DIGITAL HERITAGE (2025)
S. Campana, D. Ferdani, H. Graf, G. Guidi, Z. Hegarty, S. Pescarin, and F. Remondino (Editors)

OpenLIME: An open and flexible web framework for creating and exploring complex multi-layered relightable image models

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

[1]ISTI-CNR, Italy, [2]CRS4, Italy, [3]UNIVR, Italy

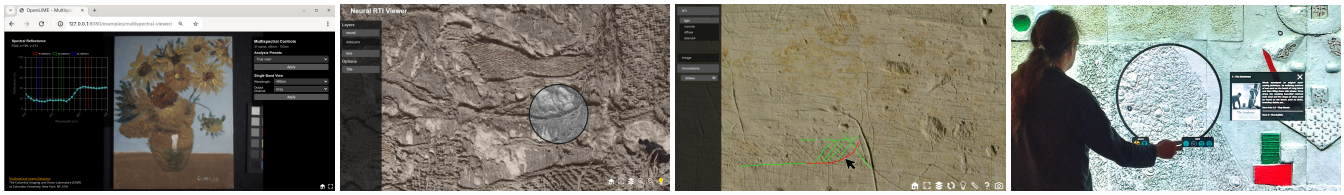


Figure 1: OpenLIME *application examples.* *From left to right: a simple 31-band multispectral image viewer; neural reflectance transformation image viewer; annotation of RTI data for archaeology; interactive audio-visual exploration of an annotated artwork in a standalone multitouch museum setup.*

Abstract

We introduce OpenLIME *(*Open Layered IMage Explorer*), an open, scalable, and flexible framework for creating web-based interactive tools to annotate and inspect large multi-layered and multi-channel standard and relightable image models. Adaptive image management and display use a data-flow approach, where images from sources of any size are efficiently streamed into screen-sized buffers that can be processed and combined using customizable WebGL shaders. The framework natively supports multispectral images, Bidirectional Reflectance Distribution Function (BRDF), and Reflectance Transformation Imaging (RTI) datasets and can be extended to accommodate other multi-channel raster datasets, such as neural representations. Multi-layer and multi-faceted visualizations are achieved through opacity adjustments, blending modes, and interactive lenses. The released library provides a set of pre-configured layers, facilitating the rapid deployment of web-based datasets and kiosk applications. Its responsive user interface is compatible with desktop, mobile, and general multitouch environments, while its modular architecture allows for extensive customization, making it adaptable to diverse annotation and visualization needs. The paper illustrates the framework's design and discusses specific use cases, including the inspection of RTI models, the integration of novel relightable image formats, archaeological data documentation and annotation, and standalone museum application creation and deployment. The main components of the framework are released as open source.*

CCS Concepts

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

1. Introduction

The virtual inspection of annotated digital representations of objects or scenes, derived from simulations or captures, plays a crucial role in various fields, including Cultural Heritage (CH) studies and valorization [TLPW24, FDAB*24]. Visualizing geometric features and material properties supports scholars in their research and contributes to developing conservation and preservation strategies [PPY*16, PDC*19]. Additionally, it enhances the presentation of cultural artifacts, either supplementing or replacing physical inspection for experts and the general public [MHJ22]. Annotations, which highlight and describe areas of interest within a dataset, can result from automated analysis or manual markup and serve as essential tools for documenting and exchanging information, as they facilitate data interpretation and improve viewer comprehension [VETL18, CCDL*20, PCDS20, AMP*23].

In these contexts, image-based models and annotations often complement or replace full-3D model generation and analysis [AMP*23]. Many cultural artifacts, including, but not limited to, paintings and bas-reliefs, are mostly flat or have a preferential viewing direction. Their full-3D modeling is more complex and resource-intensive, and does not offer significant advantages compared to its 2(.5)D counterparts [PDC*19]. Additionally, high-resolution image acquisition and management are simpler than 3D model processing, as image data can be created without the complexities of 3D view planning, occlusion handling, or multi-view fusion, and handled using standard raster structures. Such simplified setups include

multi-spectral photography, multi-light image collection (MLIC) acquisition, or a combination of both [PDC*19, KF22]. Annotation creation and exploration are also more straightforward in 2D models due to their inherent planar parameterization that simplifies markup and guidance towards annotated areas [PCDS20, BAMG21]. Furthermore, compared to fully 3D viewers, image-based exploration simplifies navigation by limiting camera motion to panning and zooming, reducing learning curves, and improving accessibility [JH15]. For annotation creation, in particular, 2D to 3D mappings let users work on multiple high-res images of objects, where mutual registration is only used to define relative positions of views and projected annotations [AMP*23].

In this work, we broaden the range and expand the capabilities of current image-based solutions (see Sec. 2) by introducing and releasing to the public *OpenLIME* (*Open Layered IMage Explorer*), a modular, scalable, and flexible web-based framework for annotated multi-channel standard and relightable raster data visualization [Ope25]. Its web architecture supports cloud-based distribution and offline kiosk applications running directly from local storage without OS dependencies and required custom server-side components. The design's core is an adaptive multi-channel raster data management system based on a data-flow approach, efficiently streaming and visualizing large datasets into screen-sized buffers. Arbitrary multi-channel data formats are decomposed into co-registered image sets with 1–4 channels stored in web-supported formats (PNG, JPG, WebP). This enables efficient multi-resolution data creation, compression, transfer, and streaming. Native web platform implementations handle decoding and uploading to the GPU, ensuring minimal processing overhead [MDN23]. Client-side, these buffers are processed and combined using customizable WebGL shaders that interpret the data, implement the visualization, and handle the combination of multiple layers. Vector graphics layers managed with SVG are employed for interface decorations and annotation display. The framework includes a set of pre-configured layers for natively supporting multispectral data, relightable imaging through Bidirectional Reflectance Distribution Function (BRDF) and Reflectance Transformation Imaging (RTI) datasets, visual annotations through vector drawing, and multi-faceted visualization through layer blending or interactive visualization lenses [BAMG21]. Additionally, it can be extended to accommodate other multi-channel raster datasets, such as hyperspectral imaging, neural rendering, and alternative reflectance models, as well as other interaction and navigation modes, such as audio-visual annotation-guided visualization tours [AMPG22]. The main framework components are released as open source software [Ope25].

This paper, after briefly analyzing related work (Sec. 2), presents the main concepts underpinning the design of the *OpenLIME* framework (Sec. 3) and explores its application in a selection of common CH use cases (Sec. 4), including Reflectance Transformation Imaging (RTI) model inspection, integration of novel relightable image formats, documentation and annotation of archaeological data, and the creation of standalone museum applications. We conclude with a summary of results and an outline of future directions (Sec. 5).

2. Related work

A full literature review is beyond this paper's scope. We discuss here only closely related approaches, referring to surveys on relighting [PDC*19], annotations [PCDS20], and artifact fruition [FDAB*24] for broader coverage.

