EUROGRAPHICS Workshop on Graphics and Cultural Heritage (2023)
A. Bucciero, B. Fanini, H. Graf, S. Pescarin, and S. Rizvic (Editors)

Effective interactive visualization of neural relightable images in a web-based multi-layered framework

Leonardo Righetto[1], Fabio Bettio[2], Federico Ponchio[3], Andrea Giachetti[1] Enrico Gobbetti[2]

[1]University of Verona, Italy, [2]CRS4, Italy, [3]ISTI-CNR, Italy,

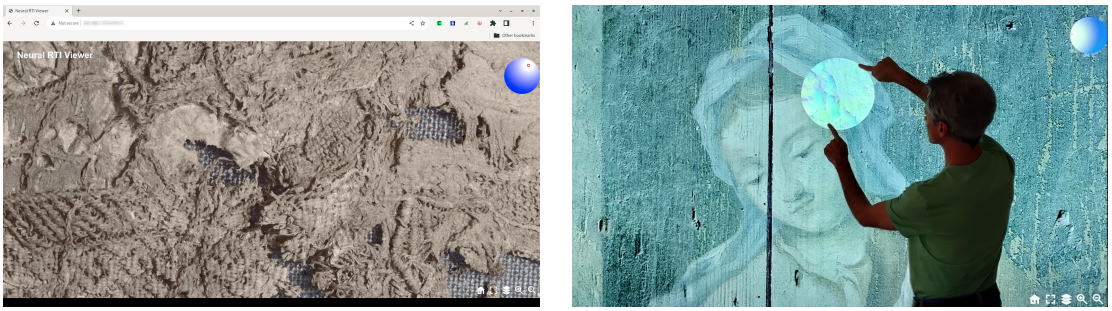

Figure 1: ***Interactive neural relighting.*** *Left: interactive exploration on a laptop with integrated graphics (full HD display, Intel Coffee Lake GT2 graphics); Right: interactive exploration on a 98-inch multi-touch display connected to a desktop PC (4K, NVIDIA RTX 2080 Ti graphics). Both applications use the same code executing in a web browser. The model on the left is derived from a high-resolution MLIC capture of textile artifacts from the Oseberg Find, coming from a Viking Age burial mound at Oseberg in South Norway. The model on the right is of an organ door with an announcing angel, 17th century, belonging to the Diocesan Museum of Vicenza and reconstructed from a handheld-light MLIC capture in collaboration with the Accademia delle Belle Arti of Verona.*

Abstract

Relightable images created from Multi-Light Image Collections (MLICs) are one of the most commonly employed models for interactive object exploration in cultural heritage. In recent years, neural representations have been shown to produce higher-quality images, at similar storage costs, with respect to the more classic analytical models such as Polynomial Texture Maps (PTM) or Hemispherical Harmonics (HSH). However, their integration in practical interactive tools has so far been limited due to the higher evaluation cost, making it difficult to employ them for interactive inspection of large images, and to the difficulty in integration cost, due to the need to incorporate deep-learning libraries in relightable renderers. In this paper, we illustrate how a state-of-the-art neural reflectance model can be directly evaluated, using common WebGL shader features, inside a multi-platform renderer. We then show how this solution can be embedded in a scalable framework capable to handle multi-layered relightable models in web settings. We finally show the performance and capabilities of the method on cultural heritage objects.

CCS Concepts

• ***Computing methodologies*** → ***Reflectance modeling; Graphics systems and interfaces;*** • ***Applied computing*** → ***Arts and humanities;***

1. Introduction

The interactive inspection of the shape and appearance of objects is of fundamental importance in many application fields. In the Cultural Heritage (CH) domain, it is routinely used by both experts (e.g., scholars and conservators) and the wider public to replace, augment, or complement the inspection of real objects [SCC*11]. In this context, in parallel to interactive viewers focusing mostly on the shape of 3D virtual replicas [PCD*15, Ske19], relighting approaches, popularized by Reflectance Transformation Imaging (RTI) viewers [CHI19], have emerged as one of the most common and effective ones.

Relighting viewers operate on a 2D view of the scene of interest, and on top of offering panning and zooming, make it possible to control a virtual light to re-illuminate the scene during an inspec-

DOI: 10.2312/gch.20231158

tion, as well as to present multiple facets of it using stratigraphic rendering techniques [JAP*21, BAMG21]. The success of this approach is due to the ease with which such scenes can be acquired using a fixed camera and a variety of setups [PDC*19], as well as to a number of practical reasons that facilitate the deployment and usage of such viewers. First of all, many cultural objects (e.g., paintings, coins, bas-reliefs, or manuscripts) have a dominant viewing direction, and many others are naturally inspected from one or a few preferential views (e.g., many statues). Second, restricting camera control to 2D panning and zooming removes the complexity of 3D camera control, which is one of the main difficulties of 3D exploration applications, reducing learning curves [JH15]. Finally, a relighting interface supports a type of visualization very appropriate to inspect fine surface and appearance details and resembles the classical physical inspection with raking light sources to reveal surface detail of actual objects under study. For this reason, the interactive inspection of relightable images has been applied to a wide range of items and has shown to be very appropriate both for expert and casual users [PDC*19].

Performing realistic interactive re-illumination based on a (physically-based) shape and material model, however, practically restricts the method to the limited number of cases in which computing such a model is feasible, and requires non-trivial runtime illumination computation at rendering time to take into account non-local lighting effects and/or transparency and translucency [PDC*19]. For this reason, the vast majority of relighting viewers employ Reflectance Transformation Imaging (RTI) methods that simply reproduce the acquired reflectance field, without any separation into shape and reflectance components. Starting from a Multi-Light Image Collection (MLIC), i.e., a set of photographs acquired with a fixed camera under varying lighting conditions, such methods strive to compactly encode acquired data using view-dependent per-pixel reflectance functions, allowing efficient transmission, storage, and generation of new images using any light direction in the hemisphere around the camera place.

A large variety of solutions have been proposed to model reflectance functions and perform fitting to input data. Classic interactive solutions to perform RTI, however, rely on low-frequency representations that fail to suitably represent the subtle illumination effects generated by the intertwining of complex local geometric and appearance characteristics [PDC*19]. In recent years, reflectance functions based on artificial neural networks have been shown to produce higher-quality images, at similar storage costs, with respect to classical analytical formulations (see Sec. 2). However, their integration into interactive tools has so far been limited, due to a number of practical reasons. First of all, current approaches still have a higher computational cost, making the per-pixel evaluation on massive models and large pixel-count displays incompatible with interactivity constraints, especially on commodity and mobile platforms. Moreover, their deployment in viewers typically requires the integration of libraries and frameworks for neural inference that require non-trivial programming efforts and complicate data communication with the rest of the graphics infrastructure (see Sec. 2). This is particularly true for current web-based viewers that must rely on standard WebGL, HTML5, and JavaScript features to ensure portability.

