

# GeoDiffusion: A Training-Free Framework for Accurate 3D Geometric Conditioning in Image Generation

## Supplementary Material

### A. GeoDiffusion Methodology Details

#### A.1. Preliminary on Diffusion Models

Diffusion models are generative models that create data by reversing a gradual noising process. In the forward diffusion process, we repeatedly add Gaussian noise  $\epsilon_t \sim \mathcal{N}$  to the original image  $x_0 \in [-1, 1]^{3 \times H \times W}$   $t \in [1, T]$  timesteps. This results in a sequence of noisy image representations  $x_t$ , where  $x_T$  approximates pure Gaussian noise  $x_T \sim \mathcal{N}(0, 1)$ .

To synthesize novel images, we gradually denoise a sample  $x_T$  by training a neural network  $\theta(x_t, t)$  to predict the original data from noisy inputs. For latent diffusion models, such as Stable Diffusion [35], this process is carried out in a lower dimensional latent space  $z$  instead of the image space  $x$ .  $z_t$  therefore denotes our latent variable at timestep  $t$ .

At each timestep  $t$ , we calculate the less noisy latent  $z_{t-1}$  from  $z_t$  by predicting the added noise for the timestep through  $\epsilon_\theta(z_t, t)$ . The neural network  $\epsilon_\theta$  is typically implemented using a U-Net architecture [36]. The U-Net is a convolutional neural network architecture that effectively decomposes the latent representation into different feature levels, capturing both local and global image structures by progressively downsampling the input to capture contextual information, and subsequently upsampling it to recover the spatial resolution.

#### A.2. 3D Object Rendering

In Section 3.1, we introduce the methodology for rendering a selected viewpoint of the 3D object in Blender. Representing any object with a known viewpoint is essential for accurate, 3D geometric modifications in image space. To allow for maximum flexibility, we opt to utilize a single 3D object as a starting point. Domain-specific 3D objects can be obtained from publicly available datasets such as ShapeNet [6], Objaverse [8] or OmniObject3D [48]. There also exist domain-specific 3D databases like DivAerNet [12] for cars. Inspired by the approach in [27], our approach loads the 3D object into a Blender scene and subsequently defines the desired position of the camera  $c$  through its distance to the object  $r$ , its elevation angle  $\theta$  and the azimuth angle  $\phi$ :

$$c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = r \cdot \begin{bmatrix} \sin(\theta) \cdot \cos(\phi) \\ \sin(\theta) \cdot \sin(\phi) \\ \cos(\theta) \end{bmatrix}. \quad (9)$$

The camera's forward direction  $f$ , its right vector  $r$  and its true up vector  $u$  are defined as:

$$f = \frac{-c}{\|c\|}, r = \frac{u_{world} \times f}{\|u_{world} \times f\|}, \text{ and } u = f \times r. \quad (10)$$

The synthesis of the reference image  $I_{ref}$ , that later acts as input for the style transfer and geometric modification, is defined as:

$$I_{ref} = \mathcal{R}(\mathcal{O}, c, K, L), \quad (11)$$

where  $\mathcal{R}$  is the rendering function and  $K$  and  $L$  are the camera intrinsic parameters and lighting conditions respectively.

Within the Blender scene, we additionally define the points that are later used for geometric control of the object. Since they are defined in 3D, we need to project them into the 2D image space. To project the points  $P_{ref}$  in the global 3D coordinate system into pixel coordinates, we first transform them into camera coordinates  $P_{cam} = R \cdot (P_{ref} - c)$ , where  $R$  is the camera's rotation matrix. Given the principal point  $(c_x, c_y)$  set to the center of the image, the pixel plane coordinates  $u$  and  $v$  of the points are calculated as  $u = f \cdot x' + c_x$  and  $v = f \cdot y' + c_y$  with  $x' = \frac{x_{cam}}{z_{cam}}$ ,  $y' = \frac{y_{cam}}{z_{cam}}$ .

#### A.3. Motivation for using 3D points as prior

We are aiming to manipulate the 3D spatial structure of an object within a 2D image, as explained in Section 3.1. For example, this could be the length of a car that should be set precisely. Therefore, we have to derive where to place or guide the keypoints of the object, such that the desired spatial structure is fulfilled. Achieving this with only 2D points is challenging as not all geometric modifications of 3D features can be accounted for in 2D image space. Figure 7 shows an example of this, where it is possible to modify the object's geometry in  $x$ - and  $y$ -direction, i.e. making it longer or higher, in both the side view (top row) and the diagonal front view (bottom row). When the modifications are additionally supposed to include the  $z$ -dimension (e.g. making the object higher and wider), this is only possible for the viewpoint in the bottom row.

#### A.4. Point Translation Function

A key contribution of our framework is the possibility to define a point translation function that allows the user to modify the geometry by editing features of the object instead of manually defining handle and target points in an image. For a specific object category, the geometric points need to be defined in accordance with the 3D object used.