Various tools for creating and inspecting image-based data in CH applications have been proposed in the recent past, ranging from static multi-spectral and stratigraphic visualization (e.g., [MAD*18, P*23]) to dynamic inspection with camera motion and relighting [PDC*19]. Initially desktop-focused, recent advancements have introduced web-based solutions for remote access and cross-platform compatibility. Among desktop applications, *RTI Viewer* [P*10, PCC*10] remains widely used in CH due to its integration with RTI Builder for creating relightable models and its support for Polynomial Texture Mapping (PTM) [MGW01] and Hemi-Spherical Harmonics (HSH) [GKPB04] reflectance models. It offers photorealistic relighting and non-photorealistic enhancements, such as diffuse gain and specular enhancement. Other notable tools include *APTool*, which provides Radial Basis Function (RBF) interpolation for continuous data visualization [PCS18], and *PLD-Viewer* [KUL19], which supports multi-spectral relighting with a range of visualization techniques. These tools were designed mostly for the single purpose of data exploration, and lack the flexibility of frameworks supporting annotations, data distribution, and adaptability to different kinds of applications. Web-based solutions such as *WebRTIViewer* [P*15], *DMViewer* [DHL17, FBKR17], *Relight* [P*19], and *Pixel+* [VPH*20] primarily focus on adaptive image streaming and interactive relighting from compressed formats, such as PTM and HSH. For improved quality of high-frequency models at comparable compression rates, *Relight* also supports Discrete Modal Decomposition (DMD) [PLGF*15] and PCA-compressed Radial-basis-function (RBF) or Bilinear interpolation of the source MLIC [PCS18]. *Marlie* [JAP*21], while limited to single-resolution image data, also supports normal and BRDF maps. While these tools offer several degrees of flexibility, e.g., embedding them into web pages and modifying the visual style through CSS, extension to other image data types or user interaction methods also requires considerable effort. Support for annotation creation and exploration is also limited. In our framework, we combine raster layers handled through WebGL with vector layers handled with Scalable Vector Graphics (SVG) for annotation and visualization, to achieve extensibility and broad compatibility. Moreover, in contrast to other solutions, especially for stratigraphic/multispectral imaging (e.g., [P*23]), we emphasize client-side processing rather than server-side computation, and we can run our application using regular web servers without any add-ons. This approach simplifies deployment in a variety of settings, including local-only inspection. Most existing image viewers rely on simple blending techniques to present multiple data layers. *Marlie* [JAP*21] has extended this functionality by incorporating interactive lenses, a method that enables dynamic, user-controlled visualization of multiple representations within the same dataset, enhancing multi-faceted data exploration. The system also supports annotations in the form of visual overlays and exploits lenses to reduce clutter when presenting annotated data [JAP*21]. Their approach, however, is limited to single-resolution images and annotations without overlap, while we also support overlapping annotations on multiresolution data.

3. Framework description

OpenLIME has a modular design that makes it possible to mix and match components with little or no external dependencies to create different applications. Base back-office components deal with format conversion and multiresolution decomposition of multi-channel raster data. Server-side, standard web formats and services are used for adaptively managing and transmitting data to clients. Client-side components neatly separate user interface, data loading, and visualization to create responsive applications. In the following, we first present the basic rationale, concepts, and structure of *OpenLIME*'s architecture (3.1). Then, we detail the handling of large multi-channel raster data (3.2) and of vector data used for decoration or overlay annotations (3.3). We finally provide an overview of event handling and user interface components (3.4).

3.1. Architecture overview

OpenLIME's core is designed as a fully client-side, web-based framework for interactive visualization of complex raster datasets. The primary goal is to enable easy deployment without requiring a dedicated server-side application, relying instead on a standard HTTP file server or asset delivery. All exploration-related data processing and rendering operations are performed directly in the client browser, leveraging WebGL for GPU-accelerated visualization.

To ensure broad compatibility and efficient data streaming, *OpenLIME* supports, in addition to web-native image formats (e.g., JPG, PNG, WebP) [MDN23], established multiresolution image formats such as *deepzoom* [Mic08], *OpenStreetmap/GoogleMaps* tiles [Ope24], and *IIIF* [A*24]. These formats are widely used for handling large-scale imagery on the web and follow a similar approach of pre-generated tiled pyramids, which *OpenLIME* combines to support multichannel data and efficiently loads and renders on demand (Sec. 3.2).

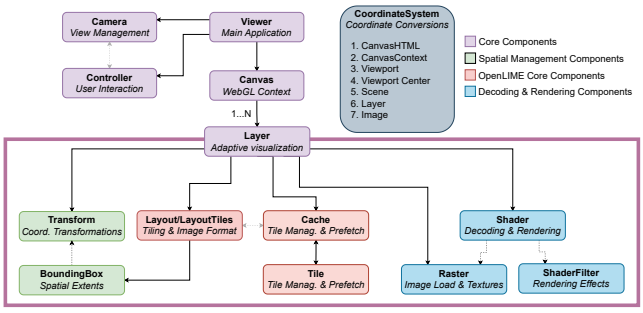


Figure 2: OpenLIME *architecture*. *The main components are depicted in the diagram. Continuous lines indicate an ownership relation, while dotted lines indicate a usage relation.*

The client architecture, see Fig. 2, is centered around a `Canvas` class, which manages a WebGL rendering surface and maintains a collection of `Layers` representing different visualization elements. The `Camera` class defines the current view, applying transformations through the Transform class to navigate the scene. *OpenLIME* supports multiple coordinate systems, including canvas, layer, window, and OpenGL coordinates, ensuring flexibility in mapping spatial data across different contexts.

Each `Layer` corresponds to a specific type of visual representation, such as images, RTI datasets, or BRDF models. Rendering is achieved through `Shader` programs, which process multichannel raster data provided by the `Raster` class. This modular approach allows each visualization type to handle its dataset structure and rendering logic while maintaining a consistent interface.

A key design goal of *OpenLIME* is to facilitate adding new visualization types with minimal effort. Developers only need to define a new `Layer` subclass, specifying how data is structured and parameters are handled, along with an associated `Shader` class to implement the rendering logic. This approach ensures extensibility and adaptability for diverse visualization needs. To facilitate users and developers, we provide a set of pre-configured layers for supporting the most common use cases in CH, including plain RGB images, various forms of RTI (PTM, HSH, and RBF), and normal map + Ward BRDF representations.

OpenLIME enables advanced compositing of layers through framebuffer operations, allowing multiple shaders to be combined efficiently. The `Canvas` class is responsible for composing the final image on screen through a multipass rendering technique: each `Layer` uses shaders that output in linear RGB and are drawn in their assigned order to an off-screen framebuffer. Layers can be merged seamlessly or displayed selectively using visualization lenses. In the final step, the `Canvas` applies a shader that converts the composed image from linear to sRGB space before displaying it. This approach ensures that the entire rendering pipeline operates in linear RGB, allowing any Layer to directly sample the contents of another Layer's framebuffer without requiring any color space conversion, thus maintaining the correctness and fidelity of intermediate results. This framebuffer-based approach simplifies shader composition, avoiding significant performance overhead while enabling real-time blending and interactive effects (Sec. 3.2).

