Towards Secure and Usable 3D Assets: A Novel Framework for Automatic Visible Watermarking

Gursimran Singh, Tianxi Hu*, Mohammad Akbari*, Qiang Tang, Yong Zhang
Huawei Technologies Canada Co. Ltd.
{gursimran.singh1, cindy.hu1, mohammad.akbari, qiang.tang, yong.zhang3}@huawei.com

A. Supplementary Materials

In this appendix, we present the supplementary materials for the paper titled "*Towards Secure and Usable 3D Assets: A Novel Framework for Automatic Visible Watermarking*".

A.1. Code and Demo

In order for the results to be reproducible, we share our test code with detailed instructions in the supplementary materials. We also uploaded a video file demonstrating qualitative examples of watermarked 3D objects from various viewing angles. Code and demo are available here[1].

A.2. Datasets Statistics

As stated in Sec. 5, we sample a subset of 50 models from two benchmark 3D datasets, namely, Manifold40 and ObjaVerse. Particularly, we used random sampling stratified by output classes from the train set of the respective datasets. Additionally, for the Meshy GenAI dataset, we downloaded 20 textured models generated using the Meshy text-to-3D AI service. The statistics of the vertices and faces of these datasets are presented in Tab. 1.

Dataset		Vertices			Faces		
Name	#Samples	Min.	Max.	Mean	Min.	Max.	Mean
Manifold40	50	6561	137055	56943.2	13134	160004	101312.9
ObjaVerse	50	64933	1122245	234601.5	130088	182460	159874.9
Meshy	20	9058	37273	20981.5	18086	74512	41924.8

Table 1. Dataset Statistics.

A.3. User Study

In order to subjectively analyze the performance of our method compared to the baseline, we conducted a user study involving 10 volunteer participants. Each participant was randomly presented with either a textured or untextured 3D object from the GenAI Meshy dataset, watermarked using our method or the baseline. Participants were then asked to answer the following six "yes/no" questions assessing the watermark quality and utility of the displayed 3D object:

- Are the watermarks **visible** from different views?
- Are the watermarks' **placement** and orientation good?
- Are the watermark texts **readable**?
- Is the asset's **geometry**/shape preserved?
- Is the asset's **semantics** preserved?
- Are the asset's **salient** areas protected?

In total, 373 data samples were collected, where a value of 1 and 0 were respectively assigned to the "yes" and "no" answers. The averaged numerical results across all samples are summarized in Tab. 2. As shown in the table, the users gave significantly higher scores to our method for both textured and untextured objects in terms of the visibility of the watermarks from multiple views (Visibility), placement and orientation (Placement), and textual readability (Readability) of the watermarks. Specifically, across textured and untextured cases, the baseline scored approximately 46%, 68%, and 64% lower than our method for placement, readability, and visibility, respectively.

On the other hand, for asset utility, users rated our and the baseline method similarly in terms of preserving the overall semantics and context (Semantics) of the asset after watermarking. However, our method demonstrated superior performance compared to the baseline in preserving the geometry (Geometry) and salient features (Saliency) of the asset, achieving approximately 37% and 0.26% higher scores, respectively.

Overall, users generally rated our method slightly higher for watermark quality on untextured objects compared to our method on textured ones. This discrepancy is often due to texture (i.e., color information), which can significantly influence the visibility and readability of watermarks, particularly when the watermark color closely matches the asset's texture. We addressed this issue as a limitation of our method in Appendix A.14, highlighting its importance for future improvements.

*Equal contribution.

[1] https://developer.huaweicloud.com/develop/aigallery/notebook/detail?id=15adbaaa-2583-4ec3-804a-61c29f001e03

Dataset	Method	Watermark Quality			Asset Utility		
		Visibility ↑	Placement ↑	Readability ↑	Geometry ↑	Semantics ↑	Saliency ↑
Untextured	Li et al.	0.387	0.066	0.208	0.660	1.0	0.566
	Ours	**0.959**	**0.793**	**0.963**	**1.0**	**1.0**	**0.793**
Textured	Li et al.	0.462	0.076	0.295	0.591	0.984	0.515
	Ours	**0.818**	**0.709**	**0.822**	**0.984**	**1.0**	**0.814**

Table 2. User study results over GenAI Meshy dataset watermarked with our method and the baseline.

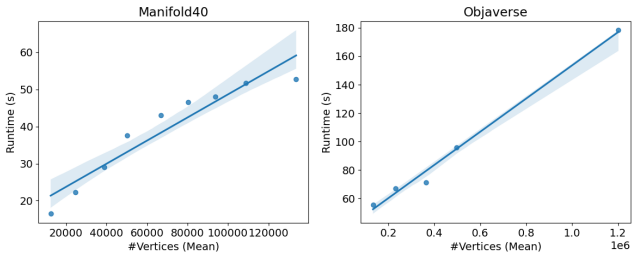

Figure 1. Average runtime (x-axis) required for watermarking models having an average number of vertices (y-axis) for Manifold40 (left) and ObjaVerse (right) datasets, respectively.

A.4. Runtime Analysis

In this section, we provide a runtime analysis of our method. We count all the time required for end-to-end watermarking of an asset, including any preprocessing time, candidate generation, optimization, filtering, and embossing time. The plots for the average runtime (in seconds) corresponding to the average number of vertices are reported in Fig. 1 for the Manifold40 and ObjaVerse datasets, respectively. Based on empirical analysis, the overall runtime grows linearly with the number of vertices of the target model. As seen in the plots, a model of 60K vertices requires $\approx$ 30s, and a model of 1.2M vertices requires $\approx$ 180s for watermarking.

A.5. Watermark Quality vs. Asset Utility Trade-off

In Sec. 5.1, we studied the trade-off between the watermark quality and asset utility by performing experiments with different numbers of watermarks $H_f = \{4, 16, 32\}$. Two trade-off curves including Semantic vs. and Placement, and SMSE vs. OCR were illustrated.

In this section, four more trade-off curves in terms of SMSE vs. Placement, Semantic vs. OCR, IPE vs. OCR, and IPE vs. Placement are shown in Fig. 2. Similar to the trade-off curves provided in the main body of the paper, increasing the number of watermarks results in higher watermark quality, but lower asset utility. However, our method achieves significantly improved trade-off results compared to the baseline ones. For example, our method can achieve an OCR score of $\approx$ 0.85, while providing an IPE error of 18.0. On the other hand, the baseline can achieve a much lower OCR score of 0.30 with a higher IPE error of 20.