In the vehicle domain, we define 22 points in 3D space. The points mark the centers of the four wheels, the head and rear lights, the bumpers in the front and in the back, the



Figure 7. Translations of the geometric points. Depending on the viewpoint, geometric modifications of the object in 3D are possible or not. While making the object "longer" and "higher" is possible in both viewpoints, making it "higher and wider" is only possible in the bottom viewpoint.

lower ends of the front and rear windshields and the highest points of the roof. The translation logic defines that for modifications in length, the points marking the lights and the bumpers, both in the front and in the back, are moved evenly along the y-axis. The points in the front are moved in the positive y-direction while the ones in the back are moved in the negative y-direction. For modifications of the width, all points on either side of the object are moved accordingly along the x-axis. If the height of the object is edited, all points except the wheels and the bumpers are moved along the z-axis. An exemplary case is given in Figure 8.

Dimensions are specified in real-world units (meters). However, images lack absolute scales—the car depicted in Figure 8 could represent a length of either 5m or 0.5m. To correlate user inputs with image modifications, we define base dimensions for the reference object. For the example in Figure 8, we utilize real-world dimensions. During rendering, we normalize the 3D object so that its largest dimension corresponds to one unit length in the scene. For the car example, this is its length. Knowing that one unit in the scene represents, for example,  $length_{ref} = 5m$ , we derive the conversion factor between scene units and real-world measurements. In this example it means that  $1unit = 5m \Rightarrow 1m = 0.2units$ . This conversion factor allows us to translate user-specified geometric modifications into movements of points in the 3D scene.

If a user wants to increase the car’s length to 6m, the difference from the base length is 1m. Since we extend the car evenly at both the front and rear, we move the corresponding points by 0.5m each. Using the conversion factor, this translates to moving the points by  $0.5m \times 0.2units/m = 0.1units$ . By applying this conversion, we adjust the positions of points in the 3D space according to user inputs, maintaining real-world scale relationships.



Figure 8. Example of the point translation function for the vehicle domain. In this case. The dimensions of the reference object (left column) are  $width_r = 1.89m, length_r = 4.79m, height_r = 1.39m$  and the target dimensions of the vehicle are  $width = 1.95m, length = 5.391m, height = 1.544m$ . For  $\delta length$ . This results in the points marking the front lights and bumpers being moved by  $+0.3005m$  along the y-axis and the points for the rear lights and bumpers by  $-0.3005m$ . The change in width of  $\delta width = 0.06m$  leads to all points on the left half of the car being moved by  $+0.03m$  along the x-axis and all points on the right by  $-0.03m$ . For  $\delta height = 0.154m$ , all points except for the wheel centers and the bumpers are moved for  $+0.154m$  in z. Depending on the viewpoint, the visible geometry points are affected by the point translation function.

### A.5. Keypoint Detection with Diffusion Hyperfeatures

As stated in Section 3.2, we modify the style of the rendered reference image  $I_{ref}$  with PnP [42] image translation to obtain the image  $I^*$ . For consistent geometric modifications of  $I^*$ , we need to re-detect the annotated points  $P_{ref}$ . We therefore use  $I_{ref}$  and  $P_{ref}$  as references to detect the points  $P$  in the image  $I^*$ , using Diffusion Hyperfeature detection [26]. An example of the detected points in a style-transferred image is shown in Figure 9.



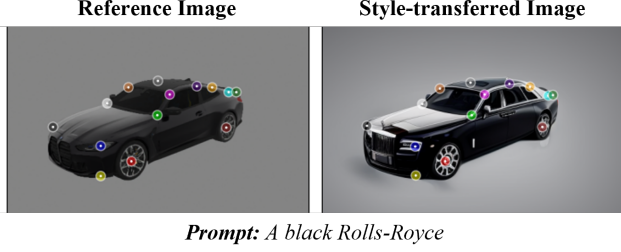


Figure 9. Geometric reference points before and after the style-transfer using Plug-and-Play Diffusion Features [42]. In the style-transferred image, the points have been detected using Diffusion Hyperfeatures [26] with the original image as reference.

## B. Details on GeoDiffusion

### B.1. Implementation Details

In our experiments, we utilize Stable Diffusion 2.1 [35]. Sampling and inversion are performed using DDIM [41] with 50 steps. For image-to-image style transfer, we employ the PnP method [42], utilizing the implementation provided by *Diffusers* [44]. Feature and attention maps are injected over 35 timesteps, applying classifier-free guidance [15] with a setting of 7.5. For geometric conditioning with GeoDrag,  $I^*$  is inverted to timestep 38. As done in existing drag-based methods [39, 54] to improve image consistency, we finetune the diffusion model on the input image using LoRA with rank 16 for 70 steps. We then perform  $K = 70$  total optimization steps, with the number of iterations per timestep set to  $B = 7$ , resulting in 10 timesteps in which dragging is conducted. For every optimization step  $k$ , we do  $J = 3$  motion supervision steps. For the optimization, we employ the Adam optimizer [21] with a learning rate of  $\eta = 0.02$ . The dragging step size per optimization is  $\beta = 4$ . For the comparison with existing approaches, we use the default parameters from the publications.

### B.2. Inference Speed

For assessing the speed of our framework, we consider the average durations of the steps on the geometry guidance dataset. All experiments are conducted on an NVIDIA RTX 6000 Ada GPU, featuring 48 GB of VRAM. Running the framework utilizes 21 GB on average.

