ꟻLIP: A Difference Evaluator for Alternating Images

PONTUS ANDERSSON, NVIDIA
JIM NILSSON, NVIDIA
TOMAS AKENINE-MÖLLER, NVIDIA
MAGNUS OSKARSSON, Lund University
KALLE ÅSTRÖM, Lund University
MARK D. FAIRCHILD, Rochester Institute of Technology

Image quality measures are becoming increasingly important in the field of computer graphics. For example, there is currently a major focus on generating photorealistic images in real time by combining path tracing with denoising, for which such quality assessment is integral. We present ꟻLIP, which is a difference evaluator with a particular focus on the differences between rendered images and corresponding ground truths. Our algorithm produces a map that approximates the difference perceived by humans when alternating between two images. ꟻLIP is a combination of modified existing building blocks, and the net result is surprisingly powerful. We have compared our work against a wide range of existing image difference algorithms and we have visually inspected over a thousand image pairs that were either retrieved from image databases or generated in-house. We also present results of a user study which indicate that our method performs substantially better, on average, than the other algorithms. To facilitate the use of ꟻLIP, we provide source code in C++, MATLAB, NumPy/SciPy, and PyTorch.

CCS Concepts: • **Computing methodologies** → **Image processing**; *Rendering*.

Additional Key Words and Phrases: image difference, image metric

ACM Reference Format:
Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. 2020. ꟻLIP: A Difference Evaluator for Alternating Images . *Proc. ACM Comput. Graph. Interact. Tech.* 3, 2, Article 15 (August 2020), 23 pages. https://doi.org/10.1145/3406183

1 INTRODUCTION

In computer graphics, image quality metrics have been used to guide algorithmic development, to compare the performance of rendering techniques, and to improve on loss functions for the ever growing adoption of deep learning approaches to post-processing [Bako et al. 2017; Vogels et al. 2018; Zwicker et al. 2015]. The foremost use of these metrics have been to compare an approximate rendered image to a ground truth version of the same image. Image metrics are key for evaluating image quality and are hence important tools in our field.

Flipping, or alternating, between two seemingly similar images reveals their differences to an observer much more distinctly than showing the images side by side. Yet, algorithms that measure the differences between two images are often developed under the assumption that the images are

Authors' addresses: Pontus Andersson, NVIDIA; Jim Nilsson, NVIDIA; Tomas Akenine-Möller, NVIDIA; Magnus Oskarsson, Centre for Mathematical Sciences, Lund University; Kalle Åström, Centre for Mathematical Sciences, Lund University; Mark D. Fairchild, Munsell Color Science Laboratory, Rochester Institute of Technology.

shown next to each other or alternated between with an empty, e.g., a black, image displayed for a short period before the next image is shown. Both of these options rely on human memory and make differences harder for an observer to evaluate. While this might be desired in certain applications, rendering researchers usually prefer to compare results by alternating between two images to identify all differences. ꟻLIP aims at filling this gap in image difference methods. Furthermore, both the development and evaluation of ꟻLIP was centered around detailed examination of error maps. This is contrary to much of the previous work, where metrics were evaluated by their ability to explain users' response to stimuli and the error maps themselves were seldom considered.

As an image comparison algorithm, ꟻLIP carefully evaluates differences in color and edges based on, or inspired by, models of the human visual system. Developed mainly for rendering, it also pays attention to differences in point-like structures, such as fireflies, i.e., isolated pixels with colors that differ greatly from the color of their surroundings. In addition, ꟻLIP is designed to neglect differences in details that the observer cannot perceive under given viewing conditions.

ꟻLIP is a full-reference image difference algorithm, whose output is a new image indicating the magnitude of the perceived difference between two images at each pixel. Alternatively, for a more compact representation of the differences, the user can request ꟻLIP to pool the per-pixel differences down to a weighted histogram, or all the way down to a single value. Inspired by the iCAM framework [Fairchild and Johnson 2004], the model includes contrast sensitivity functions, feature detection models, and a perceptually uniform color space. We provide improvements to each of these building blocks to obtain an algorithm that, ultimately, presents differences that agree well with how human observers perceive them. We show that this is indeed the case via a user study, where alternating images, together with difference maps from multiple algorithms, were shown to subjects who were asked to rank the difference maps based on how well the maps agreed with what the subjects observed when looking at the alternating images. Our user study shows that ꟻLIP performs well for several image types in addition to rendered ones, including natural images and images generated by artificial neural networks.

Our main contribution lies in a simple-to-use algorithm for capturing and visualizing the difference observed when flipping between two images. The algorithm has been battle-tested using a large set of image pairs against a substantial set of existing methods. Additionally, we provide source code for ꟻLIP in C++, MATLAB, NumPy/SciPy, and PyTorch.

This paper is structured as follows. In Section 2, we outline the design goals and limitations of ꟻLIP. Then, in Section 3, we discuss previous work in the field of image comparisons and, in Section 4, we introduce our algorithm. Our proposed way of pooling the per-pixel differences is described in Section 5. In Section 6, we analyze the difference maps generated by ꟻLIP, outline our user study, and present the results, before we conclude the paper in Section 7.

2 GOALS AND LIMITATIONS

In this section, we aim to frame ꟻLIP by noting the design goals of the algorithm and its limitations. This section should make it clear what ꟻLIP strives to accomplish and what it does not.

The core goal of ꟻLIP is to present a per-pixel difference map between two images, where the differences are approximately proportional to those seen by a human observer when flipping back and forth between the images, located in the same position and without blanking in between. This implies that the differences should depend on the observer's distance to the display as well as the display's pixel size. Furthermore, it requires the color distances to be computed in a space where they are proportional to the distances perceived by the observer, i.e., in a perceptually uniform color space. In addition, there is a need for certain attention to differences in edge content, where even small differences in color can lead to large perceived differences due to the presence of edges. Consideration should also be taken to differences in point content as these often occur

in a rendering context due to, e.g., fireflies. Finally, for rendered images, the emphasis on flipping is truly important, since this is the way rendering researchers often evaluate image quality of an algorithm, i.e., by flipping back and forth and spotting all differences.

ꟻLIP is designed to compute differences based on the criteria above while maintaining ease-of-use by keeping the complexity low and the need for user-specific parameter options minimized. In this case, the cost of simplicity is the situational need for less complex models, which are potentially less accurate than their more complex counterparts. For example, ꟻLIP does not use the most accurate color metric as it would require the user to input viewing conditions that are non-trivial to measure. Additionally, the algorithm is not designed to handle high-dynamic range input.

Our model's main limitation is its inability to detect masking [Chandler 2013], which can lead to overestimation of differences when they are not perceived due to other image content masking them out. This is a difficult problem to solve, however, as there are multiple components and effects to consider, and doing so would require a masking function that depends on the viewing conditions. To our knowledge, such a masking detection function does not exist.

We recommend to use ꟻLIP for evaluating full, per-pixel difference maps, as such maps comprise complete information about the differences' location and magnitude. We do, however, appreciate the need for pooled difference values during, e.g., larger scale comparisons, training of artificial neural networks, or automated testing of algorithms. Therefore, we suggest different levels of pooling, either down to a weighted histogram or all the way down to a single value.

3 PREVIOUS WORK

Assessment of a distorted image's quality can be done in multiple ways. On a high level, the methods of assessment can be divided into full-reference, reduced-reference, and no-reference algorithms. The difference between the three paradigms is the extent to which a non-distorted version of the image, i.e., a *reference*, is used. Full-reference methods use the entire reference image, while reduced-reference use parts of it, and no-reference algorithms operate without any reference. ꟻLIP is a full-reference algorithm and this section will focus on those. For a survey, including reduced-reference and no-reference methods, we refer to the work by Chandler [2013].

The result of an image quality assessment can contain much or little spatial information. Some algorithms generate a single, overall quality value for the image. This value often corresponds to the overall degree of distortion [Alakuijala et al. 2017; Prashnani et al. 2018; Wang et al. 2003]. At the other end of the spatial information spectrum is the option of presenting one value per pixel, which opens up the possibility to analyze where the distortions are visible [Mantiuk et al. 2011; Wolski et al. 2018], and what their magnitudes are in each pixel [Fairchild and Johnson 2004; Johnson and Fairchild 2003; Wang et al. 2004; Zhang et al. 2018; Zhang and Wandell 1997]. In our comparisons, we will only include algorithms that originally yield per-pixel values or can be naturally modified to do so. There has been a vast amount of research conducted in the field of image quality assessment and we cannot cover it all in this paper. Therefore, we have selected a subset of the available methods, including several classic ones as well as the ones that are state of the art, to briefly review here and to be subject to comparison with ꟻLIP. We note that neither of these were developed under the viewing protocol that ꟻLIP targets, i.e., alternating images with no blank image shown between the flips.

Symmetric mean absolute percentage error (SMAPE), described in the paper by Vogels et al. [2018], is a simple statistical method that computes the difference between the reference and distorted image and divides by the sum of absolute values. It has seen use in, e.g., the training of neural networks used for denoising. SSIM [Wang et al. 2004] is an index based on local image statistics. It produces a per-pixel similarity value related to the local difference in the average value, the variance, and the correlation of luminances. S-CIELAB [Johnson and Fairchild 2003; Zhang and Wandell 1997]

takes parts of the human visual system (HVS) into account in its quality assessment by filtering the images using contrast sensitivity functions and calculating the difference between the distorted and the reference image in a perceptually uniform color space. FLIP adopts and improves on these building blocks. Another method based around properties of the HVS is HDR-VDP-2 [Mantiuk et al. 2011]. Unlike the mentioned algorithms, HDR-VDP-2 attempts to predict the visibility of distortions, i.e., the probability that a distortion is detectable in a pixel, rather than the magnitude of the distortion. It does this by modeling several, complex parts of the HVS. While developed for high-dynamic range images, the algorithm can also handle low-dynamic range input.

In recent years, researchers have begun using artificial neural networks and deep learning (DL) to aid in the assessment of image quality. Zhang et al. [2018] discovered that, during image classification and other related image processing, the intermediate image representation produced by the network could, advantageously, be used for comparison with other images. Shortly thereafter, Prashnani et al. [2018] presented PieAPP, which is a DL framework where the network is tasked with mimicking human preference in regards to which of two distorted images is more similar to a reference. This is done by having the network compute a score for each of the distorted images and then compare the two scores. Once trained, the second distorted image can be neglected and the score for the first image can be used as a measure of its quality. PieAPP was not originally targeted toward computing a value per pixel, but we have adapted it so that it can (Section 6). Wolski et al. [2018] exploit user markings on distorted images (when compared to reference images), where the users indicate the locations in the images where they observe distortions. In that study, the users performed the marking when shown the distorted and the reference images next to each other, rather than during flipping. The target of the work, similar to that of HDR-VDP-2, was to produce a visibility map, i.e., a per-pixel map indicating where distortions are likely to be observed. This was done by training a new metric based on a custom convolutional neural network.

