TEXTURE- AND SHAPE-BASED ADVERSARIAL ATTACKS FOR OVERHEAD IMAGE VEHICLE DETECTION

Mikael Yeghiazaryan[1] *Sai Abhishek Siddhartha Namburu*[1] *Emily Kim*[1] *Stanislav Panev*[1]
Celso de Melo[2] *Fernando De la Torre*[1] *Jessica K. Hodgins*[1]

[1]Carnegie Mellon University, USA [2]DEVCOM Army Research Lab

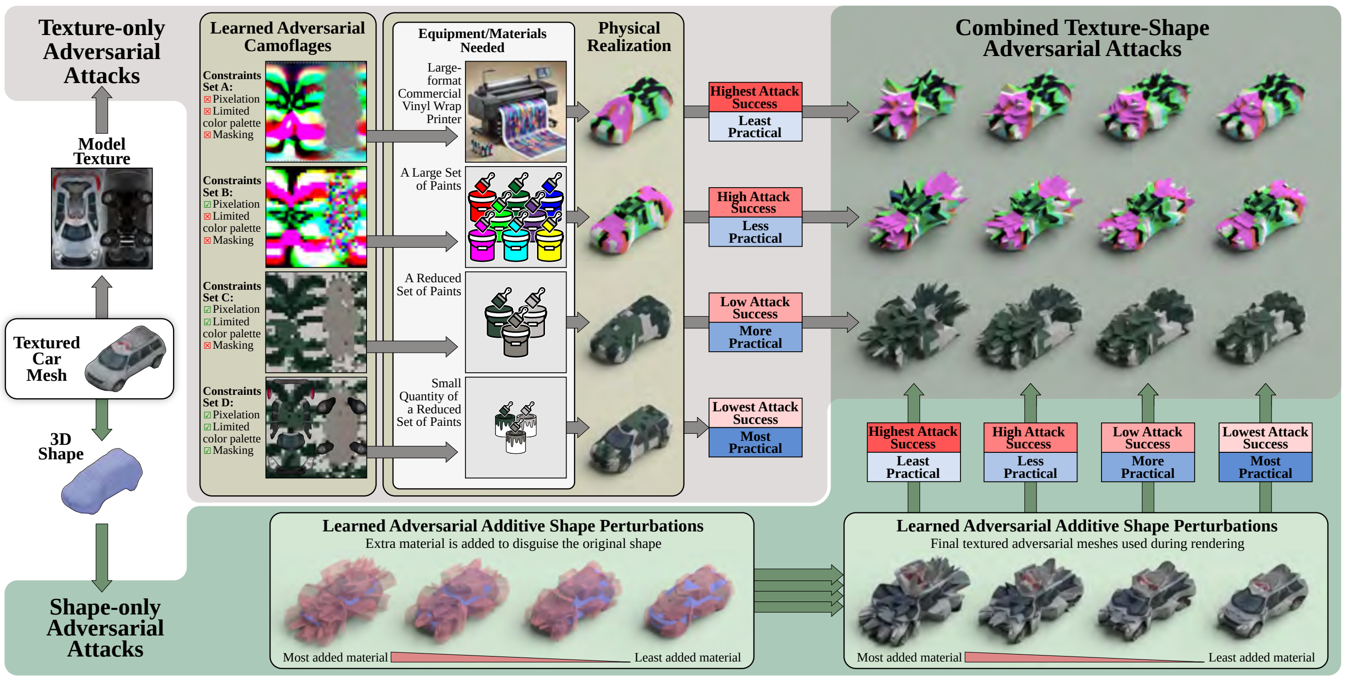

Texture-only Adversarial Attacks
Model Texture
Textured Car Mesh
3D Shape
Shape-only Adversarial Attacks
Learned Adversarial Camoflages
Constraints Set A: ☒ Pixelation ☒ Limited color palette ☒ Masking
Constraints Set B: ☑ Pixelation ☒ Limited color palette ☒ Masking
Constraints Set C: ☑ Pixelation ☑ Limited color palette ☒ Masking
Constraints Set D: ☑ Pixelation ☑ Limited color palette ☑ Masking
Equipment/Materials Needed
Large-format Commercial Vinyl Wrap Printer
A Large Set of Paints
A Reduced Set of Paints
Small Quantity of a Reduced Set of Paints
Physical Realization
Highest Attack Success
Least Practical
High Attack Success
Less Practical
Low Attack Success
More Practical
Lowest Attack Success
Most Practical
Combined Texture-Shape Adversarial Attacks
Learned Adversarial Additive Shape Perturbations
Extra material is added to disguise the original shape
Most added material
Least added material
Learned Adversarial Additive Shape Perturbations
Final textured adversarial meshes used during rendering
Most added material
Least added material

Fig. 1: We enforced various constraints on common adversarial attacks to improve their implementation in the real world. Our pipeline perturbs objects' texture, shape, or both. The latter demonstrates superior efficacy in deceiving object detectors.

ABSTRACT

Detecting vehicles in aerial images is difficult due to complex backgrounds, small object sizes, shadows, and occlusions. Although recent deep learning advancements have improved object detection, these models remain susceptible to adversarial attacks (AAs), challenging their reliability. Traditional AA strategies often ignore practical implementation constraints. Our work proposes realistic and practical constraints on texture (lowering resolution, limiting modified areas, and color ranges) and analyzes the impact of shape modifications on attack performance. We conducted extensive experiments with three object detector architectures, demonstrating the performance-practicality trade-off: more practical modifications tend to be less effective, and vice versa. We release both code and data to support reproducibility at `https://github.com/humansensinglab/texture-shape-adversarial-attacks`.

Index Terms— adversarial attacks, remote sensing, object detection

1. INTRODUCTION

Robust object detection in aerial and satellite images is vital for automating critical tasks such as traffic management, urban planning, and disaster response. State-of-the-art detectors, such as YOLO [1] and RetinaNet [2], which are based on deep neural networks (DNN), have become foundational in this domain. However, recent studies such as Szegedy *et al.* [3] have revealed that DNNs can be susceptible to adversarial examples. Given the importance of these applications, understanding this vulnerability is crucial, especially in object detection in Remote Sensing Imagery (RSI). Furthermore, there are scenarios where utilizing AAs to impede vehicle detection by computer vision systems in overhead images could offer strategic advantages, such as military camouflage.

Our primary objective is to investigate the resilience of object detectors against adversarial vehicles in RSI scenarios under realistic constraints. Traditional AA strategies often neglect the physical implementation constraints, focusing solely on task performance. For example, adversarial texture patterns typically resemble those depicted in Figure 1 (*Texture-only attacks - Constraints Set A*). However, it is essential to consider how such complex patterns can be produced practically in the physical world. Their creation often requires specialized equipment, such as expensive vinyl wrap printers, and trained professionals to install them on vehicle surfaces.

In this study, we propose a set of constraints to ensure attacks are feasible for implementation on vehicles, effectively camouflaging them. We define a *practical adversarial mesh* as a mesh modified in texture and shape such that replicating the modifications in real life requires minimal resources or specialized equipment. We consider practicality based on installation cost, difficulty, and operation. While constrained attacks are less effective than traditional AAs [4], they offer better practicality. Shape-only attacks are less effective than unconstrained texture attacks, but combining constrained texture with shape modifications improves performance, reaching levels similar to unconstrained texture attacks (Figure 1).

Our work contributes in several ways. (1) We introduce constrained AAs for shape and texture, designed to create practical 3D camouflages capable of deceiving object detectors in RSI. These constraints facilitate a more straightforward implementation compared to unconstrained camouflages. (2) We thoroughly examine how practicality and adversarial performance relate, finding that they have an inverse relationship. (3) We developed a tool for generating synthetic overhead images, contributing to the creation of synthetic datasets.

2. RELATED WORK

Adversarial attacks (AAs) have become central to computer vision research. Szegedy *et al.* [3] introduced AAs to expose vulnerabilities in deep learning models. Research has since focused on generating adversarial examples and divided AAs into digital and physical categories [5]. Digital AAs modify image pixels imperceptibly [6], while physical AAs manipulate objects in the physical world [7]. A hybrid approach, *simulated AAs*, tests perceptible attacks in simulated environments [8]. Our work aligns with simulated AAs, using synthetic data and realistic physics-based rendering for testing.

Recent studies also explore adversarial 3D geometry, mainly in autonomous driving using point clouds and LIDARs [9, 10]. Unlike those works, our focus is solely on RGB data in remote sensing imagery (RSI), where adversarial attacks are underexplored [11]. In RSI, the demand for robust automation has risen [12], driving research on AAs. Many attacks are impractical, so we focus on realistic adversarial camouflages with real-world constraints. One of the few studies addressing physical aerial adversarial attacks is [11], which uses adversarial patches to target vehicle detectors. Our approach differs in applying stricter constraints to adversarial camouflages and vehicle modifications, while applying these modifications to the entire vehicle and at a lower geo-spatial resolution. Additionally, instead of real-world tests, we evaluated our camouflages on highly realistic synthetic data produced by a physics-based renderer.

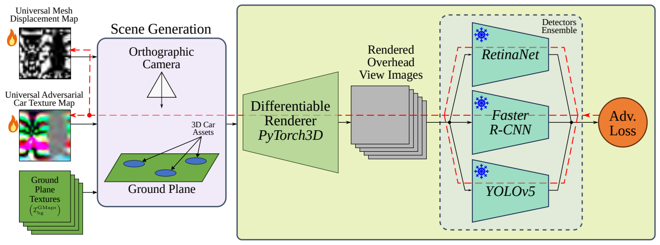

Universal Mesh Displacement Map
Universal Adversarial Car Texture Map
Ground Plane Textures $\left(\mathcal{I}_{\text{bg}}^{\text{GMaps}}\right)$
Scene Generation
Orthographic Camera
3D Car Assets
Ground Plane
Differentiable Renderer PyTorch3D
Rendered Overhead View Images
Detectors Ensemble
RetinaNet
Faster R-CNN
YOLOv5
Adv. Loss

Fig. 2: Our pipeline for adversarial attacks on an ensemble of object detectors. The back-propagation path is illustrated by the red dashed line. During inference, we evaluate each model independently.

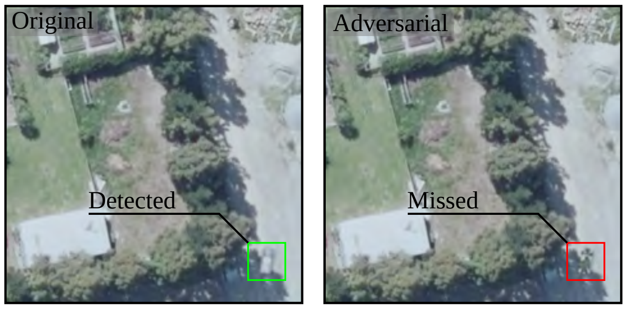


Fig. 3: Left: original image, right: corresponding adversarial image generated with Blender.

3. METHOD

3.1. PyTorch3D Data Generation

PyTorch3D (PT3D) data is generated using PyTorch3D [13] with 3D vehicle meshes from a GAN-based generator. We adapt and retrain the Textured 3D GAN (T3GAN) [14] to enable semantic segmentation map sampling, producing a set of meshes $\mathcal{M}$. Using GMaps backgrounds $\mathcal{I}_{\text{bg}}^{\text{GMaps}}$ and vehicle meshes $\mathcal{M}$, the differentiable renderer (DR) generates images $I = \text{DR}\left(I_{\text{bg}}^{\text{GMaps}}, M\right)$, where M includes texture T and shape S. Post-processing steps like blurring and anti-aliasing enhance realism. M can include either original or adversarial meshes, producing "original" or "adversarial" images, respectively. The pipeline supports synthetic dataset creation and ground-truth annotations, with adversarial attacks using optimized S_{adv} or T_{adv}. We use this data to train vehicle detectors and to optimize adversarial attacks.

3.2. Adversarial Optimization Pipeline

Each cycle of the attack uses PT3D to generate a batch of adversarial images $I_b = \{I_1, \dots, I_{N_b}\}$ such that $I_k = \text{DR}\left(I^{\text{GMaps}}_{\text{bg},k}, M_k\right), \forall\, k \in [1, \dots, N_b]$, where I_k is the k-th image from the batch, $I^{\text{GMaps}}_{\text{bg},k}$ is the k-th background image sampled from the GMaps dataset, M_k is a randomly selected mesh with shape and texture components S_k and T_k, and DR is a differentiable renderer following Section 3.1. Depending on the optimized entity, either the most recent S_{adv} or T_{adv} replaces the corresponding counterpart in M_k at the beginning of each iteration. During the attack, we ensure each image contains only one vehicle to avoid producing meshes that rely on multiple camouflages being in close proximity. The attack aims to create independently effective adversarial meshes.

Let F_i be the objective function used to train a detector model D_i. We supply I_b to D_i, producing predictions $y_{\text{pred}} = D_i(I_b)$. We then minimize a weighted loss function for an ensemble of models: $\mathcal{L} = \sum_i \lambda_i \mathbb{E}\left[F_i(D_i(I_b), y_{\text{gt}})\right], M^{\star}_{\text{adv}} = \arg\min_M \mathcal{L}(M)$, where y_{gt} are the ground-truth object locations, manually set to $\emptyset$ for adversarial attack training. We use coefficients λ_i such that the initial loss values F_i are in the same order of magnitude.

Table 1: Comparison of attack practicality with prior works. Texture practicality is the first score, shape practicality is the second. Sequential and parallel combined attacks yield identical final results. Full table is in Section S6 in the Supplementary Material.

	Camouflage	PC	DI	DO	Score	Notes
Other	Du *et al.* (ON) [11]	$+0$	$+0$	$+0$	3	Small AA area
	Du *et al.* (OFF) [11]	00	$+0$	-0	0	Limited mobility
	DTA [15]	-0	-0	$+0$	-1	Special equipment
Our	T-U	-0	-0	-0	-3	Special equipment
	T-Ma	-0	-0	$+0$	-1	Special equipment
	T-PixFcMa	$+0$	$+0$	$+0$	$+3$	Lim. color stickers
	S-O	$0-$	$0-$	$0-$	-3	Shape modification
	C-Fc	$--$	$--$	$--$	-6	Spec. eq./shape mod.

3.3. Texture-based Attacks

In texture-based attacks, we optimize a universal texture map applied to all meshes. While (**u**)nconstrained adversarial textures (abbreviated as "U") achieve excellent performance, they are impractical for real-world use. We introduce constraints to reflect practical implementation limitations: *Spatial Resolution*, *Spatial Restriction*, and *Color Restriction*.

