Deep 3D-to-2D Watermarking: Embedding Messages in 3D Meshes and Extracting Them from 2D Renderings

Innfarn Yoo[1]   Huiwen Chang[1]   Xiyang Luo[1]   Ondrej Stava[2]
Ce Liu[1*]   Peyman Milanfar[1]   Feng Yang[1]
[1]Google Research   [2]Google LLC

{innfarn, huiwenchang, xyluo, ostava, milanfar, fengyang}@google.com ce.liu@microsoft.com

Abstract

Digital watermarking is widely used for copyright protection. Traditional 3D watermarking approaches or commercial software are typically designed to embed messages into 3D meshes, and later retrieve the messages directly from distorted/undistorted watermarked 3D meshes. However, in many cases, users only have access to rendered 2D images instead of 3D meshes. Unfortunately, retrieving messages from 2D renderings of 3D meshes is still challenging and underexplored. We introduce a novel end-to-end learning framework to solve this problem through: 1) an encoder to covertly embed messages in both mesh geometry and textures; 2) a differentiable renderer to render watermarked 3D objects from different camera angles and under varied lighting conditions; 3) a decoder to recover the messages from 2D rendered images. From our experiments, we show that our model can learn to embed information visually imperceptible to humans, and to retrieve the embedded information from 2D renderings that undergo 3D distortions. In addition, we demonstrate that our method can also work with other renderers, such as ray tracers and real-time renderers with and without fine-tuning.

1. Introduction

Digital watermarking is a key technology for copyright protection, source tracking, and authentication for digital content. The goal of digital watermarking is to embed messages in another media, *e.g.*, image, video, or 3D, and be able to decode the messages even after the watermarked media is copied, edited or distorted. There are two types of watermarking: invisible or visible. In this paper, we will mainly focus on invisible watermarking which means the watermarked media should be perceptually the same as the original media.

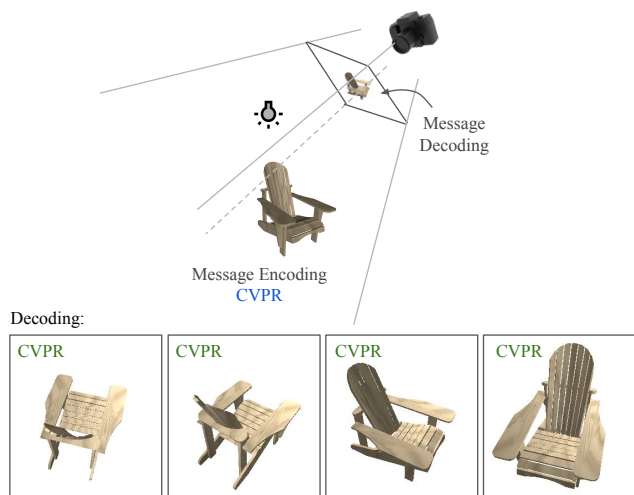3D models, due to their increasing ubiquity in a wide

Figure 1. A chair mesh is watermarked with a message "CVPR", then rendered as images. The embedded message can be retrieved from rendered images from different views and lighting conditions.

range of applications such as movie making, gaming, 3D printing, augmented reality (AR), 3D mapping, *etc.*, have become an important subject of digital watermarking. *The classic problem of watermarking for 3D models has been primarily formulated as embedding and retrieving messages, both in 3D space*. While traditional 3D watermarking methods [18,34] and commercial tools [27] are useful in 3D manufacturing and printing industries, they are inapplicable to mediums such as gaming, movie making and graphic design, where *the messages need to be retrievable from 2D renderings of the 3D models*.

In this work, we study the problem of 3D-to-2D watermarking – how to embed messages invisibly in 3D meshes and recover them robustly from 2D renderings. Recently, a number of research works [1,15,42] shows that approaches based on deep neural networks can achieve the state-of-the-art performance on image watermarking. One possible so-

---

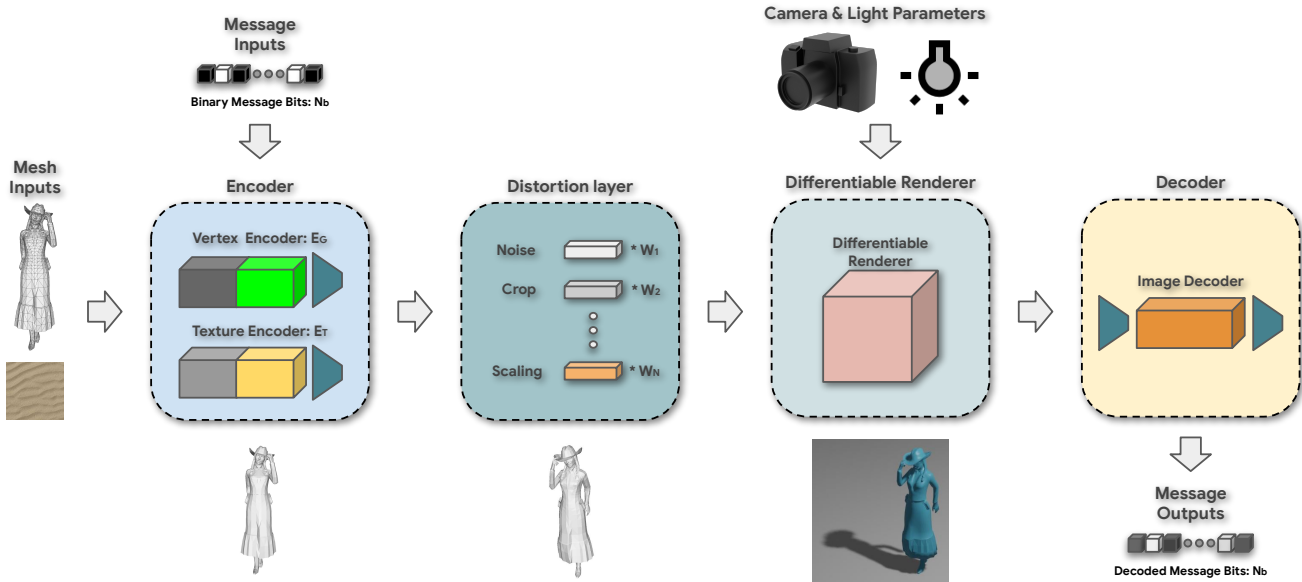*Currently affiliated with Microsoft Azure AI.

