An Open-Source Pipeline for Runtime-Optimized Heritage Photogrammetry in Game Engines

Arkoun Merchant [1], Adam Weigert[1], Chloe Dennis [2], Stephen Fai[1]

[1]Carleton Immersive Media Studio, Carleton University, 1125 Colonel By Dr, Ottawa, Canada – (amerchant, aweigert, sfai)@cims.carleton.ca

[2]Bytown Museum, Ottawa, Canada - chloedennis@bytownmuseum.ca

Keywords: Photogrammetry, 3D Tiles, Cultural Heritage, Adaptive Tiling, Octree Subdivision, Texture Optimization

Abstract

This paper presents Mesh2Tile, an open-source pipeline that converts photogrammetric meshes into runtime-optimized 3D Tiles for interactive visualization in game engines. Photogrammetry produces high-polygon meshes that remain difficult to deliver at scale through interactive platforms. Cloud-based conversion services like Cesium Ion provide a path to the OGC 3D Tiles format but impose cost barriers and raise data sovereignty concerns for confidential heritage projects. Existing open-source converters rely on uniform spatial partitioning, export redundant textures with every tile, and offer limited control over LOD generation. Mesh2Tile leverages Blender's Python API to perform adaptive octree tiling driven by triangle density, per-tile texture baking that eliminates texture redundancy, and parallel processing to generate georeferenced 3D Tiles from OBJ meshes. The pipeline is validated through a case study of the Bytown Museum Commissariat Building on the Rideau Canal UNESCO World Heritage Site. It is processed at three scales from 900 thousand to 90 million triangles. Results demonstrate linear scaling of processing time, up to 62% file size reduction for larger models, and successful runtime streaming in Unreal Engine 5 through the Cesium for Unreal plugin at 120 FPS with comparable tile balance to Cesium Ion's commercial output. The pipeline enables institutions to maintain full control over sensitive heritage data while achieving performance suitable for interactive visualization.

1. Introduction

Photogrammetry has become integral to creating high-quality 3D models from real-world assets for use in game engines. While conventional workflows required optimizations to achieve acceptable performance, recent advancements have provided alternative approaches. Unreal Engine 5's Nanite system (Karis et al., 2021) can virtualize geometry at original resolution through cluster-based hierarchical LOD and GPU-driven rendering (Drago Diaz Aleman et al., 2023). The one drawback is that Nanite operates on static geometry loaded into memory. It does not support dynamic streaming from external sources at the scales required by Architectural, Engineering and Construction (AEC) applications, where datasets regularly reach 100 GB to multi-terabyte sizes.

Open standards such as OGC 3D Tiles (OGC, 2019) offer an alternative approach through strategic spatial partitioning and dynamic Level of Detail (LOD) streaming. The 3D Tiles format has enabled runtime streaming of large geospatial datasets into game engines though open-source plugins like Cesium for Unreal. However, generating these tilesets typically requires proprietary cloud-based conversion services like Cesium Ion. This imposes significant cost barriers and introduces security concerns for confidential heritage documentation projects (Hummel et al., 2021; Kukutai and Taylor, 2016). Institutional stakeholders increasingly require that sensitive cultural data remain under their direct control through all processing stages.

Several open-source tools exist for generating 3D Tiles from mesh data. Py3Dtilers (Marnat et al., 2022) converts CityGML, IFC, and OBJ to 3D Tiles but is primarily designed for city-scale semantic models rather than photogrammetric meshes. Obj2Tiles from the OpenDroneMap project performs OBJ splitting and decimation but relies on uniform grid subdivision. The objTo3d-tiles Node.js library converts OBJ to B3DM format but does not generate multi-LOD hierarchies. These tools share common shortcomings: uniform spatial partitioning that ignores geometric density, and texture redundancy that inflates output size.

This research presents Mesh2Tile, an open-source pipeline leveraging Blender’s Python API (Chlubna et al., 2025) to generate runtime-optimized, georeferenced 3D Tiles from photogrammetric meshes. The contributions are: (1) adaptive octree tiling driven by triangle density rather than uniform subdivision, (2) per-tile texture baking that eliminates texture redundancy, (3) parallel processing across tiling, baking, and conversion stages, and (4) validation of the complete workflow from OBJ input through 3D Tiles streaming in Unreal Engine 5.

2. Related Work

2.1 Photogrammetry for Heritage Documentation

Modern SfM-MVS pipelines produce dense, textured meshes that are well-suited for documentation but poorly suited for real-time rendering (Pepe et al., 2022; Skublewska-Paszkowska et al., 2022). Commercial suites such as Agisoft Metashape (Over et al., 2021) and RealityCapture (Croce et al., 2024) output high-resolution models as monolithic files that must be optimized before interactive use.

Game engines have been adopted to bridge this gap across a range of heritage applications, from virtual museum experiences (Kersten et al., 2017) and SfM-to-VR workflows (Dhanda et al., 2019) to oblique photogrammetry visualization with quadtree LOD scheduling (Huo et al., 2021). Nanite in UE5 reduces the manual optimization burden for static scenes (Drago Diaz Aleman et al., 2023; Karis et al., 2021), but does not address streaming datasets that exceed available memory or georeferenced model placement.

