store an entry ID per each representation, so as to later distinguish the reconstructed curves during rendering and decide which to employ, which we explain in more detail in ref.

If two atlas entries fall into the same voxel, however, there is no way to determine the interpolation of which the user desires upon uplifting the RGB values inside said voxel. We therefore discard one of the entries and throw an error informing the user of the collision and suggesting to increase of the cube dimension.

We show an example of a cube seeded and subsequently fitted with the Munsell Book of Colour in fig. 3.3. Lattice points marked as black represent the seeded points. Note that a lot of these points store multiple coefficient representations.

Constraining the uplifting process with an atlas is optional. If no atlas is inputted, all lattice points are regarded as regular lattice points and the cube is fitted from the middle in the same manner as the sigmoid cube. However, the resulting uplifting structure provides no advantages over the sigmoid cube.

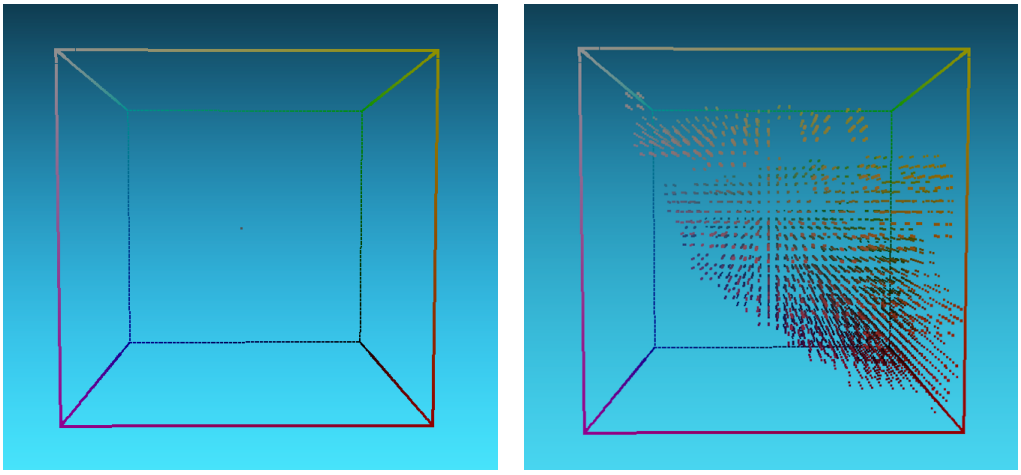


Figure 3.4: Comparison of the placement of starting points when seeding with the Munsell Book of Colour (right) and when fitting from the middle with sigmoids (left)

Supporting this option requires us to specify 3 prior coefficients for the center point, i.e. for a lattice points whose spectral curve roughly evaluates to an RGB of $(0.5, 0.5, 0.5)$. By storing spectral curves that roughly evaluate to such RGB with the trigonometric moments, we observe that the coefficients roughly evaluate to $\{0.5, 0, 0\}$. Therefore, we use them as prior.

We provide a comparison between the starting points when seeding from the middle (either with our or the sigmoid method) and when seeding with an atlas in fig. 3.4, so as to have a rough idea of their placement.

3.1.3 Fitting of starting points

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 dependent 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 for the shapes of the original atlas entry curve and the reflectance curve of the atlas lattice point to be similar. 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 section 4.1.3, we decided on using four cost functions — three for specifying the absolute difference in one of the three axes of the cube, and one for specifying the average distance between curves per wavelength sample. We also include a heuristic which sets the value of the fourth cost function to 0 if the curve distance falls below a certain threshold, and iteratively increases the threshold if the optimizer fails. The reasoning behind this is also explained in section 4.1.3.

To solve an optimization problem defined in the way as described above, we use the CERES solver. We

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 or 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 attempts 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 is one main downside to using the CERES solver. As it was designed to handle very large, sparse problems where every residual term depends on only a few of the input parameters, it is not ideal for solving problems with only one large parameter block, i.e. it might get stuck in local minima and therefore produce unsatisfactory results. Unfortunately, if an atlas lattice point is represented with a high number of coefficients, our optimization problem falls into this category.

We solve such problematic cases by applying a simple heuristic, which consists of slightly altering the first coefficient (as it has the highest influence on the shape of the curve) and running the optimizer again. However, although such an optimization greatly improves the overall performance of fitting, it remains insufficient for too high a number of coefficients, i.e. the threshold for the fourth residual must be increased to values extremely high and, by then, it loses resemblance to the original shape.

Therefore, we implement another heuristic improvement — if the coefficient count is higher than 14, we let the optimizer optimize only the first 4 coefficients while leaving the others constant. We use the threshold of $c > 14$ as that is roughly the boundary where the fitted curves begin to show undesired artifacts, and we optimize the first 4 coefficients because their number is both low enough

Algorithm 2 Fitting of one coefficient representation of a *point* from atlas lattice points

```
1: threshold ← 0.001
2: while threshold < 1 do
3:     coefsToFit ← either the first 4 coefficients or all of them depending on the coefficient count of point
4:     i ← 0
5:     while o dooptimizer is unsuccessful and i < maxIterations
6:         heuristically change coefsToFit[0]
7:         run the optimizer with parameters coefsToFit and threshold set to threshold
8:         i++
9:     if optimizer was successful then
10:        point.coefs ← coefsToFit
11:        break
12:    increase threshold
```

for the optimizer to handle without errors, and high enough so we give the fitting process enough freedom without it having to resort to sinusodial-like shapes.

We summarize the fitting process of the starting points, including the utilization of threshold for our fourth cost function (see section 4.1.3), in algorithm 2. If the optimizer is unable to fit an atlas lattice points, we throw a warning and convert it into a regular lattice point. However, as of yet, we have not encountered a failure.

3.1.4 Cube fitting

As the atlas lattice points are represented with a higher number of coefficients and may even contain multiple coefficient representations, they cannot be directly used as prior for the regular lattice points. First, they must be "converted" into a lower-dimensional representation.

