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	I^{LINZ}	49.82%	93.50%	94.80%	91.22%
Faster R-CNN	I^{LINZ}	72.13%	80.34%	93.31%	81.71%
YOLOv5	I^{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:

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

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 $r = [r_1, r_2, \ldots]^{\text{T}}$, 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 $[s(p_{i,1}), \ldots, s(p_{i,N})]^{\text{T}}$. 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

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

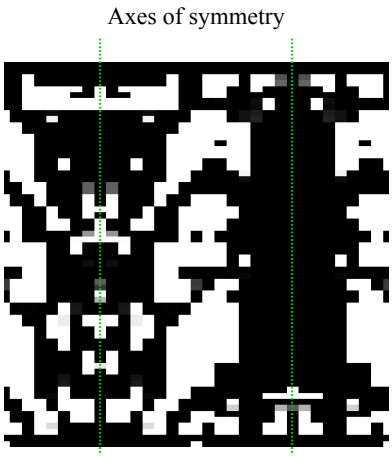


Fig. S1: An example of the axes of symmetry in a displacement map. The highlighted axes correspond to the central plane that cuts the mesh in two halves along its longitudinal direction.

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* $\text{ASR} = \frac{|V_{d,m}|}{|V_{d,d} \cup V_{d,m}|}$, 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_{or} and AP_{adv} respectively. We then obtain the average precision drop as $\text{APD} = \text{AP}_{\text{or}} - \text{AP}_{\text{adv}}$. 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 APD	Blender APD
	Pix	Lc	Fc	Ma				
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

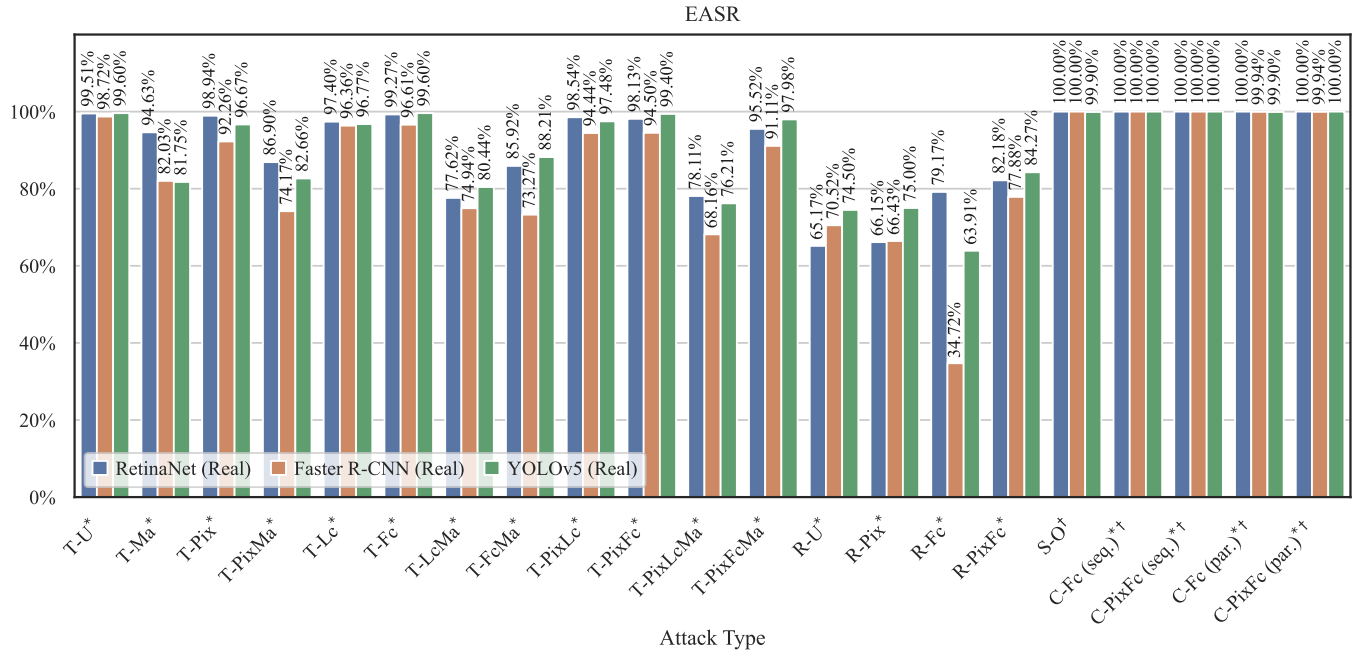


Fig. S2: Evaluation results of the models trained on real data and tested on the Blender-generated adversarial datasets. Dentations: *texture-only attacks, †shape-only attacks, and *†combined attacks.