2.2 3D Tiles and Spatial Data Streaming

The OGC 3D Tiles specification (OGC, 2019) defines a hierarchical format for streaming massive 3D geospatial datasets. Each tileset consists of a JSON manifest describing a spatial hierarchy where tiles contain renderable GLB content with bounding volumes and geometric error metrics. The runtime viewer traverses this hierarchy, loading tiles based on camera position and screen-space error.

Conversion pipelines have been developed for specific source formats, including IFC for BIM visualization (Chen et al., 2018), laser-scanned point clouds via octree partitioning (Hillmann et al., 2022), and multi-LOD CityGML for urban models (Buyukdemircioglu and Kocaman, 2020). Velayudhan et al. (2025) addressed georeferencing and tileset metadata generation for pre-modeled building assets but without spatial partitioning or LOD generation, which remain listed as future work. Game engine consumption of these formats has been evaluated for both Unreal Engine and Unity (Wurstle et al., 2022) and applied to urban digital twin applications (Rantanen et al., 2023). However, no published pipeline addresses converting photogrammetric OBJ meshes to 3D Tiles with adaptive spatial partitioning and texture optimization through a fully open-source workflow.

Mesh decimation based on Quadric Error Metrics (Garland and Heckbert, 1997) remains foundational to most simplification approaches (Luebke et al., 2003). For tiled delivery, texture handling presents a distinct challenge: full-texture spatial partitioning exports the full source texture with every tile, creating substantial redundancy. Zhang et al. (2021) proposed size-adaptive texture atlas generation for urban building models, and Potenziani et al. (2015) demonstrated multi-resolution streaming through 3DHOP with progressive encoding. The texture baking approach in the present work differs fundamentally from atlas repacking: it generates new per-tile textures through UV-based rendering, eliminating source texture dependencies entirely.

3. Methodology

The Mesh2Tile pipeline transforms large photogrammetric OBJ meshes into spatially partitioned, multi-LOD 3D Tilesets through five sequential stages (Figure 1). The pipeline is implemented in python, orchestrating Blender’s python API (bpy) for geometry operations and Node.js tooling for tileset manifest generation. The source code is publicly available under an open-source license.

3.1 Adaptive Octree Tiling

The first stage spatially partitions the input mesh using adaptive octree subdivision. Unlike uniform grid approaches, subdivision depth at each node is determined by local triangle density rather than a fixed grid resolution.

The algorithm operates recursively. If the triangle count in a region falls below a configurable threshold (default: 20,000 triangles), the region is exported as a leaf tile. If the count exceeds the threshold and the current depth has not reached the maximum level (default: 3), the mesh is subdivided into eight octants by bisecting the axis-aligned bounding box along all three principal axes. Faces are assigned to octants based on centroid position.

At each level, a decimated version of the region is also exported as the parent LOD tile using Blender's collapse-based Decimate modifier, reducing the triangle count to the threshold value. This ensures each parent tile provides a simplified representation of its children's combined geometry for progressive refinement at runtime.

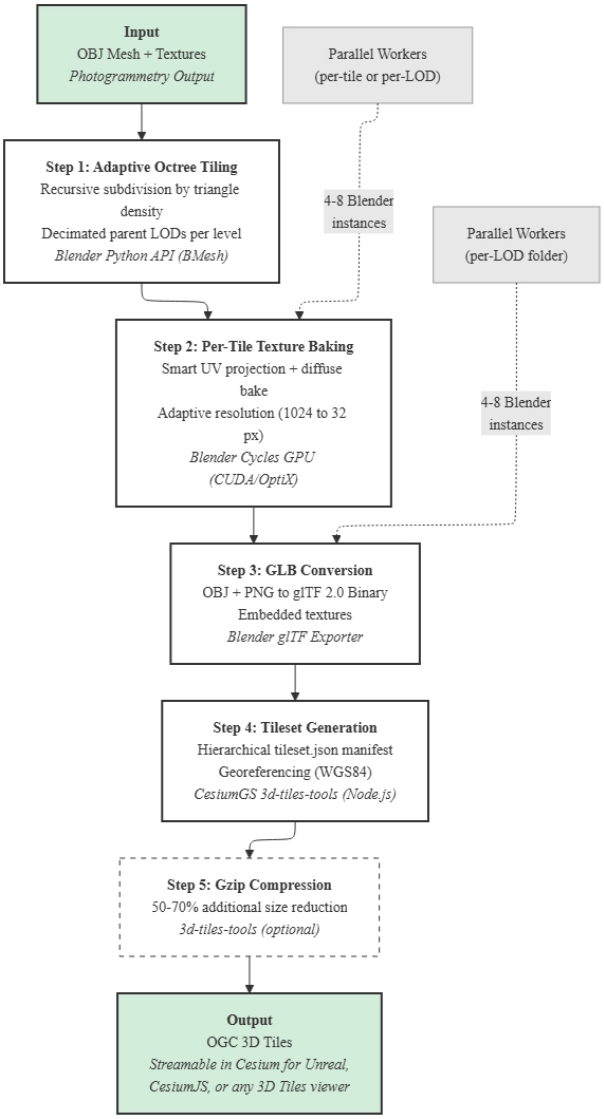

Figure 1. Mesh2Tile pipeline workflow. Input OBJ meshes are processed through five stages using Blender's Python API and CesiumGS 3d-tiles-tools. Texture baking and GLB conversion stages support parallel execution across multiple Blender instances.

