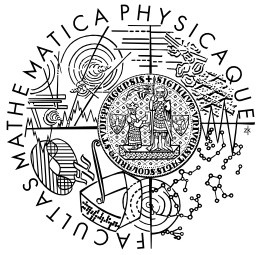


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Lucia Tódová

Constrained Spectral Uplifting

Department of Software and Computer Science Education

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

Study programme: Computer Science

Study branch: Computer Graphics and Game Development

Prague 2020

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

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

In ..... date ....

Author's signature

Dedication.

Title: Constrained Spectral Uplifting

Author: Bc. Lucia Tódová

Department: Department of Software and Computer Science Education

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

Abstract: Abstract.

Keywords: key words

Contents

Introduction **2**

1 Color Science **3**
1.1 Light and Color . . . 3
1.2 Color representation . . . 5
1.2.1 Spectral representation . . . 5
1.2.2 Tristimulus representation . . . 6
1.2.3 Color representation in rendering . . . 11

2 Spectral Uplifting **15**
2.1 Uplifting methods . . . 15
2.2 Constrained spectral uplifting . . . 19
2.2.1 Spectral sampling . . . 20

3 Implementation **25**
3.1 Uplifting model . . . 25
3.1.1 Initialization . . . 26
3.1.2 Fitting of starting points . . . 27
3.1.3 Cube fitting . . . 29
3.1.4 Cube improvement . . . 31
3.1.5 Cube storage . . . 31
3.2 ART integration . . . 32

4 Results **33**
4.1 Storing moments . . . 33
4.2 Number of moments . . . 35

Conclusion **38**

Bibliography **39**

A Software user guide **43**

B Attachments **44**
B.1 Delta E error caused by moment sampling . . . 44

Introduction

1. Color Science

Color science, or colorimetry, 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, such as Mitsuba or Corona (add a link).

1.1 Light and Color

The core of human visual perception is electromagnetic radiation, which consists of waves that 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 happen 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 is divided into bands to distinguish known types of light; low frequency light such as gamma rays or X-rays; extremely high frequency light such as radio waves.

In this thesis, we will 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, making up electromagnetic radiation. They 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 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. 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

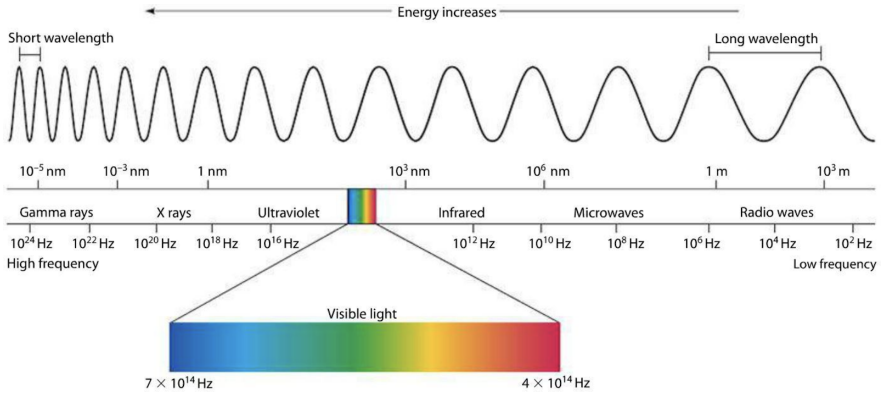


Figure 1.1: An illustration of the electromagnetic spectrum [3]

signal via an optic nerve to the brain, giving the notion of light and color. There are two types of photoreceptors in the human eye — rods and cones.

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

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

The human eye has three types of cones:

- *L-cones*, which are the most responsive to longer wavelengths at around 560nm. When they are 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 idea behind using three base colors has been adapted in color science to create multiple tristimulus color representations. We will discuss these more thoroughly in the following section.

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

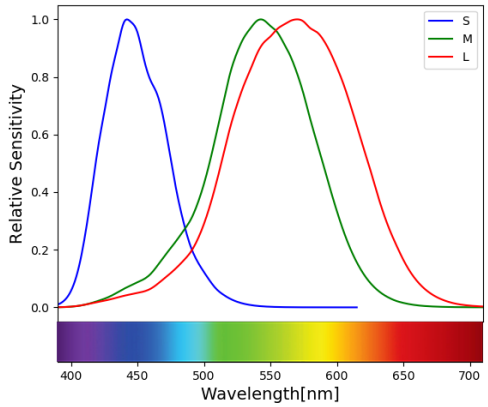


Figure 1.2: Relative sensitivity of S, M and L-cones plotted according to the data measured by Stockman and Sharpe [43].

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 light present in the scene is red, other wavelengths than red will never hit the object. Therefore, the object might reflect only a subset of wavelengths than it would under white light, which might change the resulting 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 should 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 — 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 energy and the outgoing energy at these wavelengths. The dependence of reflectance on the wavelength is called a *reflectance spectrum*, and is usually a smooth, continuous curve.

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 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, we will use SPD to describe the emissive properties of light sources, and will therefore consider SPD to be a function of wavelength and power.

To compute the color of an object under a light source, one must simply combine the light source's SPD with the reflectance curve of the object, as shown in figure. This way the physical properties of color are preserved and the result is the same as it would be in nature, which, as we will show later, is not always the case with tristimulus representation.

1.2.2 Tristimulus representation

The obvious drawback of spectral representation is the difficulty of its discretization. Another, bigger problem, is caused by the fact that there is an infinite number of possible spectral curves, but only a discrete number of colors perceptible by human eye or even possible to generate by a computer (use word domain?). Representing colors with spectral distribution therefore requires their conversion to a discrete space before arbitrary visualization process.

Tristimulus representation skips the conversion steps and saves the already discretized 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.