We refer to the conversion process as *coefficient recalculation*. It consists of reconstructing the reflectance spectrum of the atlas lattice point and subsequently saving it with 3 coefficients. Although this process causes significant loss of spectral information, it preserves the rough outline of the curve. This works to our benefit — it reduces the likelihood of significant color artifacts between the atlas lattice points and regular lattice points while keeping the spectra smooth.

If a coefficient recalculation of an atlas lattice point with multiple fits is required, we interpolate the spectra of all of its fits and save the result with 3 coefficients. In terms of the smoothness of color gradients in the cube, this approach is more reliable and stable than if we were to pick only one of its fits and work with it. To support our claim and explain it more thoroughly, we run an experiment comparing these methods.

We seed a cube with two atlas entries with vastly distinct shapes of their reflectance spectra that, however, evaluate to neighboring voxels. This implies that there must exist an atlas lattice point that contains both coefficient representation of the seeded atlas entries. We

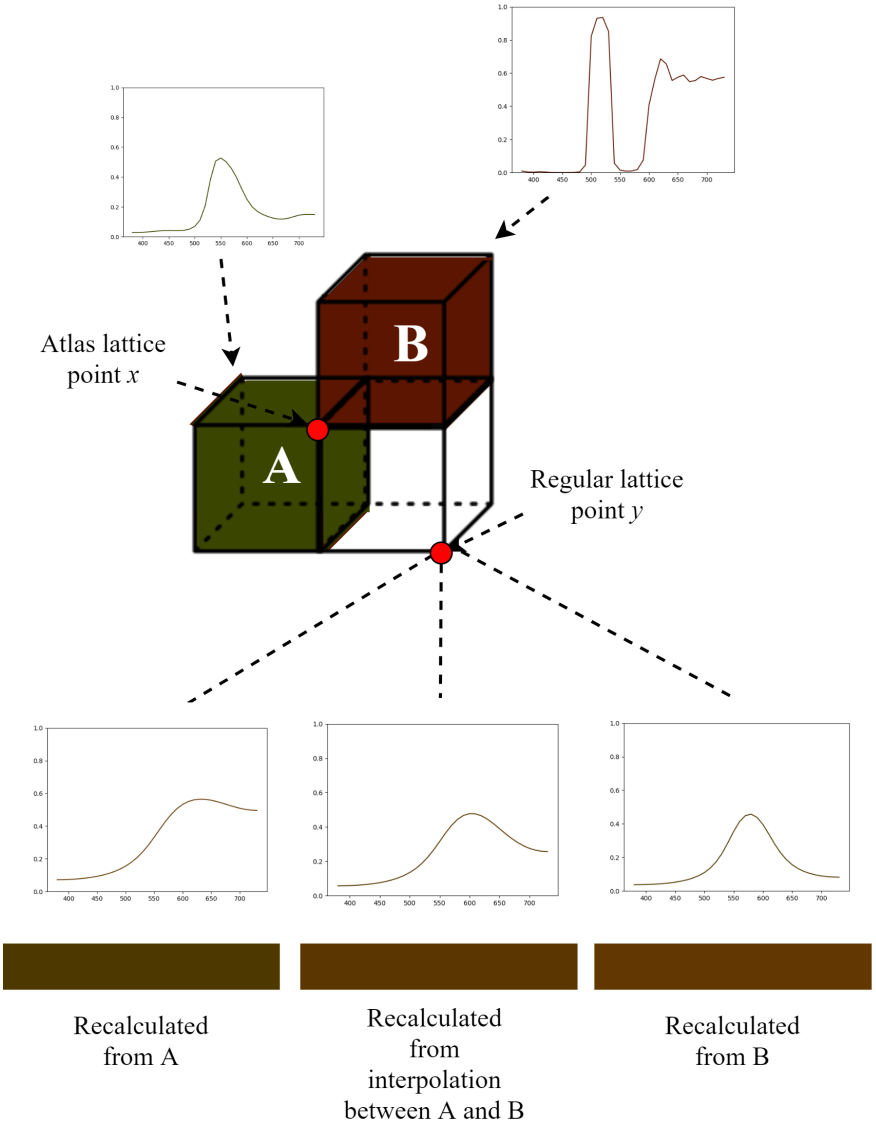
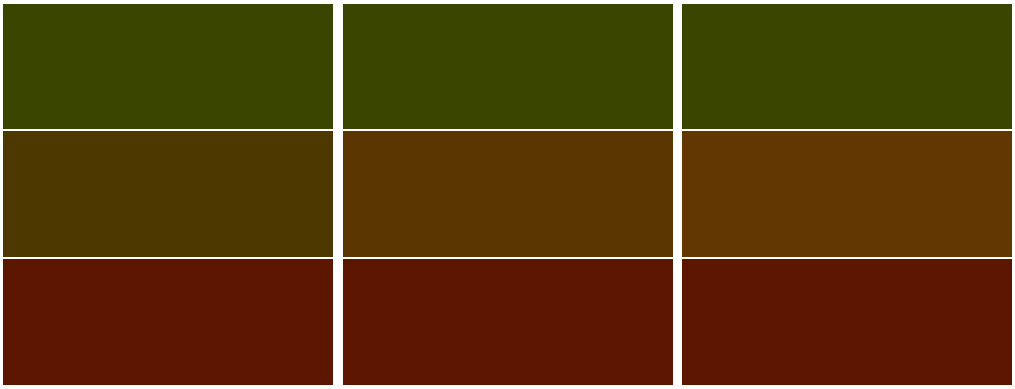


Figure 3.5: Comparison of coefficient recalculation techniques. Left: first fitted entry; right: second fitted entry; middle: result of recalculation

We pick one of the atlas lattice points that contains the coefficient representation of both of them, we pick a regular lattice points that is its neighbor, and we compare the spectra of its prior coefficients for all of these methods — if recalculated from the first representation only (see fig. 3.6a), from the second representation only (see fig. 3.6c), and from their interpolation (see fig. 3.6b). For visualization purposes, we also include the colors these curves evaluate to under the FL11 illuminant. We present the results in fig. 3.6.