Additionally, as shown in Fig. 2, the effect of the number

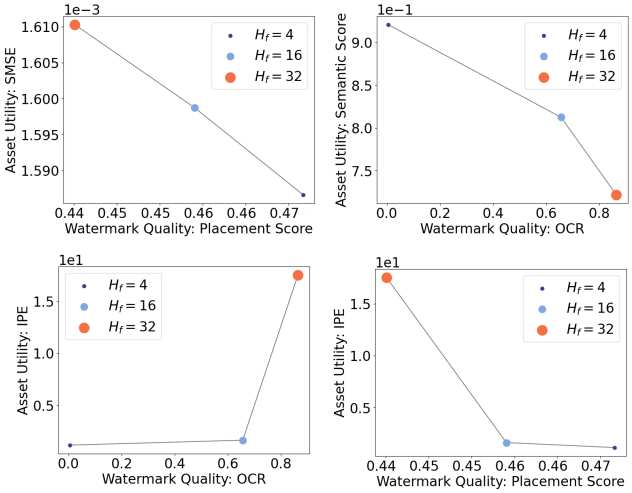

Figure 2. More trade-off results between the watermark quality and asset quality metrics on Manifold40. H_f : number of watermarks.

of watermarks on the placement score vs. the geometry-based SMSE error is very minor. In other words, regardless of the number of watermarks, our method can effectively find the optimal locations and orientations to emboss the watermark without damaging the overall geometry of the original asset.

A.6. Implementation Details

In this section, we provide more implementation details of our work. As stated in Sec. 3, the task of 3D visible watermarking has three inputs, namely 1) target model, 2) watermark text, and 3) algorithm parameters such as watermark text font, thickness, and size. We fix the input parameters for all our experiments unless stated otherwise. For input (1), since our method does not depend on texture information, we remove texture information from input models and convert them into standard OBJ file format [4].

However, our method supports watermarking textured objects, which is done by simply replacing the untextured original model with the textured one during the embossing step. We have provided some qualitative results of watermarking textured models in Fig. 10. Other than that, to avoid any variability in metrics computation, we stick to untextured models that are scaled to a fixed size of 30 and centered at origin $(0, 0, 0)$. Additionally, to preserve

Algorithm 1 Candidate Box Generation

Inputs: Target coordinates P_C^i, target normal P_N^i, watermark string Z_m, watermark size Z_s, watermark font Z_f

1: $V_i^{ws}, F_i^{ws} \leftarrow \texttt{text\_to\_3d}(Z_m, Z_s, Z_f)$
2: $V_i^{bs}, F_i^{bs} \leftarrow \texttt{oriented\_bounding\_box}(V_i^{ws}, F_i^{ws})$
3: $\alpha^i, \beta^i, \gamma^i \leftarrow \texttt{compute\_angles}([0,0,1], P_i^N)$
4: $R_i \leftarrow \texttt{generate\_rotation\_matrix}(\alpha_i, \beta_i, \gamma_i)$
5: $T_i \leftarrow \texttt{generate\_translation\_matrix}(P_i^C)$
6: $V_i \leftarrow T_i \cdot R_i \cdot V_i^{bs}$
7: **Output:** V_i

computational resources, we decimate the models to keep the number of vertices below $80,000$ for generating watermark boxes by our algorithm. However, during the watermark embossing step, we still use the original undecimated model. Further, we always use "watermark" as the watermark text (input 2) for all our experiments and use the default text font as provided by the off-the-shelf library [7] for converting text to 3D mesh. We use the thickness (distance between the front and back faces of the watermark) of 0.5 and a fixed watermark size (scale of mesh) of 4 in all our experiments unless stated otherwise. Finally, in the embossing module, we use a fixed value of 0.05 as the extrude strength in Algorithm 2.

We sample a fixed number of $H_s = 300$ points for generating the initial number of candidate boxes. From this initial set, we obtain the final H by rejecting points that are too close (radius $H_r < 1$). The number of final watermarks after filtering H_f is variable for each model, whose average value is 9.97 for Manifold40 and 8.82 for ObjaVerse. The value of J, which is the number of sampled points for computing the alignment loss (Eq. 9) is fixed to be 179, including the midpoints. For optimizing the objective in Eq. 8, we run a fixed number 200 of gradient descent steps and use stopping criteria of mean loss less than 0.005.

We used a 12-CPU-core machine with two NVIDIA GeForce GTX 1080 to run our experiments. All our code was implemented in Python 3.10 and the optimization objective including gradient back-propagation was implemented using Pytorch 3D [9]. We use off-the-shelf 3D libraries to implement many common mesh operations in this work. Note that, our work can handle all 3D models which can be converted to a mesh and which support 3D Boolean operations. We can simply convert the given format into the mesh model to obtain watermark locations and apply Boolean operations in the end to fuse watermarks into the target format.

A.7. Initialization (more details)

In this section, we provide more specific details of the Initialization module (Sec. 4.1). As mentioned earlier, we sample H equidistant points on the surface of the target model. Specifically, we start by randomly sampling H_s points $\{(x,y,z)|x,y,z \in \mathbb{R}\}_{i=1}^{H_s}$ on the surface of the target model. Then, we reject the points that are closer to each other than a radius of H_r. After this step, we denote the final set of points, that are approximately equidistant, by $\{P_C^i\}_{i=1}^H$ and their corresponding surface normal by $\{P_N^i\}_{i=1}^H$.

Then, for each of these sampled points, we use the procedure in Algorithm 1 to generate the candidate boxes. Specifically, we start (Lines 1-2) by generating a watermark mesh $W_i^s(V_i^{ws}, F_i^{ws})$ and its bounding box $B_i^s(V_i^{bs}, F_i^{bs})$ using off-the-shelf algorithms `text_to_3d` and `oriented_bounding_box`. We configure these algorithms to make sure that these meshes are generated at origin $(0,0,0)$ and the face of the 3D text faces towards the $+Z$ direction (0, 0, 1), also referred to as front direction. Then, we perform a rigid-body transformation operation (Lines 3-6) to transport the box at the i^{th} sampled location P_i^C and align the box along its normal P_i^N. Specifically, we use the `compute_angles` routine (Line 3) to compute the angles between the front direction of the box $(0,0,1)$ and the target direction P_i^N. We use these angles $\alpha_i, \beta_i, \gamma_i$ to for the rotation R_i (Line 4) and the target location P_i^C to compute the translation matrix T_i (Line 5). Finally, we use these rotation and translation matrices for the final transform (Line 6) to obtain the transformed vertices V_i.

A.8. Finetuning (visualization)

In this section, we present visualizations before and after optimization in the finetuning module (Sec. 4.2). As shown in Fig. 3-left, the boxes placed using the initialization module are misaligned with the surface of the dolphin's body. After optimization (Fig. 3-right), the boxes' alignment is corrected, and they are positioned accurately to follow the curvature of the dolphin's surface. Specifically, the optimization involves three operations: tilting, rotating, or moving the boxes to achieve proper alignment. For instance, the cyan box situated at the top fin of the dolphin cannot be rotated or tilted and thus needs to be relocated to improve its alignment. Conversely, many boxes on the body can be adjusted by simply rotating or tilting them to correct their alignment.