Spatial Resolution. Applying iridescent patterns to irregular shapes such as vehicle surfaces is often challenging. We impose a texture (**pix**)elization (abbreviated as "Pix") constraint with block sizes of 16×16 px to ensure practical resolution, corresponding to approximately 15 cm on vehicle rooftops. We implement this by storing the adversarial texture as a $32 \times 32 \times 3$ tensor, then upscaling it to the original size of $512 \times 512 \times 3$ via nearest-neighbor interpolation.

Spatial Restriction. Another notable limitation is the need for vehicle camouflage to not hinder vehicle operation. We restrict the camouflage to specific areas of the vehicle using segmentation (**ma**)sks (abbreviated as "Ma"). Certain parts, such as windows, remain free from camouflage. The segmented adversarial texture map is given by $T_{\text{seg}} = T_{\text{or}} \cdot (1 - T_{\text{mask}}) + T_{\text{adv}} \cdot T_{\text{mask}}$, and T_{or} is the original texture.

Color Restriction. Our strictest constraint limits the number of colors in the adversarial texture map, implemented in two ways: 1) fixing the color count and (**l**)earning both the (**c**)olors and their placement ("Lc"), or 2) (**f**)ixing the (**c**)olors and optimizing their placement only ("Fc"). Unlike softer constraints such as the *non-printability score*, this enforces strict color limits, a concept largely unexplored in prior works. To enforce this constraint, the color of each pixel is determined during optimization by predicting a probability distribution $p(c)$ over a fixed set of colors. This distribution is sharpened using a double softmax $s(\cdot)$ to amplify the most probable color: $p_A(c) = s(s(p(c)))$. The pixel color is initially set to $\mathbb{E}[p_A(c)]$, approximating the mode color. After optimization, each pixel is assigned $\arg\max_{c_i} p(c_i)$, producing a camouflage that satisfies the color constraint. Additional details can be found in Section S6.1 in the Supplementary Material.

3.4. Shape-based Attacks

We optimize a universal perturbation in shape-based attacks using a 2D displacement map from a common UV map. Deformations extend outward from the mesh center, preserving the original shape. We enforce *Symmetry* for balanced mass and *Perturbation Magnitude* (PM) to limit deformation.

3.5. Combined Attacks

We also conduct combined attacks where both texture and shape are optimized. These can be performed sequentially or in parallel. In *sequential combined attacks*, we first optimize the texture map and then perform a shape-based attack using the adversarial texture. This allows us to evaluate the performance of both attacks sequentially. In *parallel combined attacks*, we alternate between optimizing the texture and shape. Each entity is optimized for a fixed number of steps n_{pll}, switching between them until the loss converges.

3.6. Computational Requirements

Experiments ran on a machine with an Intel Xeon Gold 6252 CPU, 755 GB RAM, and an NVIDIA Quadro RTX 6000 GPU (24 GB). Each attack took 3 hours per epoch, using up to 5 GB RAM and 12 GB GPU memory.

Table 2: Figures show mean values from synthetic models on PT3D and Blender data. "T", "R", "S", and "C" represent texture, random texture, shape, and combined attacks. Lc and Fc are mutually exclusive. PM* and Pr* denote optimal perturbation magnitude and practicality for shape attacks. See Table S3 in the Supplementary Material for the full table.

Attack	Constraints				PM*	Pr*	PT3D EASR	Blender EASR
	Pix	Lc	Fc	Ma				
T-U					—	—	95.77%	70.02%
T-Ma				✓	—	—	75.76%	44.43%
T-Pix	✓				—	—	94.75%	63.83%
T-PixLcMa	✓	✓		✓	—	—	68.39%	42.15%
T-PixFcMa	✓		✓	✓	—	—	12.70%	44.64%
R-PixFc	✓		✓		—	—	3.16%	20.24%
S-O	—	—	—	—	0.4	0.6	89.82%	78.86%
C-Fc (seq.)			✓		0.2	0.8	86.80%	70.37%
C-Fc (par.)			✓		0.2	0.8	87.11%	68.07%
C-PixFc (seq.)	✓		✓		0.2	0.8	86.83%	75.76%
C-PixFc (par.)	✓		✓		0.2	0.8	89.34%	77.86%

4. PRACTICALITY AND COMPARISONS

Our focus is not on enhancing AAs performance but exploring the impact of realistic constraints. Given the high effectiveness of unconstrained AAs, there is limited room for improvement. We evaluate our work based on three qualitative criteria: *production cost* (PC), *difficulty of installation* (DI), and *difficulty of operation* (DO), rated as good (+), insignificant (0), or bad (−). Practical camouflages score higher. See definitions in Section S6 in the Supplementary Material.

As shown in Table 1, T-PixFcMa is the most practical camouflage, despite lower effectiveness (Table 2). Du *et al.* [11] (ON) and EVD4UAV [16] achieve similar practicality scores, but their patches are too small for effective use in aerial imagery at our resolution. More details on score assignment are in Section S6 in the Supplementary Material. Results from Tables 1 to 2 highlight the trade-off between practicality and performance. While some studies excel in AA performance, they lack practicality. We conclude that optimizing for performance reduces practicality. For example, randomly generated camouflages (Table 2) show performance reduction that may not be justified without optimization. While a more rigorous analysis incorporating real data could be done (e.g., user studies for DO), this is beyond the scope due to limited resources.

5. EXPERIMENTS AND RESULTS

While Section 4 qualitatively compares practicality — considering production cost, installation, and operational complexity — this section provides a quantitative evaluation of effectiveness under different adversarial attack scenarios.

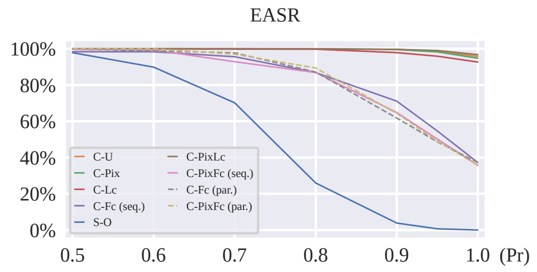


Fig. 4: EASR vs Practicality (Pr) curves for the shape-based and combined attacks on PT3D.

5.1. Evaluation metrics

We use effective attack success rate (EASR) to evaluate attack performance, defined as EASR $= \frac{|V_{d,m}|-|V_{m,d}|}{|V_{d,d} \cup V_{d,m}|}$, where the first subscript in $V_{i,j}$ indicates whether a vehicle was detected (d) or missed (m) in the original dataset, and the second subscript indicates whether it was detected (d) or missed (m) in the adversarial dataset after applying adversarial modifications. EASR is more conservative than the traditional ASR. More details can be found in Section S5.1 in the Supplementary Material.

5.2. Test Data

We describe below the datasets used to test the trained models and the camouflages generated from the attacks.

LINZ Dataset. We used the labeled LINZ dataset using LINZ Data Service aerial orthoimages[1]. The dataset was sampled into 384x384px images, resulting in 172 595 images with a $12.5\,\text{cm/px}$ resolution. We utilized only the "Small Vehicles" class for experiments, removing other labels but preserving all images. This dataset is denoted as $\mathcal{I}^{\text{LINZ}}$. A second dataset, $\mathcal{I}^{\text{LINZ}}_{\text{bg}}$, was created by removing vehicles from $\mathcal{I}^{\text{LINZ}}$ using "Inpaint Anything" [17].

GMaps Dataset The GMaps dataset provides background images $\mathcal{I}^{\text{GMaps}}_{\text{bg}}$ for the PyTorch3D [13] data generation pipeline (Section 3.1). We extracted Google Maps (GMaps) satellite images $\mathcal{I}^{\text{GMaps}}$ with matching coordinates to the LINZ aerial dataset using QGIS[2], ensuring a direct LINZ-GMaps correspondence. Real vehicles were manually removed from these images using an image editor, resulting in the background set $\mathcal{I}^{\text{GMaps}}_{\text{bg}}$, later used in the synthetic data generation process.

Blender Data Generation Blender data tests adversarial meshes in realistic settings using the Cycles physics-based renderer [18]. Unlike PT3D, Blender generates highly realistic but non-differentiable images, excluding it from adversarial optimization. Synthetic overhead images are produced by

[1]https://data.linz.govt.nz/layer/51926-selwyn-0125m-urban-aerial-photos-2012-2013/
[2]https://www.qgis.org

Blender using LINZ backgrounds $\mathcal{I}_{\text{bg}}^{\text{LINZ}}$ and meshes M, incorporating either original or adversarial textures and shapes.

5.3. Detection Models

We used RetinaNet [2], Faster R-CNN [19], and YOLOv5 [1] for vehicle center detection in RSI, representing one-stage, two-stage, and YOLO-family detectors. Each was trained on real and synthetic data, labeled "real" and "synt" models. On real test data, synthetic models achieved 50–63% AP, while real models exceeded 80%. More details can be found in Section S1 in the Supplementary Material. We attacked synthetic model ensembles, using one for inference.

5.4. Texture-based Attacks

We modify a vehicle's texture in *texture-based attacks* to conceal it from detectors, starting with random adversarial textures. There are twelve distinct texture settings, each evaluated on an ensemble of three synthetic models (RetinaNet, Faster R-CNN, YOLOv5). We compare these adversarial textures with four random texture maps (R-*). Results on PT3D test data are shown in Table 2. In Fc experiments, five colors are determined via K-means clustering of background pixels, while Lc experiments involve model-learned color placement.

To account for the distribution gap between real and synthetic datasets, we repeat the experiments using Blender. The results, presented in Table 2, show that constraints reduce performance but increase practicality. Example images are in Figure 3, with additional examples in the Supplementary Material. The performance-practicality trade-off remains even when testing on a different domain, like Blender. We also observe that unrestrained color distribution results in saturated colors, a common but underexplored issue in prior works. Further details are in Section S1 the Supplementary Material.

5.5. Shape-based Attacks

In *shape-based attacks*, we alter the geometry of the vehicle sacrificing practicality for improved adversarial performance. We link practicality, denoted as Pr, to the perturbation magnitude PM as $\text{Pr} = 1 - \text{PM}$, where $\text{PM} \in [0, 1]$. $\text{Pr} = 0$ indicates extreme mesh perturbation and $\text{Pr} = 1$ indicates no perturbation for a more realistic scenario.

Our goal is to minimize PM (maximize Pr) in shape-based attacks while maximizing EASR performance. We assess multiple attacks on synthetic models across a range of Pr values. Refer to the curve in Figure 4 under "Original" for details (utilizing original vehicle textures). Given an EASR-vs-Pr curve, we compute a practicality metric P1 as the harmonic mean of the EASR and Pr values for each point on the curve, *i.e.* $\text{P1} = 2\frac{\text{EASR}\cdot\text{Pr}}{\text{EASR}+\text{Pr}}$. We then pick the attack with the highest P1 as the optimal one. See the results, denoted as S-O in Table 2. The results suggest that when no adversarial texture is utilized along with a deformed vehicle geometry, the deformation must be large to achieve good performance, which is expensive to produce and difficult to install and operate, rendering it impractical.

5.6. Combined Attacks

In the *combined attacks*, we aim to boost the adversarial performance by attacking both mesh entitites: texture and shape. We discard the adversarial textures that use masking because a modified mesh is hard to segment into semantically meaningful parts. Thus, this leaves us with six adversarial texture maps for mesh-based attacks, each with its texture setting.

Sequential Attacks. We conduct six sequential attacks, each using one of the six adversarial textures from a non-masked setup. We follow the methodology in Section 5.5 to determine the optimal PM. The results, labeled C-* in Table 2, reveal significant insights. While the improvement over texture attacks without the fixed colors constraint (T-U, T-Pix, T-Lc, T-PixLc) is unjustified due to practicality loss, the fixed colors constraint (T-Fc and T-PixFc) justifies sacrificing some practicality for better performance. Compared to the huge practicality loss in shape-based attacks, the sequential blend of adversarial texture and shape is more efficient than shape-only attacks. We evaluate the resulting adversarial meshes on Blender data where the $\text{PM}^{\star} \neq 0$.

Parallel Attacks. We conduct parallel attacks using only two adversarial texture maps, Fc and PixFc, to reduce the PM even further than the sequential attacks. The results in Table 2 suggest no significant gain in switching to parallel attacks.

6. CONCLUSION

This study outlines a methodology for developing effective camouflage strategies to conceal vehicles in RSI. We also study the performance-practicality trade-off when implementing adversarial camouflages. While our findings could be misused, it is vital for the research community to be aware of the vulnerabilities in current models that we highlight. We show an inverse relationship between practicality and performance: unconstrained adversarial textures are highly effective against vehicle detection systems, while practical constrained textures are easier to implement but less effective. Shape-only attacks are also less impactful than texture attacks, but combining both can achieve results similar to unconstrained textures. Notably, sequential and parallel executions of shape and texture attacks demonstrate similar adversarial performance. Additionally, we present two pipelines for generating synthetic aerial images: using a differentiable renderer and a physics-based renderer.

Acknowledgements

We thank Brent Lance for valuable feedback and support. This work was supported in part by the Army Research Lab.