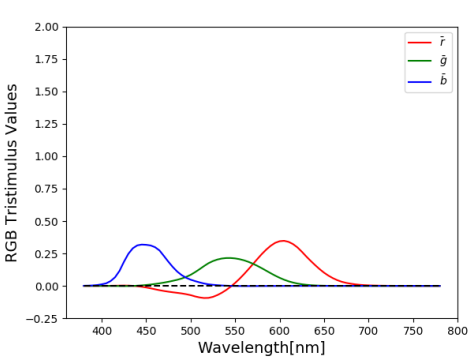
RGB color space

The RGB color space is an additive space employing three primaries — red, green and blue. In other words, if you have three lights with red, green and blue chromacities respectively and you use them to illuminate a single point, you can create any color within the RGB color space solely by changing the lights' intensities.

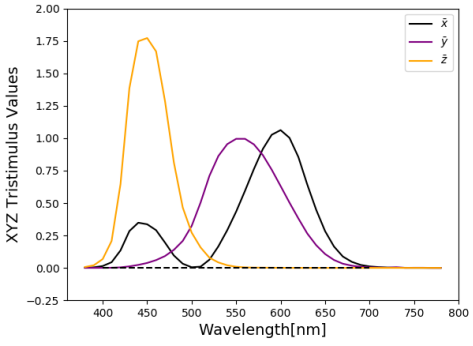
An RGB value can be therefore thought of as a point in a 3-dimensional euclidean space with each of the coordinate axes representing one of the primaries. Specifically, as the light's intensities must be bounded, we can narrow this space down to a cube starting at the base of the coordinate system. Usually, 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. 1.4.

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) $\overline{r}(\lambda)$, $\overline{g}(\lambda)$ 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 1.3: Color matching functions

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. 1.3a, the functions may also acquire negative values. This posed a problem at that time due to calculation errors. Therefore, to eliminate these negative portions of functions, CIE designed a new, imaginary color space — the XYZ color space (tu ref?).

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 therefore serve as a "middle step" when performing a conversion from SPD to an arbitrary tristimulus space, which eliminates the need for other color matching functions. 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*.

First, the following conversion from the X, Y and Z values to their bounded versions, also called *chromaticity coordinates*, is performed [13]:

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

Due to normalization ($x+y+z=1$), $z=1-x-y$, which means that we can drop the term z from the representation as it does not give any additional information about the current color. It also implies that we lost some information during the conversion — we cannot reconstruct the original XYZ triplet using only two values x and y 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 the luminance.

Plotting the values of the x and y components creates a *chromaticity diagram*, shown in fig. 1.4. Each point of the curved boundary line (which is also called the *spectral locus*) corresponds to a XYZ value that is the 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 the xyY color space is already much more intuitive in terms of human color perception, the differences between individual triplets of the system are not perceptually uniform. The Hunter's Lab color space ref addressed this issue and was designed so that the distance between its two triplets characterized roughly how different they are in chromaticity and luminance. It is based on the Opponent color theory [24], which suggests that the cones in the human eye are linked together in opposing pairs and that the visual system records the *difference* between the stimulation of the pairs rather than the cones' individual responses.

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

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

- L^* — indicates lightness, i.e. the difference between *light* and *dark*. Its

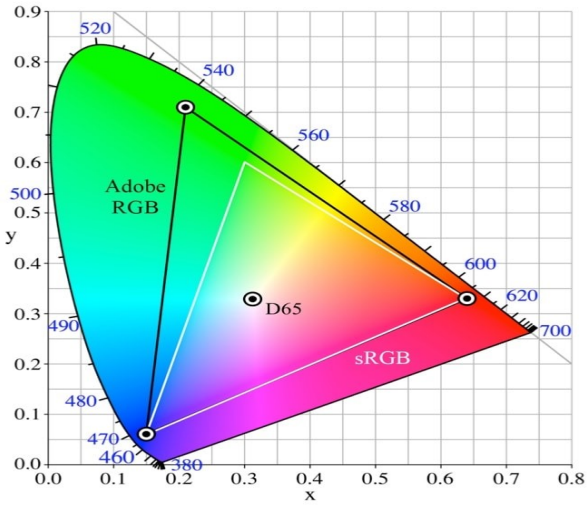


Figure 1.4: An illustrative comparison of the sRGB and Adobe RGB gamut in the chromaticity diagram based on images created by Choi et al. [7]

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 indicate red.

- *b** — defines the difference between *yellow* and *blue*. Positive values indicate the object to be more yellow, while negative values indicate 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 is used for color balance corrections by modifying the a* and b* components, and for lightness adjustments by modifying the L* component.

Another advantage and common use of the L*a*b* color space is for computing *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.:

$$\Delta E_{76} = \sqrt{(L^*_2 - L^*_1)^2 + (a^*_2 - a^*_1)^2 + (b^*_2 - b^*_1)^2}, \tag{1.4}$$

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 sensitivity of the human eye to color differences is not uniform. It is, for example, more sensitive to small color differences in dark blue colors than it is in e.g. light pastel colors. The ΔE^*_{ab} error does not take this ununiformity into account and therefore shows exaggerates differences in light colors while compressing perceptual distances between darker colors. To improve upon these shortcomings, other measuring techniques for computing Delta E have been proposed over the years, such as Delta94 and Delta2000.

Delta94 is computed by modifying the original L*a*b* values of both colors to compensate for perceptual distortions in the color space and computing Euclidean distance from the new 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. Along with the corrections added to Delta94, Delta2000 adds overall 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 regions.

From the listed Delta E equations, the Delta2000 error measurements are the most accurate in terms of human color difference perception [16] and, therefore, will also be used in the practical parts of this thesis. However, as the specifics of the Delta2000 equations are out of scope of this thesis, we refer the interested reader to the original article by Sharma et al. [41].

Other color spaces

In addition to the already named 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 CIELAB generally improves CIELUV in terms of color difference [29], CIELUV does not have as many inaccuracies in the dark regions [40]. Therefore, it is often recommended to use the CIELUV color space for characterization of color displays and CIELAB color space for the characterization of colored surfaces and dyes.

