# A Spectral Engine for Visualization

Steven Bergner, Torsten Möller, Mark S. Drew

{sbergner,torsten,mark}@cs.sfu.ca, http://gruvi.cs.sfu.ca/
Graphics, Usability, and Visualization Lab
Simon Fraser University

**Abstract**

*Full spectra allow the generation of physically correct rendering of a scene under different lighting conditions. Here, a tool is developed to design a palette of lights and materials having certain properties, such as selective metamerism or colour constancy. The mathematical underpinnings of the tool consist of two parts. Firstly, we make use of a method for optimally characterizing full spectra of lights, surfaces, and transmissioive materials by forming eigenvectors of sets of these and then transforming to an intermediate space in which spectral interactions reduce to simple component-wise multiplications. Secondly, we introduce appropriate optimization functions to create new sets of spectra from old, adding to sets of actual spectra so as to produce lights and reflectances that collude to generate spectral transfer function entries for bringing out or hiding parts of a graphics rendering.*

Technical Report SFU-CMPT-TR2004-06

# 1 Introduction

Light interacting with matter is the basis of imaging, in combination with how the human visual system operates. But whereas in graphics it is common to simply use RGB values for all interactions, it is well known that such a drastic subsampling of full spectra can lead to disastrous errors [13]. In truth, both surface graphics, including transmissions, and for volume graphics as well, we should be using at least some good approximation of full spectra.

The motivation for leaning towards such a physically-based rendition of reality is obviously that we could in principle produce photorealistic imagery, or even display images for situations not yet correctly imaged (e.g., it seems that Mars is actually butterscotch in colour, a fact not easily determined from the RGB images taken on its surface [14]).

However, it is not just that physics-based rendering supplied accurate models of reality. As well, full spectra provide a much wider design palette for configuration of scenes that can guide a user's interest and attention. For example, suppose we wish to interactively fade to black some particular elements of a volume rendering (see [2]). A surface spectral reflection function that produces a zero RGB triple under a particular light is called a *metameric black* [21] under that light. Under another light the surface appears coloured, not black. But such a spectrum must necessarily contain some negative components; and while such a non-physical spectrum can be easily accommodated under a full-spectrum scheme, barring zero RGB objects we cannot produce such using a tradition graphics scheme, wherein light-surface interactions are modelled by simply multiplying RGBs component-wise. We call the simple scheme, component-wise multiplication, a *factor model*, or more specifically, an *RGB factor model*).

The obverse of the situation involving metameric blacks is a situation in which a user navigates a scene by changing the lighting, but some materials retain their colour as the light changes; and in fact we design materials and lights in such a way that several materials all have the same colour in this process — we can refer to this situation as *colour constancy*. Materials that have the same colour under one light but whose colours split apart when the light changes are called *metamers* [21]. Then clearly if one other material is designed to appear, as it were, out of thin air, as the lights change, with other materials staying the same, we have a powerful design tool for the exploration of data sets.

However, efficiency gets in the way. For supposing that we sample the visible spectrum from 400 to 700nm in 10nm steps, with resulting 31-component spectral representation. Then instead of our usual 3-vector component-wise multiplication, we have a set of multiplies that is an order of magnitude greater. This will not hold, in a ray-tracing scenario where we may have billions of such interactions, and is even taxing in a raycasting setting. So we have need for a much more compact representation, which still has the benefits of full-spectrum power. Fortunately, we just such a representation is available [7] that accurately represents full spectra using about twice the number of "colours" (5 to 7 basis coefficients, here). And moreover, the new representation has the virtue that interactions can again be carried out using a component-wise multiplication, this time as as *spectral factor model*.

Thus the tool at hand is a design factory for lights and spectra. We must choose a set of quality criteria for spectra that are to be designed, and here we show how to incorporate these into an overall optimization objective function for minimization via least-squares.

Since we wish to make use of the new method [7] that best promotes the spectral factor model, we wish to work within a low-dimensional linear subspace spanned by the basis set. So one concern is carrying out an optimization that adheres to this subspace as best as possible. Other issues are generating spectra which generates such approximations without *perceivable* errors. And as well, we may wish to generate physical spectra having nonnegative components. Finally, smoothness is also a design goal, since real spectra are

most often smooth. All such design goals should also be optional, since e.g. smoothness could be switched off if we decide to utilize fluorescent lights with spike spectra.

The necessary computations can be done efficiently in software, but realization in hardware is also possible. This colour model is a crucial step to make spectral rendering affordable for volume rendering and real-time graphics in general.

To make spectra more usable for computer graphics a design method is proposed in § 3. The goal is to generate a palette of lights and reflectances, which have certain colours in specific combinations. In connection with spectral transfer functions it is possible to provide the user with a whole setup for a spectral scene.

# 2 Linear Colour Models and a Spectral Factor Model

To obtain a low-dimensional representation of lights and surfaces, the most accurate representation that accounts best for variance is of course some type of principal component representation (see [17] for the use of such in a surface graphics setting). However, such a representation, $m$-component, say, implies that every light interaction will necessitate an $m \times m$ matrix multiply. The method in [7] obviates this via a so-called "sharpening" transform. Motivated by human colour vision, sharpened sensors are now the norm in colour cameras (see, e.g., the sRGB space [1]). Sharpening is a simple $m \times m$ preprocessing step for putting all calculations into a basis subspace that is a linear transform away from the original sensor space [9].

For our purposes here, we wish to "sharpen" the basis set for spectra. The optimal basis set to use is that derived from *colour signals*, i.e., products of lights and reflectances [5]. Then a preprocessing step of a constrained optimization can deliver the best basis space in which to work [6]. As well, if we choose to use fluorescent lighting, then we can develop a "specialized" basis set that best describes the special spike spectra involved. In either case, the advantage is that, while we lose none of the expressive power of a principal component basis, when light participates in interactions the result in the subspace is well-modelled via a simple spectral factor model.

We describe the process of going over to a simple spectral factor model below, in § 2.2. Rendering proceeds, then using coefficients with respect to this new basis set. Only at the last step do we actually descend to 3-dimensional RGB triples. This allows us to produce real-time spectral volume rendering. Moreover, since basis coefficients for the light itself can be inserted anywhere in the string of interaction events, if we place the light coefficients into the list as the last step, and in fact make rendering to RGB part of this last step, we make the exploration of volume data interactive in real time, via a light slider that effects such a "post-illumination" strategy [2].

An important issue when considering spectral rendering is the use of more elaborate illumination models. Gondek et al. [11] acquire spectral bidirectional reflectance distribution functions (BRDFs) for layered paint by analyzing simulated micro-structure as a pre-processing step. This is an interesting extension to analytic models for interference colours or diffraction [19, 20].

## 2.1 Linear Colour Models

Linear representations for spectra of lights and reflectances are attractive for volume rendering for various reasons. If set up properly all illumination calculations can be performed in a linear subspace of reduced dimensionality. The basis can be specialized for a set of representative spectra, thus improving accuracy. Whereas in general each illumination computation is a matrix multiplication, in case of the *spectral factor*

*model* this is reduced to a simple componentwise multiplication. This feature makes a treatment using the new type of basis very similar to computation with RGB component, except extended in dimensionality from 3 to a higher dimension, about 5 to 7 elements. Details on this model are contained below in § 2.2. The property of componentwise multiplication makes it very easy to integrate the extended spectral colour model into previous RGB-based rendering algorithms.