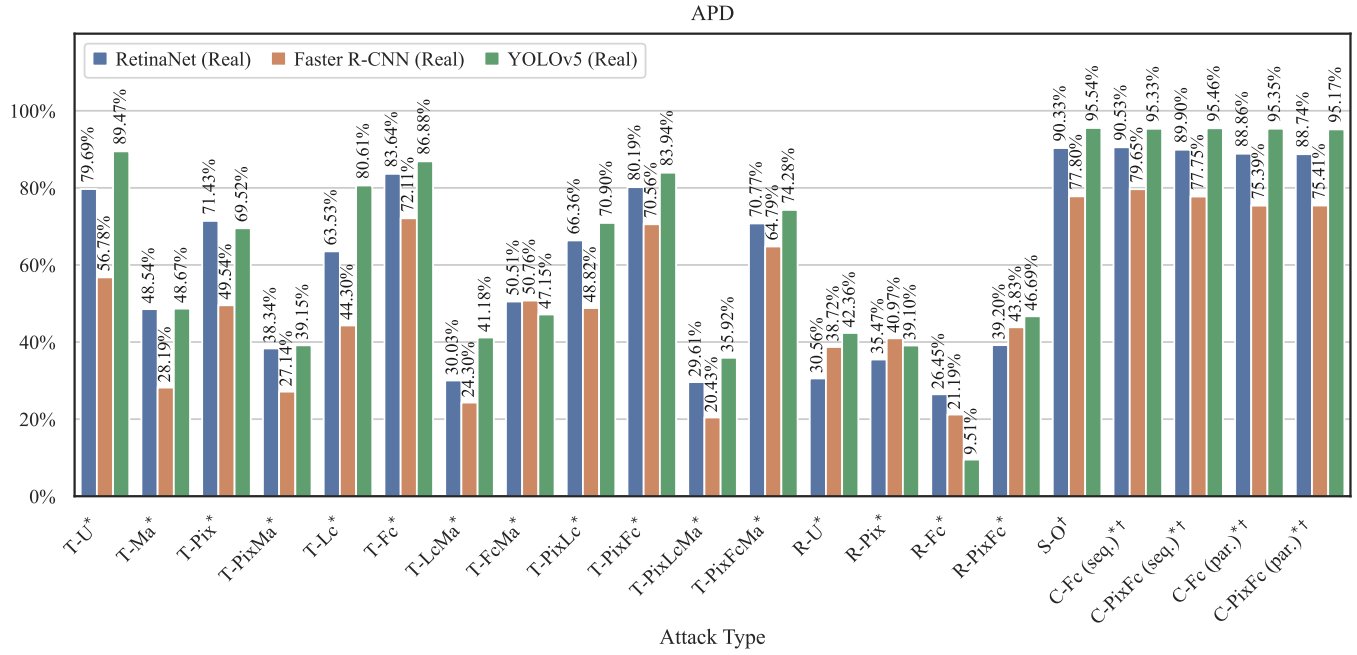


Fig. S3: Evaluation results of the models trained on real data and tested on the Blender-generated adversarial datasets. Dentations: *texture-only attacks, †shape-only attacks, and *†combined attacks.

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.

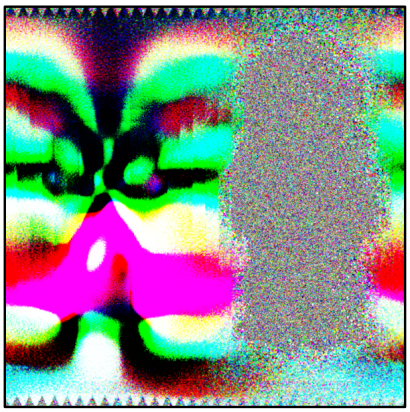
Du *et al.* make their adversarial patches public[10] in good quality, so we use them to compare. See the results of the