- *HSL* and *HSI* color spaces define color by its *hue*, *saturation* and *lightness* (or *intensity*). They are an alternative representation of the RGB color space and must therefore be defined purely with reference to an RGB space [18]. As their components correlate better with human perception of color than those of the RGB system, they are often used in image processing applications, e.g. for processes such as feature detection (edge detection [46], object recognition) or image segmentation (which can be performed solely with/by? 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 theory states that maximizing CMY values should render perfect black, in reality, the printing inks are not 100% CMY and their combinations cannot produce rich black. For this purpose, a fourth component, *black* (K), is often added, giving rise to the CMYK model.

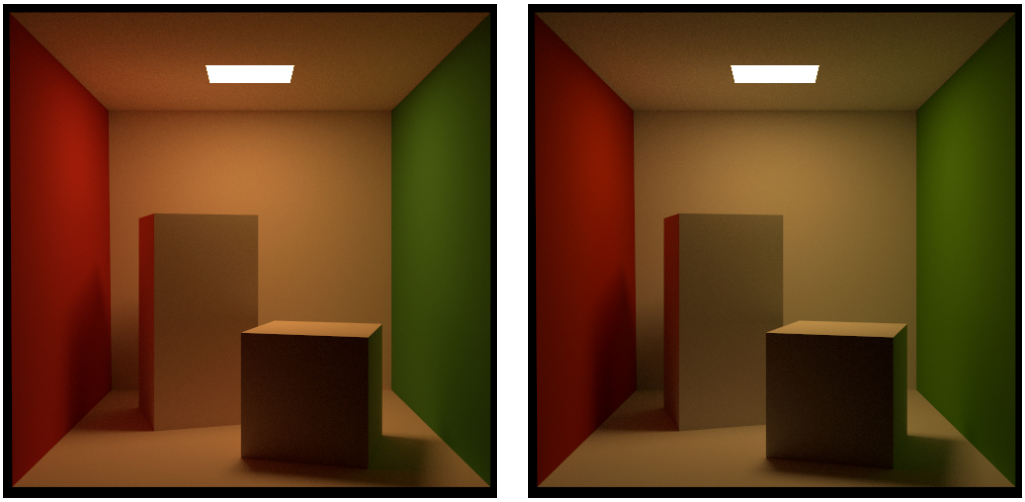


Figure 1.5: Comparison of an RGB-based rendering and spectral-based rendering as presented in the documentation of Mitsuba2 [47]. Left: Spectral reflectance data of all materials is first converted to RGB and the scene is then rendered in the RGB mode, producing an unnaturally saturated image. Right: Scene is rendered directly in the spectral mode, resulting in more realistic colors.

Other color spaces include 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, easy to implement and memory efficient.

However, light in real world does not travel as a tristimulus value, but rather as a distribution of wavelengths. As RGB renderers do not possess full-spectral information of materials and light in the scene, they cannot properly simulate the physical properties of the color during e.g. reflections or refractions when ray tracing.

Spectral rendering, on the other hand, uses full-spectral information of all materials and light in the scene during 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. 1.5, on a scene already rendered by Mitsuba2 [47].

In addition to rendering reflections and refractions more convincingly, another

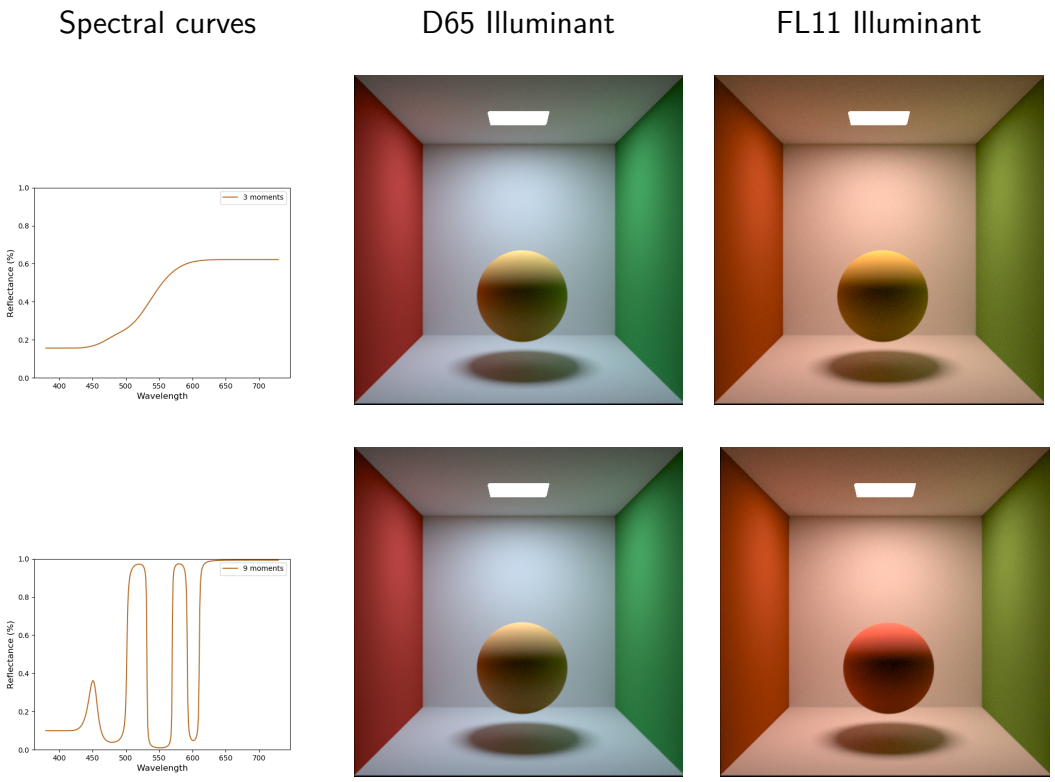


Figure 1.6: The effects of metamerism. Left: Two different spectral reflectance curves, both corresponding to RGB=(0,255,0). Middle: A box in a cornell box rendered with a A Cornell box rendered with

reason for using spectral rendering is its capability of simulating physically based phenomena that arises 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. 1.6. Although we perceive the color of both objects to be the same under D65 illuminant (daylight), when illuminating the scene with x illuminant, the color changes.

