

Variable-axes Topographical Maps: A Compute-efficient Alternative for Visualizing Large Photogrammetry Models

Hussein Yaseen; University of Mosul; Mosul, Iraq. Adil Al Qaba; University of Mosul; Mosul, Iraq. Basim Mahmood; University of Mosul; Mosul, Iraq. Zohair Al Ameen; University of Mosul; Mosul, Iraq. Julia Schnitzer; University of Brandenburg for Applied Sciences; Brandenburg, Germany

Abstract

This paper presents a space-efficient alternative for visualizing 3D data; it utilizes topographical maps to allow easier editing and viewing of 3D models. It highlights applications in generating progressive level of detail (LODs) and offers a straightforward way to generate preview models that allow users interaction. This offers a geometrically-accurate model compared to more commonly used representations for such tasks (e.g. point clouds). The topographical model is achieved using incremental slicing in each of the dimensions resulting in a lighter, more uniform shape. In our testing, we observed an improved 35.4% in memory usage, 7 times faster rendering time, and 82.4% in space savings on average compared to mesh, wireframe, and point cloud. We also provide a browser-based demo that exports standard GLB files.

1. Introduction

Photogrammetric reconstruction routinely produces 3D models of heritage sites containing tens of millions of triangles or hundreds of millions of points. We use large-scale throughout to mean exactly this primitive count: the physical extent of a site matters for performance only because larger sites tend to be captured at higher absolute triangle and point counts. At these sizes, the interactive loop that scholars and conservators depend on - orbiting, sectioning, measuring and annotating the model - degrades below the roughly 30 frames per second needed for fluid editing [1]. The same density inflates storage and bandwidth, which limits how readily these models can be shared on the web or worked with in regions with constrained connectivity.

Practitioners typically fall back on one of three representations, each with a drawback for the editing loop (Figure 1). Dense meshes are faithful but fill-rate bound and heavy to store. Point clouds are cheap to capture but present ambiguous boundaries, which complicates the snapping and tracing operations central to Heritage Building Information Modelling (HBIM) [2], [16]. Wireframes are lighter to draw but expose the full triangulation and offer no semantic simplification. Crucially, generating variable levels of detail from any of these - and tuning how much precision each level encodes - remains awkward (Section 2).



Figure 1. The same heritage model in four representations - from left to right: our variable-axes topographic lines, a point cloud, a wireframe, and a mesh. The top row shows the full model; the bottom row shows a zoomed-in detail.

We propose a different representation rather than a coarser version of the same one. Our contributions are: (i) a variable-axes slicing method that turns a mesh into a 3D topographic line model whose density is independently tunable per axis (Section 3); (ii) a progressive-LOD and saliency-ranked compression scheme over those lines (Sections 3.4-3.5); (iii) an encoding entirely within the glTF/GLB standard, so existing web viewers render the result unmodified (Section 3.6); and (iv) an open browser-based reference implementation and dataset release (Sections 4 and 9). Figure 3 shows the approach on a real heritage site, where the structure is drawn as topographic lines while a foreground element is kept as a full render to save resources.

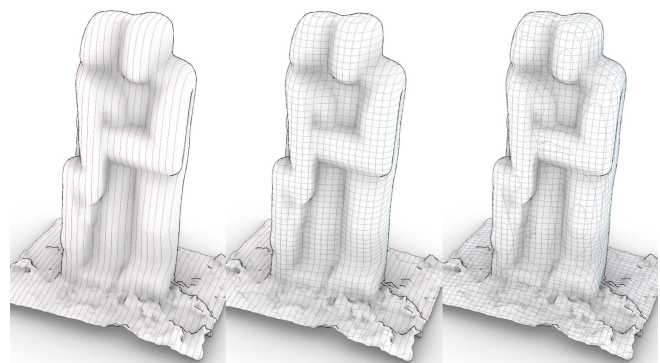


Figure 2. The topographic line representation. From left to right: (a) contours along the X-axis; (b) the X-Y combination; (c) the full X-Y-Z shell.