Figure 2. Overview of our deep watermarking pipeline. The encoder embeds messages into 3D meshes in an imperceptible fashion. Then we apply some distortions like crop, scaling to the watermarked 3D mesh. The rendered images are generated using a differentiable renderer. After that we extract the messages from the rendered 2D images.

lution that takes advantage of these methods is to embed and retrieve messages in the rendered 2D images of 3D objects. However, this approach cannot be easily adapted to decode 2D renderings of the same objects from different camera views or under varied lighting and shading conditions and thus is impractical. Another solution is to embed messages in the 2D textures of 3D meshes and extract the messages from 2D renderings. Our experiments show that it does not work since the existing deep image watermarking solution is not robust to 3D to 2D rendering distortion.

To address these issues, we propose an end-to-end trainable deep 3D-to-2D watermarking framework which is resistant to various 3D distortions and capable of extracting messages from images rendered under different lighting conditions and camera views.

Our framework consists of 1) an *encoder*; 2) a *distortion layer*; 3) a *differentiable renderer*; and 4) a *decoder* as shown in Fig. 2. Specifically, inspired by the advances of 3D reconstruction using differentiable renderers [12, 32, 38], we employ a state-of-the-art differentiable renderer to bridge the gap between the 3D encoding and 2D decoding stages. In the encoding stage, our model first embeds message bits into either 3D geometry and/or 2D textures of the original 3D data. In the decoding stage, it learns to extract message bits from 2D images generated by the differentiable renderer. We build a 3D-to-2D watermarking benchmark including 5146 3D objects with textures, and we investigate the performance of different architectures on it. We then analyze the performance of our method by measur-

ing its capacity, *i.e.*, the size of the message we can embed, its robustness, *i.e.*, the bit accuracy with respect to various distortions, and its quality or invisibility, *i.e.*, difference between watermarked/unwatermarked 2D renderings. Finally, we show that our model can work with other renderers, such as ray tracing and real-time renderers, as well as the availability of fine-tuning our decoder for other not differentiable renderers.

Our key contributions are following:

- To the best of our knowledge, this paper presents the first 3D-to-2D watermarking method which can retrieve messages encoded in 3D meshes from its rendered 2D image and broaden 3D watermarking usage.

- The use of differentiable rendering makes our method fully-differentiable, which enables to train the whole framework end-to-end with a collection of differentiable 3D distortions.

- Our decoder can decode the embedded messages from non-differentiable renderers, and can be improved further by fine-tuning.

While our model performs well as shown in Sec. 4, it is still limited in multiple aspects. For example, our method has low bit capacities compared to traditional 3D watermarking methods. However, low bit capacities are still useful for copyright protection (*e.g.*, zero-bit watermarking). Another limitation is that if attackers use totally different style rendering techniques such as cartoon rendering, our model needs to be re-trained.

2. Related Work

Methods relevant to our work are categorized as watermarking, differentiable rendering, neural network architectures, and others.

2.1. Watermarking

There is a vast body of work related to watermarking on 3D visual data. Several papers [2, 34] gave a nice survey on 3D watermarking. Early 3D watermarking approaches [16, 18, 33] leveraged Fourier or wavelet analysis on triangular or polygonal meshes. Recently, Hou *et al.* [5] introduced a 3D watermarking method using the layering artifacts in 3D printed objects. Son *et al.* [25] used mesh saliency as a perceptual metric to minimize vertex distortions. Hamidi *et al.* [4] further extended mesh saliency with wavelet transform to make 3D watermarking robust. Jing *et al.* [11] studied watermarking for point clouds through analyzing vertex curvatures.

Deep learning based methods for image watermarking [1,15,35,39,42] achieved great progress in recent years. HiDDeN [42] was one of the first deep image watermarking methods that achieved good performance compared to traditional watermarking approaches. Many extensions of this work have since been proposed, including a novel architecture based on circulant convolutions [1], improvement of the robustness for more complex distortions [15,29,36], and a novel two-stage framework for complex distortions [13].

2.2. Differentiable Rendering

Differentiable rendering has been primarily developed for explicit 3D representations such as meshes, point cloud, and volume [12, 19, 24, 38]. While explicit 3D representations are commonly used due to their ease of manipulation and popularity in the downstream applications and 3D software, they have historically been precluded from deep learning pipelines due to the difficulty in back-propagating through mesh rasterization. Loper *et al.* [14] first proposed a general differentiable renderer named OpenDR, which can efficiently approximate derivatives with respect to 3D models. Later, SoftRasterizer [12], DiffRen [3] and Differentiable Ray-tracer [10] have emerged and devised different techniques to make rendering back-propagatable. In this paper, we use 3D mesh representation to be consistent with conventional 3D watermarking work [16, 18, 33, 34], and we employ DiffRen [3] in our differentiable rendering component.

2.3. Neural Network Architectures

For 3D data, PointNet [20] and PointNet++ [21] used multiple shared multi-layer perceptrons (MLPs) with transform layers, and have been widely adopted to extract features from 3D point clouds. The PointNet architectures can be easily extended to 3D mesh vertices. Convolutional mesh

autoencoders [22] applied Chebyshev convolution filters as well as barycentric mesh resampling. A 3D variational autoencoder, which handles both mesh connectivity and geometry, was introduced in [28]. Recently, a fully convolutional mesh autoencoder [40] allows variable-size mesh inputs.