7. REFERENCES

[1] Glenn Jocher, "YOLOv5 by Ultralytics," May 2020. 1, 5

[2] Lin, Tsung-Yi and Goyal, Priya and Girshick, Ross and He, Kaiming and Dollar, Piotr, "Focal Loss for Dense Object Detection," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1, 5

[3] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013. 1, 2

[4] Suryanto, Naufal and Kim, Yongsu and Larasati, Harashta Tatimma and Kang, Hyoeun and Le, Thi-Thu-Huong and Hong, Yoonyoung and Yang, Hunmin and Oh, Se-Yoon and Kim, Howon, "ACTIVE: Towards Highly Transferable 3D Physical Camouflage for Universal and Robust Vehicle Evasion," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023, pp. 4305–4314. 2

[5] Syed M. Kazam Abbas Kazmi, Nayyer Aafaq, Mansoor Ahmad Khan, Ammar Saleem, and Zahid Ali, "Adversarial Attacks on Aerial Imagery : The State-of-the-Art and Perspective," in *2023 3rd International Conference on Artificial Intelligence (ICAI)*, 2023, pp. 95–102. 2

[6] Andrew Du, Yee Wei Law, Michele Sasdelli, Bo Chen, Ken Clarke, Michael Brown, and Tat-Jun Chin, "Adversarial Attacks against a Satellite-borne Multispectral Cloud Detector," in *2022 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 2022, pp. 1–8. 2

[7] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song, "Robust Physical-World Attacks on Deep Learning Visual Classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2

[8] Alexey Kurakin, Ian Goodfellow, Samy Bengio, et al., "Adversarial Examples in the Physical World," 2016. 2

[9] Huangxinxin Xu, Fazhi He, Linkun Fan, and Junwei Bai, "D3AdvM: A direct 3D adversarial sample attack inside mesh data," *Computer Aided Geometric Design*, vol. 97, pp. 102122, 2022. 2

[10] Kibok Lee, Zhuoyuan Chen, Xinchen Yan, Raquel Urtasun, and Ersin Yumer, "ShapeAdv: Generating Shape-Aware Adversarial 3D Point Clouds," *arXiv preprint arXiv:2005.11626*, 2020. 2

[11] Andrew Du, Bo Chen, Tat-Jun Chin, Yee Wei Law, Michele Sasdelli, Ramesh Rajasegaran, and Dillon Campbell, "Physical Adversarial Attacks on an Aerial Imagery Object Detector," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2022, pp. 1796–1806. 2, 3, 4

[12] Xingkui Zhu, Shuchang Lyu, Xu Wang, and Qi Zhao, "TPH-YOLOv5: Improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-captured Scenarios," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 2778–2788. 2

[13] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari, "Accelerating 3D Deep Learning with PyTorch3D," *arXiv:2007.08501*, 2020. 2, 4

[14] Dario Pavllo, Jonas Kohler, Thomas Hofmann, and Aurelien Lucchi, "Learning Generative Models of Textured 3D Meshes From Real-World Images," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 13879–13889. 2

[15] Suryanto, Naufal and Kim, Yongsu and Kang, Hyoeun and Larasati, Harashta Tatimma and Yun, Youngyeo and Le, Thi-Thu-Huong and Yang, Hunmin and Oh, Se-Yoon and Kim, Howon, "DTA: Physical Camouflage Attacks Using Differentiable Transformation Network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 15305–15314. 3

[16] Huiming Sun, Jiacheng Guo, Zibo Meng, Tianyun Zhang, Jianwu Fang, Yuewei Lin, and Hongkai Yu, "Evd4uav: An altitude-sensitive benchmark to evade vehicle detection in uav," *arXiv preprint arXiv:2403.05422*, 2024. 4

[17] Tao Yu, Runseng Feng, Ruoyu Feng, Jinming Liu, Xin Jin, Wenjun Zeng, and Zhibo Chen, "Inpaint Anything: Segment Anything Meets Image Inpainting," *arXiv preprint arXiv:2304.06790*, 2023. 4

[18] "Blender Cycles," `https://docs.blender.org/manual/en/latest/render/cycles/index.html`, Accessed: 2024-03-06. 4

[19] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds. 2015, vol. 28, Curran Associates, Inc. 5

Texture- and Shape-based Adversarial Attacks for Overhead Image Vehicle Detection

Supplementary Material

S1. TRAINING DETAILS & DETECTION MODELS

S1.1. Training Details

We use three model architectures in our experiments: RetinaNet, Faster R-CNN, and YOLOv5. We obtain RetinaNet and Faster R-CNN from Detectron2[3]. We retrieve YOLOv5 from its native implementation by Ultralytics[4]. We train the first two using the Detectron2 pipeline. We train YOLOv5 using its native pipeline.

S1.1.1. Real Models

The training data is derived from the LINZ dataset $\mathcal{I}^{\text{LINZ}}$, by removing all non-“Small Vehicle” class labels from the training set, resulting in 119 691 training images. We train the real RetinaNet and Faster R-CNN for 10 000 iterations and batch size 640. We train the real YOLOv5 for 50 epochs and batch size 640, which corresponds to approximately 10 000 iterations.

S1.1.2. Synthetic Models

To train the synthetic models, we produce a training synthetic dataset using PT3D consisting of 30 000 images. We train RetinaNet and Faster R-CNN for 10 000 iterations using batch size 128, while YOLOv5 is trained for 42 epochs using batch size 128 (approximately 9800 iterations).

S1.2. Detection Models

Table S1 shows the average precision scores by the real and synthetic models on the synthetic and real test sets. These results supplement Section 7.2 in the paper.

Table S1: Evaluation results of the six models on the real and synthetic test sets.

Architecture	**Training Data**	**Detection Threshold**	**AP** (real)	**AP** (synt.)	**AP** (Blender)
RetinaNet	$\mathcal{I}^{\text{LINZ}}$	49.82%	93.50%	94.80%	91.22%
Faster R-CNN	$\mathcal{I}^{\text{LINZ}}$	72.13%	80.34%	93.31%	81.71%
YOLOv5	$\mathcal{I}^{\text{LINZ}}$	59.85%	96.21%	95.51%	95.55%
RetinaNet	*PT3D*	47.64%	49.09%	99.88%	79.35%
Faster R-CNN	*PT3D*	86.95%	59.21%	99.49%	85.50%
YOLOv5	*PT3D*	60.40%	63.54%	99.95%	97.99%

[3]https://github.com/facebookresearch/detectron2
[4]https://github.com/ultralytics/yolov5

S2. PYTORCH3D DATA REALITY GAP MITIGATION

Mitigating the distribution gap between the real and PT3D datasets is essential for various reasons. The primary reason is the generalization of our results. We cannot claim that our results can generalize to one domain if we operate on a completely different domain. Hence, we attempt to minimize the distribution gap. We try to achieve this goal by optimizing specific parameters in the rendering pipeline, as described below.

S2.1. Gaussian Blur

We apply blurring to the vehicles in the images to simulate the blurring in the real images. Given a background image I_{bg} and the corresponding foreground image (*i.e.* , containing vehicles) I_{fg}, we apply Gaussian blur:

$$I_{\text{blur}} = I_{\text{bg}} + G(I_{\text{fg}} - I_{\text{bg}}), \qquad (1)$$

where $G(\cdot)$ is a Gaussian blur operator with kernel size defined by $k = 6 \cdot \lceil \sigma \rceil - 1$, where $\lceil \cdot \rceil$ is the ceiling operator, and σ is the blur level. We find that $\sigma = 2.4$ is the optimal blurring value as shown in Figure S5. Our analysis of the blur level shows that deficient levels of blur (*i.e.* , close to coarse PT3D renderings) result in less robust synthetic models when evaluated on the real data. Similarly, very high levels of blur (*i.e.* , almost vanished vehicles) also result in poor performance. As expected, the optimal value is somewhere in the middle. See the effect of applying blurring in Figure S9.

S2.2. Anti-aliasing

We use anti-aliasing techniques to remove pixelization from PyTorch3D’s coarse renderings. We apply them by rendering images four times larger than the intended size, then compressing them with the average pooling operator with kernel size 4 and stride 4. See the effect of anti-aliasing in Figure S9.

S3. DATASETS INFORMATION

This section provides technical details for the datasets we have sampled/annotated (real) or generated (synthetic) for our experiments.

S3.1. Real Datasets

We produced two real overhead-view datasets for our project. The images were sampled from two online sources: *Land In-*

formation New Zealand[5] (LINZ) and *Google Maps* (GMaps). Since both provide georeferenced imagery, the two image sets were sampled from the exact location in New Zealand - Selwyn[6].

S3.1.1. LINZ Dataset

Examples of the labeled LINZ and the background LINZ datasets are shown in Figure S10 and Figure S11, respectively. The distribution between negative (*i.e.* , empty) and positive (*i.e.* , non-empty) images in the LINZ dataset is as follows: 158 944 for negative images and 13 651 for positive images. See the distribution of vehicle categories in this set of images in Figure S6.

S3.1.2. Google Maps (GMaps) Dataset

We retrieved 173 264 images in total for the GMaps dataset, which approximately matches the number of sampled LINZ images. See examples of the GMaps dataset images in Figure S12.

S3.2. Synthetic Datasets

For our experiments, we rendered various synthetic datasets with original and adversarial objects using two rendering techniques: PyTorch3D and Blender. Here, we provide additional technical details and examples from each.

S3.2.1. PyTorch3D Datasets

Original. This dataset includes original (unmodified) car meshes. See examples of the PyTorch3D original images in Figure S13.

Adversarial and Random Textures. As described in the paper, we produce twelve adversarial texture maps and four random texture maps. See these texture maps in Figure S14. We generate 5000 validation images for each texture map that we use for evaluation. To generate an image, we first render the meshes using PT3D and then insert a background image sampled from the GMaps dataset. For each scene, we uniformly sample from one to five vehicles. We uniformly randomize the vehicle position and rotation in the scene. The camera always points to the origin of the coordinates as defined in PT3D. To sample the camera pose, we first uniformly sample a 2D-coordinate on a square, which we then re-project on a hemisphere, ensuring that the maximum elevation angle deviation from the vertical position is $20°$. View examples of these images in Figure S16.

[5]https://data.linz.govt.nz/

[6]https://data.linz.govt.nz/layer/51926-selwyn-0125m-urban-aerial-photos-2012-2013/

S3.2.2. Blender Datasets

Original. We render 14 459 images in Blender, where 998 contain vehicles and 13 461 images are empty. There are 2096 vehicles in total in the Blender data. See examples of the original Blender images in Figure S17.

Adversarial and Random Textures. We use the same adversarial and random texture maps as described for the PT3D adversarial data. We also use the same scenes as for the original Blender data, *i.e.* , 14 459 images, out of which 998 images contain 2096 vehicles in total. See example images in Figure S18.

S4. 3D MESH-BASED ADVERSARIAL ATTACKS

In this section, we describe the technical aspects detailing the methodology employed for executing adversarial attacks within each specific setting.

S4.1. Ensemble Attacks

See Table S2 for an overview of some significant hyper-parameters related to each adversarial attack reported in the paper.

Table S2: An overview of the hyper-parameters used in the ensemble attacks. The loss coefficients λ_1, λ_2 and λ_3 correspond to the loss coefficients applied to the loss objectives of RetinaNet, Faster R-CNN and YOLOv5 respectively, as described in Equation 1 in the main paper.

Attack Type	Loss coefficient λ_i			# of epochs
	λ_1	λ_2	λ_3	
A-U	0.020	10.000	10.000	3
A-Ma	0.020	10.000	10.000	3
A-Pix	0.020	10.000	10.000	3
A-PixMa	0.020	10.000	10.000	3
A-Lc	0.020	10.000	10.000	2
A-Fc	0.002	10.000	2.500	2
A-LcMa	0.007	10.000	5.000	2
A-FcMa	0.002	10.000	2.000	2
A-PixLc	0.020	10.000	10.000	2
A-PixFc	0.002	10.000	2.500	2
A-PixLcMa	0.020	10.000	5.000	2
A-PixFcMa	0.003	10.000	2.000	2
Shape Attack	0.020	15.000	32.000	2
A-Fc (seq.)	0.020	10.000	30.000	2
A-PixFc (seq.)	0.013	18.740	20.444	2
A-Fc (par.)	0.020	10.000	30.000	2
A-PixFc (par.)	0.011	13.180	20.860	2

S4.2. Texture Optimization

As outlined in Section 5.2, we employ three constraints that lead to the twelve adversarial texture settings discussed in the main paper. These constraints are *Spatial Resolution*, *Spatial Restriction* and *Color Restriction*. In this section, we discuss the implementation of each constraint. Before delving into the details of each constraint, we first describe how the adversarial texture is defined in the unconstrained attack (T-U). To execute this attack, a tensor of dimensions $512 \times 512 \times 3$ is initialized, representing the adversarial texture map. During this initialization process, each element in the tensor is uniformly sampled from 0 to 1. During the unconstrained attack, this tensor is the optimized entity.

S4.2.1. *Spatial Resolution*

To implement the spatial resolution constraint, we store the adversarial texture as a tensor of a smaller size. In our case, because we apply pixelization of size $16\,\text{px} \times 16\,\text{px}$, we store a latent representation of the adversarial texture as a $32 \times 32 \times 3$ tensor, where the first two dimensions are derived from the fact that the final texture map is expected to be $512 \times 512 \times 3$, hence $512/16 = 32$. Upon texture generation request, we upscale this tensor to $512 \times 512 \times 3$ using the nearest-neighbor interpolation, resulting in a pixelated output.

S4.2.2. *Spatial Restriction*

To implement the spatial restriction constraint, we use an adversarial texture map T_{adv} of size $512 \times 512 \times 3$, an original texture map T_{or} of a vehicle to which the adversarial texture is applied, and its corresponding binary segmentation mask T_{mask}. Using these three entities, we produce the segmented adversarial texture map

$$T_{\text{segmented}} = T_{\text{or}} \cdot (1 - T_{\text{mask}}) + T_{\text{adv}} \cdot T_{\text{mask}}. \tag{2}$$

See the "Ma" texture maps in Figure S15 to understand the final result. When combining this constraint with the Spatial Resolution constraint, we first produce a pixelated adversarial texture map and then apply masking.

S4.2.3. *Color Restriction*

Consider the following example to understand how the color constraint is implemented differently. Let p_i be the i-th pixel of a texture map, such that $p_i \in P$, where $P \in \mathbb{R}^{(H_t \cdot W_t) \times 3}$ is the set of all pixels in the texture map of size $(H_t \times W_t \times 3)$. In addition, let $C = \{c_i, \forall i = 1, 2, \ldots, N\}$, $c_i \in \mathbb{R}^3$ be the limited set of colors that we want to enforce for painting the texture map, where N is the number of allowed colors.

Ideally, we would like to be able to perform $\arg\min$ in a differentiable manner to reassign each pixel value at each attack iteration, such that $p_i \leftarrow \underset{c_i \in C}{\arg\min}(\|p_i - c_i\|_2)$. However, it is unclear how to do this differently. Therefore, we modify the pipeline to perform it in a differentiable fashion. First of all, we change the definition of each pixel in the texture map: instead of representing RGB values, each pixel now represents a set of probabilities of belonging to a particular color c_i from the set of colors C, *i.e.* , $p_i \in \mathbb{R}^N$, $\sum_k p_{i,k} = 1$, $P \in \mathbb{R}^{(H_t \cdot W_t) \times N}$ and $p_i = (p_{i,1}, p_{i,2} \ldots, p_{i,N})$, where $p_{i,k}$ is the probability that the i-th pixel in the texture map is c_k. Second, we define a softmax-like function, which we use to amplify the maximum value in a vector and suppress the non-maximum values. We control the amplification and suppression levels with a temperature parameter τ. Applying this softmax-like function to some vector $r = [r_1, r_2, \ldots]^{\mathrm{T}}$, we obtain $s(r_i) = \frac{e^{\ln(r_i)/\tau}}{\sum_j e^{\ln(r_j)/\tau}}$. For simplicity, let $s(p_i)$ represent $[s(p_{i,1}), \ldots, s(p_{i,N})]^{\mathrm{T}}$. Whenever prompted to generate a texture map with the Color Restriction constraint, we perform the following procedure on each pixel p_i to obtain its output RGB form $\hat{p}_i \in \mathbb{R}^3$:

$$\mathbf{w}_i = s\left(s\left(p_i\right)\right), \tag{3}$$

$$\hat{p}_i = \mathbf{C} \cdot \mathbf{w}_i, \tag{4}$$

where $\mathbf{C} = [c_1\, c_2\, \ldots\, c_N] \in \mathbb{R}^{3 \times N}$. In other words, we first shift the probabilities towards the maximum probability class as shown in Eq. (3), then, treating probabilities as weights, we perform a weighted sum of colors C as shown in Eq. (4). As we empirically find, performing soft-argmax (Eqs. (3) and (4)) *twice* results in a much better approximation to argmax than if it was performed only once. We tried reducing the temperature parameter τ and performing soft-argmax only once, but lower temperatures resulted in numerical instability. After each attack cycle, softmax is applied to each p_i to ensure $\sum_k p_{i,k} = 1$. After finishing an adversarial attack with this constraint, a non-differentiable $\arg\max$ assigns colors from C to each pixel, ensuring a final texture map with at most N colors. During the attack, either $\{P\}$ or $\{P, C\}$ can be optimized, contingent upon the applied restrictions ("Fc" or "Lc" respectively).

