A model of the Unity High Definition Render Pipeline, with applications to flat-panel and head-mounted display characterization

Richard F. Murray

Department of Psychology and Centre for Vision Research, York University

rfm@yorku.ca

Abstract

Game engines such as Unity and Unreal Engine have become popular tools for creating perceptual and behavioral experiments based on complex, interactive scenes. They are often used with flat-panel displays, and also with head-mounted displays. Here I describe and test a mathematical model of luminance and color in Unity's High Definition Render Pipeline (HDRP). I show that the HDRP has several non-obvious features, such as nonlinearities applied to material properties and rendered values, that must be taken into account in order to show well-controlled stimuli. I also show how the HDRP can be configured to display gamma-corrected luminance and color, and I provide software to create the specialized files needed for gamma correction.

1 Introduction

Vision science relies on a wide range of display technologies, such as stereoscopes, Maxwellian-view systems, and computer-controlled flat-panel displays, all of which have played a central role in developing and testing current theories of vision. In recent years, programming environments for creating interactive scenes, such as Unity and Unreal Engine, have been widely used for behavioral and perceptual experiments (Unity Technologies, 2022; Epic Games, 2024). These frameworks were originally developed for gaming applications, but have been useful for investigating abilities that support interactions with complex scenes, such as navigation and color constancy. These tools have been used to create experiments with traditional flat-panel displays, and also with newer display technologies such as virtual reality (VR) and augmented reality (AR) (Kimura et al., 2017; Gil Rodríguez et al., 2022; Creem-Regehr et al., 2022; Rzepka et al., 2023; Patel et al., 2024).

One challenge with using these programming environments for research is showing precisely controlled stimuli. Game engines, as well as consumer VR and AR devices, have been primarily developed for entertainment, so in research applications they often require specialized methods for precise control of stimuli. In previous work, colleagues and I described methods for

controlling luminance in Unity's Built-in Render Pipeline (BRP; Murray, Patel, & Wiedenmann, 2022). The BRP is an older pipeline, and much current development work in Unity focuses on other pipelines, such as the High-Definition Render Pipeline (HDRP), which aims to be a physics-based renderer, and includes features such as raytracing that are not present in the BRP. The HDRP also has the advantage of using meaningful photometric units for some scene properties such as lighting intensity.

Here I describe and test a mathematical model of rendering in the HDRP, with the goal of providing a model and related software tools that researchers can use to create experiments with precise control of luminance and color. I show that the HDRP has several non-obvious features, such as nonlinearities applied to material properties and rendered values, that make it important to have a model of rendering in order to maintain control over stimuli. I show how the rendering pipeline can be configured to show stimuli with gamma-corrected luminance and color. I also show how to test a wide range of hypotheses about the HDRP, so that researchers can examine additional features that are relevant to their own work. Supporting Information with Unity projects and analysis code is available at https://github.com/rfmurray/hdrp_model.

2 Previous work

Brainard et al. (2002) is a classic reference on display characterization for perceptual experiments. It focuses on characterization of cathode-ray tube (CRT) displays, but the modelling framework that it describes, as well as the discussion of issues such as spatial interactions between neighboring pixels, are highly relevant to more recent modalities as well. This paper is a valuable starting point for learning about of display characterization.

Clausen et al. (2019) tested a mathematical display model similar to the one described by Brainard et al. (2002), for two models of head-mounted VR displays, and found that the model accurately predicted displayed colors. However, they also found that it was important to set model parameters using characterization measurements for individual displays, instead of relying on standards such as sRGB or Adobe RGB, as standard models led to easily perceived errors in displayed color. Diaz-Barrancas et al. (2024) extended this work, making thorough color characterization measurements of three head-mounted display models, and examining display features such as primary spectra, color gamut, and spatial uniformity.

Toscani et al. (2019) made photometric and spectral measurements from a VR headset controlled by Unreal Engine 4. They found that with Unreal's default settings, luminance was a nonlinear and non-additive function of the simulated albedo in rendered scenes, making it difficult to generate stimuli with desired colors. However, with some of Unreal's post-processing operations disabled, luminance was a linear and approximately additive function of simulated albedo, and displayed colors were more easily controlled. Gil Rodríguez et al. (2022) built on this work, showing that colors generated by Unreal in a complex virtual environment could be predicted from rendering parameters, and that observers were able to achieve naturalistic levels of color constancy in a color matching experiment in VR.

Zaman et al. (2023) took an alternative approach to creating calibrated stimuli in Unreal, and showed how specialized materials could be used to control the chromaticity of selected objects separately from the rest of a virtual scene, so that post-processing operations did not have to be disabled for the whole scene. Duay et al. (2025) adapted this approach to Three.js, a web-based render engine, and showed how it could be used with standard materials instead of more exotic 'unlit' material

types.

Diaz-Barrancas et al. (2020, 2021) showed how hyperspectral rendering methods could be implemented in Unity, using scripts that mapped hyperspectral lighting and surface information to sRGB color coordinates in real time. Kim et al. (2021) evaluated several material and lighting settings in Unity, and measured their effects on the chromaticity of rendered objects, with the goal of finding settings that led to well-controlled and predictable colors in renderings of uniform patches and biological tissue samples.

Murray et al. (2022) developed methods for controlling luminance in Unity's Built-in Render Pipeline. We described procedures for making luminance characterization measurements, and showed how to use Unity's 'color grading' feature to make displayed luminance proportional to rendered values. The present paper extends that work, broadening its methods to handle both luminance and color, and using the HDRP instead of the BRP.

3 A model of the HDRP

In its default configuration, an HDRP project has features that make it difficult to control luminance and color. For example, a dynamic exposure feature automatically adjusts the gain on rendered images, so that as a user moves through scenes with varying levels of simulated lighting, images are mapped to a displayable range on the monitor. This is useful for applications such as gaming, but it removes the control we often need in perceptual experiments. In Appendix A, I describe configuration settings for HDRP projects that make it easier to control luminance and color. The model of the HDRP I present in the following sections assumes that these settings have been made. Throughout I use Unity 2022.3.10, which includes HDRP 14.0.9.

3.1 Lighting and material model

Lambertian material. In order to generate precisely controlled stimuli with the HDRP, we need a model of how it uses virtual lights and surfaces to compute red, green, and blue values at each pixel. Here I start by examining how the HDRP renders Lambertian surfaces.

In a typical Lambertian shading model, a material has an albedo m that indicates the proportion of light it reflects in the visible wavelength range. The material surface also has a unit normal vector $\mathbf{n}$ at each point. In a simple lighting model, lighting consists of a directional source with maximum illuminance d in direction given by unit vector $\mathbf{l}$, and an ambient source with illuminance a in all directions. According to the Lambertian model, the luminance of the surface from any viewing direction is

$$u = \frac{m}{\pi}(d \max(\cos\theta, 0) + a) \tag{1}$$

where θ is the angle between the lighting direction $\mathbf{l}$ and the surface normal $\mathbf{n}$ (i.e., $\cos\theta = \mathbf{l} \cdot \mathbf{n}$), and $\max(x, y)$ is the greater of x and y (McCluney, 2014; Pharr et al., 2023).

The HDRP supports Lambertian shading, but it parameterizes materials and lights differently than in equation (1). Unity has a material type called HDRP/ArnoldStandardSurface/ArnoldStandardSurface, which is Lambertian with its default parameter

settings. This material can be assigned a triplet $\mathbf{m} = (m_r, m_g, m_b)$ of albedo-like components for the three color channels, where each component ranges over the unit interval [0, 1]. Each point on a surface also has a unit normal vector $\mathbf{n}$, usually computed automatically from the shape that the material is assigned to. A directional light source has an intensity[1] i_d and color components $\mathbf{d} = (d_r, d_g, d_b)$. The direction of the light is controlled by rotations around the x, y, and z axes, which result in a lighting direction $\mathbf{l}$. An ambient source has intensity i_a and color components $\mathbf{a} = (a_r, a_g, a_b)$. (In Appendix A, I describe how to create these materials and light sources.) With these parameters, the rendered value in each color channel k is

$$u_k = \frac{c\, s(m_k)(i_d\, s(d_k) \max(\cos\theta, 0)/\pi + i_a a_k)}{2^e} \tag{2}$$

Here the index k stands for the three color channel indices r, g, and b. I call u_k the 'unprocessed' color coordinates, because as we will see, this stage of rendering is followed by post-processing operations.

There are several important differences between equations (1) and (2). First, equation (2) has a scale factor $c = 0.822$, which I determined empirically (see Section 7) and whose origin is unclear. It may be related to properties of the virtual camera.

Second, the albedo-like components m_k and the directional light color components d_k are transformed by a nonlinearity s, which is the 'transfer function' of the sRGB color space (Anderson et al., 1996). This nonlinearity consists of a small linear segment followed by a power function with exponent 2.4. It can be approximated by $s(x) \simeq x^{2.2}$. This transformation is why I call the components m_k albedo-like, and clearly it is important to take this nonlinearity into account when creating stimuli. Note that ambient lighting components a_k are not transformed this way. Furthermore, the directional light components d_k range over the unit interval [0, 1], whereas the ambient components a_k range over $[0, \infty)$.

Third, the directional and ambient lights differ by a factor of π in how they contribute to the unprocessed value u_k. One possible reason for this is that a uniform ambient light with visible luminance ℓ in all directions produces an incident illuminance $\pi\ell$ (McCluney, 2014). Thus the product $i_d s(d_k)$ in equation (2) can be seen as representing the maximum *incident illuminance* of the directional light, which occurs for a surface whose normal is in the lighting direction, $\mathbf{n} = \mathbf{l}$, whereas the product $i_a a_k$ represents the *visible luminance* of the ambient light in all directions, rather than the incident illuminance it creates.

