An RGB to Spectrum Conversion for Reflectances

Brian Smits[*]
University of Utah

January 21, 2000

Abstract

This paper presents a solution to the problem of using RGB data created by most modeling systems in a spectrally-based renderer. It differs from previous methods in that it attempts to create physically plausible spectra for reflectances. The method searches the metamer space for a spectrum that best fits a set of criteria. The results are used in an algorithm that is simple and efficient enough to be used within the rendering process for both importing models and for texture mapping.

1 Introduction

The desire for accuracy and realism in images requires a physically-based rendering system. Often this can mean using a full spectral representation, as RGB representations have limitations in some situations[4]. The spectral representation does come at some cost, not the least of which is that most available models and model formats specify materials and lights in terms of RGB triples. Using these models conveniently requires the conversion of RGB values into spectra [2]. The conversion process should result in spectra that are suitable for use in a physically based rendering system and be efficient, both for data import and for texturing operations.

The main difficulty in converting from RGB to spectra is that for any positive RGB color there are an infinite number of spectra that can produce that color. These spectra are known as *metamers* of each other. Although these spectra all produce the same color, when they are used as reflectances and illuminated by non-constant spectra the resulting spectra are usually not metamers. There is not enough information to determine which of the metamers for a particular color is the "right" metamer. Instead, the goal is to pick a "good" metamer.

In the physical world, spectra are functions defined over some continuous range of wavelengths. This is an impractical representation for a rendering algorithm, so a finite dimensional representation must be chosen. These range from piecewise constant basis

[*]bes@cs.utah.edu

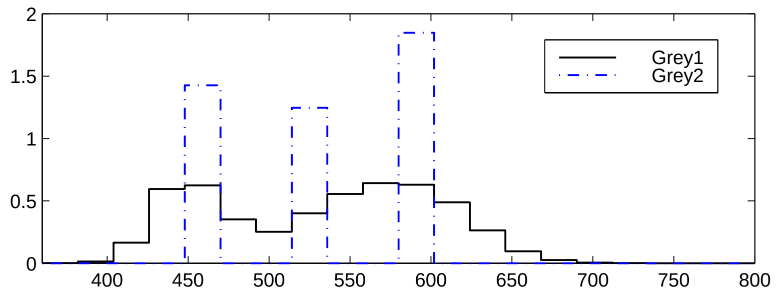

Figure 1: Two spectral representations for grey, G_1 and G_2.

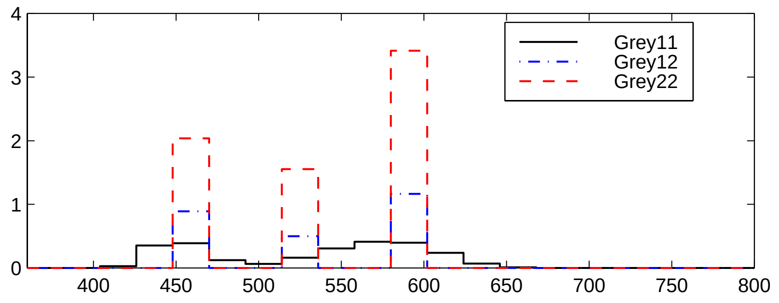

Figure 2: The three products, G_1G_1, G_1G_2, and G_2G_2.

functions to optimally chosen point samples [6] to optimal basis functions for a specific scene [8]. This paper uses piecewise constant spectra, as the relatively narrow and non-overlapping support of the basis functions make them efficient to use and capable of representing fairly saturated colors. Since they cover every wavelength in the visible spectrum, the filtering issues raised by point samples are handled automatically. For the rest of this paper, a spectrum will refer to a vector of weights for a set of piecewise constant basis functions.

The linear transform from a spectrum to the RGB color space is defined by the standard XYZ matching functions and the matrix converting from XYZ to RGB as described in many sources (e.g. Rogers [11], Wyszecki and Stiles[16]). Note that the XYZ matching functions are defined from 360nm to 800nm, while in practice rendering systems usually use a significantly smaller interval in order to focus on the region where the eye is more sensitive.