In the 2D image domain, standard convolutional neural networks (CNN) [7] have shown significant progress on classifying objects, and have been aggressively extended to many other areas. The accuracy of image classification and object detection is further improved by deeper networks. U-Net [23] symmetrically connects encoder and decoder feature maps in variational autoencoder architectures, and shows improvements on image segmentation tasks.

2.4. Other Related Work

Evaluating perceptual quality of 3D meshes is an open problem with no standardized metrics. For many applications, using simple metrics such as Hausdorff distance [6] or root mean square error is sufficient, but these metrics tend to correlate poorly with the actual quality perceived by human vision. Visual quality measures [26] used an additional smoothness factor that correlates better with human vision than pure spatial measures. The same concept was further improved in [8], where a multi-scale mesh quality metric was proposed to rely on curvature statistics on different neighborhood scales. The limitations of these methods are non-differentiability and their reliance on purely 3D geometric data without taking into consideration other factors such as materials, textures or lighting. In [9], it was shown that using image-based quality assessment on rendered images of the 3D models is a viable strategy that can implicitly handle all surface and material properties for evaluation.

3. Method

Fig. 2 shows our deep 3D watermarking pipeline, which consists of four modules: a) an encoder; b) a distortion layer; c) a differentiable renderer; and d) a decoder. We will go deep into each module in the following sections.

3.1. Definitions

We denote by $\mathbb{M}(V, F, T, P)$ the input mesh. The vertices $V \in \mathbb{R}^{N_v \times C_v}$ contain N_v vertices and each vertex has C_v vertex elements such as 3D position, normal, vertex color, and 2D texture coordinate elements. The mesh faces $F \in \{0, \cdots, N_v - 1\}^{N_f \times C_f}$ contains vertex indices, where N_f is the number of faces and C_f is the number of face indices with value 3 for a triangular mesh and 4 for a quadrilateral mesh. $T \in \mathbb{R}^{H_t \times W_t \times C_t}$ store texture information, where W_t, H_t and C_t are width, height, number of color channels of the texture respectively. The mesh material color information $P \in \mathbb{R}^{10}$ includes ambient, diffuse, specular RGB color, and a shininess. We denote by

$M \in \{0,1\}^{N_b}$ the binary message with length N_b to be embedded into the mesh.

3.2. Encoder

Changing mesh faces F produces undesirable artifacts and the material color information P is too small to hide information. Thus, we decide to embed message in vertex and texture components. Note that we only encode message into normal and texture coordinate elements in the vertex, since changing vertex position creates degenerated triangles and prohibits backpropagation. We could embed message in either component or both of them. At the encoder, we replicate each message bit N_v times to construct a tensor with dimension $N_v \times N_b$ to embed message in the vertex component. To embed the message in texture, we replicate each message bit $H_t \times W_t$ times to construct a tensor with dimension $H_t \times W_t \times N_b$. After this, we concatenate the message tensors with input vertices and/or textures. We define this concatenated tensor as $V_m \in \mathbb{R}^{N_v \times (C_v + N_b)}$ and $T_m \in \mathbb{R}^{H_t \times W_t \times (C_t + N_b)}$ for vertices case and texture case respectively. Then we get watermarked vertices $V_e = E_G(V_m) \in \mathbb{R}^{N_v \times C_v}$ and watermarked texture $T_e = E_T(T_m) \in \mathbb{R}^{H_t \times W_t \times C_t}$, where E_G and E_T are vertex and texture message embedding neural network respectively. For E_G, we made the PointNet architecture [20] to be fully-convolutional, and also used a CNN-based architecture for E_T similar to [42]. Fig. 3 and Fig. 4 show the comparisons of watermarked meshes and textures. Visually we could not tell the difference between watermarked/non-watermarked pairs.

3.3. Distortions

To make our watermarking system robust to different distortions, we add a distortion layer in our watermarking training pipeline. Several commonly used distortions are considered: 1) additive Gaussian noise with mean μ and standard deviation σ; 2) random axis-angle rotation with parameters (α, x, y, z); 3) random scaling with a parameter s; and 4) random cropping on *3D vertices*. Since all these distortions are differentiable, we could train our network end-to-end.

3.4. Differentiable Rendering

To train a 3D watermarking system that can extract messages from the 2D rendered images, we need a differentiable rendering layer. We leverage the state-of-the-art work in the field of differentiable rendering and follow the work from Genova *et al.* [3], which proceeds in three steps: 1) rasterization, which computes screen-space buffers per pixel and the barycentric coordinates of the pixel inside triangles; 2) deferred shading; 3) splatting, where each rasterized surface point is converted into a splat, centered at the pixel and colored by the corresponding shaded color. The approach computes smooth derivatives on the pixel grid

w.r.t the mesh vertices and per-vertex attributes. In this paper, we use the Phong reflection model [17] for shading. Our differentiable rendering examples are shown in Fig. 5.

In our pipeline, we assume there are a fixed number of point light sources positioned at L_P^i with intensity L_I^i. Given the lighting parameters L, camera matrices K, and meshes $\mathbb{M}$, we use our differentiable renderer to generate the output 2D images,

$$I = R_D(\mathbb{M}, K, L) \in \mathbb{R}^{H_r \times W_r \times 3}. \quad (1)$$

3.5. Decoder

We use a neural network D to retrieve a message M_r from the rendered image I, *i.e.*, $M_r = D(I)$. The decoder network D uses a global pooling layer after several convolution layers to allow variable size input images, and then several fully-connected layers. The last fully-connected layer has a fixed number of output nodes which is the length of the message (more details are in the supplementary material). To further improve the decoder performance on non-differentiable renderers such as commercial software [30], we could fine-tune or train the decoder alone with the specific rendered output. We will show a specific example, *i.e.*, real-time and physically-based renderers in Sec. 4.9.

