MeshAdv: Adversarial Meshes for Visual Recognition Supplementary Material

Chaowei Xiao [*1] Dawei Yang [*1,2] Jia Deng [2] Mingyan Liu [1] Bo Li [3]
[1] University of Michigan, Ann Arbor
[2] Princeton University
[3] UIUC

A. Formulation of Differentiable Rendering

A physically based renderer R computes a 2D image $I = R(S; P, L)$ with camera parameters P, a 3D object S and lighting parameters L by approximating physics, *e.g.* the rendering equation [1, 2]. A differentiable renderer makes such computation differentiable w.r.t. the input S, P, L by making assumptions on illumination models and surface reflectance, and simplifying the ray-casting process. Following common practice, we use 3D triangular meshes for object representation, Lambertian surface for surface modeling, directional lighting with a uniform ambient for illumination, and ignore interreflection and shadows. Here, we further explain the details regarding 3D mesh representation $S = (V, F, T)$, illumination model L and camera parameters P used in differentiable rendering in this work.

For a 3D object S in 3D triangular mesh representation, let V be the set of its n vertices in 3D space, and F be the indices of its m faces:

$$V = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_n \in \mathbb{R}^3\}, \qquad F = \{\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_m \in \mathbb{N}^3\} \tag{S1}$$

For textures, traditionally, they are represented by 2D texture images and mesh surface parameterization such that the texture images can be mapped onto the mesh's triangles. For simplicity, here we attach to each triangular face a single RGB color as its reflectance:

$$T = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \cdots, \boldsymbol{t}_m \in {\mathbb{R}^+}^3\} \tag{S2}$$

For illumination model, we use k directional light sources plus an ambient light. The lighting directions are denoted L_{dir}, and the lighting colors (in RGB color space) are denoted as L_{color} for directional light sources and $\boldsymbol{a}$ for the ambient light:

$$L_{\text{dir}} = \{\boldsymbol{l}^d_1, \boldsymbol{l}^d_2, \cdots \boldsymbol{l}^d_k \in \mathbb{R}^3\}, \qquad L_{\text{color}} = \{\boldsymbol{l}^c_1, \boldsymbol{l}^c_2, \cdots \boldsymbol{l}^c_k \in \mathbb{R}^3\} \tag{S3}$$

We put the mesh $S = (V, F, T)$ at the origin $(0, 0, 0)$, and set up our perspective camera following a common practice: the camera viewpoint is described by a quadruple $P = (d, \theta, \phi, \psi)$, where d is the distance of the camera to the origin, and θ, ϕ, ψ are azimuth, elevation and tilt angles respectively. Note that here we assume the camera intrinsics are fixed and we only need gradients for the extrinsic parameters P.

Given the above description, the 2D image produced by the differentiable renderer can be symbolized as follows:

$$I = \text{rasterize}(P, T \cdot \text{shading}(L, \text{normal}(V, F))) \tag{S4}$$

normal(·, ·) computes the normal direction $\boldsymbol{n}_i$ for each triangular face $\boldsymbol{f}_i$ in the mesh, by computing the cross product of the vectors along two edges of the face:

$$\boldsymbol{n}_i = \frac{(\boldsymbol{v}_{\boldsymbol{f}_i^{(1)}} - \boldsymbol{v}_{\boldsymbol{f}_i^{(2)}}) \times (\boldsymbol{v}_{\boldsymbol{f}_i^{(2)}} - \boldsymbol{v}_{\boldsymbol{f}_i^{(3)}})}{\|(\boldsymbol{v}_{\boldsymbol{f}_i^{(1)}} - \boldsymbol{v}_{\boldsymbol{f}_i^{(2)}}) \times (\boldsymbol{v}_{\boldsymbol{f}_i^{(2)}} - \boldsymbol{v}_{\boldsymbol{f}_i^{(3)}})\|_2} \tag{S5}$$

shading computes the shading intensity $\boldsymbol{s}_i$ on the face given the face normal direction $\boldsymbol{n}_i$ and lighting parameters:

[*]Alphabetical ordering; the first two authors contributed equally.

$$\boldsymbol{s}_i = \boldsymbol{a} + \sum_{i=1}^{k} \boldsymbol{l}_i^c \max(\boldsymbol{l}_i^d \cdot \boldsymbol{n}_i, 0) \tag{S6}$$

Given face reflectance $\boldsymbol{t}_i$ for each face i, we compute the color $\boldsymbol{c}_i$ of each face i by elementwise multiplication:

$$\boldsymbol{c}_i = \boldsymbol{t}_i \circ \boldsymbol{s}_i \tag{S7}$$

rasterize projects the computed face colors $\boldsymbol{c}_i$ in 3D space onto the 2D camera plane by raycasting and depth testing. We also cap the color values to $[0, 1]$.

For implementation, we use the off-the-shelf PyTorch implementation [4, 5] of the Neural Mesh Renderer (NMR) [3].

B. *meshAdv* on Classification

Creation of *PASCAL3D+ Renderings* For classification, we create *PASCAL3D+ renderings* using CAD models from PASCAL3D+ [6]. Those meshes are then scaled to $[-1, 1]$ and put into the scene. Then, we use Neural Mesh Renderer (NMR) to generate synthetic renderings using these unitized meshes with uniformly sampled random camera parameters: azimuth from $[0°, 360°)$, elevation from $[0°, 90°]$. As for lighting, we used a directional light and an ambient light for *PASCAL3D+ Renderings*. The direction is uniformly sampled in a cone such that the angle between the view and the lighting direction is less than $60°$.

