# Supplementary Material for Infrared Adversarial Car Stickers

Xiaopei Zhu[1] Yuqiu Liu[2] Zhanhao Hu[3] Jianmin Li[4] Xiaolin Hu[4,5,6*]
[1]School of Integrated Circuits, Tsinghua University, Beijing, China
[2]Department of Technology, Beijing Forestry University, Beijing, China
[3]Department of Electrical Engineering and Computer Sciences, UC Berkeley, California, USA
[4]Department of Computer Science and Technology, Institute for Artificial Intelligence, BNRist, Tsinghua University, Beijing, China
[5]THBI, IDG/McGovern Institute for Brain Research, Tsinghua University, Beijing, China
[6]Chinese Institute for Brain Research (CIBR), Beijing, China
{zxp18}@mails.tsinghua.edu.cn
{liuyuqiu99, zhanhaohu.cs}@gmail.com
{lijianmin, xlhu}@mail.tsinghua.edu.cn

## 1. Supplementary Video 1

Watch "SM Video 1 Sticker Manufacture.mp4" for physical implementation process for adversarial real car stickers.

## 2. Supplementary Video 2

Watch "SM Video 2 Model Car Attack.mp4" for the demo of physical attacks on model cars.

## 3. Supplementary Video 3

Watch "SM Video 3 Real Car Attack.mp4" for the demo of physical attacks on real cars.

## 4. Details for Building a 3D Infrared Car Model

Building a 3D infrared car model requires taking infrared photos at different viewing angles. We used a FLIR T560 infrared camera mounted on a tripod. The distance between the tripod and the car is about 1m, and height of tripod can be adjusted from 1m to 2m. The camera lens could be rotated horizontally or tilted vertically to capture images from different angles. We captured infrared photos from five typical angles, including a side view (for creating textures of the car doors, side windows, tires, etc.), a front view (for creating textures of the car front), a front view with a downward angle (for creating textures of the front windows, hood, the front half of the roof, etc.), a horizontal back view (for creating textures of the car rear), and a back view with a downward angle (for creating textures of the rear windows, rear hood, the rear half of the roof, etc.).

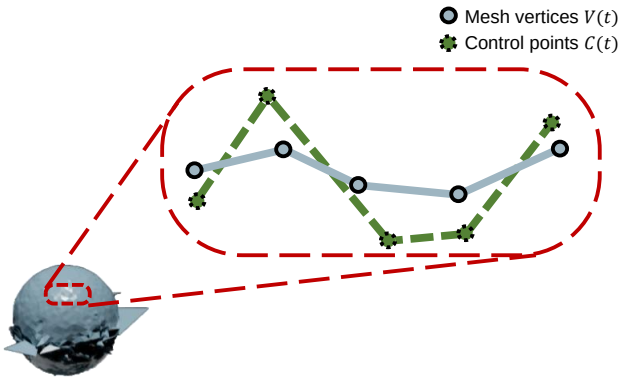
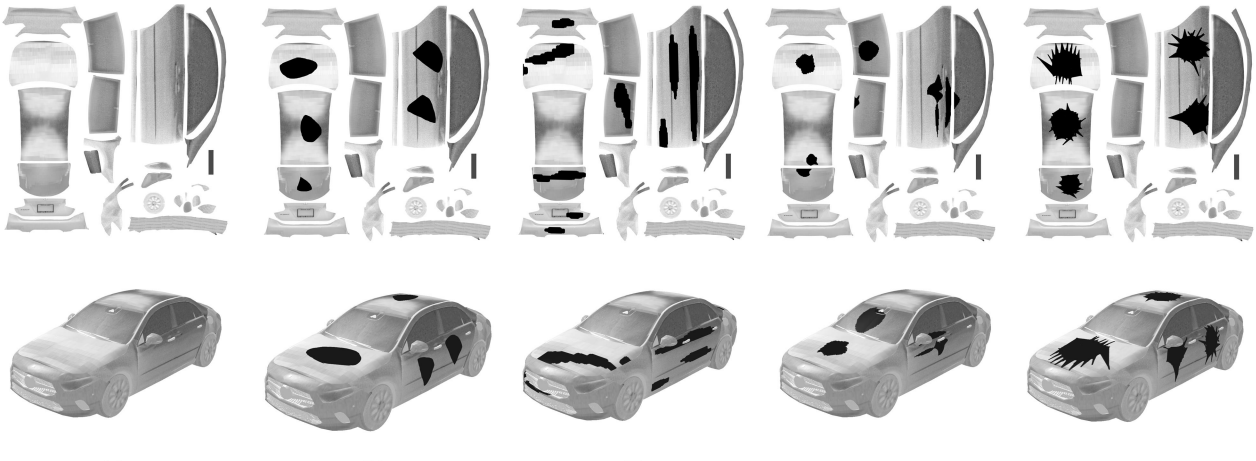



Figure S1. Schematic diagram of 3D control points-based smoothing.