Most of the colors that will need to be transformed while reading in a model, or while doing texture mapping are reflectances. Reflectances should all be between zero and one in order to satisfy physical laws. Reflectances of natural materials also tend to be smooth[5]. Additionally, spectra with a minimal amount of variation are better for computational reasons. This can be seen by looking at two possible representations for 50% grey in Figure 1. Although both are the same color, they perform very differently in a rendering system. The results of multiplying the two curves can be seen in Figure 2. None of the resulting curves are grey or average 25%. A global illumination algorithm using Grey2 would diverge very quickly, while a rendering system using Grey1 or a combination would produce unintuitive results and visually disturbing color shifts. These cases are both possible in the real world, however, they don't match our intuition of a grey surface illuminated by a white light resulting in a grey appearance. Minimal variation over the spectrum, with peaks and troughs being as wide and as close to average as possible, reduce these problems.

Previous approaches to converting RGB colors to spectra have not focused on creating spectra that are suitable for reflectances. One approach [3] takes the spectra for each of the three phosphors and weights them by the RGB triple. This obviously works, however the spectra for the phosphors are quite irregular, resulting in reflectances that are significantly greater than 1, and a white RGB results in a spectrum that is far from constant. Other approaches by Glassner [2] and Sun *et al.*[14] choose three smooth functions and use linear combinations of those functions based on the RGB values. These approaches can result in spectra with negative regions. A last and simplest approach was used by Shirley and Marschner [12]. This approach simply uses the blue value for the first $n/3$ coefficients, the green value for the next $n/3$, and the red for the last. The number of basis functions were constrained to be a multiple of 3. This method does not produce a spectrum that converts to the right color, but it can be fairly close, and results in spectra that are guaranteed to be valid reflectances. The method is tailored to spectra defined over the interval [400,700] and does not work satisfactorily for intervals significantly different than this. The resulting curves also did not look very plausible. Ideally, a good conversion for reflectances would have the bounded properties of the Shirley and Marschner approach and the smoothness and exact match properties of the method described by Glassner.

2 Algorithm

Given a spectra $s \in \mathcal{R}^n$ and a standard RGB color c, the XYZ matching functions and an XYZ to RGB transform define a transformation matrix $A \in \mathcal{R}^{3\times n}$ from spectra to RGB. We are looking for a good s such that

$$As = c.$$

This is an under constrained problem as there are an infinite number of solutions. Let s_0 be a solution and $N = \text{null}(A)$ be the matrix representing the null space of A ($N \in \mathcal{R}^{n\times n-3}$ and every column of N is orthogonal to each row of A), then

$$s^x = s^0 + Nx$$

is a valid solution for all $x \in \mathcal{R}^{n-3}$. The set of s^x are all metamers of each other. The goal is to choose a good metamer, which can be defined as $\forall i \; s^x_i >= 0$ and s^x is smooth. Smoothness can be defined in many ways. After some experimentation, it seemed that the best solution was to look at the differences between adjacent basis functions as a measure of smoothness. This can be done by minimizing the two norm of the difference vector $d \in \mathcal{R}^{n-1} : d_i = s_i - s_{i+1}$, a standard approach in vision

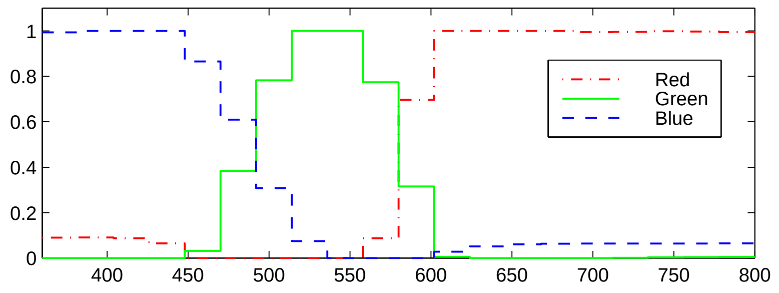

Figure 3: The spectra for red, green, and blue.