Fourth, equation (2) includes a scale factor 2^e, where e is the exposure setting described in Appendix A.

Table 1 lists the variables of this model and the range of values they take, as well as additional variables introduced in later sections. I arrived at equation (2) empirically, by trial and error. In Section 7, I report tests that largely validate equation (2) as a model of Lambertian rendering in the HDRP, although I also note some small departures from the model.

Unlit material. The HDRP also has a material type called HDRP/Unlit, which is not affected by lighting or by the exposure setting. Its rendering model is very simple, which makes it useful for testing hypotheses about the render pipeline. An unlit material is assigned a color triplet $\mathbf{m} = (m_r, m_g, m_b)$, and its unprocessed value in each color channel is

$$u_k = s(m_k) \tag{3}$$

[1] 'Intensity' is the term used in the Unity interface for a multiplicative factor in light source magnitude, so I will use that term as well. Photometrically, it is something quite different, namely luminous power per unit of solid angle. I point this out because the HDRP sometimes uses photometrically correct units, but not in this case.

Table 1: Variables in the HDRP model

	variable	value range	description
scene elements	$\mathbf{m} = (m_r, m_g, m_b)$	$[0,1]^3$	material color
	$\mathbf{n} = (n_x, n_y, n_z)$	unit sphere	surface normal
	$\mathbf{d} = (d_r, d_g, d_b)$	$[0,1]^3$	directional light color
	i_d	$[0,\infty)$	directional light intensity
	$\mathbf{l} = (l_x, l_y, l_z)$	unit sphere	directional light direction
	$\mathbf{a} = (a_r, a_g, a_b)$	$[0,\infty)^3$	ambient light color
	i_a	$[0,\infty)$	ambient light intensity
	c	0.822	rendering constant
	s	$[0,1] \to [0,1]$	sRGB transfer function
	$\mathbf{s}$	$[0,1]^3 \to [0,1]^3$	sRGB transfer function, vectorized
	e	$(-\infty,\infty)$	exposure
	$\mathbf{u} = (u_r, u_g, u_b)$	$[0,\infty)^3$	unprocessed color
post-processing	$\mathbf{t} = (t_r, t_g, t_b)$	$[0,\infty)^3$	tonemapped color
	$\mathbf{f}$	$[0,\infty)^3 \to [0,1]^3$	tonemapping function
	f	$[0,\infty) \to [0,1]$	tonemapping function, single component
	u_i^*	$[0,\infty)$	tonemapping knot points, single component
	$\mathbf{u}_{ijk} = (u^r_{ijk}, u^g_{ijk}, u^b_{ijk})$	$[0,\infty)^3$	tonemapping knot points
	$\mathbf{t}_{ijk} = (t^r_{ijk}, t^g_{ijk}, t^b_{ijk})$	$[0,1]^3$	tonemapping output at knot points
	$\mathbf{v} = (v_r, v_g, v_b)$	$[0,1]^3$	post-processed color
display	h	$[0,1] \to [0,1]$	achromatic activation function
	L_0	$[0,\infty)$	minimum displayable luminance
	L_1	$[0,\infty)$	luminance range (maximum minus minimum)
	L	$[0,\infty)$	displayed luminance
	h_r, h_g, h_b	$[0,1] \to [0,1]$	chromatic activation functions
	$\mathbf{p} = (p_r, p_g, p_b)$	$[0,1]^3$	chromatic channel activations
	$\mathbf{r}, \mathbf{g}, \mathbf{b}, \mathbf{z}$	$[0,\infty)^3$	display primaries and background term (CIE XYZ)
	$\mathbf{x}$	$[0,\infty)^3$	physical color coordinates (CIE XYZ)

Here m_k and u_k both range over [0, 1], and s is the sRGB transfer function introduced above. In Section 7 I report tests that validate this model as well, again with some small departures.

3.2 Post-processing model

The unprocessed triplets $\mathbf{u} = (u_r, u_g, u_b)$ in equations (2) and (3) are transformed into post-processed triplets $\mathbf{v} = (v_r, v_g, v_b)$ as follows.

$$\mathbf{v} = \mathbf{s}^{-1}(\mathbf{f}(\mathbf{u})) \tag{4}$$

Here $\mathbf{s}^{-1}$ is the inverse of a vectorized version $\mathbf{s}$ of the sRGB transfer function s, that applies s to each component: $\mathbf{s}((x, y, z)) = (s(x), s(y), s(z))$. The function $\mathbf{f}$ is a *tonemapping* function that transforms unprocessed color triplets, and is chosen in the tonemapping override described in Appendix A. The HDRP provides a few standard choices for the tonemapping function, and also allows the user to construct a custom tonemapping function using a 'cube' file. Below I show how to use tonemapping via

a cube file to implement gamma correction.

The unprocessed components u_k range over $[0, \infty)$, whereas v_k range over [0, 1], so one role of this post-processing stage is to map the unprocessed values to a limited, displayable range.

In this first pass at describing the HDRP, I assume that we are interested in achromatic stimuli where $u_r = u_g = u_b$, and that we wish to gamma-correct the luminance of displayed stimuli. In Section 6 I generalize this approach to chromatic stimuli and colorimetric gamma correction. As part of this simplification, I assume that we choose a tonemapping function that applies the same nonlinearity f to all three color channels: $\mathbf{f}(\mathbf{u}) = (f(u_r), f(u_g), f(u_b))$. Equation (4) then becomes

$$v_k = s^{-1}(f(u_k)) \tag{5}$$

In this achromatic model, I assume that $u_r = u_g = u_b$, and so $v_r = v_g = v_b$ as well.

One point that I have not incorporated into the model is that in Unity's default configuration, the post-processed values v_k are quantized to multiples of 1/255. This fact is relevant to the design of some experiments where observers make fine discriminations.

3.3 Display model

The post-processed triplets **v** are written to a buffer and shown on a display, such as a flat-panel LCD monitor or VR headset. The display transforms the triplets **v** into physical luminances and chromaticities, so we also need a model of the display. The XYZ color coordinates **x** of the displayed stimulus are some function of **v**. Continuing for now to model achromatic stimuli, the luminance L of the displayed stimulus can be written as the following function the post-processed components.

$$L = L_1 h(v_k) + L_0 \tag{6}$$

Here L_0 is the minimum displayable luminance and L_1 is the range of displayable luminances, i.e., the maximum minus the minimum. I will call h the 'activation function' of the display. It rises from $h(0) = 0$ to $h(1) = 1$. It is often approximately given by $h(x) = x^{2.2}$, but it is more reliable to measure the activation function as part of gamma correction, as I describe below.

3.4 Summary

To summarize this model of the HDRP, we start with models for Lambertian and unlit materials that map lighting and material parameters to unprocessed color coordinates.

$$u_k = \frac{c\, s(m_k)(i_d\, s(d_k) \max(\cos\theta, 0)/\pi + i_a a_k)}{2^e} \tag{7}$$

$$u_k = s(m_k) \tag{8}$$

This is followed by post-processing, which includes tonemapping f that can be configured by the user, and a fixed nonlinearity s^{-1}.

$$v_k = s^{-1}(f(u_k)) \tag{9}$$

Finally, the post-processed values drive a display, which results in a physical luminance.

$$L = L_1 h(v_k) + L_0 \tag{10}$$

Table 1 lists the variables of this model.

4 Tonemapping

The HDRP's tonemapping interface, described in Appendix A, allows the user to choose from a number of 'modes' that transform unprocessed values $\mathbf{u}$ as in equation (4). One option is 'None', which disables tonemapping and sets $\mathbf{f}(\mathbf{u}) = \mathbf{u}$. Another option is 'External', which applies the transformation specified in a user-supplied text file that follows the 'cube' format (Adobe Systems, 2013).

External tonemapping works as follows. The HDRP has a fixed list of floating point values u_i^*, where i ranges from 1 to n. (The default value is $n = 32$, which is the setting I will assume here. The value can be adjusted in the project settings.) From this list it constructs a 3D grid of unprocessed color triplets $\mathbf{u}_{ijk} = (u^r_{ijk}, u^g_{ijk}, u^b_{ijk}) = (u^*_i, u^*_j, u^*_k)$, where each subscript ranges from 1 to n. A user-supplied cube file specifies an equally-sized 3D grid of replacement color triplets $\mathbf{t}_{ijk} = (t^r_{ijk}, t^g_{ijk}, t^b_{ijk})$. Tonemapping maps elements of the first grid to corresponding elements of the second grid: $\mathbf{f}(\mathbf{u}_{ijk}) = \mathbf{t}_{ijk}$. Unprocessed color triplets $\mathbf{u}$ that are not equal to one of the triplets $\mathbf{u}_{ijk}$ are mapped to outputs by linear interpolation from nearby input points $\mathbf{u}_{ijk}$ and output points $\mathbf{t}_{ijk}$. In interpolation frameworks such as this one, the points $\mathbf{u}_{ijk}$ are often called 'knot points'.

In Section 7, I report tests that find the knot points points $\mathbf{u}_{ijk}$ used by the HDRP, and validate this model of tonemapping in External mode. Tonemapping can be used for gamma correction, as described in the following section.