The basic idea of using a linear model is to describe a spectrum $C$ (e.g., a colour signal [21] formed from the product of light and surface spectral reflectance function) by a linear combination of a set of basis functions $B_i$ weighted by coefficients $c_i$.

$$C(\lambda) = \sum_{i=1}^{m} c_i B_i(\lambda) \tag{1}$$

The essential question is how to choose the *right* basis. Marimont and Wandell [15] discuss different approaches to finding a basis that minimizes *perceivable* error in the sensor responses. Peercy [18] elaborated a framework for using linear colour models in illumination calculations. The "quality" of a particular basis can be summarized as the tradeoff between accuracy and computational complexity. There are two different approaches to this issue. We can form *specialized bases* tailored to the particular set of spectra in a scene. Then these spectra have only minimal error when projected to the subspace spanned by the linear model. Spectra differing from the prototype spectra may have larger error from projection. Alternatively, *global bases* are suitable for a wider set of spectra. For instance, using a Fourier basis only assumes some degree of smoothness of the modelled spectra. Certainly, all spectra that are smooth enough will be well represented; however, a Fourier representation may have negative coefficients for valid physical (i.e., nonnegative) spectra, making the model problematic to use in a hardware implementation.

In order to derive general relationships between spectral power distributions (SPDs) and their linear representation it is useful to represent continuous functions as discrete vectors. If a *full spectrum* consists of equidistant point samples taken over the visible range from 400 to 700nm at 10nm intervals, we have a 31-vector representation. Then Eq. 1 can be rewritten in terms of a *full* (point sampled) 31-component spectrum $\vec{C}$, a $31 \times N$ matrix $\mathbf{B}$ comprised of the set of basis vectors, where $m$ is the subspace dimension (usually 7), and an $m$-vector $\vec{c}$ that holds the low-dimensional coefficients $c_i$:

$$\vec{C} \;=\; \mathbf{B}\vec{c} \quad . \tag{2}$$

Note that whereas principle components may be orthogonal, the transform to a sharpened basis usually destroys this property. Thus the coefficients $\vec{c}$ for a spectrum $\vec{C}$ can be obtained via the pseudoinverse $\mathbf{B}^+$ of $\mathbf{B}$:

$$\vec{c} \;=\; \mathbf{B}^+\vec{C} \quad . \tag{3}$$

When a light $E(\lambda)$ interacts with a surface $S(\lambda)$, the resulting colour signal $C(\lambda)$ equals a componentwise product of the two SPDs. In terms of discretized *full* spectra this is a componentwise multiplication (indicated by operator $*$) between $\vec{E}$ and $\vec{S}$:

$$\vec{C} \;=\; \mathbf{diag}(\vec{E})\,\vec{S} \;=\; \mathbf{diag}(\vec{S})\,\vec{E} \;=\; \vec{E} * \vec{S} \;=\; \vec{S} * \vec{E} \quad , \tag{4}$$

where $\mathbf{diag}(\vec{S})$ is a diagonal matrix composed from the elements of $\vec{S}$. To express this equation in terms of linear coefficients, we combine the previous equations:

$$c_i \;=\; \vec{B}_i^{+}\,(\sum_{j=1}^{m} e_j \vec{B}_j * \sum_{k=1}^{m} s_k \vec{B}_k) \quad . \tag{5}$$

The two operands inside the parentheses are $\vec{E}$ and $\vec{S}$ from Eq. 4 replaced by their subspace representatives in terms of coefficients for basis functions $\vec{B}_i$, which are the columns of the matrix $\mathbf{B}$ from Eq. 2. The result of the spectral multiplication (in the *full* space) is then projected back onto the basis functions, as in Eq. 3. The vector $\vec{B}_i^+$ denotes the $i$th row of the inverted basis $\mathbf{B}^+$. By reordering, we obtain

$$c_i = \sum_{j=1}^{m} \sum_{k=1}^{m} \vec{B}_i^+ \, (\vec{B}_j * \vec{B}_k) \, e_j s_k. \tag{6}$$

This equation can be rewritten as a matrix multiplication. To do so, we define a new matrix $\mathbf{\Lambda}$ written in terms of either $\vec{s}$ or $\vec{e}$:

$$\begin{aligned} \vec{c} = \mathbf{\Lambda}\,(\vec{s})\,\vec{e} \;&=\; \mathbf{\Lambda}\,(\vec{e})\,\vec{s}, \\ \text{with} \;\; \mathbf{\Lambda}_{\,ij}(\vec{x}) &= \sum_{k=1}^{m} \vec{B}_i^+ \, (\vec{B}_j * \vec{B}_k) \, x_k \end{aligned} \tag{7}$$

The $m \times m$ matrix $\mathbf{\Lambda}$ carries out any reflectance computation inside the linear subspace. Equation 7 also shows that a simple choice of some basis does not necessarily lead to a diagonalization of $\mathbf{\Lambda}$ . However, it is at least possible to use specialized hardware to apply this matrix at every reflection event [16].

In the following, we discuss how to modify the basis functions such that *componentwise multiplications alone, with diagonalization of* $\mathbf{\Lambda}$ , will suffice for such computations.

## 2.2 The RGB Factor Model and the Spectral Factor Model

### 2.2.1 Spectral Sharpening

The linear colour model we use here is based on an extension, to spectral bases, of the idea of using *Spectral Sharpening* in colour constancy algorithms in computer vision [10]. Spectral Sharpening is a method of making camera sensors more narrowband in terms of their spectral response function. The result is a better approximation of colours after an interaction via a simple component-wise multiplication of RGB values in the resulting new colour space.

In graphics, we usually simply multiply RGB to form colour. In surface graphics, for example, we multiply illuminant RGB times surface RGB in order to generate a product RGB. When lighting changes, we would like to model RGB change as a simple diagonal transform, with elements given by the ratios of lighting RGBs, applied to all RGBs. This is not physically accurate, but can be made more so by a preliminary matrix transformation of camera sensors to generate an intermediary colour space. For multiple light-surface interactions, however, such a simple transform will not lead to enough accuracy and we must fall back on full spectra.

The factor model for RGB 3-vectors in physics-based vision considers explicitly the role of a camera in colour formation. Here, for a spectral version of Spectral Sharpening, in place of a camera we must instead use the basis set $\mathbf{B}$. The idea of spectral sharpening is to form camera filter combinations that are more narrowband; and that is just what is also required here, but for the basis instead of the camera.

How RGB spectral sharpening proceeds is as follows. Suppose we have a $31 \times 3$ set of camera sensors $\mathbf{R}$. For illuminant $E(\lambda)$ interacting with surface reflectance function $S(\lambda)$, a physically correct RGB colour 3-vector $\vec{r}$ is given by

$$r_k \;\equiv\; \int E(\lambda)\, S(\lambda)\, R_k(\lambda)\, d\lambda, \quad k = \{R, G, B\} \quad . \tag{8}$$

An approximation is formed by

$$r_k \simeq \sigma_k \epsilon_k / w_k \ , \quad k = 1..3 \tag{9}$$

where $\sigma_k$ is the surface colour under equi-energy white light,

$$\vec{\sigma} = \mathbf{R}^T \vec{S} \tag{10}$$

and $\epsilon_k$ is the colour of the illuminant,

$$\vec{\epsilon} = \mathbf{R}^T \vec{E} \quad . \tag{11}$$

The camera scaling term is

$$\vec{w} = \mathbf{R}^T \ \vec{1}_{31} \quad , \tag{12}$$

where $\vec{1}_{31}$ is a vector of 31 ones. Borges [3] carefully considered this approximation and showed that it is accurate provided illuminants (or surfaces) are "white enough". In practice, the light can be relatively non–white and still give fairly accurate results under an RGB factor model. More importantly, for our application, it is clear that if the camera sensors **R** are *narrowband* enough then a factor model will hold.

Spectral sharpening provides just this needed bandlimiting, again simply by a judicious combination of the original, broadband, sensors (see [6] for a discussion of optimization methods for producing such a transform [1] ).

Usually, since we have the freedom of defining the intermediate, sharpened colour space, we set the $L_1$ norm of each new camera filter to unity (each column of the new **R** sums to 1). So in this case Eq. 9 simplifies to

$$r_k \simeq \sigma_k \epsilon_k \ , \quad k = 1..3 \tag{13}$$

and spectral sharpening allows us to approximate surface colour by a simple componentwise multiplication.