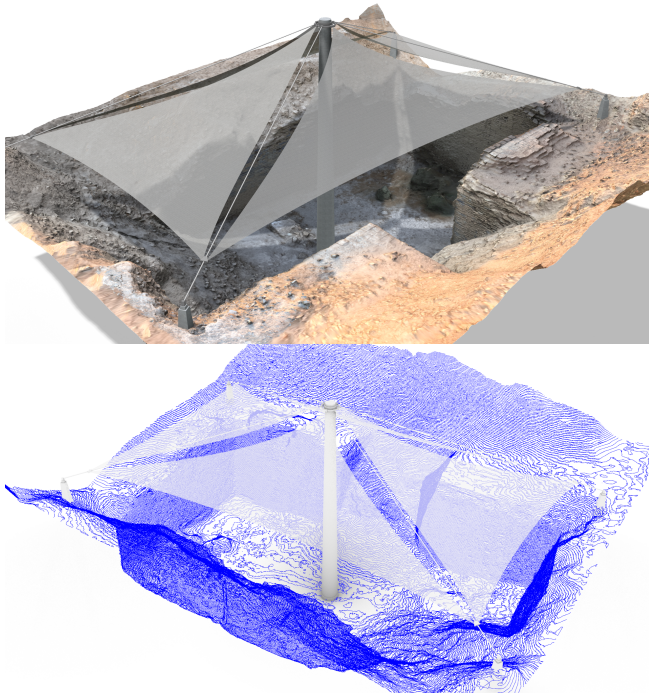


Figure 3. The method on a real heritage site: (a) the full photogrammetry render; (b) the same site as variable-axes topographic lines, with only the foreground umbrella kept as a full render to save resources.

2. Background and Related Work

Level-of-detail (LOD) generation is well studied in computer graphics. Mesh simplification by quadric error metrics [4] and progressive meshes [5] produce a continuum of lower-polygon approximations, and modern virtualized-geometry systems stream fine-grained clusters view-dependently. These methods are highly effective at what they target - reducing triangle count while bounding visual error - but they preserve the same representation: a filled surface. For the heritage editing loop this carries three costs. First, rendering remains fill-rate and shading bound, so the gains are sublinear in the perceived simplification. Second, aggressive decimation erodes precisely the sharp edges, tool marks and profiles that conservators care about, so the fidelity-versus-cost trade-off is dictated by the geometry rather than the task. Third, a decimated surface still offers no privileged lines to snap or measure against. Point-cloud pipelines take the complementary route, streaming octree-organized subsets so that only as many points as the viewport needs are drawn [6]. This scales well but inherits the boundary ambiguity of points [2], [3]: there is no surface, so edges, sections and contacts must be inferred - exactly where HBIM editing needs determinacy.

Contour and section drawings are the traditional language of architectural and archaeological documentation, and recent work argues for exploiting all geometric dimensions of a 3D surface rather than a single projection [1]. Our method can be read as generalizing single-axis contouring to three (or more) axes and treating the resulting line set as a first-class, editable, streamable representation. In contrast to decimation and point-cloud LOD, which make the same representation cheaper, we change the representation to one

that is sparse, boundary-explicit and cheap to draw, then recover LOD and compression as natural operations on it.

3. Method

3.1. Input and Preprocessing

Our input is a triangle mesh, the standard output of a photogrammetry pipeline after dense reconstruction and surface meshing. We assume a mesh rather than a raw point cloud deliberately: slicing a point cloud directly is ill-posed because a cutting plane almost never coincides with sampled points, whereas a mesh provides the continuous surface a plane can intersect. When only a point cloud is available we first reconstruct a surface (for example, screened Poisson reconstruction) [3]. The input is therefore not a heightmap or single-view depth map but a full 3D surface, which is what allows slicing along all three axes. Reconstruction noise, when present, appears as small lateral wobble in the contours; we optionally apply a light Laplacian smoothing along each 1D contour to suppress it without affecting the global shape.

3.2. Variable-axes Slicing

For each enabled axis we compute the model's axis-aligned bounding box and place N cutting planes at evenly spaced positions $t_i = \min + \text{extent} * (i + 0.5) / N$, where N is the per-axis line count. The half-step inset keeps planes off the bounding-box faces, avoiding degenerate intersections. Each plane is intersected with the mesh by the standard triangle-plane test, in the lineage of classic contouring methods [9]: a triangle straddling the plane has exactly two edges crossing it, yielding two points and hence one line segment, while triangles lying entirely on one side contribute nothing. Testing every triangle against every plane is wasteful on multi-million-triangle models, so we build a bounding-volume hierarchy (BVH) once per model and query it per plane, visiting only the triangles whose bounds straddle the plane - reducing the per-plane cost from linear to logarithmic in triangle count and making interactive density changes feasible. The output for one axis is a single batch of line segments; combining the three axes yields the grid-like contour shell of Figure 2.