The duration for modifying the geometry of an object in an image with our GeoDiffusion framework depends on multiple factors and parameters. When the framework is used from scratch, the duration for defining the geometrical reference points and the parametric relationships (point translation functions) depends on the complexity and the users' proficiency with Blender. Rendering the reference image in the desired viewpoint takes 3 seconds in our experiments.

The extraction of the feature and attention maps of the

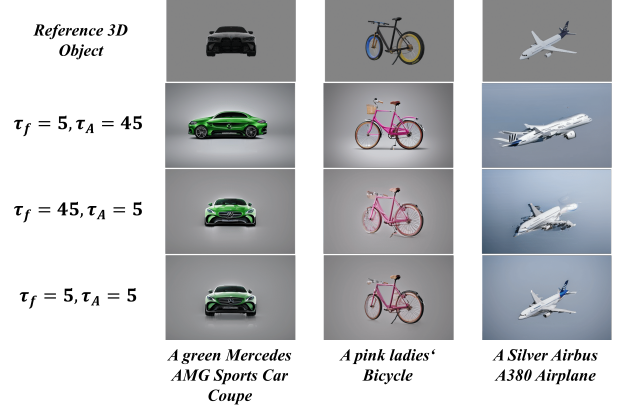


Figure 10. Image-to-image style transfer with different configurations of the feature- and attention injection thresholds  $\tau_f$  and  $\tau_A$ .

reference image using the PnP-method takes 70 seconds on average. For use cases where the same reference image is used repeatedly, the feature and attention maps can be pre-computed and cached, effectively reducing the time to zero. This is especially important for applications in industrial settings where the user only modifies the geometry of an existing object, as this reduces the skill barrier and increases efficiency. The subsequent PnP-feature injection and point detection take a combined 16 seconds.

The duration for geometric modification with our GeoDrag approach depends on the parameter configuration and the quantity of geometric points. As summarized in Table 7, our approach allows for a trade-off between accuracy and speed. The LoRA-finetuning step used in most dragging-based image editing methods consistently takes about 20 seconds in our experiments. Image refinement with SDXL adds 5 seconds to the inference time.

### B.3. Image-to-Image Style Transfer

Within our GeoDiffusion framework, test various configurations of the PnP-method [42] with regards to its ability to change the objects' style according to the user input while maintaining geometry and viewpoint, measuring the MD of the keypoints in the rendered reference image and the style-transferred image, CLIP- and HPSv2-scores and visually examining the results. High threshold values  $\tau_f$  and  $\tau_A$  indicate low impact of the feature and attention maps, as they are not injected into most of the generation process. We find setting the injection thresholds for the feature and self-attention maps to  $\tau_f = 15$ ,  $\tau_A = 15$ , and the classifier-free guidance [15] strength to 7.5 to produce the best results. When  $\tau_f$  is set to a higher value, the fidelity of the style-transferred image suffers. For high  $\tau_A$ , the attention injection is not sufficient to accurately guide the viewpoint. Some examples of this are visualized in Figure 10.

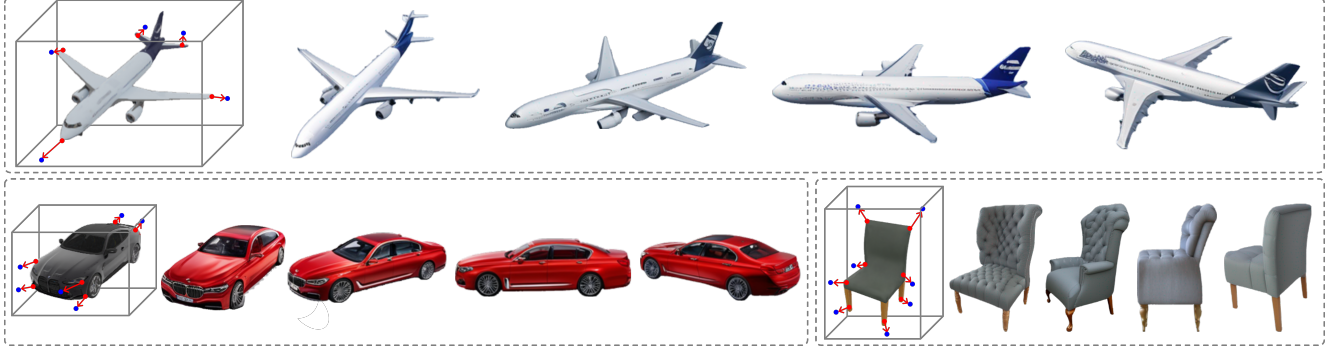


Figure 11. Modifications of geometries conducted on different viewpoints of the same object. The dragging instructions are set once for the 3D object and then carried out for the different viewpoints. The same text prompt is used for each viewpoint. Stylistic differences between different angles of the same object stem from the image-to-image translation with PnP [42] and image refinement with SDXL [34].

Table 3. Per-category quantitative metrics of GeoDiffusion on the Geometry Guidance Dataset.