Next, we used Photoshop software to crop the captured infrared photos into different parts, such as doors, windows, etc. These cropped images were then pasted onto the corresponding parts of the car's faces map (Figure 2(b)). It's important to note that the cropped images may not perfectly align with the corresponding parts in the faces map. In such cases, we utilized the distortion function in Photoshop to slightly deform the cropped infrared images, ensuring a better fit with the corresponding parts on the faces map.

One special part is the roof. Limited by the capturing equipment, the photos of the car roof often had a large tilt, and sometimes the complete car roof was not captured in one shot. First, we utilized the deformation function in Photoshop to deform the roof photo to resemble a top-down view. If the deformed roof photo wasn't complete, we repeated the deformation process with an infrared photo cap-

*Corresponding author.



(a) (b) (c) (d) (e)

Figure S2. Different car models and their textures. (a) Clean. (b) Random shape. (c) AIP. (d) UAP. (e) Ours.

tured from another angle. Subsequently, we stitched the two photos together to obtain a complete image of the car's roof texture.

Through the above steps, we can use the captured infrared photos to fill the faces map of the car (Figure 2(b)), obtaining the infrared texture map of the car (Figure 2(c)), which could be used to render the 3D car model. The rendered infrared car model is shown in Figure 2(d) .

## 5. Details for 3D Control Points-Based Smoothing

If we directly optimize the vertices coordinates $V$ of the mesh $M_{adv}$, many "peaks" will appear on the mesh surface, which makes the shadow $S_{adv}$ very complex and brings difficulties to the physical implementation. Inspired by the Gaussian smoothing and spline interpolation method, we propose a smoothing algorithm for 3D mesh vertices. We use a set of 3D control points $C$ as anchor points, and compute the offset $\Delta V$ of the mesh vertices $V$ by the weighted average of the offsets $\Delta C$ of the control points $C$. The offsets indicate the difference between the current coordinates and the initial coordinates.

Specifically, the offsets of the mesh vertices and control points at $t$-th time step are defined by

$$\Delta V_i(t) = V_i(t) - V_i(0), \tag{1}$$
$$\Delta C_i(t) = C_i(t) - C_i(0), \tag{2}$$

respectively. Assuming that there are $N_v$ mesh vertices and $N_c$ control points, the offset of the $i$-th mesh vertices at the $t$-th time step is computed by

$$\Delta V_i(t) = \sum_{j=1}^{N_c} w_{ij} \Delta C_j(t). \tag{3}$$

The weight

$$w_{ij} = \frac{\exp(-\frac{D(V_i(0), C_j(0))^2}{\sigma^2})}{\Gamma_j}, \tag{4}$$

$$\Gamma_j = \sum_{i=1}^{N_v} \exp(-\frac{D(V_i(0), C_j(0))^2}{\sigma^2}), \tag{5}$$

where $D(\cdot, \cdot)$ denotes the Euclidean distance function. The weight $w_{ij}$ increases as the distance between the $i$-th initial mesh vertex and the $j$-th initial control point becomes smaller. At each step $t$, the coordinate of the $i$-th mesh vertex is then calculated by

$$V_i(t) = V_i(0) + \Delta V_i(t). \tag{6}$$

We denote the above operation by $\Theta$ and have

$$V(t) = \Theta(C(t)). \tag{7}$$

Figure S1 shows the schematic diagram of 3D control points-based smoothing method.

## 6. Details for Attacking Faseter RCNN in the Digital World

We simulated $N = 5$ adversarial patterns $S_{\text{adv}}^{(i)}, i = 1, 2, ...5$, with one pasted on the car roof and two each on the car doors (since the car doors are symmetrical, the patterns on the left and right doors are identical), one on the car hood, and one on the car rear. These five adversarial patterns $S_{\text{adv}}^{(i)}, i = 1, 2, ...5$, were the shadows of five adversarial meshes $M_{\text{adv}}^{(i)}, i = 1, 2, ...5$. The mesh shadowing angles $\varphi^{(i)}, i = 1, 2, ...5$ were all initialized to 0. Each adversarial mesh $M_{\text{adv}}$ is initialized as a standard sphere (with 15,360