3.3. Representation and Rendering

A natural question is whether the contours are converted to a low-polygon mesh, or rasterized in some special way, before display. They are neither. The representation is native line geometry - a vertex buffer of segment endpoints with a line index buffer - drawn directly as line primitives, one draw call per axis regardless of line count. Because lines carry no faces, rendering incurs no fill-rate or shading cost, which is the source of the frame-time advantage in Section 6. The same property makes the representation trivial to serialize: it maps one-to-one onto a glTF line primitive (Section 3.6).

3.4. Progressive Level of Detail

Line count is a direct, continuous control over detail, so a family of LODs follows for free: a coarse tier uses few planes per axis and a fine tier many. We construct nested tiers in which each finer tier is a superset of the coarser one, so refinement only adds lines and never reloads geometry already on screen. A viewer can therefore stream the major contours first and progressively densify as bandwidth

allows (Figure 4), which is particularly valuable for low-connectivity access to heritage collections.

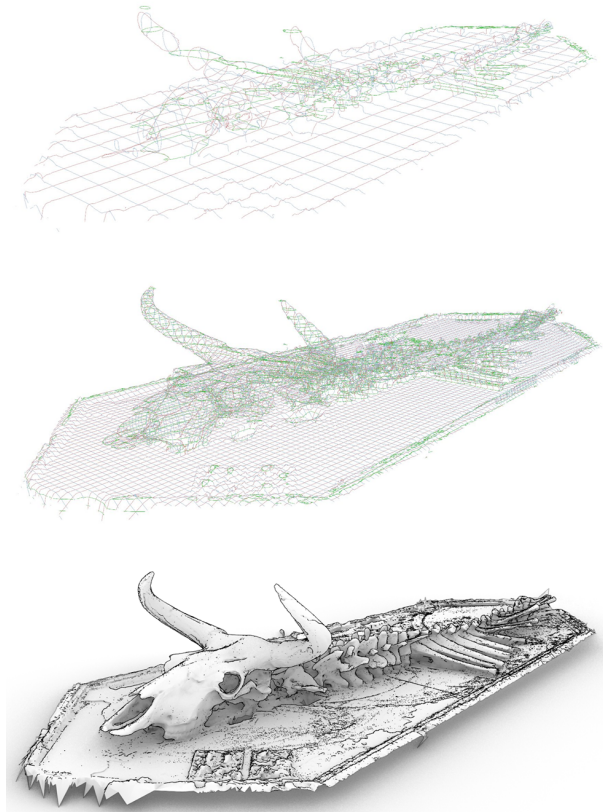


Figure 4. Progressive level of detail, from the major contours (top) to finer lines (bottom) until the shape is fully resolved. Finer tiers are supersets of coarser ones, so refinement only adds geometry.

3.5. Saliency-ranked Compression

Not every contour contributes equally to the perceived shape: lines crossing flat regions are largely redundant, while those tracing edges and reliefs carry the form. We therefore compress by ranking individual lines and discarding low-ranked ones below a threshold (Figure 5). We drive this ranking by mesh saliency [7], [8] so that lines passing through visually important regions are retained preferentially and pruning concentrates where it is least perceptible. This couples compression to a principled, perceptual criterion rather than a purely geometric one, and the same saliency signal offers a route to adaptive density - spending lines where the surface is informative - which we are actively integrating (Sections 6 and 8).

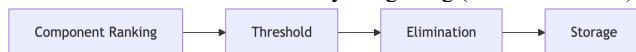


Figure 5. Saliency-ranked compression. Each line is ranked by its contribution to the perceived shape (via mesh saliency), thresholded, and low-ranked lines are eliminated before storage

3.6. Encoding Within glTF/GLB

A central concern for adoption is whether the representation fits existing formats and viewers or demands a new pipeline. It fits. Each axis exports as a glTF 2.0 [10] primitive in LINES mode with an unlit material (KHR_materials_unlit [11]) carrying the axis colour, grouped under a single named Topography node. Any glTF viewer draws these primitives today. To make the topographic layer toggleable we attach the ratified KHR_node_visibility extension [12] to the Topography node and, as a fallback, rely on the named-node convention that every scene-graph viewer and modelling tool already supports, so viewers that do not implement the extension simply show the layer rather than failing. The lines also import cleanly into CAD and modelling software, where endpoint grid-snapping lets a conservator trace, measure and align against the contours instead of the dense mesh - answering how the sliced output is used in subsequent editing. The same geometry is expressible in OpenUSD as UsdGeomBasisCurves [15], which we note as a straightforward path to USDZ (Section 8).