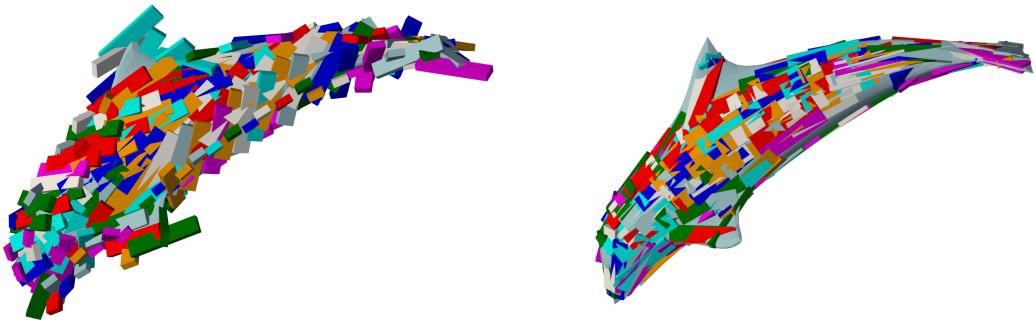

Figure 3. Effect of optimization on candidate boxes. The left figure shows misaligned candidate boxes placed using Algorithm 1 that are fixed (right) by either moving, rotating, or tilting these boxes using the proposed rigid body optimization.

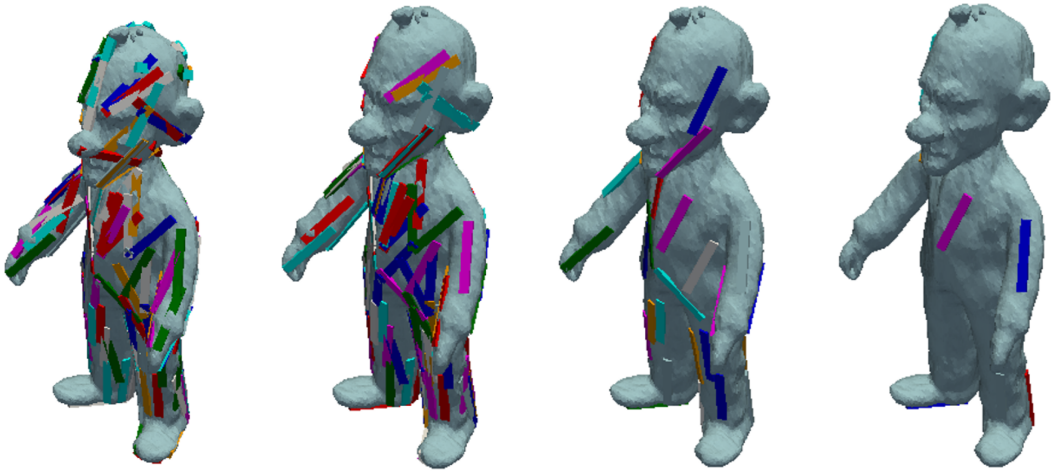

Figure 4. Step-by-Step Filtering Process: This illustration visualizes the progressive filtering of bounding boxes. From left to right, each image displays the remaining boxes after applying a specific filter. The first image shows the result after the low roughness score filter. The second image depicts the results after discarding boxes with low loss. The third image presents the outcome of filtering out overlapping and occluded boxes. Finally, the fourth image displays the final set after applying a multi-octant and multi-angle visibility filter.

A.9. Filtering (more details)

In this section, we provide more details about the individual filtering steps discussed in Sec. 4.3. Going from left to right, Fig. 4 shows the results of various steps of filtering operation. As seen, each filtering step prunes the undesirable boxes and keeps the most important boxes with an aim of high watermark visibility and high asset utility.

Well Aligned Boxes with Low Loss: The loss defined in Eq. 9 quantifies the alignment accuracy of the box with the mesh surface. To reject sub-optimal boxes that are misaligned, we apply straightforward thresholding on the individual box loss. Through observation, we have determined that a loss less than 0.005 typically indicates well-aligned boxes.

Boxes with Low Local Roughness: We calculate the local roughness beneath each candidate bounding box and discard boxes exceeding a specific threshold. Here's the detailed procedure. First, we identify all vertices within the i-th bounding box B_i. From these vertices, we randomly sample H_r points and compute the average cross-product of their normals. The local roughness score is defined as the inverse of this average cross-product $\mathcal{R}(B_i) = \frac{1}{{H_r}^2} \sum_{j=1}^{H_r} \sum_{k=1}^{H_r} \frac{1}{\cos(N_j^i, N_k^i)}$ where (N_j^i, N_k^i) are the normals of points inside box B_i. Through analysis, we have determined that a roughness score less than 1.25 typically indicates boxes located on flatter surfaces.

Non-Overlapping Boxes: To handle potential overlaps among candidate boxes, we employ a greedy approach. Initially, we randomly select a box and iteratively discard any box that overlaps with those already chosen. Overlap is determined by checking for intersections among the vertices of the original mesh contained within pairs of bounding boxes.

Non-Occluding Boxes: To mitigate potential occlusions of some boxes by parts of the target model, such as under

Algorithm 2 Curve Matching Fusion

Inputs: original mesh M, watermark meshes $\{W_f^i\}_{i=1}^{H_f}$, extrude strength H_y

1: **for** $W_i \in \{W_i\}_{i=1}^{H_f}$ **do**
2: $\quad \overline{W}_i \leftarrow \texttt{boolean\_intersection}(M, W_i)$
3: $\quad N_i^E \leftarrow \texttt{closest\_normal}(W_i)$
4: $\quad \tilde{W}_i \leftarrow \texttt{perform\_extrusion}(\overline{W}_i, N_i^W, H_y)$
5: **end for**
6: $M' \leftarrow \texttt{boolean\_union}(M, \{\tilde{W}_i\}_{i=1}^{H_f})$
7: **Output:** M'

the arms or between the thighs in humanoid models, leading to diminished watermark visibility, we utilize a ray casting method. This approach helps identify and subsequently remove watermarks that are occluded. We sample equidistant points from the front face of each bounding box and cast rays along the normal direction of the watermark. If any of these cast rays intersect with parts of the target model, the watermark is classified as occluded and is subsequently removed.

Multi-Octant Presence: To deter model theft, watermarks should be distributed across diverse locations of the model. We achieve this by dividing the model into 8 octants using planes along the X, Y, and Z axes passing through the model's centroid. Each octant is assigned a watermark. If multiple watermark options exist per octant, we select the one furthest from the watermarks in adjacent octants.