vertices $V$ and 15,360 control points $C$), where the coordinates of control points $C$ were initialized as the same as that of mesh vertices $V$. We established a Cartesian coordinate system centered at the midpoint of the infrared car's texture map $T_{\text{origin}}$, where the left and top of $T_{\text{origin}}$ represented the positive directions of the x and y axes, respectively, and defined the side length of $T_{\text{origin}}$ as 2. The pasting positions $P = \{p^{(1)}, ..., p^{(5)}\}$of $S^{(i)}_{\text{adv}}, i = 1, 2, ...5$, were initialized as $[-0.38, 0.6]$, $[-0.38, 0.1]$, $[0.62, 0.5]$, $[0.62, 0]$, $[0.62, -0.5]$. These initial coordinates corresponded to the centers of different parts of the car, for example, $[0.62, 0]$ represented the center of the car roof. Using the Equation 7 (main submission), we obtained the car's texture map $T_{\text{adv}}$ with adversarial shadows.

Next, we employed the differentiable renderer $R$ provided by Pytorch3D to render $T_{\text{adv}}$ onto the Mercedes-Benz car model, resulting in the rendered infrared images $I_{\text{adv}}$. The parameters $\theta$ of the renderer $R$ were set such that, in each iteration, the virtual camera's horizontal angle from the car ranged from 0 to 360 degrees with random variations, the pitch angle from the car varied from 0 to 90 degrees with random variations, and the distance from the car ranged from 1m to 8m with random variations, simulating various viewpoints of a real car. Subsequently, we inputted the rendered images into Faster R-CNN and computed the loss function $L$ according to Equation 10 (main submission). The weights $w_1$, $w_2$, $w_3$, and $w_4$ in loss function $L$ were 0.01, 1.0, 1.0, 0.1, respectively. We updated the optimization parameters $\left(C^{(i)}, P^{(i)}, \varphi^{(i)}\right), i = 1, 2, ...5$, using the backpropagation algorithm. We utilized the Adam optimizer with a learning rate of 0.003. Optimization was performed on a single A100 GPU for five epochs, with 1491 iterations per epoch, taking approximately 2.5 hours to complete. After optimization, we obtained the adversarial shadow patterns (Figure S2(e), top), and the rendered car with adversarial shadow patterns (Figure S2(e), bottom).

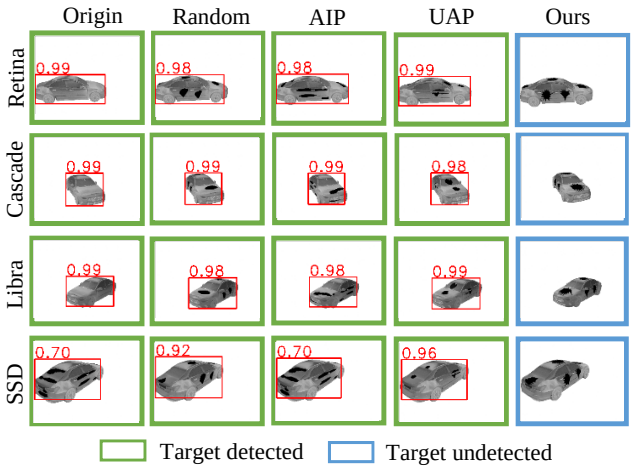



Figure S3. Examples of detection results of more detectors for target cars with different textures. The numbers above the red bounding boxes are the object confidence scores, with a threshold of 0.6.

After that, we evaluated the attack effectiveness of the adversarial shadow patterns. For a fair comparison, we employed the original car pattern (without any sticker, Figure S2(a)) and random shape patterns (manually painted using Photoshop software, Figure S2(b)) as control patterns. These patterns were rendered onto the same car model, and the resulting images were inputted into Faster R-CNN.

## 7. Examples of Attacking More Detectors

Figure 5 shows one set of typical examples of detection results of Faster RCNN, YOLOv3, Deformable DETR for target cars with different textures. We added typical examples of detection results of RetinaNet, Cascade RCNN, Libra RCNN, and SSD in Figure S3.

## 8. Details for Ablation Study

To evaluate the effectiveness of the 3D control points-based mesh smoothing algorithm (CMS) and a set of smoothing losses (SMLS), we performed ablation experiments. The loss function settings include using only $L_{\text{det}}$, using $L_{\text{det}}$ and CMS, using $L_{\text{det}}$ and SMLS, and using $L_{\text{det}}$, CMS, and SMLS. Other optimization settings were consistent with Section 6 in *Supplementary Material*. Through optimization, we obtained 3D adversarial meshes and 2D adversarial patterns using different loss functions, as shown in Figure S4. Note that in Figure S4 we show a typical adversarial mesh (of the 5 adversarial meshes) and its corresponding shadow for each case for convenience.