In order to obtain the groundtruth labels, we map the object classes in PASCAL3D+ to the corresponding classes in the ImageNet. Next, we feed the synthetic renderings to DenseNet and Inception-v3 and filter out the samples that are misclassified by either network, so that both models have 100% prediction accuracy on our *PASCAL3D+ renderings*. We then save the rendering configurations for evaluation of *meshAdv*.

Additional Results for DenseNet Figure A shows the generated "adversarial meshes" against DenseNet, similar to Figure. 2 in the main paper.

C. Human Perceptual Study Procedures

We conduct a user study on Amazon Mechanical Turk (AMT) in order to quantify the realism of the "adversarial meshes" generated by *meshAdv*. We uploaded the adversarial images on which DenseNet and Inception-v3 misclassify the object. Participants were asked to classify those adversarial images to one of the two classes (the groundtruth class and the target class). The order of these two classes was randomized and the adversarial images appeared for 2 seconds in the middle of the screen on each trial. After disappearing, the participant had unlimited time to select the more feasible class according to her perception. For each participant, one could only conduct at most 50 trials, and each adversarial image was shown to 5 different participants.

References

[1] D. S. Immel, M. F. Cohen, and D. P. Greenberg. A radiosity method for non-diffuse environments. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 133–142, New York, NY, USA, 1986. ACM. ISBN 0-89791-196-2. doi: 10.1145/15922.15901. 1

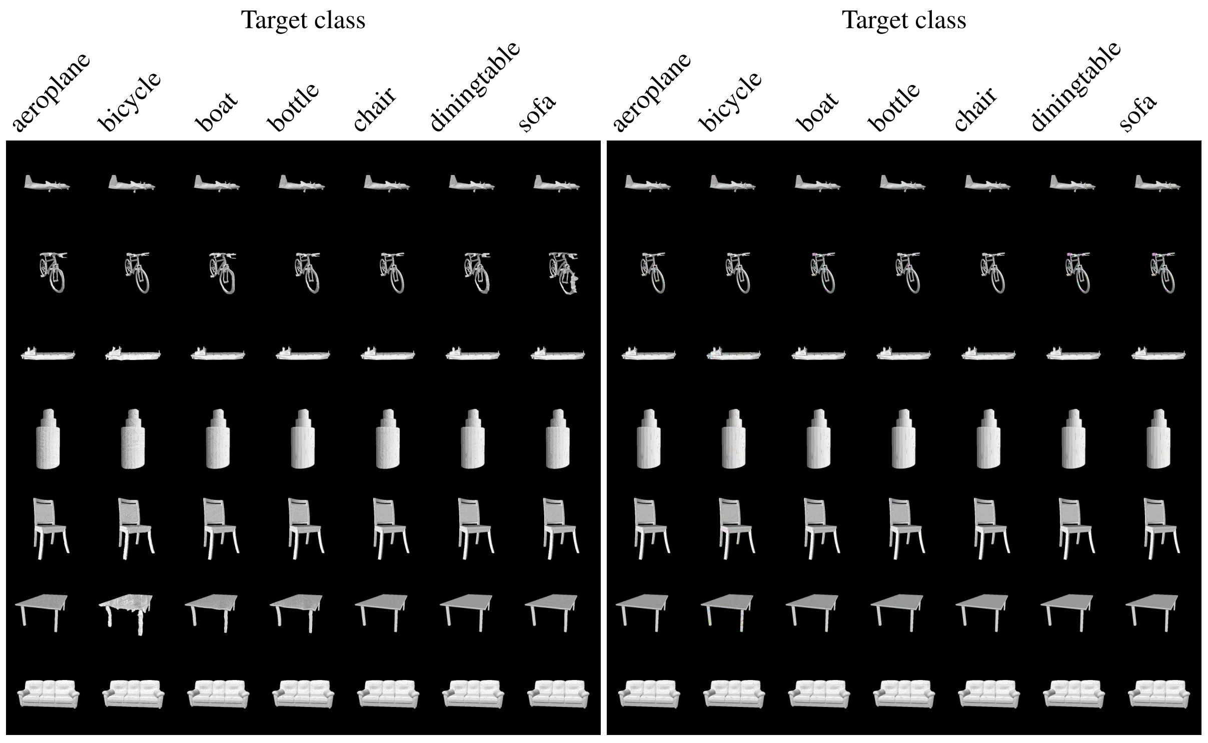
[2] J. T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM. ISBN 0-89791-196-2. doi: 10.1145/15922.15902. 1

[3] H. Kato, Y. Ushiku, and T. Harada. Neural 3d mesh renderer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[4] N. Kolotouros. Pytorch implememtation of the neural mesh renderer. `https://github.com/daniilidis-group/neural_renderer`, 2018. Accessed: 2018-09-10. 2

[5] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017. 2

[6] Y. Xiang, R. Mottaghi, and S. Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *Applications of Computer Vision (WACV), 2014 IEEE Winter Conference on*, pages 75–82. IEEE, 2014. 2



(a) Perturbation on Shape

(b) Perturbation on Texture

Figure A: Benign images (diagonal) and corresponding adversarial meshes generated by *meshAdv* on PASCAL3D+ shapes against DenseNet, targeting at different classes as shown on the top . (a) Presents the "adversarial meshes" by manipulating the shape; (b) by manipulating texture.