5 Achromatic gamma correction

For achromatic stimuli, the goal of gamma correction is to make the physical luminance L of each pixel proportional to the unprocessed value u_k that is rendered from scene parameters such as material properties and lighting. Combining the post-processing and display models in equations (5) and (6), the relationship between L and u_k is

$$L = L_1 h(s^{-1}(f(u_k))) + L_0 \tag{11}$$

Alternatively, we can define $w = L_0/L_1$, and write equation (11) as

$$L = L_1(h(s^{-1}(f(u_k))) + w) \tag{12}$$

This provides a solution for gamma correction: we can make L proportional to u_k by setting the tonemapping function f to

$$f(u_k) = s(h^{-1}((1+w)\frac{u_k}{r} - w)) \tag{13}$$

where r is a constant. Then equation (12) reduces to

$$L = (L_0 + L_1)\frac{u_k}{r} \tag{14}$$

and L is proportional to u_k.

One point I have glossed over in the previous paragraph is that h^{-1} maps [0, 1] to [0, 1], but for small values of u_k, the input to h^{-1} in equation (13) is negative. (Consider $u_k = 0$, for instance.) This occurs for small values of u_k that would ideally be mapped to luminances less than L_0, the minimum displayable luminance. A natural solution is to revise equation (13) so that the input to h^{-1} is clipped to non-negative values, as follows.

$$f(u_k) = s(h^{-1}(\max((1+w)\frac{u_k}{r} - w, 0))) \tag{15}$$

Then, as u_k approaches zero, the luminance goes no lower than L_0, the lowest displayable luminance, as we might expect. Equation (14) is then replaced by

$$L = \begin{cases} (L_0 + L_1)\frac{u_k}{r} & u_k \geq \frac{rw}{1+w} \equiv u_0 \\ L_0 & u_k < u_0 \end{cases} \tag{16}$$

and L is proportional to u_k, except when $u_k < u_0$, in which case $L = L_0$. I will consider typical values for the cutoff point u_0 below, but to anticipate, it is typically very small compared to the displayable range of u_k.

To create the tonemapping function defined in equation (15) we need to know s, h, L_0, L_1, and r. The function s is fixed, and tests in Section 7 show that it is the sRGB transfer function. We can find h, L_0, and L_1 by measuring the mapping from post-processed values v_k to luminance L on the display. To make these measurements, we can display post-processed values v_k using the unlit material type discussed earlier. Equations (3) and (5) show that if tonemapping is disabled (which sets $f(x) = x$), then the post-processed values of an unlit material with color coordinates m_k are

$$v_k = s^{-1}(f(s(m_k))) = m_k \tag{17}$$

That is, assigning an unlit material color coordinates m_k results in post-processed values v_k. To measure h, L_0, and L_1, we can display surfaces with a range of post-processed values v_k, and measure the resulting luminances.

Finally, the constant r can chosen arbitrarily, and it determines the range of unprocessed values u_k that is mapped to the displayable range of luminances. Equation (16) shows that the maximum luminance $L_0 + L_1$ is shown when $u_k/r = 1$, so the displayable range of u_k is $[0, r]$. We can choose $r = 1$, in which case the displayable range of u_k is [0, 1].

Now we can consider typical values for the cutoff point u_0 in equation (16). We can choose $r = 1$, and typically $w = L_0/L_1$ is no larger than 0.01, which results in $u_0 \approx 0.01$. In this case luminance L equals L_0 for u_k in [0, 0.01], and L is proportional to u_k over the much larger range [0.01, 1].

By choosing appropriate output triplets $\mathbf{t}_{ijk}$ for tonemapping in External mode, we can create an approximation to the tonemapping function defined for gamma correction in equation (15). The most straightforward approach is for the tonemapping function to process each color channel independently, and to map the input points $\mathbf{u}_{ijk} = (u_i^*, u_j^*, u_k^*)$ to the values defined by equation (15):

$$\mathbf{t}_{ijk} = (t_{ijk}^r, t_{ijk}^g, t_{ijk}^b) = (f(u_i^*), f(u_j^*), f(u_k^*)) \tag{18}$$

This gives an approximation to the tonemapping function in equation (15). We can improve this approximation slightly by taking the $\mathbf{t}_{ijk}$ in equation (18) as initial values in an optimization problem that finds the $\mathbf{t}_{ijk}$ that minimize the difference between the approximation and the function defined by equation (15), using a global error measure such as the sum-of-squares difference between the functions. In Section 7.3, I show results from a Unity project included in the Supporting Information that takes this approach to gamma correction.

6 A chromatic model

Here I extend the model in the previous sections to include color and methods for chromatic gamma correction.

6.1 A chromatic display model

The CIE XYZ chromaticity coordinates $\mathbf{x}$ of a stimulus are some function of the post-processed color coordinates $\mathbf{v}$. In many displays, each coordinate v_k controls the intensity of a primary color ($\mathbf{r}$, $\mathbf{g}$, or $\mathbf{b}$), the three scaled primaries sum, and there is also a small, fixed background term ($\mathbf{z}$).

$$\mathbf{x} = h_r(v_r)\,\mathbf{r} + h_g(v_g)\,\mathbf{g} + h_b(v_b)\,\mathbf{b} \;+\; \mathbf{z} \tag{19}$$

The activation functions h_k describe how the primary intensities depend on v_k. We can also refer to the 'activation' $p_k = h_k(v_k)$ of each chromatic channel. Equation (19) expresses the important assumptions that (a) the chromaticities of the primaries do not change with their level of activation, and (b) the three chromatic channels combine in a purely additive fashion.

If the background term can be expressed as a sum of the primaries, $\mathbf{z} = w_r\,\mathbf{r} + w_g\,\mathbf{g} + w_b\,\mathbf{b}$, then it can be absorbed into the primary coefficients.

$$\mathbf{x} = (h_r(v_r) + w_r)\,\mathbf{r} + (h_g(v_g) + w_g)\,\mathbf{g} + (h_b(v_b) + w_b)\,\mathbf{b} \tag{20}$$

The three components of this equation consist of an activation term and a fixed background term, much like equation (6) in the achromatic luminance model. This suggests that we can carry out chromatic gamma correction simply by applying the method described for achromatic gamma correction to each color channel independently, and this is the approach I will take.

6.2 Chromatic gamma correction

One possible goal for chromatic gamma correction is that the primary coefficients in equation (20) should be proportional to the unprocessed color coordinates u_k.

$$h_k(v_k) + w_k \propto u_k \tag{21}$$

We can also choose the constant of proportionality in equation (21) so that at $u_k = 1$, the corresponding primary is at its maximum intensity, e.g., $u_r = 1$ creates $(1 + w_r)\mathbf{r}$.

Equation (4) describes how post-processing maps unprocessed color coordinates $\mathbf{u}$ to post-processed coordinates $\mathbf{v}$. I repeat it here:

$$\mathbf{v} = \mathbf{s}^{-1}(\mathbf{f}(\mathbf{u})) \tag{22}$$

Our goal is to choose the tonemapping function $\mathbf{f}$ so that u_k and v_k are related as in equation (21). We can choose a tonemapping function that consists of three scalar functions f_k that are applied to the three color channels independently.

$$v_k = s^{-1}(f_k(u_k)) \tag{23}$$

Following the approach used for achromatic gamma correction, I define the components of the tonemapping function to be

$$f_k(u_k) = s(h_k^{-1}(\max((1 + w_k)u_k - w_k, 0))) \tag{24}$$

With this definition of f_k,

$$\begin{aligned} h_k(v_k) &= h_k(s^{-1}(f_k(u_k))) \\ &= \begin{cases} (1 + w_k)u_k - w_k & u_k \geq w_k/(1 + w_k) \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

and $h_k(v_k) + w_k$ is proportional to u_k, as was the goal in equation (21), except at small values of u_k. In the latter case, the displayed stimulus $\mathbf{x}$ cannot approach zero, and is limited by the background component $\mathbf{z}$. Furthermore, when $u_k = 1$, the corresponding primary is displayed with intensity $h_k(v_k) + w_k = 1 + w_k$, as was also the goal.

Additional goals are possible for chromatic gamma correction. We might also want to control the chromaticity displayed when $u_r = u_g = u_b$ (i.e., the white point), and to use arbitrary color primaries for $\mathbf{r}$, $\mathbf{g}$, and $\mathbf{b}$ instead of the physical primaries of the display. The mapping that can be specified in External tonemapping mode may be flexible enough to incorporate these

features, but I will not develop them here.

7 Tests of the HDRP model

Here I report tests of the model described in the previous sections. The Supporting Information includes all Unity projects and analysis scripts required to carry out these tests, as well as detailed instructions.

7.1 Tests without tonemapping

To test the HDRP model, I created a Unity project (render_random) where the scene consists of a plane under directional and ambient lights. The project assigns random color coordinates m_k to the plane, and rotates it to a random orientation $\mathbf{n}$. It also assigns random color coordinates d_k, a_k and intensities i_d, i_a to the directional and ambient lights, and a random direction $\mathbf{l}$ to the directional light. It then captures the post-processed color coordinates v_k at one pixel in the rendered image of the plane. It records the random scene parameters and the rendered values. This process is repeated many times, providing many samples of randomly generated material and lighting parameters, and resulting rendered values.

From this data, Figure 1 plots the unprocessed color coordinates u_k predicted by equation (2) for a Lambertian surface without tonemapping, but without the scale constant c, against the unprocessed color coordinates $u_k = s(v_k)$ calculated from the actual post-processed values v_k. The predicted and actual values are highly correlated, but they differ by a scale factor. The solid red line is the least-squares regression line, and c can be estimated as the inverse of the regression slope. The value $c = 0.822$ reported earlier is based on 5000 samples, which repeated runs show to provide a stable estimate.