The spatial partitioning uses Blender's BMesh API for direct mesh manipulation. The octree bisection operates in a single pass through all faces, assigning each to its corresponding octant based on centroid coordinates, avoiding the cost of duplicating the mesh eight times and performing boolean cuts. This achieves a 3-5x speedup over iterative copying methods. UV coordinates are preserved during subdivision through explicit per-loop UV copying to maintain texture mapping fidelity.

The tile naming convention encodes the spatial hierarchy: each tile is named *level_ix_iy_iz* where *level* indicates octree depth and *ix*, *iy*, *iz* encode spatial position. Child indices relate to parent indices through the formula: $child = parent * 2 + offset$ where offset is 0 or 1 for each axis.

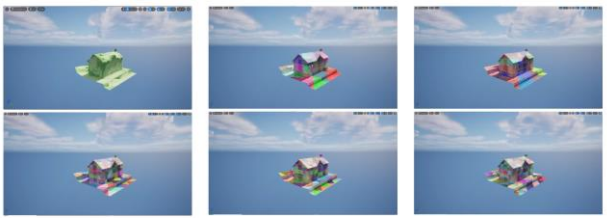

Figure 2. Adaptive octree tile hierarchy on the 90 million triangle model, rendered in Unreal Engine 5 with Cesium tile debug mode. From LOD 0 (top left, single decimated root tile) through progressive subdivision to LOD 5 (bottom right), each colour represents an individual tile. Subdivision concentrates tiles around areas of geometric complexity.

3.2 Per-Tile Texture Baking

The second stage addresses texture redundancy. When a photogrammetric mesh with a large source texture (commonly 8,192x8,192 or 16,384x16,384 pixels) is spatially partitioned, each tile retains a reference to the full source texture despite using only a small fraction of its texel data (Figure 3). At deeper LOD levels, individual tiles may reference less than 1% of the source texture area.

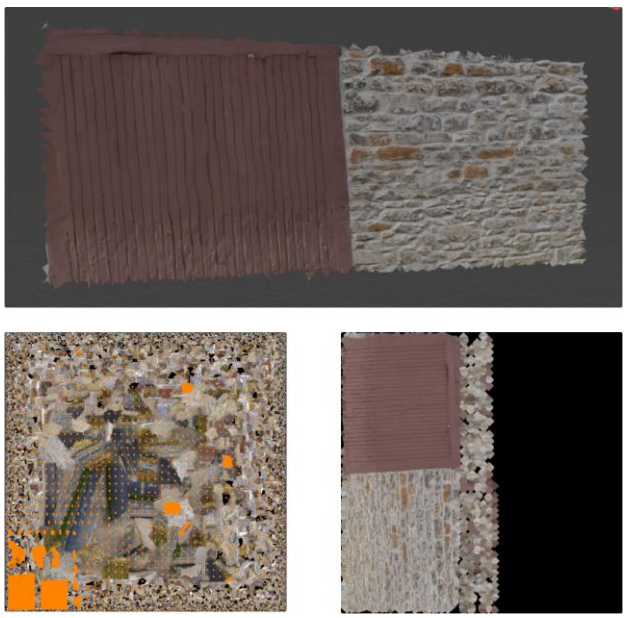

Figure 3. Texture handling comparison. Top: The source tile. Bottom left: the same tile's geometry showing the spatial region covered across the 16,384x16,384 pixels. Bottom right: Mesh2Tile per-tile baked texture (1,024x1,024 pixels) containing only the texel data relevant to that tile's geometry

The pipeline resolves this through UV-based texture baking. For each tile, the process duplicates the geometry, generates a new UV layout using Blender's Smart UV Projection, creates a texture image at an adaptive resolution, and bakes the diffuse colour from the original UV-mapped geometry onto the new texture. The result is a self-contained texture per tile at a fraction of the original size.

Texture resolution adapts per LOD level using a budget-based allocation. The algorithm accumulates total texture area across tiles at each depth level (accounting for the octree branching factor of 8) until cumulative texel count exceeds the source texture area. Tiles at or below this budget exhaustion level receive the base resolution (1,024 x 1,024 pixels); tiles beyond receive progressively reduced resolutions scaled by sqrt(8) per level, clamped to powers of two between 32 and 1,024 pixels.

Baking uses Blender's Cycles engine with GPU acceleration (CUDA, OptiX, or HIP). Cage extrusion, UV margin, and sampling parameters scale with target resolution to balance quality against processing time. Denoising compensates for reduced sample counts.

3.3 GLB Conversion

Each baked OBJ tile is converted to glTF 2.0 Binary (GLB) format using Blender's native glTF exporter. GLB is the required content format for 3D Tiles 1.1 and provides efficient binary packing of geometry, materials, and embedded textures in a single file. The export preserves UV coordinates and normals while converting Blender's Principled BSDF material to glTF PBR representation.