Multi-Angle Visibility: In this step, we add extra boxes to ensure the watermark is visible from multiple viewing angles. This prevents attackers from using 2D renders of a 3D object where the watermark might not be visible due to camera angles. Our goal is to position at least one watermark on the visible portion of the model's surface for each viewing angle. To achieve this, we iterate through fixed angle increments of 30° around the X and Z axes and add a watermark if no other existing watermarks are found for that angle.

A.10. Watermark Embossing

In this section, we provide more details of the novel curve-matching fusion presented in Sec. 4.4. We start by generating 3D-text watermark meshes $\{W_i\}_{i=1}^{H_f}$ using a standard text-to-3D algorithm [7], positioned and oriented according to selected bounding boxes $\{B_f^i\}_{i=1}^{H_f}$. Then, given the target mesh M and the generated 3D watermarks $\{W_i\}_{i=1}^{H_f}$, we use Algorithm 2 to obtain the watermarked mesh M'. Specifically, first, we apply a boolean intersection operation (Line 2) between the original mesh M and i-th watermark mesh W_i. Then, we find the extruding normal N_i^E by computing the normal of the closest point on the mesh (Line 3) from the centroid of watermark mesh W_i. Next, we perform the extrusion operation (Line 4) of the intersection $\bar{W}_i$ silhouette to give an embossing effect. Finally, we simply take a Boolean union [6] of the extruded watermark meshes $\{\tilde{W}_i\}_{i=1}^{H_f}$ and the original mesh M to obtain the final watermarked result M'.

A.11. Evaluation Metrics

In this section, we provide additional details of the evaluation metrics discussed in Sec. 5.

A.11.1 Watermark Quality

Watermark Placement Score (WPS): WPS measures the alignment between the watermarks and the mesh. The inputs consist of a mesh M and a bounding box B_i. We start by computing vertices of the mesh M that lie inside the bounding box B_i and denote them as V^{in}. Then, we find all faces that have at least one vertex in the set V^{in} and denote them as F^c. From these faces, we only consider faces that lie completely inside the bounding box (all three vertices in V^{in}) and compute their areas. Then, we project these areas in the direction of the front face of the box. Finally, we sum up the areas and divide the sum by the area of the front face of the box to compute the watermark placement score. For multiple bounding boxes, we simply report the mean score computed across multiple B_is. Note that, we used this approximate procedure to compute area for efficiency purposes as the standard methods do not scale well to a large number of vertices.

Ray Casting Visibility (Ray): Ray measures the visibility of watermarks from all views of the model. We begin by generating views of the watermarked mesh M' by rotating the camera around the X and Z axes in 30° increments. For each view c_w^t, we identify candidate watermark meshes oriented within 45 degrees of the camera's direction. Using ray casting, multiple random rays are projected from the top face of each candidate's bounding box towards the camera. A per-watermark score of 1 is assigned if all rays reach the camera without obstruction; otherwise, it is 0. The per-view score is computed by averaging across all per-watermark scores in that view. Finally, the final ray score is obtained by averaging overall per-view scores.

OCR Visibility (OCR): OCR measures the readability of watermarks from all views of the model. We begin by generating renders of the watermarked mesh M' for each view c_w^t, obtained by rotating the camera around the X and Z axes in increments of 30°. Next, we utilize an off-the-shelf OCR detector [2] to identify candidate 2D boxes that potentially contain readable text. Subsequently, to account for text orientations that are not left-to-right aligned, we augment the candidate boxes by adding rotations of 90°, 180°, and 270°. Then, we use an off-the-shelf OCR recognizer [2] to generate candidate text recognitions. These candidates are then scored using a popular sequence matcher [1] to quantify their similarity to the ground truth watermark text. Finally, for each view c_w^t, we take the maximum score and average these scores across all views to obtain the final OCR score.

A.11.2 Asset Utility

Sampled Mean Squared Error (SMSE): SMSE aims to compute the Mean Squared Error (MSE) between the watermarked mesh and the original mesh. Since the number of vertices and faces changes after watermarking, it is not possible to compute the MSE directly. Instead, we start by randomly sampling a large number of points on the surface of the watermarked mesh M'. Then, we compute the distances of these sampled points from the original mesh M using a standard routine in the Trimesh package [3]. Finally, we report the inverse of the mean distance values as the final SMSE score.

Isolated Parts Error (IPE): IPE is a measurement of the change in mesh topology before and after watermarking. It is computed as the difference in the total number of isolated parts between the watermarked mesh M' and the original mesh M. The motivation is based on the intuition that the number of isolated parts in a model should remain identical after watermarking. An increased IPE captures the cases when a part of the watermark text is disconnected from the model surface. Such isolated parts degrade the asset utility and can be easily removed to damage the watermark message. Lower IPE indicates less change in the mesh topology and therefore better watermark placement.

Local Curvature Error (LCE): LCE measures how well the surface curvature of watermarked areas is preserved (as discussed in Sec. 4.4 and Appendix A.10). For each vertex on the top face of a watermark, the distance to its nearest neighbor on the original mesh surface M is computed. This process is repeated for every vertex and watermark, and the LCE is calculated as the variance of these distances. A lower LCE indicates that the watermark conforms to the surface curvature, while a higher LCE indicates deviation from the underlying surface curves.

Saliency Error (SE): SE is designed to assess if any of the watermark placed covers the salient features of the original mesh. It is computed by first calculating a normalized continuous saliency map of the mesh M using an off-the-shelf implementation in [8]. Then, we threshold the saliency map using Otsu's method [10] to have binary per-vertex salient/non-salient scores. Next, for each watermark bounding box B_i, we compute a binary saliency vote for each bounding box indicating if it is placed on a highly salient area. Specifically, we assign a value of 1 if the average of thresholded saliency values within the box is greater than 0.5, and a value of 0 otherwise. The average value of saliency votes overall bounding boxes is reported as the saliency score.

Semantics Score (SS): SS is used to measure how well a model's semantics is preserved through measuring the change in visual features after watermarking. To compute it, we start by generating renders of the target M and watermarked mesh M' by rotating the camera around the X and Z axes in 30° increments. Then, for each pair of corresponding 2D renders, we compute the cosine similarity between their feature vectors extracted using a pretrained ResNet50 [5]. Finally, we average these per-view cosine similarity scores over all views $\{(c_w^t, c_o^t)\}_{t=1}^T$ (taken at 30° increments around the X and Z axes) to obtain the final score.

A.12. More Attacks Analysis

In Sec. 5.4, we presented a preliminary analysis of attacks and robustness. Specifically, we demonstrated the superiority of our method compared to the Li et al. (visible) and Wang et al. (invisible) baselines against three attacks: cropping, unauthorized removal (see Tab. 3 in the main body of the paper), and remeshing attacks (see Fig. 5 in the main body of the paper). In the following section, we extend this analysis and provide additional results.