As part of the Guetzli system to optimize JPEG compression [Alakuijala et al. 2017], the aim of the Butteraugli metric is to detect if an image includes noticeable compression artifacts. This is done by examining the largest value in the generated difference map. However, considering the map in its entirety could still be beneficial, as it provides indications of where distortions are visible and how severe they are. There is no published description of the Butteraugli method, except for what the source code suggests.

4 ALGORITHM

The goal of FLIP is to compute an image that is proportional to the difference perceived by a human when alternating between two images on top of each other (see Section 2). This viewing protocol was adopted throughout the development of FLIP and, as such, it permeates the entire algorithm. For an evaluator of image differences under a different viewing protocol, other design choices would likely, but not necessarily, be made.

Henceforth, we will focus on differences between a *reference* and a *test* image, where the latter usually is an approximation of the former, generated, e.g., by a less complex algorithm or subject to some form of distortion. Thus, the image computed by FLIP can be interpreted as an *error map*, where pixel values are approximately proportional to the perceived error in the test image.

FLIP is inspired by the iCAM framework [Fairchild and Johnson 2004] and is outlined in Figure 1. As can be seen in the figure, there are two parallel pipelines: a color pipeline and a feature pipeline, whose purposes are to compute a color difference and a feature difference, respectively. These are combined in a final step at the end of the pipeline. In Appendix A, we present pseudocode for FLIP.

In the color pipeline, we first apply spatial filters to each image. These filters are based on the human visual system's contrast sensitivity functions (CSFs) and they are applied to remove high-frequency details that we cannot perceive at a given distance from a given display. After that, we

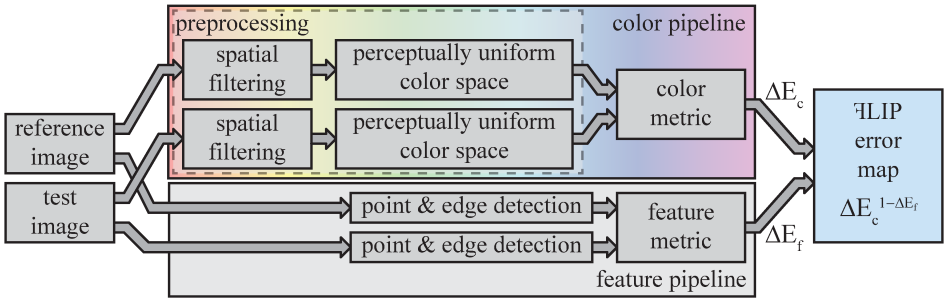

Fig. 1. The ꟻLIP pipeline.

transform the images' pixel values into the perceptually uniform color space (PUCS) L*a*b* [CIE 1978], where we have one achromatic and two chromatic components. The L*a*b* space was one of the first proposed PUCS and, while it is simple and easy to use, it is not without flaws [Fairchild 2013]. In particular, it does not handle the Hunt effect, i.e., the fact that color differences appear larger at higher luminance than lower [Hunt 1952]. We address this by performing a slight adjustment of the colors in the filtered images. Once filtered and adjusted, we compare the images' colors on a per-pixel basis. The color difference between two pixels is computed in the adjusted L*a*b* space with a metric that yields differences that agree better with our perception than, e.g., standard Euclidean distance in the RGB space. Once computed, the differences are mapped to the target $[0, 1]$ range. We also perform feature detection, where we localize edges and points in both images. The per-pixel difference in feature content is then used to enhance the color differences, yielding the final ꟻLIP error map. In the following, we describe the pipeline's building blocks in more detail.

4.1 Color Pipeline

During the preprocessing step (see Figure 1), both the reference and the test images are filtered such that details that cannot be perceived under the given viewing conditions are removed and, thus, do not affect the final error map. Both images are also adjusted to accommodate for the Hunt effect so that chromatic errors become smaller with lower luminance.

We assume that the input images are stored in sRGB space, where a pixel's color is denoted $\{R_s, G_s, B_s\}$, with R_s, G_s, and B_s representing the red, green, and blue color channels, respectively.

4.1.1 Spatial Filtering. As filtering must be done in a linear space, we first linearize the sRGB colors using standard linearization formulae [Poynton 2003], resulting in linear RGB colors, $\{R', G', B'\}$. ꟻLIP's spatial filtering is carried out in opponent color space, where the CSFs are defined and where there is one achromatic channel, one red-green channel, and one blue-yellow channel. Several such opponent spaces have been proposed. The one used in the S-CIELAB research [Johnson and Fairchild 2003; Zhang and Wandell 1997] leads to undesirable color shifts during filtering, where achromatic images may become chromatic. To avoid this, we choose to instead use the $Y_y c_x c_z$ space [Flohr et al. 1993], a linearized version of L*a*b* , which has been shown to perform well as an opponent space [Monga et al. 2003]. Here, Y_y contains the achromatic information, c_x the red-green information, and c_z the blue-yellow information. From linear RGB space, the transformation to $Y_y c_x c_z$ is done via the XYZ tristimulus values [Flohr et al. 1993] and with the D65 standard illuminant with luminance normalized to 1.0 [Poynton 2003].

There is one CSF per opponent channel, denoted S_{Y_y}, S_{c_x}, and S_{c_z}, for the achromatic and the two chromatic channels, respectively. Each CSF serves to tell how sensitive we are to spatial changes

Table 1. Table of CSF constants used by Johnson and Fairchild [2003]. In their publication, the b_{21} value is erroneously rounded to 0. The actual value, as confirmed by the authors, was 0.00001.

Parameters	Values		
a_{01}, b_{01}, c_{01}	75.0000,	0.20000,	0.8000
a_{11}, b_{11}, c_{11}	109.1413,	0.00040,	3.4244
a_{12}, b_{12}, c_{12}	93.5971,	0.00370,	2.1677
a_{21}, b_{21}, c_{21}	7.0328,	0.00001,	4.2582
a_{22}, b_{22}, c_{22}	40.6910,	0.10390,	1.6487

in the corresponding channel [DeValois and DeValois 1990]. The CSFs measure sensitivity as a function of cycles per degree of visual angle. For a pixel-based display setting, we relate cycles to pixels by assuming that one cycle corresponds to two pixels. We compute the number of pixels per degree (PPD), p, of a given monitor of size $W_\mathrm{m} \times H_\mathrm{m}$ square meters and resolution $W_\mathrm{p} \times H_\mathrm{p}$ pixels, for an observer at distance d meters from the monitor, as

$$p = d \frac{W_\mathrm{p}}{W_\mathrm{m}} \frac{\pi}{180}, \tag{1}$$

where we have assumed that the aspect ratio of a pixel is 1:1, so the number of PPD in the x and y directions is the same. Assuming a monitor of size $W_\mathrm{m} \times H_\mathrm{m} = 0.69 \times 0.39$ m^2 and (4k) resolution $W_\mathrm{p} \times H_\mathrm{p} = 3840 \times 2160$ pixels, with an observer at $d = 0.70$ meters from the monitor, Equation 1 yields $p = 67$ PPD. This is the default setup used in our experiments. At a larger number of PPD, we are less sensitive to high frequencies. For example, a black/white checkerboard will look gray at a large distance from the monitor or when shown with very small pixels.

We base the CSFs on the work of Movshon and Kirpes [1988] and Johnson and Fairchild [2003]. Note that these CSFs were not measured for the $\mathrm{Y_y c_x c_z}$ opponent space and are thus not completely valid within it. However, the inaccuracy caused by this is negligible in the larger context of the difference evaluator [Johnson and Fairchild 2003]. The achromatic CSF is a band-pass filter, while the chromatic ones are low-pass. Despite its band-pass nature, we choose to change the achromatic CSF into a low-pass filter so that it retains the DC component, as described by Johnson and Fairchild [2001], and does not modulate lower frequencies. That is, we set the achromatic CSF, $S_{\mathrm{Y_y}}$, to 1 for each frequency lower than the frequency corresponding to the peak of the original band-pass filter. Furthermore, each CSF is normalized so that its largest value is 1. With these changes, the three CSFs, $S_{\mathrm{Y_y}}$, $S_{\mathrm{c_x}}$, and $S_{\mathrm{c_z}}$, are given by

$$\begin{aligned}
S_{\mathrm{Y_y}}(f) &= \begin{cases} \hat{S}_{\mathrm{Y_y}}(f) / \hat{S}_{\mathrm{Y_y}}\left(\frac{c_{01}}{b_{01}}\right), & \text{if } f \geq \frac{c_{01}}{b_{01}}, \\ 1, & \text{otherwise,} \end{cases} \\
S_{\mathrm{c_x}}(f) &= \left(a_{11} e^{-b_{11} f^{c_{11}}} + a_{12} e^{-b_{12} f^{c_{12}}}\right) / (a_{11} + a_{12}), \\
S_{\mathrm{c_z}}(f) &= \left(a_{21} e^{-b_{21} f^{c_{21}}} + a_{22} e^{-b_{22} f^{c_{22}}}\right) / (a_{21} + a_{22}),
\end{aligned} \tag{2}$$

where $\hat{S}_{\mathrm{Y_y}}(f) = a_{01} f^{c_{01}} e^{-b_{01} f}$ and where the maximum points were found with differentiation of the unnormalized functions. The constants are shown in Table 1.

For faster filtering, especially on GPUs, we transform the CSFs from the frequency domain to the spatial domain. One way of accomplishing this is by approximating the CSFs in Equation 2 with a sum of Gaussians. To avoid ringing artifacts in our images, which may result from directly using the CSFs in Equation 2, we instead approximate the CSFs by one or two zero-centered Gaussians for each channel.

In the continuous frequency domain (f), a Gaussian can be described by

$$G(f) = ae^{-bf^2}, \tag{3}$$

which can be inverse Fourier transformed to the spatial domain (x), resulting in

$$g(x) = a\sqrt{\frac{\pi}{b}}e^{-\frac{\pi^2}{b}x^2}. \tag{4}$$

We approximate the CSFs in Equation 2 with Gaussians (Equation 3), found with optimization functions in MATLAB. To approximate the luminance and red-green CSFs, $S_{\mathrm{Y_y}}$ and $S_{\mathrm{c_x}}$, respectively, one Gaussian suffices and we found $b_{\mathrm{Y_y}} = 0.0047$ and $b_{\mathrm{c_x}} = 0.0053$. Since the filters are later normalized in the spatial domain, the amplitudes for these can be neglected. For the blue-yellow CSF, $S_{\mathrm{c_z}}$, a satisfactory fit requires the sum of two Gaussians. We found $a_{\mathrm{c_z},1} = 34.1$ and $b_{\mathrm{c_z},1} = 0.04$ for the first Gaussian and $a_{\mathrm{c_z},2} = 13.5$ and $b_{\mathrm{c_z},2} = 0.025$ for the second.