Figure 2 shows further analyses of data from a Lambertian surface without tonemapping. Panel (a) plots the post-processed color coordinates v_k predicted by equations (2), (3), and (4), now incorporating the scale factor $c = 0.822$, against the actual post-processed values. Overall, the model is quite accurate. Panel (b) shows the prediction error, i.e., predicted minus actual values of v_k, as a function of the actual values v_k. The horizontal lines show $y = \pm 1/255$, to indicate the scale of error we would expect from rounding v_k to eight-bit precision. Errors are mostly within this range, and almost always within a few multiples of this range. The median absolute prediction error, expressed as a multiple of 1/255, is 0.53/255. The source of this residual error is unclear. One evident bias is that the predictions tend to be too low at low values of post-processed color coordinates v_k. Panel (c) shows that this bias is largely a function of the material color coordinates m_k: at low values of m_k, the predicted v_k tends to be too low. Fortunately, the largest negative biases occur for material colors in the range $m_k < 0.2$, which correspond to albedos of $s(0.2) = 0.033$ and lower. It is rare for everyday materials to have albedos less than 0.03, so the strongest biases in the model occur with very dark materials that are uncommon in natural scenes. Panel (d) replots the data in panel (b), without points where $m_k < 0.2$, and shows that without these points the median absolute error is reduced to 0.48/255. Overall, these errors are small, but they could be limiting factors in experiments where observers make fine perceptual discriminations.

Figure 3 shows results of the same tests with an unlit material without tonemapping. Here too, the model is overall quite accurate. Even with this simple material, though, the predictions are not perfect, and the median absolute error is 0.58/255. For

an unlit material, material color m_k is the only parameter that affects the rendered value v_k, so any biases are again a function of m_k.

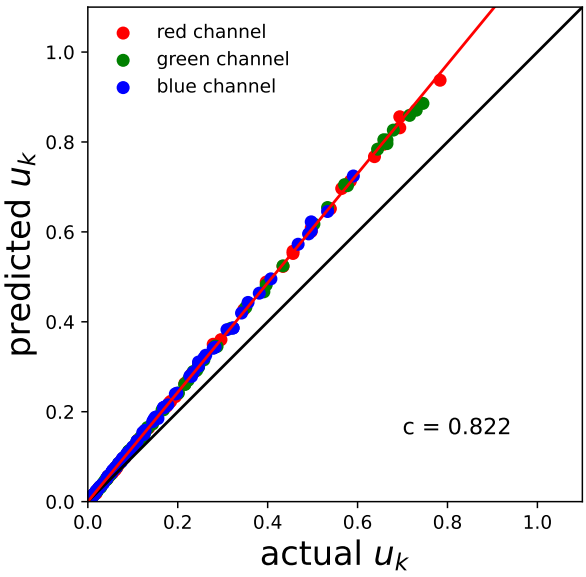

Figure 1: Model predictions of unprocessed values u_k from a Lambertian surface with no tonemapping, plotted against the unprocessed values inferred from actual post-processed values as $u_k = s(v_k)$. The solid black line shows $y = x$, and the solid red line is the least-squares regression line, constrained to pass through the origin.

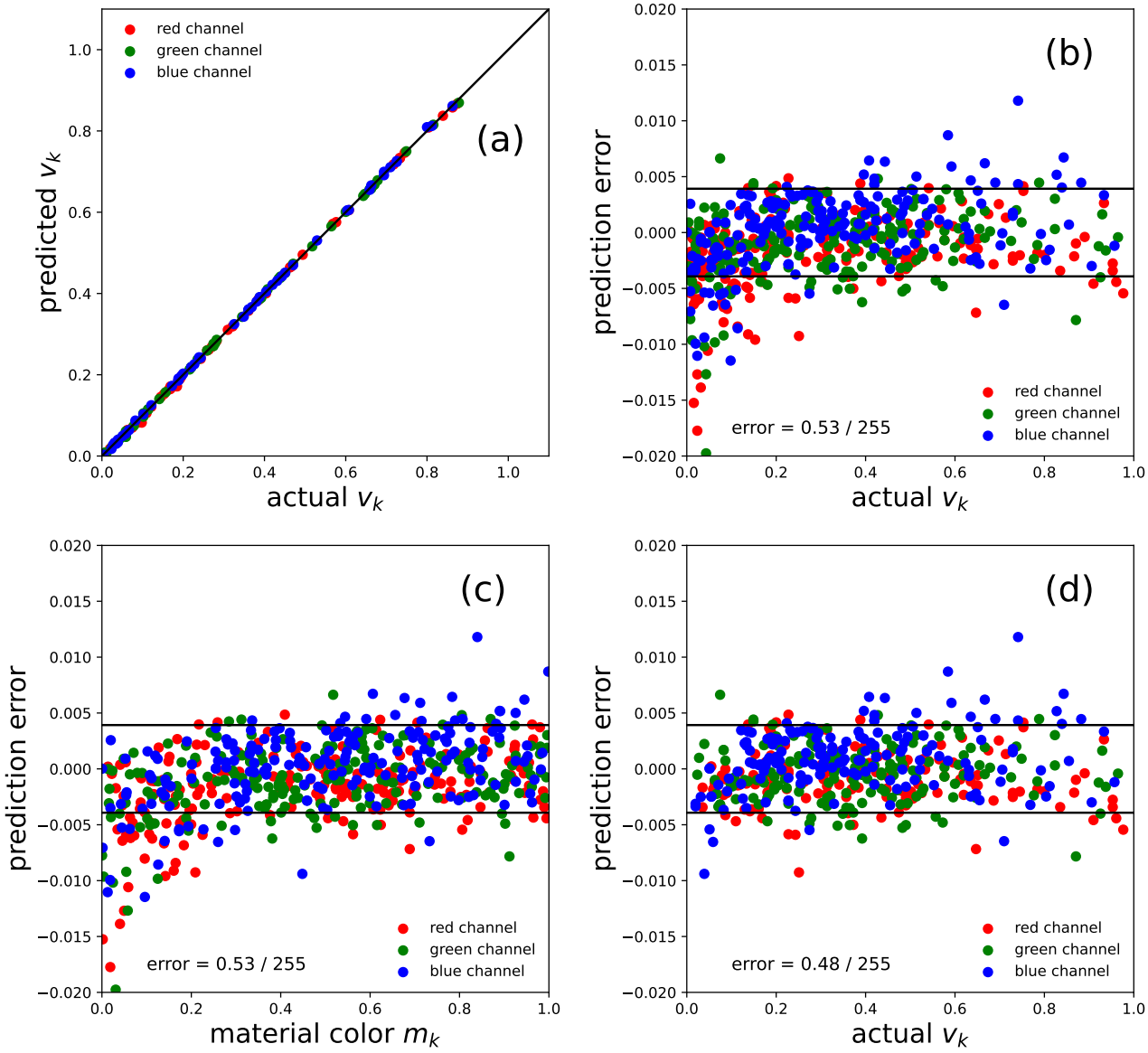

Figure 2: (a) Model predictions of post-processed values v_k in renderings of a Lambertian surface with no tonemapping, plotted against the actual post-processed values. The solid line shows $y = x$. (b) Prediction error for post-processed values v_k, i.e., predicted minus actual value. The solid lines show $y = \pm\, 1/255$. The error value is the median absolute error, reported as a multiple of 1/255. (c) Prediction error as a function of material color m_k. (d) Repeat of panel (b), without points with material color coordinates $m_k < 0.2$.

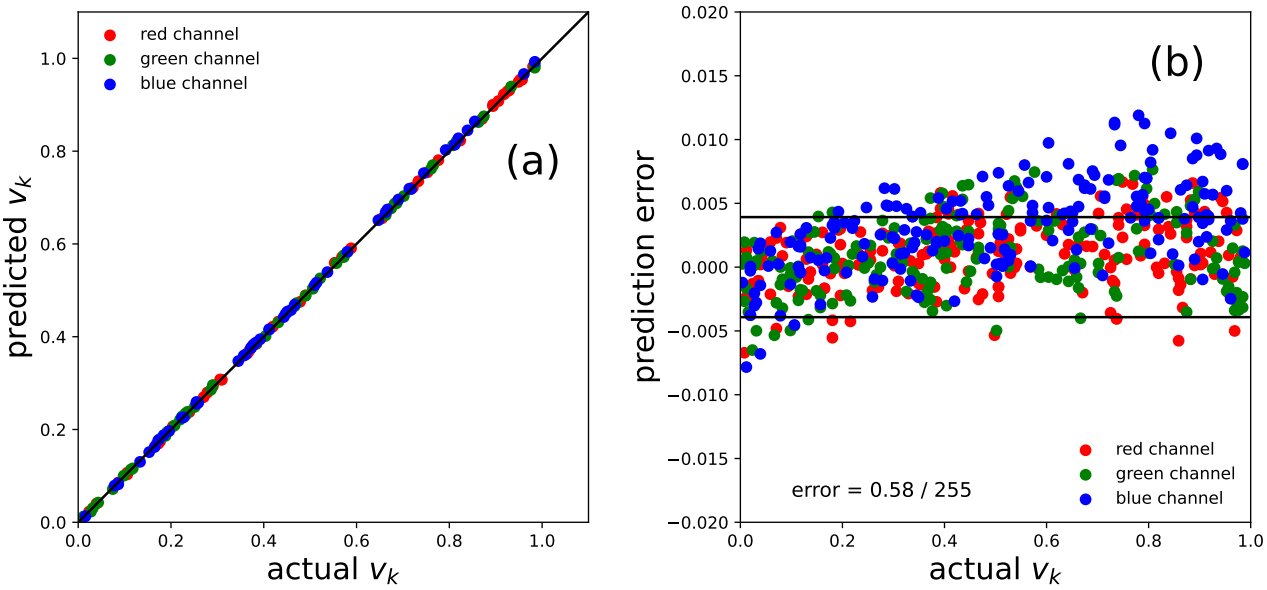

Figure 3: (a) Model predictions of post-processed values v_k in renderings of an unlit material type with no tonemapping, plotted against the actual post-processed values. (b) Prediction error for post-processed values v_k.