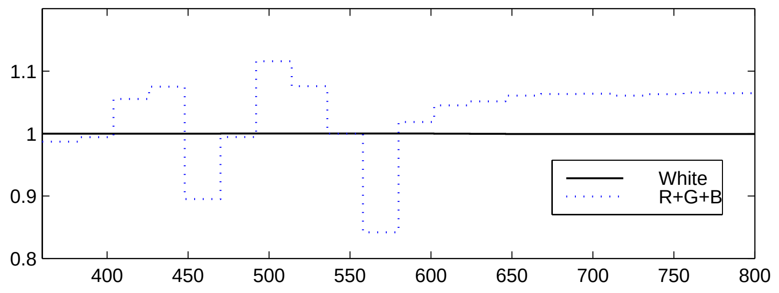

Figure 4: The spectrum for white versus the sum of the spectra for red, green, and blue.

when using snakes to fit a curve[15]. Constant regions are considered ideal, and if the function needs to vary, several small steps are better than a single large step. This tends to create wide peaks and valleys, and works to minimize the overall difference. This fits the definition of "good" as described in Section 1. In order to make these spectra valid reflectances, the values are constrained to be greater than zero. Ideally a constraint keeping the spectrum less than one would also be added, but as will be explained later, this can be impossible to satisfy for some implementations of the visible spectrum. Instead, a penalty factor is added to the cost function when any value of the spectrum is greater than one.

Although all of the previous steps can be done very simply and efficiently in Matlab[7], they are still too costly to do for every conversion. The immediate solution to this is to exploit the linearity of the conversion and create spectra for red, green, and blue. The plot of these can be seen in Figure 3. The immediate problem with this approach can be seen by looking at the sum. Its color is white, and it should be as smooth as possible. As can be seen in Figure 4, the conversion for white looks much better than the sum of the three primaries. A reasonable solution is to continue to exploit the linearity of the conversions, and create spectra for red, green, blue, cyan, magenta, yellow, and white. The spectra for cyan, magenta, and yellow can be found in Figure 5. In the same way that the white spectrum is better than a sum of red, green, and blue, the cyan, magenta, and yellow spectra are better than the sum of the appropriate red, green, and blue spectra. Ideally, the conversion would use as much of the constant white spec-

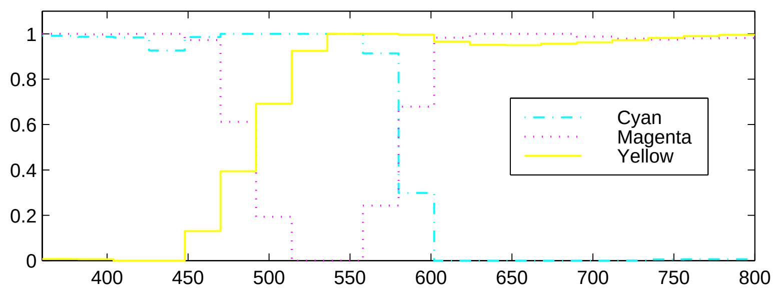

Figure 5: The spectra for cyan, magenta, and yellow.

tra as possible, followed by the secondary colors, and mix in primary colors only as needed. Any color can be expressed as a sum of white, plus one of cyan, magenta, or yellow, plus one of red, green, or blue. The conversion process works by subtracting off as much of the wider spectra (first white, then either cyan, magenta, or yellow) as possible before converting the remainder using the red, green, or blue spectra. This can be expressed in pseudocode as follows for a spectrum where red is less than green and blue:

```
Spectrum **RGBToSpectrum**(red,green,blue)
    Spectrum ret = 0;
    **if**(red $\leq$ green && red $\leq$ blue)
        ret += red * whiteSpectrum;
        **if**(green $\leq$ blue)
            ret += (green - red) * cyanSpectrum;
            ret += (blue - green) * blueSpectrum;
        **else**
            ret += (blue - red) * cyanSpectrum;
            ret += (green - blue) * greenSpectrum;
    **else if**(green $\leq$ red && green $\leq$ blue)
        ⋮
    **else**        // *blue $\leq$ red && blue $\leq$ green*
        ⋮
```

The other two cases are similar.

3 Unresolved Issues

The algorithm described in the previous section gives a “good”conversions from an RGB triple to a spectrum, but unfortunately that is only part of the solution. This section discusses some of the issues not addressed by the algorithm and gives options that may partially address the problems.