If all the fits of an atlas lattice point contribute to the prior coefficients, the color difference between the result and the original fits is relatively similar for each fit. If, on the other hand, the point's prior coefficients come from one fit only, the reduced color difference between the point and the original fit comes at a cost of rather significant artifacts between the other entries.

As we prefer to keep the cube's gradients as smooth as possible, we opt for the interpolation of fits.



(a) Recalculation of only the first entry

(b) Recalculation of the interpolation of curves of both entries

(c) Recalculation of only the second entry

Figure 3.6: Comparison of coefficient recalculation techniques. Left: first fitted entry; right: second fitted entry; middle: result of recalculation

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 3.

(TOTO neviem ci tam pridat? ci to ukazat)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.

3.1.5 Cube improvement

During our implementation of Borgtool, we use the ART

Due to the shortcomings

To minimize the shortcomings of the optimizer mentioned in **??**, we add a heuristic-based improvement of the coefficients, implemented in steps **??** through **??**. 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 3.1.5.

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

Algorithm 3 Fitting of the cube from starting points

$fittingRound \leftarrow 0$
$unfittedPoints \leftarrow$ a list of all points in $RGBCube \setminus startingPoints$
for all $point \in unfittedPoints$ **do**
    $point.fittingDistance = MAX\_DOUBLE$
while $unfittedPoints$ is not empty **do**
    $currRoundPts \leftarrow$ points from $unfittedPoints$ that have at least one fitted neighbor
    for all $point \in currRoundPts$ **do**
        for all $fittedNeigbor \in$ fitted neighbors of $point$ **do**
            if $fittedNeighbor \in atlasLatticePoint$ **then**
                $point.coefs \leftarrow$ recalculateCoefs($fittedNeighbor.coefs$)
            else
                $point.coefs \leftarrow fittedNeighbor.coefs$
            $[sDist, sCoefs] \leftarrow$ CERES.Solve($point.coefs$, $costFunctions$)
            if $sDist \leq point.fittingDistance$ **then**
                $point.fittingDistance \leftarrow sDist$
                $point.coefs \leftarrow sCoefs$
            if $cDist \leq fittingThreshold$ **then**
                $point.treated = true$
                break
        if $point.fittingDistance > fittingThreshold$ **or** $point$ has tried the coefficients of all of its neighbors **then**
            remove $point$ from $unfittedPoints$
    $fittingRound \leftarrow fittingRound + 1$

listed and terminate if any of them succeed:

Step 1 utilize the same improvement method as mentioned in **??**, 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.

3.1.6 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.

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).

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

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, - navyse daco s tymto spravit

4.1 Implementation parameters

4.1.1 Storing moments

The first thing we analyze and decide on is the technique used for mapping wavelengths to a signal for the storage and subsequent reconstruction of moments. 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.

Note that, although we aim for the highest possible precision in terms of curve reconstruction for the atlas lattice points, so as to lose as little information about the original atlas entry as possible, this is not the case for regular lattice points. On the contrary — as they do not have a prior atlas entry that their spectra must approximate, we mainly aim for their smoothness in order to prevent color artifacts upon interpolation.

Additionally, when choosing the correct wavelength mapping technique, we must also take the performance of the optimizer under it into account.

We start by focusing on the accuracy of reconstruction for the atlas lattice points. We run an experiment across multiple color atlases (specifically the Pantone Color System, Munsell Book of Colours and the Macbeth Colour Checker SG) and multiple CIE illuminants in which we compare the original color of spectral curve under an illuminant with the color of a spectrum obtained by reconstruction from the original curve's coefficients under said illuminant. We then compute the average and maximum obtained color difference. For these purposes, we use the simple Delta E error due to its continuous nature.

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

Coefficients	Methods							
	M&W		M&nW		nM&W		nM&nW	
	Avg	Max	Avg	Max	Avg	Max	Avg	Max
1	23.88	130.27	23.95	130.62	23.86	130.38	23.95	130.62
2	13.09	97.76	16.92	107.23	—	—	—	—
3	1.39	21.71	10.18	74.62	4.08	51.23	8.48	67.12
4	0.74	7.43	6.3	60.36	—	—	—	—
5	0.49	5.46	2.58	26.36	1.31	20.87	2.56	20.14
6	0.35	3.95	1.1	6.83	—	—	—	—
7	0.31	3.19	0.73	6.12	0.71	8.18	0.89	6.36
8	0.28	2.85	0.71	5.67	—	—	—	—
9	0.27	2.42	0.61	3.94	0.61	5.16	0.46	3.62
10	0.21	2.41	0.43	3.78	—	—	—	—
11	0.21	2.41	0.26	2.62	0.47	4.11	0.37	3.2
12	0.17	2.4	0.2	2.26	—	—	—	—
13	0.17	2.39	0.2	2.32	0.32	3.07	0.27	2.73
14	0.16	2.36	0.2	2.29	—	—	—	—
15	0.16	2.32	0.2	1.93	0.29	2.28	0.24	2.26
16	0.15	2.26	0.18	1.2	—	—	—	—
17	0.15	2.23	0.17	1.17	0.29	1.89	0.23	1.52
18	0.15	2.21	0.15	1.12	—	—	—	—
19	0.15	2.19	0.14	1.12	0.27	1.8	0.19	1.1
20	0.15	2.16	0.14	1.08	—	—	—	—

Table 4.1: The average and maximum *Delta E* error originating from round-trips under various illuminants. M represents mirroring, W warping, and the symbol n stands for their negation.