Table 2: Estimates of knot points coordinates u_i^* used by cube files, for i from 3 to 32

(a) Estimates from delta functions									
0.0002606	0.003104	0.007305	0.01288	0.02056	0.03061	0.04468	0.06393	0.09056	0.1245
0.1711	0.2354	0.3236	0.4406	0.5938	0.8165	1.111	1.498	2.039	2.776
3.780	5.094	6.935	9.441	12.72	17.32	23.35	31.78	43.27	58.90
(b) Estimates from optimizing model predictions									
1.657×10^{-9}	0.002830	0.007137	0.01269	0.02051	0.03086	0.4479	0.06444	0.08989	0.1252
0.1726	0.2370	0.3253	0.4422	0.6039	0.8207	1.104	1.495	2.032	2.756
3.738	5.083	6.864	9.347	12.62	17.18	23.24	31.48	42.75	57.66

7.2 Tests with tonemapping

As explained in Section 4, tonemapping relies on a set of values u_i^* that are used to generate knot points $\mathbf{u}_{ijk} = (u_i^*, u_j^*, u_k^*)$. Tonemapping maps these knot points to corresponding values $\mathbf{t}_{ijk} = (t^r_{ijk}, t^g_{ijk}, t^b_{ijk})$ that are given in a cube file. Color triplets $\mathbf{u}$ between the knot points are mapped to values $\mathbf{f}(\mathbf{u})$ by interpolation.

In order to create a cube file that defines a specific tonemapping function, we need to know u_i^*. Figure 4 illustrates one approach to measuring these values. Panel (a) shows the tonemapped value t_k, as a function of the unprocessed value u_k, with tonemapping carried out according to a cube file that maps knot points $\mathbf{u}_{ijk} = (u_i^*, u_j^*, u_k^*)$ to $\mathbf{t}_{ijk} = (\delta_{i,16}, \delta_{j,16}, \delta_{k,16})$, where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. That is, in each color channel the sixteenth knot point value is mapped to one, and all others are mapped to zero. With this configuration, the output of tonemapping peaks for an input value of $u_k = 0.44$, which we can take as an estimate of the knot point component u_{16}^*.

Figure 4(b) shows this analysis repeated with 32 cube files delta_m.cube, where m ranges from 1 to 32. File delta_m.cube maps knot points $\mathbf{u}_{ijk} = (u_i^*, u_j^*, u_k^*)$ to $\mathbf{t}_{ijk} = (\delta_{im}, \delta_{jm}, \delta_{km})$. The peak of each dataset is labelled with the number 1-32 of the cube file that was used to create it. In each case, we can take the input value that produces the largest output as an estimate of u_m^*.

This analysis provides much useful information about tonemapping. First, points u_1^* and u_2^* do not appear to play a role. For cube files delta_01.cube and delta_02.cube, all input values are mapped to zero, which is why there are no peaks with labels 1 and 2 in Figure 4. Second, the knot point coordinates u_i^* are approximately (though only approximately) logarithmically spaced between $u_3^* = 0.00026$ and $u_{32}^* = 59$. Third, all input values below u_3^* and above u_{32}^* are mapped to a constant value. Fourth, this analysis gives estimates of u_i^* for i from 3 to 32, which we can use to create cube files for specific tonemapping functions. Table 2(a) lists these estimates of u_i^*. Fifth, interpolation between neighboring points is linear, as indicated by the triangular responses to delta functions. Sixth, there is an anomaly in interpolation between points u_3^* and u_4^*, where instead of smooth interpolation there is a discontinuity near u_3^*.

The estimates of u_i^* from this analysis (see Table 2(a)) allow us to model the effect of tonemapping with cube files. Tonemapping maps knot points $\mathbf{u}_{ijk} = (u_i^*, u_j^*, u_k^*)$ to values $\mathbf{t}_{ijk}$ specified in a cube file, and maps other values of $\mathbf{u}$ by interpolation. The Supporting Information defines a Python class, TonemapCube, that loads a user-supplied cube file and implements this model of the tonemapping function $\mathbf{f}$. This makes it possible to use equation (4) to calculate predictions for

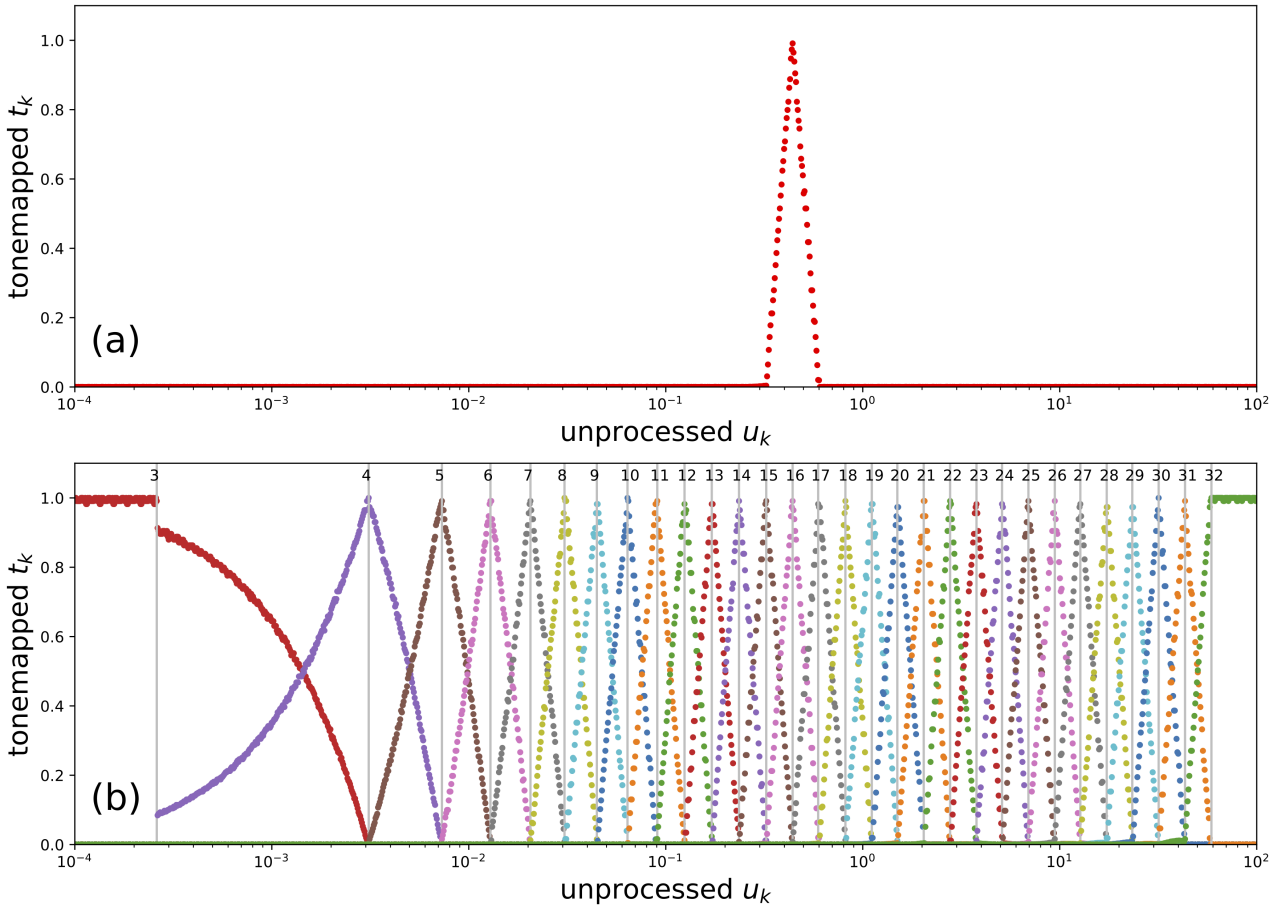

Figure 4: (a) Tonemapped values t_k resulting from a wide range of unprocessed values u_k, using a cube file that maps knot point coordinate u^*_{16} to one, and all other knot points to zero. (b) The same analysis repeated 32 times, using cube files that each map a single knot point coordinate u^*_m to one, and all others to zero. Results from different cube files are shown in different colors, and the peak of each curve is labelled with the number m of the knot point that is mapped to one.

post-processed color coordinates v_k that incorporate tonemapping.

Figure 5 shows results of tests like those reported in Section 7.1, with the HDRP model extended to include tonemapping. This test combines data from random, rendered scenes, generated by the render_random project described above, with three different tonemapping functions: the tonemapped value t_k is identical to, the square root of, or the square of the unprocessed value u_k. This gives an indication of how the model performs across a range of tonemapping functions. The model's predictions are broadly accurate for all three tonemapping nonlinearities, but show somewhat higher errors than without tonemapping. This suggests that the estimates of u^*_i can be improved. Just as rendered values v_k are subject to small random variations and biases (e.g., see Figure 2), the tonemapped values t_k that we used to estimate the knot point coordinates u^*_i have small departures from their idealized values as well. As a result, the effects of cube files that implement delta functions reveal useful information about tonemapping, as we have seen, but they do not allow us to estimate u^*_i with arbitrary precision.