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

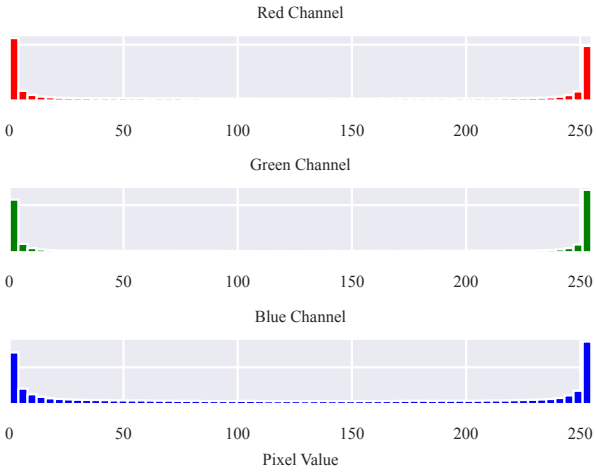
[9] https://jmlr.org/papers/v9/vandermaaten08a.html

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

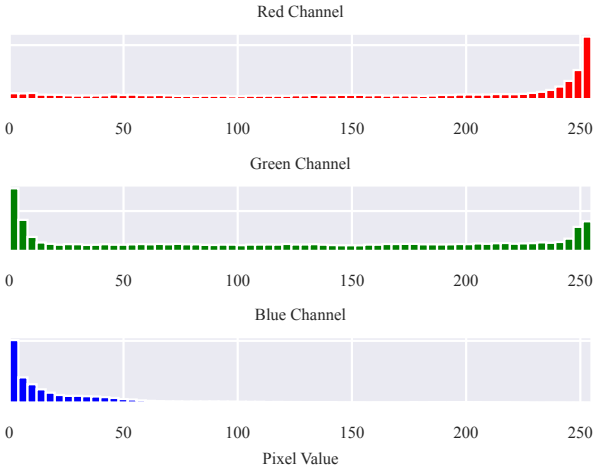
comparison in Figure S4. Interestingly, they conclude that weather augmentations do not considerably improve the results. However, we find that weather augmentations during optimization significantly affect the distribution of colors by pushing a significant fraction of pixels away from the edges of 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.)

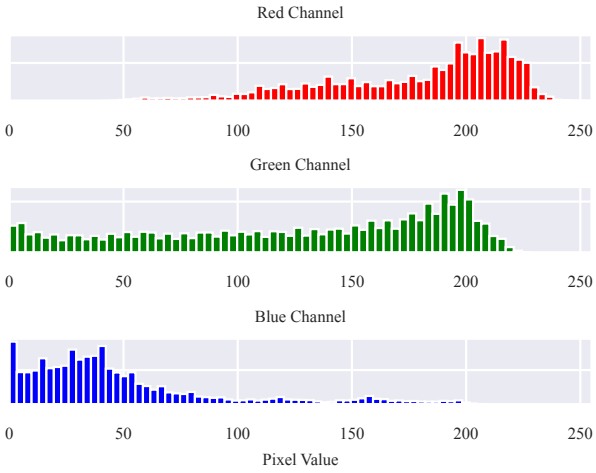


Fig. S4: Color distribution in adversarial patches.

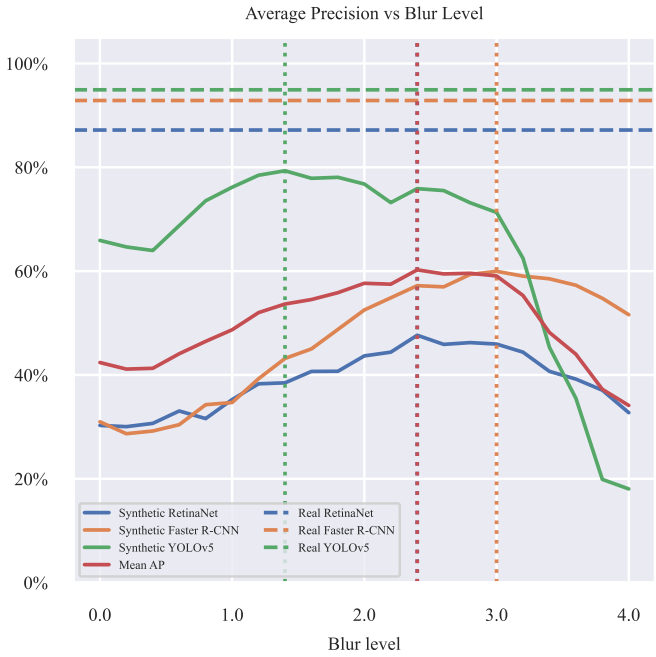


Fig. S5: Each point on the solid lines corresponds to a synthetic model trained using the corresponding blur level. Each such model is evaluated on the real validation set. The horizontal dashed lines represent the real models' performance on the real validation set. The vertical dotted lines represent 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.