The visualization of multi-layered raster datasets is augmented with active decoration and annotation elements, see Sec. 3.3, primarily implemented using vector graphics. *OpenLIME* adopts SVG for annotations due to its lightweight nature, scalability, and compatibility with existing web standards. This decision aligns with the Web Annotation Data Model, a W3C recommendation that is the foundation for IIIF's Image API Selector extension [A*24]. Using DOM-based SVG elements for annotations, *OpenLIME* simplifies user interaction handling. Click, hover, and other event-driven behaviors can be managed directly using standard event listeners, making implementing intuitive, interactive features easier without requiring complex WebGL-based input processing.

The *OpenLIME* graphical user interface is built with standard DOM elements, separate from the WebGL renderer, allowing flexible, customizable, application-specific interface design. Specialized classes handle pointer interactions, mapping user input to layer parameters and view changes (Sec. 3.4). This separation lets developers customize interfaces without altering the core visualization logic. The default *OpenLIME* GUI provides a robust and intuitive control system that can be quickly assembled by instantiating predefined classes, but custom interfaces can be implemented as needed without modifying the core rendering logic.

3.2. Scalable access, rendering, and composition of massive multilayered multi-field raster data

OpenLIME is designed to simplify the handling of large multi-field raster datasets and their combination into multi-layered visualizations. We assume that each layer's raster data is represented by an arbitrary number of scalar values per pixel, transformed to colors, and fused with other layer colors at presentation time based on the current viewing, lighting, and user-defined characteristics.

For simplicity and efficiency reasons, by default, multi-channel rasters are decomposed into a set of co-registered image planes with one to four channels each, stored in web-supported formats (PNG, JPG, WebP). These formats provide quantization and lossy+lossless compression and enable transparent natively-implemented accelerated decoding on the web client [MDN23]. Each image plane is quantized and compressed, and bias and scale values are stored in an image descriptor for dequantization. Large images are then decomposed into a tiled pyramid stored in one of the many possible multiresolution formats (e.g., *deepzoom*, *OpenStreetmap/GoogleMaps tiles*, or *IIIF*). Since, by default, all data is handled in standard formats, we can reuse existing tools (e.g., *vips* [VIP22]) for this task. Optionally, 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 easily move the entire representation among different machines and file systems, and supports the efficient extraction of individual tiles with simple HTTP Range requests through any modern standard HTTP server.

Client-side, an adaptive renderer incrementally fetches image tiles, adjusting the level of detail based on the zoom value, viewport size, and interaction behavior. For consistency, all the image planes comprising a given tile are grouped in an instance of the `Tile` class, which asynchronously fetches the planes and marks completion when all the planes have arrived. The `LayoutTiles` class organizes tiles at multiple resolutions, allowing the system to refine the displayed image as the user zooms in progressively. By prioritizing visible tiles and employing a cache of already loaded tiles, the system reduces visual latency and ensures that high-resolution textures replace low-resolution ones only when needed, reducing memory overhead. A fully interactive visualization experience is further supported by decoupling decoding and rendering of cached tiles from asynchronous tile retrieval. Rendering is performed by moving all the tiles to viewport-sized buffers and then executing the decoding and shading pipeline on them. Moreover, rather than only supporting the usual power-of-two refinement, *OpenLIME* makes it possible to decouple the decoding and rendering resolution from the resolution at which data is loaded, which is achieved by downscaling the buffers, performing decoding and rendering, and then upscaling the results to produce the final viewport colors for each layer. This makes it possible to meet interactivity constraints for heavy-to-decode data, such as neural representations (Sec. 4.2).

GPU-accelerated decoding and rendering are achieved by executing WebGL shaders, attached to layers and encapsulated in instances of the `Shader` class. The class simplifies WebGL shader management by automatically handling the allocation and updating of sampler units associated with image planes, uniform variables, and attribute variables, as well as compiling shader programs on the GPU. Shaders attached to *OpenLIME* layers access the various planes composing the image representation and use other information provided in uniform variables (e.g., light color, light direction, view direction) to produce a pixel's color. To support the easy dynamic insertion of post-processing operations, standard *OpenLIME* fragment shaders implement a parameterless data() function that returns a `vec4` color instead of the typical main() function. Instances of the `ShaderFilter` class also implement a `data()` function, taking the result of the previous `data()` as a parameter. At run-time, each time a filter is added or removed, *OpenLIME* automatically regenerates the proper main function that composes all the active filters, providing extensive versatility.

For multi-layered visualization, the `LayerCombiner` module provides various dynamic compositing options of multiple graphical layers via framebuffer operations and custom shaders. In addition to the common blending operations, the `LayerLens` module makes it possible to implement visualization lenses that define different combination modes inside and outside the lens focus area, by default a movable and scalable circle (see, e.g., Fig. 1, Fig. 3, and Fig. 4).

3.3. Decorations and annotations

Multi-layered visualizations of raster data are enhanced with active decoration and annotation elements, primarily using vector graphics. Decorations, fixed in viewport space, leverage a centralized Skin system that manages SVG assets through external files, enabling easy theming without altering core code. They also support dynamic toolbar creation via SVG element cloning and binding to event listeners (see Sec. 3.4). In contrast, annotations tie overlays and external information to specific model regions and, therefore, stay aligned with the model as the view changes.

The *OpenLIME* framework exposes an annotation system based on a hierarchy of specialized layer classes that are all children of the core `Layer` class. At the base is the `LayerAnnotation` class that provides the basic infrastructure for working with annotation collections. Instances of this class are responsible for loading annotations from remote sources, keeping selection states, and providing a single entry point to control the visibility of annotations. They link into the viewer's UI via a configurable annotations list entry system, whereby interactions with annotations are facilitated via a separate interface panel. On this foundation, `LayerSvgAnnotation` expands the annotation system to enable the rendering of vector graphics directly onto the canvas in the form of SVG elements. By placing the annotation elements in an SVG overlay that sits above the WebGL canvas, we can create complex vector graphics with interactive features that might have been difficult to achieve purely in WebGL. The SVG annotation layer guarantees that transformation synchronization between the WebGL coordinate system and the SVG viewBox is correctly handled. The styling of custom elements is also supported through a class-based system whereby different annotation types can be assigned different visual properties.

The `Annotation` class defines the properties and behaviors of a single annotation element in a very generic way. Each annotation instance holds information about metadata (ID, label, description), visual properties (SVG content, style, class), state (visibility,

selection), spatial information (bounding box, region), and extra custom properties for domain- or application-specific needs (see, e.g., Sec. 4.3 and Sec. 4.4). It is capable of handling many types of annotations like Vector-based SVG annotations (lines, shapes, text), Region annotations (rectangles, polygons), Point annotations (markers, pins), and aggregates of diverse element types.

The annotations can be added to the application either by passing an array of annotation objects or by using a JSON endpoint (local or remote). When loading annotations from a remote URL, a `LayerAnnotation` instance downloads annotation data, constructs one `Annotation` object for each entry using a distinct ID, and sets properties like initial visibility based on metadata. Once the processing is complete, the layer emits events to inform the system that the annotations are loaded and ready to be rendered.