### 2.2.2 Spectral Factor Model

Here, we are interested in multiplying full spectra. The best we can do in an optimal fashion for representing spectra that participate in image formation is to form a principal component basis for the spectral curves [8]. Then, as we have seen, since each spectrum is represented as a sum over basis coefficients this necessarily implies a matrix multiply of current light coefficients times the next interaction spectrum coefficients. However, just as we 'sharpened' the RGBs by a matrix transform and worked in the intermediate space, here we can apply the same idea to the basis set. In a camera, we form combinations of the R, G, and B sensors that are optimally narrowband; here, we form combinations of the basis set vectors.

That is, we pre-'sharpen' the basis by a simple matrix transform and then agree to operate within the sharpened basis for all surface or volume interactions [4]. Note that no information is lost by such a transform, and accuracy to within the adopted dimensionality of the underlying finite-dimensional model is maintained.

Then indeed we can represent spectral interactions in terms of the low-dimensional coefficients (typically 5 to 7D) and calculate interactions using simple componentwise multiplication of the coefficients. Using the new basis, Eq. 6 is reduced to a simple componentwise multiply involving the coefficients of the current light, $e_i$, times those for the next interaction surface, $s_i$:

$$\vec{B}_i^{+} \ (\vec{B}_j * \vec{B}_k) \simeq \delta_{i,j} \delta_{j,k} \tag{14}$$

[1] Note that many present digital cameras customarily employ such a sharpening transform automatically, in software.

so that

$$c_i \simeq e_i s_i , \quad i = 1..m \qquad . \tag{15}$$

A set of sharpened basis vectors approximately obeys this equality [4].

Hence we have re-cast the colour interaction equation Eq. 13 with a basis in the place of camera, and spectral coefficients in the place of colour.

Since spectral light colour change is now simply one componentwise multiplication, it can be the final multiply, and hence adding light amounts to a *post-illumination* step. In the last step, descending to 3D RGB colour, we simply need a $3 \times m$ matrix multiply for each pixel to create colour on the screen, where $m$ is the dimension of 'colour' (i.e., coefficients, now) in the basis coefficient space.

#### 2.2.3 Accuracy

One problem with using linear models is that only the spectra from which the basis functions are derived are likely to be represented with good accuracy. For this we make use of singular value decomposition (SVD) for a set of given spectra, and the first $m$ significant vectors are taken to span an orthonormal linear subspace for the spectra. Other spectra, which have not been considered during the construction of this basis may be very different from their projections into that space.

Particularly in the case of fluorescent (spiky) lights or sharp cutoff bands, we should make use of a dedicated, or 'specialized', basis. Each illumination step as described in Eq. 7 makes use of a result projected back into the linear subspace and hence at every interaction the linear representation may move farther from an accurate representation of the product spectrum. This problem is especially relevant if we use multiple scattering or spectral volume absorption. The highest accuracy is achieved when only very few illumination calculations are performed. In case of a local illumination model in combination with 'flat' absorption (alpha blending), only one scattering event is considered with no further transmission events. Another technique perfect for linear colour models is sub-surface scattering [12]. This method uses only very few reflections, going on under the surface. Yet the spectral absorption (participating medium) is important for realistic results, so using spectra can greatly improve correctness; since there are only relatively few absorption events the accuracy is still acceptable.

## 3 Designing Spectra

### 3.1 Guiding Principles

Using spectra in volume rendering makes transfer functions more complex. Instead of simply assigning RGB colours as transfer function values, reflectance spectra have to be assigned to each intensity. As already pointed out, the reflectances do not completely determine the colour of a material: they need to be illuminated by a light. This provides more freedom in the design of transfer functions. To efficiently make use of this new freedom, we propose a *design* method for spectra for creating sets of both reflectances and light sources. It is also possible to have a real world setup of measured material and light spectra. Based on such a foundation of real materials, artificial materials and lights can be created to exhibit specific colours, metamerism, or colour constancy. Here, we mean by the term "colour constancy" that RGB colours *do not change* as the lights are altered. The term originates in vision research, however, and there means the less restrictive approximate persistence of human perception of colour across lighting change: grass is seen to be green, whether indoors or outdoors, even though camera RGBs may change substantially.

The input to the design process is represented as a spectral palette (see Fig. 1). The interface uses columns for lights and rows for reflectances. Every spectrum can either be input to the algorithm or remain open for redesign. For any light-reflectance combination, the user may define a design colour giving the desired colour for a reflectance under a specific light. As well, it is possible to weight a design colour according to its importance. This is necessary because in some cases no valid spectrum fulfils all design criteria. Then the weights can be used to control which compromise will be chosen.

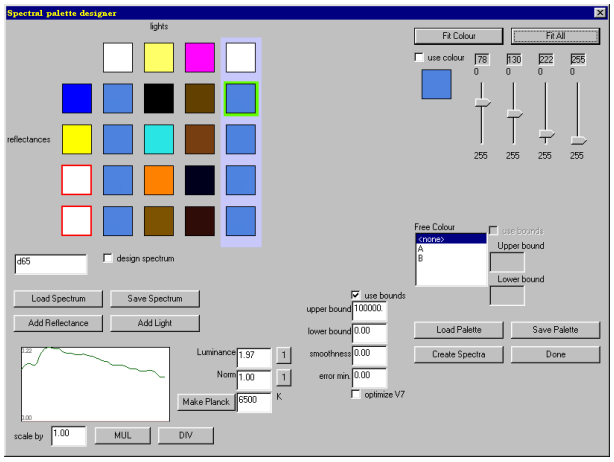


Figure 1: GUI for designing a spectral palette of reflectances (rows) and lights (columns)

The algorithm is based on finding a vector $\vec{x}$ minimizing the normal equation $\|\mathbf{M}\vec{x} - \vec{y}\|$. The matrix $\mathbf{M}$ and the result $\vec{y}$ are set up to attain the following desirable properties for an SPD:

- produces the designed colour in combination with specific reflectances/lights;
- minimal error when represented in the linear model;
- smoothness, to avoid zigzag spectra, which may be correct but are far from being realistic or 'reasonable';
- positivity, since real SPD cannot have negative magnitudes.

In the following, each of these points (except the last one) will be expressed in the form of a normal equation. These will be combined in the end to form a single minimization problem, which can be solved to find a set of spectra that fulfill all of the above conditions.

### 3.2 Matrix Formulation

For being able to specify colours in terms of 3-vector RGBs, we need $3 \times 31$ or $3 \times m$ colour transform matrices from the models 31- or $m$-component vectors to RGBs. The transform from the space of *full* spectra (see § 2.1) is

$$\mathbf{Q}_{31\to rgb} = \mathbf{Q}_{xyz\to rgb}\,\mathbf{Q}_{31\to xyz} \quad , \tag{16}$$

where the rows of $\mathbf{Q}_{31\to xyz}$ are the colour matching functions in the CIE XYZ model [21] and $\mathbf{Q}_{xyz\to rgb}$ is a hardware (monitor) dependent matrix to transform XYZ to RGB. The spectra designed using a certain monitor should produce approximately the same colours on different display hardware. Importantly,

metamers found in XYZ or a hardware dependent RGB space are also metameric in any isomorphic space. Thus we can carry out a design procedure in XYZ space and in RGB equivalently. However, we must of course take into consideration the differing gamuts of colour spaces. To avoid materials that are metameric simply because of gamut mapping (e.g., due to clamping) it is better to design RGB values that are safely inside the gamut. The matrix transforming from the $m$-dimensional linear space of the basis coefficient colour model (Eq. 2) to RGB is

$$\mathbf{Q}_{m\to rgb} = \mathbf{Q}_{31\to rgb}\,\mathbf{B} \quad . \tag{17}$$

To generate spectra we follow our general guidelines outlined above. Most important is the first criterion — design colours $\vec{d}_{ij}$ for each combination of a light $\vec{E}_j$ with a surface reflectance $\vec{S}_i$. For each possible combination we solve the minimization problem

$$\begin{aligned} &\min_{\vec{S}_i} \ \|\mathbf{Q}_{31\to rgb}^{(\vec{E}_j)}\,\vec{S}_i - \vec{d}_{ij}\|, \ \text{ with} \\ &\mathbf{Q}_{31\to rgb}^{(\vec{E}_j)} = \mathbf{Q}_{31\to rgb}\,\mathbf{diag}(\vec{E}_\mathbf{j}) \quad . \end{aligned} \tag{18}$$

Here, we are considering the design of a reflectance $\vec{S}_i$ for a given set of lights $\vec{E}_j$ and design colours $\vec{d}_{ij}$. (In case a light has to be designed instead of a reflectance, all equations can be rewritten with $\vec{S}_i$ and $\vec{E}_j$ exchanged.) Note that here we begin by working in the full spectral space, not the basis subspace.