The final message bits $M_{rb} \in \{0,1\}^{N_b}$ are from the binarization of the predicted message with Eq. 2.

$$M_{rb} = \text{clamp}(\text{sign}(M_r - 0.5), 0, 1). \quad (2)$$

Note that M_r is used for computing the message loss in Sec. 3.6, and M_{rb} is used for evaluating bit accuracy.

3.6. Losses

We model the objective of 3D-to-2D watermarking by optimizing: 1) the vertex watermarking loss, 2) the texture watermarking loss, 3) the 2D rendering loss, and 4) the message loss.

Among them, the vertex watermarking loss computes the distance between the watermarked mesh vertices V_e and the original mesh vertices V,

$$L_{vertex}(V, V_e) = \sum_i w^i \frac{\sum_\alpha |V^i[\alpha] - V_e^i[\alpha]|}{N_v C_v}, \quad (3)$$

where i indicates a component of vertices, such as normal or texture coordinates, and w^i is the weight for each component. By adjusting w_i, we can adjust the sensitivity of change for each vertex component. Similarly, texture loss computes the difference between the watermarked texture T_e and the original texture T,

$$L_{texture}(T, T_e) = \frac{\sum_\alpha |T[\alpha] - T_e[\alpha]|}{H_t W_t C_t}. \quad (4)$$

(a) Vertex Difference   (b) Not watermarked   (c) Watermarked   (d) Difference (4x)
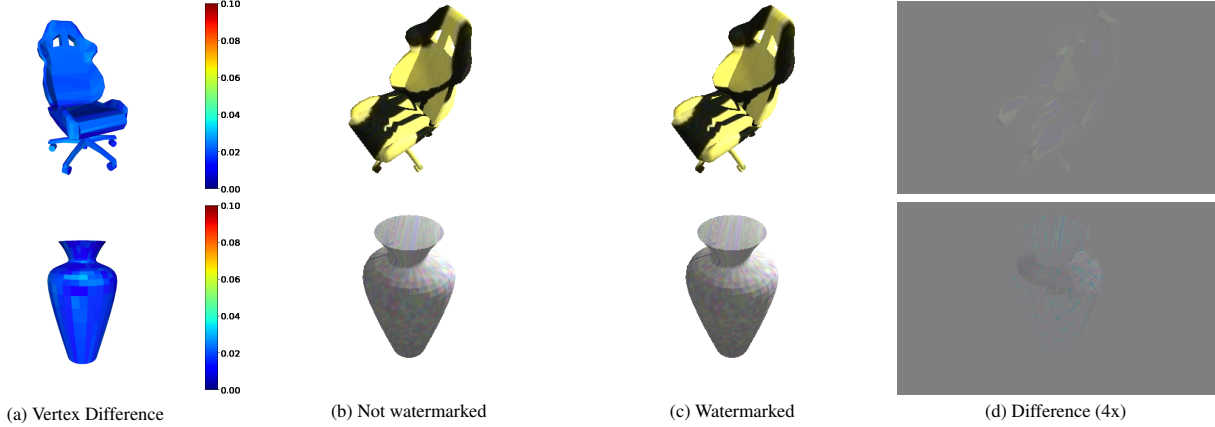
Figure 3. The differences of geometric elements of a mesh are color-coded and shown in (a). The rendered image from input mesh, the rendered image from watermarked mesh, and the (4x) difference between input and watermarked images in (b), (c), and (d) respectively.

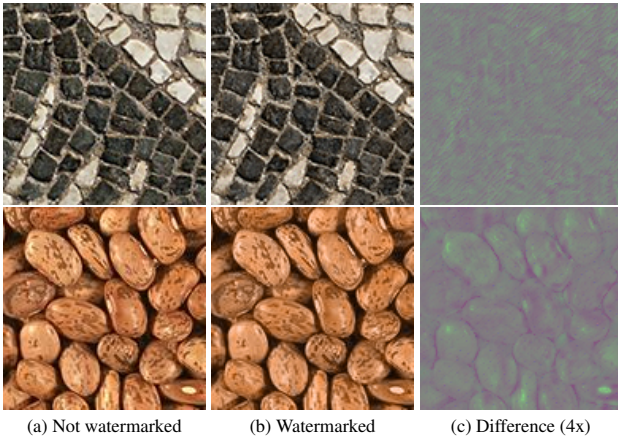(a) Not watermarked   (b) Watermarked   (c) Difference (4x)

Figure 4. The input texture, the watermarked texture, and the (4x) difference images are shown in (a), (b), and (c) respectively.
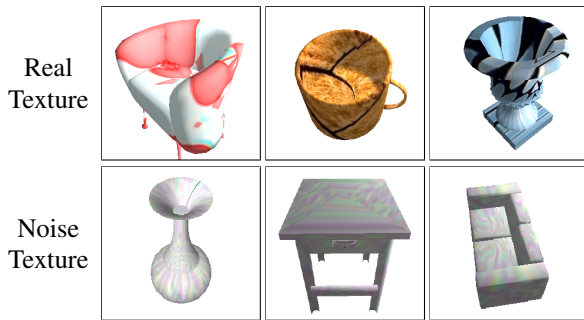

Figure 5. Examples of differentiable rendering results.