4. Implementation and Interactive Tool

We implement the method as a static, client-side web application built on three.js, with BVH construction and slicing running in a Web Worker so the interface stays responsive during recomputation; Draco- [13] and meshopt-compressed [14] inputs are decoded on load so real-world compressed scans open directly. Because everything runs in the browser, the model never leaves the user's machine - a relevant property for heritage institutions with data-sovereignty constraints. The tool (Figure 6) exposes one density slider per axis with an optional lock, per-axis colour, a source-mesh toggle, and live statistics (triangle count, per-axis segment counts and byte sizes), and it exports the result as a GLB containing the original model plus the toggleable Topography layer. The tool thus doubles as the empirical answer to the format and viewer questions of Section 3.6. The source code and reference implementation are openly available (Section 9).

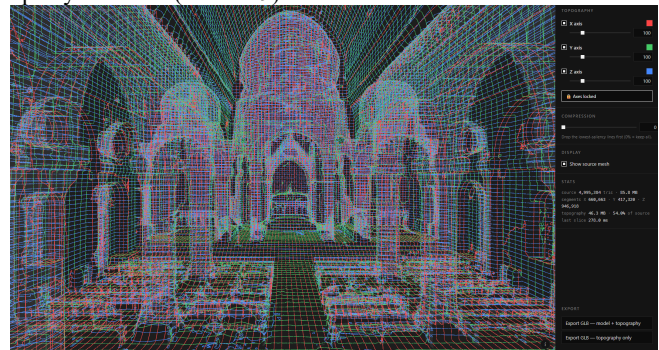


Figure 6. The browser-based reference tool: per-axis density sliders with lock, per-axis colour, a source-mesh toggle, live statistics, and GLB export of the model plus a toggleable Topography layer.

5. Experimental Setup

We evaluated on a laptop with an NVIDIA GeForce RTX 3070 GPU (8 GB VRAM), an 8-core CPU and 32 GB RAM, rendering in a Chromium-based browser on Windows 11. Reporting the GPU and its memory matters because a model that struggles on a laptop GPU

may be fluid on a desktop one; the figures below should be read against this configuration.

Our primary benchmark is a 5-million-triangle photogrammetry model. For the point-cloud baseline, whose full-resolution form could not be loaded for interactive editing, we used a 1-million-triangle equivalent; the wireframe baseline shares the mesh geometry under a different projection, hence reports no separate storage figure. To report variation across model sizes we additionally measured small, medium and large variants (Table 2). All datasets are openly released (Section 9).

We report three quantities. Memory is peak resident GPU buffer usage for the loaded representation. For interactivity we measure average frame time, the mean wall-clock time to redraw the viewport, obtained by timing 100 successive viewport regenerations and dividing by 100; we report it in milliseconds and, equivalently, as frames per second. (This quantity appeared as RVW in the proposal; we define it explicitly here.) Storage is the on-disk size of the encoded asset - Draco-compressed glTF for the mesh, a compressed point cloud for the cloud, and our GLB line export for the proposed method. Because the method targets editing, we also exercised representative operations - cross-section tracing, boundary snapping and point-to-point measurement - at coarse, medium and fine LODs, noting at which density each task becomes reliable.

6. Results and Evaluation

6.1. Resource Use

Table 1 compares the four representations on the benchmark model. Our representation is the lightest on every axis: 320 MB of memory (36% below the mesh and 44% below the point cloud), 23.5 MB on disk (89% below the mesh), and an average frame time of 13 ms. Converting frame time to rate, the point cloud and mesh render at roughly 6 and 14 fps respectively - both below the 30 fps editing threshold of Section 1 - while our method sustains about 76 fps, with the wireframe between at 22 fps. The proposed method is thus the only representation in the comparison that stays comfortably interactive for editing on this hardware.

Table 1. Resource use on the 5M-triangle benchmark (the point cloud was measured on a 1M-triangle equivalent). Frame time is the mean over 100 viewport regenerations; FPS is its reciprocal. Wireframe shares the mesh's storage. These are initial-stage measurements; Table 2 reports later, tool-based figures.