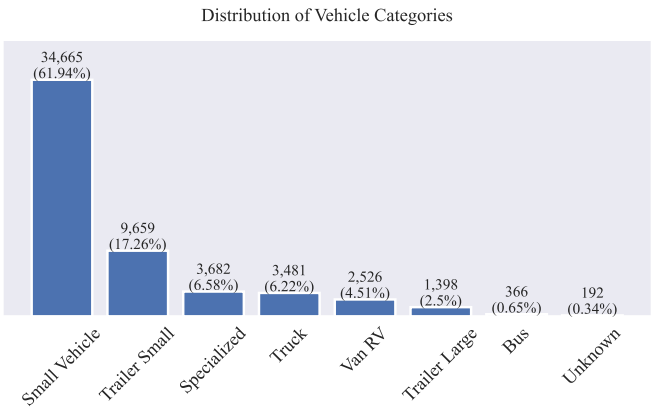


Fig. S6: Visualization of vehicle category distribution in the LINZ dataset: each bar signifies the number of samples associated with a specific vehicle category. The figures within the brackets indicate the proportion of total vehicles represented by each class.

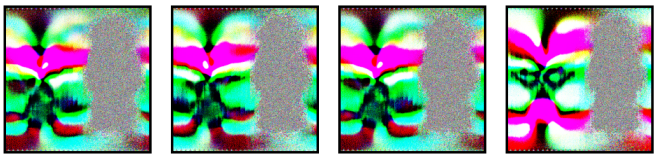


Fig. S7: Examples of T-U textures obtained with different initializations. The grey region maps to the underside of the car which is ignored by the adversarial optimization.

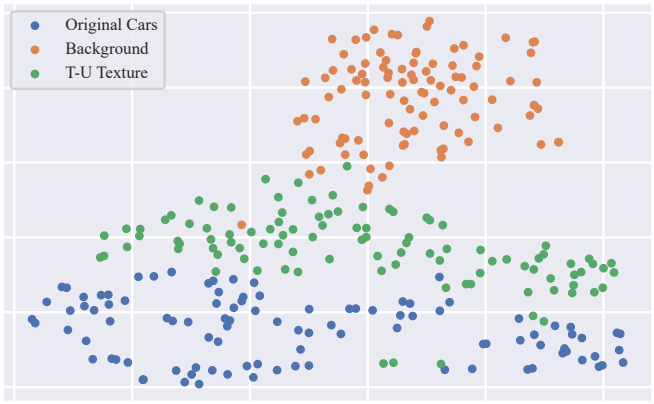


Fig. S8: Embeddings of background images and vehicles with original and T-U texture maps.

No anti-aliasing, no blurring

Anti-aliasing, no blurring

Anti-aliasing, blurring

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.