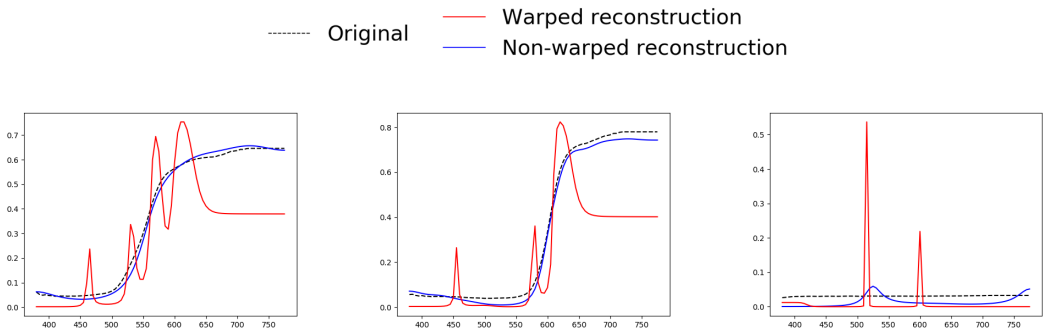
passing to the optimizer) rather than the number of moments, we surmise the contents of appendix B.1 in table 4.1, where we present the errors according to the number of coefficients.

By observing table 4.1, we conclude that it is beneficial to mirror the signal. Although the non-mirroring technique performs slightly better for $c \leq 9$, we are looking for a technique with which to save even the most complicated spectra which requires a lot more coefficients, and for that, non-mirroring is insufficient.

By applying the same principle when choosing whether to warp the signal, we assume that warping should provide better results. However, as the results from table 4.1 are not entirely clear on this matter, as for $c \geq 16$, the maximum error of non-warping is lower, we put this theory to practice and test the performance of the optimizer with both warping and non-warping.

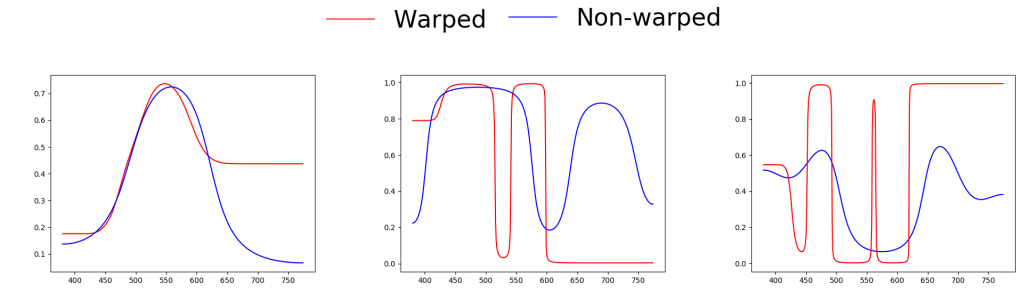
To achieve a certain color error upon reconstruction, the warping technique usually requires less coefficients. To give an example, if we require Delta E to be $\Delta E^*_{ab} = 0.1$ under the FL11 illuminant and we allow the coefficient count to grow up to 24, the Munsell Book of Colours requires 12 coefficients on average if using warping and 16 if not.

However, even despite the benefit of having less coefficients to optimize, the



(a) “orange” patch (b) “red” patch (c) “black” patch

Figure 4.1: Failure of warping when fitting atlas lattice points, shown on patches from the Macbeth Color Chart



(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

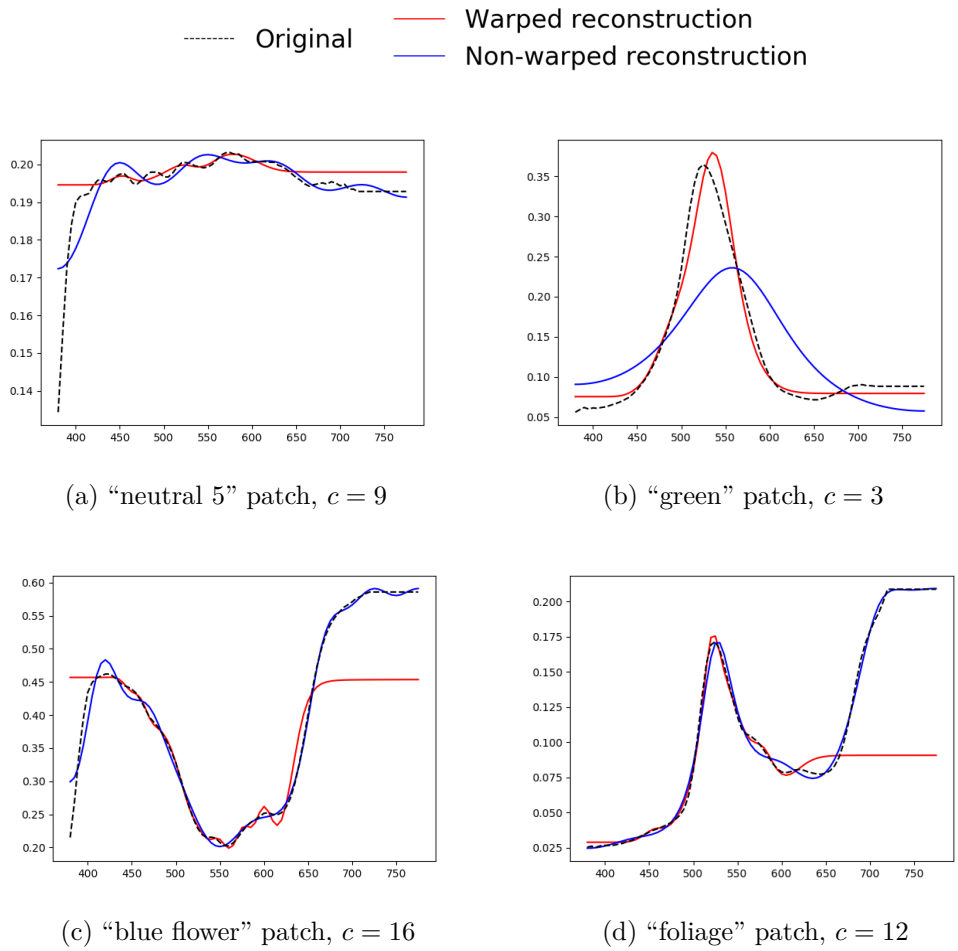
Figure 4.2: Comparison of warping and non-warping when used for fitting regular atlas entries. Note that the figures are illustrative, as they were created with an older version of the cube and therefore may not correspond to the current results.