3.1 Convergence Problems

It is often possible to get into situations where it is impossible to create spectra between zero and one for the various components (usually the red curve). For spectra defined from 400nm to 700nm and an XYZ to RGB matrix based on Sony monitor data with a white point set at (.333,.333)[1], it is not possible to get the red curve below one. The area of the spectrum contributing solely to red is not large enough. Even though all the matching functions are low past 700nm, the red is still significant and more importantly, the other two matching functions are negative. Two factors are responsible for causing curves to rise above one: the width of the visible spectrum, and the chromaticity coordinates for the primaries used to create the XYZ to RGB matrix. The width of the spectrum can be increased, however for small numbers of spectral samples (9 or less), uniform bin size results in poor sampling of the regions of the spectrum where the matching functions are changing more frequently. As the primaries get more saturated, this problem gets worse. Consider three monochromatic primaries. The only way to (almost) match these is to have a zero for every basis function except the one containing that wavelength. For the sum of the red, green, and blue curves to have the same intensity as a constant unit spectrum, the value of the single non-zero coefficient must be much larger than one.

3.2 White Point Considerations

The white point of a monitor is the color produced by an RGB triple of (1,1,1), and can vary significantly between different types of monitors. The whiteness constancy effect causes us to perceive these different colors as white, as long as they can't be easily compared with each other, such as when two different monitors are placed next to each other. Because of this effect, the conversion to and from spectra should be set up with a white point of (.333, .333) for most uses. This is the only way white can be a constant spectrum.

There are three other options that may make sense in various circumstances. First, if spectral data is mixed with RGB data it may be more reasonable to convert to spectra using a white point of (.333,.333) and convert from spectra using the correct white point. The RGB data won't match exactly, but should be reasonably close, and the reflectances will be valid.

If achieving an accurate match is important and spectral data will also be used, the correct white point can be used for conversions in both directions. Spectra may no longer be automatically bounded by one due to white no longer being constant, a problem discussed in the next section. Although using the same white point makes the conversion invertible, since white is no longer constant, white light shining on a white object will exhibit color shifts. A variation on this option is to use the correct white point for converting materials and using a white point of $(.333, .333)$ for lights. This will allow matching the RGB values of a conventional renderer for white lights and getting very close for others.

[1] $R_{xy} = (.625, .34), G_{xy} = (.28, .595), B_{xy} = (.155, .07), W_{xy} = (.333, .333)$

3.3 Scale Factor for Reflectances

The algorithm described above ignores scaling issues in the conversion from spectra to RGB, which is a function of the area under the XYZ matching functions (roughly 106), conversion to lumens, and the white point luminance chosen in the XYZ to RGB transform. This means that a grey RGB triple (0.7,0.7,0.7) may be converted into a constant spectrum with values nowhere near 0.7. This is clearly not going to work correctly. This is easily solved by making the white point luminance of XYZ to RGB transform exactly equal to the area under the XYZ matching functions weighted by the lumens conversion factor. This means a constant spectrum with value 1 converts to (1,1,1), and therefore the inverse will have the right range. If this is not done, or if chromaticity coordinates or white points must be set so that the various curves have values greater than one, it may be necessary to scale or clip the resulting spectrum if values between zero and one are desired. This will introduce some error for those spectra.

Another issue has to do with specifying appearance rather than reflectance. Most users of modeling software set white walls to a reflectance of 1, whereas white paint actually has a reflectance that is usually below 0.8. This is because they want a white wall, not a grey wall and 0.8 definitely looks grey. This can cause significant problems in global illumination systems as solutions will tend not to converge. One solution that works in practice is to scale all incoming data by a factor of about 0.8 (roughly the reflectance of ANSI safety white[10]).

3.4 Gamma Correction

Until this point, gamma correction has been ignored. In many ways it is a separate and complicated problem all its own[9]. The spectral conversion process should be done in the physical intensity domain since it relies on linearity properties that are no longer valid after the gamma correction used to get display values. If gamma is correctly treated as a display issue, there are no problems. More commonly, however, the modeling package will not do any (or enough) gamma correction, and the spectral renderer will do gamma correction before displaying the result. In this case, the RGB data in the model are display values, and an inverse gamma correction should be done before the spectral conversion process in order to get physical intensities.