Method	Memory (MB)	Storage (MB)	Frame (ms)	FPS
Point cloud	572	95.9	168	6
Mesh	498	221.0	69	14
Wireframe	435	-	45	22
Our method	320	23.5	13	76

6.2. Scaling With Model Size

Table 2 reports three size variants. Triangle count is an important factor in the size of the generated representation: because the contour lines follow the source mesh, a line must change trajectory

each time it crosses a triangle, which increases the number of vertices and sub-lines that make up each main contour line. This effect is countered by manipulating the level of detail (line density). Table 2. Storage across three model sizes, measured with our interactive web tool; large-scale is defined here by triangle count. Absolute values differ from Table 1, whose earlier measurements were taken during initial testing and are less precise.

Model	Triangles	Mesh storage (MB)	Our storage (MB)
Small	500K	8.6	3.9
Medium	2.0M	34.3	12.8
Large	5.0M	85.6	46.3

6.3. Fidelity and Saliency

Because the representation is deliberately sparse, the question is not whether it reproduces the surface exactly but whether it preserves the features that carry meaning - edges, profiles and reliefs - while discarding only flat, low-information regions. To check this we compare the contour shell against the original surface and measure where the two differ. The largest differences fall in flat areas and in the gaps between slicing planes, whereas sharp features are captured wherever a plane crosses them. To make the measurement reflect what a viewer actually notices, we weight each difference by mesh saliency [7], [8] - a per-region estimate of visual importance - so that an error on a prominent edge counts for more than the same error on a flat wall. On a controlled sample model (a sphere of 18,432 triangles sliced at 64 lines per axis), measured over 50,000 surface samples, the unweighted mean deviation between the contour shell and the surface is 6.95% of the bounding-box diagonal, whereas the saliency-weighted deviation is only 0.16% - a weighted-to-unweighted ratio of 0.02. The largest deviations therefore fall almost entirely in low-saliency, near-planar regions, while the salient features that carry the shape are preserved. The same multi-scale saliency signal (computed in about 1.1 s) also drives the compression of Section 3.5: ranking and thresholding the 54,384 segments at this density removes 61% of them (keeping 21,038) with negligible loss in salient regions. Placing more lines where the surface is most salient is the natural next step (Section 8).

6.4. Editing and CAD Interfacing

Across the operations of Section 5, the contour representation interfaces more directly with CAD and modelling tools than points or dense meshes, because their grid-snapping acts on line endpoints rather than ambiguous surfaces; tracing a section or measuring a span reduces to snapping between existing contour vertices. Coarse LODs suffice for navigation and gross measurement, while fine LODs are needed for tracing tight features - consistent with the LOD-streaming workflow of Section 3.4. In practice the required density depends on the ratio between the working-area mesh and the total mesh visible in the source model: 50 to 140 lines per axis (LPA) suffices for most applications, with some cases requiring more than 200 LPA.

6.5. Viewer Compatibility

Because the export uses only standard glTF line primitives, it renders in any glTF 2.0 viewer with no bespoke decoding path; we

verified this in three.js, Sketchfab, the Babylon.js sandbox, and Blender import. The named Topography node can be hidden in any viewer that exposes a scene tree, and viewers implementing KHR_node_visibility additionally honour the embedded toggle. The practical consequence directly addresses the central adoption concern: the representation lives inside today's web 3D ecosystem rather than alongside it.

7. Limitations

The method has clear failure modes, several inherent to slicing. Features thinner than the slice spacing fall between planes and are missed at low density; raising density or using saliency-adaptive spacing recovers them at the cost of more lines. Geometry that runs parallel to a cutting plane is under-sampled by that axis, though the orthogonal axes usually capture it - the motivation for the multi-axis design and for optional angled sub-axes. On open or non-manifold scan data the contours are correspondingly open, which is faithful but can read as broken loops without the source surface beneath. For a hollow object with a narrow opening it is worth separating two issues: if interior surfaces are present in the mesh, slicing does capture them - the plane intersects every triangle it crosses, producing nested interior and exterior contours - so the limitation is upstream, in whether photogrammetry could see through the opening to reconstruct the interior at all; and where interiors are present, the overlapping nested contours can themselves be visually ambiguous unless the source mesh is shown for context. Finally, the current method slices a static bind pose and does not address skinned or animated meshes.

8. Future Work

Several extensions follow directly. Saliency-adaptive density would place lines where the surface is informative rather than uniformly, improving the fidelity-per-line trade-off and tightening the compression of Section 3.5; we are presently assessing how best to integrate a saliency estimator into the interactive tool. Angled sub-axes would capture features poorly aligned with the principal axes. A UsdGeomBasisCurves exporter would extend the format story to USDZ and Apple's ecosystem. Finally, carrying per-line attributes such as source colour or material identity would let the contour shell stand in for the textured model in a wider range of tasks.