S4.3. Shape Optimization

As outlined in Section 5.3 in the main paper, we optimize the displacement map, transforming the pixel values into vertex deformations. We employ two constraints *Symmetry* and *Magnitude*.

We initialize a negative displacement tensor of shape $64 \times 64 \times 1$ representing a single channel (grayscale) image. This ensures that the number of pixels in the displacement map (4096) is always greater than the number of vertices of the car meshes we use (1000). Contrary to the texture map initialization, we initialize this tensor with zeros, aiming to start from the un-deformed state.

Additionally, a topology map is calculated from the static UV map of each mesh. The topology map gathers and retains the information of each unique vertex from the UV map. This helps align the displacement maps, UV maps, and texture maps.

The deformation at each vertex of the mesh is calculated by using the equation

$$\Delta V_{\text{i}} = R_i \cdot D_i, \tag{5}$$

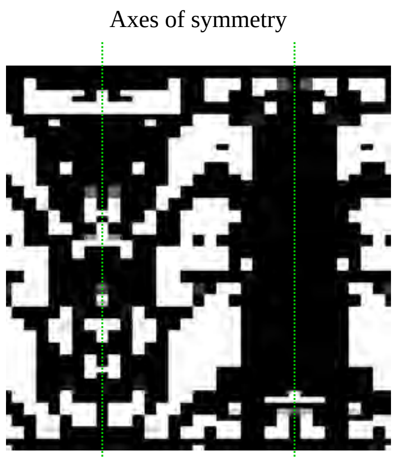


Fig. S1: An example of the axes of symmetry in a displacement map. The highlighted axes correspond to the central plane that cuts the mesh in two halves along its longitudinal direction.

where R_i is the vector defined by joining the geometric mean of all the vertices of the vehicle mesh and the corresponding vertex coordinate V_i. To calculate D_i, the displacement tensor is circularly padded to match the dimensions of the UV map. Then, each point sampled from the topology map is used to interpolate the aligned displacement map bi-linearly to calculate the corresponding deformation D_i. The two constraints are imposed as follows.

S4.3.1. Symmetry

To ensure the symmetrical layout of the mesh, we apply a symmetry mask while deforming the mesh. This symmetry mask is calculated from the UV map with two axes of symmetry. The axes of symmetry for the corresponding displacement map are shown in Figure S1.

S4.3.2. Magnitude

The maximum amount of perturbation is crucial to determine the practicality. We determine the width of the car mesh by finding the maximum difference of vertex positions along the width. The sigmoid function $\sigma(x) = 1/\left(1 + e^{-x}\right)$ is used on the displacement tensor to make sure its values lie between 0 and 1. The final deformations are calculated by extending Equation (5).

$$\Delta V_i = \text{PM} \cdot W \cdot \sigma\left(R_i \cdot D_i\right) \tag{6}$$

where PM is the perturbation magnitude as defined in Section 5.3 in the main paper and W is the width of the car. The displacement maps corresponding to optimal perturbations, when greater than 0, as described in Table 2 in the main paper are shown in Figure S19.

S5. ADDITIONAL RESULTS

In this section, we report some further results to complement the arguments from the main paper.

S5.1. EASR

To evaluate the effectiveness of the adversarial meshes, we rendered two matched image datasets, D_{or} and D_{adv}, from each experiment. For D_{or}, we composed and rendered 3D scenes with the original car meshes. For D_{adv}, we apply adversarial modifications to the meshes of the same scenes. Thus, each image from D_{or} has an identical version (the same background, lighting, camera parameters, and car locations and orientations) with adversarial cars. We compute the percentage of vehicles **d**etected in D_{or} but **m**issed in D_{adv}. $V_{d,m}$ represents such vehicles, and $V_{d,d}$ denotes vehicles detected in both D_{or} and D_{adv}. This computation yields the *Attack Success Rate* ASR $= \frac{|V_{d,m}|}{|V_{d,d} \cup V_{d,m}|}$, where $|\cdot|$ is the cardinality operator. In our task, avoiding introducing new detections $V_{m,d}$ after applying the adversarial entity is also important. Thus, we modify ASR to account for this:

$$\text{EASR} = \frac{|V_{d,m}| - |V_{m,d}|}{|V_{d,d} \cup V_{d,m}|} = \text{ASR} - \text{ER}, \tag{7}$$

where EASR is the *Effective Attack success Rate* and ER is the *erroneous rate*, *i.e.* fraction of true-positive detections that emerged after introducing the adversarial entity.

S5.2. APD

In addition to computing the EASR, we evaluate the average precision drop (APD) when running adversarial attacks. To calculate APD, we first compute the AP on a dataset of original images D_{or} and on a dataset of adversarial images (D_{adv} (where both are as defined in Section 7.1 in the main paper), resulting in AP$_{\text{or}}$ and AP$_{\text{adv}}$ respectively. We then obtain the average precision drop as APD $=$ AP$_{\text{or}}$ $-$ AP$_{\text{adv}}$. See the results in Table S3.

As anticipated and previously noted, the findings indicate that introducing constraints diminishes performance while incorporating shape modifications alongside texture alterations restores performance. We also highlight the notably low APD observed in randomly generated texture maps, implying that replicating adversarial modifications randomly may yield poor results.

S5.3. Original Blender Data Evaluation

We also report the evaluation results of all models using the original Blender data. See example images in Section S3.2.2, and the evaluation results in Table S1. The evaluation results suggest that almost all models perform quite well on the Blender original data. It could be the consequence of the

Table S3: The figures show mean values from evaluations of individual synthetic models on PT3D and Blender data. “T”, “R”, “S” and “C” represent the texture, random texture, shape, and combined attacks. Note that Lc and Fc are mutually exclusive by definition. The constraints follow the definitions outlined in Section 5.2. PM* and Pr* represent the optimal perturbation magnitude and practicality of the attacks involving shape modifications.

Attack	**Constraints**				PM*	Pr*	**PT3D**	**Blender**
	Pix	Lc	Fc	Ma			**APD**	**APD**
T-U					—	—	50.97%	63.17%
T-Ma				✓	—	—	32.68%	40.67%
T-Pix	✓				—	—	53.28%	59.03%
T-PixMa	✓			✓	—	—	24.20%	37.47%
T-Lc		✓			—	—	46.17%	64.95%
T-Fc			✓		—	—	7.54%	48.46%
T-LcMa		✓		✓	—	—	26.83%	46.80%
T-FcMa			✓	✓	—	—	2.56%	23.11%
T-PixLc	✓	✓			—	—	56.30%	62.13%
T-PixFc	✓		✓		—	—	9.62%	48.92%
T-PixLcMa	✓	✓		✓	—	—	22.97%	37.59%
T-PixFcMa	✓		✓	✓	—	—	2.53%	38.32%
R-U					—	—	0.21%	13.69%
R-Pix	✓				—	—	0.51%	16.77%
R-Fc			✓		—	—	0.85%	15.82%
R-PixFc	✓		✓		—	—	0.62%	17.75%
S-O	—	—	—	—	0.4	0.6	53.18%	72.47%
C-U					0.0	1.0	55.43%	—
C-Pix	✓				0.0	1.0	58.09%	—
C-Lc		✓			0.0	1.0	50.49%	—
C-Fc (seq.)			✓		0.2	0.8	18.35%	62.96%
C-Fc (par.)			✓		0.2	0.8	37.39%	62.57%
C-PixLc	✓	✓			0.0	1.0	58.13%	—
C-PixFc (seq.)	✓		✓		0.2	0.8	36.09%	68.64%
C-PixFc (par.)	✓		✓		0.2	0.8	37.79%	71.18%

Blender dataset being a high quality simulation of real world data, *i.e.* it is between the coarse PT3D data and the fine-grained LINZ data. Consequently, both the real and synthetic models exhibit strong performance, attributed to the close resemblance between the Blender dataset and the training sets of both sets of models.

S5.4. Evaluating Real Data Models on the Adversarial Data

We also evaluate the real models (*i.e.* , trained on real data) on all adversarial datasets rendered using Blender. We run these experiments to assess how robust models trained on real data would react to adversarial samples that are highly realistic. However, we recognize the distribution gap between the real data and the synthetically produced data with Blender. See the results of the evaluations in Figures S2 and S3.

S6. PRACTICALITY AND COMPARISONS

This section extends the arguments discussed in Section 6 in the main paper. See the extended version of Table 1 from the main paper below in Table S4. Because the optimal Pr level found for C-U, C-Pix, C-Lc, and C-PixLc is 1.0, *i.e.* PM $= 0.0$, hence these combined attacks are not considered in Table S4, because they correspond to their texture-based counterparts T-U, T-Pix, T-Lc, and T-PixLc, respectively.

Table S4: Comparing the practicality of the attacks explored in our study to previous works. We do not distinguish between sequential and parallel combined attacks as they only impact the process, not the final result’s form. The first symbol reflects the texture-related practicality score, the second symbol reflects the shape-related practicality score. This table complements Table 1 from the main paper. Compared to the table in the main paper, the “Notes” column is missing because we discuss the score of each camouflage in detail in the text, instead of leaving brief notes in the table.

	Camouflage	**PC**	**DI**	**DO**	**Total Score**
Other works	Du *et al.* (ON) [22]	+0	+0	+0	+3
	Du *et al.* (OFF) [22]	00	+0	−0	0
	EVD4UAV [71]	+0	+0	+0	+3
	FCA [75]	−0	−0	+0	−1
	ACTIVE [73]	−0	−0	+0	−1
	DTA [72]	−0	−0	+0	−1
Our	T-U	−0	−0	−0	−3
	T-Ma	−0	−0	+0	−1
	T-Pix	−0	+0	−0	−1
	T-PixMa	−0	+0	+0	+1
	T-Lc	−0	−0	−0	−3
	T-Fc	−0	−0	−0	−3
	T-LcMa	−0	−0	+0	−1
	T-FcMa	−0	−0	+0	−1
	T-PixLc	00	+0	−0	0
	T-PixFc	+0	+0	−0	+1
	T-PixLcMa	00	+0	+0	+2
	T-PixFcMa	+0	+0	+0	+3
	S-O	0−	0−	0−	−3
	C-Fc	−−	−−	−−	−6
	C-PixFc	+−	+−	−−	−2

The constraints that we implement affect the practicality of the final adversarial meshes, expressed through the production cost (PC), the difficulty of installation (DI), and difficulty of operation (DO). Production cost refers to the estimated cost of producing the physical camouflage, including material, printing expenses, and labor time. Difficulty of installation refers to how easy or difficult it is to physically apply or set up the camouflage on a vehicle. Difficulty of operation assesses the extent to which the camouflage affects the normal operation or mobility of the vehicle. See detailed discussions below.

S6.1. Texture-Based Attacks

We first consider the texture modifications and their effect on the practicality.

The **Spatial Resolution** constraint (abbreviated as “Pix”) provides a practical approach to camouflage implementation

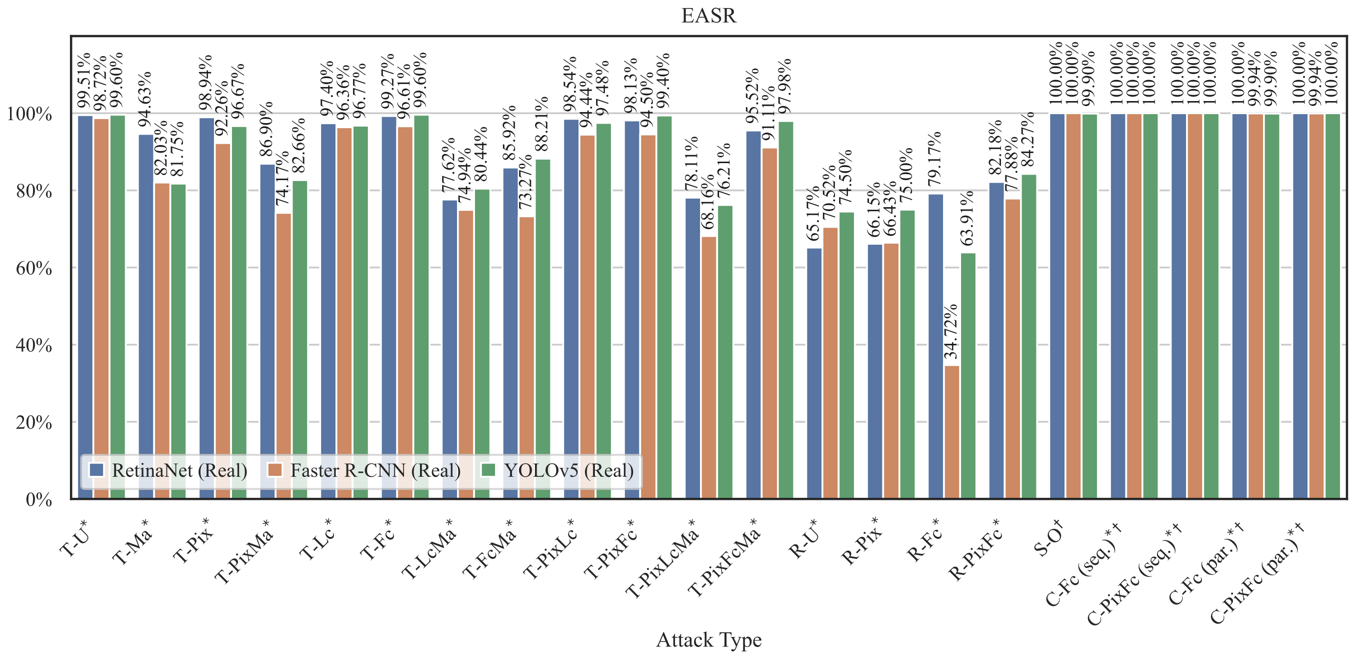


Fig. S2: Evaluation results of the models trained on real data and tested on the Blender-generated adversarial datasets. Dentations: *texture-only attacks, †shape-only attacks, and *†combined attacks.