In this paper, we report on our work aiming at integrating complex relightable representations into modern scalable systems for inspecting stratigraphic models. After providing a short overview of the related work (Sec. 2), we illustrate how a state-of-the-art neural reflectance model based on an asymmetric auto-encoder design [DFP*20] can be efficiently integrated into a shader architecture, evaluating it directly into fragment shaders using common WebGL features (Sec. 3). Similar techniques could be used on other current low-complexity decoders. Then, we show how this solution can be embedded in a scalable framework capable to handle multi-layered relightable models in web settings. We do so by exploiting interpolation of the latent representation to construct level-of-detail (LOD) approximations of the neural representation, using it both offline, to build a discrete LOD hierarchy, and at run-time, to support fine-grained adaptive time-critical rendering through resampling (Sec. 4). Finally, we analyze the performance and capabilities of the method on the inspection of cultural heritage models (Sec. 5) and conclude with a discussion and view of future works (Sec. 6).

The solutions introduced in this paper will be released as open-source software within the *OpenLIME* framework [Ope22], a new open-source initiative focused on the web-based exploration of stratigraphic relightable models.

2. Related work

Image-based relighting techniques using specialized reflectance models, and web deployment of those solutions are vast and well-researched subjects, and complete coverage of the literature is out of the scope of this paper. We discuss here only the approaches most closely related to ours, and we refer the reader to established surveys for a wider coverage [PDC*19, TFT*20].

Relightable image models and neural representations. Image-based relighting relies on storing for each image location a description that is used in real-time to produce, while panning and zooming, the rendering of areas of the image under a novel, virtual illumination. Methods that separate shape and material information can produce flexible physically-based descriptions (e.g., normal/depth fields and BRDFs) that generate realistic images and can be integrated into standard real-time and high-quality renderers. However, they are very hard to generate from sampled data and are limited in terms of classes of objects and material behaviors [GGG*16, PAZ*23]. For this reason, the vast majority of methods approximate the reflectance field with a formulation that provides the mapping from lighting parameters to final renderable values, without explicitly separating shape and material information [PDC*19]. The seminal approach is Polynomial Texture Mapping (PTM) [MGW01], which stores per-pixel coefficients of a bi-quadratic polynomial that best fits the color variations of the pixel as a function of the incident light direction. Follow-ups increased the quality by changing the polynomial formulation [ZD14], using a Hemi-Spherical Harmonics (HSH) formulation to better fit angular data [GKPB04], or a Discrete Modal Decomposition (DMD) [PLGF*15]. The compactness and low complexity make these techniques suitable for fast interactive relighting in local and remote visualization. Without extra information, however, these methods are limited to model only low-frequency behavior [DHOMH12]. Radial Basis Function (RBF) interpolation

has been proposed as an alternative to simple parametric functions [GCD*18], but the method requires run-time access to the original massive image stack and is not suitable for interactive relighting. The approach was later combined with Principal Component Analysis (PCA) compression of the image stack and RBF interpolation in light space to improve efficiency at the cost of a slight reduction in quality [PCS18]. In recent years, neural networks have emerged as a viable technique for compression, approximation, and interpolation tasks from large amounts of data, and have been also applied to similar settings [TFT*20]. Representative examples are light-transport-matrix interpolation [RDL*15] and deep relighting [XSHR18]. These approaches, however, were not directly applied in a MLIC setting. The NeuralRTI approach [DFP*20, PB23] uses a fully connected asymmetric autoencoder to encode the original per-pixel information into a low-dimensional vector and to decode it to reconstruct pixel values from the pixel encoding and a novel light direction. The method has improved quality with respect to classic solutions but has performance limitations that make interactive relighting difficult for large models and/or screen sizes. In this work, we show how to exploit the continuity of the latent representation to produce a multiresolution structure that can be efficiently encoded in web- and GPU-friendly formats and exploited in a flexible adaptive rendering pipeline.

Integration of neural models in web viewers. Running deep-learning-powered Web applications in browsers has been the target of many recent efforts since the web environment promises to simplify cross-platform portability and serves a large variety of users [MXZ*19]. For this reason, several JavaScript-based deep-learning frameworks and libraries have been introduced in the recent past. These include *ConvNetJS* [Kar22], *WebDNN* [HKUH17], *Mind* [Mil22] and *TensorFlow.js* [STA*19]. These solutions, however, strive to offer general-purpose ways to build entire applications or embed complex networks. Interoperability with existing rendering systems is possible but requires particular care, in particular for the processing and post-processing of data. Since relighting networks are typically extremely small and simple in their structure (e.g, reduced feed-forward decoders), realizing them directly within the shader infrastructure simplifies the deployment of applications by reducing dependencies, and makes it possible to streamline the data encoding, as well as efficiently integrate pre- and post-processing within a single shader pass.

Interactive relighting tools. Many interactive exploration tools for flat but visually and geometrically rich models have been proposed and, sometimes, deployed for open, public use [PDC*19]. While some of these methods target static exploration of image data (e.g., multispectral or stratigraphic data [MAD*18] or multi-light image collections [VHW*18, Mac15]), the vast majority exploit specific compact reflectance models for supporting dynamic exploration through virtual relighting. Tools that support shape and material models, e.g. normals and BRDFs [JAP*21], make it possible to emulate realistic local lighting at negligible costs but are applicable only when such a representation exists or is reasonable to generate [AAB*22]. For this reason, the vast variety of tools exploits specific relightable image formats, such as PTM [MGW01] and HSH [GKPB04]. While early software tools were designed for desktop use and locally resident data [CHI19], recently web-based solutions have emerged as a flexible means to support multi-platform exploration. These tools typically use JavaScript, HTML5, and WebGL to interactively display the models and to tailor the exploration experience to a variety of setups and displays [PCD*15, BEJZ09]. Web-based tools for image-based relighting, e.g., *WebRTIViewer* [P*19a], *PLD-Webviewer* [KUL19], *Digital Materiality* [DHL17, FBKR17], and *Relight* [P*19b, PCS18], typically support only specific parametric formulations of relightable images (in particular, PTM and/or HSH), and use them to provide interactive relighting and some enhancement capabilities. In this work, we extend the open-source *OpenLIME* framework [Ope22] to flexibly support neural relighting in addition to PTM, HSH, Normal and BRDF, and RBF formats. In addition to showing for the first time an effective integration of neural models through simple WebGL shaders into a multiscale tiled renderer, we also introduce adaptive resampling techniques that could be employed to ensure interactivity also for other computationally-costly techniques.