This stage replaced an earlier approach using the npm-based obj23dtiles library (B3DM output), which suffered cache contention when running multiple conversion processes in parallel.

3.4 Tileset Manifest Generation

The fourth stage generates the “tileset.json” manifest using the CesiumGS 3d-tiles-tools npm package to compute bounding volumes and assign geographic coordinates. The pipeline then restructures the flat tile list into a hierarchical tree reflecting octree spatial relationships.

The restructuring algorithm parses tile indices from filenames, groups tiles by level, and recursively builds parent-child relationships using the octree index formula. A structural root nod is created with the LOD 0 tile as its single child. Geometric error for each tile is calculated from its bounding volume as max_half_extent / 8, so LOD transitions adapt to actual tile size rather than relying on fixed constants. The root node applies a 16x multiplier to ensure visibility at distant camera positions. The refinement strategy is set to REPLACE, where the runtime viewer substitutes parent tiles with children rather than additively loading both.

Geographic positioning uses WGS84 coordinates (longitude, latitude, height) to place the tileset on a virtual globe, enabling combination with other geospatial datasets. An optional gzip compression pass achieves 50-70% additional file size reduction for network delivery; the Cesium runtime transparently decompresses gzipped tiles during streaming.

3.5 Parallel Processing

The pipeline uses Python's ProcessPoolExecutor for parallel execution across computationally intensive stages. Texture baking (49-72% of total conversion time) supports two modes: a default per-tile mode that distributes individual tiles across worker processes, each spawning an independent Blender instance, and an alternative batch mode that assigns entire LOD folders to workers to reduce process startup overhead. GLB conversion is similarly parallelized across LOD folders.

An optional parallel tiling mode performs the first octree split sequentially to produce eight initial chunks, then distributes recursive subdivision of each chunk across workers. This benefits

models exceeding one million triangles where subdivision time justifies per-process startup overhead.

For models that exceed available system memory (typically above 20 million triangles on a 64 GB workstation), the pipeline includes an out-of-core pre-splitting stage. A streaming Python parser reads the OBJ file line-by-line, computes the bounding box from vertex positions, and spatially bisects the mesh along the longest axis by assigning faces to halves based on centroid position. This process recurses until each chunk falls below a configurable face threshold. Vertex indices, texture coordinates, and normals are remapped per chunk to produce valid standalone OBJ files. Each chunk is then processed through the full pipeline independently, and the resulting tilesets are merged into a single unified tileset.json. This method produces more tiles than the unified octree approach. Each chunk generates its own parent LOD tiles independently, introducing redundancy across chunk boundaries.

4. Case Study and Workflow

4.1 The Commissariat Building

To illustrate the results of this research, this paper uses the case study of Bytown Museum’s Commissariat Building. It was built in 1827 and is the oldest stone building built in Canada’s national capital, Ottawa. Constructed as part of the Rideau Canal, a UNESCO World Heritage Site, it has become home to the Bytown Museum. Given its prominence in Ottawa’s history and cultural heritage, it has been documented multiple times with the most recent documentation in 2024 resulting in a high-resolution photogrammetry mesh of the building’s exterior.

4.2 Baseline Comparison

Before developing Mesh2Tile, both proprietary and open-source conversion tools were tested against a Cesium Ion baseline. Agisoft Metashape and Safe FME produced tiled outputs but skipped spatial partitioning and multi-LOD generation, loading as monolithic assets without progressive refinement.

Among open-source tools, Obj2Tiles, objTo3d-tiles, and gltf-3dtiles were tested. All exhibited uniform grid subdivision that created tiles with highly uneven triangle counts and texture redundancy where each tile exported the full source texture (up to 16,384x16,384 pixels) despite using only a fraction of its area. This redundancy inflated output sizes well beyond the original input.

Table 1 compares Mesh2Tile against Obj2Tiles v1.0.13 and Cesium Ion on the 900K and 9M models. Obj2Tiles is the only open-source tool that performs spatial partitioning of a single photogrammetric mesh; Py3DTilers treats a monolithic OBJ as one indivisible feature, and objTo3d-tiles is a format converter with no tiling.

Cesium Ion produces the smallest output through Draco mesh compression and KTX2 texture encoding, neither of which Mesh2Tile currently employs. Obj2Tiles uses ADD refinement, retaining parent geometry alongside children, which inflates total triangle count (1.9M vs 900K source for the 900K model, 18.2M vs 9M source for the 9M model). Mesh2Tile's REPLACE strategy produces leaner tilesets (1.5M and 15.0M total triangles respectively).

Metric			
900k Model	Mesh2Tile	Obj2Tile	Cesium Ion
Output size	223 MB	195 MB	71.5 MB
Tile count	168	48	336
LOD depth	4	3	6
Processing time	192s	253s	183s
Refinement	REPLACE	ADD	REPLACE
9M Model	Mesh2Tile	Obj2Tile	Cesium Ion
Output size	747 MB	580 MB	109 MB
Tile count	1,997	48	663
LOD depth	5	3	7
Processing time	2,378s	2,773s	~257s
Refinement	REPLACE	ADD	REPLACE

Table 1. Pipeline comparison on the Commissariat Building at two scales. All local tools benchmarked on an Intel i9-14900F, RTX 4080 SUPER, 64 GB RAM. Cesium Ion timings are server-side.