	MD↓		CLIP↑		Time↓			
Dataset	GoodDrag	GeoDrag	GoodDrag	GeoDrag	GoodDrag	GeoDrag	Samples	Points
Cars	9.760	<b>8.626</b>	25.024	<b>25.131</b>	76.950	<b>74.327</b>	40	8-15
Bikes	12.871	<b>10.504</b>	25.455	<b>25.685</b>	57.438	<b>43.771</b>	18	6
Airplanes	15.386	<b>10.031</b>	23.175	<b>23.245</b>	44.111	<b>36.761</b>	12	4
Chairs	13.293	<b>9.802</b>	25.889	<b>26.490</b>	64.413	<b>43.414</b>	10	9
Buildings	11.823	<b>8.236</b>	24.460	<b>25.975</b>	46.935	<b>30.906</b>	10	4

## B.4. Experiment Details

**Qualitative Results:** In addition to Figure 4a, we provide qualitative comparisons with additional dragging frameworks in Figure 15. We also provide more qualitative results of geometry-guided image generations with GeoDiffusion in Figure 16 and Figure 17.

**Quantitative Results:** The Geometry Guidance dataset benchmarks the GeoDiffusion framework with cases from the application in engineering concept design. We added detailed per-category results in Table 3. Our improvements consistently hold across all categories tested, not merely cars. To robustly demonstrate general applicability beyond the engineering design domain, we separately evaluated our core GeoDrag module, responsible for accurate geometric modifications, using the general-purpose DragBench dataset, which addresses diverse object categories (animals, buildings, faces, etc.) and modification tasks (Figure 5, Figure 18). We provide new experimental examples for non-linear and more complex geometric conditions.

## B.5. Comparison of GeoDiffusion to Object Editing Frameworks

In Table 1, we compare GeoDiffusion to other frameworks that allow editing of object features. By using a class-specific 3D object as a prior for the generation, we enable the user to freely select a custom viewpoint. Staying true to the desired viewpoint is essential for accurate modifications of 3D object features in 2D image space, but remains a challenge in text-conditioned image generation, as shown in Figure 12.



Figure 12. Attempting to generate specific viewpoints of objects in image generation remains a challenge when relying on text-conditioning. For this example, Stable Diffusion 2.1 was used.

GeoDiffusion is the only framework that enables the user to implement parametric relations, i.e. in the form of equations, to correlate the geometric keypoints. This allows to directly introduce design rules into the geometric modification process. Other approaches require either manually setting the dragging instructions or modifying the 3D geometry by hand. Additionally, we allow editing the geometry in image space instead of relying on a 3D environment that poses a high skill barrier.

By utilizing a single, class-specific 3D object and subsequently modifying it, GeoDiffusion is essentially training-free. Other approaches require training of specialized embedding models or network adapters, limiting their direct applicability in custom domains for practitioners.

## B.6. Multi-View Image Modification

GeoDiffusion is able to perform the modification of 3D object features in 2D images for multiple viewpoints, seen in Figure 11. The geometry of the object is modified consistently. Minor semantic and stylistic changes between different viewpoints of an object stem from uncertainty in the utilized Plug-and-Play [42] style transfer. This is because there is no information flow between different viewpoints of the same object. Each viewpoint is generated separately. With this example, we aim to demonstrate that the geometry of the object is consistently modified according to the provided modification instructions. To achieve increased semantic

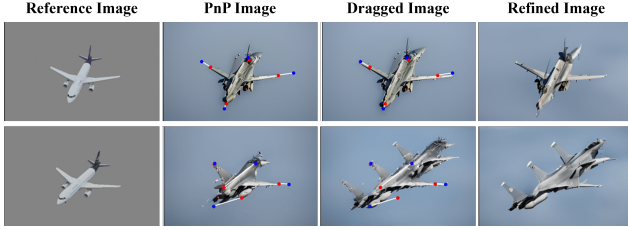


Figure 13. Failure cases in geometric modifications due to poor PnP image quality.

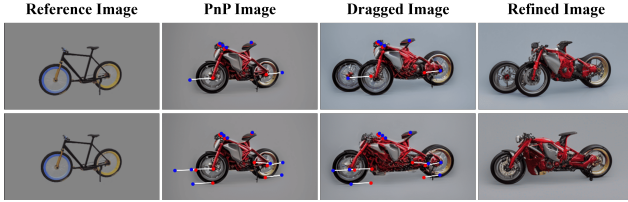


Figure 14. Utilizing multiple geometric points (bottom row) to move large object features over longer distances is advantageous compared to only using a single point (top row).

style consistency between the viewpoints, our method could be applied to dedicated multi-view image generation models such as SV3D [43].

### B.7. Limitations and Failure Cases

While GeoDiffusion effectively guides image viewpoint and geometry, it occasionally fails to achieve the desired results due to limitations in individual steps. For example, the PnP method may not fully align with the prompt. Occasionally, the poor quality in the PnP image cannot be rectified by image refinement alone, as shown in Figure 13. Severe perturbations in the PnP image cause keypoint detection to fail, leading to unsuccessful dragging. These issues are more prevalent in the airplane domain and occasionally with bikes.

Additionally, we observe instances where the dragging step does not fully succeed. This occurs in some bike test cases from our benchmark, where the wheels are not correctly moved to their target positions. A likely explanation is that large dragging distances make a single point insufficient for precise guidance. Our proposed solution is to leverage multiple points and increase the number of optimization steps to strengthen guidance toward the target geometry. This approach leads to the desired output as shown in Figure 14.