The colour computation can also be performed in the linear subspace. Working within the linear model, we use $\mathbf{Q}_{31\to m} = \mathbf{B}^+$ to perform the conversion from full 31-vectors to $m$-vectors (Eq. 3). The matrix $\mathbf{Q}_{m\to rgb} = \mathbf{Q}_{31\to rgb}\,\mathbf{B}$ directly converts an $m$-vector to RGB. Thus the form Eq. 18 takes on when the illumination calculation is performed in the linear subspace is as follows:

$$\begin{aligned} &\min \ \|\mathbf{Q}_{31\to m\to rgb}^{(\vec{E}_j)}\,\vec{S}_i - \vec{d}_{ij}\|, \ \text{ with} \\ &\mathbf{Q}_{31\to m\to rgb}^{(\boldsymbol{E}_j)} = \mathbf{Q}_{m\to rgb}\,\mathbf{\Lambda}\,(\mathbf{Q}_{31\to m}\,\vec{E}_j)\,\mathbf{Q}_{31\to m} \quad , \end{aligned} \tag{19}$$

where $\mathbf{\Lambda}\,(\vec{x})$ is the matrix from Eq. 7 that performs the illumination calculation in the subspace. As mentioned in § 2.2, using the spectral factor model this matrix is replaced by $\mathbf{diag}(\vec{x})$. In order to have a general description we retain matrix $\mathbf{\Lambda}\,(\vec{x})$ in the following, however.

In the linear subspace, some spectra are better approximated than others. This takes us to the second criterion — reducing this error. We compare the colours resulting from the full spectra and their $m$-dimensional projections. The minimization is thus aimed at bringing this difference close to a zero 3-vector colour $\vec{0}_3$:

$$\begin{aligned} &\min \ \|\mathbf{F}(\vec{E}_j)\,\vec{S}_i - \vec{0}_3\|, \ \text{ for each } \vec{E}_j \text{ used with } \vec{S}_i \ , \\ &\mathbf{F}(\vec{E}_j) = \mathbf{Q}_{31\to rgb}^{(\vec{E}_j)} - \mathbf{Q}_{31\to m\to rgb}^{(\vec{E}_j)} \ , \quad \vec{0}_3 = \text{zero 3-vector} \quad . \end{aligned} \tag{20}$$

The third criterion is smoothness. While the first two criteria are aimed at correctness, this one is more for physical likelihood. A minimum solution for given design colours with minimum error can lead to zigzag spectra with high extrema. Smoothness is not necessary for correct results, but using it usually limits the range of a solution and yields much more 'reasonable' spectra. A commonly used indicator for roughness of a curve is the integral over the squared second derivative (and other indicators are possible). Then smoothness should be included via another normal equation in the following form:

$$\begin{aligned} &\min \ \|\mathbf{D}\,\vec{S}_i - \vec{0}_{31}\|, \ \text{ with} \\ &\mathbf{D} = \ \text{symmetric\_toeplitz}([\ -1 \quad 2 \quad -1 \quad 0 \quad \cdots \quad 0\ ]) \ , \ \vec{0}_{31} = \text{zero 31-vector} \quad . \end{aligned} \tag{21}$$

We use (31-3)=28 zeros to fill the entire row. Since we are using 31 components for full spectra, $\mathbf{D}$ is a 31x31 Toeplitz matrix. In particular it is a tri-diagonal matrix having 3-vector $\{-1,\ 2,\ -1\}$ on the three middle diagonals. The rest of the matrix is set to zero. The whole matrix $\mathbf{D}$ should be normalized by $1/(\sqrt{31}\,\|-1\ 2\ -1\|)$: the $\sqrt{31}$ takes care of the number of rows of the matrix so as not to make smoothness more important than the design colour matrices. These should themselves be normalized by $1/\sqrt{3}$ in order to have comparable importance. Each of these design criteria is expressed as a matrix in the set $\{\mathbf{Q}^{(\vec{X})}_{31\to rgb}, \mathbf{Q}^{(\vec{X})}_{31\to m\to rgb}, \mathbf{F}, \mathbf{D}\}$.

### 3.3 Optimization Function

The minimization matrix $\mathbf{M}$ is formed by stacking these criteria matrices, as shown in Eq. 22. Similarly, the associated forcing vectors are stacked to form $\vec{y}$. In the design matrix $\mathbf{M}$, the different criteria are weighted by $\omega_{\{ij|F|D\}}$ for design colours $\vec{d}_{ij}$, error matrix $\mathbf{F}$, and smoothness $\mathbf{D}$ respectively. This provides control over the convergence of the minimization. Finding a minimum error solution for $\mathbf{M}\,\vec{x} = \vec{y}$ is equivalent to solving a set of stacked equations for a set of stacked spectral vectors, concatenated into solution $\vec{x}$:

$$\begin{bmatrix} \omega_{i1}\mathbf{Q}^{(\vec{E}_1)}_{31\to rgb} & \mathbf{O}_{3\times 31} \\ \omega_F\omega_{i1}\mathbf{F}(\vec{E}_1) & \mathbf{O}_{3\times 31} \\ \vdots & \vdots \\ \omega_{in}\mathbf{Q}^{(\vec{E}_n)}_{31\to rgb} & \mathbf{O}_{3\times 31} \\ \omega_F\omega_{in}\mathbf{F}(\vec{E}_n) & \mathbf{O}_{3\times 31} \\ \omega_D\mathbf{D} & \mathbf{O}_{31\times 31} \\ & \\ \mathbf{O}_{3\times 31} & \omega_{1j}\mathbf{Q}^{(\vec{S}_1)}_{31\to rgb} \\ \mathbf{O}_{3\times 31} & \omega_F\omega_{1j}\mathbf{F}(\vec{S}_1) \\ \vdots & \vdots \\ \mathbf{O}_{3\times 31} & \omega_{pj}\mathbf{Q}^{(\vec{S}_p)}_{31\to rgb} \\ \mathbf{O}_{3\times 31} & \omega_F\omega_{pj}\mathbf{F}(\vec{S}_p) \\ \mathbf{O}_{31\times 31} & \omega_D\mathbf{D} \end{bmatrix} \cdot \vec{x} = \vec{y} = \begin{bmatrix} \omega_{i1}\vec{d}_{i1} \\ \vec{0}_3 \\ \vdots \\ \omega_{in}\vec{d}_{in} \\ \vec{0}_3 \\ \vec{0}_{31} \\ \\ \omega_{1j}\vec{d}_{1j} \\ \vec{0}_3 \\ \vdots \\ \omega_{pj}\vec{d}_{pj} \\ \vec{0}_3 \\ \vec{0}_{31} \end{bmatrix} \tag{22}$$

In Eq. 22, we show the equations for designing a reflectance $\vec{S}_i$ in the first column block and a light $\vec{E}_j$ in the second column block. All columns not belonging to a block are effectively removed from the calculation using a matrix of zeros $\mathbf{O}_{n\times 31}$. The resulting vector $\vec{x}$ will contain both 31-vectors, concatenated. Each 31-vector is subject to a single smoothness constraint involving $\mathbf{D}$. The number of column blocks depends on the number of spectra (lights or reflectances) to be designed. Each spectrum that a column spectrum can be combined with gets a row block of its own. The vector $\vec{y}$ contains the stacked solutions of the normal equations. Row blocks are omitted if the corresponding weight $\omega_{\{ij|F|D\}}$ is zero.