3. Neural relightable representation

NeuralRTI [DFP*20] is the neural network that we employ for our relightable image representation. In this section, we provide background information on the network structure and training process and detail how it can be efficiently encoded and executed using shaders to compute per-pixel colors as a function of light direction. In the next section, we will detail how we build a tiled multiresolution format on top of the single-resolution representation described here, and how we exploit the representation for adaptive time-critical rendering during interaction.

3.1. Network structure and training

NeuralRTI [DFP*20] uses an asymmetric encoder-decoder structure. The encoder processes all the observations of a single pixel (N RGB tuples associated with the N sampled input light directions) with fully connected layers and ELU activation functions (Fig. 2 top), and produces $K \ll 3N$ latent-space features. The decoder takes latent-space features, concatenates them with a given light direction, and produces the single associated RGB value for the pixel. The network is trained end-to-end on all the pixels (or a random subset of) the original MLIC, by minimizing the mean squared loss between predicted and ground truth pixel values on the set of given light directions. As a result, the network learns, per pixel, how to produce the color associated with light directions. By suitably distributing input lights over the visible hemisphere, we can approximate continuous relighting functions.

Once the training phase is finished, it is possible to use the encoder to produce a final version of the latent features associated with the observations of each pixel, and to encode them in a map of per-pixel latent features. Afterward, the encoder is discarded, and relighted images can be computed just from the learned decoder's parameters (weights and biases of the decoder network), which are common for the entire image, and the per-pixel latent features, associating them to interactively-set light directions to produce the input of the decoder (Fig. 2 bottom).

In our implementation, we reduced the number of fully connected layers in the decoder and added one more layer in the en-

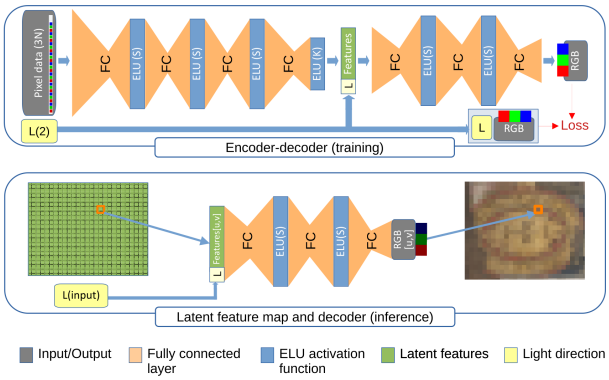

Figure 2: *NeuralRTI scheme. In our modified implementation, the encoder (top image, on the left) has 4 layers, and the decoder (top image, on the right) has 3 layers. The latent features array K is concatenated with the light directions vector $L = [L_x, L_y]$. At inference time, the encoder is replaced by a precomputed latent feature map, and only the decoder needs to be executed.*

coder, as this choice reduces the rendering complexity without a relevant impact on the quality of the results. The number of latent features and the size of the intermediate network layers are constant in this model and chosen freely in our framework. For the results in this paper, we used $K = 9$ as it has been shown that this number of features is sufficient to provide a relighting with better quality than $3^r d$-order HSH encoding using 48 per-pixel parameters [DFP*20]. Similarly to previous work, we also set the size of inner layers to the number of lights of a typical dome, e.g., $S = 50$.

In our model, the total number of decoder weights (W) and biases (B) is thus calculated using the following expressions:

$$\begin{aligned} W &= (K+2) \times S + S \times S + S \times 3 \\ B &= S + S + 3 \end{aligned} \tag{1}$$

With decoder layers of size $S = 50$ and a latent feature vector of size $K = 9$, the number of weights and related multiplications is $W = 3200$, and the number of biases and related additions is $B = 103$. $K = 9$ latent code values are, instead, stored per pixel. We currently perform feature quantization after training, to store each latent code in 8 bits. In the future, we plan to include quantization directly in the training process, to further improve accuracy.

3.2. Practical decoding and custom fragment shader

While the NeuralRTI relighting has improved quality over standard relighting methods at similar or minor storage costs, this comes at the cost of a more complex online rendering procedure. A NeuralRTI real-time decoder must, in fact, perform the following calculations for each pixel: internal product between the layer's input and weights matrix, addition with biases vector, and application of the activation function.

We implemented a custom WebGL 2 shader to perform these tasks, avoiding external library dependencies and easing the integration with a rendering subsystem. The network sequence of operations is the following: sample the latent space feature using the pixel's coordinate, combine it with a given light direction, and perform forward propagation of these values in the network until we reach the final stage with the output color, each layer computing its output as a dot product of input and weights, a sum with biases, and an application of the activation function (ELU for inner layers, identity for the final one).

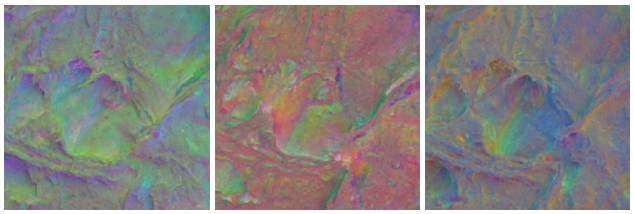

Figure 3: *The 9 per-pixel latent features are quantized and stored in three parallel RGB textures*

The latent space features are generated as a matrix of size $H \times W \times K$, where $H \times W$ is the resolution of the input images and K is the number of features saved for each pixel, that we set equal to 9 in the experiments. In order to facilitate the sampling of this representation, we encode them in a series of RGB textures. The matrix is, thus, split into sub-matrices of dimensions $H \times W \times 3$, and values quantized to 8 bits, with associated per-channel slope/intercept values for floating point conversion. We will see in Sec. 4 that we further split these textures into tiles, and build a multiresolution hierarchy, to enable adaptive rendering. Fig. 3 shows the texture encoding of the latent feature map of a 256x256 portion of the dataset explored in Fig. 1 left.

At the beginning of the shader, all the textures are sampled at the target pixel coordinates, and the result is stored in a vector of size $K + 2$, with the last two components filled with the light direction communicated to the shader in a uniform variable. This vector constitutes the input of the first network layer. Also, note that it would be possible to control per-pixel light direction through a light map.

The storage cost of the weights and biases of the decoding network, which has to be transmitted once per image, is less than 3.5K floating point values for typical network configurations (Equation 1). We have thus decided to store it in uniform arrays of 4-component vectors (`vec4`). This grouping makes it possible to fit the full network within the limits on array size for uniform arrays and, most importantly, allows us to exploit alignment and vectorized dot products and sums to compute the input to the activation function. By padding arrays of size not multiple of four with zeros, we can handle any network configuration. The same padding to a multiple of four is performed on the $K + 2$ input vector (features+light direction), leading to three *vec*4 variables when $K = 9$.

Since all the pixels compute the exact same sequence of vectorized operations to produce the output pixel, and all dot products and sums are aligned on 16-byte boundaries, there is no divergence in the shading threads.