We conducted a subjective evaluation on the smoothness scores of the 3D adversarial meshes and 2D adversarial patterns. The experiments were approved by the Institutional Review Board (IRB). Scores ranged from 1 to 10, with higher scores indicating smoother results. We invited 10 volunteers (Age 22-28, 5 male, 5 female) to rate and calculated the average and variance. We also evaluated the physical implementation time of 2D adversarial patterns optimized with four combinations of loss functions as mentioned above. Two volunteers manufactured the adversarial stickers according to the different optimized patterns. We recorded the implementation time of different stickers as shown in Table S1. The results indicate that both CMS and SMLS improved the smoothness of adversarial meshes and patterns, and their combination was better. Besides, these methods effectively reduced the physical implementation time of adversarial patterns. We also found that there was a trade-off between smoothness and ASR. Improving the smoothness of adversarial meshes and patterns would decrease the ASR of adversarial patterns but would save physical implementation time of these adversarial patterns.

Table S1. Ablation study

| Setting \ Metrics | 3D mesh | 2D shadow | ASR (%) | Smoothness(1-10) | Implementation time (h) |
|---|---|---|---|---|---|
| $L_{\text{det}}$ | Fig S4(a), top | Fig S4(a), bottom | 98.46 | 2.6±1.0 | around 2.5 |
| $L_{\text{det}}$+CMS | Fig S4(b), top | Fig S4(b), bottom | 96.71 | 4.8±1.0 | around 1.6 |
| $L_{\text{det}}$+SMLS | Fig S4(c), top | Fig S4(c), bottom | 96.98 | 6.1±0.7 | around 1.1 |
| $L_{\text{det}}$+CMS+SMLS | Fig S4(d), top | Fig S4(d), bottom | 96.31 | 7.7±1.1 | around 0.8 |

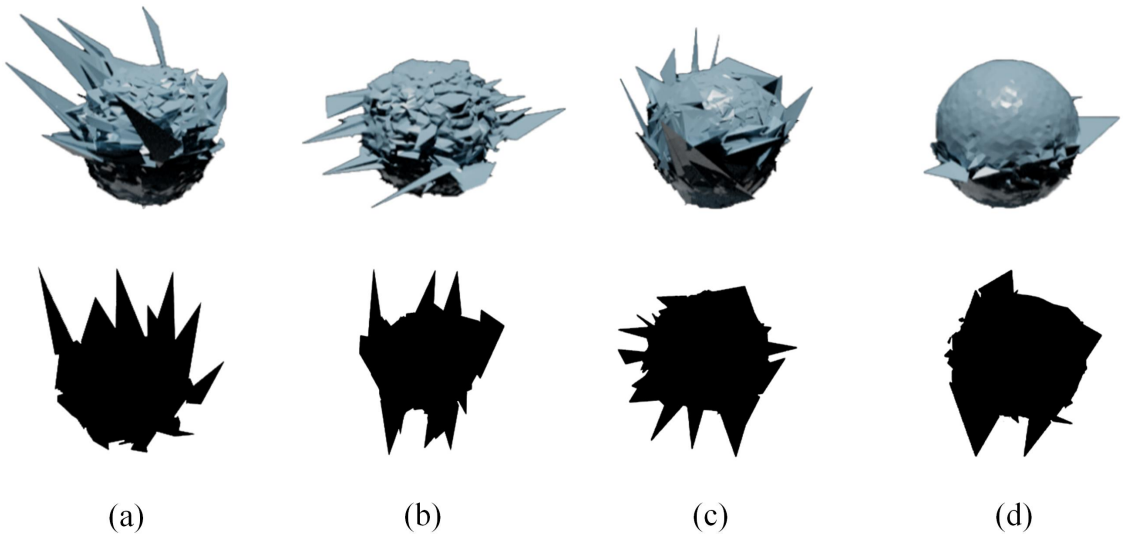



Figure S4. Typical 3D adversarial mesh and its 2D adversarial shadow under different optimization settings. (a) $L_{\text{det}}$ (b) $L_{\text{det}}$+CMS (c) $L_{\text{det}}$+SMLS (d) $L_{\text{det}}$+CMS+SMLS

### 8.1. Details for Exploring the Interpretability of the Attack

To gain deeper insights into our attack methods, we utilized the GradCAM [7] technique to analyze the changes in network attention maps before and after the attack. We employed the pytorch-grad-cam [2] library and extracted attention features using a ResNet50 model, which is the same architecture as the backbone of our Faster R-CNN model. The target class was set as "car". Figure S5 illustrates the attention maps before and after the attack. After our attack, the network's attention to cars significantly weakens or disappears, potentially leading to incorrect judgments. This, to some extent, explains the effectiveness of our method.