We begin by analyzing the effects of unauthorized removal attacks on our approach and the Li et al. baseline, as detailed in Sec. 5.4. The qualitative results of this attack are presented in Fig. 7. In this attack scenario, we assume a sophisticated adversary who can identify all vertices and faces of the watermarks and remove them using mesh editing software. This task can be quite challenging unless the watermarks are colored with a distinct color. As shown in the figure, even when the attacker knows the vertices and faces, watermarks remain clearly visible in our method due to the silhouette created by the holes. However, for the baseline method, since the watermarks may not fully touch the model surface due to poor orientation, the resulting silhouettes are partial, making the watermarks unreadable.

Next, we conducted tests on typical mesh editing operations such as decimation and simplification using visible watermarks (ours) and invisible watermarks (Wang et al.). We applied a decimation strength of 0.9 and a subdivision

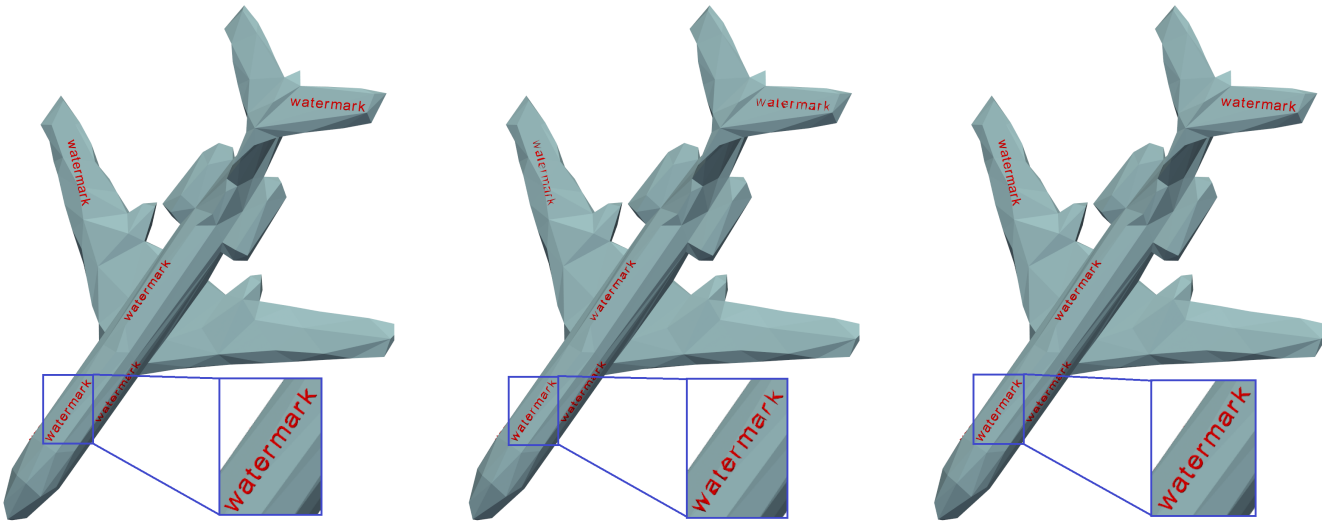

Figure 5. Impact of mesh editing attacks. From left to right, the first figure shows the original object with no attacks. The second shows the effect of a severe decimation attack (strength = 0.9) and the third shows the effect of a subdivision attack (strength = 2). As seen the watermarks are slightly affected by the decimation attack but they are still visible. On the other hand, the watermarks are unaffected by subdivision attacks.

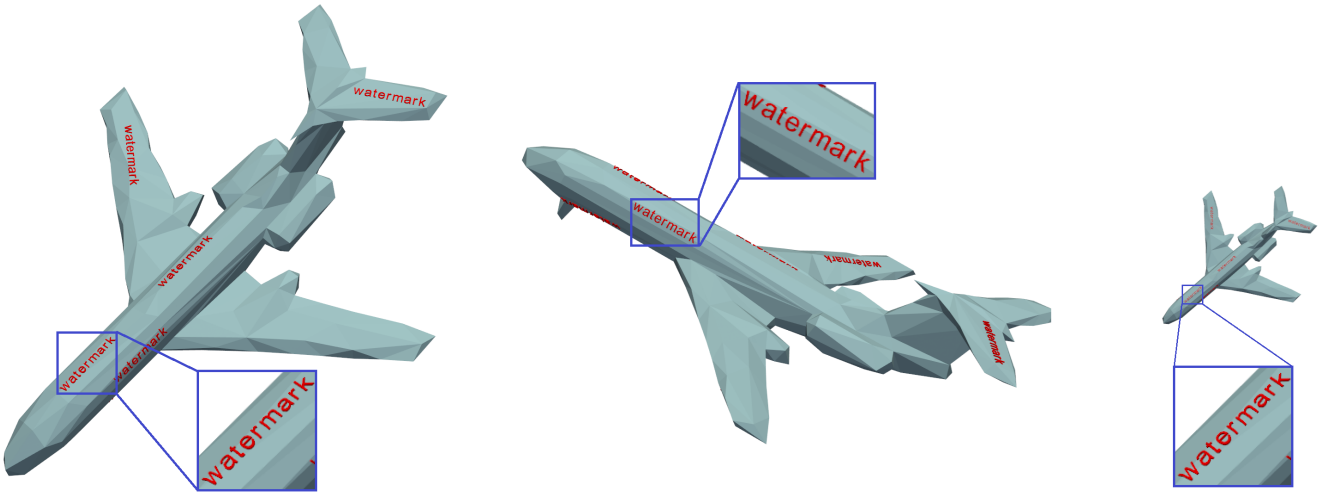

Figure 6. Impact of geometric attacks. From left to right, the first figure shows the effect of translation to a random position. The second shows the effect of rotation by a random angle and the third shows the effect of scaling the object. As shown, the watermarks move synchronously and are not affected by these inadvertent transformations.

strength of 2. These values were chosen to be sufficiently high while ensuring that the visual integrity of the asset remains intact to a normal eye. The results are presented in Fig. 5. In both attacks, the invisible watermark could not be successfully extracted (with less than 50% bit accuracy), whereas our method preserved the watermark well enough for the message to be clearly readable from multiple angles. Specifically, the clarity of watermarks degraded slightly under the decimation attack but remained completely unaffected by the subdivision attack. We observed that increasing the strength of the decimation attacks could completely erode the watermark, but at that point, the utility of the asset was also significantly degraded.

Lastly, we present qualitative results demonstrating inadvertent geometric operations performed in mesh editing software in Fig. 6. For these operations, such as rotation, scaling, and translation, the watermarks are also transformed synchronously, hence the watermark quality remains unaffected.