In the example shown, the two spectra could as well be designed using two independent equations since they do not influence each other. This can be different when 'free' colours are used: instead of defining a desired colour as part of $\vec{y}$, it can be incorporated into the matrix $\mathbf{M}$. This is useful if two reflectances are supposed to look the same under a light $\vec{E}_a$, without regard for what colour they will actually form.

Additionally, this can be combined with further design colours for a light $\vec{E}_b$. We can solve for a

weighted solution $\{\vec{d}_{ia},\ \vec{S}_i,\ \vec{S}_j\}$ of the system

$$\begin{aligned} \mathbf{Q}_{31\to rgb}^{(\vec{E}_a)}\,\vec{S}_i &= \mathbf{Q}_{31\to rgb}^{(\vec{E}_a)}\,\vec{S}_j \quad , \\ \mathbf{Q}_{31\to rgb}^{(\vec{E}_b)}\,\vec{S}_i &= \vec{d}_{1b} \end{aligned} \tag{23}$$

with colour $\vec{d}_{1b}$ known for some particular surface $\vec{S}_1$ under light $\vec{E}_b$: i.e., we ask that surface $i$ under light $a$ have the same colour $\vec{d}_{ia}$ as surface $j$ under that light, and that surfaces $i$ and 1 have equal, known, colour under light $b$.

As a stacked matrix set this becomes

$$\begin{bmatrix} -1 & 0 & 0 & & \vec{0}_{31} \\ 0 & -1 & 0 & \omega_{1a}\mathbf{Q}_{31\to rgb}^{(\vec{E}_a)} & \vec{0}_{31} \\ 0 & 0 & -1 & & \vec{0}_{31} \\ -1 & 0 & 0 & \vec{0}_{31} & \\ 0 & -1 & 0 & \vec{0}_{31} & \omega_{2a}\mathbf{Q}_{31\to rgb}^{(\vec{E}_a)} \\ 0 & 0 & -1 & \vec{0}_{31} & \\ 0 & 0 & 0 & & \vec{0}_{31} \\ 0 & 0 & 0 & \omega_{1a}\mathbf{Q}_{31\to rgb}^{(\vec{E}_b)} & \vec{0}_{31} \\ 0 & 0 & 0 & & \vec{0}_{31} \end{bmatrix} \cdot \vec{x} = \vec{y} = \begin{bmatrix} \vec{0}_3 \\ \vec{0}_3 \\ \omega_{1b}\vec{d}_{1b} \end{bmatrix} \tag{24}$$

The resulting $\vec{x}$ contains the 'free' colour in the the first three components and after that two 31-component vectors for reflectance $\vec{S}_i$ and $\vec{S}_j$. This setup becomes interesting when used with upper and lower bounds on $\vec{x}$, because then the 'free' colour can be forced to a given interval without being specified precisely.

The equation systems can be solved for a minimum error in the over-determined case or a minimum norm of $\vec{x}$ in the under-determined case. For the first case we can use the pseudo-inverse $\mathbf{M}^+ = (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T$ with $\vec{x} = \mathbf{M}^+\vec{y}$. The under-determined case has more minimum error solutions. In that case the Moore-Penrose pseudo-inverse using singular value decomposition, $\mathbf{M}^+ = \mathbf{U}^T\mathbf{\Sigma}^+\mathbf{V}$, gives a minimum length solution for $\vec{x}$. We always use the second method since it works with any configuration and reduces to the first pseudo-inverse in the first situation. The nice thing about this method is that it finds a global minimum for the criteria. Iteratively searching through a high dimensional space can be very expensive and is not guaranteed to find a solution. The drawback of global minimization is that the resulting spectra can be negative. To satisfy the positivity condition we have to carry out constrained minimization, for which we use a conjugate gradient method. To help convergence we start with a projection of the global minimum.

As implied above, some setups ask for colour combinations that are not realizable with 'valid' spectra. In software implementation it is possible to not care about negative spectra and have full freedom in designing colour combinations. If this is not desired, one solution would be to adjust the weights $\omega_{\{ij|F|D\}}$ to suppress criteria that are not too relevant for the visualization. Another way is to choose different design colours that are more likely to agree with the natural colours of the given light sources or reflectances.

# 4 Creating a Palette using the Spectral Engine

### 4.0.1 Design criteria

Having an idea of what colour constellations are desirable for visualization, we now have the task of finding spectra for lights and materials that fit these criteria. One approach is to use materials previously acquired from real world scenes. Since spectral rendering typically is an approach to increase realism in computer graphics, using real reflectances and light spectra is very desirable. In fact, it only becomes possible through the extension to a spectral rendering framework.

Nevertheless, for most applications it would be impractical to have to look for real materials and lights that fit the desired properties (metamerism, colour constancy, metameric blacks). For that purpose it is necessary to come up with a design procedure that combines real spectra with new artificial ones generating a set of lights and reflectances that fits the needs of the visualization.

For the design process each spectrum (reflectance or light) is represented by a 31-component vector that results from 10 nm equidistant samples over the visible range of light from 400 nm to 700 nm. As discussed in § 2.2 we are employing a more compact representation during rendering to gain speed and save storage. For the design process it is most useful to also have the spectra reconstituted to full 31-component vectors. That way, it is easier to control properties such as positivity or smoothness for the full spectrum. This takes us to the question of what criteria should be applied to guide the design. First of all, the design process itself is a minimization. Most of the criteria are expressed as least-squares contributions to an objective function: we are finding a vector $\vec{x}$ minimizing the normal equation $\|\mathbf{M}\vec{x} - \vec{y}\|$. By weighting and stacking all matrices $\mathbf{M}$ and vectors $\vec{y}$, the entire set of criteria becomes one minimization problem that we can solve globally using the pseudo-inverse. We obtain one solution vector $\vec{x}$ that contains all missing spectra, with the desired properties.

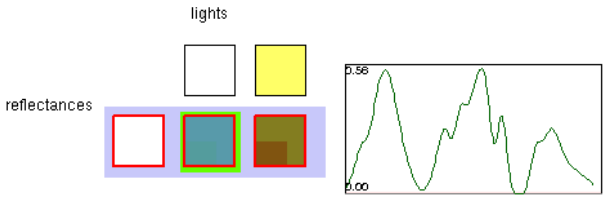


Figure 2: Initial colour setup to create an example material (represented by the colours in the lower row) under two different lights (left is standard illuminant D65, right is Tungsten). The graph on the right shows the reflectance spectrum in its current design state. It shows the amplitude of the wavelengths from 400 nm to 700 nm

The first and foremost criteria are the desired colours. Fig. 2 displays a set of colours that result from combining all lights and reflectances. Each row represents a reflectance, and each column shows the colours that one specific light generates in combination with the reflectances. Colour patches along the top show the colour of lights reflected from a flat white spectrum consisting of all 1s. There are just 2 lights shown in the figure. Colour patches down the right (just one, in the figure) show the colour of a reflectance patch illuminated by flat white light. The design matrix is the set of colours generated by all combinations of lights interacting with surfaces.

To explicitly look for spectra that produce the desired colours, and also best obey Eq. 15, we add a criterion that minimizes the difference between the colours resulting from the two colour models (the full 31-component vectors and the spectral factor model). An example is shown in Fig. 3. Thus, we preferably

pick spectra whose multiplication in the subspace yields approximately the same colour as in the full space.

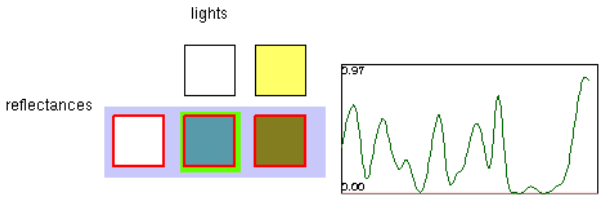


Figure 3: Shows change to previous figure after applying error minimization. The slight colour variations inside the brown box (indicating colour differences among the models) are now gone and the entire box appears as one colour. The graph on the right shows the reflectance spectrum that corresponds to the colours shown in the lower row.