3.4. Device mapping and user interface

OpenLIME unifies events generated by input devices to handle single-touch (mouse or pen) and multi-touch interaction through a portable abstraction layer built on top of *PointerEvents* [W3C25]. At the most fundamental level, a lightweight event handling system that employs the observer pattern is implemented through its `Signals` module. It extends prototypes with signal capabilities via an `addSignals` function, which registers named event types and provides methods to subscribe to those events, listen for one-time messages, and emit the events themselves. Throughout the framework, class instances use this system to communicate only with event listeners. Such architecture allows modules to be loosely coupled, avoiding direct dependencies. The event system supports standard features such as event registration (listeners), event removal, one-time listeners, and event capturing.

OpenLIME implements on top of `Signals` a comprehensive input interaction system through its `PointerManager` module that abstracts device-specific inputs into a unified event-handling framework. Based on the Web API's PointerEvent interface, it seamlessly supports mouse and touch interactions, maintaining consistent behavior across diverse input devices. The system processes single-pointer events individually while correlating multiple concurrent events to recognize complex gestures. At the low level, instances of the `SinglePointerHandler` class track single-pointer inputs throughout their complete lifecycle and maintain a state of the pointer's position, movement, and temporal characteristics. These individual events are then mapped into a higher-level gesture by a recognition system that identifies patterns such as taps, double-taps, holds, pans, and pinch-to-zoom operations.

The `PointerManager` architecture is consistent with the Web API's event-handling system. The original `PointerEvent` is preserved and enriched with additional data useful for the module's functionality. Native Event methods remain fully accessible—developers can, for example, stop event propagation using `stopPropagation()` or prevent default behaviors with `preventDefault()`. Handling events from input devices is greatly simplified by the recognition of gestural input – for example, writing just three callbacks (`PanStart`, `PanMove`, and `PanEnd`) is enough to define how the system should respond to a user's panning gesture.

A `PointerManager` instance works in concert with several `Controller` instances to define behaviors in response to user actions. Events circulate through controllers in a priority order, and controllers can capture events, preventing them from reaching other controllers until the controller releases control. This allows the implementation of per-layer controllers through the priority mechanism and composite actions through the capture/release mechanism. The system includes default controllers such as `ControllerPanZoom` for camera navigation, `ControllerLightSphere` for light direction, `ControllerLens` for lens positioning, and `ControllerFocusAndContext` for coordinated lens and camera control [BAMG21]. An example is discussed in Sec. 4.1 and illustrated in Fig. 4.

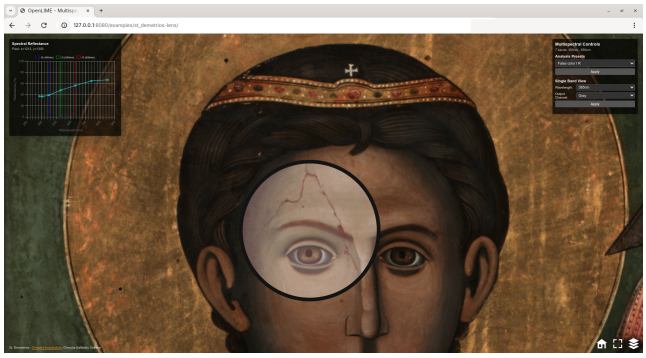


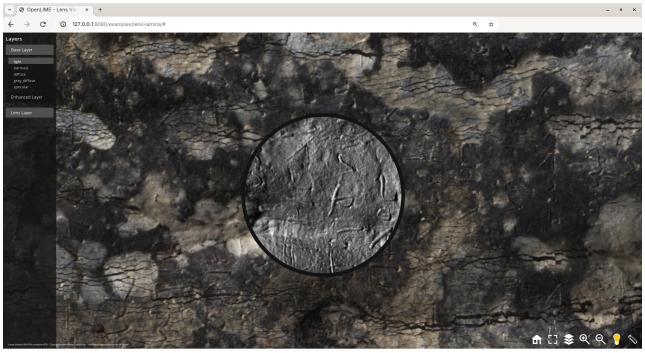
Figure 3: ***Multilayered multispectral visualization****. The model shows the 7-band albedo of a Saint-Demetrios Icon, computed from 50 photos under UV (385nm, 405nm), visible (RGB), and IR (740nm, 850nm) directional lighting. The graph displays the multispectral reflectance of a selected skin pixel. The lens shows False Color Infrared (R=IR-850nm, G=VIS-R, B=VIS-G), while the context shows visible channels.*

4. Use cases

To exemplify *OpenLIME*'s adaptability across use cases and setups, we briefly describe how it has been used to build applications with widely varying required features, supported data, and targeted user base. First, we show how the basic components provided by the framework are used to build an RTI processing and exploration pipeline (Sec. 4.1). Then, we discuss how the raster data formats and the shading pipeline have been integrated to design and support novel neural relighting models (Sec. 4.2). We finally focus on two CH-specific applications. The first one targets domain experts, and concerns the documentation and annotation of archaeological data (Sec. 4.3). The second targets instead the general public, with the virtual exploration of a massive mural using a standalone museum application driving a large touch screen (Sec. 4.4).

4.1. Inspecting acquired models

One of the main applications of scalable image viewers is remote inspection of artifacts. For instance, the ability of RTI to faithfully reproduce the appearance of objects with challenging materials makes a good case for museum and virtual exhibits aimed at the general public, where direct access to the artifact is inconvenient for any reason. At the same time, relightable images are used for analysis by professionals in a variety of cultural heritage applications such as inscriptions, cuneiform tablets, manuscripts, paintings, mosaics,



```
<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>OpenLIME - ...</title>
    <link rel="stylesheet" href="css/skin.css" />
    <style>
        /* ... style customization here */
    </style>
</head>
<body>
    <div class="openlime"></div>
    <script src="js/openlime.js"></script>
    <script>
      const lime = new OpenLIME.Viewer('.openlime');
      // Add a layer with RTI data
      const layer0 = new OpenLIME.Layer({
          label: 'Base Layer',
          layout: 'tarzoom', type: 'rti', url: 'assets/lamina/ptm/info.json',
      });
      layer0.setMode('light'); // default mode
      lime.addLayer('lamina', layer0);
      // Add a second layer with the same data, also enabling nornals
      const layer1 = new OpenLIME.Layer({
          label: 'Enhanced Layer',
          layout: 'tarzoom', type: 'rti', url: 'assets/lamina/ptm/info.json',
          normals: true
      });
      layer1.setMode('specular'); // default mode
      lime.addLayer('lamina-geo', layer1);
      // Combine the above layers in a third one through a lens
      const layer2 = new OpenLIME.Layer({
          type: "lens", camera: lime.camera,
          visible: true
      });
      lime.addLayer('Lens Layer', layer2);
      layer2.setBaseLayer(layer1);
      // Enable joint control of lens and camera
      const lensController = new OpenLIME.ControllerFocusContext({
            lensLayer: layer2
      });
      lime.addController(lensController);
      // Set skin
      OpenLIME.Skin.setUrl('skin/skin.svg');
      const ui = new OpenLIME.UIBasic(lime, {
          showLightDirections: true
      });
      // Enable interactive light control
      ui.actions.light.display = true;
      // Text to be printed for attribution
      ui.attribution = `...';
    </script>
</body>
</html>
```

Figure 4: ***Multilayered RTI visualization****. The simple code at the bottom creates the multilayered visualization on top. The RTI model is a lead sheet found in the 1960s in Caesarea Maritima. The item, now at the Archaeological Museum of Milan, was acquired to study the engraved inscriptions.*

coins, and medallions. The ability to change the light direction and apply enhancement algorithms on the dataset allows remote viewers to reveal subtle details of the surface. Similar consideration can be made for multi-spectral visualization, where combining visible and invisible light measurements helps analysis and exploration of models (see, e.g., Fig. 3). Both use cases are directly supported by *OpenLIME*'s base classes.