5. Results

5.1 Processing Performance

Performance was validated across three versions of the Commissariat Building model: 900 thousand triangles (91 MB input), 9 million triangles (964 MB input), and 90 million triangles (10.0 GB input). Processing ran on an Intel i9-14900F workstation with an NVIDIA RTX 4080 SUPER (16 GB), 64 GB RAM, and 8 parallel workers.

Model tri count	900k	9 million	90 million
Input Size	124 MB	1.0 GB	10.0 GB
Output Size	223 MB	747 MB	3.81 GB
Size Reduction	+80 %	-25%	-62%
No. Of Tiles	174	1,997	13,078
LOD levels	4	5	7
Time to Convert	192s	2,378s	26,617.05s
Adaptive Tiling	35s	570s	9,995.97s
Texture Baking	146s	1,722s	14,417.27s
GLB Conversion	8s	63s	1,216.07s
Tileset Generation	3s	20s	333.63s

Table 2. Pipeline performance across three model scales of the Commissariat Building. The 90M model used out-of-core pre-splitting (11 chunks at 20M face threshold); the tiling time includes the pre-split stage. Baking, conversion, and tileset times for 90M are aggregated across chunks.

Processing times scaled approximately linearly with input size, from 3.2 minutes for the 900K model to 40 minutes for the 9M model. Texture baking dominated at all scales, consuming 72-76% of total time. The number of LOD levels increased with model complexity: the 900K model required 4 levels, the 9M model 5 levels, and the 90M model 7 levels, with the adaptive octree automatically determining subdivision depth based on local triangle density.

The 900K produced larger output +80% because per-tile metadata and minimum-resolution textures (1,024x1,024 per tile)

exceed the compression gains when tile counts are high relative to input size. At the 9M and 90M scale, the adaptive texture resolution scaling takes full effect: deeper LOD tiles receive progressively smaller textures, producing upto 62% size reduction despite 13,078 tiles.

5.2 Texture Efficiency

To quantify the texture baking advantage, we compared Mesh2Tile's per-tile baked textures against the full-texture approach used by Obj2Tiles, which embeds the complete source texture with every tile. Obj2Tiles produced 48 tiles at 580 MB for the 9M model, embedding the full 61 MB source texture in each B3DM tile. At Mesh2Tile's tile count of 1,997, this full-texture approach would produce approximately 122 GB of texture data. Mesh2Tile's per-tile baking produces textures embedded within the 747 MB total output, eliminating this redundancy entirely.

5.3 Adaptive Tiling Characteristics

The adaptive octree generated tile hierarchies with depth proportional to model complexity: the 900K model produced 168 tiles across 4 LOD levels (0-3), the 9M model produced 1,997 tiles across 5 levels (0-4), and the 90M model produced 13,078 tiles across 7 levels (0-6). Empty octants were automatically pruned, avoiding tile generation in regions with no geometry.

Table 3 compares the triangle distribution per tile between Mesh2Tile, Obj2Tiles, and Cesium Ion on the 9M model. Tile balance is critical for runtime performance: highly uneven tiles force the GPU to render excessive geometry for some tiles while others contribute little visual information.

Stat	Mesh2Tile	Obj2Tiles	Cesium Ion
Min	1	207	6
Max	138,565	1,391,958	64,641
Mean	7,434	379,711	16,676
Median	5,174	223,934	12,726
Std.Dev	7,559	382,651	14,574
Total	14,964,557	18,226,119	11,056,079

Table 3. Triangle distribution per tile on the 9M model. Lower standard deviation (Std.Dev) indicates more balanced tiles.

Mesh2Tile's standard deviation (7,559) is approximately half of Cesium Ion's (14,574) and 51x lower than Obj2Tiles' (382,651). Obj2Tiles' largest tile contains 1.4 million triangles, which contributed to the 72 FPS at close range and 21.8 GB memory usage observed in runtime testing (Table 4). The Obj2Tiles total triangle count (18.2M) is 2.0x the source due to ADD refinement retaining geometry at all LOD levels, while Mesh2Tile's REPLACE strategy with deeper LOD subdivision produces a total of 15.0M triangles across 1,997 tiles.

The triangle threshold of 20,000 per tile balances visual quality against tile count. At this threshold, each tile contains sufficient geometric detail for close-range inspection while remaining within the performance budget of typical consumer GPUs

5.4 Runtime Validation in Unreal Engine

We loaded the generated tilesets into Unreal Engine 5.7 using the Cesium for Unreal plugin. The LOD hierarchy functioned as intended: at distant views, the engine loaded only the root tile (LOD 0, 20,000 triangles). As the camera approached, higher-detail child tiles progressively replaced their parents using the REPLACE refinement strategy, avoiding the memory overhead of retaining both parent and child tiles simultaneously.