average distance between the original and the fitted curves when using either of the cost functions (presented in section 4.1.3) is higher for warping. This does necessarily need to imply failure — as the edges of the curve do not often bear significant color information, we do not require them to be reconstructed as precisely. However, occasionally, the optimizer fails in so as the fitted curve does not follow the original at all but rather ends up as a spiky spectrum susceptible to metameric artifacts. We show some of these results in fig. 4.1, where we compare them to the results with the non-warping technique.

For the regular lattice points, we present a similar comparison in fig. 4.2, where we seed cubes of different sizes with distinct atlases and analyze the spectra at specific points. Not warping the signal is superior to warping even in this case, as it creates smoother spectra with less sharp edges.

As we eventually decide on using only 3 coefficients per regular lattice points (see section 4.1.2), and for that purpose, even warping works reasonably well, we do not need necessarily to account for regular lattice points when choosing whether to warp the signal. However, regardless, all the presented evidence points to the superiority of non-warping.

To justify the failures of warping, we first analyze its behavior during round-



(a) "neutral 5" patch, $c = 9$

(b) "green" patch, $c = 3$

(c) "blue flower" patch, $c = 16$

(d) "foliage" patch, $c = 12$

Figure 4.3: Comparison between the warped and non-warped reconstructed signal shown on multiple patches of the Munsell Book of Colours

trips in comparison to that of not warping. In fig. 4.3, we present a few such examples on different curves with different number of coefficients. We can clearly observe the behavior for which warping was designated — while the slight waves in the middle of the curve (at around 550nm) are reconstructed almost perfectly, the edges of the curve (i.e. the part of the curve leading up to 450nm and from 650nm) are flat and attain constant values.

If warping the signal, the optimizer therefore has very little influence on the edges of the curve, and therefore opts for optimizing only its middle, which might result in amplifying the already created sinusoidal-shapes.

By not warping the signal, on the other hand, the optimizer focuses on the spectra as a whole and is therefore less prone to spiky artifacts.

Therefore, we conclude that mirroring, but not warping the signal is the optimal approach for our purposes.

4.1.2 Number of moments

Another important parameter to determine is the number of moments (or coefficients) with which to store spectra at both atlas lattice points and regular lattice points.

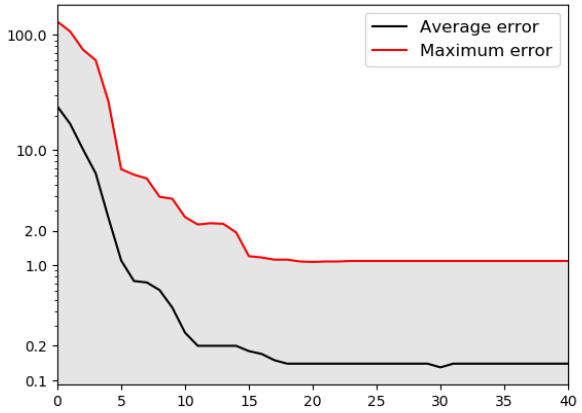


Figure 4.4: An

We start by analyzing the atlas lattice points. Specifically, we focus on the number of coefficients with which to store atlas entries in step one ref of our implementation, as those are the coefficients that are then used as prior for atlas lattice points. We wish for these coefficients to reconstruct spectra that resemble the original ones as much as possible, so as to avoid color errors under arbitrary illuminants.

By observing fig. 4.4, we see that the Delta E error obtained by round-trips under the CIE illuminants stabilizes at $c = 24$. As the curve precision does not worsen by adding more coefficients to the representation, storing all atlas entries with 24 coefficients might seem to be the optimal solution. However, as already mentioned in ref, it is beneficial for the performance of the optimizer (both from the viewpoint of the resulting curve's shape and the time complexity) to use as little coefficients as possible.

Therefore, we decide to store each spectrum with only the necessary number of coefficients. For each atlas entry, we obtain them iteratively — starting from $c = 4$, we check whether the coefficient count is sufficient, and, if not, we increase it and move on to the next iteration, repeating this process up to $m = 24$.

We determine the sufficiency of a coefficient representation by picking one of the most error-prone illuminants and determining the round-trip error under said illuminant. If the error falls below a certain, pre-defined threshold, we declare the representation sufficient.

We pick the illuminant used for these purposes from the list of the CIE illuminants (excluding the E and D73 illuminants, as these are not yet supported in ART, where we run the experiments). First, we compute the average number of required coefficients to achieve a specific error for each of them. By the assumption that these results give us a rough approximation of the average error, we conclude that the illuminant with the highest coefficient count is the most error-prone.

We present the results of our experiment in table 4.2. We use an error of $\Delta E^*_{ab} = 0.1$ due to the coefficient count variability under it, but any other would give similar results in terms of the order of the illuminants' coefficient counts.

Since the FL11 illuminant requires the highest number of coefficients, we use

Illuminant	Average error	Illuminant	Average error	Illuminant	Average error
A	14.8	F	12.91	F7	12.55
B	5.19	F2	13.01	F8	12.01
C	16.11	F3	12.93	F9	12.09
D50	15.56	F4	12.98	F10	16.24
D65	15.64	F5	12.95	F11	16.46
D75	15.76	F6	13.08	F12	16.30

Table 4.2: The average number of coefficients needed to achieve a round-trip error of $\Delta E^*_{ab} = 0.1$ for different CIE illuminants

it for our purposes, and move on to examining the optimal threshold.