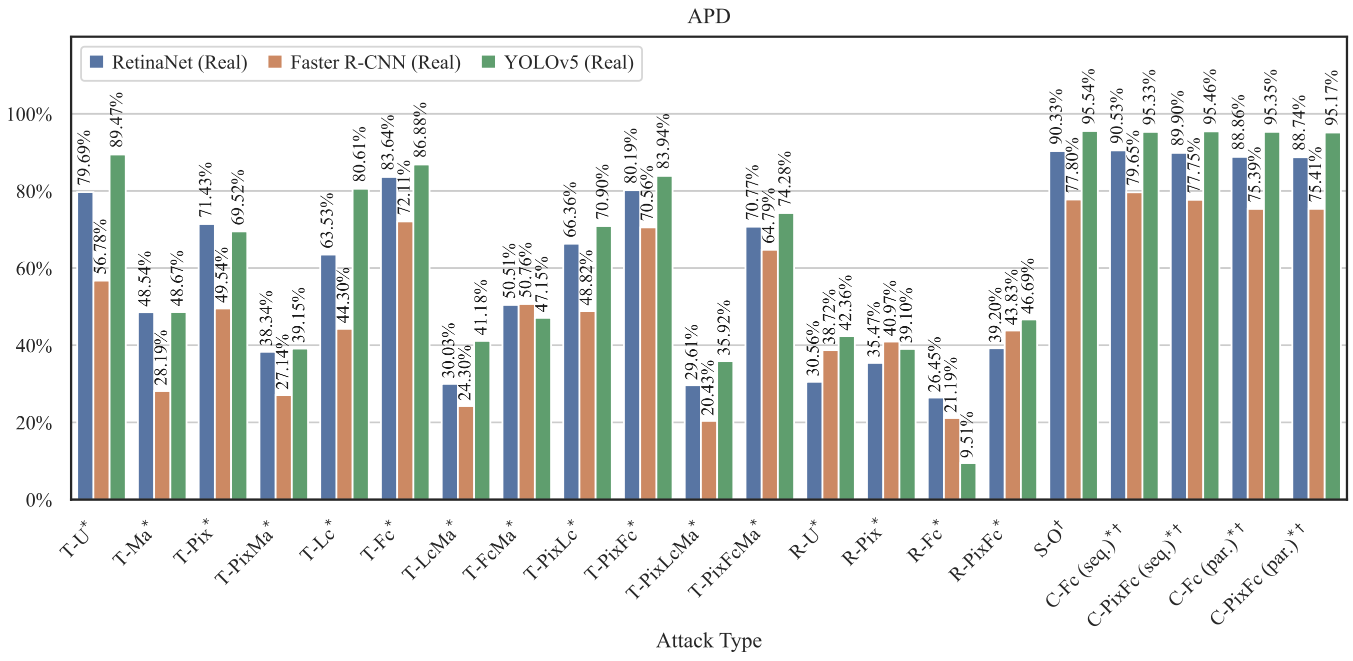


Fig. S3: Evaluation results of the models trained on real data and tested on the Blender-generated adversarial datasets. Dentations: *texture-only attacks, †shape-only attacks, and *†combined attacks.

by utilizing stickers or painting squares rather than applying the entire camouflage in one go (for example, by using vinyl wraps, as discussed below). This method enhances the DI score, resulting in improved outcomes. Consequently, camouflages adhering to the "Pix" constraint receive a positive texture DI score (+X), whereas those that do not adhere to it receive a negative score (−X). Here, X is a placeholder for the shape-related score.

Secondly, the **Spatial Restriction** constraint (referred to as "Ma") takes into consideration the potential challenges of operating a vehicle covered entirely by camouflage, which can restrict the vehicle's mobility. Our findings indicate that full-coverage camouflages are more effective (refer to Table 2 in the main paper; attacks involving "Ma" consistently yield lower EASR compared to their non-"Ma" counterparts). However, such camouflages are only suitable for stationary vehicles, limiting operational flexibility. Introducing this constraint allows for maintaining mobility. Therefore, camouflages adhering to this constraint receive a positive texture DO score (+X), while those not adhering to it receive a negative score (−X).

The **Color Restriction** (denoted as "Lc" or "Fc") constraint minimizes the color palette for generating adversarial texture maps, impacting camouflage production costs (PC). Without any color restriction, we assume a negative texture PC score (−X), as full-color printing, typical for such cases, incurs high costs (e.g., starting from $2000 for vinyl wraps).[7] Simply reducing colors does not cut costs, as vinyl wraps remain necessary. However, combining color restriction with spatial resolution (e.g., "PixFc") lowers costs by using stickers or manually coloring squares using a small predefined set of colors. Such combinations positively affect both PC and DI scores (+X). For the limited color constraint ("Lc"), where colors are automatically identified, we assume no impact on PC score (0X) due to potentially hard-to-obtain colors.

S6.2. Shape-Based and Combined Attacks

The shape-based attacks do not involve any texture alterations, resulting in texture-related scores of 0 across all three criteria. Moreover, reproducing shape modifications proves challenging, resulting in negative scores across all three shape-related criteria. Estimating the cost and difficulty of installation of such modifications remains uncertain, dependent on the vehicle's original shape and the extent of planned alterations. Similarly, assessing the difficulty of operation proves challenging and contingent on various factors. The PC, DI, and DO scores for the texture components in combined attacks mirror those of the texture-only attacks.

[7] https://www.jdpower.com/cars/shopping-guides/how-much-does-it-cost-to-wrap-a-car

S6.3. Other Works

Du *et al.* introduce two types of camouflages: ON and OFF. The ON type is applied on the rooftop of a vehicle, while the OFF type is placed outside of the vehicle. We find that the ON type, as well as EVD4UAV, is as practical as our most constrained texture-based attack, but its limited coverage area renders it impractical within the geospatial resolution context of our study. It could be considered as a tighter version of our implemented spatial restriction constraint ("Ma"), leading to lower performance. The OFF type of camouflage scores lower on DO due to mobility limitations. Additionally, the production cost of such camouflage remains unclear, resulting in a neutral PC texture score.

The remaining three works (FCA, ACTIVE, and DTA) share similarities with our T-Ma camouflage. Therefore, we assign them the same scores as the T-Ma camouflage.

S7. ANALYSIS OF ADVERSARIAL TEXTURES

Throughout our experiments, we observed a striking similarity in the prevalence of highly saturated colors, between unconstrained adversarial texture maps and adversarial patches generated in prior studies, such as those mentioned in [22,4,11]. Further analysis of our results and those of other researchers revealed that adversarial texture maps or patches generated in setups with minimal constraints tend to saturate colors located at the edges of the RGB color cube. They consistently exhibited extreme color saturation. Figure S7 shows the different T-U textures obtained with different attack initializations. As you can see, despite different initializations, they are very similar.

Additionally, our analysis of the latent space indicated that adversarial attacks could shift vehicle embeddings toward the background distribution but were unable to achieve complete blending, see Figure S8. We used PCA[8] and t-SNE[9] on features from a synthetic RetinaNet model. Replicating the background underneath the car would be the most optimal solution resulting in the perfect camouflage. As a result, the features extracted from adversarial vehicles did not closely resemble the original vehicle or the background embeddings but instead fell somewhere in between.

As mentioned, some other works produce adversarial patches with highly saturated colors, similar to our T-U texture map. Therefore, we analyze the color distribution to verify that the colors appear at the edges of the RGB cube. To do so, we plot the distribution of pixel values along the red, green, and blue channels for each texture map.

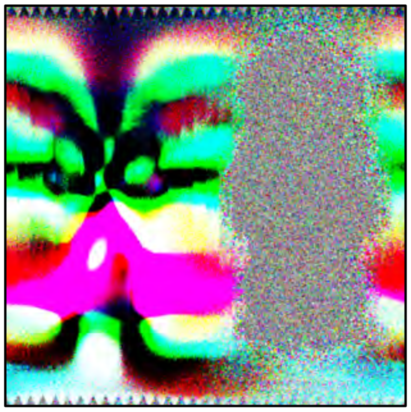
Du *et al.* make their adversarial patches public[10] in good quality, so we use them to compare. See the results of the

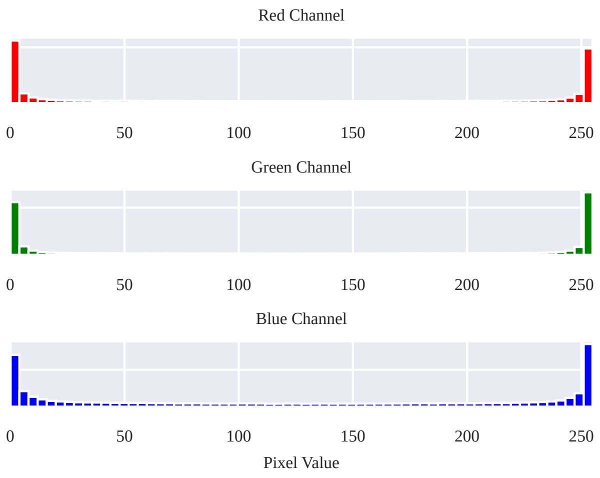
[8] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

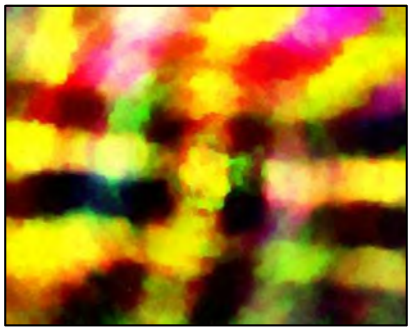
[9] https://jmlr.org/papers/v9/vandermaaten08a.html

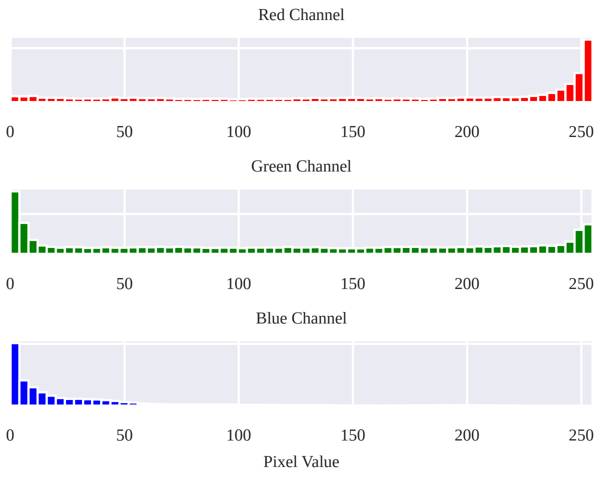
[10] https://github.com/andrewpatrickdu/adversarial-yolov3-cowc

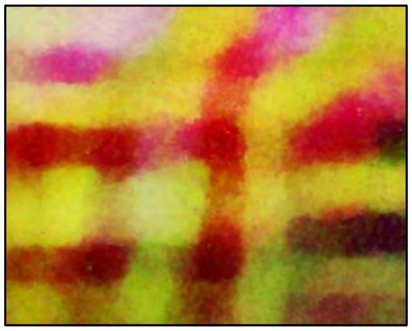
comparison in Figure S4. Interestingly, they conclude that weather augmentations do not considerably improve the results. However, we find that weather augmentations during optimization significantly affect the distribution of colors by pushing a significant fraction of pixels away from the edges of the RGB cube. This type of effect can be used to constrain the optimized space, which, without any strict constraints, seems to attempt to drive the optimization outside the RGB cube (hence causing crowding at the edges).


A-U (Our)




CP-On-GC (Du et al.)




CP-On-GCW (Du et al.)

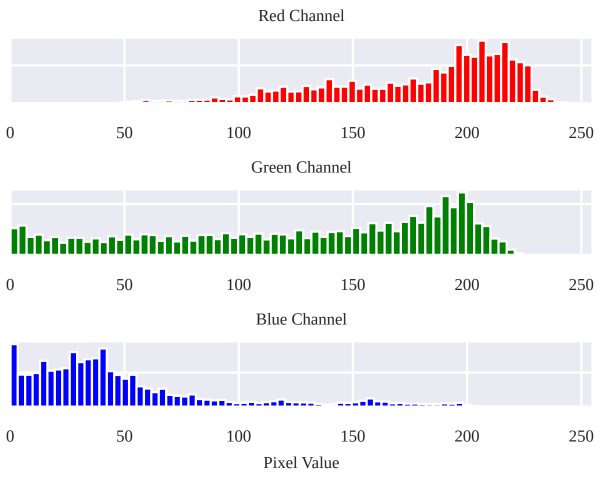


Fig. S4: Color distribution in adversarial patches.

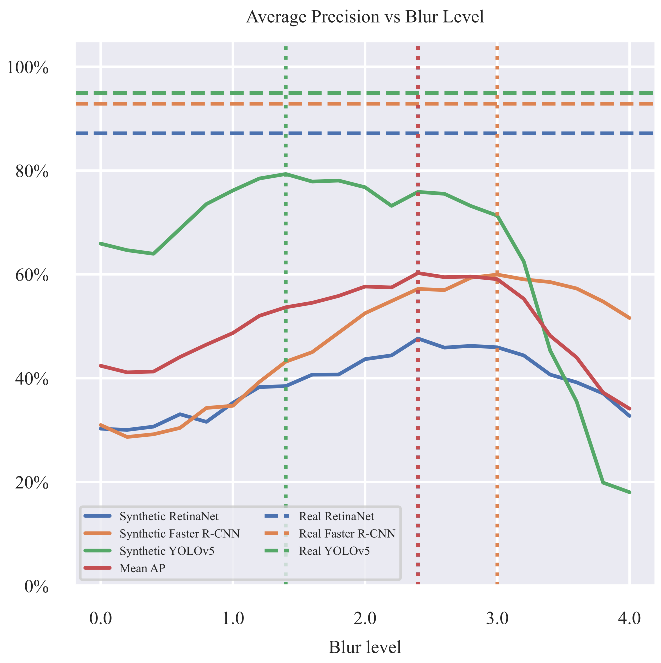


Fig. S5: Each point on the solid lines corresponds to a synthetic model trained using the corresponding blur level. Each such model is evaluated on the real validation set. The horizontal dashed lines represent the real models' performance on the real validation set. The vertical dotted lines represent the maxima. The red line represents the average curve of the other three curves. As shown by this analysis, $\sigma = 2.4$ is the optimal blur level.