The 2D rendering loss calculates the difference between the rendered images I_o of the original meshes and the ones I_w of watermarked mesh

$$L_{image}(I_o, I_w) = \frac{\sum_\alpha |I_o[\alpha] - I_w[\alpha]|}{3H_w W_w}. \quad (5)$$

The message loss penalizes the decoding error, which is defined by the difference between the predicted message M_{rb} and the ground truth message M,

$$L_{message}(M, M_r) = \frac{\sum_\alpha |M[\alpha] - M_r[\alpha]|}{N_b}. \quad (6)$$

Finally, the total loss is the weighted sum of all losses,

$$L_{total} = \lambda L_{vertex} + \gamma L_{texture} + \\ \delta L_{image} + \theta L_{message} + \eta L_{reg}, \quad (7)$$

where L_{reg} is the regularization loss, $i.e.$, the L_2 sum of all network weights, λ, γ, δ, θ, and η are hyper-parameters.

4. Experiments

4.1. Dataset

We use ModelNet 40-class dataset [37] for training and testing. We normalize the input vertices' position elements $\{x, y, z\}$ to $[-1, 1]$, while keeping ratios between width, height and depth. Then, the input meshes are simplified to fix the number of triangles and vertices using the CGAL library [31] as shown in Fig. 6. Fixing the number of vertices and triangles in the dataset is for training efficiency and memory limitation. But our network could deal with meshes with different sizes by applying a global pooling layer in the vertex encoder architecture (the details are discussed in the supplementary material). We also manually filter out low quality simplified meshes. As a result, the number of meshes in ModelNet 40-class dataset is reduced to 5146 meshes for training and 1329 meshes for testing. Note that the original dataset has 9843 and 2468 meshes for training and testing, respectively. ModelNet dataset does not contain texture information, thus we allocate texture coordinates using spherical mapping and get texture from either Gaussian smoothing a white noise or randomly selecting and cropping a texture from an open texture dataset [41].
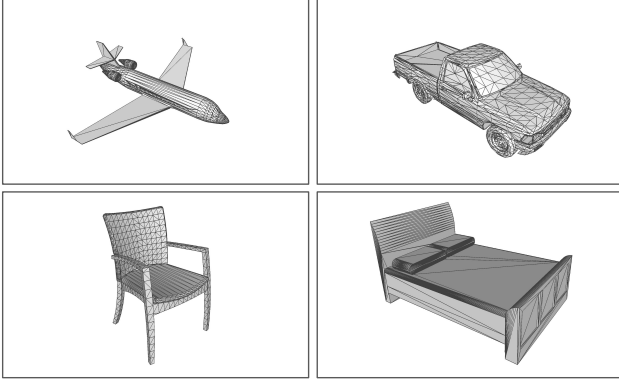
Figure 6. Examples of input (simplified) meshes.

4.2. Implementation Details

Mesh parameters The input meshes have $N_v = 5000$, $N_f = 5000$, $C_f = 3$, and $C_v = 5$. The input texture is with height 128, width 128 and number of channels 3. The rendered image is with width 600 and height 400.

Training and testing parameters For evaluation of our method, we randomly choose 100 meshes in the test dataset, and take the average of output values. We use batch size 4, learning rate 0.0001 for training. And also we set the weights in the total loss function as $\lambda = 2.0$, $\gamma = 1.0$, $\delta = 1.0$, $\theta \in [0.1, 2.0]$, and $\eta = 0.01$. For deciding hyper-parameters, we sweep broad ranges of hyper-parameters and manually choose the best.

Rendering parameters We use common camera parameters, which include camera location, camera look-at location, camera up, and field of view (fov_y). We use a point light source, the parameters are a point light location, colors, and attenuation factors. While training, camera and lighting parameters are *randomly generated in the given ranges*. In our experiment, we use *random camera locations* with ranges, $x = 0$, $y \in$ [-3, -2], $z \in [2, 4]$. Camera is always looking at the object center with $fov_y = 60°$. *The point light location is also randomly sampled* from a Gaussian distribution with mean $\mu = (2, 1, 2)$ and standard deviation $\sigma = 0.2$.

We put all the network architecture and training details in supplemental material. Note that the following experiments embed messages with 8 bits unless mentioned specifically.

4.3. Architectures

We investigated different architectures for the vertex encoder, the texture encoder, and the image decoder.

For the vertex encoder, we compared two architectures: 1) PointNet; and 2) fully convolutional PointNet (noted PointNet v2). Tab. 1 and Tab. 2 show the comparisons with and without a texture encoder respectively. As shown in the tables, both architectures showed similar performance on

bit accuracy, geometry L_1 differences, and rendered image quantitative metrics, however, *we chose the fully convolutional PointNet, since it can accept meshes of different sizes as an input at inference.*

Architectures	Bit Accuracy			Geometry L_1 Diff		Rendered Image	
	Best	μ	σ	Normal	Texcoord	PSNR	SSIM
PointNet	**0.8553**	0.7591	0.0789	0.0529	0.0805	**25.57**	**0.9425**
PointNet v2	0.8207	0.7213	0.0754	**0.0415**	**0.0745**	25.10	0.9416

Table 1. Comparisons of vertex encoder architectures with a texture encoder.

Architectures	Bit Accuracy			Geometry L_1 Diff		Rendered Image	
	Best	μ	σ	Normal	Texcoord	PSNR	SSIM
PointNet	**0.6837**	0.6151	0.0366	0.1873	0.0546	28.47	**0.9563**
PointNet v2	0.6616	0.6255	0.0216	**0.1506**	**0.0430**	**28.83**	0.9557

Table 2. Comparisons of vertex encoder architectures without a texture encoder.

For the texture encoder, we compare the network architecture used a CNN-based architecture [42] and a fully convolutional U-Net architecture as shown in Tab. 3. U-Net got better bit accuracy while HiddeN generated less visible watermarked texture. Note that [15] has the same encoder and decoder architecture with HiDDeN [42].