Even though the network is customizable with different numbers of features and layer sizes, a size-specific shader is produced at compile time, transforming all network configuration parameters

to constants in the shader source. This permits to use constant-size uniform arrays and fixed-size loops that can be effectively unrolled.

Moreover, we exploit the dataflow design of *OpenLIME* shaders [Ope22] to combine NeuralRTI color computation with a sequence of post-processing operations, without the need for multipass rendering. In this design, the shading functions associated with each representation are not implemented in the `main()` function of the fragment shader, but as a shader function that produces a fragment color given their input. Filtering functions can then be appended to the shader, each one modifying the input fragment color to produce another one. The system takes care to generate a suitable `main()` function for the fragment shader, that calls all the individual shading functions in the correct sequence. We use this, in particular, to perform gamma, brightness, color-remapping, and contrast correction on neural renderings using the same tools the framework makes available to all other rendering representations (Normal+BRDF, PTM, HSH, RBF).

4. Scalable exploration of multi-layered models

Interactive exploration of large datasets on web platforms requires the usage of adaptive techniques to ensure sufficient low-latency and high-frequency feedback while coping with limitations in end-to-end bandwidth, memory available on client devices, and computation power. Multi-layered relightable models require additional resources both for the number of layers and the complexity and cost of the mathematical models used for rendering. At the core of these methods is the precomputation of levels-of-detail (Sec. 4.1), their efficient storage and transmission (Sec. 4.2), and their exploitation in adaptive rendering methods that maintain and render an adaptive working set (Sec. 4.3)

4.1. Precomputed level-of-detail structure

Scalability on large planar or 2.5D datasets is usually achieved with precomputation and storage of the source model in a pyramidal format, typically a quadtree [PG07]. To construct the pyramidal representation, the dataset is iteratively filtered and scaled by a factor of two, and each level of detail (LOD) is cut into small tiles. This allows to explore arbitrarily large datasets with limited resources. By matching the tile resolution in the working set with the screen resolution, rendering can be achieved with just one sample per pixel at all scales, and the amount of resources (the number of tiles) required to be transmitted, locally stored, and processed is determined only by the size of the screen.

This approach, originally developed for large image exploration, is routinely employed for relighting models. The construction of the various levels of detail, however, requires performing repeated filtering of higher-resolution data to construct the coarser levels, as wall as to interpolate among them to avoid restricting adaptivity to power-of-two resolutions. In order to avoid aliasing, this filtering cannot be a simple sub-sampling but has to suitably combine the values in the filter's kernel. Using simple image processing filters (e.g., averaging), on PTM and HSH models has been shown to produce reasonable results, which is justified by the linearity (with respect to the per-pixel coefficients) of the mathematical model. Performing these operations off-line on the original data prior to encoding is not a full solution, as it does not support run-time adaptation to non-power-of-two scales.

While the NeuralRTI representation is not linear, we have also empirically verified, in practice, that averaging nearby latent space features also tend to produce a pixel whose relighting behavior is similar to that of the averaged pixels (i.e., close to averaging the reflected colors at similar incident angles). As one can see in Fig. 3, the distribution of values in latent textures mimics the color distribution in the original image. Fig. 4 compares a reference zoomed image of a portion of the Oseberg Find dataset in Fig. 1 left, obtained by performing relighting and then resampling the relighted colors, with the image obtained by resampling the latent features at the output resolution and then performing relighting. The two images are very similar, as also visible in the ꟻLIP [ANA*20] difference map and ꟻLIP difference histogram and statistics. For this reason, we are directly performing resampling and filtering in latent space, using the same tools as for other representations. It should be noted that this approach has also been used for creating mipmaps for nonlinear representations, such as BRDFs (e.g, [JAP*21], that builds a pyramid over Ward parameters). While the results are already acceptable for the datasets we have used, we plan, for the future, to explicitly introduce constraints in the network to ensure the interpolability conditions at training time, as done, e.g., in generative networks for image interpolation [CXTJ19, LXL*23]

4.2. Web-friendly compression and decompression

In remote visualization, data compression is essential to cope with bandwidth constraints. In our case, the transmission cost is almost entirely due to the latent feature maps storing per-pixel data, since the network structure, weights, and biases are shared for the entire dataset and amount to a few kilobytes (see Sec. 3). For compressing the latent maps, the execution on a web platform encourages the usage of PNG, JPEG, or WebP, which are the formats natively supported in all common browsers [MDN23]. As seen in Fig. 3, the images resulting from the coefficient planes exhibit a structure similar to that of RGB photographic images, making JPEG a suitable choice for efficient compression of the image tiles that encode per-pixel latent-space features. The default parameters for JPEG compression are, however, optimized for perceptually-aware quantization. For example, the compressor handles chroma and luminance at different sampling rates and strives to take advantage of the correlation among RGB planes. However, neural coefficients do not exhibit a direct or strong correlation with these characteristics in the final image (unlike, e.g., PTM). As a result, we have chosen to disable the default conversion to the YUV color space, deactivate chroma subsampling, and utilize non-biased quantization tables.

4.3. Adaptive rendering

The computational power required for interactive neural network relighting can easily become a bottleneck depending on the hardware available, especially over the web, where a wide variety of devices needs to be supported. Therefore, we optimized our server to be able to meet time constraints. Since the computation of the final color given the latent space features and a light direction is the most costly operation, we can improve performance by controlling the number of relighted pixels. All the optimizations exploit

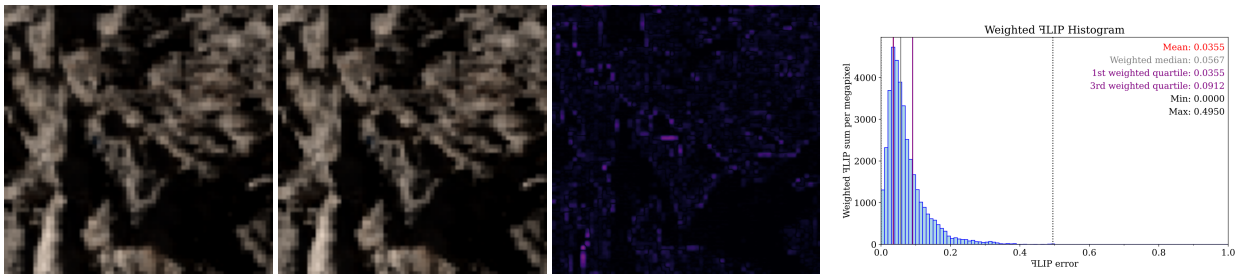

Figure 4: *From the left: reference zoomed image obtained by relighting and then resampling in color space; zoomed image obtained by resampling in latent space and relighting individual pixels; ꟻLIP difference map; ꟻLIP difference histogram and statistics.*

auxiliary memory (framebuffer) to render offline the tiles before presenting them on screen.