Table 4 compares runtime performance across all three tools on the 9M model, loaded from local files on the same hardware (RTX 4080 SUPER, render resolution 1424x876).

Distance	Metric	Mesh2Tile	Obj2Tiles	Cesium Ion
Far	FPS	120	100	120
	GPU Time	4.75 ms	8.6 ms	4.95 ms
	Triangles Drawn	392K	17.5M	139K
	Memory	8.1 GB	21.8 GB	8.1 GB
Mid	FPS	120	90	120
	GPU Time	5.23 ms	9.7 ms	5.1 ms
	Triangles Drawn	1.67M	17.5M	665K
Close	FPS	120	72	120
	GPU Time	6.26 ms	14.0 ms	5.6 ms
	Triangles Drawn	4.0M	17.5M	3.0M

Table 4. Runtime streaming performance in UE5 via Cesium for Unreal, 9M model (1,997 tiles, 5 LOD levels). Obj2Tiles uses ADD refinement, rendering all 17.5M triangles at every distance with no LOD culling, 2.7x the memory usage, and declining FPS at closer distances despite constant triangle count (increased per-pixel overdraw from overlapping LOD layers). Mesh2Tile and Cesium Ion both used REPLACE refinement with distance-dependent LOD selection.

Figure 4. 3D Tiles output from the Mesh2Tile pipeline loaded in Unreal Engine 5 via Cesium for Unreal. Top row: lit shading mode at 900K (left), 9 million (centre), and 90 million (right) triangles. Bottom row: Cesium colorized tile debug mode at the same three scales, where each colour represents an individual tile.

Mesh2Tile and Cesium Ion both maintained 120 FPS at all distances using REPLACE refinement, with triangle counts scaling appropriately by distance. Obj2Tiles' ADD refinement produced a fundamentally different runtime profile: all 17.5 million triangles rendered at every distance regardless of camera position, consuming 21.8 GB of GPU memory (2.7x more than Mesh2Tile or Cesium Ion) and dropping to 72 FPS at close range. This demonstrates the practical importance of refinement strategy for runtime streaming performance.

Mesh2Tile rendered more triangles than Cesium Ion at comparable distances (2.8x at far, 1.3x at close) due to less aggressive LOD decimation and absence of Draco mesh

compression (Figure 4). Despite this, GPU time remained within interactive budgets (4.75 vs 4.95 ms at far, 6.26 vs 5.6 ms at close). Mesh2Tile's Blender Decimate modifier introduced slight geometric warping at lower LOD levels compared to Cesium Ion's cleaner decimation, suggesting that alternative simplification algorithms could improve visual quality at aggressive reduction ratios. Georeferencing placed the model at its real-world location alongside terrain, satellite imagery, and additional 3D Tiles from other sources.

5.5 Comparison with Cesium Ion

Metric	Mesh2Tile	Cesium Ion
900K Model		
Output Size Tile Count Processing Time	223 MB 168 192s	71.5 MB 336 183s
9M Model		
Output Size Tile Count Processing Time	747 MB 1,997 2,378s	109 MB 663 257s
90M Model		
Output Size Tile Count Processing Time	3.81 GB 13,078 192s	Failed Failed Failed

Table 5. Mesh2Tile vs. Cesium Ion across three model scales. Cesium Ion timings include upload, conversion, and geolocation.

Both pipelines produced functional tilesets that streamed correctly in Unreal Engine at the 900K and 9M scales. Cesium Ion's output is 3-6x smaller due to Draco mesh compression and KTX2/Basis Universal GPU-compressed textures, neither of which Mesh2Tile currently employs. These are additive post-processing techniques orthogonal to the tiling pipeline and are planned for future integration.

Cesium Ion failed to process the 90M model (10 GB OBJ), encountering a pipeline error after 28 minutes and 51 seconds. Mesh2Tile completed the same model in 2.9 hours using out-of-core pre-splitting on a single workstation with 64 GB RAM (Table 5). This demonstrates the pipeline's scalability advantage for large heritage datasets that exceed cloud service limits.

6. Discussion

6.1 Tile Balance and Spatial Efficiency

The adaptive octree approach concentrates subdivision around regions of geometric complexity rather than applying a fixed grid. In heritage photogrammetry models, detail clusters around architectural features such as cornices, window frames, and ornamental stonework, while flat wall surfaces contain sparse geometry.

The quantitative impact is visible in Table 3: Mesh2Tile's tile balance (Std.Dev 7,559) is approximately half of Cesium Ion's commercial output (Std.Dev 14,574) and 51x lower than Obj2Tiles' uniform grid (Std.Dev 382,651). Obj2Tiles produces tiles varying from 207 to 1.4 million triangles; a single 1.4M triangle tile negates the benefit of tiling altogether as the viewer must allocate GPU resources for the largest tile in view.

6.2 Texture Redundancy at Scale

The texture baking approach eliminates a problem that compounds with tile count. For the 9M model split into 1,997 tiles, full-texture export of the full 16,384x16,384 source texture (61 MB JPG) with every tile would produce approximately 122 GB of texture data. Mesh2Tile's per-tile baking produces textures embedded within the 747 MB total output, a 163x reduction in texture data.