Surprisingly, utilizing the Delta E error in our iterative process of acquiring coefficients did not provide as satisfactory results as expected — the entries represented with high number of coefficients ($c \geq 20$) were often unnecessarily accurate, while the entries represented with low number of coefficients ($c \leq 10$) often lacked precision. Therefore, we decided to compute the error as an absolute difference over all three RGB components, which outperformed both the Delta E error and even the Euclidean error in the RGB space (which was prone to similar, but less noticeable, behavior than the Delta E error). We set the threshold to 0.1 for an RGB range of $(0, 255)$, as we found it to perform well when tested for the actual uplifting.

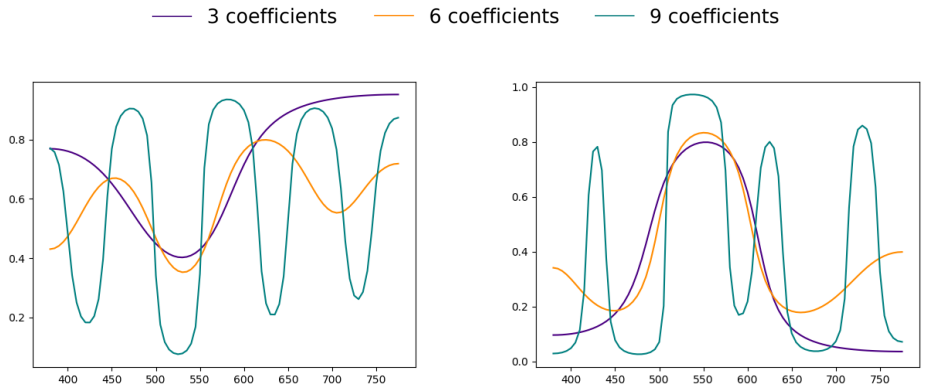
Although we attribute the failure of Delta E to its non-linearity, we have not yet obtained evidence as to support this claim.

Our approach for determining the sufficiency of coefficients is definitely not flawless. Firstly, neither the threshold, nor the computation of the error have been properly examined, which implies that there may be other, more suitable options. Secondly, since we examine only 18 CIE illuminants, there is a high chance that other, more error-prone, exist.

Even the assumption that the average coefficient counts determine the worst-performing illuminant falls short. As we have come across multiple atlas entries that attain the highest color error under an illuminant other than FL11, checking whether the error satisfies the threshold for all illuminants might be a more effective approach.

Furthermore, even if we were to perfect our method, there exist a lot more, vastly different approaches that might be used. However, their exploration and analysis is not within the scope of this thesis, and, as our method produces reasonable results, we leave it for future work.

Finding the sufficient coefficient count for regular lattice points is a lot more straightforward than for the atlas lattice points. We remind that, in order to avoid interpolation-caused artifacts, we require their spectra to be smooth. This property is especially important if two significantly different, spiky atlas entries are fitted in voxels close to each other. Propagating their original shapes during the cube fitting process would, under a different illuminant, create a noticeable border between the two colors, which is undesired. Rather, we want to see smooth gradients.



(a) $RGB = (222.1, 106.9, 164.5)$, fitting round $= 8/17$

(b) $RGB = (98.7, 205.6, 41.1)$, fitting round $= 9/17$

Figure 4.5: Comparison of spectra at regular lattice points when fitting . Cube

Figure fig. 4.2 already suggests that using less coefficients might benefit our smoothness requirement. We put this theory to test by comparing spectra of regular lattice points fitted with a different number of coefficients in cubes of otherwise identical parameters. We present some of the results in fig. 4.5.

Our assumptions have proven to be correct. By using less coefficients, we limit the optimizer's ability to reconstruct complicated spectra, therefore forcing it to create smooth, simple shapes. Specifically, we decide on using 3 coefficients — as their shapes are sufficient, using more is unnecessary, however 2 were unable to recreate shapes and often ended up being constant, straight lines.

Additional benefit of using only 3 coefficients per regular lattice point is the lower run-time of cube fitting in comparison to higher number of coefficients, as well as less optimizer failures and therefore no need for invocation of heuristic improvements.

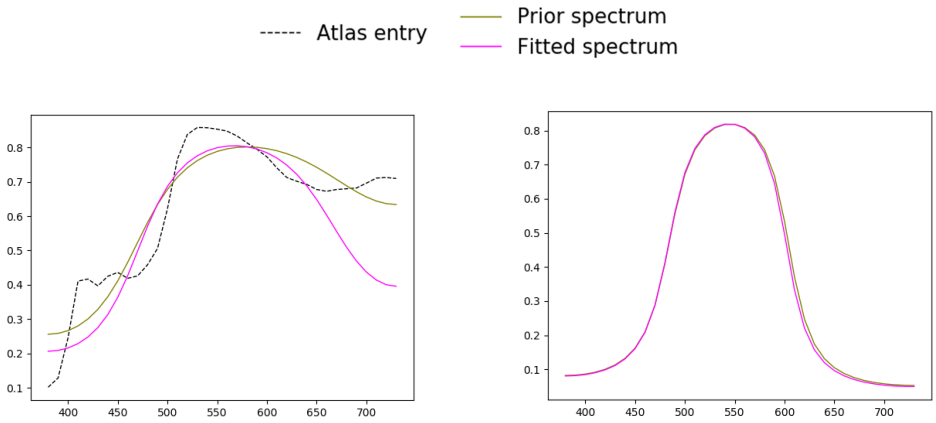
4.1.3 Cost functions

In addition to the moment storage technique, another thing greatly affecting the performance of the fitting are the cost functions of optimizer.

For the fitting of the sigmoids, Borgtool uses three cost functions, or *residuals*, each of them specifying the absolute color difference in one axis of the RGB cube. Such an approach outperforms both the 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.