If we simply apply the two criteria — correct colours and error minimization — the resulting spectra can have very extreme components. Besides causing numerical problems, spectra of this shape do not seem very reasonable. Therefore we include a criterion to favour smoothness of the spectrum. To do so, we minimize the norm of the vector of second order differences: the resulting normal equation can again be expressed in matrix form. Besides producing spectra that are less spiky, the smoothness criterion also helps convergence of the minimization. Even when performing a global minimization (using a pseudo-inverse), a small smoothness term greatly helps to improve the quality of the resulting spectra. The result of including smoothness into the design can be seen in Fig. 4.

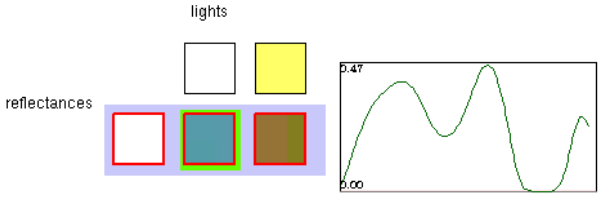


Figure 4: Applying a smoothness criterion to the previous figure. The graph on the right shows the reflectance spectrum.

Taking all quality criteria into account leads to a design matrix setup, with the set of vectors giving spectra as solution. As already pointed out, it is possible to use the pseudo-inverse of the matrix to obtain a global solution. By performing an additional constrained iterative minimization it is possible to also move all negative components into non-negatives. This way, we get physically possible spectra, fitting better with the underlying colour model. In the previous examples we have already applied this criterion. Fig. 5 shows the change in the reflectance spectrum with this constraint disabled. In the example, the colours don't change much. But in some other setups, allowing negative wavelength amplitudes greatly improves the visual quality of the resulting colours.

Our design procedure generates artificial spectra. Applying the constraint of positive wavelength amplitudes gives us potentially real materials. Unfortunately, for designing metamers, and especially metameric blacks, positivity is a very restrictive assumption leaving rather few possible colour combinations. For that reason we also allow negative spectra, if the positivity constrained minimization process does not yield satisfactory results for a given setup.

With the above procedure we can create a realistic scene setup of acquired lights and reflectances. Additionally, we are able to fill in more lights and materials that yield visual effects. Thus, this tool extends the usability of spectral computer graphics for visualization purposes.

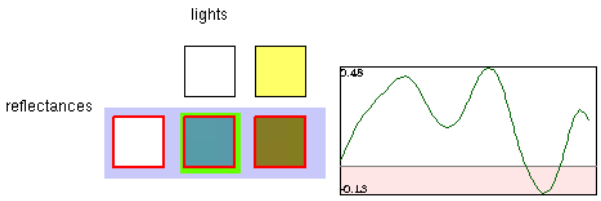


Figure 5: Shows change to previous figure after disabling positivity constraint. The graph on the right shows the reflectance spectrum.

**Design Matrix** As shown in Fig. 1, our spectral palette designer tool works within the framework of constraints described in § 3: the colour that each light-reflectance combination produces; the smoothness of a spectrum; the error in perceived colour between multiplication of spectra in the full 31D space vs. in the linear model 7D subspace; the positivity of a spectrum; and the metamerism of a combination colour with another light-reflectance combination (possibly implemented as a 'free' colour).

Note that not all criteria have to be used. In fact, since spectral creation is implemented as a minimization, a compromise will be chosen if the problem is over-constrained. It is possible to control the compromise by adjusting the weights for each criterion.

Fig. 1 shows the intial user interface, just after the latest (physical) light has been loaded — in this case, it is standard illumination D65, a medium daylight [21]. The top left quadrant of the GUI shows an RGB-displaying design matrix for a set of desired colours. Columns correspond to lights, and rows correspond to reflectances. Lights appear across the top, reflected from a flat white (equi-energy) spectrum consisting of all unity components. Reflectances appear down the left-hand side, as they appear when illuminated by flat white light. The design matrix is the set of interaction colours, which shows how lights reflected from surfaces will look as RGB. In the case of Fig. 1 the number of lights is 4 (i.e., 4 boxes across the top), and the number of reflectances is also 4 (i.e., 4 boxes down the left). This produces a total of 16 different interaction colours (colours of the corresponding colour signals).

The intial situation is that both lights and surfaces have flat, equi-energy white spectra. Adding lights or reflectances increases the size of the design matrix. For each spectrum, either light or reflectance, a mouse click on the light itself or reflectance itself highlights the entire column or row, respectively. A scaling textbox allows one to multiply or divide the scene element by a value, via scaling the lighting or reflectance. A common normalization is to set the luminance (the Y-component of the CIE XYZ triple [21]) to 1, and therefore a Luminance-Normalization scaling value textbox is provided. Alternatively, one may wish to normalize a 31-vector in the $L_2$ norm, via another textbox.

Instead of loading a physical spectrum, one may wish to make use of a theoretical SPD. The 'Make Planck' button creates a spectrum for a Planckian blackbody radiator parameterized by a given temperature $T$ (in degrees Kelvin). This SPD has the form [21]:

$$E(\lambda, T) \;=\; c_1 \lambda^{-5} \left( e^{\frac{c_2}{T\lambda}} - 1 \right)^{-1} \tag{25}$$

Using these tools, one can load or create spectra for input to the design process, or use to change spectra generated by the design. By checking the `'design spectrum'` check box (or double clicking the colour box of the spectrum) one identify a light or reflectance as being open to the optimization design process, and not fixed. This is shown by the main colour patch for that spectrum acquiring a red border.

**Design by Colours** Colours in the design matrix can be changed to the user's specification. Left-clicking a design matrix colour patch creates a green border on that patch and copies the patch RGB colour into the top-right quadrant of the GUI, which is dedicated to dealing with RGB. There, four sliders are available. Three (the rightmost three) are for altering the patch colour; the last (the leftmost one) indicates the importance (weight) of the colour in the design process. A checkbox, `'use colour'`, when checked, is the same as double-clicking that patch in the design matrix and controls whether the colour is a free colour and can be altered as the optimization sees fit. Naturally, it is not possible to use a combination colour in the design process if both its light and reflectance are to be designed. The `'fit colour'` button chooses the colour to be the one that is currently produced by the appropriate light and reflectance combination. `'Fit All'` does the same for the entire matrix.

"Fig. 2" shows an example for a design colour. The light blue background coming from the top indicates that its light is selected. The green border shows that this colour box is selected, with its colour copied into the colour-changing 4-slider area at the top-right of the GUI. The red border indicates that this patch is available as a free colour for the optimization to change.

Note that there are three colours shown inside the box. The lighter green on the right is the desired colour that has been set by the user (using the sliders). The two colours on the left show what the reflectance actually looks like under this light using the full 31-component model (top, green colour), as opposed to the colour that is seen using the reduced linear model (bottom, yellow colour). For a visualization scheme that uses the reduced dimensionality scheme for raytracing etc. [2] optimally these two colours would be equal. However, under some setups they can turn out to be very different. In that case one can increase the weight for error minimization (as explained below).

**Design Procedure** The basic steps in setting up a design problem, then, are to first load a set of (measured, physical) lights, and possibly some specific reflectance functions as well. To start the creation of lights and materials, one clicks on `'Create Spectra'` button at the lower right of the GUI. The process can take from less than a second up to several seconds depending on the number of criteria involved. Usually, enabling the positivity criterion causes the optimization to take a little more time because it is implemented as an iterative minimization. Also, the smoothness criterion adds a $31 \times 31$ matrix for each spectrum that can further delay the process.