The adaptive resolution scaling drives the largest savings at deeper LOD levels, which contain the most tiles but represent the smallest spatial regions. Figure 5 compares the highest LOD output of Mesh2Tiles against Obj2Tiles and Cesium Ion.

Figure 5. Close-up visual fidelity comparison on the 9M model. Left: Mesh2Tile (per-tile baked textures, 1,024 px). Centre: Obj2Tiles (full 16K source texture). Right: Cesium Ion (Draco + KTX2 compression). Obj2Tiles retains the sharpest texture at the cost of higher output size and memory. Cesium Ion shows reduced resolution from mesh and texture compression.

A limitation of the baking approach is that it only preserves diffuse colour. PBR material properties (roughness, metallic) present in some photogrammetry outputs are not carried through. Future work will extend the pipeline to bake additional material channels.

6.3 Limitations

The current pipeline has several limitations. First, it processes each input OBJ independently; there is no support for combining multiple models into a single tileset hierarchy. Second, geographic positioning uses a single coordinate for the entire tileset rather than deriving positions from survey control points embedded in the photogrammetry. Third, the pipeline requires manual configuration of the Blender executable path and triangle threshold parameters. Fourth, large models (90M+ triangles) require substantial processing time (7+ hours) on a single workstation, though the parallel architecture scales with available CPU cores and GPU resources.

7. Conclusions

This paper presented Mesh2Tile, an open-source pipeline for converting photogrammetric OBJ meshes into runtime-optimized 3D Tiles for streaming in game engines. The pipeline addresses gaps in existing workflows: cloud-based converters impose cost and data sovereignty constraints, while existing open-source tools rely on uniform tiling and export redundant textures with every tile. No prior work has validated an end-to-end open-source workflow from photogrammetry output to game engine visualization with adaptive spatial partitioning.

Adaptive octree tiling driven by triangle density produces balanced tile hierarchies and is well suited to the irregular geometry distribution found with heritage photogrammetry. This achieved a tile balance comparable to Cesium Ion's commercial output while outperforming the leading open-source alternative Obj2Tiles. Per-tile texture baking eliminates the redundancy observed in existing converters, achieving up to 62% file size reduction for the largest model while trading output size for deeper LOD hierarchies at smaller scales. The pipeline successfully processed a 90 million triangle model (10 GB) on a 64 GB workstation where Cesium Ion's cloud service failed. Validation in Unreal Engine 5 via Cesium for Unreal confirmed correct LOD streaming at 120 FPS across all camera distances on consumer hardware, with triangle counts within 1.3-2.8x of Cesium Ion at comparable viewing distances.

The pipeline is fully open-source and runs on local hardware using Python, Blender, and Node.js. It enables heritage institutions to maintain data sovereignty while producing 3D Tiles compatible with any viewer supporting the OGC standard.

Future work will investigate Draco mesh compression and KTX2/Basis Universal GPU-compressed textures for further size reduction. Benchmarking against Cesium Ion suggests these two techniques account for much of the remaining output size gap. Additional directions include multi-channel PBR texture baking, implicit tiling specification support for reduced manifest size, and Docker containerization for reproducible deployment.

Acknowledgements

The authors acknowledge the Bytown Museum for providing access to the Commissariat Building for photogrammetric documentation. This research was supported by the Carleton Immersive Media Studio (CIMS) at Carleton University.

References

Buyukdemircioglu, M., Kocaman, S., 2020. Reconstruction and Efficient Visualization of Heterogeneous 3D City Models. *Remote Sensing*, 12(13), 2128. Doi.org/10.3390/rs12132128.

Chen, Y., Shooraj, E., Rajabifard, A., Sabri, S., 2018. From IFC to 3D Tiles: An Integrated Open-Source Solution for Visualising BIMs on Cesium. *ISPRS Int. J. Geo-Inf.*, 7(10), 393. Doi.org/10.3390/ijgi7100393.

Chlubna, T., et al., 2025. Survey of FOSS 3D/2D Graphics Software Blender Usage in Science, Academia, and Industry. *The Visual Computer*, Springer. doi.org/10.1007/s00371-025-04281-1.

Croce, V., Billi, D., Caroti, G., Piemonte, A., De Luca, L., Veron, P., 2024. Comparative Assessment of Neural Radiance Fields and Photogrammetry in Digital Heritage: Impact of Varying Image Conditions on 3D Reconstruction. *Remote Sensing*, 16(2), 301. Doi.org/10.3390/rs16020301.

Dhanda, A., Reina Ortiz, M., Weigert, A., Paladini, A., Min, A., Gyi, M., Su, S., Fai, S., Santana Quintero, M., 2019. Recreating Cultural Heritage Environments for VR Using Photogrammetry. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, XLII-2/W9, 305-310. doi.org/10.5194/isprs-archives-xlii-2-w9-305-2019.

Drago Diaz Aleman, M., Amador-Garcia, E.M., Diaz Gonzalez, E., de la Torre-Cantero, J., 2023. Nanite as a Disruptive Technology for the Interactive Visualisation of Cultural Heritage 3D Models. *Heritage*, 6(8), 5607-5618. Doi.org/10.3390/heritage6080295.