Architectures	Bit Accuracy			Texture		
	Best	μ	σ	L_1 Diff	PSNR	SSIM
CNN (HiDDeN)	0.8269	0.7614	0.0411	**0.03536**	**26.20**	0.9113
U-Net	**0.8474**	0.7451	0.1289	0.05692	23.55	**0.9143**

Table 3. Comparisons of texture encoder architectures.

Lastly, for the decoder, we evaluated HiDDeN [42] and two simple baselines – a standard CNN (4 layers) and a residual CNN (4 layers). As shown in Tab. 4, all the architectures can detect the message and among them, the HiDDeN decoder, probably benefited from its larger model size and capacity, performs better than the other models.

Architectures	Bit Accuracy		
	Best	μ	σ
Simple CNN (4-layers)	0.6396	0.6184	0.0415
Residual CNN (4-layers)	0.7390	0.6453	0.0332
CNN (HiDDeN)	**0.8269**	0.7614	0.0411

Table 4. Comparisons of decoder architectures.

4.4. Bit Accuracy vs. Message Length

We use message length as $N_b \in \{4, 8, 16, 32, 48\}$, and launch 10 experiments for each message length and show the relationship between bit accuracy and the length of message in Fig. 7. The blue line and areas show the average bit accuracy, 1σ, 2σ, and 3σ ranges, respectively. We could see that the bit accuracy drops when the number of bits increases.
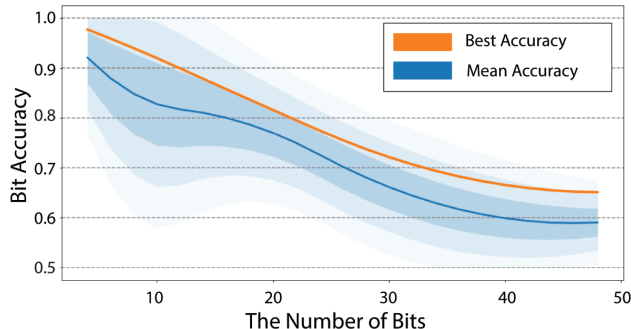
Figure 7. The relationship between bit accuracy and the length of message. We trained our pipeline 10 times with the different number of bits, then calculated best and mean accuracies as well as the standard deviations.

4.5. Message Embedding Strategies

As shown in Tab. 6, when using "vertex only" encoder, the watermarked mesh can hold less bits compared to "texture only" or "vertex + texture" encoder. Combining vertex and texture encoders shows better bit accuracy compared to the "texture only" encoder in higher numbers of bits.

4.6. Bit Accuracy vs. Mesh Quality

As shown in Tab. 5, there is a clear trade-off between bit accuracy and mesh quality. Higher bit accuracy results in higher L_1 difference as well as lower PSNR and SSIM, both in textures and rendered images. We could control this trade-off by changing the message loss weight θ.

4.7. Distortions

We evaluated the robustness of our methods to different distortions for example noise, rotation, scaling and cropping in Tab. 7. We could see that our method is quite robust to different distortions. More details about the experiments above are discussed in the supplementary material.

4.8. Comparison with Deep Image Watermarking

We also compared with a deep image watermarking method [42]. To make a fair comparison, we trained HiD-DeN encoder and decoder on our texture images. For our method, we used the trained HiDDeN image encoder as our texture encoder, fixed it during training, and only trained our vertex encoder and image decoder. After training our network, we generated the rendered images, and compared the performance of decoding message bits from the rendered images using HiDDeN decoder and our decoder. As shown in Tab. 8, our decoder could successfully decode message bits while the other showed nearly random bits. This shows the effectiveness of our end-to-end trainable 3D-2D watermarking framework.

4.9. Decoding from Other Renderers

In Tab. 9, row a-d show the bit accuracy of our decoder (without fine-tuning), decoding from the rendering results of multiple renderers. It includes our differentiable renderer, a real-time (RT) renderer, and two physically-based (PB) renderers. Note that although the bit accuracy is still low, it is not random. Our decoder could still recover some information. With technologies like channel coding, we could achieve higher bit accuracy by sacrificing some capacity, *i.e.*, the number of useful bits we embedded. Examples of rendering results are shown in Fig. 8.

Moreover, though we cannot train our framework end-to-end for non-differentiable rendering methods, we can fine-tune our decoder with their rendering outputs to improve the bit accuracy as shown in Tab. 9e. Here we created a dataset of 6000 watermarked meshes for training and 1500 for testing, and then generated their rendering results from the Eevee renderer. Using the rendered images, we fine-tuned our decoder. Fig. 9 shows the ray traced images from an original mesh and a watermarked mesh as well as the difference between the two images.
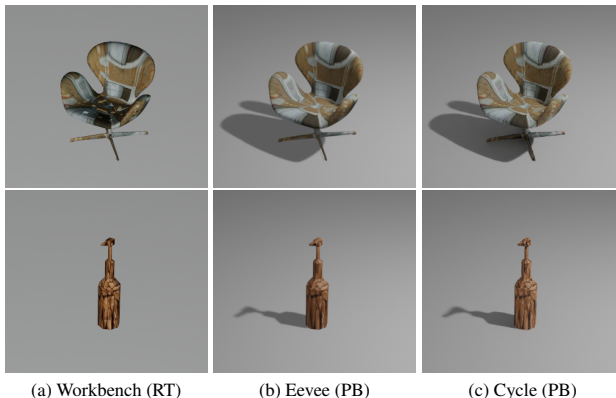

| | | |
| (a) Workbench (RT) | (b) Eevee (PB) | (c) Cycle (PB) |

Figure 8. Examples of different rendering outputs: (a) a RT renderer, (b) a PB renderer, and (c) a PB ray tracer.