Fig. 4 illustrates the structure of a basic multilayered viewer, together with a screenshot taken during interactive inspection. The item, found in the 1960s in Caesarea Maritima, is a non-flat engraved lead sheet with different degrees of roughness. PTM data was created from images taken with a light dome (47 LED) and a Nikon D810 DSLR camera, to study hardly visible inscriptions. The example code creates two layers from the same PTM data in tarzoom format. A third layer provides a combined display, realized with a visualization lens with the second layer as the focus and the first as the context. A focus-and-context controller is then created to jointly control the lens and the camera. From this terse description, *OpenLIME* automatically creates the data loaders, the shaders for the various RTI display modes, the menus to choose the selected layer (first, second, or combined with lens), and the toolbar, which also includes modal camera/light direction. In the image above the code, the lens focus presents a synthetic monochrome reflective surface, while the context uses the original reflectance. Interactive relighting with a raking light emphasizes the inscriptions. The structure of the multispectral viewer depicted in Fig. 3 is very similar: only the layer types are modified, and the selection of channel combinations available in the interface is determined by a JSON file that associates a preset name to a particular weighted channel combination.

4.2. Exploring novel relightable image representations

Classic relightable models, such as PTM, HSH, and DMD, are compact and low-complexity formulations of widespread use for fast interactive relighting in local and remote visualization. Without extra information, however, these methods are limited to modeling only low-frequency behavior [PDC*19]. In recent years, neural networks have emerged as a viable technique for compression and nonlinear approximation from large amounts of data and have also been applied to rendering settings [TFT*20]. In the RTI realm, the NeuralRTI approach [DFP*20, DRP*24] introduced a fully connected asymmetric autoencoder to encode the original per-pixel information into a low-dimensional vector and decode it to reconstruct pixel values from the pixel encoding and a novel light direction. The image quality is higher than that of classic solutions at equal storage cost [DFP*20]. *OpenLIME* offered a framework where the method could be integrated as an optimized first-class component interoperable with the rest of the platform [RKG*24].

In NeuralRTI, 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 to produce $K \ll 3N$ 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. Thus, once the training phase is finished, it is possible to use the encoder to produce a static map of per-pixel latent features. Afterward, the encoder is discarded, and relighted images can be computed just from the per-pixel latent features and interactively set light directions, which feed the decoder, whose parameters (i.e., weights and biases) are common for the entire image.

The storage and 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, which can be loaded once and for all at model opening time. In a typical implementation [DFP*20], latent maps require K=9 channels per pixel, which *OpenLIME* separates into

three different RGB images. Righetto et al. [RKG*24] have empirically determined that, while the NeuralRTI representation is not linear, averaging nearby latent space features tends 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). This makes it possible to reuse the same machinery used for plain images and PTM to produce tiled pyramids of latent features. The only custom component of the processing pipeline is, thus, the encoding, based on *Python* and *Keras*, for transforming the input high-resolution MLIC into latent feature maps stored as separate images. Pyramid creation, tiling, and compaction are then performed with *vips* [VIP22] and *OpenLIME*'s *tarzoom*, see Sec. 3.2. For compressing the latent maps, the execution on a web platform encourages the usage of natively supported PNG, JPEG, or WebP images [MDN23]. Since NeuralRTI's latent features have structures similar to those of RGB photographic images, JPEG was selected for compression, but, since neural coefficient values are not directly perceived as colors by users in the final image (unlike, e.g., PTM), JPEG compression is set up by disabling RGB to YUV conversion, deactivating chroma subsampling, and using non-biased quantization tables.

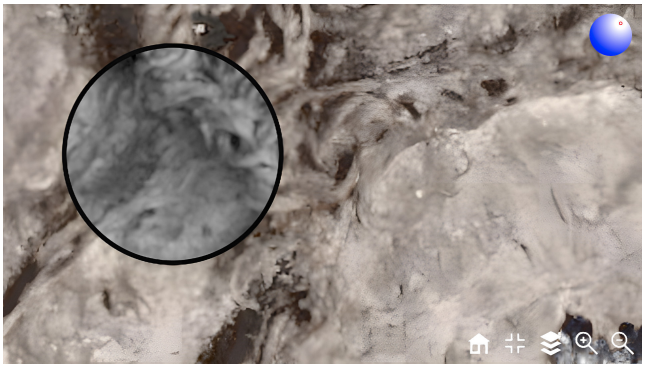


Figure 5: ***Multilayered rendering of neural and classic relightable models****. Inside the lens, the model is rendered from a PTM format, while outside it is rendered using a NeuralRTI representation, which provides increased details at similar compression rates. The desaturation effect in the lens is obtained through the concatenation of a shader filter. Both models are derived from a high-resolution MLIC capture of textile artifacts from the Oseberg Find from a Viking Age burial mound at Oseberg in South Norway.*

The integration into the rendering pipeline was, then, realized by implementing a specific `NeuralRTILayer`, extending the basic `Layer` class. The class loads from a JSON file the decoder weights and biases and refers to the latent feature maps through three image pyramids, which are asynchronously loaded into buffers during exploration using the standard tiled image loader of *OpenLIME*. The layer contains a standard Vertex shader and a custom WebGL 2 fragment shader. The fragment shader has weights and biases stored in uniform arrays, and, at execution, samples the latent space feature from the three texture maps using the pixel's coordinate, combines it with a given light direction, provided as uniform variables, and propagates these values through the network until we reach the final stage with the output color. Each layer computes 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). Since all the pixels compute the same sequence of vectorized operations to produce their output, and all dot products and sums are aligned on 16-byte boundaries, shading threads do not diverge.

`NeuralRTILayer` is a standard *OpenLIME* layer, and `NeuralRTIShader` is a standard *OpenLIME* shader. Thus, one can exploit our dataflow design to combine NeuralRTI color computation with a sequence of post-processing operations through cascading filter integration in the fragment shaders. In particular, gamma, brightness, color-remapping, and contrast correction on neural renderings can be achieved using the same tools the framework makes available to all other rendering representations (Normal+BRDF, PTM, HSH, RBF). Moreover, multi-layered visualizations can mix a `NeuraRTILayer` with other standard layers with no particular coding effort (see, e.g., Fig. 5).