  Obviously, this behavior is irreproducible by an RGB renderer, as it cannot replicate the behavior of spectral reflectance under an illuminant.

- *Fluorescence*

  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 unrealistically bright material appearance, perceivable 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 it through custom shaders [52]. As it produces satisfactory results and is immensely easier to implement than physical simulation, 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 their color according to the current viewing angle. It arises when the object's physical structure causes interferences between light waves (e.g. inside extremely thin dielectric layers), yielding rich color variations [4]. It can be perceived in nature in certain plants, specific minerals, butterfly wings, peacock's feathers, snakes, but also in man-made products such as oil leaks, soap bubbles or car paints.

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

- *Dispersion*

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

  Probably the most popular example of this phenomena is white light hitting a dispersive prism. Upon interaction, light is split into a spectrum, creating a "rainbow" effect.

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

- *Polarization*

  Electromagnetic waves traveling through space are *transverse waves* — their oscillation is perpendicular to their path of propagation. By default, the directions of oscillations are arbitrary for each photon — this type of light is called an *unpolarized light*. Restrictions to the directions of oscillations

(also called *polarization*) render *polarized light*. Such phenomenon usually occurs upon light's interaction with certain materials.

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

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

2. Spectral Uplifting

Although spectral rendering has multiple advantages, many renderers do not consider them to compensate for the ease of use and memory efficiency of the RGB representation. Even the physically-based phenomena can be "faked" by a few simple tricks, and, therefore, many conventional rendering systems are RGB-based. This implies that most textures and materials created for renderers are also RGB-based.

However, spectral renderers are still used both in the research and in the commercial sphere (e.g. ART, Mitsuba, Manuka). In comparison to an RGB-based texture, however, creating spectral textures is much more complicated and usually requires a real-life model whose reflectance spectra can be measured with a spectrometer, which is, in many cases, virtually impossible.

The obvious solution is to convert the already existing RGB models to their spectral variants. We refer to this process as *spectral uplifting*, however, other sources also use the term *spectral upsampling* [19].

By being able to uplift RGB values, we could utilize the RGB textures and materials and therefore eliminate the need for a repeated creation of the textures from scratch by the tedious process of measuring specific spectral values.

However, converting an RGB value into its respective spectrum poses multiple difficulties. As the relationship between the spectral and 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 causes distinct spectra to behave differently, which consequently results in *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 new technique, *constrained spectral uplifting*, which provides means for solving the above mentioned problems.

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 (e.g. one method may output reflectance spectra with values outside the (0,1) range, other method might work only for saturated colors etc.).

We base most of this section on an article by Jung et al. [22], as it overviews multiple spectral uplifting techniques. It also proposes a new technique, which is considered to be the current state-of-the-art.

One of the first techniques was proposed by MacAdam [27]. The main goal of his research was to achieve the highest possible brightness for a given color saturation in printing. The uplifting process was only a byproduct of proof of limits

to the brightness of colors, created especially for representing the reflectance of the researched colors (i.e colors of maximum brightness for any given saturation). Although this method is not limited to a specific input, it produces spectra that are box shaped and only consist of rising and falling edges. This type of representation is unsuitable for colors usually found in nature, as they tend to have smooth spectra.

Another technique was proposed by Smits [42]. In this case, the uplifting is based on a box basis split into 10 discrete bins, which are derived using an optimization algorithm that accounts for energy conservation and aims for overall smoothness of the spectra. This approach is practically implemented and widely used, as it provides satisfactory results in the sRGB gamut [20]. However, in some cases, the uplifted spectra acquire values above 1, which does not satisfy the (0,1) range criterion. Furthermore, conversion of an RGB value to spectra and then back produces slight differences, which are 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 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 if trying to uplift bright colors.

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

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

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

The main goal of their research was to create a spectral representation that would be both energy-conserving and would have a successful round-trip, i.e. the DeltaE 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 DeltaE error, a simple analytical model has been created. Spectra in accordance with this model are represented as following:

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

Algorithm 1 Spectral uplifting by Jakob and Hanika [20]

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

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 extremely smooth and simple, which corresponds to many spectra typically found in nature. Another great advantage is memory efficiency, as storing one spectrum requires only three values. However, representing spectra as such also has its drawbacks. For example, there is currently no straightforward, well-defined computation of the RGB $\rightarrow$ spectrum conversion in such a domain. To uplift an RGB value, one must keep "guessing" the coefficients until the spectrum evaluates to the desired RGB.

In this specific implementation, the "guessing" process is performed mostly by the CERES solver [1] (note:should I find out how this works?). It requires only an initial guess and a metric according to which it improves the guess (i.e. the DeltaE error originating from round-trips) and requires only a few iterations to converge to 0.

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

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

Discretization of the RGB space does not, however, eliminate the need for uplifting RGB values that have no mapping in the spectral uplifting model. In

such case, the unknown spectral reflectance values must be computed from the already existing mappings. Without much elaboration, three straightforward methods of how to create a mapping from an arbitrary point in the RGB cube come to mind:

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

The original paper suggests that interpolating coefficients should, within limits, produce reasonable spectra without unexpected artifacts. However, even despite the longer rendering time, the spectral uplifting tool in ART interpolates spectra instead [33]. The reasoning behind is that it provides higher round-trip accuracy, especially in a case such as this — when the spectra are smooth and similar to those of their neighbors. The approach of copying the coefficients is not usually used, as the output image is slightly darker and less saturated even compared to the interpolation of spectra or coefficients.