Garland, M., Heckbert, P.S., 1997. Surface Simplification Using Quadric Error Metrics. *Proceedings of SIGGRAPH '97*, 209-216. Doi.org/10.1145/258734.258849.

Hillmann, M., Igelbrink, F., Wiemann, T., 2022. Computing Arbitrarily Large Meshes with Level-of-Detail Support for Cesium 3D Tiles. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, XLVIII-2/W1-2022, 109-116. doi.org/10.5194/isprs-archives-xlviii-2-w1-2022-109-2022.

Hummel, P., Braun, M., Tretter, M., Dabrock, P., 2021. Data sovereignty: A review. *Big Data & Society*, 8(1). Doi.org/10.1177/2053951720982012.

Huo, Y., Yang, A., Jia, Q., Chen, Y., He, B., Li, J., 2021. Efficient Visualization of Large-Scale Oblique Photogrammetry Models in Unreal Engine. *ISPRS Int. J. Geo-Inf.*, 10(10), 643. Doi.org/10.3390/ijgi10100643.

Karis, B., Stubbe, R., Wihlidal, G., 2021. A Deep Dive into Nanite Virtualized Geometry. *SIGGRAPH 2021, Advances in Real-Time Rendering in Games*.

Kersten, T.P., Tschirschwitz, F., Deggim, S., 2017. Development of a Virtual Museum Including a 4D Presentation of Building History in Virtual Reality. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, XLII-2/W3, 361-367. doi.org/10.5194/isprs-archives-XLII-2-W3-361-2017.

Kukutai, T., Taylor, J. (Eds.), 2016. *Indigenous Data Sovereignty: Toward an Agenda*. ANU Press. Doi.org/10.22459/CAEPR38.11.2016.

Luebke, D., Reddy, M., Cohen, J.D., Varshney, A., Watson, B., Huebner, R., 2003. *Level of Detail for 3D Graphics*. Morgan Kaufmann / Elsevier. doi.org/10.1016/B978-155860838-2/50000-0.

Marnat, L., Gautier, C., Colin, C., Gesquiere, G., 2022. Py3DTilers: An Open Source Toolkit for Creating and Managing 2D/3D Geospatial Data. *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, X-4/W3-2022, 165-172. doi.org/10.5194/isprs-annals-X-4-W3-2022-165-2022.

OGC, 2019. OGC 3D Tiles Specification 1.0 (OGC 18-053r2). Open Geospatial Consortium.

Over, J.-S.R., Ritchie, A.C., Kranenburg, C.J., Brown, J., Buscombe, D., Noble, T., Sherwood, C.R., Warrick, J.A., Wernette, P.A., 2021. Processing Coastal Imagery with Agisoft Metashape Professional Edition, Version 1.6 - Structure from Motion Workflow Documentation. *U.S. Geological Survey Open-File Report 2021-1039*. Doi.org/10.3133/ofr20211039.

Pepe, M., Alfio, V.S., Costantino, D., 2022. UAV Platforms and the SfM-MVS Approach in the 3D Surveys and Modelling: A Review in the Cultural Heritage Field. *Applied Sciences*, 12(24), 12886. Doi.org/10.3390/app122412886.

Potenziani, M., Callieri, M., Dellepiane, M., Corsini, M., Ponchio, F., Scopigno, R., 2015. 3DHOP: 3D Heritage Online

Presenter. *Computers & Graphics*, 52, 129-141. Doi.org/10.1016/j.cag.2015.07.001.

Rantanen, T., Julin, A., Virtanen, J.-P., Hyyppa, H., Vaaja, M., 2023. Open Geospatial Data Integration in Game Engine for Urban Digital Twin Applications. *ISPRS Int. J. Geo-Inf.*, 12(8), 310. Doi.org/10.3390/ijgi12080310.

Skublewska-Paszkowska, M., Milosz, M., Powroznik, P., Lukasik, E., 2022. 3D Technologies for Intangible Cultural Heritage Preservation - Literature Review for Selected Databases. *Heritage Science*, 10, 3. doi.org/10.1186/s40494-021-00633-x.

Velayudhan, S.R., Bouveyron, T., Willmes, C., 2025. 3DTiler: A Tool to Georeference 3D Models and Generate 3D Tiles. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, XLVIII-4/W15-2025, 157-163. doi.org/10.5194/isprs-archives-xlviii-4-w15-2025-157-2025.

Wurstle, P., Padsala, R., Santhanavanich, T., Coors, V., 2022. Viability Testing of Game Engine Usage for Visualization of 3D Geospatial Data with OGC Standards. *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, X-4/W2-2022, 281-288. doi.org/10.5194/isprs-annals-x-4-w2-2022-281-2022.

Zhang, X., Liu, W., Liu, B., Zhao, X., Hu, Z., 2021. Size-Adaptive Texture Atlas Generation and Remapping for 3D Urban Building Models. *ISPRS Int. J. Geo-Inf.*, 10(12), 798. doi.org/10.3390/ijgi10120798.