Following the work by Johnson and Fairchild [2003], the frequency domain CSFs should be evaluated for frequencies in the range $\left[-\frac{p}{2}, +\frac{p}{2}\right]$, where $\frac{p}{2}$ corresponds to half the sampling frequency, F_s. The spacing, Δ, between two samples in the spatial domain is then given by

$$\Delta = \frac{1}{F_s} = \frac{1}{p}. \tag{5}$$

Next, we determine the radius, r, of our spatial filter. To simplify implementations, we choose to use the same radius for each of the three CSF filters. For a single Gaussian with standard deviation σ, a filter with radius 3σ is known to retain 99.7% of the function's energy, which we deem sufficient for our spatial filter kernels. For the blue-yellow CSF, where we sum two Gaussians with different amplitudes, this result does not hold. However, by using the radius corresponding to the Gaussian with largest standard deviation, 99.7% (or more) of the energy is still preserved. On the form where it includes the standard deviation, σ, a Gaussian is proportional to $e^{-x^2/(2\sigma^2)}$ and, comparing to Equation 4, we get $\sigma = \sqrt{\frac{b}{2\pi^2}}$. The radius (in pixels), r, of the spatial filter becomes

$$r = \left\lceil \frac{3\sigma}{\Delta} \right\rceil = \lceil 3\sigma_{\max} p \rceil, \tag{6}$$

where $\sigma_{\max} = \sqrt{\frac{b_{\max}}{2\pi^2}}$ and $b_{\max} = \max\{b_{\mathrm{Y_y}}, b_{\mathrm{c_x}}, b_{\mathrm{c_z},1}, b_{\mathrm{c_z},2}\} = 0.04$. We evaluate Equation 4 at $\{0, \pm\Delta, \pm 2\Delta, \ldots, \pm r\Delta\}$ in order to generate the spatial filter kernel. The approximated CSFs are shown in Figure 2. To go from one-dimensional to two-dimensional filters, we assume rotational symmetry and, for each location, (x, y), $x, y \in \{0, \pm 1, \pm 2, \ldots, \pm r\}$, in the filters, we evaluate Equation 4 at $d(x, y) = \sqrt{(x\Delta)^2 + (y\Delta)^2} = \Delta\sqrt{x^2 + y^2}$. Finally, we normalize the filters such that each filter's weights sum to one. With the CSFs described, we can now perform the filtering of our images by convolving each channel with the respective two-dimensional spatial CSF filter. Since we use different Gaussians for the different channels and the RGB cube is not axis-aligned in $\mathrm{Y_y c_x c_z}$ space, this operation could result in colors outside the RGB cube. To retrieve filtered colors within a known range, which is essential for our color metric (see Section 4.1.3), we transform the filtered colors, $\{\widetilde{\mathrm{Y}}_\mathrm{y}, \widetilde{\mathrm{c}}_\mathrm{x}, \widetilde{\mathrm{c}}_\mathrm{z}\}$, to linear RGB and clamp them to the $[0, 1]^3$ cube before proceeding.

4.1.2 Perceptually Uniform Color Space. The idea behind a perceptually uniform color space is that the numeric distances between colors in such a space agree with the perceived distance between them [Wyszecki and Stiles 1982]. Next, we therefore transform the clamped, filtered linear RGB colors, found in Section 4.1.1, to L*a*b* values, $\left\{\widetilde{L}^*, \widetilde{a}^*, \widetilde{b}^*\right\}$, again via the XYZ tristimulus values and with the D65 standard illuminant with luminance normalized to 1.0 [Poynton 2003].

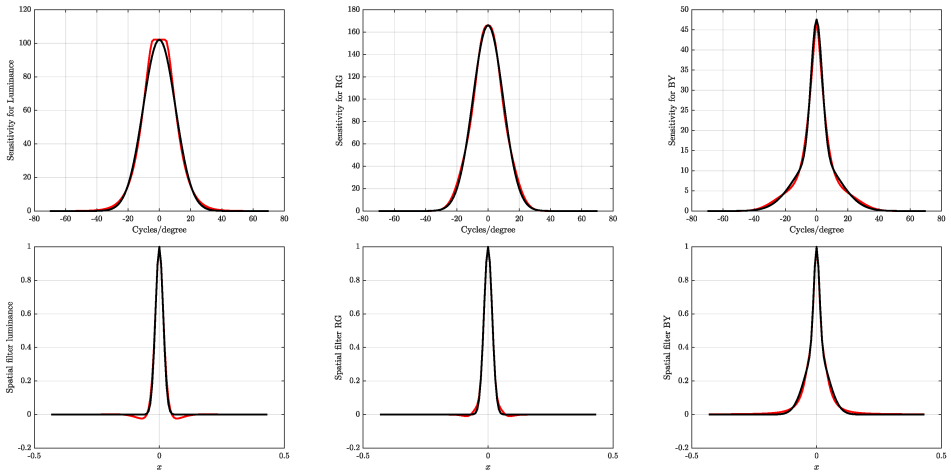

Fig. 2. Top: the original CSFs (Equation 2) are shown in red and our approximation, using a single Gaussian, or two in the case of the blue-yellow channel, in black. Bottom: the red curves are inverse discrete Fourier transformed versions of the CSFs in Equation 2, but they have negative lobes, which we wish to avoid. The black curves are the Gaussians from the top row, analytically inverse Fourier-transformed to the spatial domain. These do not include any negative lobes and are the ones used by ꟻLIP.

Fig. 3. Left: two colors with the same, high luminance, where the second color was generated using a rotation of the chrominance of the first. Right: two colors with the same, low luminance, with the same chrominance components as the colors in the first color pair. The difference between the colors is perceived as larger in the left pair due to the Hunt effect.

With higher luminance, chromatic differences appear larger. This is related to the Hunt effect [Hunt 1952], which is an important effect to consider in color distance computations. Figure 3 illustrates this. The chrominance components of both color pairs in the figure are the same, but in the first pair, the luminance level is higher than in the latter. We see how the intra-pair difference appears larger in the first pair, even though the Euclidean distance, which is the standard metric in the L*a*b* space, between both pairs is the same. This is because the Hunt effect is not accounted for in the original L*a*b* space [Fairchild 1996]. To account for this, without adopting a more complex color space and metric, we lower the distance between the chrominance components at lower luminance levels by pushing them toward the luminance axis, i.e.,

$$\left\{L^*_{\mathrm{h}}, a^*_{\mathrm{h}}, b^*_{\mathrm{h}}\right\} = \left\{L^*, f(L^*, a^*), f(L^*, b^*)\right\}, \tag{7}$$

where we use the h subscript to denote Hunt-adjusted colors and $f(L^*, c) = \frac{L^*}{100}c$, with $c \in \{a^*, b^*\}$. In the L*a*b* space, the luminance, L^*, is in the $[0, 100]$ range, implying that $f(L^*, c) \in [0, c]$. Furthermore, we note that the luminance component is unaffected by the Hunt adjustment. Applying the Hunt-adjustment to our filtered colors, $\left\{\widetilde{L^*}, \widetilde{a^*}, \widetilde{b^*}\right\}$, transforms them to $\left\{\widetilde{L^*_{\mathrm{h}}}, \widetilde{a^*_{\mathrm{h}}}, \widetilde{b^*_{\mathrm{h}}}\right\}$.

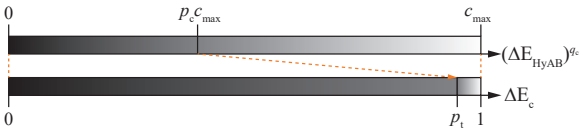

Fig. 4. Illustration of our error redistribution, compressing large color distances into a smaller range.

4.1.3 Color Metric. Most metrics in perceptually uniform color spaces, including the Euclidean distance, originally suggested for the L*a*b* space [CIE 1978], are only valid if the distance between the colors is small. However, in rendering applications, we often have large differences due to, e.g., fireflies or differences in the antialiasing techniques used. Hence, we have selected HyAB—a metric designed to handle larger color distances—as our core color metric [Abasi et al. 2020]. In the L*a*b* space, the HyAB distance, ΔE_{HyAB}, between two colors, $R = \{L^*_R, a^*_R, b^*_R\}$ and $T = \{L^*_T, a^*_T, b^*_T\}$, is given by

$$\Delta E_{HyAB}(R, T) = |L^*_R - L^*_T| + \sqrt{\left(a^*_R - a^*_T\right)^2 + \left(b^*_R - b^*_T\right)^2}. \tag{8}$$

Translating the $[0, 1]^3$ RGB cube, which contains all the colors possible in our input images, to the L*a*b* space, the largest HyAB distance occurs between blue ($\{0, 0, 1\}$ in RGB) and green ($\{0, 1, 0\}$ in RGB) and is $\hat{c}_{max} = 308$. For comparison, the distance between black and white is 100. If the intra-pair differences in two color pairs are perceived as very large, it is difficult to determine which colors are further apart (is blue and green three times further apart than black and white, as the HyAB distance implies?). To reduce the gap between large differences in luminance versus large differences in chrominance, we propose to not simply divide our computed HyAB distances by $\hat{c}_{max}$ to map them into the $[0, 1]$ range, but instead compress large distances into the top of the $[0, 1]$ range and map smaller distances to the remaining part of it.

Inspired by the work of Huang et al. [2015], we raise our computed distances to the power of $q_c = 0.7$, implying that the distance between two Hunt-adjusted, filtered colors (Section 4.1.2), $\widetilde{R}_h$ and $\widetilde{T}_h$ is $\left(\Delta E_{HyAB}\left(\widetilde{R}_h, \widetilde{T}_h\right)\right)^{q_c}$. Before performing the $[0, 1]$ mapping of the HyAB distance, we note that, due to the Hunt adjustment, our blue and green colors are modified (but are still the two colors yielding the maximum distance) and the distance between them was found to be $c_{h,max} = 203$. The maximum distance, or error, between two colors in our images is then, after application of the power function, $c_{max} = \left(c_{h,max}\right)^{q_c} = 41$.

Finally, we map the distance to the $[0, 1]$ range. As mentioned above, larger errors will be compressed into the top of the range, while smaller errors will occupy a larger part of it. We set two parameters, $p_c, p_t \in [0, 1]$, to determine the mapping, which is illustrated in Figure 4. Errors in the range $[0, p_c c_{max})$ are mapped linearly to the range $[0, p_t)$, while errors in the range $[p_c c_{max}, c_{max}]$ are mapped linearly to the range $[p_t, 1]$, where compression of the large error range is achieved when $p_c < p_t$. For ꟻLIP, we use $p_c = 0.4$ and $p_t = 0.95$. The result of this mapping, i.e., the final color difference after spatial filtering, Hunt-adjustment, and re-mapping, is denoted ΔE_c.