These other criteria can be set up in the area in the middle of the interface. If positivity of spectra is desired then the `'use bounds'` box should be checked. This introduces a lower and an upper bound as indicated by the two fields below it. The other two fields are used to set the optimization weights for (a) smoothness, and (b) error minimization, for the linear model as compared to full spectra. Suitable weights to begin with are 0.001 for smoothness and 0.1 for error minimization. After each change, another click on the `'Create Spectra'` button is required, so see the changes induced on spectra.

Every time new spectra are created, some of the colour patches in the colour desin matrix show two slightly different colours, side by side. These differ by the difference between the desired colour on the left, set by the user, and the actual colour of the current light and reflectance on the right.

The right half of some fields may also be split vertically into two different colours. This shows the difference between using full 31-vector spectra (top) and the reduced linear model (the spectral factor model, bottom). Ideally, this difference is not noticeable, in which case the created spectra have an accurate representation in the subspace of the linear model.

If the last checkbox, `'use V7'`, is checked then the fitting of the colours is performed primarily using the 7-dimensional subspace, not the full space. In that case, the lower colours (those from the linear subspace

model) in the design matrix colour patch will more accurately tend to fit the desired colour. Instead, the error minimization specification textbox can be applied to drag the colours for the full 31-dimensional and the 7-dimensional subspace space together.

**Free Colours** The '`free colours`' are a method to find a metameric colour (e.g. for different reflectances under one light) without caring about the actual colour. Select the combination colours in the matrix that you want to be metameric and pick 'free colour' A. Also enable the colour to be used in the design process. If you want to follow the example of creating reflectances, also enable the according reflectances to be designed. It is possible to bound the RGB values for the resulting 'free' colour. The desired colour set by the user doesn't play a role if you use a 'free' colour. To give the design process a way to figure out a colour other than black, you have to define at least one colour for one reflectance under another light. Setting up all criteria properly is rather tricky. If it doesn't work right away, try to disable bounds (positivity) for the spectra you create. Also try to add a little smoothness or error minimization. As it is just one more criterion to minimize especially smoothness sometimes helps convergence.

# 5 Results

## 5.1 Rendering Volumes Interactively

To illustrate the efficiency of the design method and the palette tool, we consider two datasets: engine and frog. These are rendered via a *post-illumination* step: the images are first rendered with no light, or, rather, lighting having all-unity basis coefficients. Then lighting is changed. To make changing lights simple, we make use either of several light sliders, for given lights, or a multi-light 2D light dial discussed in [4]. The actual raycasting is performed once for a given viewpoint and all subsequent images for changing light can be computed in real time by simply multiplying by the new light's coefficient-to-RGB $3 \times 7$ matrix. In the examples, a "light dial" in the lower right illustrates the light setup. Blue points represent the lights and the yellow point is a light slider that can be moved by the user in order to change the weighting of the vertices, the design set of lights, that is in use in a particular rendering. Each dataset has a specific spectral transfer function, with a palette that assigns reflectances to distinct parts of the data.

In Fig. 6(a) an engine block and inner parts are metameric; as well, "smoke" (reconstruction noise) is present, but is coloured black via the metameric black mechanism. Thus it is invisible. Fig. 6(b) Has a different palette: a second light has been designed such that inner parts are now distinguishable from the engine block, which itself has kept a constant colour from the previous light. As well, the smoke appears white now. Fig. 6(c) keeps the same colour for the inner parts as in Fig. 6(b), but the engine block has now changed to the complementary colour; the smoke has again become invisible. In Fig. 6(d) the smoke is glowing red. As well, the 'purple light' has been switched off. This allows the user to slide between the lights and have the block stay green while the smoke changes colour or disappears.

Another example, this time of a frog, is shown in Fig. 7. Again, the first image shows all materials as being metameric. Notably, this colour was not chosen directly, but instead arises as a 'free colour', as described in § 3 — the design process has chosen the colour, on the basis of the optimization. In Fig. 7(b), the slider has been moved towards light 2, so that metamers begin to break apart. Additionally, light 2 makes the body of the frog go black and disappear. In Fig. 7(c), the blue slider nodes have been moved and, with a light comprising a mixture of mainly light 2 and 3, all parts have different colours. In Fig. 7(d), the slider is in the same position as in (b), but light 1 has been switched off. As a result, the body goes dark. The slight visibility results from the distant influence of the light used in (c).

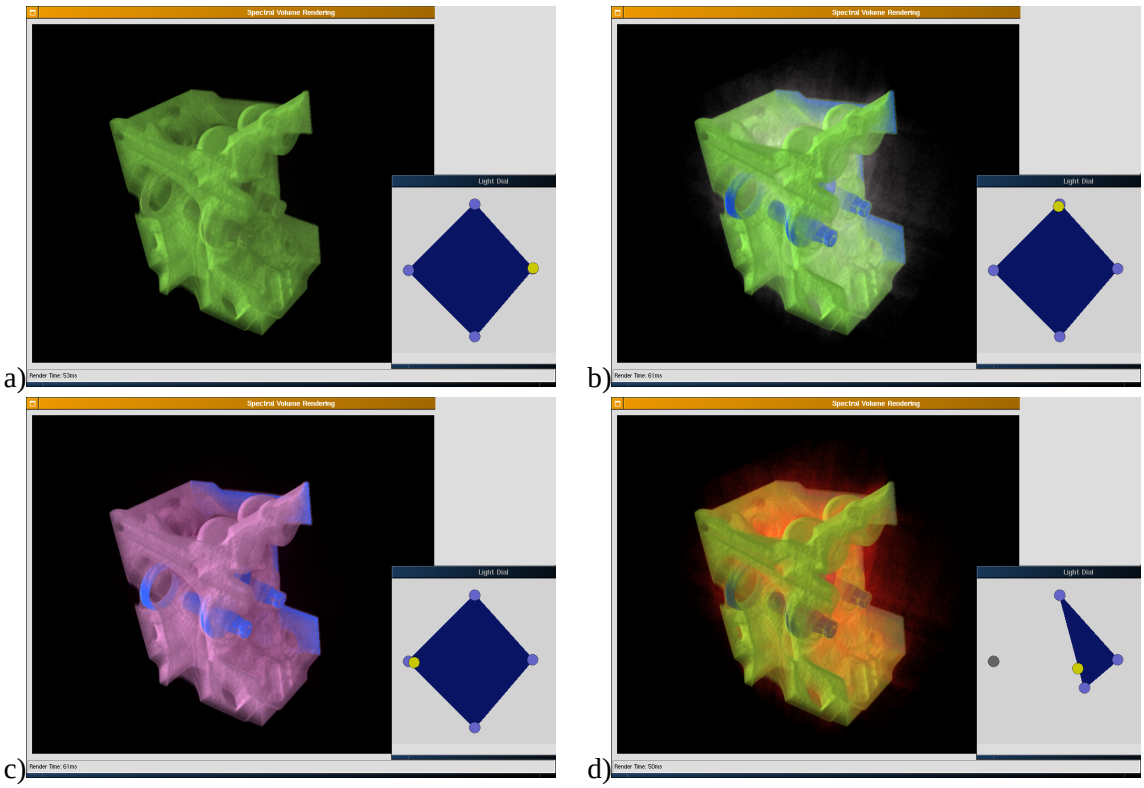


Figure 6: Engine block rendered using metamers and colour constancy. The four images in are simply re-illuminated without repeating the raycasting.

# 6 Conclusions

We have devised a spectral palette tool that allows the user to design a set of lights and reflectances such that different parts of a rendering can be enhanced, or be made to disappear, in real time and interactively. The design scheme and optimization is novel in both graphics as well as in colour science. The resulting set of spectra and colours have most utility in the visualization of volume data, but their usefulness is not restricted to this arena. In fact, since we are at liberty to inject actual physical spectra for any of the components, we can in fact design appropriate lights for viewing real physical structures, such as medical subjects.