4.3. Annotating archaeological findings

In addition to visual inspection, archaeologists need to document, discuss, and present their interpretations through annotations on images of the artefact. Annotating directly on an RTI, rather than on a static image, is more effective, particularly when physical access to the object is limited, since analyzing the object under different illumination conditions may reveal hidden features and patterns [PDC*19].

An example of *OpenLIME*'s relighting and annotation support's usage is the documentation of an inscribed limestone block discovered in Marseille, France: *Le Bloc de l'Alcazar*. Approximately two meters wide, this stone is densely covered with overlapping graffiti dating back over 2,600 years. These engravings' complexity, layering, and faintness made them very difficult to decipher through conventional imaging or direct observation.

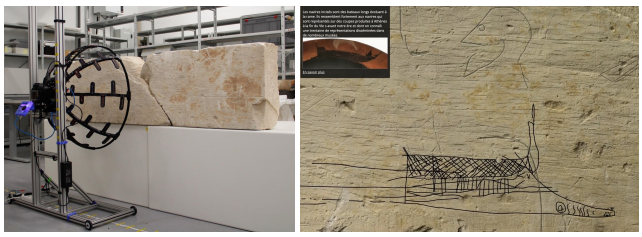


Figure 6: ***RTI capture and annotation****. Left: A dome with 100 lights was used to capture 120 RTI datasets; Light direction and intensity were compensated, and photogrammetry was employed to merge the datasets into a single RTI panorama, resulting in 600 million pixels. Right: annotation of Greek pentekonteres (long warships).*

120 RTI datasets of the block's surface were captured with a 100-light dome (Fig. 6 left). Light directions and intensity were compensated, and photogrammetry was employed to merge the datasets into a single RTI panorama, resulting in a 600Mpixel raster in the PTM format. Converted to tarzoom (Sec. 3.2), a multiresolution relightable model was then made available online for scholarly inspection and annotation.

OpenLIME includes an integrated annotation editor, implemented in the `EditorSvgAnnotation` class. Users can draw points, lines, polygons, and rectangles directly on the surface to create annotations, and specify properties such as label, description, and class (Sec. 3.3). Freehand drawing of lines and polygons is enhanced with

a simplification algorithm that reduces annotation complexity while preserving shape, and supports fine path adjustments, similar to vector editing in graphic design tools. Freehand drawing is facilitated by the native support for drawing tablets, with pen input for drawing and multitouch input for image scaling and panning (Sec. 3.4). As most users still rely on a mouse connected to a desktop or laptop, however, manual drawing can be cumbersome and imprecise, and the ability to edit paths significantly improves the user experience. We plan to further extend this approach to guide the user during annotation, in particular by using feature detection on relighted models and snapping the annotation markup to the detected features.

To create, edit, or delete annotations stored in a repository, *OpenLIME* was extended to provide a Representational State Transfer Application Programming Interface (REST API) along with a minimal backend implemented in `PHP` or `Express.js`, supporting either file-based or relational database storage. The current simple supported workflow allows each user to independently add or remove annotations without an explicit locking mechanisms. Concurrent edits are managed by handling each annotation update through API endpoints that execute database operations within transactions. As future work, we plan to extend this basic protocol to support automatic local database view updates through notification broadcasts and annotation versioning. The current setup enables easy deployment of a basic annotation backend for custom applications or straightforward integration with existing systems through simple middleware, and has been later integrated into *OpenLIME*'s core. By installing *OpenLIME* on a shared accessible server, scholars from around the world could remotely access the RTI, highlight features of interest, share observations, and collaboratively develop interpretations. Through this digital platform, researchers identified iconography of ships (Fig. 6 right), numerous human and animal figures, and Greek letters. Finally, a presentation setup, without annotation editing features, was created and opened to the public.

4.4. Interactive museum presentation

Figure 7: ***Museum exploration of an annotated multi-layered model****. Image taken live during the Cabras exhibition. Another image of the same exhibition is in Fig. 1. The* OpenLIME *instance is isolated and running from a small Intel NUC placed behind the multitouch display.*

Interactive exploration of high-quality digital replicas of artworks is increasingly common in museums since it enhances the overall experience and engages visitors better than passive media [FDAB*24]. Here, we illustrate how *OpenLIME* was employed to support a large multi-disciplinary project targeting the physical reproduction, digital documentation, and public presentation of a monumental artwork [AAB*22]. The artwork is a semi-abstract 23m × 5m sandcast relief by the Italian sculptor Costantino Nivola that was created in 1954 for the Olivetti Showroom on 5th Avenue in Manhattan [AC22]. The artwork, originally conceived to be painted, was installed preserving the natural appearance of the sand alone. The artwork was then moved to the Harvard University Science Center in 1972, where color reminiscent of the earlier studies was applied. In two exhibitions, a first one where a virtual presentation enriched a physical replica [ACCS], and a second purely virtual one [ACC*], *OpenLIME* applications let visitors explore an annotated multi-layered version of the artwork (see Fig. 7).

The exhibition curators desired to use digital means to inform and engage the visitor, focusing on school audiences and casual museum-goers, adapting the quantity and the quality of information accordingly. Nivola's sandcast relief blends myth and modernity, featuring Sardinian gods, dynamic figures, marine textures, personal imprints, and rhythmic patterns [AC22]. Due to its intricate visual and semantic complexity, domain experts decided to design the museum presentation around annotation-driven exploration for guided and interactive tours, and to offer a comparison between Manhattan's original plain installation and Harvard's colored version.

To create the *OpenLIME* base image data, the 3D model acquired and reconstructed for documentation and fabrication purposes was rasterized at ≈2 pixel/mm through orthographic rendering of the mural using *MeshLab* [CCC*08], resulting in a normal map, a monochrome Lambertian BRDF map, to show the model without a default sand color as in the Olivetti Showroom, and a colored Lambertian BRDF map, to show the current state as acquired in Harvard. The maps, each 45695 × 14953 (683M) pixels, were then converted using the same tools discussed in Sec. 3.2 to two *OpenLIME* Layers, each referencing tiled pyramids in the normal+BRDF format.

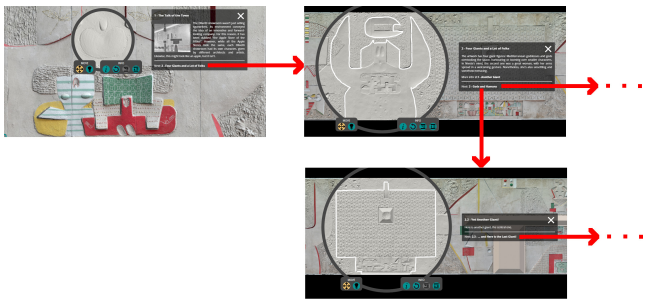


Figure 8: ***Hierarchical annotations****. A multi-level visit is created by connecting annotations. Each annotation connects a particular position on the mural to a viewing state, a visual overlay, and a text + video + audio description. Connections to the next annotation at the same level and to the first at the more detailed level structure the visit.*

The curators then created annotations using *OpenLIME* tools (Sec. 3.3), classifying them thematically in *History*, *Content*, *Technique and style*, and *Trivia*. In addition, they were divided according to their importance, creating three possible depths of exploration to accommodate the viewer's available time and level of interest. Each annotation connects a particular position on the mural to a *viewing state*, *visual overlay*, and *description*. The viewing state

is a recording of the viewer parameters (i.e., camera, lighting, active layers, and so on) at the moment of annotation creation. The *visual overlay* is an SVG markup, and the *description* contains a short rich text and an optional audio clip. A natural sequence was then created by connecting each annotation to the next one at the same level (if available) and the first at the more detailed level (if available). As a result, an annotation tree was obtained (see Fig. 8). This enhanced annotation structure was created by extending, using the extra custom properties field, the basic annotation provided by the *OpenLIME* core, described in Sec. 3.3.