4.2 Feature Pipeline

Edges are particularly well detected by the human visual system [McIlhagga 2018] and, as a consequence, edge comparisons are often part of image difference calculations [Fairchild and Johnson 2004]. In addition, differences in points are of importance in the evaluation of rendering algorithms. Because of this, we developed a feature detection mechanism that captures the presence of edges and points in images. We run the detector on both the reference and the test image, compare the outputs, and use the results to amplify the color differences computed in Section 4.1.

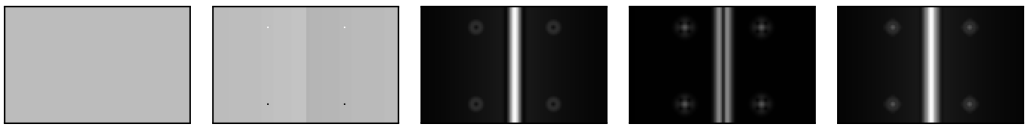

Fig. 5. From left to right: 1) the reference image, 2) the test image, including a small edge generated by a short ramp on either side and some salt and pepper noise, 3) the edge difference (first argument in Equation 9), 4) the point difference (second argument in Equation 9), and 5) the maximum of the edge and point difference in each pixel. Note that all differences are scaled for visibility.

If there is a feature in one image, but not the other, or if the feature is more pronounced in one of the images, we enhance the color differences. Similar to the spatial filtering, the feature detection depends on the number of pixels per degree of visual angle (PPD), i.e., the observer's distance to the display and the size of the display's pixels.

4.2.1 Feature Detection. The first step of our feature pipeline is transforming the reference and the test image from linear RGB to the $Y_y c_x c_z$ opponent space, as described in Section 4.1.1. Feature detection is then performed on their achromatic (Y_y) channel, in which most of the high-frequency spatial information is contained [Mullen 1985], normalized to the $[0, 1]$ range by addition with 16 followed by division with 100. The detection is done by convolving the achromatic images with kernels that depend on the feature (edge or point) we want to detect as well as the number of PPD, p. For both detectors, to obtain feature values in the $[0, 1]$ range, we normalize the sum of the positive kernel weights to 1 and the negative weights to -1.

ꟻLIP detects edges in images by convolving them with the first partial derivatives of symmetric, two-dimensional Gaussians, resembling how edge detection is done by the human visual system [McIlhagga 2018]. Here, the width of the Gaussian partial derivatives is dependent on the number of PPD, p, computed in Equation 1. Given that the edge detector in the human visual system has a width, from the highest to the lowest amplitude value, of $w = 0.082$ degrees [McIlhagga 2018], the standard deviation of our digital edge detection filters (in pixels) is $\sigma(w, p) = \frac{1}{2}wp$. The radius of the kernels is $\lceil 3\sigma(w, p) \rceil$ pixels, similar to the radius of the kernels used in Section 4.1.1. Performing convolutions of an image, $\mathbf{I}$, with the x and y partial derivatives, respectively, yields a per-pixel gradient image. The magnitude of the gradient in each pixel is then the *edge feature value* and the full magnitude map is denoted $\|\nabla \mathbf{I}\|$.

Our point detection is done similarly, but with the second partial derivative instead of the first. The standard deviation of the point detection filter is the same as the one used for edge detection, meaning that it too depends on the number of PPD. The effect of this is that structures larger than one pixel are considered to be points at a larger number of PPD, which is also what they are then perceived as. As with the edge detection, we perform two convolutions of the image, $\mathbf{I}$, in this case with the second partial derivative in the x and the y directions, respectively. With the results, we compute a per-pixel magnitude map, denoted $\|\nabla^2 \mathbf{I}\|$, where each magnitude is a *point feature value.*

4.2.2 Feature Metric. Both edge and point differences are noticeable as we flip between images. Furthermore, they are independent in the sense that edges and points do not occur in the same pixels. This is illustrated in Figure 5 where we see how the edge error is large along the edge, but small on the pixels containing salt and pepper noise, while the opposite is true for the point error. Based on this, we choose the feature difference, ΔE_f, between pixels in the reference image, $\mathbf{R}$, and the test image, $\mathbf{T}$, to be the maximum of the differences in edge and point feature values, i.e.,

$$\Delta E_f = \left(\frac{1}{\sqrt{2}} \max \left(\left| \|\nabla \mathbf{R}\| - \|\nabla \mathbf{T}\| \right|, \left| \|\nabla^2 \mathbf{R}\| - \|\nabla^2 \mathbf{T}\| \right| \right) \right)^{q_f}, \tag{9}$$

Fig. 6. From left to right: 1) reference image, 2) test image, including aliasing artifacts, 3) color error, ΔE_c, produced by the pipeline in Section 4.1, and 4) color error enhanced based on feature differences (Equation 10). The heatmap we use is shown to the right, where bright yellow corresponds to maximum error and dark to minimum error. Note that this heatmap was selected since its luminance is a linear ramp from dark to bright. The images are have been enlarged to more clearly show the differences.

where pixel indices are omitted for brevity and $q_f = 1/2$ is used to increase the impact of the feature difference on the final ꟻLIP error. With the division by $\sqrt{2}$, the feature error obtained in Equation 9 is in the $[0, 1]$ range. In Figure 6, we present an example showcasing how the enhancement of color differences, based on differences in feature content, affects the ꟻLIP error image. By enhancing the color errors, we obtain an error map which agrees better with the errors we perceive between the reference and test image than the one showing only the color error, ΔE_c.

4.3 Final Difference Map

With the per-pixel color and feature differences, ΔE_c (Section 4.1) and ΔE_f (Section 4.2), respectively, the ꟻLIP difference value, ΔE, per pixel, is given by

$$\Delta E = (\Delta E_c)^{1-\Delta E_f}. \tag{10}$$

By raising the color difference to the power of one minus the feature difference, both of which are in the $[0, 1]$ range, a feature difference can only increase the color difference. If there are no feature differences, the ꟻLIP value is completely determined by the difference in color between the images. Furthermore, if the colors in a pixel, after filtering, are identical in the reference and test image, the error is minimal and equal to 0. Maximum error for a pixel is generated if the pixel is blue in the filtered version of one image and green in the other, i.e., if we have maximum color difference, or if there is a feature difference such that $\Delta E_f = 1$, which would imply that the exponent in Equation 10 is zero and, subsequently, that $\Delta E = 1$. Finally, the model parameters q_c, q_f, p_c, and p_t, were chosen based on visual inspection of the produced error maps for a large set of image pairs.

5 POOLING

Pooling is the process of condensing an error image down to a smaller set of values or, in the most extreme case, to a single value. As with any irreversible compression of data, information is inevitably lost in this process. If possible, it is thus best avoided, but in some cases, e.g., for large scale image analysis and deep learning loss functions, a single value is often required. In this section, we present a set of pooling methods of different granularity.

As a first step to reduce the amount of data, we create a histogram of all pixels' ꟻLIP error (Equation 10). In a classic histogram, a pixel with, e.g., error 0.2 counts as much as a pixel with error 0.4, i.e., they contribute with the same amount of height to the diagram. To better convey what error levels contribute the most to the total error in the image, we instead use a weighted histogram, where the count in each of the histogram's buckets is multiplied by the ꟻLIP value in the middle of the corresponding bucket. For easier comparison of such weighted histograms between image pairs with different resolutions, we divide the weighted bucket count by the number of megapixels

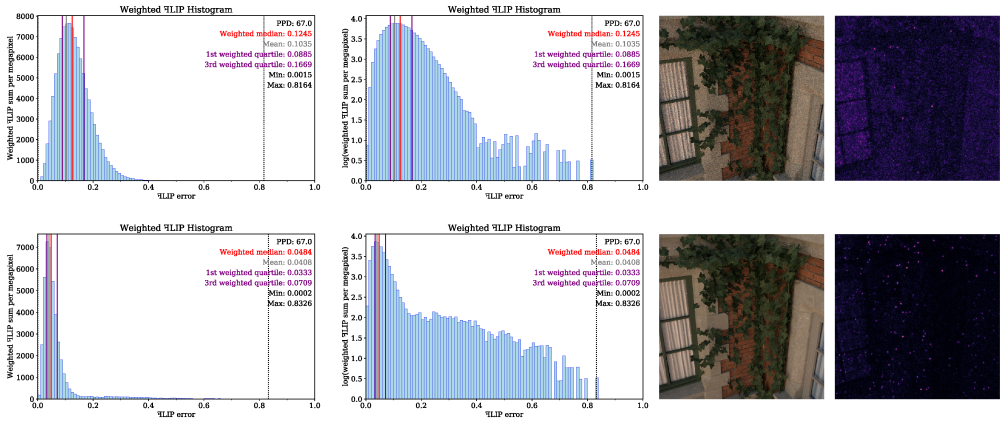

Fig. 7. Top row, from left to right: 1) a weighted ꟻLIP histogram, generated using the third image on this row, showing the weighted median, the mean, the 1st and 3rd weighted quartiles, and the minimum and the maximum value. 2) same as to the left, except that the y-axis is in log space, resulting in the larger errors (with low counts) becoming more pronounced, which may be beneficial in some cases. In the left diagram, it can be seen that the area to the left of the weighted median is the same as the area to the right. 3) the path traced test image with 32 samples per pixel (SPP). 4) the ꟻLIP error map, which was computed against a path traced reference image with 2^{22} SPP. Bottom row: same as the top row, except that the test image was path traced using 1,024 SPP. Note that the image on the bottom row has more pixels with medium to high errors and that this is clearly visible only in the log diagram.

in the image. In our open source C++ implementation of ꟻLIP, we generate the histograms both as a comma separated value (CSV) file and as a Python file that generates, using `pyplot`, weighted histograms like the ones shown in Figure 7. Optionally, logarithmic scale can be used on the y-axis. The reason for having this option is that, for rendered images, there is often an abundance of pixels with very low errors and the remaining pixel errors tend to get suppressed by those. This is particularly noticeable in the first column of Figure 7, where the test image in the top row used 32 samples per pixel (SPP) and the test image in the bottom row used 1,024 SPP. As expected, much of the errors were "pushed" to the left by using 1,024 SPP, but there are also more values in the buckets with larger errors (e.g., around 0.7 on the x-axis). The reason for this is that, when using naive path tracing, there is a $1024/32 = 32$ times higher risk of rendering a firefly (a pixel with very high intensity compared to its surrounding pixels) at 1,024 SPP than at 32 SPP. At the same time, there is not a sufficient number of samples to average out those firefly values. The weighted histogram in log space reveals this well.