The spectra constructed allow the user to obtain distinct impressions of the data under different lights. As used to date [2], the spectral design method helps to prepare specific illumination situations to highlight certain aspects of the data. This artificial creation of spectra is complementary to using real spectra. The design framework helps to make spectra more usable for computer graphics in general. For many applications it is essential to have control over specific properties of spectra while leaving others ‘realistic’. There is an interesting duality inherent to spectral rendering. On the one hand it provides improved realism, and on the other it enhances possibilities for artificially manipulating a scene. Both properties make spectra a powerful tool for computer graphics.

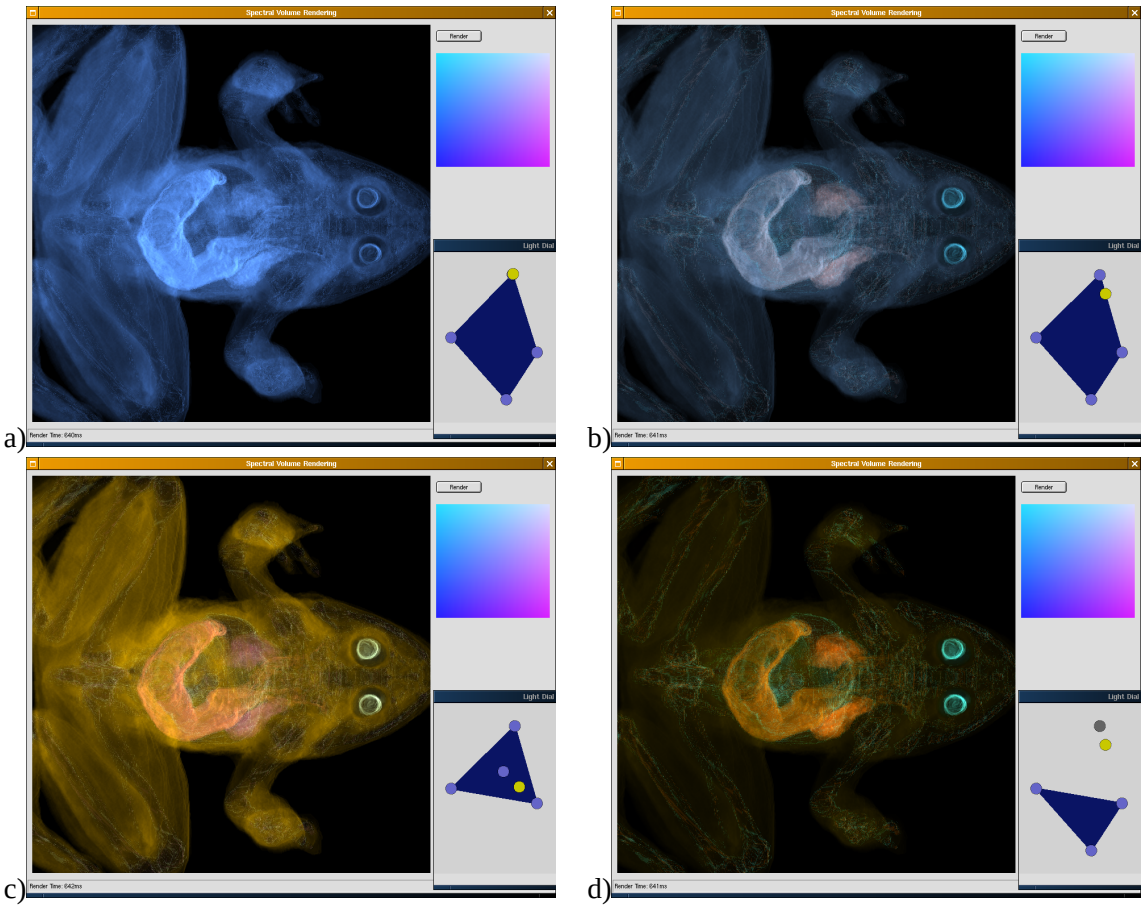


Figure 7: Frog rendered with four different materials at different light slider constellations.

## References

[1] M. Anderson, R. Motta, S. Chandrasekarand, and M. Stokes. Proposal for a standard default color space for the internet — srgb. In *Fourth Color Imaging Conference: Color, Science, Systems and Applications.*, pages 238–245. Society for Imaging Science & Technology (IS&T)/Society for Information Display (SID) joint conference, 1996.

[2] S. Bergner, T. Möller, M.S. Drew, and G.D. Finlayson. Interactive spectral volume rendering. In *IEEE Visualization*, pages 101–108. IEEE, Boston 2002.

[3] C.F. Borges. Trichromatic approximation method for surface illumination. *J. Opt. Soc. Am. A*, 8:1319–1323, 1991.

[4] Mark S. Drew and Graham D. Finlayson. Multispectral processing without spectra. Technical Report SFU-CMPT-03/02-TR2002-02, Simon Fraser University School of Computing Science, March 2002. Also: Representation of colour in a colour display system, UK Patent Application No. 0206916.9. Under review, British Patent Office.

[5] M.S. Drew and B.V. Funt. Natural metamers. *CVGIP:Image Understanding*, 56:139–151, 1992.

[6] M.S. Drew and G.D. Finlayson. Spectral sharpening with positivity. *J. Opt. Soc. Am. A*, 17:1361–1370, 2000. http://www.cs.sfu.ca/∼mark/ftp/Josa00a/sharppos.pdf.

[7] M.S. Drew and G.D. Finlayson. Multispectral processing without spectra, July 2003. http://www.cs.sfu.ca/∼mark/ftp/Josa03/specwospec.pdf.

[8] M.S. Drew and B.V. Funt. Natural metamers. *CVGIP:Image Understanding*, 56:139–151, 1992.

[9] G.D. Finlayson, M.S. Drew, and B.V. Funt. Spectral sharpening: sensor transformations for improved color constancy. *J. Opt. Soc. Am. A*, 11(5):1553–1563, May 1994.

[10] G.D. Finlayson, M.S. Drew, and B.V. Funt. Spectral sharpening: sensor transformations for improved color constancy. *J. Opt. Soc. Am. A*, 11(5):1553–1563, May 1994.

[11] J.S. Gondek, G.W. Meyer, and J.G. Newman. Wavelength dependent reflectance functions. In *Proc. SIGGRAPH 1994*, pages 213–220, 1994.

[12] Pat Hanrahan and Wolfgang Krueger. Reflection from layered surfaces due to subsurface scattering. In *Proceedings of ACM SIGGRAPH 1993*, Computer Graphics Proceedings, Annual Conference Series, pages 165–174. ACM SIGGRAPH, August 1993.

[13] G.M. Johnson and M.D. Fairchild. Full-spectral color calculations in realistic image synthesis. *Computer Graphics and Applications*, 19(4: July/August):47–53, 1999.

[14] J.N. Maki, J.J. Lorre, P.H. Smith, R.D. Brandt, and D.J. Steinwand. The color of Mars: Spectrophotometric measurements at the Pathfinder landing site. *J. Geophysical Research*, 104:8781–8794, 1999.

[15] David H. Marimont and Brian A. Wandell. Linear models of surface and illuminant spectra. *J. Opt. Soc. Am. A*, 11:1905–1913, 1992.

[16] Mark S. Peercy, Benjamin M. Zhu, and Daniel R. Baum. Interactive full spectral rendering. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 67–ff. ACM Press, 1995.

[17] M.S. Peercy. Linear color representations for full spectral rendering. In *Computer Graphics (SIGGRAPH '93)*, volume 27, pages 191–198, 1993.

[18] M.S. Peercy. Linear color representations for full spectral rendering. In *Computer Graphics (SIGGRAPH '93)*, pages 191–198, 1993.

[19] Jos Stam. Diffraction shaders. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 101–110, Los Angeles, California, August 1999. ACM SIGGRAPH / Addison Wesley Longman. ISBN 0-20148-560-5.

[20] Yinlong Sun, F. David Fracchia, and Mark S. Drew. Rendering diamonds. In *Proc. WCGS 2000*, pages 9–15, 2000.

[21] G. Wyszecki and W.S. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulas*. Wiley, New York, 2nd edition, 1982.