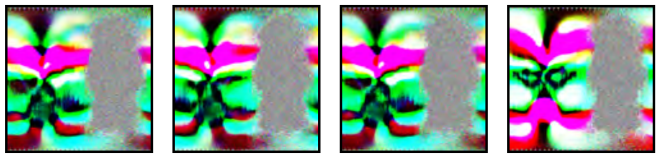


Fig. S7: Examples of T-U textures obtained with different initializations. The grey region maps to the underside of the car which is ignored by the adversarial optimization.

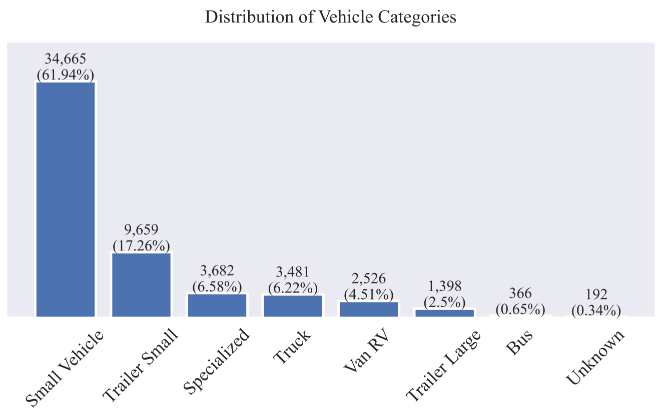


Fig. S6: Visualization of vehicle category distribution in the LINZ dataset: each bar signifies the number of samples associated with a specific vehicle category. The figures within the brackets indicate the proportion of total vehicles represented by each class.

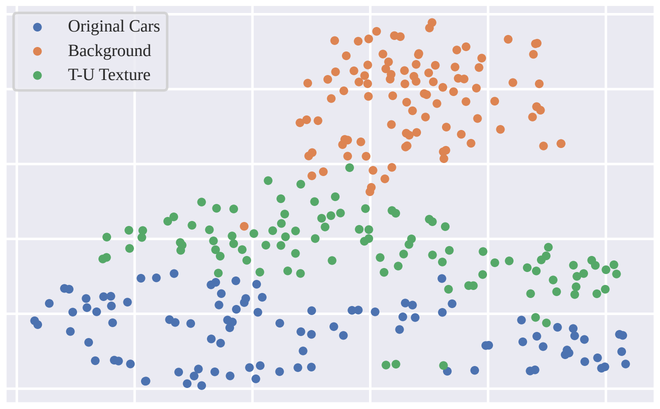


Fig. S8: Embeddings of background images and vehicles with original and T-U texture maps.

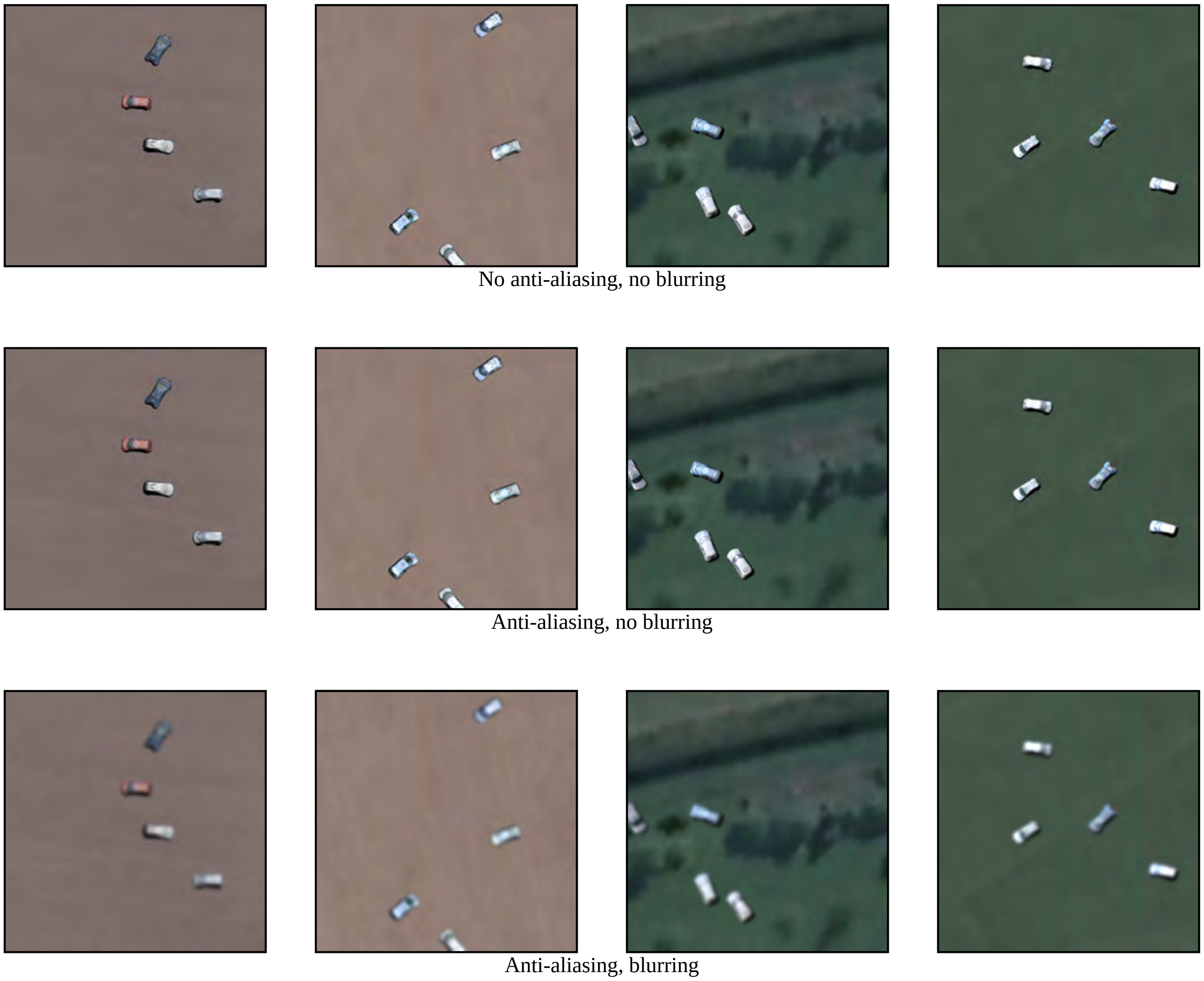


Fig. S9: The first row represents the coarse renderings by PyTorch3D. The second row represents the result of applying anti-aliasing. The third row represents the result of applying both anti-aliasing and blurring.

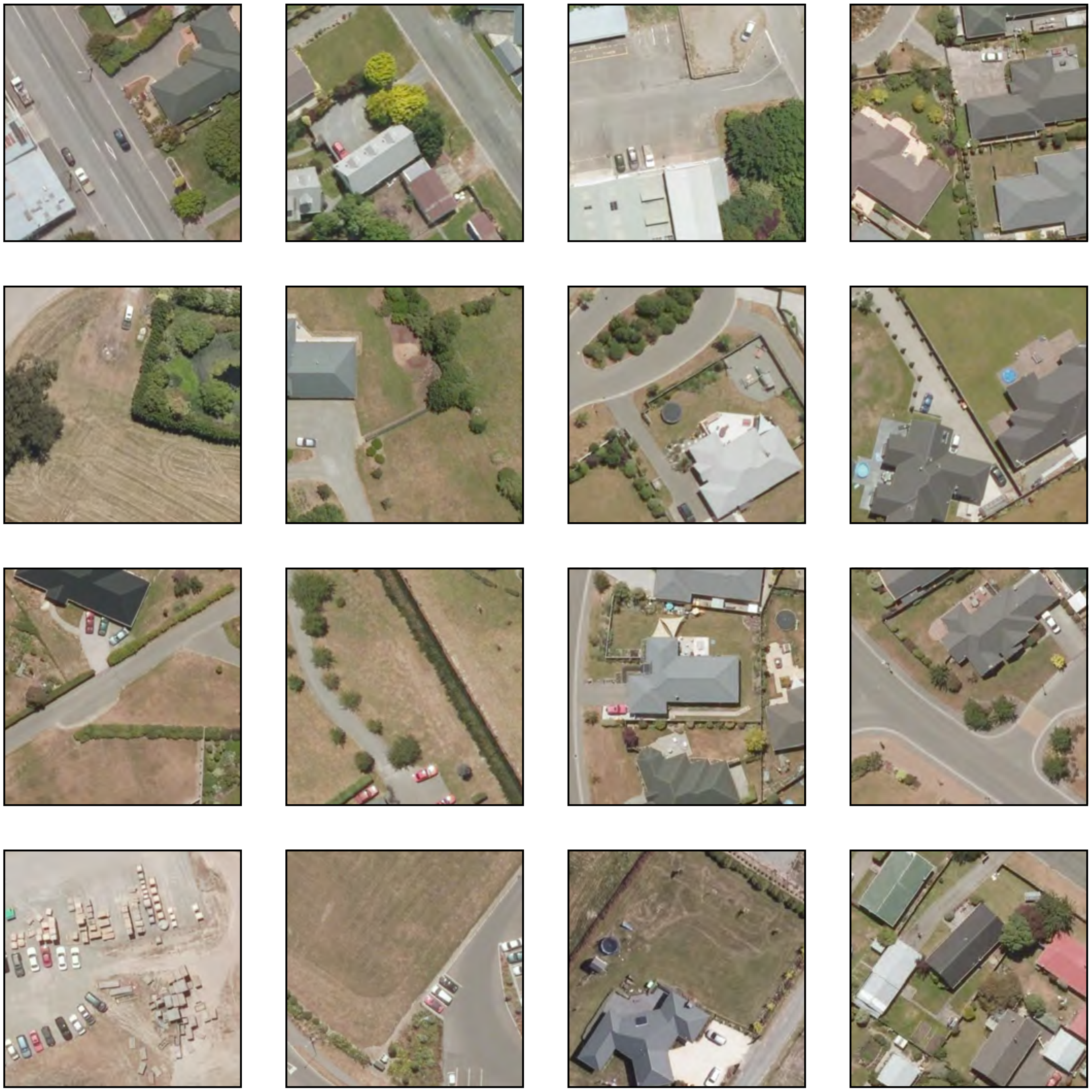


Fig. S10: Examples of the labeled LINZ dataset.

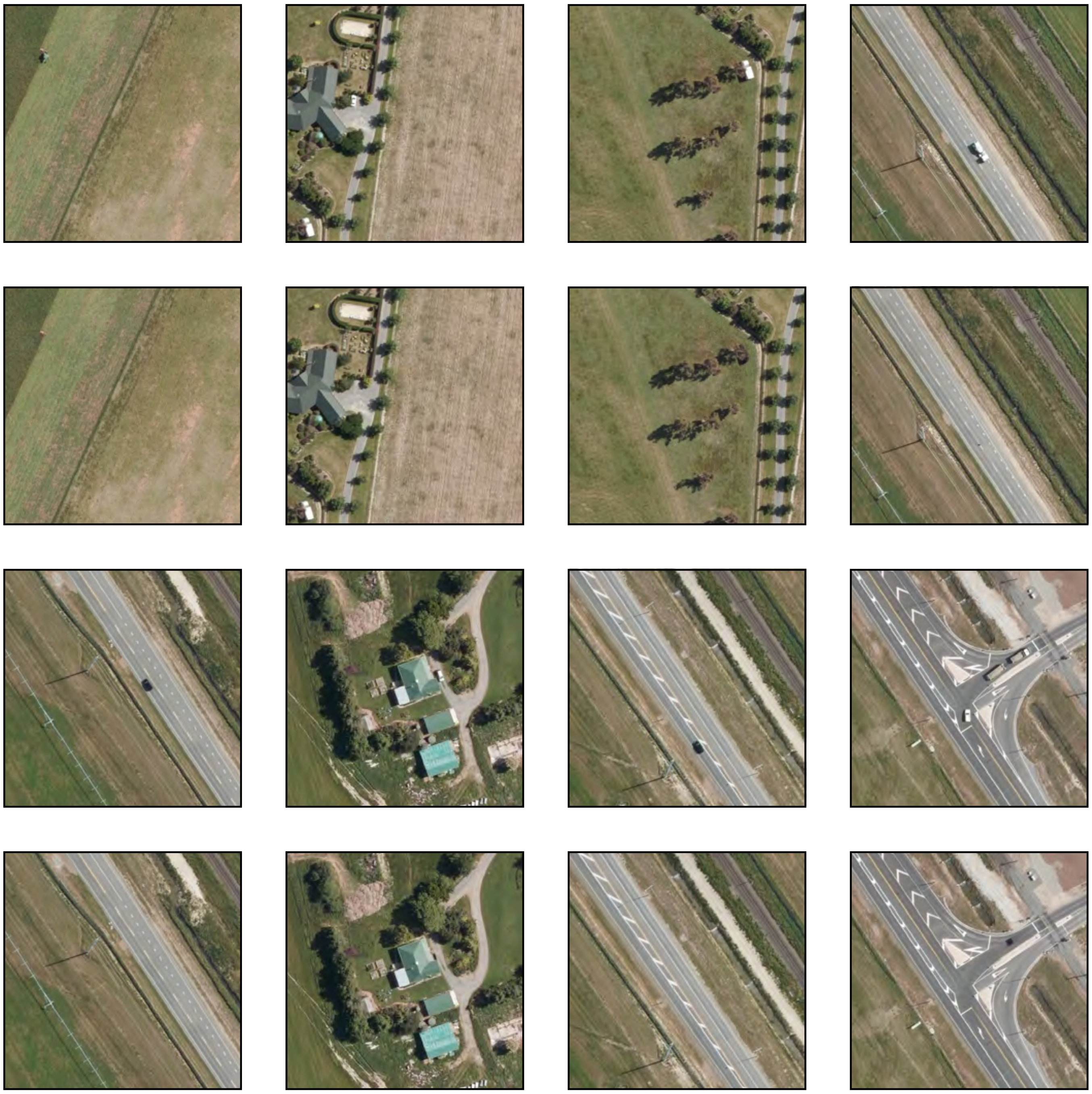


Fig. S11: The odd rows represent original images from the LINZ dataset. The even rows represent the corresponding background LINZ images, where the vehicles have been automatically removed.