To further compress the data, we also compute the weighted median and the arithmetic mean of all errors. The weighted median is the median of the weighted histogram and it is located where the area to the left in the weighted histogram is the same as the area to the right. This can be seen to the left in Figure 7. When a single error value is to be reported, we recommend using the weighted median due to its intuitive interpretation. An exception to this recommendation are deep learning applications, for which the mean is preferred, as it is easier to differentiate. In addition to the weighted median, we present the 25% and 75% weighted percentiles (first and third weighted quartiles), the minimum value, and the maximum value. Together, these quantities can be used to get a coarse overview of the error distribution. In Figure 8, these values are shown for a set of rendered images with increasing number of samples per pixel, where it can be observed that the

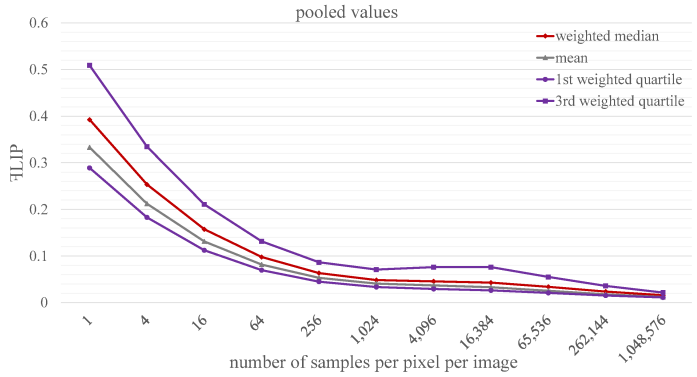

Fig. 8. Pooled ꟻLIP values for the Bistro leaves images, partially shown in Figures 7 and 10. Note that the third quartile has a local peak at 4,096 and 16,348 SPP. The reason for this is that the corresponding images have an increased number of fireflies compared to the lower SPP images, which can be seen when zooming in on the corresponding images in Figure 10.

third weighted quartile reveals an increased number of fireflies around 4,096 and 16,384 SPP. The proposed pooling method is simple and has an intuitive interpretation. It might not be the preferred option under all circumstances, however, and we leave it to the user to evaluate and choose the option best suited for their use case.

We note that ꟻLIP is differentiable and can be used as an objective function for, e.g., deep learning-based image-to-image applications. Initial tests show that the mean value of the ꟻLIP map can be successfully used as a training loss. A thorough evaluation of this is, however, left for future work.

6 EVALUATION

In this section, we evaluate ꟻLIP using two methods. In Section 6.1, we analyze the error maps generated by ꟻLIP. This is done in the form of case-by-case comparisons with other metrics for some real-world examples. In Section 6.2, we examine how ꟻLIP compares to other algorithms on a larger scale by conducting a user study. In the study, the subjects are tasked with determining how well the produced error maps correspond to their perception of the differences between alternating reference and test images. Finally, in Section 6.3, we briefly discuss possible reasons for the outcome of the evaluation.

In addition to several image pairs that we generated ourselves, we used the LocVis [Wolski et al. 2018] and the CS-IQ image data bases [Larson and Chandler 2010], as well as the data bases by Herzog et al. [2012] and Čadík et al. [2012], during the development of ꟻLIP. A large set of these images are renderings that exhibit common rendering artifacts including aliasing, Monte Carlo noise, and color shifts. Additionally, we have investigated a set of natural images suffering from various distortions such as compression and dithering artifacts, blur, and contrast changes. All in all, we have used over 1,500 image pairs for visual inspection and looked at error maps generated with Euclidean RGB distance, SMAPE [Vogels et al. 2018], SSIM [Wang et al. 2004], S-CIELAB [Johnson and Fairchild 2003], HDR-VDP-2 [Mantiuk et al. 2011], LPIPS [Zhang et al. 2018], PieAPP [Prashnani et al. 2018], the CNN visibility metric [Wolski et al. 2018] (CNN, for short), and Butteraugli [Alakuijala et al. 2017]. For visualization, some of the metrics were modified to

generate results in $[0, 1]$, as explained in the following paragraphs. The 1,500 image pairs, together with the corresponding error maps, can be accessed via our research page.[1]

SSIM originally generates values in the range $[-1, 1]$, where 1 indicates no error (perfect similarity), 0 indicates maximum error (no similarity) in either luminance or the variance of luminance, and the negative values can arise due to the correlation factor. To instead retrieve errors from 0 to 1, we re-map the SSIM error, s, by $s' = 1 - s$, yielding error $s' = 0$ when SSIM claims perfect similarity and error $s' = 1$ when it detects no similarity. In the analysis section, we use another color map for $s' \in (1, 2]$ to indicate negative values, which are uninterpretable in the full SSIM expression. For the user study, we clamp the original SSIM value, s, to $[0, 1]$.

PieAPP is designed to generate one value per image pair, indicating the magnitude of the perceived error between the images. This is done via a weighted sum over per-patch scores, where patches are of size 64×64 and separated by a given stride length. We modified PieAPP by setting the stride length to 1, which results in one patch centered around each pixel, and the score for the patch is used as the score for the pixel. Originally, the score is in an undetermined range, which we assume to be $(-\infty, \infty)$, with 0 representing no error and $\pm\infty$ representing maximum error. To map the score, a, to $[0, 1]$, we use the sigmoid function to compute the PieAPP per-pixel error, a', as

$$a' = \left| 1 - \frac{2}{1 + e^a} \right|. \tag{11}$$

For Euclidean RGB distance and S-CIELAB, we identify the maximum error possible given input in the RGB cube. In the case of Euclidean distance in RGB space, the largest error is $\sqrt{3}$. For S-CIELAB, with the original L*a*b* color metric [CIE 1978], the largest error possible was found to be 259, which is the distance between blue and green. The SMAPE value was computed per pixel. While SMAPE is most commonly used for high-dynamic range input, it is still included for completeness.

Butteraugli has an option to produce a heatmap of the errors based on its internal per-pixel score, which is in the $[0, 1]$ range, with 0 indicating minimal error, and 1 indicating maximum error. We simply change the original color map to the one we use and use the score directly. Similar approaches were used to generate error maps for the CNN visibility metric, HDR-VDP-2, and LPIPS.

6.1 Analysis

This analysis focuses on the image pairs presented in Figure 9 and Figure 10, with the purpose of highlighting strengths of FLIP versus the other metrics listed in the previous section. Both figures show a set of reference and test images together with the FLIP maps and the error maps from the other metrics. The FLIP maps were computed given $p = 67$ PPD, which is the assumed viewing condition in the evaluation.

In the first column of Figure 9, we see the `Attic_Aliasing_l1_aliasing` reference and test images together with the FLIP map and the map generated with the CNN visibility metric. As the CNN visibility metric generates the per-pixel probability that the distortion is detected while FLIP outputs error magnitudes, a comparison of the two error maps' magnitudes is not fair in this case. However, we can note the fact that the CNN visibility metric's localization, i.e., where it claims there are visible errors, is not precise. An error seen in a few pixels is spread out over multiple pixels in which it cannot be seen. Since the CNN visibility metric was trained on user markings, which were often done with a coarse marker, however, this behavior is not surprising. FLIP shows better localization in this case.

The second column shows the `Flag_Aliasing_l1_aliasing` reference and test images, the FLIP map, and the LPIPS error map. Similar to the CNN visibility metric, LPIPS shows weaknesses

[1]Link to research page: https://research.nvidia.com/publication/2020-07_FLIP.

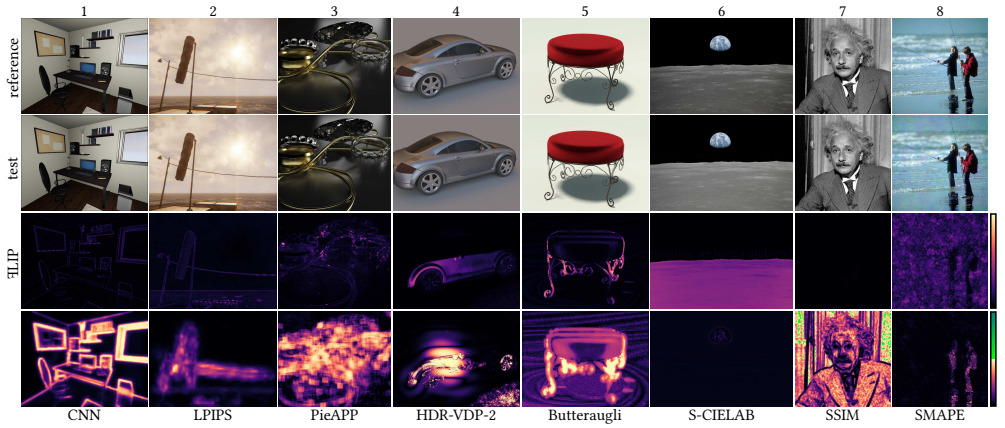

Fig. 9. A collection of example images and corresponding error maps. Each column contains one reference image, one test image, the ꟻLIP map, and the error map produced by one of the other metrics which ꟻLIP is compared against. Recall that ꟻLIP has been developed for the flip test, so instead of zooming in on the images in the paper, we recommend that the reader looks at the images in our supplemental material at a distance such that $p = 67$ pixels per degree. Here, image pairs 1–5 are rendered, while image pairs 6–8 are natural images. Image pair 6 is courtesy of NASA and image pair 7 of Lambert/Keystone/Getty Images.

in terms of localization. Again, ꟻLIP is more accurate in this regard. This can be observed, e.g., along the cable near the center of the image or at the bottom left of the windsock itself, where errors are indicated far from the pixels where they are actually seen. We also see that the ꟻLIP error map correspond better with the perceived error than the LPIPS map does, as it correctly indicates small errors whereas LPIPS indicates large errors in multiple pixels.

An example of an error map generated with PieAPP is shown in the third column. The reference and test images were rendered using different algorithms and the latter contains a large amount of noise. PieAPP detects much of the perceivable noise, but not all of it. For example, it indicates a very small error on a part of the rightmost ring, where an error is clearly visible. As seems to be the general issue with the deep learning-based metrics we compare to in this paper, PieAPP also exhibits problems with localization. Still, despite spreading errors too widely, we see that it does find most of the fireflies. ꟻLIP also finds the fireflies, but with better precision. When zooming in on the ꟻLIP map, one can notice dark borders around many of the fireflies. The borders are results of the spatial filtering, where the blurred color of a firefly at some distance from it coincides, or at least is close to, the reference color and therefore yields a small error. This is a property inherent to the algorithm and it is the expected behavior.