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

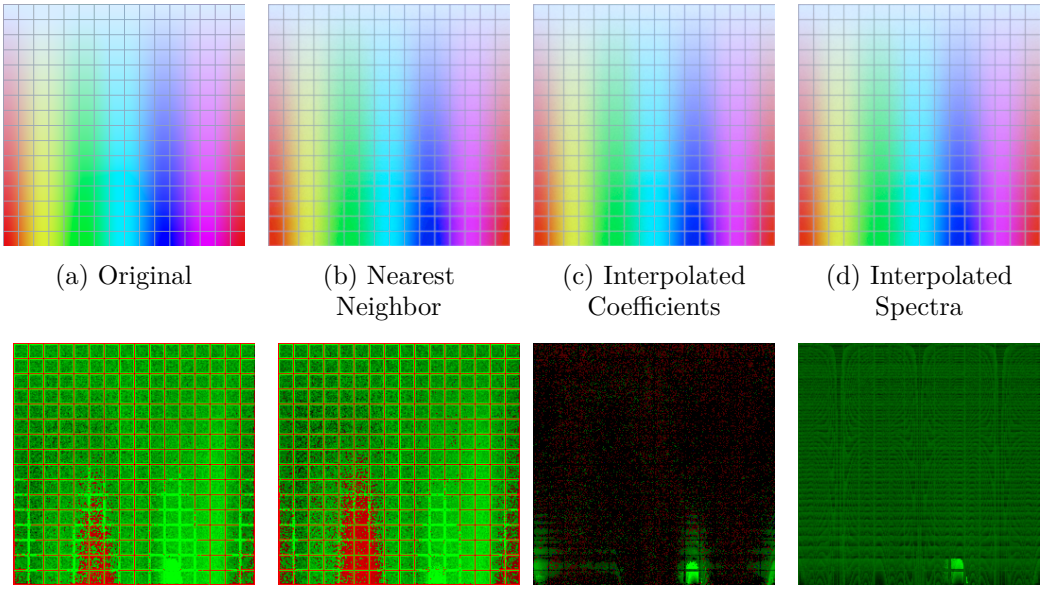
As seen in fig. 2.1g, the approaches of interpolating spectra and coefficients are extremely similar. This and results from both fig. 2.1e and fig. 2.1f imply that all approaches tend to create a bit darker images overall, with an exception of the green region, which is, in contrast, a bit lighter. The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

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

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

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

The approach by Jung et al. [21] tries to solve this issue by extending the set of three sigmoid coefficients with three additional ones specifically designed for handling fluorescence. Its goal is to avoid creating blocky spectra at gamut boundaries and rather create smooth spectra with added fluorescent dyes to compensate for the lack of saturation. Similarly to Jakob and Hanika [20], the uplifting model is also based on an RGB cube structure optimized by the CERES solver. It is



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

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

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

expected for the optimization to take longer, as it has 6 coefficients to consider. However, the method is the first one capable of simulating fluorescent spectra, which is especially useful for wide-gamut input textures.

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

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

2.2 Constrained spectral uplifting

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

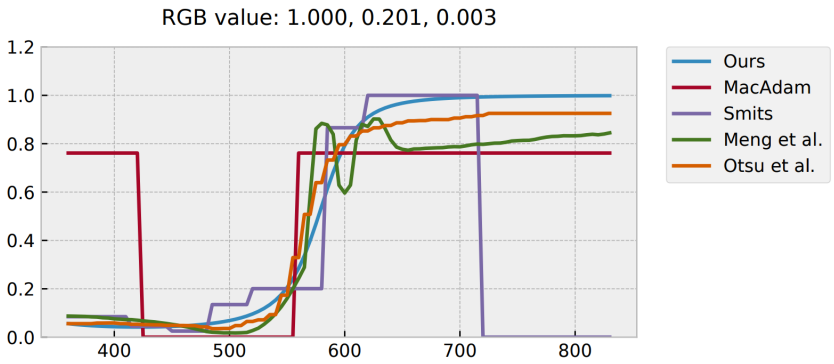


Figure 2.2: Comparison of spectral uplifting techniques as shown by Jakob and Hanika [20]. All spectra were created by uplifting the (1, 0.201, 0.003) RGB value and the results were plotted according to the corresponding techniques. The "Ours" approach, in this case, refers to the approach by Jakob and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

for the creation of said models. As it is pretty common for the 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 base colors would be extremely useful in such cases.

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

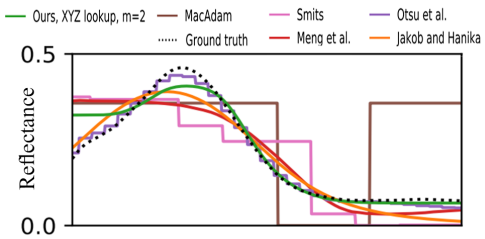
We call this process *constrained spectral uplifting*. The fact that it does not provide as much freedom as other spectral uplifting approaches works for our benefit, as the results are not a subject to high metamerism, which is, after all, the goal of this thesis.

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

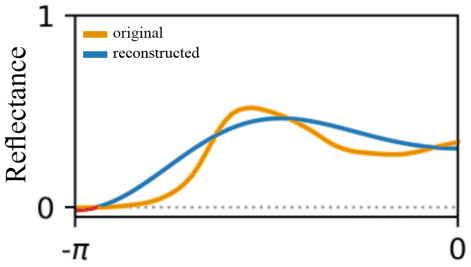
2.2.1 Spectral sampling

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

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



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



(b) Reconstruction with truncated Fourier series [36]

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

extremely memory inefficient, especially when taking into account all the other mappings that must be created later in the system.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The summary of the reconstruction process is as follows:

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

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

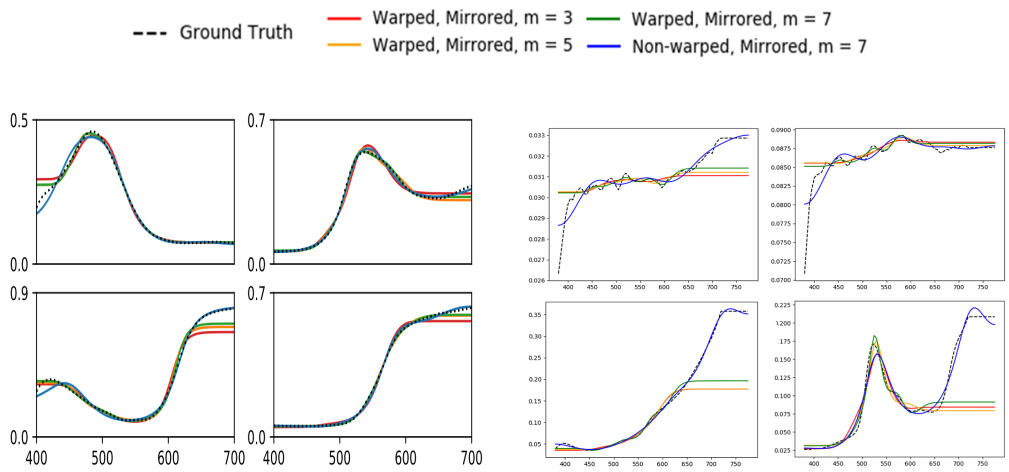
This technique can also be used for the storage of emission spectra. As these tend to be more spiky and sharp, a lot more moments is required than for the reflectance spectra. To give an example, even a mirrored approached with $m = 15$ does not produce satisfactory results for some emission spectra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

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

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

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

Adding a few unimportant words for now, just to see how the images will be aligned when a new paragraph will be added Adding a few unimportant words



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

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

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

for now, just to see how the images will be aligned when a new paragraph will be added

3. Implementation

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

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

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

3.1 Uplifting model

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

Currently, 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:

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

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

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

the regular points only. We explain the reasoning behind this decision and its impact on rendering and overall performance more thoroughly as we continue with this section.

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

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

3.1.1 Initialization

This part of the run is responsible for three things:

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

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

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

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

We store the spectral curves of the individual atlas entries by using the Fourier coefficients as described in section 2.2.1. We use the maximum number of available moments (currently 9, see explanation in section 3.1.2), and we both mirror and warp the signal prior to coefficient computation. We explain the reasoning behind this in section 4.1, where we run experiments to decide on the most efficient and precise method.

```
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.

3.1.2 Fitting of starting points

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

For these purposes, we utilize the user-specified color atlas. Ideally, every entry of the color atlas should correspond to one lattice point. We try to achieve this by iterating over the atlas entries, finding the cube voxel in which the current entry should reside in terms of RGB and then choosing the closest lattice point from all 8 corners of the voxel. We classify this lattice point as an *atlas lattice point* and we assign it the coefficients saved for the current atlas entry. We call this process the *seeding* of the cube.

Obviously, achieving a complete injection between atlas entries and their respective lattice points is not always possible. In some cases, we might find that the closest lattice point to an atlas entry we are about to map has already been seeded. This might be due to the following reasons:

- the number of atlas entries is higher than the size of the cube
- the atlas entries are concentrated around specific colors (e.g. we seed a very small cube with only the Page 14 from the Munsell Book of Colors as in ukazka)

In both cases, we issue a warning and recommend the user to use a higher dimensional cube.

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. It is apparent that the distance may be higher than the defined fitting threshold. In such cases, we must "improve" upon the coefficients so that the resulting color difference is as low as possible.

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

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

where $x = \{x_0, x_1, x_2, ...\}$ is a parameter block that we are trying to improve (i.e. our coefficients) and f_i are so-called *cost functions*. The definition of cost functions is dependant solely on the current problem. In our case, we primarily require to minimize the difference between the lattice and the target RGB. Our secondary requirement is the shape similarity of the original atlas entry curve and the resulting curve of the atlas lattice point. This gives rise to multiple choices for cost functions, such as using the difference between curves along with the Delta E error, using one or multiple cost functions for the RGB error etc... After implementing some of them and testing their performance, the results of which we provide in ref, we decide on neglecting the shape similarity requirement and using three cost functions, each specifying the absolute difference in one of the three axes of the cube.

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

CERES solver

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

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

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

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

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

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

The need for a heuristic also suggests considerable difference between the prior and the resulting coefficients, which implies distinct spectral curves. Such a behavior is undesired, as it may result in strong metameric artifacts.

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

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

As we do not require the user to always specify a color atlas, we provide an option for starting in the middle as well. This requires us to specify a set of prior coefficients for the middle point. We have determined it by iterating over multiple existing color atlases and searching for spectral curves that roughly evaluate to an RGB of (0.5, 0.5, 0.5). By obtaining the coefficients of such curves, we have eventually established the prior coefficients to be $\{0.5, 0, 0, 0, ...\}$, i.e. an array of zeroes except for the first coefficient. We use these prior coefficients for all available moments and cube dimensions.

3.1.3 Cube fitting

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

The core part of the algorithm is extremely similar to the algorithm used for fitting with sigmoids. However, we also add a heuristic-based improvement of the coefficients, implemented in steps 16 through 21. This part aims to minimize the shortcomings of the optimizer mentioned in section 3.1.2.

We implement the heuristics by changing up the first coefficient a pre-defined amount of times. In constrast to the heuristics applied when fitting atlas entries, we ommit the changing of the second coefficient completely and we also use significantly less iterations when changing the first coefficient. 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

Algorithm 2 Fitting of the cube from starting points

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

the cube fitting is done, we still run an "improvement" step for the unsuccessful points. We discuss the specifics of this step in more detail in section 3.1.4.

The reason behind using improvement heuristics both during and after fitting is both for improved time performance and a more desirable resulting shape of the spectral curves. Cube improvement after fitting is extremely time-consuming in contrast to a simple change in coefficients. Moreover, due to reasons explained in section 3.1.4, the resulting curve could take a shape dramatically distinct from the shape of its neighbors, which is an undesirable behavior. On the other hand, trying to improve all coefficients during the cube fitting would require a complicated heuristic including a lot of optimizer runs, which might eventually lead to a lowered time performance. We show all of these effects in ref.

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

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

3.1.4 Cube improvement

In extreme cases, such as when using too many coefficients to store entries of a cube with either a low dimension or a low fitting threshold, fitting some of the points may be unsuccessful even after applying heuristic improvements. We attempt to solve this by simply iterating over all the already fitted points and using their coefficients as prior for our point. We even apply heuristics similar to the ones used during the fitting of atlas entries. This shows to be useful especially if we have an extremely limited number of sets of prior coefficients, which happens usually if the overall size of the cube is small.

We call this process *cube improvement*. Although the heuristic may seem simple and illogical at first, we must once again zvyraznit that it is aimed solely at the most extreme cases. As a matter of fact, it usually does not even get triggered and, if it does, it is only for a highly problematic point of two. We present the exact statistics in ref.

Obviosuly, improving the points as such has its drawbacks. We do not focus on the time performance, as we mainly want this part to be successful rather than fast. Moreover, as this step runs for a small set of numbers only, the execution time is negligible in comparison to other steps.

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.

multithreaded

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

3.1.5 Cube storage

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

- *version*
- *features*
- *dimension*
- *coefficient count*, i.e. how many coefficients are used to reconstruct spectral information for every lattice point
- *illuminant* under which the cube has been fitted
- *fitting threshold*
- for every point, we store:
  - *coefficients*

- *lattice RGB*
- *target RGB*
- *fitting distance*, i.e. the distance between lattice and target RGB, should be below fitting threshold
- *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.

We therefore ommit parameters that serve for
We do not, however, save all the cube's available information, but rather
Their structure is different than those of the sigmoid cube as, obviously, they have different number of coefficients and we must also store the number of coefficients in the cube structure, which is not required in sigmoid.

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.

4. Results

In this chapter, we talk about the results.

4.1 Storing moments

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

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

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

According to the observation of table 4.1, it is clearly beneficial to use both mirroring and warping. However, correct round-trips are not the only factor we need to take into consideration when fitting the cube. We also need to focus on both the *smoothness of the resulting spectra* and the *behavior of the optimizer* under the current technique.

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