The first optimization exploits the decoupling between camera motion and shading under direct illumination. Since the final color is independent of camera motion, we can cache the relighted result, and reuse already available shaded tiles when the camera moves. In particular, under panning, only a few newly entering tiles per frame need to be computed.

The second optimization introduces time-critical features, in which we adaptively control the number of relighted pixels per frame. We do so indirectly, by setting a target framerate (e.g., 30fps) and deriving the relighted pixel budget from performance measures. At each frame, we then dynamically adapt the resolution of intermediate textures to the available budget by setting the virtual rendering resolution to be the minimum between the viewport resolution and the resolution dictated by the available budget. The input coefficients for each target pixel are then resampled from the closest available LOD. This LOD is the first one with a resolution higher or equal to the required resolution, if available, or the highest resolution parent if not available. Bilinear interpolation is used for sampling.

5. Implementation and results

An experimental software library supporting the methods described in this work has been implemented using JavaScript and WebGL2 and integrated within the *OpenLIME* framework [Ope22]. The basic features for neural rendering have been implemented in a specific shader class, while the features describing interactive adaptive rendering have been realized by modifying the overall rendering framework and are, therefore, available also to other rendering tools. The preprocessing features for the offline creation of image pyramids containing the neural coefficients were, instead, realized using Python and Keras for the fitting part and the creation of the full resolution representation, and *vips* [VIP22] for the conversion to the multiresolution *deepzoom* format [Mic08]. We employed a tile size of 256×256 with no overlap. Using the *tarzoom* utility provided by *OpenLime*, the directory tree containing all the tiles is then sequentially concatenated into a single file, augmented with an index that contains the start offset of each tile (and thus implicitly also its size). Having a single data file makes it possible to move the entire representation quickly among different machines and file systems, and supports very efficiently the extraction of individual tiles with simple range queries on a locally stored file (through *mmap*) or remotely (through any modern HTTP server).

5.1. Relighting quality

The improved quality of data-driven neural relighting with respect to methods using fixed bases with a similar or moderately higher number of coefficients has been assessed in previous works [DFP*20], and we do not perform an extensive analysis here. We just report that our modified version of NeuralRTI, with a reduced decoder and an improved encoder, provides results that are better than the original code. Tab. 1 summarizes the results obtained on the SyhthRTI benchmark [DFP*20]. We report results obtained with a 3rd order HSH (16 coefficients per color channel and a total of 48 bytes per pixel), the original NeuralRTI (9 bytes per pixel), and our modified architecture (9 bytes per pixel). As we can see, the proposed method has a significantly higher PSNR not only with respect to the original NeuralRTI at the same storage cost but also with respect to an HSH representation with over 5 times the storage and bandwidth requirements.

Visually, the improved quality is clearly visible in the reproduction of specular highlights and global illumination effects. An example can be seen in Fig. 5.

Dataset	HSH 3rd ord.	Neural (orig)	Ours
Texture Bytes/pixel	48	9	9
SynthRTI Single	33.33	31.00	**34.60**
SynthRTI Multi	26.46	27.39	**28.80**

Table 1: *Comparison of average PSNR obtained on the SyhthRTI benchmark [DFP*20]. We report results obtained with a 3rd-order HSH (48 coefficients), the original NeuralRTI (9 coefficients), and our modified architecture (9 coefficients).*

5.2. Rendering performance

We have extensively tested our system with a number of datasets, evaluating the relighted image quality and testing the effect of the different optimization choices on the performance of the interactive relighting on different platforms. As a representative example of typical use cases, Fig. 1 presents an exploration of two cultural

(a) Ground truth (b) 3rd-order HSH (c) Neural (d) HSH ꟻLIP difference (e) Neural ꟻLIP difference

Figure 5: *(a) Original image from a a MLIC of modern carnelian beads, Indian origin (Angkur Gems society), courtesy of Modern Repository from F. Debrabant and Captured by Mercurio Imaging; (b) Image relighted with the same light direction using a 3rd order HSH model fitted on the original MLIC. (c) Image relighted with the same light direction using our neural relighting implementation trained on the original MLIC. (d) ꟻLIP difference map comparing the HSH relighting with the ground truth. High values are clearly visible in the regions with highlights and shadows. (d) ꟻLIP difference map comparing the neural relighting with the ground truth: perceptual differences are strongly decreased.*

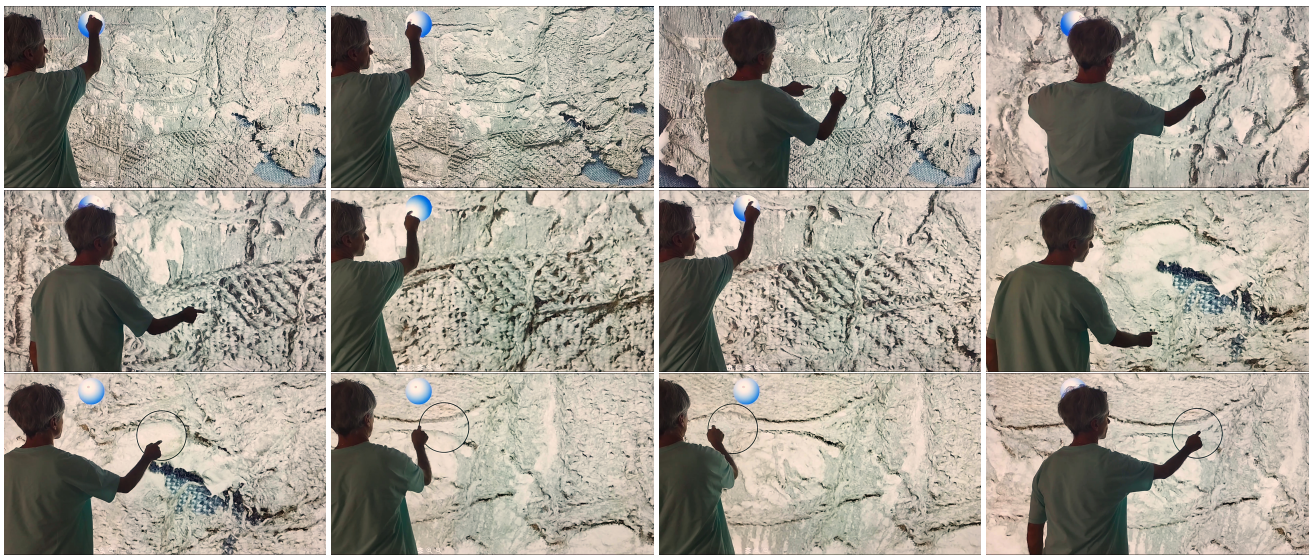

Figure 6: *Frames from one of the sequences in the accompanying video. Real-time capture of the exploration of the Oseberg Find dataset using the neural representation for the entire image and PTM inside a virtual lens. The user interacts on a large touch-screen display driven by a desktop PC.*