In the fourth column, we show the `cadik-siggraph-asia-2012-tt` reference and test images, together with the corresponding HDR-VDP-2 error. As can be seen, this metric suffers from ringing artifacts. While it does show some sense of where the errors are located, the ringing lowers the error map's correspondence to the perceived error significantly. Like the CNN visibility metric, HDR-VDP-2 produces probabilities of distortion detection and, therefore, we refrain from commenting on the error magnitudes in relation to those of the ꟻLIP map. In that map, however, we see that the errors on the car are large, which agrees with the perceived error. Furthermore, we see, in the lower right corner, that ꟻLIP reacts reasonably to the low-amplitude noise observed there.

The Butteraugli error map, shown in the fifth column, can occasionally overreact to unnoticeable differences, as the floor in this example illustrates. Similar to the mentioned metrics, Butteraugli also shows some problems with localization as an error can cover many more pixels than it can be

observed in. This can be seen, for example, around the legs of the chair. In the case of ꟻLIP, we again see better localization. Additionally, we notice how the large errors perceived on the chair's legs are of adequate magnitudes in the error map. The example reference and test images used here are the `Furniture3_l1_cgibr` images.

The last three columns show examples of S-CIELAB, SSIM, and SMAPE and we see additional positive effects of ꟻLIP. First, the frequency filter version of S-CIELAB, as described by Johnson and Fairchild [2003], shows substantial issues due to the usage of the original, band-pass version of the achromatic CSF. With its significant modulation of low frequencies, the filter can result in full-image blur. For the earth/moon image in Figure 9, this implies that, after filtering, the entire reference and test image both have almost the same gray level, resulting in small errors over the entire image. Upon close inspection, it can be seen that the error on the moon's surface and in the background is almost the same, despite the fact that the only visible error is on the moon's surface. Using the CSFs proposed in Section 4.1.1, ꟻLIP evidently avoids this problem.

Next, we consider the SSIM error map between the images of Einstein (based on a photo by Lambert/Keystone/Getty Images), which differ in their dither patterns, in the seventh column of Figure 9. In this example, it is evident that SSIM does not consider viewing distance and pixel size, as it produces large errors despite the observer not being able to see any, unless viewing the images at an unusually large number of PPD. Furthermore, it produces a considerable amount of uninterpretable values, as indicated by the green pixels in the error image. The PPD dependency of ꟻLIP helps it reduce the errors greatly, which agrees well with the perceived error at $p = 67$ PPD.

Finally, in the last column, we see an example where SMAPE was used. For this image pair (`fisher.fnoise.2`), SMAPE produces small errors on the beach, in the water, and in the background, where errors are easily observed by a viewer. On the other hand, due to SMAPE's normalizing denominator, it produces large errors in the darker regions of the people in the image, where the errors, in general, are much harder to notice. ꟻLIP solves this example better, as it generates large errors everywhere except in the dark regions of the people. This is partly due to ꟻLIP's Hunt adjustment, explained in Section 4.1.2.

In Figure 10, we showcase the different metrics' error maps during a Monte Carlo-based rendering of one part of the `bistro` scene. That is, we show error maps corresponding to test images with an increasing number of samples per pixel (SPP). In general, and as can be seen in the test images, the perceived error is reduced with an increased sample count. As noted in Section 5, however, there is also a possibility that new fireflies appear as the sample count increases. Considering the error maps, we see that each of them follows the general trend of an error decrease with larger sample counts. We also see that many of them react to the many fireflies present in, e.g., the 4,096 SPP image. For S-CIELAB, the errors seen in the test images are underestimated because they are much smaller than the maximum error in that metric. ꟻLIP improved on this in Section 4.1.3 and the benefits are evident. Similar to the previous maps shown in this section, we see that the deep learning-based metrics show problems with localization. In the higher SPP images, we see how also Butteraugli and SSIM sometimes spread errors too widely, particularly near fireflies. The ꟻLIP maps exhibit better localization in these cases. Furthermore, SSIM yields several uninterpretable values for the low SPP images and tends to generate low errors along edges, despite the levels of the perceived error being the same there as in nearby parts of the images, where significantly larger errors values are generated. HDR-VDP-2 and SMAPE show the same issues as discussed earlier, i.e., ringing in the HDR-VDP-2 maps and overestimation of errors in dark regions relative to brighter ones in SMAPE's case.

With Figures 9 and 10 and the discussion that followed, we have illustrated many of ꟻLIP's strengths. One effect that is not handled by ꟻLIP is *masking*, i.e, where image differences are present, but also hard to notice due to high amounts of irregularities or contrast shifts. Such masking can

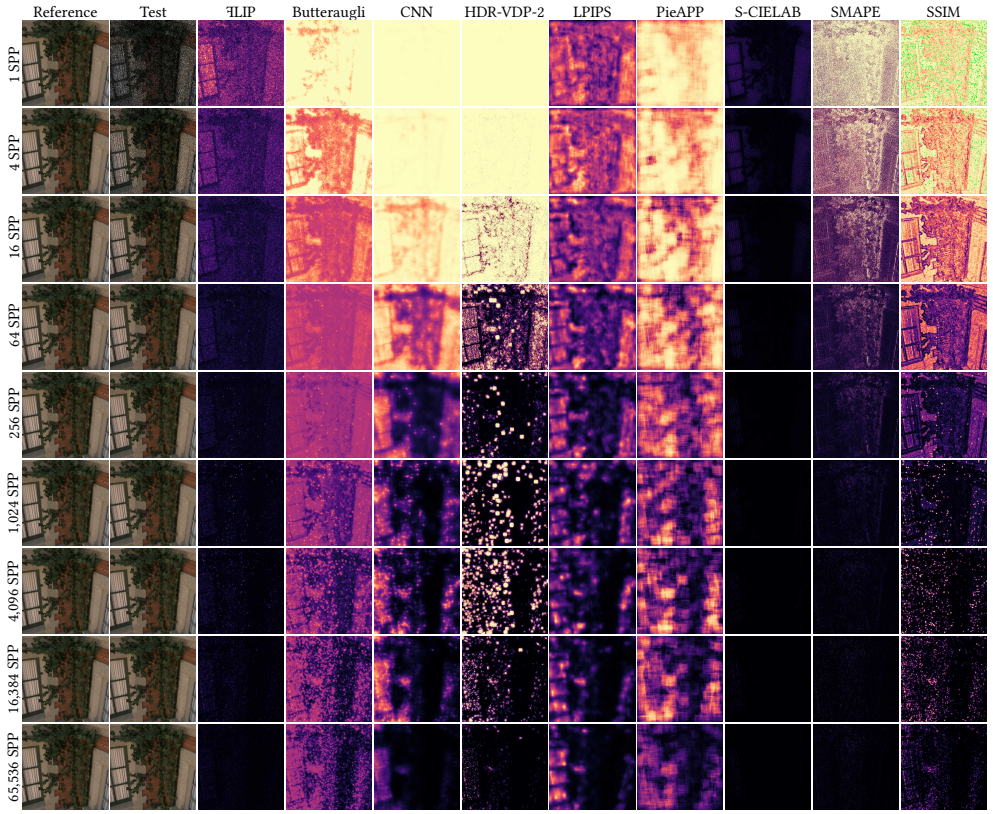

Fig. 10. Here, we show an example of the progression of a naive path tracer. In the leftmost column, we show the same reference images in each row (generated with 2^{22} samples per pixel), followed by test images, where each test image has 4× more samples than the one above it. We then present the corresponding error maps, for each test image and each metric, in the remaining columns. Note that there is a substantial amount of firefly detection with ꟻLIP in the images with 4,096 and 16,384 SPP (visible when zooming in).

lead to an overestimation of the metric output compared to the perceived errors. This can be seen in Figure 11, where we have used the image pair from `Peacock-02_l2_lici`. Since we have not been able to find any distance-dependent masking detection function, we leave this for future work.

While a thorough performance evaluation was not carried out in this work, we note that, with an unoptimized GPU implementation and at $p = 67$ PPD, ꟻLIP evaluates the difference between two 1920×1080 images in 180 ms on an NVIDIA RTX 2080 Ti.

6.2 User Study

Even though the visual inspection of error maps is useful, it is also desirable with a non-biased measure on how well different error maps correspond to the perceived error during flipping. To produce this, we devised a user study, which we describe here.

In the user study, the test subjects were presented with a series of alternating reference/test image pairs, with a period of 0.5 s, together with corresponding per-pixel error maps from ten metrics, which were ꟻLIP, Butteraugli, CNN, HDR-VDP-2, LPIPS, PieAPP, Euclidean RGB distance, S-CIELAB, SMAPE, and SSIM. All images used in the study can be found in the supplemental

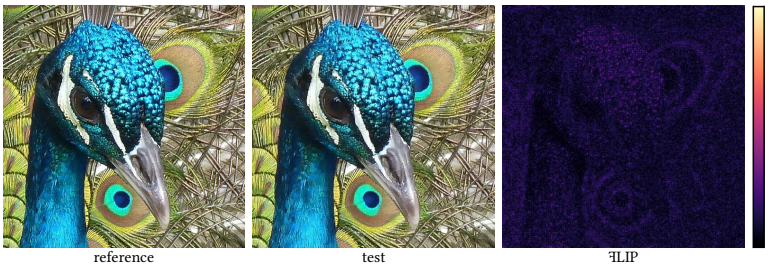

Fig. 11. A compressed photograph exemplifying ꟻLIP's tendency to overestimate the perceived error in the presence of masking.

material. The subjects were tasked to grade the error maps from *bad* to *good* based on how well they found the error maps corresponded to the differences they perceived during flipping between the reference and test images. During the study, each image metric was graded using an integer slider from 0 to 3, where 0 represented no correspondence between the error map and the perceived difference and 3 represented complete agreement. The users were also asked to pay close attention to details, i.e., if an error map showed *false positives* (indicating differences when there are none) or *false negatives* (not indicating differences when they exist). Magnitude and localization, i.e., that the metric response indicated the correct level and position of the perceived difference, were important aspects of this. The order in which the image pairs were presented was randomized. This was also the case for the order of the error maps for the different metrics. At the start of the test, each user had to enter their display width in meters and in pixels, and a script then computed a distance where the user was to position their head in order to observe $p = 67$ pixels per degree of visual angle, which was the value used when computing all error maps. To provide the user with a sense of what kind of image errors could be expected throughout the study, it started with showing the user four reference/test pairs that each exhibited different types of errors of different magnitudes.

The 21 image pairs in our user study are shown in Figure 12. The top row (11 image pairs, **R1**–**R11**) consists of rendered images, either generated in-house or extracted from different data bases [Čadík et al. 2012; Wolski et al. 2018]. To test whether ꟻLIP works for natural images with various image distortions, we included the 10 image pairs on the bottom row (**N1**–**N10**). Five of them are from the CS-IQ data base [Larson and Chandler 2010] and three are from the LocVis data base [Wolski et al. 2018], while two (earth/moon and Einstein) are distortions of our own. The earth/moon test image was generated with a luminance shift on the moon's surface and an added, dark text on the black background, while the images of Einstein include different dither patterns.