An alternative, more direct approach to estimating the knot point coordinates u^*_i is to search for values of u^*_i that optimize the model's predictions. Figure 6 shows results from a test where knot points were estimated this way. The data consisted of random scene parameters and rendered values, generated by render_random, with tonemapping set to identity, square root, and

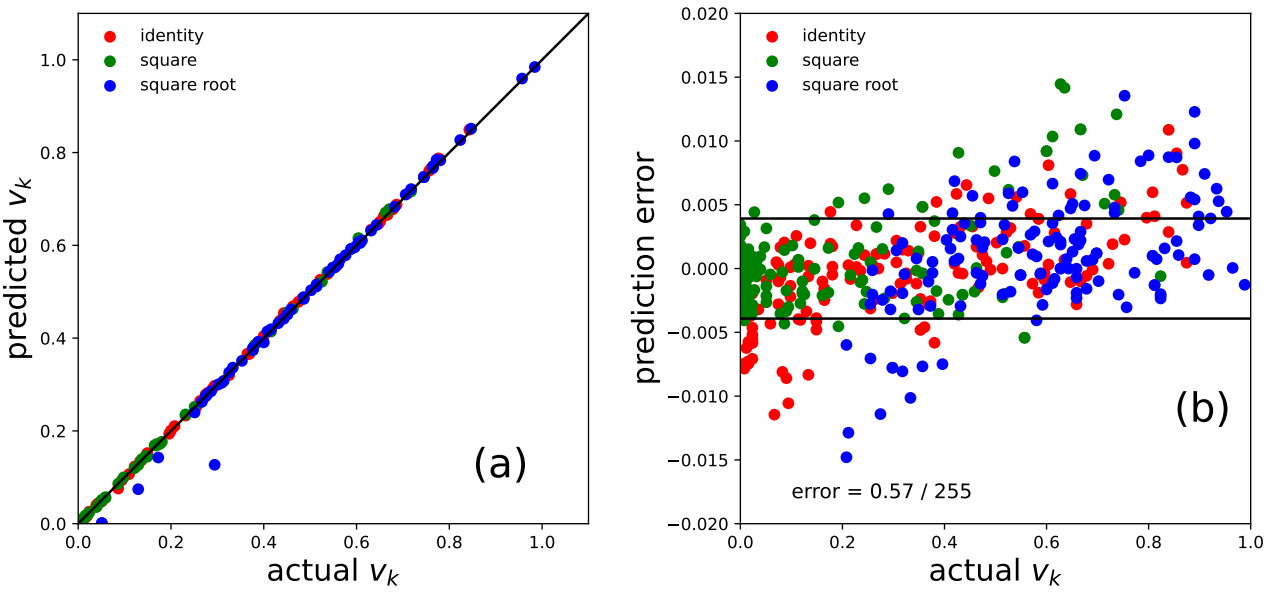

Figure 5: (a) Model predictions of post-processed values v_k in renderings of a Lambertian surface with three kinds of tonemapping (identity, square, and square root functions), plotted against the actual post-processed values. Tonemapping predictions were made using knot point coordinates u_i^* estimated by tonemapping delta functions. (b) Prediction error for post-processed values v_k.

square functions. Samples with $m_k < 0.2$ were eliminated from the dataset, since tests without tonemapping had shown the HDRP model to be biased for these values, and I did not want the estimates of u_i^* to be biased in order to compensate for the model's biases at low m_k. Starting from initial estimates of u_i^* given by the delta function analysis, a nonlinear optimization routine searched for the values of u_i^* that minimized the sum-of-squares error in the HDRP model's predictions. Table 2(b) reports the resulting values. Figure 6 shows the predictions and prediction error when the resulting model was applied to new random samples from the same dataset, i.e., samples not used to estimate u_i^*, in order to avoid overfitting. The model was about as accurate as the model without tonemapping, with a median absolute error of 0.50/255. As was found with scenes without tonemapping, the largest errors occured with low material color coordinates, $m_k < 0.2$, and when these samples were eliminated, most of the larger errors disappeared, and the median absolute error decreased to 0.49 (figure not shown).

Figure 7 shows results of the same test with an unlit material. The predictions are generally accurate, though with a somewhat higher median absolute error of 0.69/255.

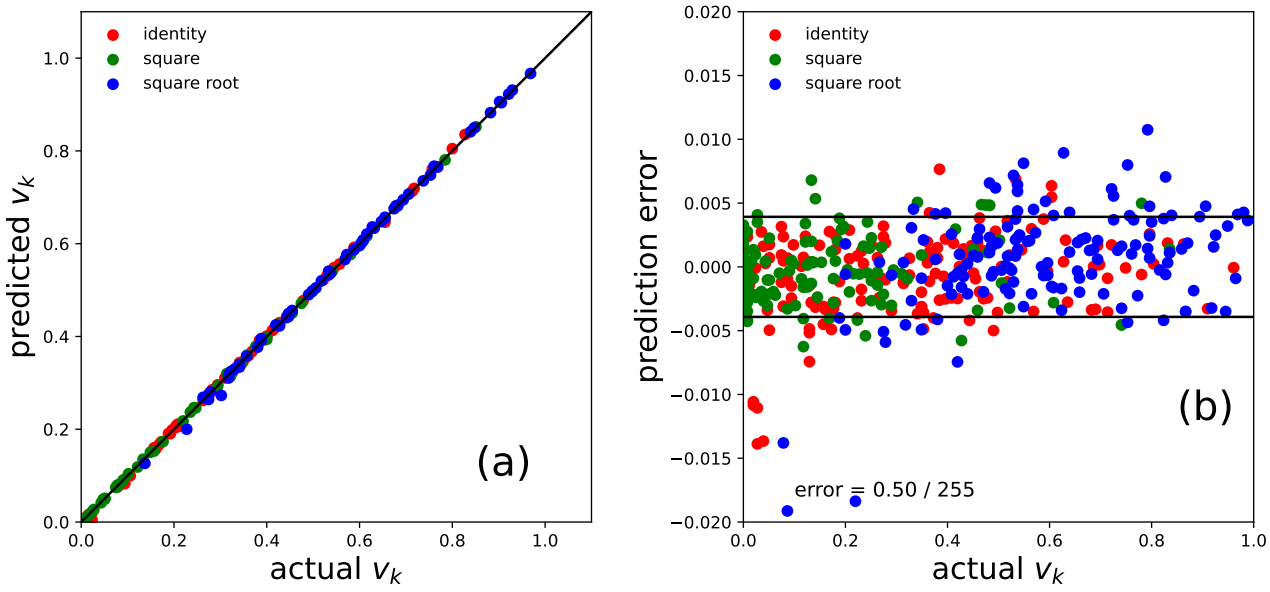

Figure 6: (a) Model predictions of post-processed values v_k in renderings of a Lambertian surface with tonemapping, plotted against the actual post-processed values. Tonemapping predictions were made using knot point coordinates u_i^* estimated by optimizing model predictions. (b) Prediction error for post-processed values v_k.

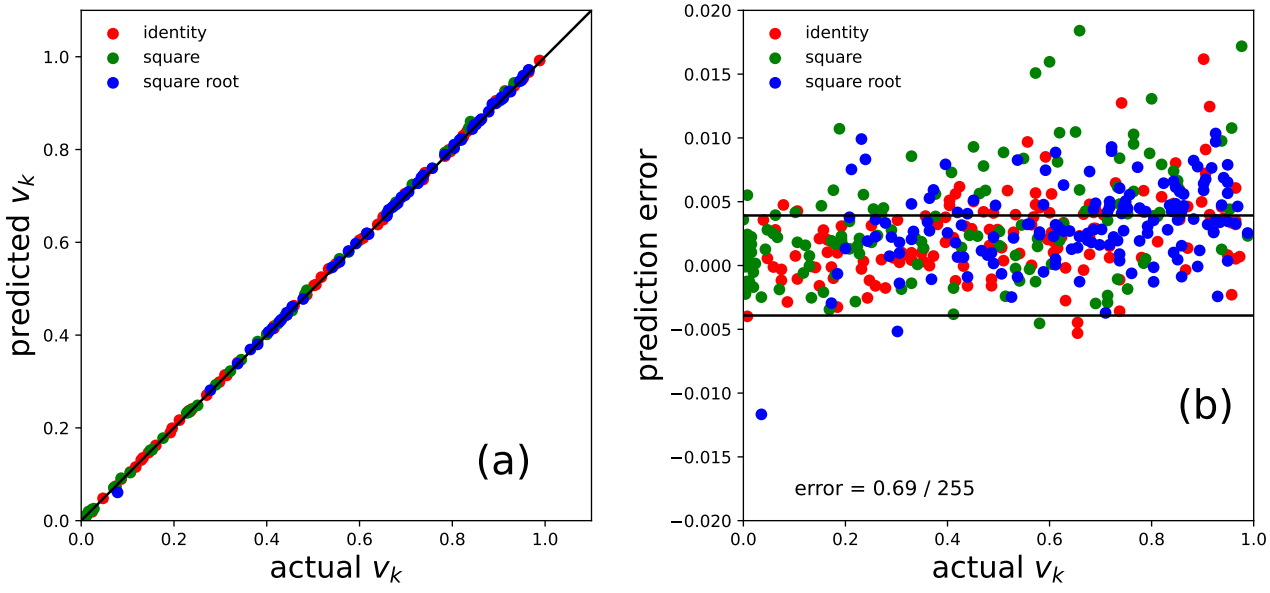

Figure 7: (a) Model predictions of post-processed values v_k in renderings of an unlit material with tonemapping, plotted against the actual post-processed values. Tonemapping predictions were made using knot point coordinates u_i^* estimated by optimizing model predictions. (b) Prediction error for post-processed values v_k.