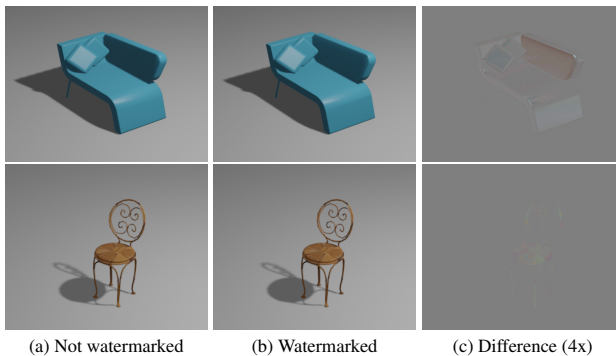| | | |
| (a) Not watermarked | (b) Watermarked | (c) Difference (4x) |

Figure 9. Examples of a ray traced image from an original mesh (a), a ray traced image from a watermarked mesh (b), and the (4x) difference image.

	Bit Accuracy			Geometry L_1 Diff		Texture Diff			Rendered		
θ	Best	μ	σ	Normal	Texture	L_1	PSNR	SSIM	L_1	PSNR	SSIM
0.01	0.5475	0.4997	0.0090	0.0475	**0.0537**	**0.0042**	**46.57**	**0.9849**	**0.0055**	**36.91**	**0.9700**
0.1	0.7947	0.7256	0.0396	0.0472	0.0621	0.0056	41.11	0.9844	0.0062	35.67	0.9614
1.0	0.9262	0.8857	0.0255	**0.0437**	0.0699	0.0186	30.74	0.8639	0.0063	35.35	0.9609
2.0	0.9325	**0.8921**	0.0432	0.2071	0.2404	0.0229	29.37	0.8062	0.0071	34.70	0.9603
5.0	0.8775	0.8562	0.0146	0.2563	0.2873	0.0363	25.45	0.6626	0.0088	32.90	0.9552
10.0	**0.9362**	0.8288	0.1135	0.1041	0.1810	0.0549	22.42	0.5088	0.0105	31.10	0.9426

Table 5. The message loss weight shows a trade-off relationship with the differences between the input and watermarked meshes. Higher message loss weight provides higher bit accuracy, however the output watermarked meshes are changed more.

	4 Bits	8 Bits	16 Bits	32 Bits	48 Bits
Vertex Only	0.7646	0.6289	0.5717	0.5627	0.5370
Texture Only	**0.9554**	**0.8970**	0.7975	0.6364	0.5679
Vertex + Texture	0.9208	0.8480	**0.8000**	**0.6441**	**0.5905**

Table 6. The comparisons of bit accuracies and different message embedding strategies. The accuracies are the averages of 10 experiments.

Distortion Type	Bit Accuracy
No Distortion	0.9046
Noise ($\sigma = 0.01$)	0.9036
Rotation ($\pm\pi/6$)	0.9028
Scaling ($< 25\%$)	0.8953
Cropping ($< 20\%$)	0.8840

Table 7. The effect of different distortions.

	HiDDeN [42]	Our Method
Bit Acc	0.5132	0.8213

Table 8. The comparison with a deep image watermarking method.

	Renderer Type	Bit Acc
a	Differentiable Renderer	0.7490
b	Workbench (RT) Renderer	0.6341
c	Cycle (PB) Ray Tracer	0.6634
d	Eevee (PB) Renderer	0.5973
e	Fine-tuned (Eevee)	0.7679

Table 9. The accuracy of decoded message bits from different renderers including a RT renderer, PB renderers (a-d) by our default decoder, and a fine-tuned decoder (e).

4.10. User Study on Rendered Images

The quantitative metrics, such as L_1, PSNR, and SSIM, may not be able to fully reflect human perceptions. To assess whether users can distinguish original and watermarked images, we conduct two crowd-sourced user studies through Amazon's Mechanical Turk. We show the original and watermarked renderings side-by-side, in shuffled left-and-right order. We then ask participants to determine whether the two images are "identical" or "not identical" in a few sec-

onds. A total of 200 pairs in each case were rated, where every pair was independently rated by 5 raters.

	Same Rating	PSNR	SSIM	L_1 Diff
Texture	74.07 %	47.76	0.997	0.0009
Ray Tracing	78.06 %	54.52	0.999	0.0004

Table 10. User study about watermarked and non watermarked images. Raters were asked whether the two images are "identical" or "not identical".

Tab. 10 shows that in 74% of the noise-textured cases, participants responded "identical". For the user study, we used the rendered images from our differentiable renderer. For the ray traced images, in 78% of the cases, participants responded "identical".

5. Conclusion, Limitations, and Future Work

We study the problem of 3D-to-2D watermarking, a key step towards an automatic and secure solution to protect copyright and integrity for 3D models from their 2D rendered images. Our contributions include the first end-to-end trainable model resistant to various 3D distortions and manipulations with quantitative and qualitative evaluations of its robustness, capacity and accuracy. In addition, our method generalizes 3D watermarking use cases. We further analyze the efficiency of different architectures and explore the performance of our framework on several real-world renderers.

It is worth noting that more work needs to be done to make it more applicable to real-world cases. For example, our decoder needs to be re-trained for totally different types of rendering techniques. Improving and generalizing our decoder is an interesting research topic. For future work, the robustness to non-differentiable 3D distortions and making the decoder to be able to retrieve messages encoded by traditional encoders are worth studying. Another direction is to explore better differentiable renderers, so that the framework can simulate and generalize to more complex 3D models or shading effects.