## C. Details on GeoDrag

### C.1. Methodology Details

We present a summarized representation of our GeoDrag algorithm for accurate dragging-based image modifications in the form of pseudo code Algorithm 1.

Table 4. Evaluation of GeoDrag configurations on the geometry guidance benchmark. The configurations are in the format  $(K, B, r_1, r_2, l, u)$ . Other parameters are kept consistent. The radii for the gradient mask and copy-&-paste mechanism are set to  $r_{grad} = r_{cp} = r_1$ .

Method	MD <sub>out</sub> ↓	CLIP <sub>out</sub> ↑	HPSv2 <sub>out</sub> ↑	Time↓
GoodDrag [54]	11.77	25.21	<b>25.93</b>	66.30s
GeoDrag <sub>(64,8,2,4,4,6)</sub>	14.19	25.21	25.78	<b>40.77s</b>
GeoDrag <sub>(70,7,2,7,4,6)</sub>	12.63	25.10	25.88	41.43s
GeoDrag <sub>(70,7,2,7,4,4)</sub>	11.89	25.65	<b>25.84</b>	44.40s
GeoDrag <sub>(64,8,4,8,4,4)</sub>	11.79	25.29	25.77	49.29s
GeoDrag <sub>(64,8,4,10,2,4)</sub>	11.51	25.58	25.75	53.84s
<b>GeoDrag<sub>(70,7,4,10,2,4)</sub></b>	<b>10.80</b>	25.31	25.71	60.03s

### C.2. Comparison to Alternatives

This section quantitatively evaluates GeoDrag and existing alternatives more in-depth. The results are summarized in Table 5, which is an extension to Section 4.3.2. In addition to the quantitative metrics presented in Table 2, we additionally measure MD, CLIP-score and HPSv2 before the image refinement with SDXL. We observe that the refinement decreases the dragging accuracy consistently for all evaluated methods. For GeoDrag, the MD is increased by 1.45 pixels on average, which yields a 13.4% decrease in accuracy.

Among all evaluated approaches, GeoDrag remains the most accurate before and after image refinement. Compared to GoodDrag, the MD improves by 2.17 pixels when measured before refinement and by 0.97 pixels when measured after refinement.

In terms of CLIP-score and HPSv2, we observe that image refinement increases both metrics for all approaches. For GeoDrag, CLIP-score increases by 0.36 points while HPSv2 increases by 1.49 points. This increase in image fidelity is verified by visual examination. In alignment with the discussed results in Section 4.3.2, we emphasize again that, although the scores for image fidelity are higher for DragonDiffusion [31], the achieved accuracy is significantly lower and viewpoint consistency and structural integrity of the dragged objects are not ensured.

Table 7 presents an extension to Table 4. We compare more parameter configurations of GeoDrag and measure accuracy, prompt alignment, image fidelity and inference speed. We confirm that setting the thresholds for the point fixation mechanism more tightly consistently improves dragging accuracy at the cost of inference speed. There is no significant influence on prompt alignment and image fidelity.

### C.3. Hyperparameter Configurations

Table 4 is an extension to Section 4.4. We evaluate the quantitative performance of different GeoDrag hyperparameter configurations and compare it to GoodDrag.  $K$  denotes the total number of optimization steps for conducting the geometric modifications via dragging,  $B$  are iterations per

Table 5. Evaluation results of different dragging methods on the geometry guidance dataset. The best result for each metric is marked in bold. For the two methods marked with an asterisk GoodDrag\* and GeoDrag\*, the optimization is performed for 200 iterations.

Method	MD <sub>drag</sub> ↓	MD <sub>out</sub> ↓	CLIP <sub>drag</sub> ↑	CLIP <sub>out</sub> ↑	HPSv2 <sub>drag</sub> ↑	HPSv2 <sub>out</sub> ↑	Time↓
DragDiffusion	43.61	44.38	22.25	23.09	20.52	22.61	50.53s
DragonDiffusion	26.09	27.62	<b>26.73</b>	<b>26.83</b>	<b>26.10</b>	<b>27.15</b>	<b>21.63s</b>
SDE-Drag	22.79	23.49	24.89	25.43	25.05	26.35	25.95s
FreeDrag	21.00	22.15	24.82	25.19	24.34	25.41	68.28s
EasyDrag	19.67	21.13	25.44	25.75	24.57	25.82	54.36s
DragNoise	22.62	23.33	24.95	25.32	24.08	25.21	64.63s
GoodDrag	11.52	11.77	24.82	25.21	24.35	25.93	66.30s
GoodDrag*	10.20	11.24	24.71	24.90	24.07	25.56	122.91s
GeoDrag	<b>9.35</b>	10.80	24.95	25.31	24.22	25.71	60.03s
GeoDrag*	10.72	<b>10.47</b>	25.13	25.22	24.12	25.73	87.09s

Table 6. **Image Identity Scores:** CLIP-similarity-, SSIM []- and LPIPS between the style-transferred image and the final image for GoodDrag and GeoDrag on our benchmark dataset. **Non-Vehicle Benchmark:** Quantitative results on 20 non-vehicle cases.