heritage dataset on a laptop with integrated graphics and on a museum installation driven by a desktop PC. The dataset in the left image is a High-resolution RTI capture of textile artifacts from the Oseberg Find, coming from a Viking Age burial mound at Oseberg in south Norway. Data is courtesy of Tomasz Łojewski (AGH University of Science and Technology, Kraków). The resolution of the processed images is 6240x4140. The dataset on the right image, instead, depicts an organ door with an announcing angel, 17th century, belonging to the Diocesan Museum of Vicenza and captured with handheld-light RTI in collaboration with the Accademia delle Belle Arti of Verona. The cropped and processed dataset has a resolution of 6555x3942. In terms of hardware configurations, the laptop has an Intel Coffee Lake GT2 graphics, and drives a full-HD display. The desktop PIC, instead, has an NVIDIA RTX 2080 Ti graphics board, driving a 4K 98-inch multi-touch display. Both PCs run Linux and use the Chromium Web browser.

The accompanying video illustrates the performance and quality achieved by showing interactive captures of exploration sequences. Fig. 6 shows representative frames from an inspection sequence included in the video.

In this paper, we report on the performance obtained for the exploration of the Oseberg Find dataset on the laptop PC, which is the most challenging situation given the much lower performance of the mobile integrated graphics board with respect to the gaming card used in the desktop PC.

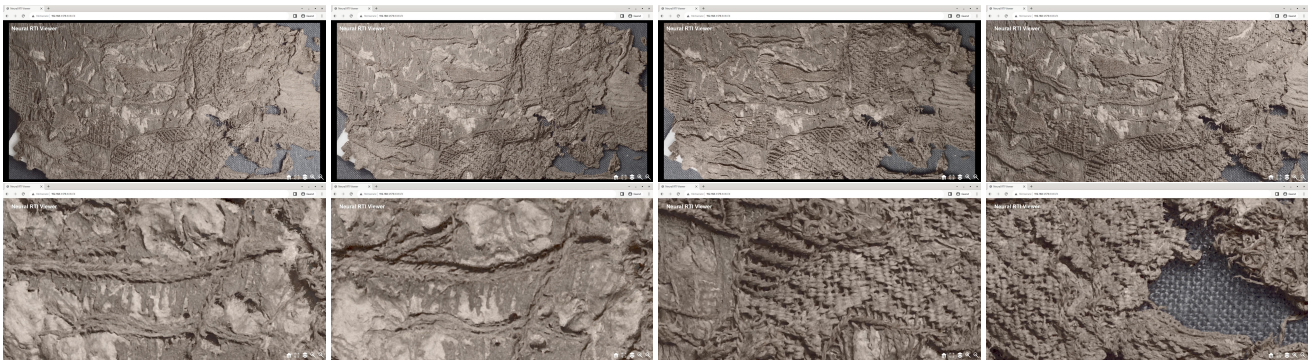

Figure 7: *Selected frames from the inspection of the Oseberg Find dataset used for benchmarking (adaptive version). The frames are ordered left to right, top to bottom. The sequence starts with the interactive motion of the light (frames 1-3), followed by a zooming sequence (frames 3=5), another light motion (frames 5-6), and a final panning (frames 7-8). The application is running on a laptop with integrated graphics.*

We recorded a short inspection sequence that involved manipulating the lighting, followed by zooming, another lighting adjustment, and finally panning the dataset. Representative frames are Fig. 7). We recorded user interaction, and repeated the sequence with and without the adaptive rendering optimization, with data resident on a local HTTP server and cleared cache, with the browser taking the entire full HD screen.

Without optimization, the renderer achieves, on average, 14.79fps and recomputes on average 393216 pixels/frame. In this benchmark, the recomputed pixel count is about 20% of the viewport size, since caching reduces the amount of required relighting during panning motion, and zooming on the dataset achieves a 2x magnification. It should be noted, also, that the renderer performs asynchronous loading. Since rendering takes some time, there are visual discontinuities when new tiles arrive and are incorporated within the view.

When adaptive optimization is enabled with a target frame rate of 30fps, the renderer achieves a performance of 35.16fps (i.e., close to the target). The higher and more constant performance is obtained by adaptively reducing the number of shaded pixels per frame during the motion to an average of 148226. Since about half of the pixels are shaded in the adaptive version, there is some slight blurring during motion with respect to the non-adaptive version, but with the advantage of a much smoother constant frame rate. When the image becomes still, moreover, the renderer progressively improves quality until the maximum attainable.

The accompanying video illustrates the behavior of the system, also when combining multiple layers. In particular, we present a use case in which the neural representation is used for the entire image, while a PTM representation is displayed inside a visualization lens. Fig. 6 shows selected frames from a recorded sequence included in the video, in which the user directly manipulates camera, light, and lens with touch interaction on a 98-inch 4K multitouch display. The example shows the possibility of using the web-based interface also for museum installations, by configuring the browser to run in kiosk mode.

Even though the inspected model is opaque and has a diffuse BRDF, in the video and the frames, it is also possible to see that the neural representations provides more details than PTM at the same storage and bandwidth cost. In particular, the shadows in the lens area appear oversmoothed, making it more difficult to perceive the patterns in the textile. Shadows in the neural relighting are more similar to ground truth data, and their improved contrast can help the experts' analysis. An even larger improvement would occur in the analysis of objects with more complex or specular appearances, such as the artefact in Fig. 5.

Since the neural representation has the same storage and bandwidth cost of PTM, and can be constructed from the exact same input data with a similar end-to-end pipeline, we expect that our approach for providing interactive performance inside a web-based platforms will boost its adoption and increase the quality of relightable models.

6. Conclusions

We have reported on our work targeting the integration of complex relightable models based on neural encoding into modern web frameworks for the inspection of stratigraphic relightable models.

By relying on the fact that typical neural reflectance field approximations need only to implement a simple decoder (typically a low-depth fully-connected network), we have shown that it is possible to directly encode weights and input data into web- and GPU-friendly formats that exploit common image formats and texturing features. By doing so, we have been able to implement the network in a way that does not significantly differ from the design of other typical RTI shaders. This solution has made it possible to avoid the incorporation of, and communication with, external deep-learning libraries, and to incorporate extra per-fragment filtering steps (e.g., gamma correction) without the need to resort to multipass solutions.