As this approach performs rather decently in terms of both time complexity and the obtained results for the sigmoids, we try it out for the purposes of our optimization as well.

In case of fitting of the regular lattice points (see section 3.1.4), which requires the manipulation of 3 coefficients, the obtained results are satisfactory, both for the fitting in the second round and in the latter rounds (see fig. 4.6). The resulting curves are smooth, which suits the interpolation process during rendering, they evaluate to the desired RGB values, and the run-time is even better than for the



(a) Fitting in the second round, i.e. the prior coefficients are the result of "re-computation" of the fitted coefficients of an atlas lattice point

(b) Fitting in round 8/20, where the prior spectrum is that of a regular lattice point

Figure 4.6: Fitting of regular lattice points with 3 cost functions specifying the RGB difference

sigmoids. Therefore, we utilize this approach for the regular lattice points.

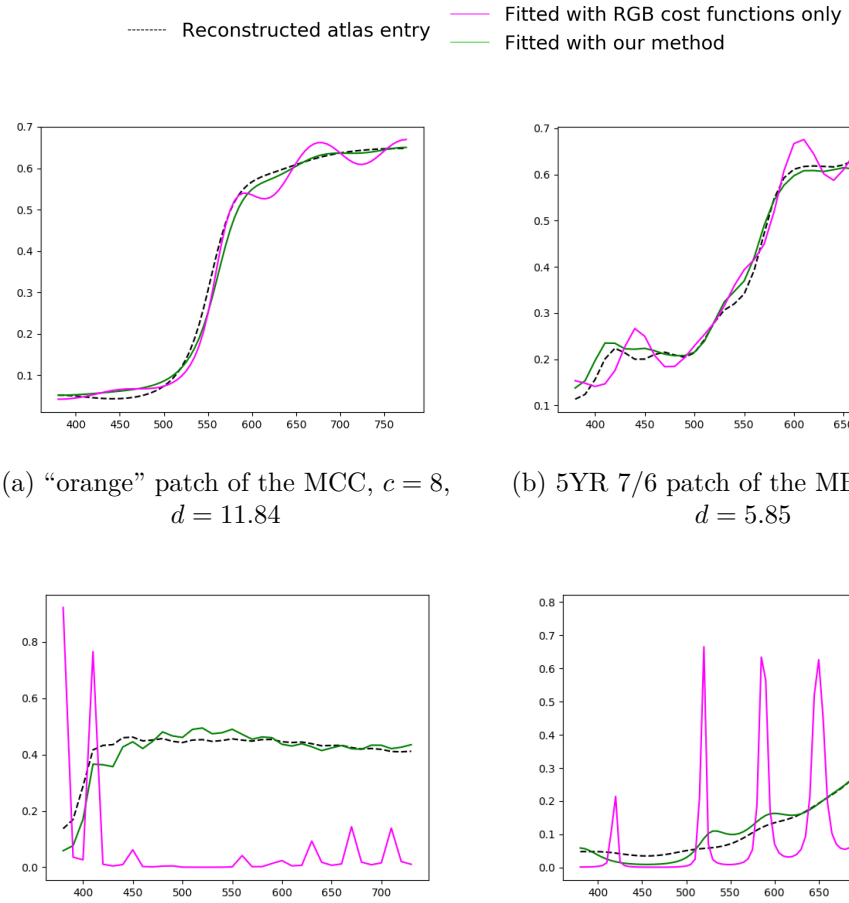
However, when fitting the atlas lattice points (see section 3.1.3), the fact that the cost functions do not take the resulting shape of the spectra into account in any way works to our disadvantage. We show an example of this in fig. 4.7 on the magenta plot, which demonstrates the performance of fitting of atlas lattice points with RGB cost functions only. Although it terminates as successful (as the RGB of the resulting curve is within the fitting threshold of the target RGB), the reflectance curve takes on a sinusoidal shape with a rather high amplitude, therefore losing resemblance to the original curve. This is due to definition of coefficients, which are, in their nature, Fourier coefficients, and are therefore prone to exhibiting this type of behavior. This is mainly perceivable when the color distance d between the atlas entry and atlas lattice point is rather high, as it gives the optimizer enough room to make visible changes in the shape of the reconstructed spectrum before converging below the fitting threshold.

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

We therefore conclude that we must incorporate the requirement of curve shape similarity into our cost functions. Following, we review the cost functions that we tried along with their performance.

Firstly, we implemented an approach similar to that for determining the number of coefficients with which to store atlas entries (see ref), i.e. we utilized the color error under a fluorescent illuminant (specifically, FL11). We defined three additional cost functions, each specifying the difference between the original and the reconstructed curve's RGB under the FL11 illuminant in one of the axes. If their values fell below a specific threshold, we terminated the fitting process as successful, if not, we increased the threshold and tried again.

Although this approach was successful on average (the average threshold was around $t = 0.025$), on occasion, the threshold often needed to be increased to



(a) "orange" patch of the MCC, $c = 8$, $d = 11.84$

(b) 5YR 7/6 patch of the MBC, $c = 14$, $d = 5.85$

(c) N 7.25 patch of the MBC, $c = 20$, $d = 12.67$

(d) "dark skin" patch of the MCC, $c = 12$, $d = 11.55$

Figure 4.7: Comparison between the RGB cost functions and our method for fitting atlas lattice points

values so high that it became obsolete. Using only one error, either the Euclidean distance or the Delta E error, caused similar issues.

Therefore, we decided to focus on the actual distance between the two curves. Our first attempt consisted of defining one residual per wavelength sample specifying the absolute distance between the two spectra at said wavelength (as the least square error proved to perform worse). We examined the performance of both around 360 residuals (i.e. 1nm increment between samples) and 36 residuals (10nm increment, as defined in most color atlases) when used alongside the 3 already defined RGB residuals. For $c \leq 9$, both of these options performed reasonably well. When fitting more coefficients, the sufficient threshold was, on average, around $t = 0.0096$, both before and after the introduction of our improvement of optimizing only first 4 coefficients, and although its maximum ended up being $t = 0.23$, overall, this method outperformed the previous one.

