1 Supporting Information: Instructions for replicating analyses

This document gives detailed instructions on how to carry out the tests described in the main text. It shows how to generate data from Unity for tests of the render pipeline, but in order to make first attempts at replicating the tests easier to carry out, the Supporting Information also includes the data files and the resulting figures.

1.1 Estimating the scale constant c

The project render_random, located in the 'unity' directory of the Supporting Information, generates the data for this test. Add this project to the Unity Hub, and open the project. In the version of the project included in the Supporting Information, many files that Unity can reconstruct have been deleted in order to reduce the size of the repository. As a result, the first time you open the project, you will have to double-click the OutdoorsScene object in the Assets panel in order to make it the active scene.

When the project is open and OutdoorsScene is active, select the 'Main Camera' object in the Hierarchy view, and in the Inspector view there will be GUI elements you can use to configure the test. For this test, check the box labelled 'Test Lambertian', uncheck the box labelled 'Test Tonemap', and enter 1000 in the text box labelled 'Samples'. Run the project, and in the Game view you will see a plane take many random orientations and colors. The random scene parameters and rendered values are recorded in a text file named data_L1_T0.txt, which will appear in the directory where the render_random project is located. The digit after the letter L indicates whether the test was run with a Lamberitan material (0 = Unlit, 1 = Lambertian), and the digit after T indicates whether it was run with tonemapping (0 = no, 1 = yes). The data from the first run after opening the project often contains a few outliers, so I suggest making a first run and deleting the data file, then making a second run and using that data.

The analysis for this test is implemented in the Python script estimate_c.py. The module hdrp.py should be in the same directory as the script. Place the data file data_L1_T0.txt in the subdirectory data/tonemap_off below the script. Run the script. The script uses equations (2) and (4) in the main text, without the scale constant c, to predict the unprocessed values u_k from the random lighting and material parameters. The script uses linear regression to estimate the scale constant c, which it prints to the console. It also generates Figure 1 in the main text. All figures that can be generated using the instructions in this document are included in the 'figures' directory of the Supporting Information.

1.2 Testing model predictions for Lambertian material without tonemapping

This test is implemented in the script model_test_tonemap_off.py, and uses the same data file data_L1_T0.txt as the test in the previous section. Again, the module hdrp.py should be in the same directory as the script, and the data file data_L1_T0.txt should be in the subdirectory data/tonemap_off. The script has a boolean variable testLambertian that indicates whether the data to be tested comes from a run of render_random with a Lambertian material. Set testLambertian to True. Run the script. The script plots post-processed values v_k predicted by equations (2) and (4) in the main text against the actual post-processed values, and also plots the prediction error as a function of the post-processed values v_k, the material color m_k, and the directional light color d_k. This is Figure 2 in the main text.

In order to make the rendering conditions similar to those in an experiment, the script does not assign a new, random exposure setting to the scene on each sample. However, the exposure can be adjusted by choosing the Global Volume object in the Hierarchy view in the render_random project, and in the Inspector view entering a value for 'Fixed Exposure' in the Exposure panel. The project adjusts the random lighting intensities based on this value, so that the rendered scenes are neither too dark nor oversaturated. The exposure value is saved to the data file, and the analysis scripts take it into account when predicting rendered values. Results with different exposure values are similar to those with the default value of zero, which was used for the figures in the paper.

1.3 Testing model predictions for unlit material without tonemapping

To test the model for unlit materials, run the render_random project again, as in Subsection 1.1, but this time with the box labelled 'Test Lambertian' unchecked. This produces a data file data_L0_T0.txt, which contains results for an unlit material. Put this file in the subdirectory data/tonemap_off. Run the same script model_test_tonemap_off.py as in Subsection 1.2, this time with the variable testLambertian set to False. The script plots post-processed values v_k predicted by equations (3) and (4) in the main text against the actual post-processed values, and also plots the prediction error as a function of the post-processed values v_k. This is Figure 3 in the main text.