9. Conclusions

We presented variable-axes topographic maps, a representation that replaces the dense surface or point cloud with a sparse, boundary-explicit shell of contour lines sliced along each principal axis. The representation is cheap to draw, naturally multi-resolution, compressible by perceptual saliency, and - importantly for adoption - expressible entirely within standard glTF/GLB, so existing web viewers and CAD tools handle it without modification. On a 5-million-triangle heritage model it cut memory by 36% and disk size by 89% relative to the source mesh while sustaining interactive frame rates where the mesh and point cloud could not, and it offers a more determinate, snappable target for HBIM editing. By keeping the entire path client-side and web-native, it also lowers the bandwidth barrier to working with heritage models in low-connectivity settings. Our datasets and a browser-based reference implementation are openly available at <https://github.com/alqaba/variable-axes-topography>.

References

- [1] G. J. Verhoeven, "Mesh is more - using all geometric dimensions for the archaeological analysis and interpretative mapping of 3D surfaces," *Journal of Archaeological Method and Theory*, vol. 24, no. 4, pp. 999-1033, 2017.
- [2] L. Linsen, "Point cloud representation," Tech. Report, Univ. of Karlsruhe, 2001.
- [3] F. Remondino, "From point cloud to surface: the modeling and visualization problem," *Int. Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 34, no. 5/W10, 2003.
- [4] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proc. SIGGRAPH*, 1997, pp. 209-216.
- [5] H. Hoppe, "Progressive meshes," in *Proc. SIGGRAPH*, 1996, pp. 99-108.
- [6] M. Schuetz, "Potree: Rendering large point clouds in web browsers," M.S. thesis, TU Wien, 2016.
- [7] C. H. Lee, A. Varshney and D. W. Jacobs, "Mesh saliency," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 659-666, 2005.
- [8] O. Lezoray and A. Nouri, "3D mesh saliency from local spiral hop descriptors," in *IS&T Electronic Imaging*, vol. 37, no. 10, Society for Imaging Science and Technology, 2023.
- [9] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Proc. SIGGRAPH*, 1987, pp. 163-169.
- [10] Khronos Group, "glTF 2.0 Specification," 2021. [Online]. Available: <https://www.khronos.org/glTF/>
- [11] Khronos Group, "KHR_materials_unlit glTF extension," 2019.
- [12] Khronos Group, "KHR_node_visibility glTF extension," 2024.
- [13] Google, "Draco 3D data compression." [Online]. Available: <https://google.github.io/draco/>
- [14] A. Kapoulkine, "meshoptimizer and EXT_meshopt_compression." [Online]. Available: <https://github.com/zeux/meshoptimizer>
- [15] Alliance for OpenUSD, "OpenUSD: UsdGeomBasisCurves." [Online]. Available: <https://openusd.org/>
- [16] M. Murphy, E. McGovern and S. Pavia, "Historic building information modelling (HBIM)," *Structural Survey*, vol. 27, no. 4, pp. 311-327, 2009.

Author Biography

Hussein Yaseen is a senior architectural engineering student (B.Eng.) at the University of Mosul and researcher experienced in architectural design and 3D reconstruction. His research interests focus on cultural heritage preservation and reconstruction.

Adil Al Qaba is a software engineer currently doing a Computer Engineering (B.Sc.) at the University of Mosul. He is the Head of Development at the Alnoor Center for the Preservation of Cultural Heritage. His research interests focus on heritage preservation and robotics.

Basim Mahmood received his Ph.D. degree in Computer Science from Florida Institute of Technology, USA, in 2015. He is currently the head of the Computer Science Department at the University of Mosul. His research interests focus on Complex Networks and Systems. He is also interested in the area of digitalizing and preserving cultural heritage.

Zohair Al-Ameen received his Ph.D. degree in Computer Science from the Technological University of Malaysia in 2015. Since 2016, he has been a tenured faculty member at the University of Mosul, where he is engaged in teaching and research. His research interests focus on developing low-complexity algorithms that produce high-quality results for digital image processing, with specific emphasis on image enhancement and restoration.

Julia Schnitzer is Professor of Interaction Design in the Department of Computer Science and Media at Brandenburg University. Her research focuses on user-centered design of interactive environments. As part of the UNESCO Chair on the Digitalization of Cultural Heritage in MENA Regions at Risk, she has explored efficient low-cost photogrammetry workflows. Since March 2026, she has served as Vice President for Education and Internationalization.