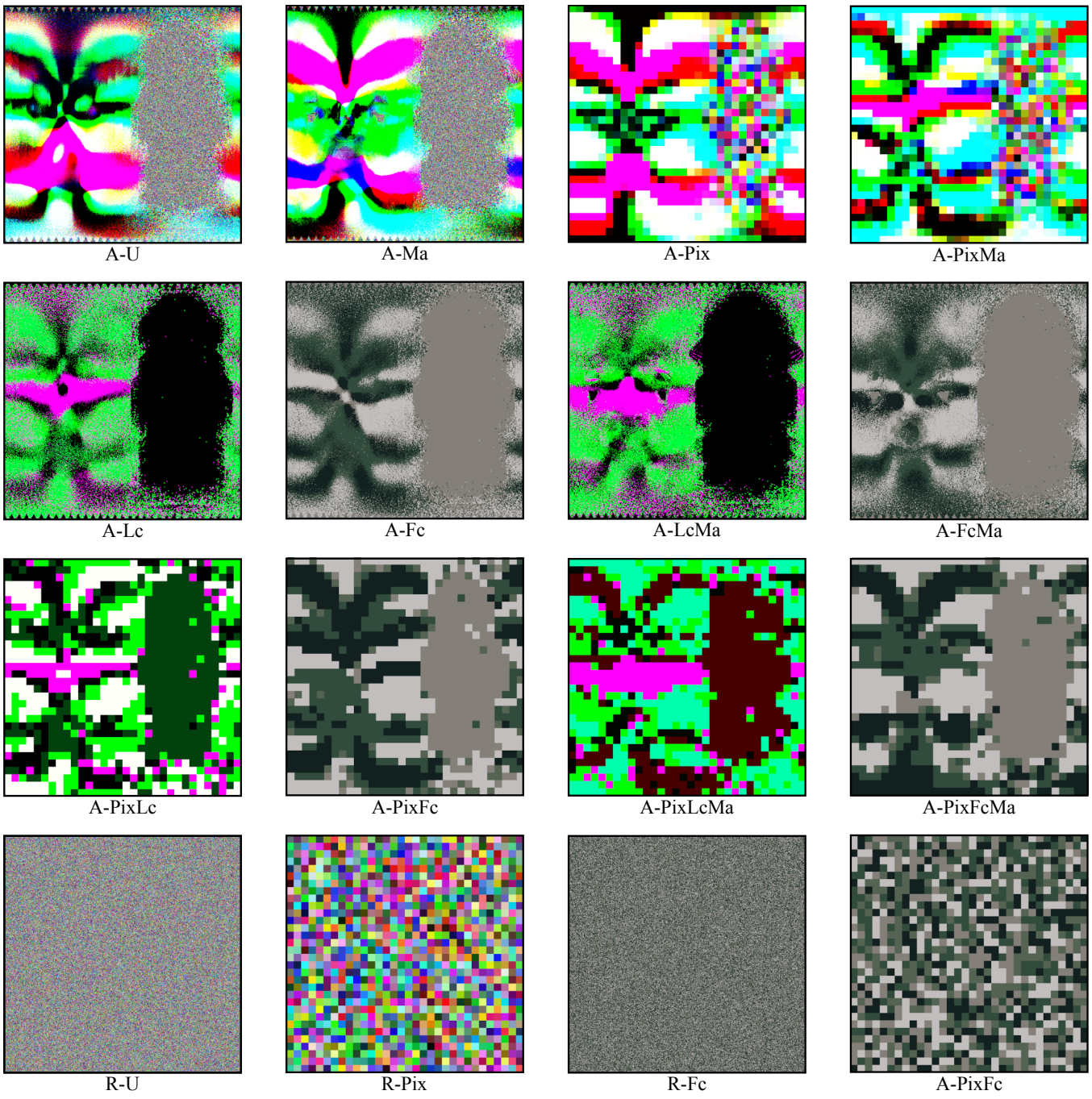


Fig. S14: Adversarial and random texture maps. The right side corresponds to the car's underside, which the adversarial optimization ignores.