We have also empirically demonstrated that, in practice, the latent representation of nearby pixels, in the employed network, can be combined, filtered, and resampled without producing significant disturbing artifacts on the generated output image. This fact has made it possible to use filtering and resampling as major building blocks to build levels of detail and adaptive rendering

solutions. It should be noted that the assumption that some deep neural networks can model input data as flat and smooth distributions are specifically employed, e.g., in latent-space image interpolation approaches, where, if x and y are sampled from two respective domains X and Y, moving from x toward y in the latent space continuously produces realistic images from domain X to Y [CXTJ19, LXL*23]. While this property was only empirically verified for our network [DFP*20], an important avenue of research is the design of other (guaranteed) interpolable networks for reflectance function encoding. This sort of approach has already been explored for Neural BRDFs [SRRW21], but, to the best of our knowledge, not in reflectance modeling for RTI.

Furthermore, the solutions employed for ensuring interactivity through a combination of precomputed discrete levels of details, run-time adaptive resampling, smart caching, and exploitation of decoupling between lighting and camera control, are not limited to neural representations but promise to be general solutions to also efficiently incorporate other costly relighting methods into an interactive viewer. Interesting candidates are those based on RBF interpolation [GCD*18, PCS18].

The presented results on cultural heritage item inspections have shown the method's appeal and flexibility. As the method supports the same features of more standard PTM and HSH solutions, we expect it to become a possible plug-in replacement that provides a higher-quality experience with respect to current low-frequency solutions. To the benefit of the community, we plan to release its implementation as open source within the *OpenLIME* framework [Ope22].

Acknowledgments. The authors thank Tomasz Łojewski (AGH University of Science and Technology, Kraków) for the provision of the Oseberg Find data, Mercurio Imaging for capturing the modern carnelian beads, and the Diocesan Museum of Vicenza and Accademia delle Belle Arti of Verona for giving access to their artworks for the purpose of digitization. This study was partially partially carried out within the PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS00000043). We also thank Fabio Marton for collaboration in the development of OpenLime tools. Authors EG and GB acknowledge the contribution of Sardinian Regional Authorities under project XDATA.

References

[AAB*22] Ahsan M., Altea G., Bettio F., Callieri M., Camarda A., Cignoni P., Gobbetti E., Ledda P., Lutzu A., Marton F., Mignemi G., Ponchio F.: Ebb & flow: Uncovering Costantino Nivola's Olivetti sandcast through 3D fabrication and virtual exploration. In *Proc. GCH* (September 2022), pp. 85–94. doi:10.2312/gch.20221230. 3

[ANA*20] Andersson P., Nilsson J., Akenine-Möller T., Oskarsson M., Åström K., Fairchild M. D.: ꟻLIP: A Difference Evaluator for Alternating Images. *ACM Trans. Graph. 3*, 2 (2020), 15:1–15:23. doi:10.1145/3406183. 5

[BAMG21] Bettio F., Ahsan M., Marton F., Gobbetti E.: A novel approach for exploring annotated data with interactive lenses. *Computer Graphics Forum 40*, 3 (2021), 387–398. doi:10.1111/cgf.14315. 2

[BEJZ09] Behr J., Eschler P., Jung Y., Zöllner M.: X3DOM: a DOM-based HTML5/X3D integration model. In *Proceedings of the 14th international conference on 3D web technology* (2009), pp. 127–135. doi:10.1145/1559764.1559784. 3

[CHI19] CHI: Cultural heritage imaging website, 2019. [Online; accessed 22-May-2023]. URL: http://culturalheritageimaging.org. 1, 3

[CXTJ19] Chen Y., Xu X., Tian Z., Jia J.: Homomorphic latent space interpolation for unpaired image-to-image translation. In *Proc. CVPR* (2019), pp. 2403–2411. doi:10.1109/CVPR.2019.00251. 5, 9

[DFP*20] Dulecha T. G., Fanni F. A., Ponchio F., Pellacini F., Giachetti A.: Neural reflectance transformation imaging. *The Visual Computer 36* (2020), 2161–2174. doi:10.1007/s00371-020-01910-9. 2, 3, 4, 6, 9

[DHL17] DHLAB: RTI tools at DHLAB Basel, 2017. [Online; accessed 22-May-2023]. URL: https://github.com/dhlab-basel/rti.js. 3

[DHOMH12] Drew M. S., Hel-Or Y., Malzbender T., Hajari N.: Robust estimation of surface properties and interpolation of shadow/specularity components. *Image and Vision Computing 30*, 4-5 (2012), 317–331. doi:10.1016/j.imavis.2012.02.012. 2

[FBKR17] Fornaro P., Bianco A., Kaiser A., Rosenthaler L.: Enhanced RTI for gloss reproduction. *Electronic Imaging 2017*, 8 (2017), 66–72. doi:10.2352/ISSN.2470-1173.2017.8.MAAP-284. 3

[GCD*18] Giachetti A., Ciortan I., Daffara C., Marchioro G., Pintus R., Gobbetti E.: A novel framework for highlight reflectance transformation imaging. *Computer Vision and Image Understanding 168* (2018), 118–131. doi:10.1016/j.cviu.2017.05.014. 3, 9

[GGG*16] Guarnera D., Guarnera G. C., Ghosh A., Denk C., Glencross M.: BRDF representation and acquisition. *Computer Graphics Forum 35*, 2 (2016), 625–650. doi:10.1111/cgf.12867. 2

[GKPB04] Gautron P., Křivánek J., Pattanaik S. N., Bouatouch K.: A novel hemispherical basis for accurate and efficient rendering. *Rendering Techniques 2004* (2004), 321–330. doi:10.2312/EGWR/EGSR04/321-330. 2, 3

[HKUH17] Hidaka M., Kikura Y., Ushiku Y., Harada T.: WebDNN: Fastest DNN execution framework on web browser. In *Proc. ACM Multimedia* (2017), pp. 1213–1216. doi:10.1145/3123266.3129394. 3

[JAP*21] Jaspe Villanueva A., Ahsan M., Pintus R., Giachetti A., Gobbetti E.: Web-based exploration of annotated multi-layered relightable image models. *ACM JOCCH 14*, 2 (2021), 24:1–24:31. doi:10.1145/3430846. 2, 3, 5

[JH15] Jankowski J., Hachet M.: Advances in interaction with 3d environments. *Comput. Graph. Forum 34*, 1 (2015), 152–190. doi:10.1111/cgf.12466. 2

[Kar22] Karpathy A.: ConvNetJS: Deep learning in your browser, 2022. [Online; accessed 19-May-2023]. URL: https://cs.stanford.edu/people/karpathy/convnetjs/. 3

[KUL19] KUL: PLD software KU-Leuven, 2019. [Online; accessed 22-May-2023]. URL: https://portablelightdome.wordpress.com/software. 3

[LXL*23] Liu Y., Xiong Z., Li Y., Tian X., Zha Z.-J.: Domain generalization via encoding and resampling in a unified latent space. *IEEE Transactions on Multimedia 25* (2023), 126–139. doi:10.1109/TMM.2021.3121564. 5, 9