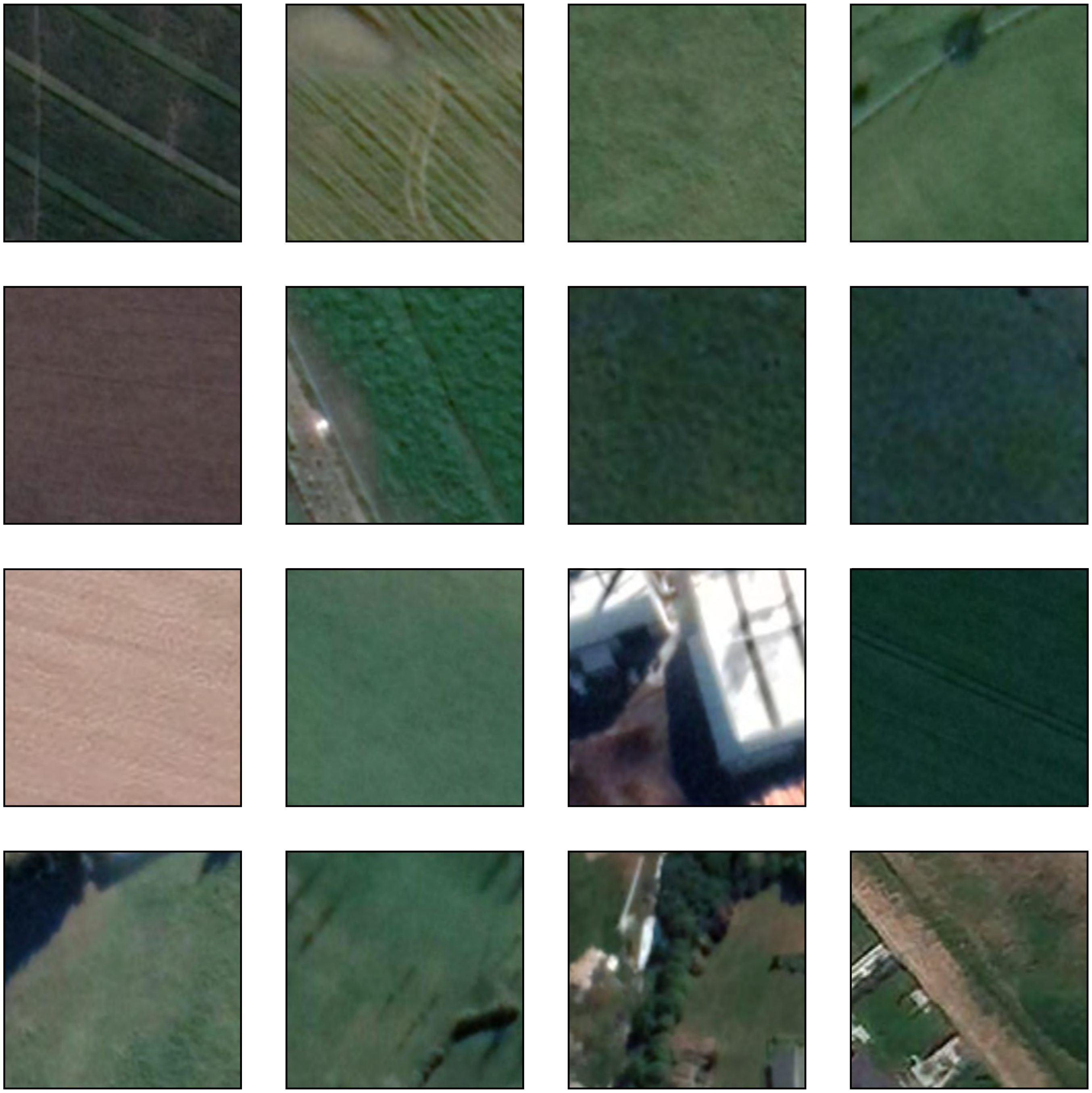


Fig. S12: Examples of the GMaps background dataset.

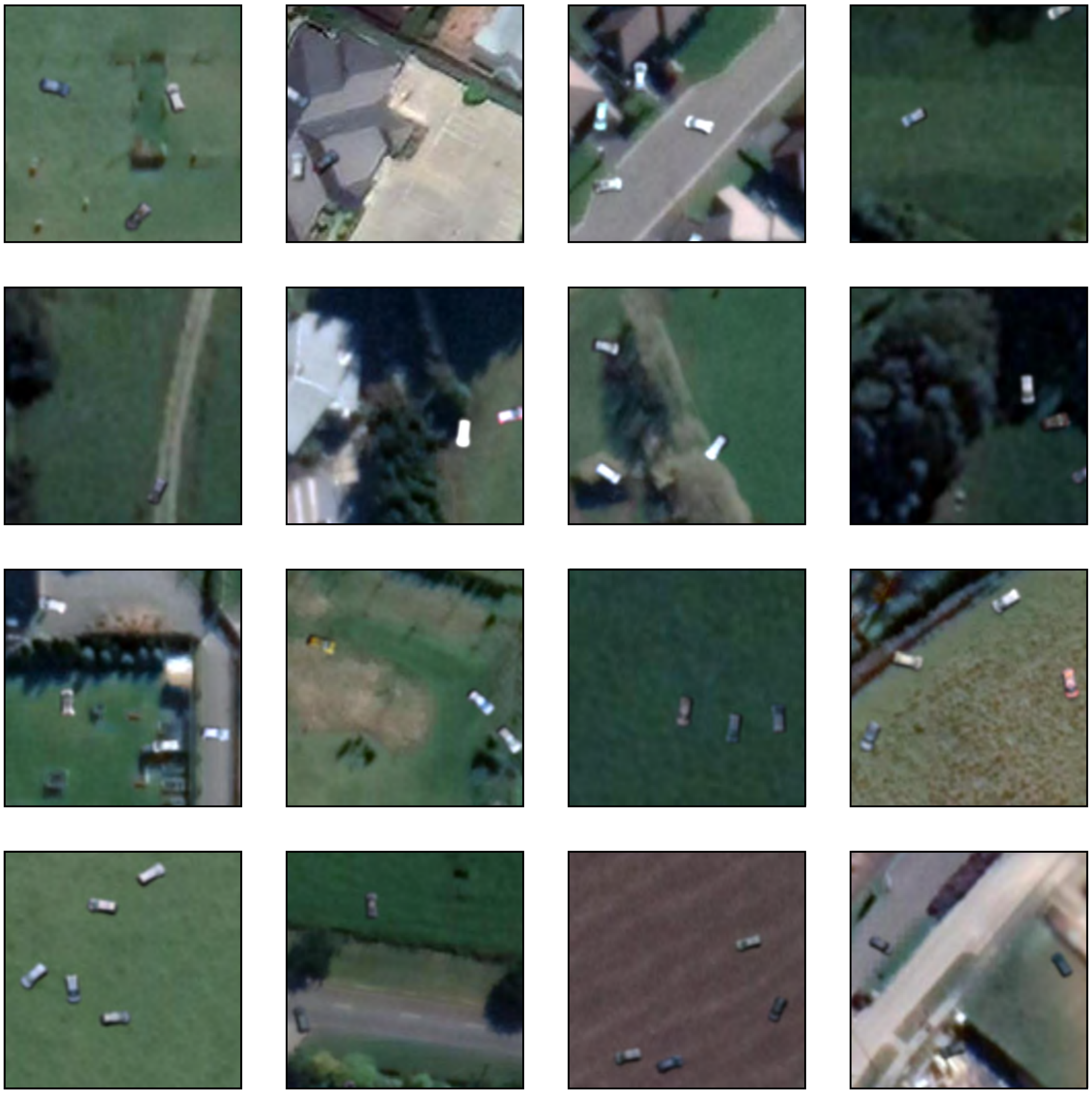


Fig. S13: Examples of the original PT3D images.

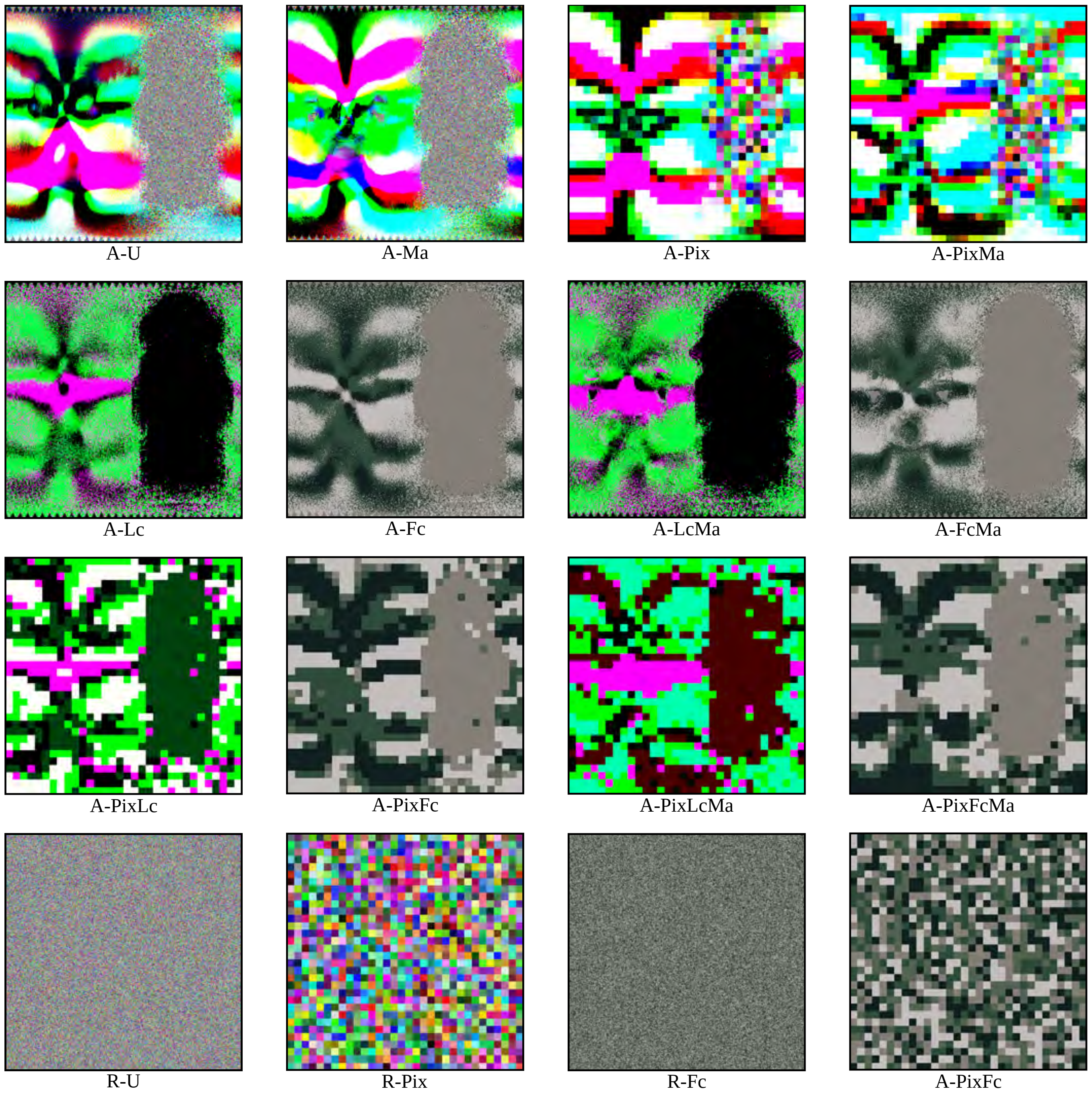


Fig. S14: Adversarial and random texture maps. The right side corresponds to the car's underside, which the adversarial optimization ignores.

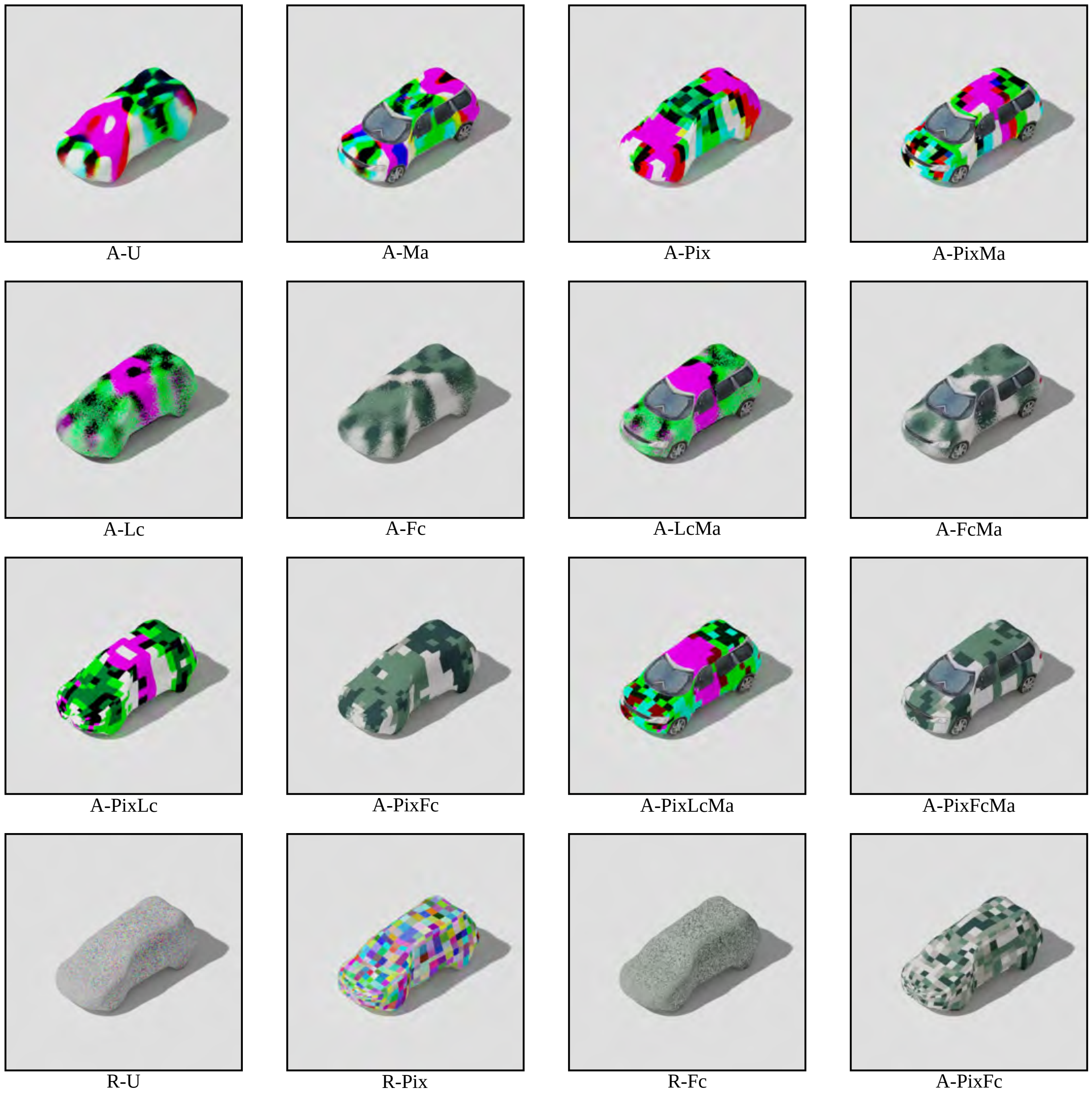


Fig. S15: Visualizations of the textures from Figure S14 applied to a car mesh.