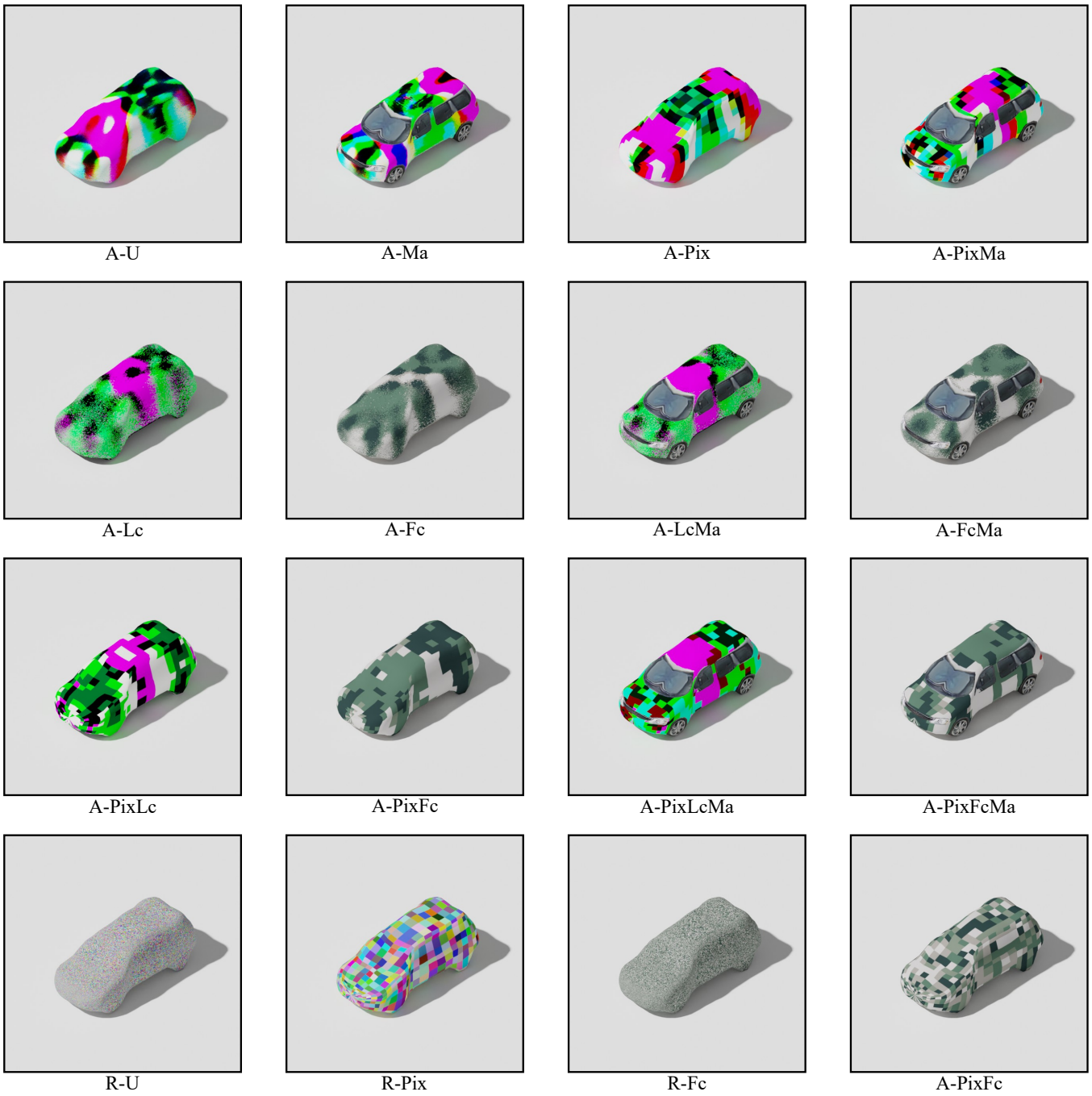


Fig. S15: Visualizations of the textures from Figure S14 applied to a car mesh.