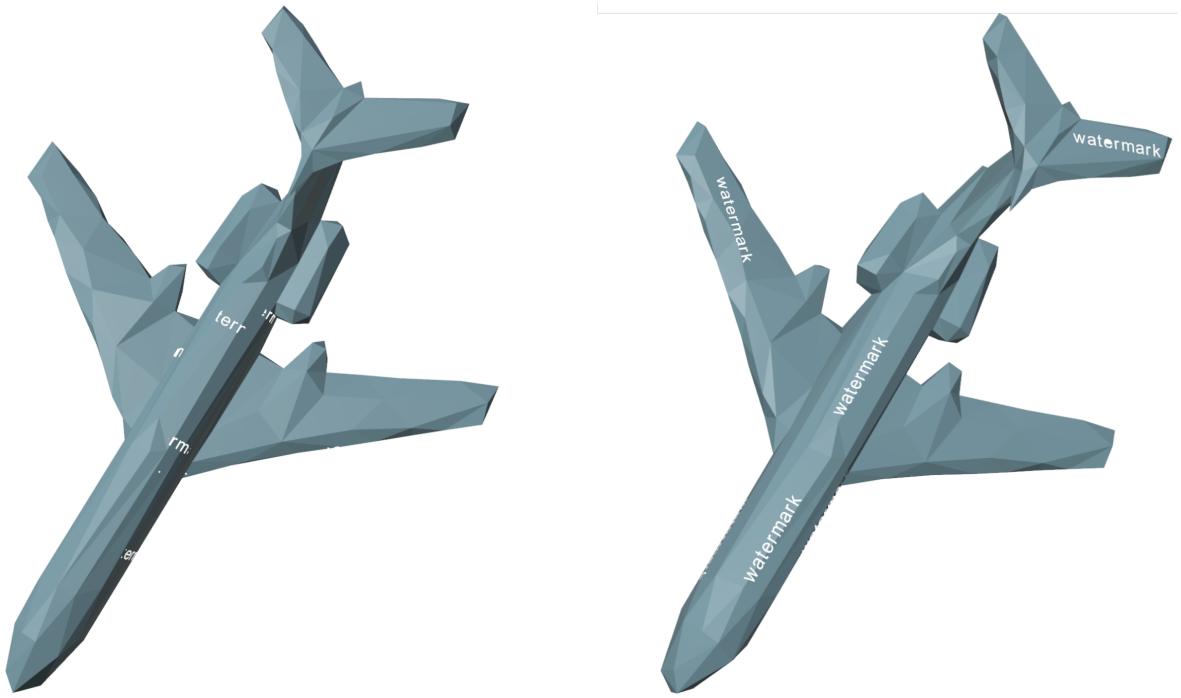

Figure 7. Impact of Unauthorized Removal Attack: This figure demonstrates the effect of an unauthorized removal attack, where an attacker attempts to eliminate watermarks by deleting all faces and vertices associated with the watermark mesh. The left side shows the attack applied to Li et al.'s baseline method, where the complete watermark message cannot be read. Conversely, the right side showcases the attack on our proposed method, where we can still easily identify the watermark message, demonstrating its superior resilience compared to the baseline.

A.13. More Qualitative Results

Fig. 8 and Fig. 9 show some visual examples of the models (from Manifold40 and ObjaVerse) watermarked with our method and the baseline. As shown, compared to the baseline, our method generates watermarks with significantly better placement, orientation, readability, and visibility from multiple views. On the other hand, the baseline produces watermarks that either fly out or are hidden under the surface.

Please note that we colored (i.e., adding texture) all the watermarks in Red in all the qualitative results for better observability for the reader. However, as also shown in Fig. 10, the untextured watermarks still provide high visibility for the IP protection of the objects.

Moreover, three textured models (from the collected GenAI Meshy dataset) watermarked with our method and the baseline are illustrated in Fig. 10. Similar to the visual examples related to human-made datasets in Fig. 8 and Fig. 9, our method generates watermarks with significantly better placement, orientation, readability, and visibility from multiple views compared to the baseline.

It should be noted that our method has been optimized to preserve the most salient features of the mesh without considering the texture information. As a result, for the textured models, it is possible that our method places a watermark on the areas that are visually seen as highly salient due to the presence of the texture (i.e., color information). For example, in the textured shark model in Fig. 10, a watermark is placed near the eyes and nose of the shark. However, as also illustrated in the untextured version of the object, such details are not present in the mesh, and the selected area to emboss the watermark is smooth without any salient features.

A.14. Limitations and Future Work

Automated visible watermarking offers a practical framework for several critical scenarios, such as GenAI misuse, merchandise protection, and copyright violation. However, being the first work in this direction, it has several limitations that present significant opportunities for future research.

One significant concern revolves around how well our proposed benchmarks for watermark quality and asset utility align with human perception. The impact of visible watermarks on the perceived asset utility can vary significantly depending on the specific downstream application. Addi-

tionally, factors like viewing angle, texture, lighting conditions, and background complexity can influence how watermark quality is perceived in different contexts. This variability and subjectivity complicate the usability and reliability of our proposed metrics across all scenarios universally. Addressing these challenges represents an intriguing direction for future research.

Additionally, the robustness of visible watermarks against more intentional attacks poses a significant challenge in certain scenarios. A determined adversary may employ skilled 3D artists to manually remove watermarks and illegitimately sell or use the unwatermarked asset, violating the copyright of the owner. Although such effort will come at a significant cost, due consideration needs to be given to this possibility while employing this technology for practical purposes. Further, it would be an interesting direction for future work to explore and evaluate more sophisticated and automated attacks against our solution and propose a better and a resilient solution.

Further, our proposed solution works in four independent steps that are not end-to-end optimized for the most efficient performance. Specifically, we believe better performance can be significantly improved by combining the bounding-box fine-tuning and the filtering steps into one single optimization objective. However, this optimization can be challenging due to the various discrete operations in the gradient back-propagation process. We leave it to future work to solve these challenges and propose an improved solution that can further the state-of-the-art in this promising direction.

Finally, beyond technical considerations, visible 3D watermarking raises practical concerns regarding artistic integrity, where a fine balance needs to be maintained between content protection and user acceptance. Additionally, incorporating 3D visible watermarking in real-time or interactive applications imposes computational overhead that may impact performance and user experience. Addressing these multifaceted and practical challenges is essential for unlocking the full potential of visible 3D watermarking.