7.3 Tests of gamma correction

The Supporting Information includes a Unity project (caldemo) that illustrates how to use the methods described here for gamma correction in an experiment. Unity has many configuration options, so I recommend integrating routines for characterization measurements into the experiment itself, rather having them in a separate project. This makes it less likely that there are important configuration differences between the routines where characterization measurements are made, and the experiment where they are used.

The caldemo project is a simple orientation discrimination experiment that includes characterization code. A checkbox in the Unity editor flags whether achromatic or chromatic characterization should be used. At any point during the experiment, the user can press a key to switch the program into characterization mode, at which point it shows a series of achromatic or chromatic test stimuli, for which the user can measure the luminance or XYZ color coordinates. Separate Python scripts use these measurements to generate a cube file for tonemapping, which can then be loaded into the project. When the experiment is run again and the characterization measurements are made again, luminance or XYZ coordinates are proportional to unprocessed values u_k. The Supporting Information includes instructions on how to carry out these tasks.

Figure 8 shows results for these tests of achromatic and chromatic characterization. I found that several monitors had gamma functions close to the sRGB standard, and as a result the nonlinearities in the HDRP produced luminance values and XYZ coordinates that were fairly close to a linear function of unprocessed color coordinates u_k, even without using tonemapping for gamma correction. To show that the methods described here linearize the response even when there are significant departures from the sRGB standard, I used MATLAB and the Psychtoolbox (Kleiner et al., 2007) to set a monitor's lookup table so that the response was clearly nonlinear (Figure 8). Luminance measurements were made using a Minolta LS-110 photometer, from an LCD monitor (Samsung, model 24T350) driven by an NVIDIA GeForce 2080 graphics card on a PC running Windows 11. Figure 8(a) shows the nonlinear response in luminance characterization measurements without tonemapping, as well as the linearized response when tonemapping was applied using a cube file generated by code provided in the Supporting Information. Figure 8(b) shows results of a corresponding test for chromatic gamma correction, using measurements made on the same monitor using a Photo Research PR-655 spectroradiometer. In both cases, tonemapping approximately linearized the relationship between the rendered values u_k and the physical stimulus. There were some small departures from linearity, which may be due to the inability of a gamma function, which was used to create the cube file, to fit the empirical measurements exactly; a more flexible function such as a spline may give slightly better results.

For this method of gamma correction to work, the display must meet the assumptions of the rendering model. I ran the same tests on an iMac (24" display, Apple M1 chip, 2021, macOS 15.3.2), and found that gamma correction failed. The reason was that the post-processed values $\mathbf{v}$ rendered by Unity were not the final framebuffer values, as revealed for instance by macOS's Digital Color Meter. That is, the operating system applied a transformation to Unity's final rendered values before displaying them. The nature of this transformation was unclear, but it was affected by the ICC color profile chosen for the display in the system preferences. As a result, each framebuffer color channel depended on all three rendered color channels $\mathbf{v}$, and since the framebuffer values were nonlinearly related to the physical stimulus, the assumption of channel additivity was violated. Modern

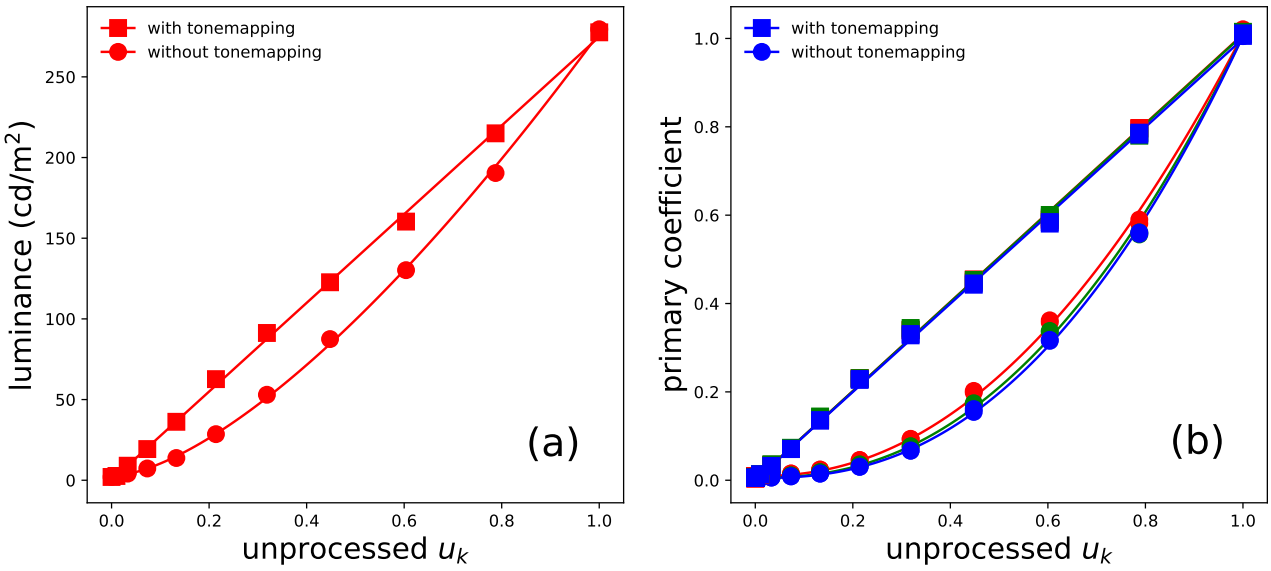

Figure 8: Display characterization measurements. The curves through measurements without tonemapping are fits of a gamma function. The lines through measurements with tonemapping are fits of a straight line constrained to pass through the origin. (a) Luminance characterization. (b) Color characterization, with each plot color representing the corresponding color channel (red, green, or blue), and the y-axis representing the primary coefficients in equation (20).

operating systems include color management methods whose effects are not always easy to determine, and so it is important to test the success or failure of gamma correction methods.

8 Discussion

Game engines such as Unity and Unreal are complex programming environments, and one lesson from the present work is that it is important to have a well-tested model of color and other stimulus features that we want to control in experiments. Consider some of the idiosyncratic features that are documented in equation (2) for Unity's treatment of Lambertian materials: the nonlinearity s is applied to material color coordinates m_k and directional light intensity d_k, but not to ambient light intensity a_k; the effects of the parameters for directional and ambient light intensity, i_d and i_a, differ by a factor of π; and the rendered values depend on a constant $c = 0.822$. Furthermore, rendered values u_k are passed through additional nonlinearities, as in equation (4). A model of these processes is necessary in order to reliably control stimuli, and without such a model, errors are likely.

I have presented a model of the HDRP in a simple configuration, namely a Lambertian surface under directional and ambient lighting. One advantage of game engines is that they allow us to carry out experiments in complex, somewhat realistic, rendered scenes, so it will be important to extend this work to examine more advanced features of the HDRP, such as light probe illumination and more realistic material models. The Unity projects and analysis code provided as Supporting Information provide a starting point for such work.

Table 1 lists the range of possible values for variables in the rendering model. In fact, some variables can be assigned values outside these ranges, e.g., in a C# script, a Lambertian material can be assigned color coordinates m_k greater than one, which

implies an an albedo greater than one. Some other variables can be assigned out-of-range values as well. However, I have not tested the model thoroughly for such values, and I do not recommend using them without further testing.

One challenge for characterizing head-mounted displays is simply measuring displayed luminances and colors. This problem is largely outside the scope of this paper, but the methods presented here depend on it. The ideal solution is to use specialized equipment that has been designed for head-mounted displays, but such equipment is currently quite costly. Many labs have spot photometers, colorimeters, or spectrophotometers, but the optics of these devices are typically not matched to head-mounted displays. Murray et al. (2022) gave some evidence that a Minolta LS-110 spot photometer provides measurements that are proportional to the luminance displayed in an Oculus Rift S VR headset, but they also showed that the measurements integrate over a region more than 15° in diameter, well beyond the photometer's nominal integration region of 0.33°. The luminance and chromaticity of a display can vary appreciably over regions this large, which may compromise measurements. It would be useful to have studies that compare measurements of luminance and color in head-mounted displays, when made with specialized equipment and with more widely available tools such as spot photometers. Similarly, technical specifications for head-mounted displays often report their color primaries and gamma functions, for example as conforming to a standard such as sRGB, and it would be useful to know how accurate and stable these specifications are in practice for consumer devices.

I developed the model described in this paper for the most part empirically, by trial and error, with some guidance from Unity's documentation. As shown here, I have validated the model under some rendering conditions. Nevertheless, I encourage users to use the code provided in the Supporting Information to test the model in their own experiments, and in general to use the model cautiously. It may need to be extended for rendering conditions beyond those tested here.

9 Acknowledgements

I thank David Brainard and Fangfang Hong of the University of Pennsylvania for helpful discussions. This work was supported by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada.

Appendix A Configuring an HDRP project

Here I describe how to create and configure a Unity project where the models described in this paper hold. I also explain what parameters of Unity lights and materials correspond to the variables in equations (2) and (3). I assume the reader is familar with basic procedures for using Unity. For an introduction to Unity, I recommend the first eight chapters of Geig (2022). They cover the Built-in Render Pipeline, but most of their content applies to the HDRP as well.

In the Unity Hub, create a new project using the 'High Definition 3D' template. After the project opens in the Unity Editor, the default HDRP configuration is found in the project settings (menu entry Edit / Project Settings ...). Instead of changing these settings directly, we will override them in a Volume object.