The subjects in our user study were mainly computer graphics experts, but we also included color scientists and computer vision researchers. Due to extraordinary circumstances, we were unable to set up a laboratory with a controlled environment. Instead, we shared a URL that brought users to a website where they could participate in the study. The users were asked to keep their lighting conditions from being too bright and too dim, but no more exact instructions were given in that regard. Furthermore, we did not impose any requirement on display calibration.

In total, 42 people participated in the user study. We present the metrics' average scores for each image, together with each metric's average score over all images, as a radar diagram in Figure 13. In the adjacent bar plot, we record the overall averages, the averages for the rendered images (**R1**–**R11**), and the averages for the natural images (**N1**–**N10**), together with the corresponding 95% confidence intervals. The overall averages indicate that the maps generated by ꟻLIP, which achieved an average score of 2.1, correspond significantly better to the perceived error than any of

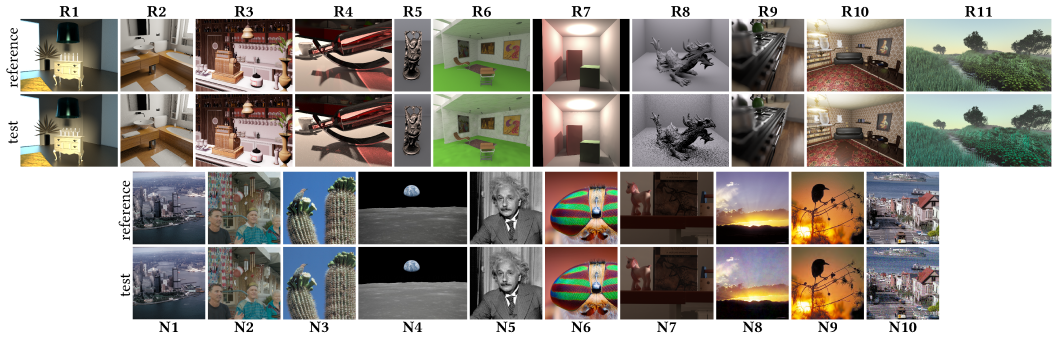

Fig. 12. The image pairs used in our user study. The top row shows rendered images, while the bottom row shows natural images with various distortions. All images, together with the error maps generated by the different metrics used in the study, are included in our supplemental material. **N4** is courtesy of NASA and **N5** is based on a photo by Lambert/Keystone/Getty Images.

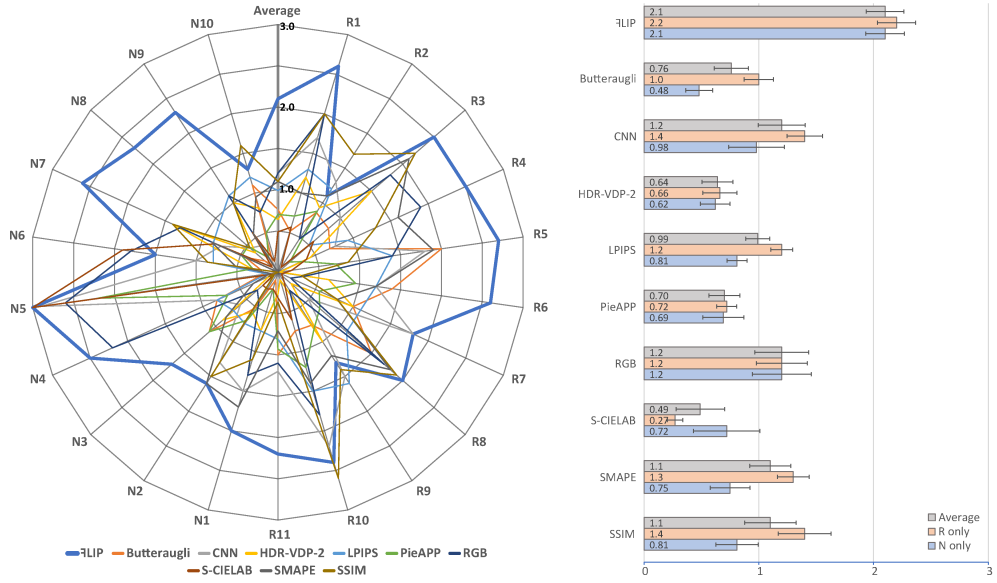

Fig. 13. Left: the results from our user study with per-image and per-metric averages. The rendered images (**Rx**) and the natural images (**Nx**) are the ones shown in Figure 12. Right: averages and 95% confidence intervals over all images (gray), only rendered images (orange), and only natural images (blue), respectively, where the x-axis is the score, from 0 (minimum) to 3 (maximum), in the user study.

the other metrics in the study. Furthermore, we note that there does not seem to be a distinct second place metric, as the CNN visibility metric, SSIM, LPIPS, SMAPE, and, surprisingly, the Euclidean RGB distance, all have scores close to 1.1, on average. Figure 13 shows that ꟻLIP scored the best, or shared best, for 16 out of the 21 image pairs. In the bar plot, we see that ꟻLIP performs well both for rendered and natural images. We also see that, in general, most metrics achieved best results on the rendered images.

ꟻLIP's worst score, on average, was given for image pair **R2**, which portrays a bathroom rendered with path tracing. This case is similar to the examples in Figure 10, where we showed that ꟻLIP

behaves well for such images when seen together with error maps from images rendered with other SPP. The result for image pair **R2** likely follows from the fact that the subjects were not shown such a range of images during the study and that the ꟻLIP values for this pair are small compared to some of the other metrics', such as SSIM, which scored the best for that pair. SSIM also outperformed ꟻLIP and the other metrics for image pair **N10**, which could be due to its error map showing more structure than most of the other ones. However, since the distortion is created by adding high-amplitude, white Gaussian noise over the entire reference image, structure in the error map is not to be expected.

For image pairs **R9** and **N6**, all metrics received low scores on average. Image pair **R9** is a difficult case in the sense that it is hard to discern exactly how the errors are manifested. Consequently, it is difficult to assess the error maps themselves, which could have caused the low scores. Viewed on a calibrated display, at $p = 67$ PPD, there is little to no visible distortion between the images in pair **N6**, which was also indicated by the ꟻLIP map, along with the Euclidean RGB and S-CIELAB error maps (see the supplemental material). Despite this, the corresponding scores are relatively low. This could stem from the relaxation of the calibrated display requirement. On an uncalibrated display, errors could be perceived as larger or smaller during flipping than on a calibrated display. In addition, the error maps themselves could indicate too large errors in these cases, as they could become brighter in general. We note that ꟻLIP scored lower than the Euclidean RGB distance and S-CIELAB for this pair, which is reasonable as the ꟻLIP values are sufficiently large to be seen in the error image, whereas the other two metrics generate seemingly black images.

6.3 Discussion

With the results in Sections 6.1 and 6.2, we have shown that ꟻLIP, in general, generates error maps that correspond better with the perceived errors between reference and test images than the ones generated by the other metrics. In this section, we summarize our views on why this is so. The most significant reason is that the competing algorithms tend to generate several false positives, either in terms of poor localization or overestimation of imperceivable errors. One general cause for this is that neither of them were designed for flipping images back and forth. However, many of the other metrics' shortcomings are prevalent also under other viewing protocols.

We see how the deep learning-based metrics, i.e., LPIPS, PieAPP, and CNN, show issues with localization. This likely stems from the fact that neither of them was trained with detailed localization as a target. Both LPIPS and PieAPP were trained to generate a single value per image pair, without any constraints on the full error map. CNN does aim for a per-pixel visibility map, but it was trained with coarse user markings, which likely leads to worse localization than if user markings were more exact. With more accurate ground truth error maps, deep learning-based metrics could potentially be more precise. However, such ground truth error maps are difficult to obtain and until they are available in large quantities, handcrafted metrics such as ꟻLIP will be competitive in terms of localization.

To achieve proper localization, care must be taken also for handcrafted metrics. We see severe ringing artifacts, and thus unsatisfactory localization, in the error maps produced by HDR-VDP-2, which likely arise due to its multi-scale approach. Even though ꟻLIP does not incorporate the concept of multi-scale, it too would show some ringing artifacts had we not done careful approximations of the contrast sensitivity functions (see Section 4.1.1). Had we adopted the original CSFs used in S-CIELAB directly, ꟻLIP would exhibit similar issues as S-CIELAB in terms of full image blur due to excessive modulation of low frequencies, as mentioned in Section 6.1. The low user study scores for S-CIELAB was likely due to it generally producing low values, and thus showing poor accuracy in terms of error magnitude, following the normalization with its maximum possible error. While this could have been improved with an error redistribution approach similar to the one described

in Section 4.1.3, the fact that its opponent color space and CSFs lead to significant color bleeding, together with the lack of a feature enhancement, imply that the S-CIELAB metric would still not perform on par with ꟻLIP.

As our work considers only low-dynamic range images, with pixel values in the $[0, 1]$ range, SMAPE will naturally tend to generate large errors in dark regions, as mentioned in Section 6.1. This could possibly be helped by performing some form of pixel value scaling before computing the SMAPE value. However, even if SMAPE would treat darker regions better, it is still a simple metric which does not consider any perceptual aspects and can therefore be expected to, for example, overestimate errors at a larger number of pixels per degree.

Nilsson and Akenine-Möller [2020] provide a thorough discussion about SSIM and the error maps it produces. Based on the shortcomings of SSIM explained there, including the generation of uninterpretable values, the poor handling of color, and its absence of any pixel per degree dependency, it is expected that ꟻLIP, which is designed to better handle these aspects, shows superior results in our evaluation.

7 CONCLUSIONS AND FUTURE WORK

By detailed visual inspection of more than 1,500 image pairs under a viewing protocol commonly adopted by graphics researchers and practitioners and by carefully lending principles from perceptual research, we have aimed to create a novel approach to image quality assessment with a particular focus on rendering.

The result is ꟻLIP, which targets rendered images and how rendering researchers often compare images manually by flipping, or alternating, between two images. With the ꟻLIP output at hand, a developer can make justified trade-offs between design alternatives. We set up a user study with over 40 test subjects, but, due to extraordinary circumstances, we were not able to use a laboratory setting for our tests. This meant that the lighting conditions and display calibrations were different between most users. Despite this, the results of our user study show that ꟻLIP performs substantially better, on average, compared to many existing image metrics, including state-of-the-art techniques. Furthermore, the results indicate that this is not only true for rendered images, but also for natural images with various distortions. To simplify further use of ꟻLIP, we will release our source code. It is our hope that ꟻLIP can be an important tool for rendering researchers.