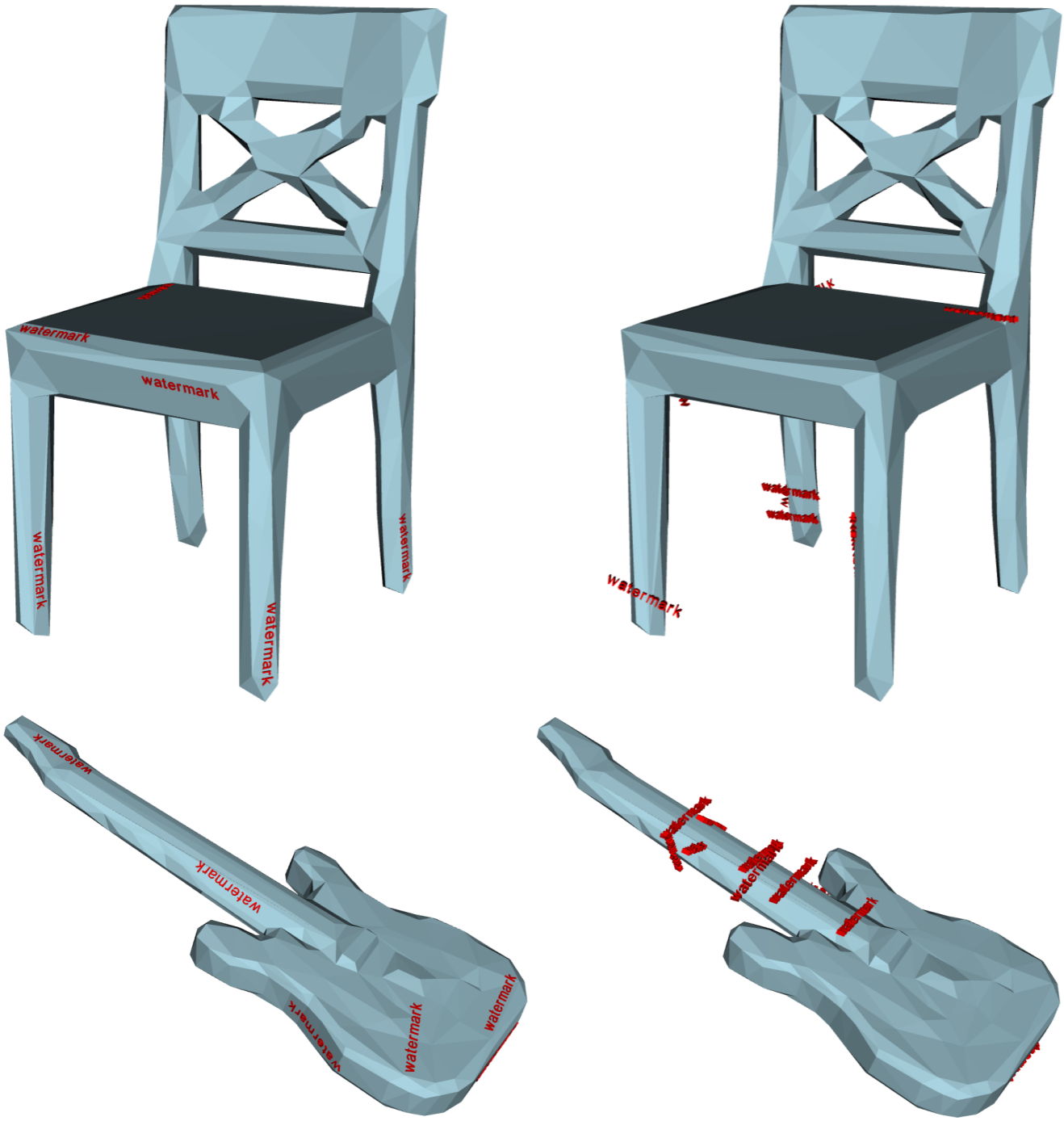

Figure 8. Two visual examples from Manifold40 showing 3D models watermarked with our method (left) and Li et al. baseline right). Ours provides better placement quality, readability, and viewability. We colored the watermarks in Red for better observability for the reader.