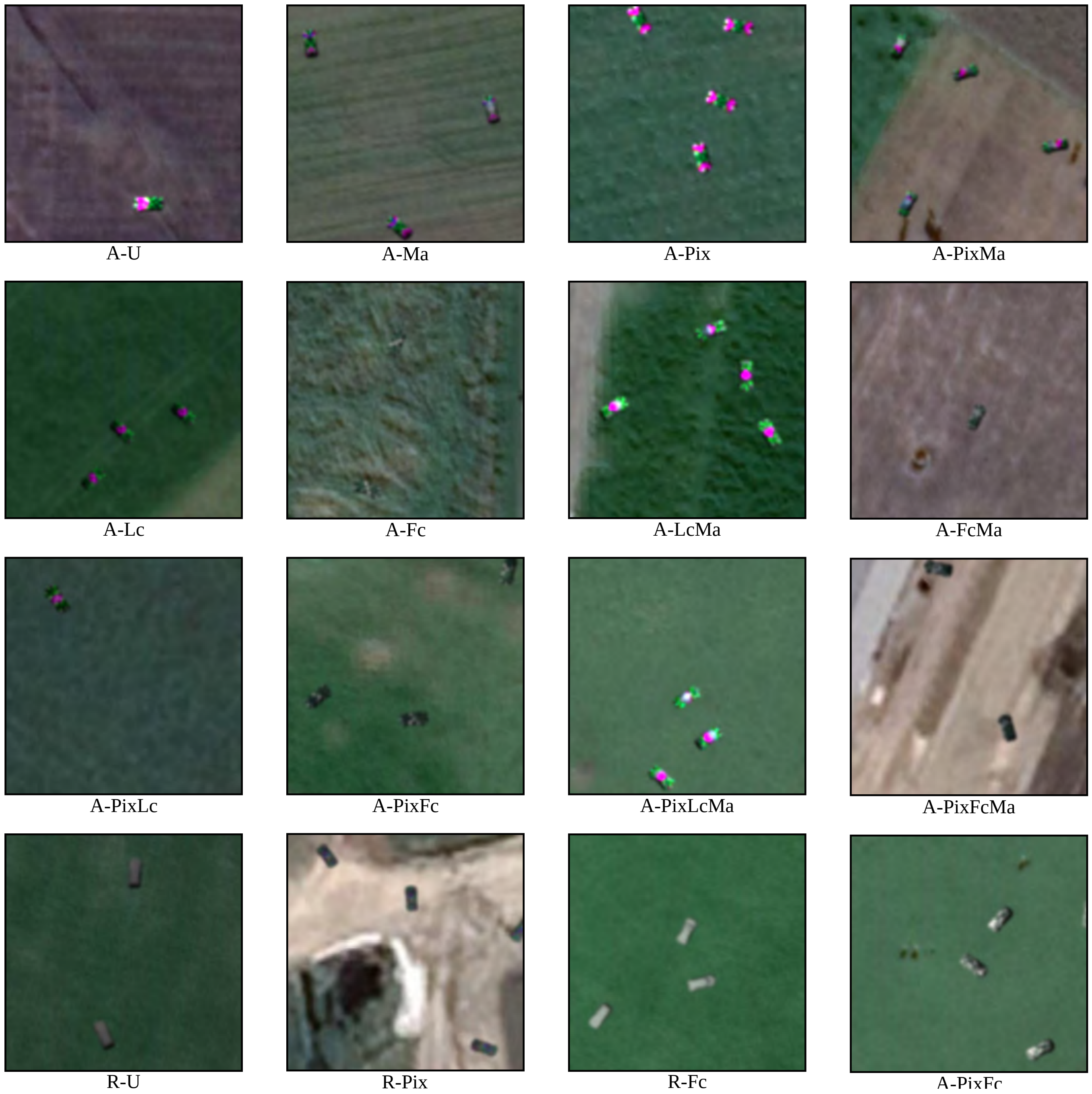


Fig. S16: Illustrations of vehicles sourced from the PT3D datasets featuring adversarial and random texture maps.

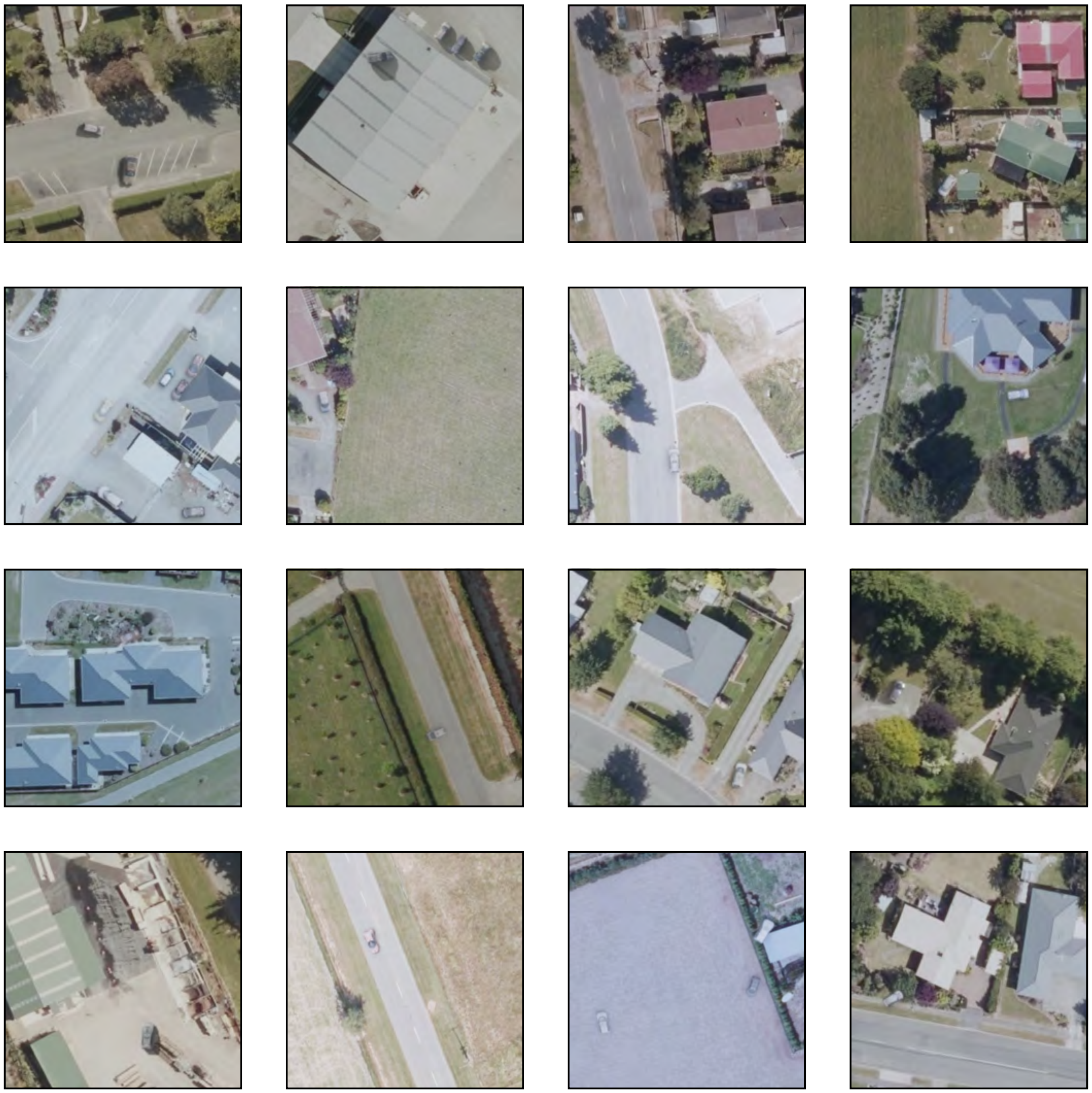


Fig. S17: Examples of the original Blender images.

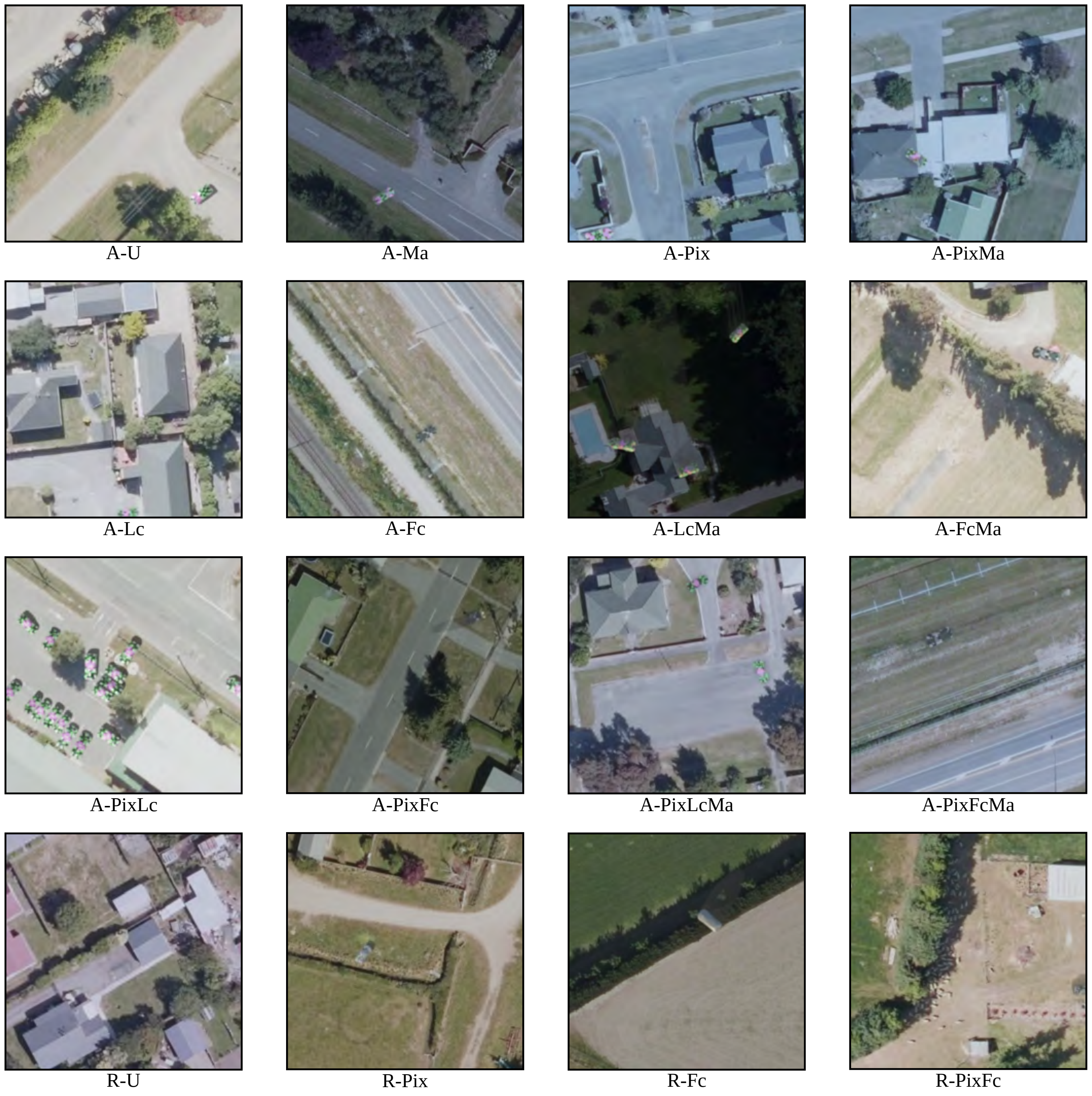


Fig. S18: Illustrations of vehicles sourced from the Blender datasets featuring adversarial and random texture maps.

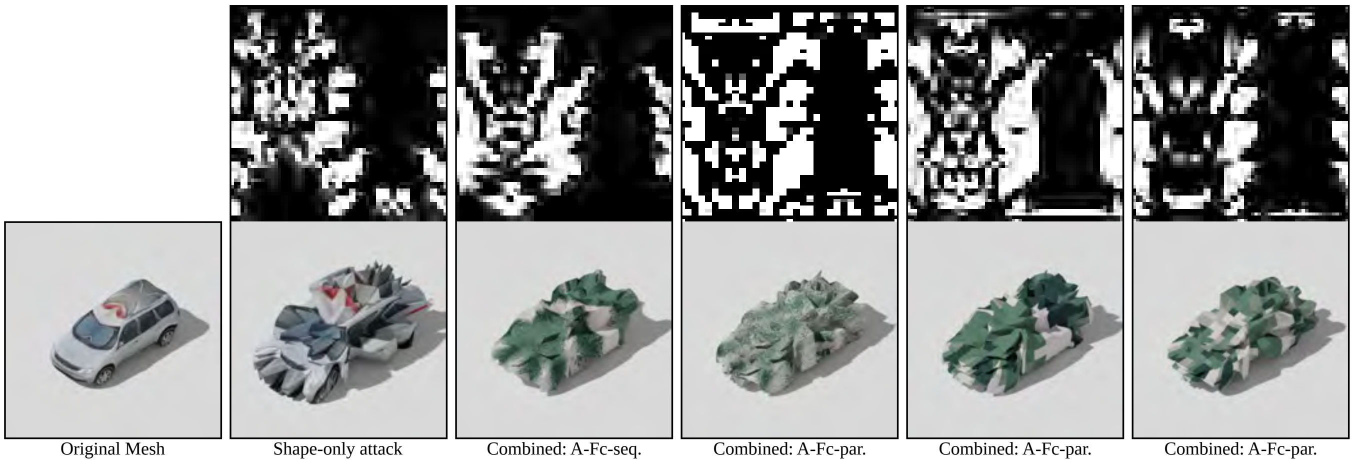


Fig. S19: Visualization of different shape-based attacks and their corresponding displacement maps.