In the Hierarchy view, delete the 'Sky and Fog Volume'. Create a 'Volume / Global Volume'. This object controls rendering properties in the scene. In the Inspector view for this volume, next to Profile, click New.

Still in the Inspector view for Global Volume, click 'Add override', and choose type Exposure. This override configures the exposure feature described in the main text, that in its default state dynamically adjusts the gain on rendered images. Click the checkbox next to Mode, and leave the mode at its default value, which is Fixed. Click 'Fixed Exposure', and enter a value of 0. This is the value of e in equation (2), which shows that rendered values are now divided by $2^0 = 1$.

Add another override, and this time choose type 'Post-processing / Bloom'. The bloom feature causes light scattering around bright objects. Click Quality, and set it to Low. Click Intensity, and enter a value of zero. This feature is now off.

Add a 'Post-processing / Tonemapping' override. Click Mode, and leave it at its default value, which is None. Tonemapping is now disabled, i.e., $\mathbf{f}(\mathbf{u}) = \mathbf{u}$. Alternatively, set the mode to External. Drag a cube file into the Assets view. (Cube files are included in the Supporting Information.) In the Inspector view for Global Volume, in the Tonemapping override, check 'Lookup Texture'. Click the small circle to the right, and in the window that appears select the cube file you put in the Assets view. The tonemapping feature is now active, and it uses the mapping specified in the cube file.

The next two overrides configure the ambient lighting. First, add a 'Visual Environment' override. Click 'Sky Type', and choose 'Gradient Sky'. Second, add a 'Sky / Gradient Sky' override. This override allows the user to set the color of the top, middle, and bottom parts of the sky. We will set all three to the same value, to create a classic ambient light source with the same color and intensity in all directions. Click Top. Click the color patch to the right, and in the window that appears, set the color coordinate mode to 'RGB 0-1.0', enter the value 1 for each of the three colors (these are the values a_k in equation (2)), and close the color picker window. Repeat this procedure for the Middle and Bottom features of the Gradient Sky override. Then click 'Intensity Mode', and set it to Multiplier. Click Multiplier, and set the value to 1. This is i_a in equation (2). The ambient light is now achromatic and equally strong in all directions.

To create a directional light source, delete the Sun object in the Hierarchy view, and create a 'Light / Directional Light'. In the Inspector view for this light, in the Emission panel, set 'Light Appearance' to Color. In the same Emission panel, the color coordinates of the Color property are d_k in equation (2), and the Intensity property is i_d.

There is no single Unity parameter that corresponds to the lighting direction $\mathbf{l}$ in equation (2). Instead, the lighting direction is controlled by the Rotation parameters in the Inspector view of the directional light. With Rotation parameters $X = Y = Z = 0$, directional light travels from $-z$ to $+z$, i.e., in direction (0, 0, 1). From this starting point, the Rotation parameters (which are Euler angles) rotate the light source about the corresponding axes, e.g., $X = 50$ means a 50° rotation about the x-axis. There are three points to note about these rotations. First, Unity uses a left-handed coordinate system. Second, the z-axis rotation is applied first, then the x-axis rotation, and then the y-axis rotation. Third, the rotations are extrinsic, meaning that they are made around fixed axes, e.g., the z-axis rotation, which is first, does not change the orientations of the x- and y-axes that are used for the remaining two rotations. These facts, along with the requirement that $\mathbf{l}$ in equation (2) points in the direction the light comes from, imply that

$$\mathbf{l} = (-\cos X \sin Y, \sin X, -\cos X \cos Y) \tag{25}$$

where X, Y, and Z are the Rotation parameters of the directional light source. As an alternative, the Unity project render_random in the Supporting Information shows how to control lighting direction in a C# script using a direction vector instead of Euler

angles.

Next we create a Lambertian object. In the Hierarchy view, create a '3D Object / Sphere'. In the Assets view, create a Material and give it a name such as 'myLambertian'. In the Inspector view for this material, near the top, set the Shader to 'HDRP / ArnoldStandardSurface / ArnoldStandardSurface'. With its default settings, this material type is Lambertian. In the Inspector view for this material, click the color patch to the right of 'Base Color'. The three color channel values in the window that opens are the material color coordinates m_k in equation (2). Leave them at their default value of 1, and close the color picker window. Drag the material from the Assets view onto the sphere in the Hierarchy view. This assigns the material type to the sphere. View the sphere in the Inspector view, and at the bottom of the panel you should see that its material type is now myLambertian.

To create an object with an unlit material, follow the steps in the previous paragraph, but set the Shader to 'HDRP / Unlit'. In the Inspector view for this material, in the 'Surface Inputs' panel, the color coordinates of the Color property are m_k in equation (3).

In the Built-in Render Pipeline, Unity offers a choice between 'Gamma' and 'Linear' color spaces, and some of the nonlinearities in the model described here are applied only in Linear mode. At the time of writing, only the Linear color space is available in the HDRP.

References

Adobe Systems. (2013). *Cube LUT specification 1.0*.

Anderson, M., Motta, R., Chandrasekar, S., & Stokes, M. (1996). Proposal for a standard default color space for the internet – sRGB. In *The Fourth Color Imaging Conference: Color Science, Systems and Applications* (p. 238-245).

Brainard, D. H., Pelli, D. G., & Robson, T. (2002). Display characterization. In J. Hornak (Ed.), *Encyclopedia of imaging science and technology* (p. 172-188). Wiley.

Clausen, O., Fischer, G., Fuhrmann, A., & Marroquim, R. (2019). Towards predictive virtual prototyping: color calibration of consumer VR HMDs. In *Virtuelle und Erweiterte Realität: 16. Workshop der GI-Fachgruppe VR/AR* (p. 13-24). Shaker Verlag.

Creem-Regehr, S. H., Stefanucci, J. K., & Bodenheimer, B. (2022). Perceiving distance in virtual reality: theoretical insights from contemporary technologies. *Philosophical Transactions of the Royal Society B*, *378*(20210456), 1-12.

Diaz-Barrancas, F., Cwierz, H., & Pardo, P. J. (2021). Real-time application of computer graphics improvement techniques using hyperspectral textures in a virtual reality system. *Electronics*, *10*.

Diaz-Barrancas, F., Cwierz, H., Pardo, P. J., Perez, A. L., & Suero, M. I. (2020). Spectral color management in virtual reality scenes. *Sensors*, *20*.

Diaz-Barrancas, F., Rodriguez, R. Q., Bayer, F. S., Aizenman, A., & Gegenfurtner, K. R. (2024). High-fidelity color characterization in virtual reality across head mounted displays, game engines, and materials. *Optics Express*, *32*.

Duay, K., Mizokami, Y., & Nagai, T. (2025). VR HMD color calibration and accurate control of emitted light using Three.js. *Journal of Vision*, *25(2):4*, 1-20.

Epic Games. (2024). *Unreal Engine.* Retrieved from `https://www.unrealengine.com`

Geig, M. (2022). *Sams teach yourself Unity game development in 24 hours, fourth edition*. Sams Publishing.

Gil Rodríguez, R., Bayer, F., Toscani, M., Guarnera, D., Guarnera, G. C., & Gegenfurtner, K. R. (2022). Colour calibration of a head mounted display for colour vision research using virtual reality. *SN Computer Science*, *3*(1), 1–10.

Kim, A. S., Cheung, W.-C., Beams, R., & Badano, A. (2021). Color rendering in medical extended-reality applications. *Journal of Digital Imaging*, *34*, 16-26.

Kimura, K., Reichert, J. F., Olson, A., Pouya, O. R., Wang, X., Moussavi, Z., & Kelly, D. M. (2017). Orientation in virtual reality does not fully measure up to the real-world. *Scientific Reports*, *7*(1), 1-8.

Kleiner, M., Brainard, D. H., & Pelli, D. G. (2007). What's new in Psychtoolbox-3? *Perception*, *36*, Supplement.

McCluney, R. (2014). *Introduction to radiometry and photometry, second edition*. Artech House, Inc.

Murray, R. F., Patel, K. Y., & Wiedenmann, E. S. (2022). Luminance calibration of virtual reality displays in unity. *Journal of Vision*, *22(13):1*, 1-9.

Patel, K. Y., Wilcox, L. M., Maloney, L. T., Ehinger, K. A., Patel, J. Y., Widenmann, E., & Murray, R. F. (2024). Lightness constancy in reality, in virtual reality, and on flat-panel displays. *Behavior Research Methods*, *56*, 6389-6407.

Pharr, M., Wenzel, J., & Humphreys, G. (2023). *Physically based rendering, fourth edition*. The MIT Press.

Rzepka, A. M., Hussey, K. J., Maltz, M. V., Babin, K., Wilcox, L. M., & Culham, J. C. (2023). Familiar size affects perception differently in virtual reality and the real world. *Philosophical Transactions of the Royal Society B*, *378*(1869), 1-14.

Toscani, M., Gil, R., Guarnera, D., Guarnera, G. C., Kalouaz, A., & Gegenfurtner, K. R. (2019). Assessment of OLED head mounted display for vision research with virtual reality. In *15th Annual Conference on Signal-Image Technology and Internet-Based Systems (SITIS)* (p. 738-745).

Unity Technologies. (2022). *Unity, version 2022.3.* Retrieved from `https://unity.com`

Zaman, N., Sarker, P., & Tavakkoli, A. (2023). Calibration of head mounted displays for vision research with virtual reality. *Journal of Vision*, *23(6):7*, 1-20.