4 Examples and Discussion

The algorithm described in Section 2 was tested on a simple environment with many highly saturated objects. The three images on the walls are texture maps of the three bright faces of the RGB color cube (R, G, or B = 1). The lights used for the first test (Figure 6) are white. The white point for the RGB conversion was set at (.333,.333) as described in Section 3. Appendix A gives the exact data for the conversion. The left image in Figure 6 was rendered with the ray tracer inside Maya[1], while the right image was rendered with a spectrally and physically based renderer. In both cases, only direct illumination was used. The spectral renderer computed the texture maps

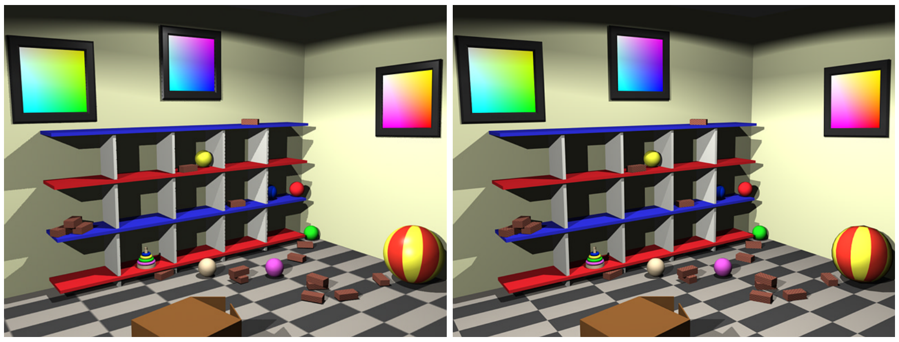

Figure 6: Rendering of direct illumination using RGB renderer (left) and spectral renderer(right).

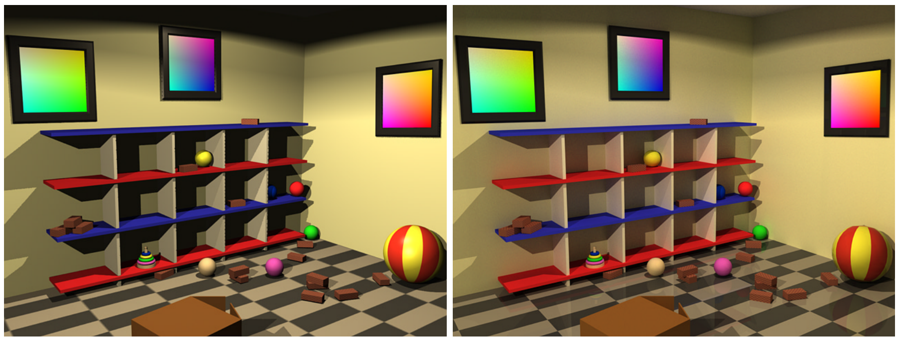

Figure 7: Rendering with non-white light sources using RGB renderer with direct lighting (left) and spectral render with direct and indirect illumination (right).

in RGB space and converted to a spectrum when the value was needed by the shader. The two pictures are not exact, but are very close, and the colors are indistinguishable visually between the two images. Because the light sources are white, and only direct illumination is being rendered, the mathematics say the values should match exactly. Differences are likely due to variations in BRDF models and lighting calculations between the two renderers, as well as the final scaling done to convert to displayable intensities. As the goal of this paper is to be able to bring RGB data into a spectral renderer while preserving appearance, the minor differences are not significant.

The second set of images (shown in Figure 7) were created with lights that are not white. In addition, indirect illumination was computed by the spectrally-based renderer (using standard Monte Carlo methods). Although there is now a visual difference, these differences are unavoidable. As discussed in Section 1, once non-constant spectra are multiplied there is no right answer. The differences are not large enough to hinder color selection in standard RGB modeling/rendering systems, or to create difficulties in importing models with RGB data.

The conversion method presented here is guaranteed to be invertible and is efficient enough to be used within the texture stage of the renderer. It produces plausible reflectances that capture the appearance of the surfaces even under non-white illumination and with multiple interreflections. Although this paper uses uniformly sized bins to represent the spectra, that fact was not used by the algorithm. The only critical restriction on the representation is that a matrix can be used to convert the spectral representation to RGB values.

It is possible to find situations where the results do not match reasonably well, such as highly saturated lights or transparent filters with (relatively) selective transparency, however, that is almost certainly unavoidable with any method. While the resulting spectra may look plausible and have nice properties within global illumination algorithms, there is no guarantee they are representative of any real material. They simply allow the same renderer to render models with RGB data, spectral data, or both.