Image Identity Scores				Non-vehicle Benchmark			
Method	CLIP-similarity ↑	SSIM ↑	LPIPS ↓	MD ↓	CLIP ↑	HPSv2 ↑	Time ↓
GoodDrag	<b>0.9202</b>	<b>0.8067</b>	<b>0.4112</b>	12.55	25.18	23.54	55.68s
<b>GeoDrag</b>	0.9167	0.8055	0.4249	<b>9.02</b>	<b>26.23</b>	<b>24.06</b>	<b>37.16s</b>
Δ	0.38%	0.15%	3.28%	+32.7%	+4.1%	+2.2%	+39.9%

Table 7. Evaluation of GeoDrag configurations on the geometry guidance benchmark. The configurations are in the the format  $K, B, r_1, r_2, l, u$  and are GeoDrag<sup>1</sup>(64, 8, 2, 4, 4, 6), GeoDrag<sup>2</sup>(70, 7, 2, 7, 4, 6), GeoDrag<sup>3</sup>(64, 8, 2, 4, 4, 4), GeoDrag<sup>4</sup>(70, 7, 2, 7, 4, 4), GeoDrag<sup>5</sup>(64, 8, 4, 8, 4, 4), GeoDrag<sup>6</sup>(70, 7, 2, 7, 2, 4), GeoDrag<sup>7</sup>(64, 8, 4, 10, 2, 4), GeoDrag<sup>8</sup>(70, 7, 2, 4, 2, 4), GeoDrag<sup>9</sup>(70, 7, 4, 10, 2, 4). Other parameters are kept consistent. The radii for the gradient mask and copy-&-paste mechanism are set to  $r_{grad} = r_{cp} = r_1$ . The number of copy-&-paste steps after the optimization is 15 and image-to-image refinement is performed with a strength of 0.3 and a guidance scale of 4. LoRA finetuning is not accounted in the runtime.

Method	MD <sub>drag</sub> ↓	MD <sub>out</sub> ↓	CLIP <sub>out</sub> ↑	HPSv2 <sub>drag</sub> ↑	HPSv2 <sub>out</sub> ↑	Time↓
GeoDrag <sup>1</sup>	11.72	14.19	25.21	24.27	25.78	<b>40.77s</b>
GeoDrag <sup>2</sup>	10.89	12.63	25.10	24.43	25.88	41.43s
GeoDrag <sup>3</sup>	10.44	12.01	25.55	<b>24.46</b>	25.90	43.86s
GeoDrag <sup>4</sup>	10.75	11.89	<b>25.65</b>	24.39	25.84	44.40s
GeoDrag <sup>5</sup>	9.71	11.79	25.29	24.23	25.77	49.29s
GeoDrag <sup>6</sup>	10.20	12.66	25.48	24.39	25.85	52.92s
GeoDrag <sup>7</sup>	9.38	11.51	25.58	24.27	25.75	53.84s
GeoDrag <sup>8</sup>	10.36	13.98	25.50	24.32	25.92	54.90s
GeoDrag <sup>9</sup>	<b>9.35</b>	<b>10.80</b>	25.31	24.22	25.71	60.03s
GoodDrag	11.52	11.77	25.21	24.35	<b>25.93</b>	66.30s

timestep,  $r_1$  and  $r_2$  are the radii defining the square patch around the dragged points to obtain the feature maps associated with the points. As expected, longer dragging durations generally lead to better accuracy. We observe that the average duration for the dragging operation is not solely dependent on the number of iteration steps, but also corresponds to the setting of the radii for point tracking and the thresholds for the point fixation mechanism.

We further observe that our approach is consistent when it comes to output image fidelity, measured as CLIP- and HPSv2-scores. The variance between different GeoDrag hyperparameter configurations and GoodDrag is 0.073 points, which is below 3% performance fluctuation.



---

**Algorithm 1** Pseudocode for GeoDrag.

---

**Input:** Original latent image  $z_0$ , handle points  $\{h_i^0\}_{i=1}^n$ , target points  $\{g_i\}_{i=1}^n$ , U-Net  $\epsilon_\theta$ , inversion timestep  $T$ , total number of optimization steps  $K$ , total number of optimizations per denoising step  $B$ , number of motion supervision steps per point tracking  $J$ , max copy paste steps  $N$