1.4 Estimating knot point coordinates u_i^* using delta functions

The data for this test is generated by the Unity project render_delta, located in the 'unity' directory. Load this project into the Unity Hub, open it, and double-click the OutdoorScene object to make it the active scene. Run the project. It will take around 20 minutes to finish, and the rendered scene will be simply black for much of the time. The project generates a text file data_delta.txt. The analysis is carried out by the script knots_from_delta.py. Put the file data_delta.txt in the subdirectory 'data' below the script, and run the script. The script generates Figure 4 in the main text, and prints estimates of the knot points to the console.

1.5 Estimating knot point coordinates u_i^* by optimizing model predictions

This analysis requires data from six runs of render_random with six different cube files used for tonemapping. For the first run, select 'Main Camera' in the Hierarchy view, and then in the Inspector view check the boxes for 'Test Lambertian' and 'Test Tonemap', and enter the value 1000 for 'Samples'. Next, select 'Global Volume' in the Hierarchy view, and then in the Inspector view, in the Tonemapping panel, for the 'Lookup Texture' field, choose linear_max1. This selects a cube file that specifies a linear mapping from rendered values u_k in the interval [0, 1] to tonemapped values in the interval [0, 1]. Run the project. This will create a text file data_L1_T1_linear_max1.txt. Move this file to the directory data/tonemap_on_fit.

Repeat this procedure an additional five times, with following five values for the 'Lookup Texture' field: square_max1, square_root_max1, linear_max58, square_max58, square_root_max58. Each of these cube files specifies a mapping from rendered values u_k to tonemapped values. The tags 'linear', 'square', and 'square_root' in the filenames indicate the mapping. Files ending with 1 map the interval [0, 1] to [0, 1], and are useful in the case described in the main text where we want to map the rendered range $u_k \in [0, 1]$ to the displayable range on the monitor. Files ending with 58 map u_k in the interval [0, 58] to [0, 1], and are useful for estimating the knot points u_i^*, which range from almost zero to approximately 58. (Readers who are interested in how I generated these cube files can examine the Python script make_cubes.py in the Supporting Information.) Each run of the project with a different cube file will generate a data file data_L1_T1_[name].txt, where [name] is replaced by the name of the cube file.

With all six data files in data/tonemap_on_fit, run the script knots_from_model.py. This script makes two passes through the data. In the first pass, it uses the data from runs of render_random with cube files whose names end in 58, to estimate all knot points u_i^*. The analysis uses data from all three cube files whose names end in 58, in order to use a range of tonemapping data to estimate the knot points. In the second run, it uses data from runs with cube files whose names end in 1, to fine-tune the knot points in the range [0, 1]. The script prints its final estimates of all knot points to the console, which are the values in Table 2(b) in the main text.

1.6 Testing model predictions for Lambertian material with tonemapping

This test requires new samples from render_random, of the same kind generated in the previous section. We could re-use the data from the previous section, but that would mean using the same data for estimating the knot points u_i^* and for testing model predictions based on those knot points, which could result in overfitting. Instead, re-run the procedure described in the first two paragraphs of the previous section, and put the resulting six data files in data/tonemap_on_test. Run model_test_tonemap_on.py with the variable testLambertian set to True. This produces Figure 6 in the main text.

1.7 Testing model predictions for unlit material with tonemapping

This test requires data like that used in the previous section, but from an unlit material. Re-run the procedure described in the first two paragraphs of Section 1.5, but with the 'Test Lambertian' box unchecked. Put the resulting six data files in data/tonemap_on_test, and run model_test_tonemap_on.py, with the variable testLambertian set to False. This produces Figure 7 in the main text.

1.8 Testing achromatic gamma correction

[run characterization routines in caldemo with tonemapping off; run char_achromatic_1.py to generate cube file; load cube file into caldemo; run characterization routines again with tonemapping on; run char_achromatic_2.py to check that luminance is a linear function of u_k]

1.9 Testing chromatic gamma correction

[run characterization routines in caldemo with tonemapping off; run char_chromatic_1.py to generate cube file; load cube file into caldemo; run characterization routines again with tonemapping on; run char_chromatic_2.py to check that chromaticity coordinates are a linear function of u_k]