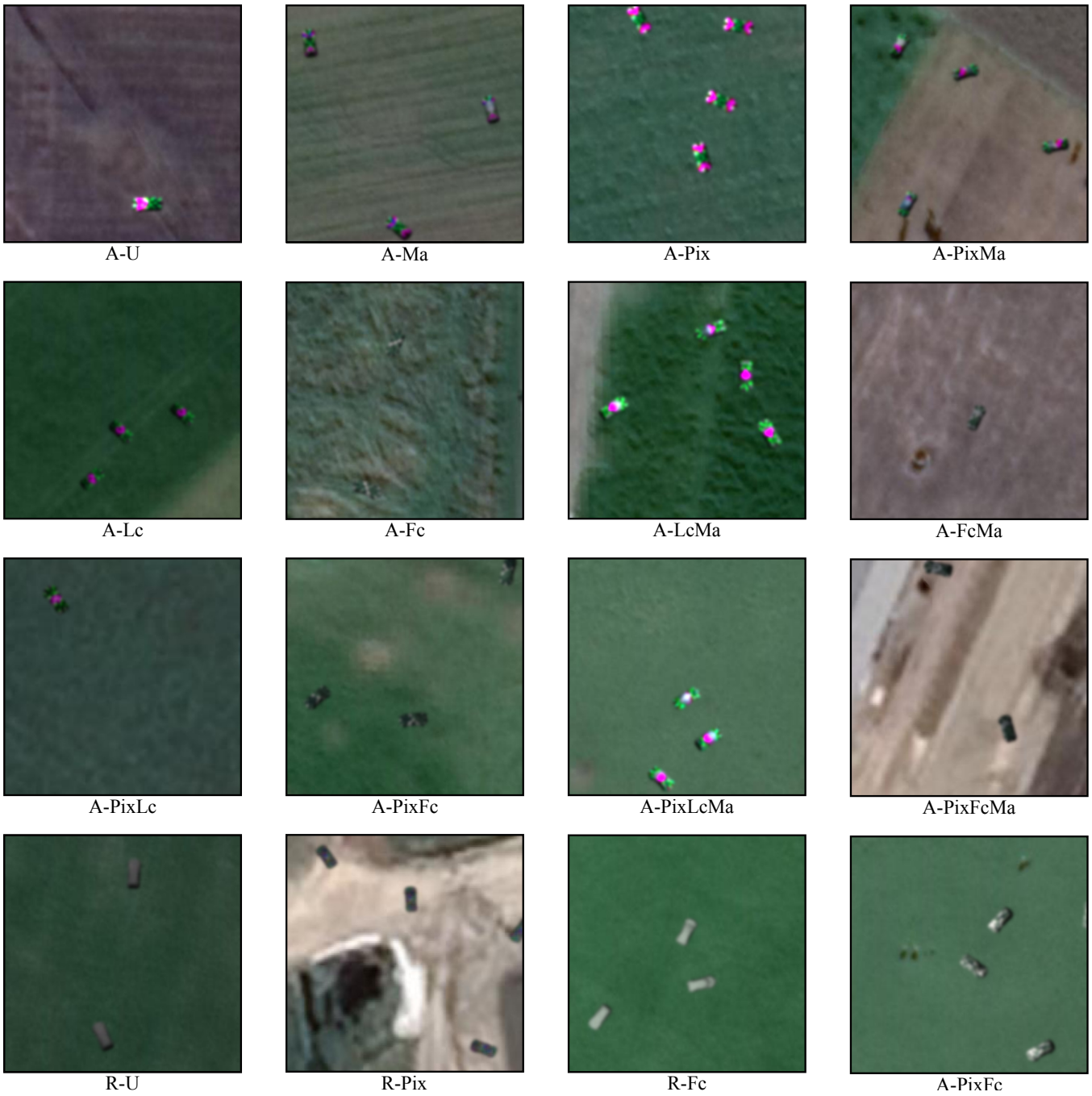


Fig. S16: Illustrations of vehicles sourced from the PT3D datasets featuring adversarial and random texture maps.

Fig. S17: Examples of the original Blender images.

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

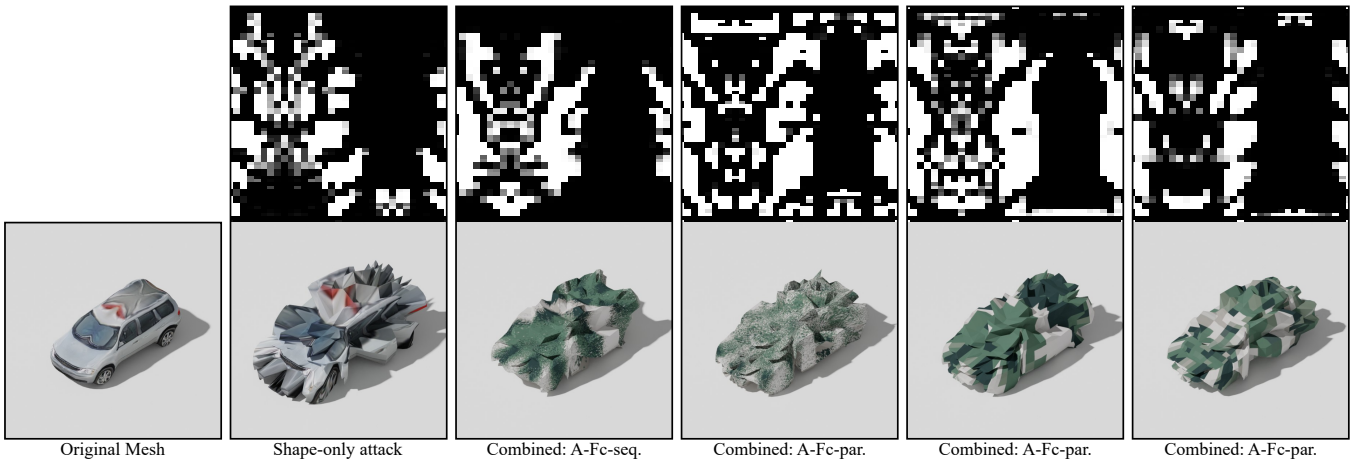


Fig. S19: Visualization of different shape-based attacks and their corresponding displacement maps.