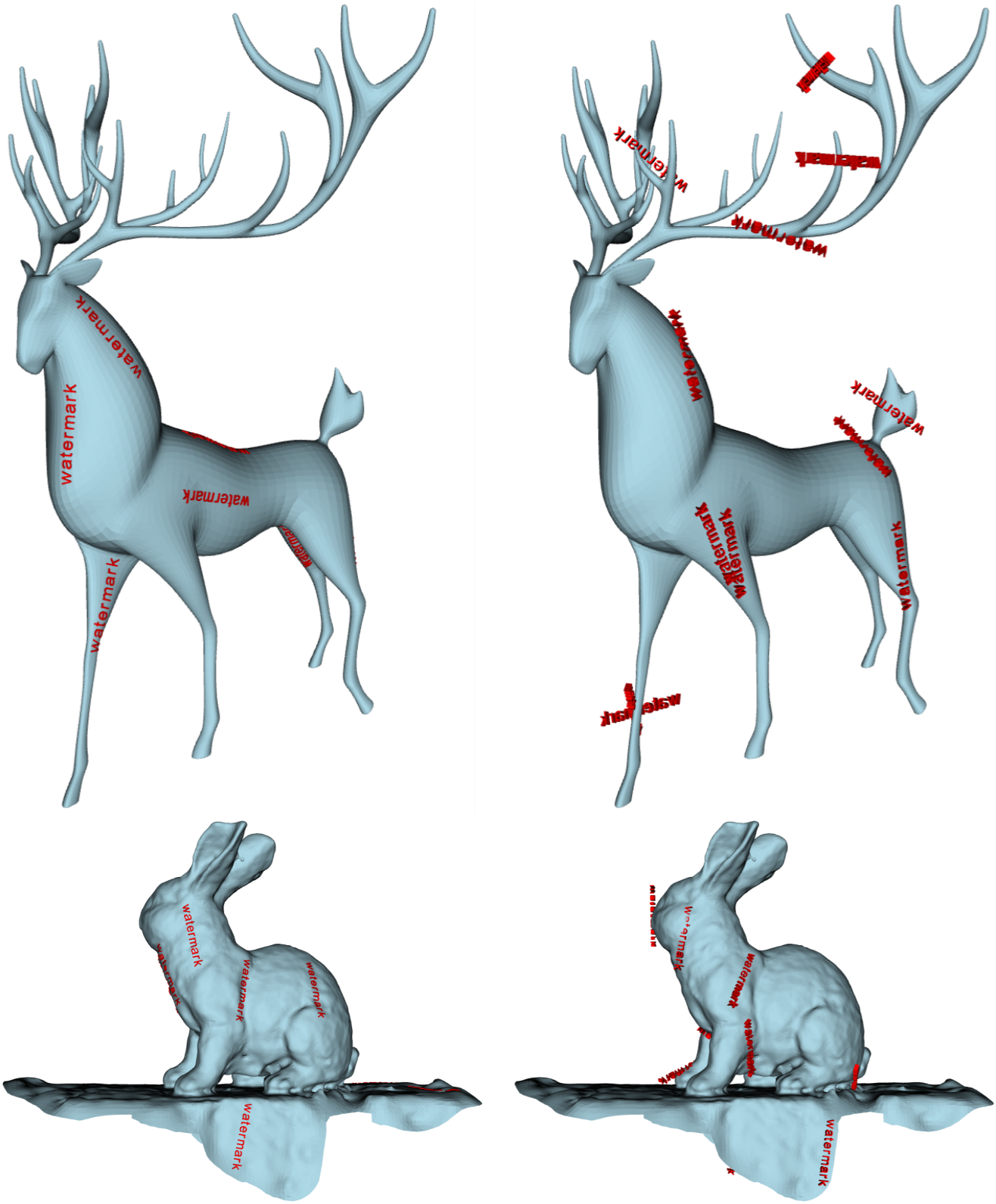

Figure 9. Two visual examples from ObjaVerse showing 3D models watermarked with our method (left) and Li et al. baseline (right). Ours provides better placement quality, readability, and viewability. We colored the watermarks in Red for better observability for the reader.

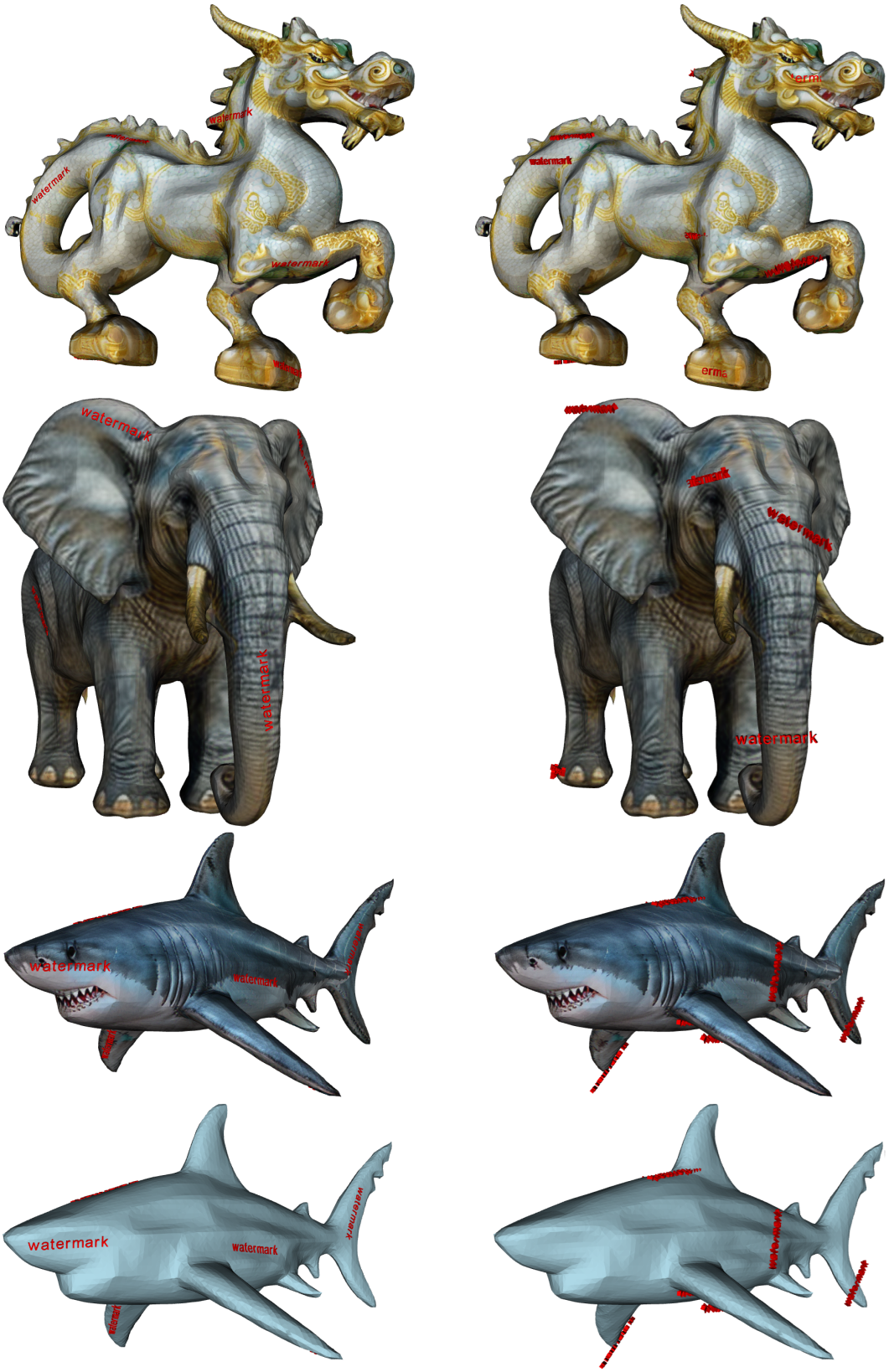

Figure 10. Two visual examples from GenAI Meshy showing textured 3D models watermarked with our method (left) and Li et al. baseline (right). Ours provides better placement quality, readability, and viewability.

References

[1] Virginia Brancato, Joaquim Miguel Oliveira, Vitor Manuel Correlo, Rui Luis Reis, and Subhas C. Kundu. Could 3D models of cancer enhance drug screening? *Biomaterials*, 232:119744, Feb. 2020.

[2] Keras OCR contributers. A packaged and flexible version of the craft text detector and keras crnn recognition model.

[3] Dawson-Haggerty et al. trimesh. python library for loading and using triangular meshes.

[4] Planning for Library of Congress Collections. Wavefront obj file format.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, Dec. 2015. arXiv:1512.03385 [cs].

[6] An-Bo Li, Hao Chen, and Xian-Li Xie. Visible watermarking for 3D models based on 3D Boolean operation. *Egyptian Informatics Journal*, 25:100436, Mar. 2024.

[7] Harishankar Narayanan. codetiger/Font23D, July 2024. original-date: 2015-04-24T08:53:52Z.

[8] Stavros Nousias, Gerasimos Arvanitis, Aris S. Lalos, and K. Moustakas. Mesh Saliency Detection Using Convolutional Neural Networks. *IEEE International Conference on Multimedia and Expo*, 2020.

[9] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3D Deep Learning with PyTorch3D, July 2020. arXiv:2007.08501 [cs].

[10] Xiangyang Xu, Shengzhou Xu, Lianghai Jin, and Enmin Song. Characteristic analysis of Otsu threshold and its applications. *Pattern Recognition Letters*, 32:956–961, May 2011.