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

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

Round trips are important

This drawback of the optimizer can, for example, be perceived when using warping. As the core idea behind warping is to focus accuracy on the more

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

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

"important" regions of the curve (i.e. around the 550nm wavelength), the reconstruction of the curve's edges is deemed to fail. Moreover, it amplifies the slight bumps that sometimes occur around the middle. This is a desired behavior when we are aiming for correct color reconstruction. However, if we were to use such a curve as prior to a lattice point with similar RGB, we would require only slight changes in its shape, e.g. minor tweaks at the edges of the curve. This, however, is not possible to do when using warping, as this method trades off the capability to influence edges with having too much freedom around the middle. The optimizer is therefore apt to amplify the already existing bumps even more in order to achieve the desired RGB value.

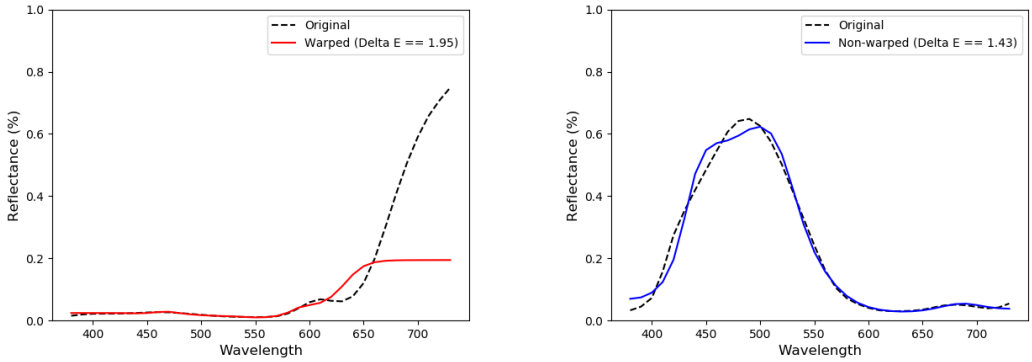
In figure 2, we present the curves reconstructed from the coefficients of the lattice point with the target RGB of (). Both figures

In a, the cube was fitted from the middle by using the warping technique, while the cube in b did not use warping. For both fittings, we used 9 coefficients and we We can see that by the time the cube grew enough so that the lattic

Determining whether to use warp is a bit more complicated. Peters et al. [35] recommend using warping at all times, especially if the number of moments is below 5. The table suggests that although warping results in a higher average error, the maximum error is lower. As we place greater importance on minimizing the maximum error, we are inclined to use warping.

However, warping does not perform as well in terms of curve reconstruction. As its core idea is to focus accuracy on the more "important" regions of the curve (i.e. around the 550nm wavelength), it does not reconstruct the edges at all.

Warping obviously provides better color reconstruction. We lose information in the edges (PICTURE), but that does not matter in terms of color reconstruction. More coefficients can then be focused on the middle, which makes us able to simulate the shape in the middle more accurately, we can e.g. see some slight curves that we cannot do with the same number of moments without warping



(a) Warped, Delta E == 1.95, MUNSELL 025R 02 06, 2.5R 2/6 sample

(b) Pantone 3551 C, Delta E == 1.43

Figure 4.1: Worst case scenarios

(PICTURE). (mozeme naznacit ze to z sekcie jedna metamerizmus je warping)

However, as we put a lot of attention to the middle, we therefore allow coefficients to create some extremely crazy shapes. show PICTURE. It does not work well with the optimizer - the optimizer gets prior coefficients, which are fixating on the middle and the optimizer does not care about the shape of the curve but only about the resulting RGB. Therefore, it always zvyrazni, zvacsi the effects of the coefficients (the slight curves that they create). We can see this in picture daco.

Therefore, we do not use warping. We use mirroring - non-mirroring would make us use only a little number of coefficients, which is nevyhodne.

4.2 Number of moments

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

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

Therefore, we go for fitting with an atlas. The more coefficients, the better the reconstruction when saving an atlas entry. However, we must remind that we do not need to only save the coefs, we need to do the following things: - create coefficients from atlas entry - find lattice point closest to atlas entry - use these coefs as prior for optimizer - run the optimizer This already suggests that the optimizer might be the problem, and it is. Imagine we have only an 8-dimensional cube, and the atlas entry is in the middle of voxel. The euclidean distance could be as high as daco, which is too far. Optimizer sice takes the coefficients, but has to change them way too much so they fit what it wants to do. This results in extreme changes and the information from the atlas entry is not utilized at all. We show this in PICTURE.

Our only way ako to obist je proste pouzit a larger cube. Another slight trick is to use a lower number of coefficients - we therefore do not get as big artifacts because we cannot possibly reconstruct such crazy spectra with low number of coefficients.

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

What I did so far: - created the support for multiple coefficients - added warping to the signal because the results are definitely better with warping - however, the problem occurs when we need to actually fit the points - e.g. even if we are using a 32 dimensional cube, white atlas entry of the macbeth is basically right in the middle of the voxel and therefore is far away - the coefficients that we used to save the curve do not reconstruct the curve exactly but that does not cause problem, the optimizer does - by saving the signal of white with 9 coefficients WITH warping, the optimizer actually has a lot of work to fit especially the stuff in the middle of voxel -¿ the resulting curve is way different from the one that we started with, with a lot of sharp things, which cause a lot of metameric artifacts - if using warping, it is strongly recommended then to use a lot less moments as they force the curve to be smooth and the limited curves help in this case - we can show some of the results with warping, obviously they suck

We talk about the following results:

- the quality of the fitting - i.e. how many times we get stuck in local minima, how does that behave,

- how many moments should we be using (input the table that we have already created) - we say how many moments we actually support, also mention we could support more (more residual blocks in optimizer) but that would take way too long

- the results when fitting from the middle

- performance of the optimizer

- the quality of the fitting (how far away do we fit,)

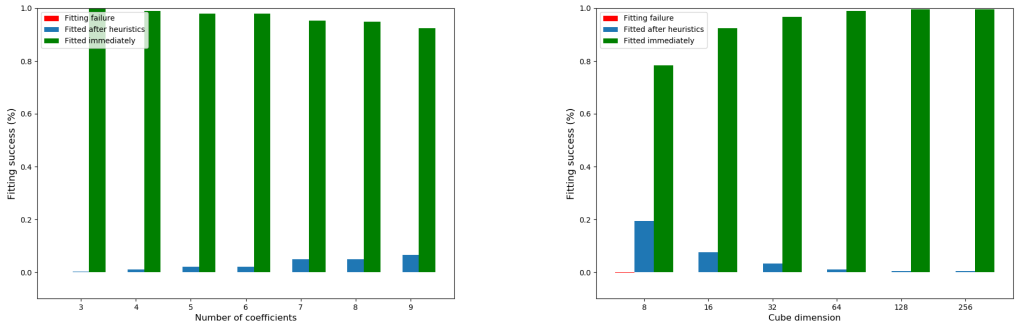
Problems with the optimizer:

It can get stuck in local minima, which usually happens in three cases and in combination of them: - threshold - we cannot control this - number of coefficients - the lower the better - cube dimension - the higher the better We tested this stucking in both when fitting lattice points, where it is crucial (we therefore added a heuristic and tested how often we get in the heuristic):

- for 9 moments (using the 4 atlases, Munsell Book of Colours, RAL, Macbeth color checker, Macbeth Color Checker SG): - cd 8 = 1 absolutely not fitted, 355 overall, 278 were fitted immediately, 69 after first, 8 required second -¿ 19.44 percent therefore after first, 78.3 percent immediately, 2.2 percent after second

- cd 16 = all fitted, 851 overall, 787 fitted immediately (92.47 percent), 56 after first (6.58 percent), 8 required second (0.9 percent)

- cd 32 = all fitted (1389), 1344 fitted immediately, 43 fitted after first, 2



(a) Examples of reconst

(b) Reconstru.

Figure 4.2: Exampod.

required second

- cd 64 = all fitted, 1648 fitted immediately, 15 fitted after first, 2 required second

-cd 128 - 1861 fitted immediately, 8 fitted after first, no second

-cd 256 - 1911 fitted immediately, 8 fitted after first, no second

- for 6 moments: - cd 8 = all fitted, 355 overall, 326 fitted immediately (91.83 percent), 29 after first (8.16 percent), none required second

- for 3 moments: - cd 8 = all fitted, 355 overall, 352 fitted immediately (99 percent), 3 after first (1 percent), none required second

- cd 16 = all fitted, 851 overall, 849 fitted immediately, 2 after first

- cd 32 = all fitted immediately,

Other moments for cd16: 9: 787 fitted immediately (92.47 percent), 56 after first (6.58 percent), 8 required second (0.9 percent) 8: 809 fitted immediately, 40 after first, 2 after second 7: 1 unfitted at all, 810 immediately, 39 after first, 2 after second 6: all fitted, 833 immediately, 18 after first, 0 after second 5: 833 immediately, 18 after first 4: 842 fitted immediately, 6 after first, 3 after second 3: 849 fitted immediately, 2 after first

We also tested stucking in the normal fitting.

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

Conclusion

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[41] Gaurav Sharma, Wencheng Wu, and Edul N Dalal. The ciede2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *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*, 30(1):21–30, 2005.

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

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

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

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

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

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

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

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

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

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

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

A. Software user guide

B. Attachments

B.1 Delta E error caused by moment sampling

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