References

[1] Mahdi Ahmadi, Alireza Norouzi, Nader Karimi, Shadrokh Samavi, and Ali Emami. ReDMark: Framework for residual diffusion watermarking based on deep networks. *Expert Syst. Appl*, 146, 2020. 1, 3

[2] Jihane Bennour and Jean-Luc Dugelay. Toward a 3d watermarking benchmark. In *MMSPW*, 2007. 3

[3] Kyle Genova, Forrester Cole, Aaron Maschinot, Aaron Sarna, Daniel Vlasic, and William T Freeman. Unsupervised training for 3d morphable model regression. In *CVPR*, 2018. 3, 4

[4] Mohamed Hamidi, Aladine Chetouani, Mohamed El Hazithi, Mohammed El Hassouni, and Hocine Cherifi. Blind robust 3d mesh watermarking based on mesh saliency and wavelet transform for copyright protection. *Information*, 10(2), 2019. 3

[5] Jong-Uk Hou, Do-Gon Kim, and Heung-Kyu Lee. Blind 3d mesh watermarking for 3d printed model by analyzing layering artifact. *TIFS*, 12(11):2712–2725, 2017. 3

[6] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the hausdorff distance. *TPAMI*, 15(9):850–863, 1993. 3

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 3

[8] Guillaume Lavoué. A multiscale metric for 3d mesh visual quality assessment. *CGF*, 30(5):1427–1437, 2011. 3

[9] Guillaume Lavoué, Mohamed Chaker Larabi, and Libor Váša. On the efficiency of image metrics for evaluating the visual quality of 3d models. *TVCG*, 22(8):1987–1999, 2016. 3

[10] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *TOG*, 37(6), 2018. 3

[11] Jing Liu, Yajie Yang, Douli Ma, Wenjuan He, and Yinghui Wang. A novel watermarking algorithm for three-dimensional point-cloud models based on vertex curvature. *IJDSN*, 15(1), 2019. 3

[12] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *ICCV*, 2019. 2, 3

[13] Yang Liu, Mengxi Guo, Jian Zhang, Yuesheng Zhu, and Xiaodong Xie. A novel two-stage separable deep learning framework for practical blind watermarking. In *MM*, 2019. 3

[14] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *ECCV*, 2014. 3

[15] Xiyang Luo, Ruohan Zhan, Huiwen Chang, Feng Yang, and Peyman Milanfar. Distortion agnostic deep watermarking. In *CVPR*, 2020. 1, 3, 6

[16] Ryutarou Ohbuchi, Akio Mukaiyama, and Shigeo Takahashi. A frequency-domain approach to watermarking 3d shapes. *CGF*, 21(3):373–382, 2002. 3

[17] Bui Tuong Phong. Illumination for computer generated pictures. *ACM Comm.*, 18(6), 1975. 4

[18] Emil Praun, Hugues Hoppe, and Adam Finkelstein. Robust mesh watermarking. In *SIGGRAPH*, 1999. 1, 3

[19] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 3

[20] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 3, 4

[21] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017. 3

[22] Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J. Black. Generating 3d faces using convolutional mesh autoencoders. In *ECCV*, 2018. 3

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 3

[24] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. Deepvoxels: Learning persistent 3d feature embeddings. In *CVPR*, 2019. 3

[25] Jeongho Son, Dongkyu Kim, Hak-Yeol Choi, Han-Ul Jang, and Sunghee Choi. Perceptual 3d watermarking using mesh saliency. In *ICISA*, 2017. 3

[26] Olga Sorkine, Daniel Cohen-Or, and Sivan Toledo. High-pass quantization for mesh encoding. In *SGP*, 2003. 3

[27] support@treatstock.com. Watermark3d. https://www.watermark3d.com. 1

[28] Qingyang Tan, Lin Gao, Yu-Kun Lai, and Shihong Xia. Variational autoencoders for deforming 3d mesh models. In *CVPR*, 2018. 3

[29] Matthew Tancik, Ben Mildenhall, and Ren Ng. Stegastamp: Invisible hyperlinks in physical photographs. In *CVPR*, 2020. 3

[30] The Blender Foundation. Blender. https://www.blender.org, November 2020. 4

[31] The CGAL Project. CGAL: The computational geometry algorithms library. https://cgal.org, September 2020. 5

[32] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *CVPR*, 2017. 2

[33] Francesca Uccheddu, Massimiliano Corsini, and Mauro Barni. Wavelet-based blind watermarking of 3d models. In *MM&Sec*, 2004. 3

[34] Kai Wang, Guillaume Lavoue, Florence Denis, and Atilla Baskurt. A comprehensive survey on three-dimensional mesh watermarking. *TMM*, 10(8):1513–1527, 2008. 1, 3

[35] Bingyang Wen and Sergul Aydore. ROMark: A robust watermarking system using adversarial training. *arXiv preprint arXiv:1910.01221*, 2019. 3

[36] Eric Wengrowski and Kristin Dana. Light field messaging with deep photographic steganography. In *CVPR*, 2019. 3

[37] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d ShapeNets: A deep representation for volumetric shapes. In *CVPR*, 2015. 5

[38] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *TOG*, 38(6), 2019. 2, 3

[39] Kevin A. Zhang, Alfredo Cuesta-Infante, Lei Xu, and Kalyan Veeramachaneni. SteganoGAN: High capacity image steganography with gans. *arXiv preprint arXiv:1901.03892*, 2019. 3

[40] Yi Zhou, Chenglei Wu, Zimo Li, Chen Cao, Yuting Ye, Jason Saragih, Hao Li, and Yaser Sheikh. Fully convolutional mesh autoencoder using efficient spatially varying kernels. *arXiv preprint arXiv:2006.04325*, 2020. 3

[41] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. *arXiv preprint arXiv:1805.04487*, 2018. 5

[42] Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. Hidden: Hiding data with deep networks. In *ECCV*, 2018. 1, 3, 4, 6, 7, 8