To obtain a walk-up-and-use installation, a simplified user interface lets visitors control a single virtual object: an interactive visualization lens. Moving and scaling the lens positions the focus area and automatically centers the viewport around it [BAMG21]. Inside the lens, the monochrome layer is displayed, while outside the lens, visitors see the colored model. Moreover, an attached dashboard around the lens lets users trigger all interactively controlled actions, including enabling relighting by light direction manipulation and navigation through annotations. The area for displaying the annotation description is also attached to the lens. To avoid clutter, only one annotation at a time is presented. Moreover, a preferred path connecting the different annotations was authored. This path connects each annotation to the next one at the same level (if available) and the first at a more detailed level (if available). An interactive audiovisual tour, presenting all the annotations in the authored order, is triggered when explicitly requested with a user action or the application remains idle for a predefined amount of time (i.e., no visitors are interacting with it). Instead, when visitors interact with the lens, they can explore areas of the mural by panning and zooming, or move to annotated areas by selecting a "next" or "down" button. When a new annotation is activated, its rich text is displayed near the lens, and the audio clip, if present, is played.

The Nivola Mural Museum Application is deployed on a compact and low-cost Intel NUC8BEB, which has an Intel Core i7-8559U processor (2.70GHz, 4 cores/8 threads), 16GB DDR4-2400 RAM, and a 1TB Samsung SSD 860 EVO for quick storage. The system has Intel Iris Plus Graphics 655 with 7931MB of video memory. The application is shown on an 86-inch Iiyama UHD 4K touchscreen display (3840×2160, 400 cd/m²) in landscape mode. The system is completely standalone and does not need a network connection to operate. It is running in a streamlined Manjaro Linux/X11 environment, with the content being served locally from a *Caddy* [Mat19] web server. The kiosk implementation is done via a custom system service that starts at boot and ensures reliable operation by using a shell script that detects which display is present and connected, sets the appropriate resolution, turns off screen timeout, hides the cursor, and then launches *Chromium* in kiosk mode. This makes for a dedicated, low-maintenance interactive exhibit that runs automatically at system boot through *systemd* integration.

5. Conclusions

We have presented *OpenLIME*, a scalable web-based framework for creating local, mobile, and online applications for interactive inspection of annotated multi-layered, multi-channel and/or relightable image models. It efficiently and flexibly manages large annotated raster datasets in multiple image formats using a data-flow approach, supporting advanced inspection through real-time adaptive streaming and client-side WebGL-based processing. With a modular and responsive design, *OpenLIME* supports desktop, mobile, and multitouch environments, facilitating rapid deployment for web-based datasets and kiosk applications. The presented use cases demonstrate that, in its current form, the framework is already capable of supporting a range of use cases, including multispectral and relightable image inspection, visual computing research, archaeological documentation, and museum applications. The framework is released as open source [Ope25] and can be readily used for applications in cultural heritage and other application domains. We are currently working on implementing configurable pipelines using buffers with different bit counts, extending the lens subsystem to support general multi-faceted data visualization, and integrating the viewer into a general cloud-based system for annotating and inspecting cultural heritage models, also using it as a tool to inspect areas of 3D models.

Acknowledgments The project received funding from the EU under Grant Agreement 101157364 – ECHOES and from Sardinian Regional Authorities under project XDATA (RAS Art9 LR 20/2015). Project REFLEX (PRIN2022, EU Next-GenerationEU PNRR M4C2 Inv. 1.1) contributed to supporting the study of NeuralRTI components. The authors thank: The Columbia Imaging and Vision Laboratory (CAVE) at Columbia University for multispectral data [YMIN08] (Fig. 1-1); Tomasz Lojewski and the AGH University of Science and Technology in Krakow for the provision of the Oseberg Find data (Fig. 1-2 and Fig. 5); CNRS, Mercurio Imaging and the Marseille History Museum for the Alcazar collaboration (Fig. 1-3, Fig. 6); Giuliana Altea, Antonella Camarda, and the Nivola Museum for their contribution to the Nivola project (Fig. 1-4, Fig. 7, and Fig. 8); Ormylia Foundation for the Saint Demetrios icon (Fig. 3); Attilio Mastrocinque and the Civic Archaeological Museum in Milan for the lamina item (Fig. 4).

References

[A*24] Appleby M., et al.: Image API 3.0, 2024. [Accessed 2025-03-31]. URL: https://iiif.io/api/image/3.0/. 3

[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* (2022), pp. 85–94. doi:10.2312/gch.20221230. 8

[AC22] Altea G., Camarda A. (Eds.): *Lo showroom Olivetti a New York. Costantino Nivola e la cultura italiana negli Stati Uniti*. Edizioni di Comunità, Roma/Ivrea, 2022. 8

[ACC*] Altea G., Camarda A., Cheri L., Depalmas A., Stein C.: On the Shoulders of Giants. The Modern Prehistory of Costantino Nivola. Museo Civico Giovanni Marongiu, Cabras & Museo Nivola, Orani, Italy. Nov 30, 2024–Mar 23, 2025. 8

[ACCS] Altea G., Camarda A., Cheri L., Stein C.: Nivola and New York. From the Olivetti Showroom to the Unbelievable City. Museo Nivola, Orani, Italy. Apr 15-Aug 29, 2022. 8

[AMP*23] Abergel V., Manuel A., Pamart A., Cao I., De Luca L.: Aïoli: A reality-based 3D annotation cloud platform for the collaborative documentation of cultural heritage artefacts. *DAACH 30* (2023), e00285. doi:10.1016/j.daach.2023.e00285. 1, 2

[AMPG22] Ahsan M., Marton F., Pintus R., Gobbetti E.: Audiovisual annotation graphs for guiding lens-based scene exploration. *Computers & Graphics 105* (2022), 131–145. doi:10.1016/j.cag.2022.05.003. 2

[BAMG21] Bettio F., Ahsan M., Marton F., Gobbetti E.: A novel approach for exploring annotated data with interactive lenses. *CGF 40*, 3 (2021), 387–398. doi:10.1111/cgf.14315. 2, 5, 9

[CCC*08] Cignoni P., Callieri M., Corsini M., Dellepiane M., Ganovelli F., Ranzuglia G.: MeshLab: an Open-Source Mesh Processing Tool. In *EG Italy* (2008), pp. 129–