[Mac15] Macdonald L. W.: *Realistic visualisation of cultural heritage objects*. PhD thesis, UCL (University College London), 2015. 3

[MAD*18] Moutafidou A., Adamopoulos G., Drosou A., Tzovaras D., Fudos I.: Multiple material layer visualization for cultural heritage artifacts. In *Proc. GCH* (2018), pp. 155–159. doi:10.2312/gch.20181353. 3

[MDN23] MDN M.: Image file type and format guide, 2023. [Online; accessed 28-May-2023]. URL: https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Image_types. 5

[MGW01] MALZBENDER T., GELB D., WOLTERS H.: Polynomial texture maps. In *Proc. SIGGRAPH* (2001), pp. 519–528. doi:10.1145/383259.383320. 2, 3

[Mic08] MICROSOFT: DeepZooom, 2008. [Online; accessed 22-May-2023]. URL: http://www.seadragon.com/developer/creating-content/file-formats/. 6

[Mil22] MILLER S.: MIND: Deep learning in your browser, 2022. [Online; accessed 19-May-2023]. URL: https://github.com/stevenmiller888/mind. 3

[MXZ*19] MA Y., XIANG D., ZHENG S., TIAN D., LIU X.: Moving deep learning into web browser: How far can we go? In *Proc. WWW* (2019), pp. 1234–1244. doi:10.1145/3308558.3313639. 3

[Ope22] OPENLIME TEAM: OpenLime: Open Layered IMage Explorer, 2022. URL: https://github.com/cnr-isti-vclab/openlime and https://github.com/crs4/openlime [Online; accessed 22-May-2023]. 2, 3, 5, 6, 9

[P*19a] PALMA G., ET AL.: Webrti viewer, 2019. [Online; accessed 22-May-2023]. URL: http://vcg.isti.cnr.it/rti/webviewer.php. 3

[P*19b] PONCHIO F., ET AL.: Relight, 2019. [Online; accessed 22-May-2023]. URL: http://vcg.isti.cnr.it/relight/. 3

[PAZ*23] PINTUS R., AHSAN M., ZORCOLO A., BETTIO F., MARTON F., GOBBETTI E.: Exploiting local shape and material similarity for effective sv-brdf reconstruction from sparse multi-light image collections. *ACM JOCCH* (2023). doi:10.1145/3593428. 2

[PB23] PISTELLATO M., BERGAMASCO F.: On-the-go reflectance transformation imaging with ordinary smartphones. In *Proc. ECCV Workshops, Part I* (2023), pp. 251–267. doi:10.1007/978-3-031-25056-9_17. 3

[PCD*15] POTENZIANI M., CALLIERI M., DELLEPIANE M., CORSINI M., PONCHIO F., SCOPIGNO R.: 3DHOP: 3D heritage online presenter. *Computers & Graphics 52* (2015), 129–141. doi:10.1016/j.cag.2015.07.001. 1, 3

[PCS18] PONCHIO F., CORSINI M., SCOPIGNO R.: A compact representation of relightable images for the web. In *Proc. ACM Web3D* (2018), pp. 1:1–1:10. doi:10.1145/3208806.3208820. 3, 9

[PDC*19] PINTUS R., DULACHE T., CIORTAN I., GOBBETTI E., GIACHETTI A.: State-of-the-art in multi-light image collections for surface visualization and analysis. *Computer Graphics Forum 38*, 3 (2019), 909–934. doi:10.1111/cgf.13732. 2, 3

[PG07] PAJAROLA R., GOBBETTI E.: Survey on semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer 23*, 8 (2007), 583–605. doi:10.1007/s00371-007-0163-2. 5

[PLGF*15] PITARD G., LE GOÏC G., FAVRELIÈRE H., SAMPER S., DESAGE S.-F., PILLET M.: Discrete modal decomposition for surface appearance modelling and rendering. In *Optical Measurement Systems for Industrial Inspection IX* (2015), vol. 9525, SPIE, pp. 489–498. doi:10.1117/12.2184840. 2

[RDL*15] REN P., DONG Y., LIN S., TONG X., GUO B.: Image based relighting using neural networks. *ACM TOG 34*, 4 (2015), 111:1–111:12. doi:10.1145/2766899. 3

[SCC*11] SCOPIGNO R., CALLIERI M., CIGNONI P., CORSINI M., DELLEPIANE M., PONCHIO F., RANZUGLIA G.: 3D models for cultural heritage: Beyond plain visualization. *Computer 44*, 7 (2011), 48–55. doi:10.1109/MC.2011.196. 1

[Ske19] SKETCHFAB: Sketchfab - publish and find 3d models online, 2019. [Online; accessed 22-May-2023]. URL: https://sketchfab.com/. 1

[SRRW21] SZTRAJMAN A., RAINER G., RITSCHEL T., WEYRICH T.: Neural BRDF representation and importance sampling. *Computer Graphics Forum 40*, 6 (2021), 332–346. doi:10.1111/cgf.14335. 9

[STA*19] SMILKOV D., THORAT N., ASSOGBA Y., NICHOLSON C., KREEGER N., YU P., CAI S., NIELSEN E., SOEGEL D., BILESCHI S., ET AL.: Tensorflow.js: Machine learning for the web and beyond. *Proc. Machine Learning and Systems 1* (2019), 309–321. 3

[TFT*20] TEWARI A., FRIED O., THIES J., SITZMANN V., LOMBARDI S., SUNKAVALLI K., MARTIN-BRUALLA R., SIMON T., SARAGIH J., NIESSNER M., ET AL.: State of the art on neural rendering. *Computer Graphics Forum 39*, 2 (2020), 701–727. doi:10.1111/cgf.14022. 2, 3

[VHW*18] VANDERMEULEN B., HAMEEUW H., WATTEEUW L., VAN GOOL L., PROESMANS M.: Bridging multi-light & multi-spectral images to study, preserve and disseminate archival documents. In *Proc. Archiving Conference* (2018), vol. 2018, pp. 64–69. doi:10.2352/issn.2168-3204.2018.1.0.15. 3

[VIP22] VIPS: libvips:a fast image processing library with low memory needs, 2022. [Online; accessed 09-May-2023]. URL: https://www.libvips.org/. 6

[XSHR18] XU Z., SUNKAVALLI K., HADAP S., RAMAMOORTHI R.: Deep image-based relighting from optimal sparse samples. *ACM TOG 37*, 4 (2018), 126:1–126:13. doi:10.1145/3197517.3201313. 3

[ZD14] ZHANG M., DREW M. S.: Efficient robust image interpolation and surface properties using polynomial texture mapping. *EURASIP Journal on Image and Video Processing 2014*, 1 (2014), 25. doi:10.1186/1687-5281-2014-25. 2