5 Acknowledgments

I would like to thank Peter Shirley for helpful discussions during the writing of this paper. Thanks to Peter Shirley, Amy Gooch, Simon Premoze, and the reviewers for their comments. Also, thanks to Alias/Wavefront for providing Maya. This work was supported by NSF awards 97-31859 and 97-20192.

A Data

The data presented here is based on a 10 bin spectral representation from 380nm to 720nm. The bins are all equal size. The data for the XYZ to RGB conversion are $R_{xy} = (.64, .33), G_{xy} = (.3, .6), B_{xy} = (.15, .06), W_{xy} = (.333, .333), W_Y = 106.8$. The phosphor chromaticities are based on the IUT-R BT.709 standard as discussed by a W3C recommendation describing a default color space for the Internet[13]. The Matlab scripts used to generate the data may be found at the web site listed at the end of the article.

bin	white	cyan	magenta	yellow	red	green	blue
1	1.0000	0.9710	1.0000	0.0001	0.1012	0.0000	1.0000
2	1.0000	0.9426	1.0000	0.0000	0.0515	0.0000	1.0000
3	0.9999	1.0007	0.9685	0.1088	0.0000	0.0273	0.8916
4	0.9993	1.0007	0.2229	0.6651	0.0000	0.7937	0.3323
5	0.9992	1.0007	0.0000	1.0000	0.0000	1.0000	0.0000
6	0.9998	1.0007	0.0458	1.0000	0.0000	0.9418	0.0000
7	1.0000	0.1564	0.8369	0.9996	0.8325	0.1719	0.0003
8	1.0000	0.0000	1.0000	0.9586	1.0149	0.0000	0.0369
9	1.0000	0.0000	1.0000	0.9685	1.0149	0.0000	0.0483
10	1.0000	0.0000	0.9959	0.9840	1.0149	0.0025	0.0496

References

[1] ALIAS|WAVEFRONT. *Maya v. 1.5*. Toronto, Canada, 1998.

[2] GLASSNER, A. S. How to derive a spectrum from an RGB triplet. *IEEE Computer Graphics and Applications 9*, 4 (July 1989), 95–99.

[3] GLASSNER, A. S. *Principles of Digital Image Synthesis*. Morgan-Kaufman, San Francisco, 1995.

[4] HALL, R. *Illumination and Color in Computer Generated Imagery*. Springer-Verlag, New York, 1989. includes C code for radiosity algorithms.

[5] MALONEY, L. T. Evaluation of linear models of surface spectral reflectance with small numbers of parameters. *Journal of the Optical Society of America* (1986).

[6] MEYER, G. W. Wavelength selection for synthetic image generation. Technical Report CIS-TR-87-14, University of Oregon, Nov. 1987.

[7] PÄRT-ENANDER, E., SJÖBERG, A., MELIN, B., AND ISAKSSON, P. *The Matlab Handbook*. Addison-Wesley, Harlow, England, 1996.

[8] PEERCY, M. S. Linear color representations for full spectral rendering. In *Computer Graphics (SIGGRAPH '93 Proceedings)* (Aug. 1993), J. T. Kajiya, Ed., vol. 27, pp. 191–198.

[9] POYNTON, C. *A Technical Introduction to Digital Video*. Wiley and Sons, 1996. http://www.inforamp.net/ poynton/Poynton-T-I-Digital-Video.html.

[10] REA, M. S., Ed. *The Illumination Engineering Society Lighting Handbook*, 8th ed. Illumination Engineering Society, New York, NY, 1993.

[11] ROGERS, D. F. *Procedural Elements for Computer Graphics*. McGraw-Hill, 1985.

[12] SHIRLEY, P., 1998. Personal Correspondance.

[13] STOKES, M., ANDERSON, M., CHANDRASEKAR, S., AND MOTTA, R. A standard default color space for the internet - sRGB, 1996. http://www.w3.org/Graphics/Color/sRGB.

[14] SUN, Y., FRACCHIA, F. D., CALVERT, T. W., AND DREW, M. S. Deriving spectra from colors and rendering light interference. *IEEE Computer Graphics and Applications 19*, 4 (July/August 1999).

[15] TRUCCO, E., AND VERRI, A. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, New Jersey, 1998.

[16] WYSZECKI, G., AND STILES, W. S. *Color Science: Concepts and Methods, Quantitative Data and Formulae*. Wiley, 1982.