136. doi:10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136. 8

[CCDL*20] CROCE V., CAROTI G., DE LUCA L., PIEMONTE A., VÉRON P.: Semantic annotations on heritage models: 2D/3D approaches and future research challenges. *ISPRS Archives 43* (2020), 829–836. doi:10.5194/isprs-archives-XLIII-B2-2020-829-2020. 1

[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. 6

[DHL17] DHLAB: RTI tools, 2017. [Accessed 2025-03-17]. URL: https://github.com/dhlab-basel/rti.js. 2

[DRP*24] DULECHA T., RIGHETTO L., PINTUS R., GOBBETTI E., GIACHETTI A.: Disk-NeuralRTI: Optimized NeuralRTI relighting through knowledge distillation. In *Proc. STAG* (2024). doi:10.2312/stag.20241340. 6

[FBKR17] FORNARO P., BIANCO A., KAISER A., ROSENTHALER L.: Enhanced RTI for gloss reproduction. *Elec. Imaging 2017*, 8 (2017), 66–72. doi:10.2352/ISSN.2470-1173.2017.8.MAAP-284. 2

[FDAB*24] FURFERI R., DI ANGELO L., BERTINI M., MAZZANTI P., DE VECCHIS K., BIFFI M.: Enhancing traditional museum fruition: current state and emerging tendencies. *Heritage Science 12*, 1 (2024), 20. doi:10.1186/s40494-024-01139-y. 1, 2, 8

[GKPB04] GAUTRON P., KRIVÁNEK 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

[JAP*21] JASPE 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

[JH15] JANKOWSKI J., HACHET M.: Advances in interaction with 3D environments. *CGF 34*, 1 (2015), 152–190. doi:10.1111/cgf.12466. 2

[KF22] KUZIO O., FARNAND S.: Comparing practical spectral imaging methods for cultural heritage studio photography. *ACM JOCCH 16*, 1 (2022). doi:10.1145/3531019. 2

[KUL19] KUL: PLD, 2019. [Accessed 2025-03-17]. URL: https://portablelightdome.wordpress.com/software. 2

[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. 2

[Mat19] MATTHEW HOLT AND OTHERS: Caddy, 2019. [Accessed 2025-03-24]. URL: https://github.com/caddyserver/caddy. 9

[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. 2, 3, 4, 7

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

[MHJ22] MEINECKE C., HALL C., JÄNICKE S.: Towards enhancing virtual museums by contextualizing art through interactive visualizations. *ACM JOCCH 15*, 4 (2022), 1–26. doi:10.1145/3527619. 1

[Mic08] MICROSOFT: DeepZooom, 2008. [Accessed 2025-03-19]. URL: http://www.seadragon.com/developer/creating-content/file-formats/. 3

[Ope24] OPENSTREETMAPWIKI: Slippy map tilenames, 2024. [Accessed 2025-03-31]. URL: https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames. 3

[Ope25] OPENLIME TEAM: OpenLime: Open Layered IMage Explorer, 2025. [Accessed 2025-03-17]. URL: https://github.com/cnr-isti-vclab/openlime and https://github.com/crs4/openlime. 2, 9

[P*10] PALMA G., ET AL.: RTIViewer, 2010. [Accessed 2025-03-17]. URL: https://vcg.isti.cnr.it/~palma/rti/rtiviewer.php. 2

[P*15] PALMA G., ET AL.: WebRTI viewer, 2015. [Accessed 2025-03-17]. URL: http://vcg.isti.cnr.it/rti/webviewer.php. 2

[P*19] PONCHIO F., ET AL.: Relight, 2019. [Accessed 2025-03-17]. URL: http://vcg.isti.cnr.it/relight/. 2

[P*23] PILLAY R., ET AL.: IIIPImage 1.2, 2023. [Accessed 2025-03-31]. URL: https://iipimage.sourceforge.io/. 2

[PCC*10] PALMA G., CORSINI M., CIGNONI P., SCOPIGNO R., MUDGE M.: Dynamic shading enhancement for reflectance transformation imaging. *ACM JOCCH 3*, 2 (2010), 1–20. doi:10.1145/1841317.184132. 2

[PCDS20] PONCHIO F., CALLIERI M., DELLEPIANE M., SCOPIGNO R.: Effective annotations over 3D models. *CGF 39*, 1 (2020), 89–105. doi:10.1111/cgf.13664. 1, 2

[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. 2

[PDC*19] PINTUS R., DULECHA T., CIORTAN I., GOBBETTI E., GIACHETTI A.: State-of-the-art in multi-light image collections for surface visualization and analysis. *CGF 38*, 3 (2019), 909–934. doi:10.1111/cgf.13732. 1, 2, 6, 7

[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 *Opt. Meas. Systems for Industr. Inspect. IX* (2015), vol. 9525, pp. 489–498. doi:10.1117/12.2184840. 2

[PPY*16] PINTUS R., PAL K., YANG Y., WEYRICH T., GOBBETTI E., RUSHMEIER H.: A survey of geometric analysis in cultural heritage. *CGF 35*, 1 (2016), 4–31. doi:10.1111/cgf.12668. 1

[RKG*24] RIGHETTO L., KHADEMIZADEH M., GIACHETTI A., PONCHIO F., GIGILASHVILI D., BETTIO F., GOBBETTI E.: Efficient and user-friendly visualization of neural relightable images for cultural heritage applications. *ACM JOCCH 17*, 4 (2024), 54:1–54:24. doi:10.1145/3690390. 6, 7

[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. *CGF 39*, 2 (2020), 701–727. doi:10.1111/cgf.14022. 6

[TLPW24] TANG Y., LIU L., PAN T., WU Z.: A bibliometric analysis of cultural heritage visualisation based on web of science from 1998 to 2023: a literature overview. *Human. and Social Sciences Commun. 11*, 1 (2024), 1–11. doi:10.1057/s41599-024-03567-4. 1

[VETL18] VANHULST P., EVEQUOZ F., TUOR R., LALANNE D.: A descriptive attribute-based framework for annotations in data visualization. In *Proc. VISIGRAPP* (2018), pp. 143–166. doi:10.1007/978-3-030-26756-8_7. 1

[VIP22] VIPS: libvips: A fast image processing library with low memory needs, 2022. [Accessed 2025-03-19]. URL: https://www.libvips.org/. 4, 7

[VPH*20] VANWEDDINGEN V., PROESMANS M., HAMEEUW H., VANDERMEULEN B., VAN DER PERRE A., VASTENHOUD C., LEMMERS F., L. W., L. V. G.: Pixel+ viewer, 2020. [Accessed 2025-06-24]. URL: https://www.heritage-visualisation.org/pixelplusviewer.html. 2

[W3C25] W3C: Pointer Events Level 4, 2025. [Accessed 2025-03-18]. URL: https://www.w3.org/TR/pointerevents4/. 5

[YMIN08] YASUMA F., MITSUNAGA T., ISO D., NAYAR S.: *Generalized assorted pixel camera: post-capture control of resolution, dynamic range, and spectrum*. Tech. Rep. CUCS-061-08, CS Dept., Columbia University, 2008. 9