**Output:** Dragged image  $\hat{z}_0$

---

```
1:  $LoRA\_finetuning(\epsilon_\theta, z_0)$ 
2:  $z_T \leftarrow DDIM\_inversion(z_0, T)$ 
3:  $z_T^0 \leftarrow z_T$ 
4:  $\mathcal{I}_T^0 \leftarrow [\cdot], M_\nabla \leftarrow \mathbb{K}$ 
5: for  $k$  in  $range(K)$  do ▷  $K$  optimization steps
6:    $t = T - \lfloor \frac{k}{B} \rfloor$  ▷ timestep  $t$  for optimization  $k$ 
7:    $z_{t,0}^k \leftarrow z_t^k$ 
8:   for  $j$  in  $range(J)$  do ▷  $J$  motion supervisions per optimization  $k$ 
9:     ▷ Update ignored points and gradient mask  $M_\nabla$  based on remaining distance
10:     $\mathcal{I}_t^k, M_\nabla \leftarrow update\_ignored(\mathcal{I}_t^k, M_\nabla)$ 
11:
12:     $z_{t,j+1}^k = z_{t,j}^k - \eta \cdot M_\nabla \odot \nabla_{z_{t,j}^k} \mathcal{L}_{ms}(z_{t,j}^k)$  ▷ Motion supervision step Eq. 5 + 6
13:  end for
14:   $z_t^{k+1} \leftarrow z_{t,J}^k$ 
15:   $\mathcal{I}_t^{k+1} \leftarrow \mathcal{I}_t^k$ 
16:   $\{h_i^{k+1}\}_{i=1}^n \leftarrow point\_tracking(\{h_i^k\}_{i=1}^n)$  ▷ Point tracking step
17:
18:  if  $(k+1) \bmod B = 0$  then ▷ After  $B$  optimizations move to next timestep
19:     $z_t^{k+1} \leftarrow copy\_paste(z_t^{k+1}, \mathcal{I}_t^k)$  ▷ Copy & paste only ignored points
20:     $z_{t-1}^{k+1} \leftarrow denoise\_step(z_t^{k+1})$  ▷ Denoise step
21:     $\mathcal{I}_{t-1}^{k+1} \leftarrow \mathcal{I}_t^{k+1}$ 
22:  end if
23: end for
24:
25:  $counter \leftarrow 0$ 
26: for  $t$  in  $[T - \lfloor \frac{K}{B} \rfloor, \dots, 1]$  do
27:   if  $counter = 0$  then
28:     $z_t^K \leftarrow copy\_paste(z_t^K, \{h_i^K\}_{i=1}^n)$  ▷ Copy & paste all points
29:   else if  $counter < N$  then
30:     $z_t^K \leftarrow copy\_paste\_original(z_t^K, z_t, \{h_i^0\}_{i=1}^n)$  ▷ Copy & paste all points using
▷ original handle positions and latents
31:   end if
32:    $z_{t-1}^K \leftarrow denoise\_step(z_t^K)$ 
33:    $counter += 1$ 
34: end for
35:  $\hat{z}_0 \leftarrow z_0^K$ 
36: return  $\hat{z}_0$ 
```