There are several avenues for future work. The most important ones are devising a distance-dependent masking function, developing an even more reliable color space for large color differences, evaluating ꟻLIP as an objective function in deep learning applications, and augmenting the evaluator to the spatiotemporal domain.

ACKNOWLEDGMENTS

We would like to thank Jacob Munkberg, Jon Hasselgren, Jan Novák, and Matt Pharr, together with their colleagues, for sharing images with us. We also thank the creators of the scenes and models used for those in-house-generated images: Alex Telfold (jewelry scene in Figure 9), Amazon Lumberyard [2017] (the scene in Figures 7 and 10 and **R3**), Bitterli [2016], nacimus (scene in **R2**), Jay-Artist (scene in **R9**), the Stanford Computer Graphics Library (model in **R8**), and Deussen et al. [1998] (scene in **R11**). Furthermore, we would like to thank Robert Toth for his valuable feedback to our user study, Maria Sandsten for her assistance with the Fourier domain, and Jacob Munkberg for proofreading and helpful discussions. Our gratitude is also extended toward each of the 42 people who participated in our user study.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

How to write ꟻLIP

In LaTeX, you can add the following three lines to the preamble of your main `.tex` file:

```
\usepackage{mathtools}
\usepackage{xspace}
\newcommand{\FLIP}{\protect\reflectbox{F}LIP\xspace}
```

Then you can write `\FLIP` in your text, which will produce "ꟻLIP." In HTML, you can write `&#xA7FB;` to get a flipped F. The UTF-8 (hex) encoding is `0xEA 0x9F 0xBB` (`0xEA9FBB`), and the UTF-16 (hex) encoding is `0xA7FB`.

REFERENCES

Saeedeh Abasi, Mohammad Amani Tehran, and Mark D. Fairchild. 2020. Distance Metrics for Very Large Color Differences. *Color Research and Application* 45, 2 (2020), 208–223.

Jyrki Alakuijala, Robert Obryk, Ostap Stoliarchuk, Zoltan Szabadka, Lode Vandevenne, and Jan Wassenberg. 2017. Guetzli: Perceptually Guided JPEG Encoder. *CoRR* abs/1703.04421 (2017). arXiv:1703.04421 http://arxiv.org/abs/1703.04421

Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony DeRose, and Fabrice Rousselle. 2017. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Transactions on Graphics* 36, 4 (2017), 97:1–97:14.

Benedikt Bitterli. 2016. Rendering Resources. https://benedikt-bitterli.me/resources/.

Martin Čadík, Robert Herzog, Rafał Mantiuk, Karol Myszkowski, and Hans-Peter Seidel. 2012. New Measurements Reveal Weaknesses of Image Quality Metrics in Evaluating Graphics Artifacts. *ACM Transactions of Graphics* 31, 6 (2012), 147:1–147:10.

Damon M Chandler. 2013. Seven Challenges in Image Quality Assessment: Past, Present, and Future Research. *ISRN Signal Processing* 2013 (2013), 1–53.

Comission Internationale de l'Eclairage CIE. 1978. Recommendations on Uniform Color Spaces, Color-Difference Equations, Psychometric Color Terms. *Supplement No.2 to CIE publication No.15 (E.-1.3.1) 1971/(TC-1.3.)* (1978).

Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomír Měch, Matt Pharr, and Przemyslaw Prusinkiewicz. 1998. Realistic Modeling and Rendering of Plant Ecosystems. In *Proceedings of SIGGRAPH*. 275–286.

Russell L. DeValois and Karen K. DeValois. 1990. *Spatial Vision*. Oxford University Press.

Mark D Fairchild. 1996. Refinement of the RLAB Color Space. *Color Research & Application* 21, 5 (1996), 338–346.

Mark D. Fairchild. 2013. *Color Appearance Models* (3rd ed.). John Wiley & Sons.

Mark D. Fairchild and Garrett M. Johnson. 2004. The iCAM Framework for Image Appearance, Image Differences, and Image Quality. *Journal of Electronic Imaging* 13, 1 (2004), 126–138.

Thomas J Flohr, Bernd W Kolpatzik, Raja Balasubramanian, David A Carrara, Charles A Bouman, and Jan P Allebach. 1993. Model-Based Color Image Quantization. In *Human Vision, Visual Processing, and Digital Display IV*, Vol. 1913. 270–281.

Robert Herzog, Martin Čadík, Tunç O. Aydın, Kwawng In Kim, Karol Myszkowski, and Hans-Peter Seidel. 2012. NoRM: No-Reference Image Quality Metric for Realistic Image Synthesis. *Computer Graphics Forum* 31, 2 (2012), 545–554.

Min Huang, Guihua Cui, Manuel Melgosa, Manuel Sánchez-Mara nón, Changjun Li, M. Ronnier Luo, and Haoxue Liu. 2015. Power Functions Improving the Performance of Color-Difference Formulas. *Optics Express* 23, 1 (Jan 2015), 597–610.

Robert William Gainer Hunt. 1952. Light and Dark Adaptation and the Perception of Color. *Journal of the Optical Society of America* 42, 3 (1952), 190–199.

Garrett M Johnson and Mark D Fairchild. 2001. Darwinism of Color Image Difference Models. In *Color and Imaging Conference*, Vol. 2001. 108–112.

Garrett M. Johnson and Mark D. Fairchild. 2003. A Top Down Description of S-CIELAB and CIEDE2000. *Color Research & Application* 28, 6 (2003), 425–435.

E. C. Larson and D. M. Chandler. 2010. Most Apparent Distortion: Full-Reference Image Quality Assessment and the Role of Strategy. *Journal of Electronic Imaging* 19, 1 (2010).

Amazon Lumberyard. 2017. Amazon Lumberyard Bistro, Open Research Content Archive (ORCA). http://developer.nvidia.com/orca/amazon-lumberyard-bistro

Rafał Mantiuk, Kil Joong Kim, Allan G. Rempel, and Wolfgang Heidrich. 2011. HDR-VDP-2: A Calibrated Visual Metric for Visibility and Quality Predictions in All Luminance Conditions. *ACM Transactions on Graphics* 30, 4 (2011), 40:1–40:14.

William McIlhagga. 2018. Estimates of Edge Detection Filters in Human Vision. *Vision Research* 153 (2018), 30–36.

Vishal Monga, Wilson S Geisler, and Brian L Evans. 2003. Linear Color-Separable Human Visual System Models for Vector Error Diffusion Halftoning. *IEEE Signal Processing Letters* 10, 4 (2003), 93–97.

J. Anthony Movshon and Lynne Kiorpes. 1988. Analysis of the Development of Spatial Contrast Sensitivity in Monkey and Human Infants. *Journal of the Optical Society of America* 5, 12 (1988), 2166–2172.

Kathy T. Mullen. 1985. The Contrast Sensitivity of Human Colour Vision to Red-Green and Blue-Yellow Chromatic Gratings. *The Journal of Physiology* 359, 1 (1985), 381–400.

Jim Nilsson and Tomas Akenine-Möller. 2020. Understanding SSIM. arXiv:2006.13846 [eess.IV]

Charles Poynton. 2003. *Digital Video and HDTV – Algorithms and Interfaces.* Morgan Kaufmann.

Ekta Prashnani, Hong Cai, Yasamin Mostofi, and Pradeep Sen. 2018. PieAPP: Perceptual Image-Error Assessment through Pairwise Preference. *CoRR* abs/1806.02067 (2018). arXiv:1806.02067 http://arxiv.org/abs/1806.02067

Thijs Vogels, Fabrice Rousselle, Brian Mcwilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with Kernel Prediction and Asymmetric Loss Functions. *ACM Transactions of Graphics* 37, 4, Article 124 (2018).

Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image Quality Assessment: from Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.

Zhou Wang, Eero P Simoncelli, and Alan C Bovik. 2003. Multiscale Structural Similarity for Image Quality Assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers*, Vol. 2. IEEE, 1398–1402.

Krzysztof Wolski, Daniele Giunchi, Nanyang Ye, Piotr Didyk, Karol Myszkowski, Radosław Mantiuk, Hans-Peter Seidel, Anthony Steed, and Rafał K. Mantiuk. 2018. Dataset and Metrics for Predicting Local Visible Differences. *ACM Transactions on Graphics* 37, 5, Article 172 (2018).

Gunter Wyszecki and Walter Stanley Stiles. 1982. *Color Science.* Vol. 8. Wiley New York.

R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *IEEE Conference on Computer Vision and Pattern Recognition.* 586–595.

Xuemei Zhang and Brian A Wandell. 1997. A Spatial Extension of CIELAB for Digital Color-Image Reproduction. *Journal of the Society for Information Display* 5, 1 (1997), 61–63.

Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and Sung-Eui Yoon. 2015. Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering. *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports)* 34, 2 (2015), 667–681.

A PSEUDOCODE

Below, `.^` is an element-wise power function, whereas `^` is a scalar power function, and `./` represents element-wise division. The per-pixel Euclidean norm of the image `x` is denoted `||x||`.

```
FLIP(ref, test, pixelsPerDegree)
    ref = TransformToOpponentSpace(ref)
    test = TransformToOpponentSpace(test)

    #### Color pipeline ####
    # Spatial filtering
    filteredRef = clamp(ConvolveWithCSFs(ref, pixelsPerDegree))
    filteredTest = clamp(ConvolveWithCSFs(test, pixelsPerDegree))
    # Perceptually uniform color space
    preprocessedRef = HuntAdjustment(TransformToCIELAB(filteredRef))
    preprocessedTest = HuntAdjustment(TransformToCIELAB(filteredTest))
    # Color metric
    deltaEhyab = HyAB(preprocessedRef, preprocessedTest)
    cmax = (HyAB(HuntAdjustment(labGreen), HuntAdjustment(labBlue))) ^ qc
    deltaEc = RedistributeErrors(deltaEhyab .^ qc, cmax)

    #### Feature pipeline ####
    # Edge and point detection
    refY = extractAndNormalizeAchromaticComponent(ref)
    testY = extractAndNormalizeAchromaticComponent(test)
    edgesReference = ConvolveWithFirstGaussianDerivative(refY, pixelsPerDegree)
    pointsReference = ConvolveWithSecondGaussianDerivative(refY, pixelsPerDegree)
    edgesTest = ConvolveWithFirstGaussianDerivative(testY, pixelsPerDegree)
    pointsTest = ConvolveWithSecondGaussianDerivative(testY, pixelsPerDegree)
    # Feature metric
    deltaEf = (max(abs(||edgesReference|| - ||edgesTest||),
                   abs(||pointsReference|| - ||pointsTest||)) ./ sqrt(2)) .^ qf

    ### Final error ###
    deltaE = deltaEc .^ (1 - deltaEf)
    return deltaE
```