As we suspected that the failures were caused by the abundance of cost functions, we summed up their values and saved them into a single residual, which, when divided by the number of spectral samples, represented the average error per sample.

As the importance of curve samples in terms of proper color reconstruction

is mainly placed on their middle (at around 550nm), we attempted to add a heuristic-based weighting factor in an effort to focus on minimizing the distance between the two curve there. However, we did not succeed in improving our results, and we therefore dropped the experiment and examined the performance without weighting the values.

Such an approach substantially outperformed the previous ones, and even more after we decided to specify the absolute error rather than the least square error. The average error ended up being only about $d =$, and, of yet, no entry requiring $d >$ has been encountered. Therefore, we concluded that the optimal solution to our problem is to set 4 residuals, 3 specifying the RGB difference, and 1 specifying the average distance per sample.

We present some of the results achieved with our cost functions on the green plot in fig. 4.7, where we compare them to fitting with RGB cost functions only. In fig. 4.7c and fig. 4.7d, we specifically focus on the most problematic spectra with the highest distance d between the atlas lattice point and the atlas entry.

Although our approach substantially reduces the appearance of sinosidual-like behavior, it does not diminish it completely. It can be observed especially for atlas lattice points with $c > 14$ (see fig. 4.7c). This is due to the fact that we optimize only the first 4 coefficients, which, in their nature, are prone to creating such patterns.

Another drawback of our approach is the substantially larger time complexity, especially if improvement heuristics need to be applied. By examining the scope of the optimizer and utilizing its options further, or maybe even resorting to a different method of optimization, we might be able to improve upon both the time complexity, and the resulting spectral shape.

However, as the runtime is not the focus of this thesis and as the resulting shapes are satisfactory for the purposes of this these, we focused our efforts elsewhere and leave these improvements for future work.

4.2 Performace

When fitting other spectral data, the optimizer does not behave as drastically as shown in fig. 4.7. 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.

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.

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.

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] WS Mokrzycki and M Tatol. Colour difference e-a survey. *Mach. Graph. Vis*, 20(4):383–411, 2011.

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

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

[35] 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.

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.

[41] 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.

[42] 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.

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

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

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

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

[47] 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.

[48] 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`.

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

[50] 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.

[51] 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.

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

[53] 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	23.88	130.27	23.95	130.62	23.86	130.38	23.95	130.62
1	13.09	97.76	16.92	107.23	4.08	51.23	8.48	67.12
2	1.39	21.71	10.18	74.62	1.31	20.87	2.56	20.14
3	0.74	7.43	6.3	60.36	0.71	8.18	0.89	6.36
4	0.49	5.46	2.58	26.36	0.61	5.16	0.46	3.62
5	0.35	3.95	1.1	6.83	0.47	4.11	0.37	3.2
6	0.31	3.19	0.73	6.12	0.32	3.07	0.27	2.73
7	0.28	2.85	0.71	5.67	0.29	2.28	0.24	2.26
8	0.27	2.42	0.61	3.94	0.29	1.89	0.23	1.52
9	0.21	2.41	0.43	3.78	0.27	1.8	0.19	1.1
10	0.21	2.41	0.26	2.62	0.28	1.81	0.18	1.28
11	0.17	2.4	0.2	2.26	0.29	1.81	0.16	1.07
12	0.17	2.39	0.2	2.32	0.27	1.64	0.15	1.09
13	0.16	2.36	0.2	2.29	0.25	1.58	0.15	1.09
14	0.16	2.32	0.2	1.93	0.24	1.56	0.16	1.09
15	0.15	2.26	0.18	1.2	0.2	2.66	0.16	1.09
16	0.15	2.23	0.17	1.17	0.21	4.3	0.15	1.08
17	0.15	2.21	0.15	1.12	0.25	4.27	0.14	1.08
18	0.15	2.19	0.14	1.12	0.29	3.3	0.14	1.08
19	0.15	2.16	0.14	1.08	0.44	4.04	0.14	1.08
20	0.15	2.12	0.14	1.07	0.57	4.24	0.15	1.07
21	0.15	2.06	0.14	1.08	0.65	3.42	0.15	1.08
22	0.15	2.01	0.14	1.08	0.37	2.53	0.14	1.09
23	0.15	1.98	0.14	1.09	0.36	2.16	0.14	1.1
24	0.15	1.96	0.14	1.09	0.22	1.44	0.14	1.12
25	0.16	1.95	0.14	1.09	0.22	1.17	0.15	1.14
26	0.16	1.93	0.14	1.09	0.19	1.04	0.16	1.19
27	0.16	1.9	0.14	1.09	0.24	1.11	0.16	1.24
28	0.16	1.87	0.14	1.09	0.19	0.84	0.15	1.29
29	0.17	1.82	0.14	1.09	0.14	0.97	0.13	0.94
30	0.17	1.78	0.13	1.09	0.15	0.99	0.16	1.13
31	0.18	1.94	0.14	1.09	0.16	1.04	0.21	1.59
32	0.19	1.89	0.14	1.09	0.15	0.9	0.29	1.62
33	0.21	1.76	0.14	1.09	0.13	0.95	0.27	1.62
34	0.21	1.86	0.14	1.09	0.13	1.26	0.2	1.46
35	0.19	1.79	0.14	1.09	0.16	1.84	0.23	1.17
36	0.2	1.82	0.14	1.09	0.16	1.71	0.22	1.19
37	0.16	1.79	0.14	1.09	0.19	1.82	0.15	1.08
38	0.11	1.77	0.14	1.09	0.23	2.23	0.1	0.72
39	0.12	1.67	0.14	1.09	0.34	2.28	0.07	0.73
40	0.13	1.68	0.14	1.09	0.27	1.54	0.06	0.73