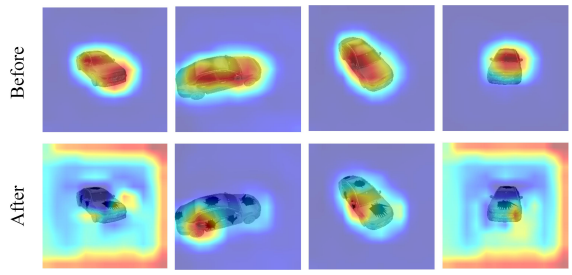



Figure S5. The attention map generated by GradCAM (a) before and (b) after our attack.

## 9. Details for Comparison with 2D Optimization Methods

We extended the previous 2D infrared model car attack methods [8, 9] to our 3D car model. We generated adversarial car textures on our car model based on the original papers [8, 9] and codes. As these two methods need to optimize the adversarial patterns for each image, we sampled 4 images of our 3D infrared car model from four typical viewing angles, including the front view (horizontal angle 0 degree, pitch angle 0 degree), side view (horizontal angle 90 degree, pitch angle 0 degree), top view (horizontal angle 0 degree, pitch angle 90 degree), back view (horizontal angle 180 degree, pitch angle 0 degree). Following the settings and hyper-parameters in the original papers [8, 9] and codes, we optimized the adversarial patterns for each image, and then pasted the adversarial patterns onto the cor-

Table S2. Defense Methods

| Defense Methods | ASR | ASR drop by |
|---|---|---|
| No defense | 96.31% | 0.00% |
| Adversarial Training | 88.83% | 7.48% |
| PixelMask | 94.81% | 1.50% |
| Bit squeezing | 90.62% | 5.69% |
| JPEG compression | 93.91% | 2.40% |
| Total variation minimization | 92.12% | 4.19% |

responding parts in the *texture map* (Figure S2(a), top) of our 3D car model. For example, we optimized the adversarial patterns based on the infrared image sampled from the front view, and after that we pasted the adversarial patterns on the textures of the car front in the *texture map*. Then we obtained the adversarial car texture of AIP (Figure S2(c)), and the adversarial car texture of UAP (Figure S2(d)).

## 10. Details for Adversarial Defense

We tested five adversarial defense methods to defend our attack methods in the digital world, including adversarial training [3], PixelMask [1], Bit squeezing [10], JPEG compression [4] and Total variation minimization [4]. For adversarial training, we used the data augmentation method. The adversarial training data contains clean images and adversarial images collected from real world, and the ratio of adversarial images to clean images was 1:9. For PixelMask, we used a 20×20 pixel mask, which was applied to erase the adversarial car texture at random positions. For Bit squeezing, we used the Numpy [5] library to reduce the 8-bit adversarial images to 7-bit depth. For JPEG compression, we used the JpegCompression module in the Adversarial Robustness Toolbox [6] library, and set the compression rate to 90%. For Total Variance Minimization, we used the Total Variance Minimization module in Adversarial Robustness Toolbox [6] library. Other optimization settings were the same as that of Section 4.4 in the main submission. We used the adversarial texture shown in Figure S2(e) to test the effectiveness of these defense methods. The results are shown in Table S2.

The results show that although these methods had a certain defense effect, the ASR of our method still reached 88.83%-94.81% after adding defense, which indicates that our method is a powerful attack method.

## References


[1] Akshay Agarwal, Mayank Vatsa, Richa Singh, and Nalini Ratha. Cognitive data augmentation for adversarial defense via pixel masking. *Pattern Recognition Letters*, 146:244–251, 2021. 5

[2] Jacob Gildenblat and contributors. Pytorch library for cam methods. https://github.com/jacobgil/pytorch-grad-cam, 2021. 4

[3] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Int. Conf. Learn. Represent.*, 2015. 5

[4] Chuan Guo, Mayank Rana, Moustapha Cissé, and Laurens van der Maaten. Countering adversarial images using input transformations. In *Int. Conf. Learn. Represent.*, 2018. 5

[5] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. 5

[6] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox v1.2.0. *CoRR*, 1807.01069, 2018. 5

[7] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017. 4

[8] Xingxing Wei, Yao Huang, Yitong Sun, and Jie Yu. Unified adversarial patch for cross-modal attacks in the physical world. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4445–4454, 2023. 4

[9] Xingxing Wei, Jie Yu, and Yao Huang. Physically adversarial infrared patches with learnable shapes and locations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12334–12342, 2023. 4

[10] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS*, 2018. 5