---

## D. Additional Examples

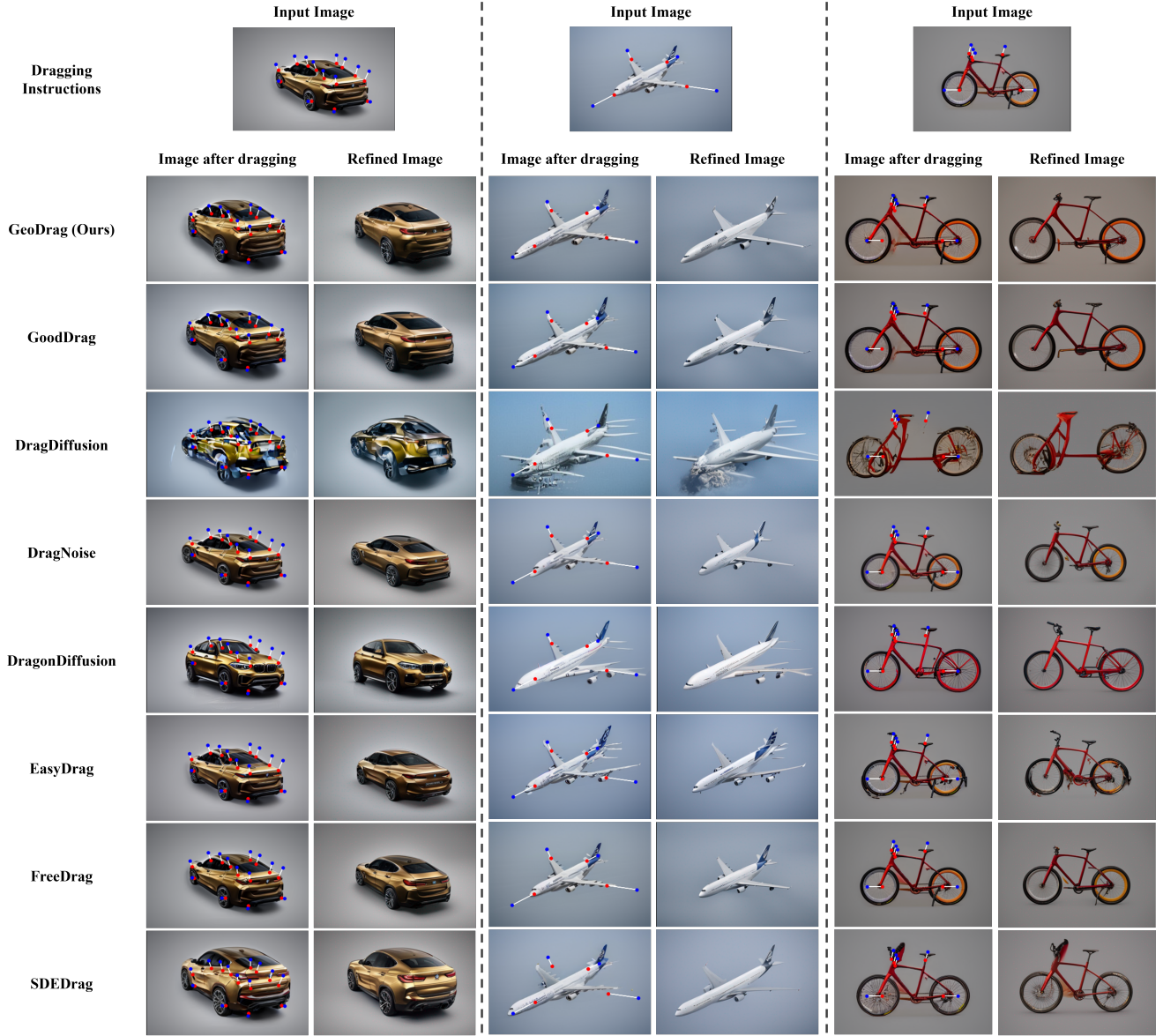


Figure 15. More qualitative examples for comparison of dragging results using GeoDrag and alternative methods. GeoDrag and GoodDrag are the only approaches that achieve high dragging accuracy while maintaining the viewpoint and the integrity of the modified object. DragonDiffusion does not provide viewpoint consistency in the vehicle example and fails to maintain the geometry in the case of the airplane. EasyDrag and SDE-Drag also suffer from structural inaccuracies and FreeDrag achieves low dragging accuracy. In each of the three cases, the left column presents the result after dragging and the right column shows the refined image.


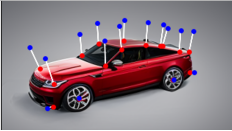
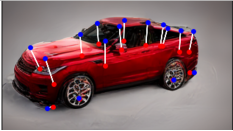



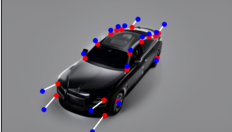







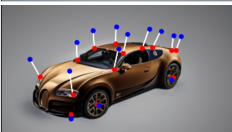
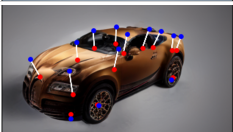








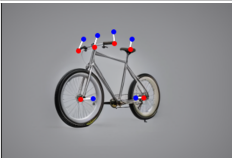
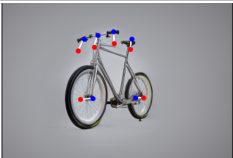

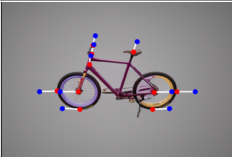
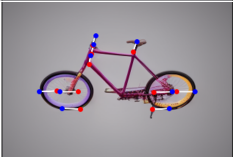

	Rendered 3D Object	Style-transferred Image with Dragging Instructions	Modified Image	Refined Image
<i>A Red Range Rover SUV</i>				
<i>A Pink Mercedes Sports Car Coupe</i>				
<i>A Black Rolls-Royce Ghost Limousine</i>				
<i>A Baby Blue Bentley Continental</i>				
<i>A Bronze Bugatti Veyron as an SUV</i>				
<i>A Blue City Bicycle</i>				
<i>A Red Bicycle</i>				
<i>A Silver Bike with thin tires</i>				
<i>A Pink Mountain Bike</i>				

Figure 16. More examples of geometric modification results using GeoDiffusion on engineering design objects from the geometry guidance dataset. The 3D objects are taken from Objaverse [8]. Visualized examples are from the geometry guidance dataset. The prompt for the image-to-image style transfer is shown in the left column.




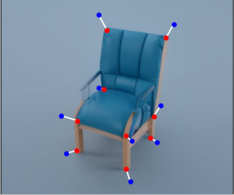
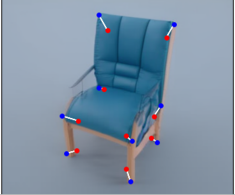


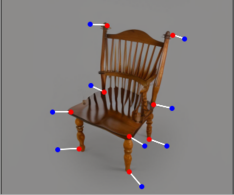
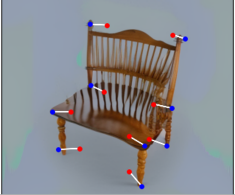


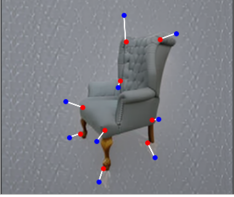
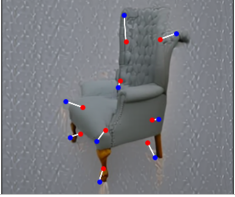














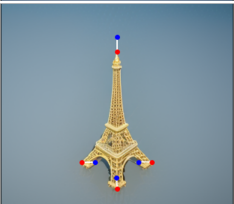



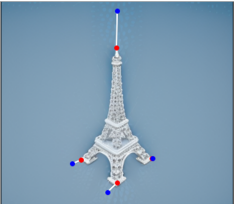
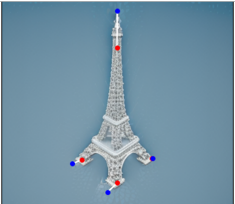

	Rendered 3D Object	Style-transferred Image with Dragging Instructions	Modified Image	Refined Image
<i>A Comfortable Blue Recliner Chair</i>				
<i>A Windsor Chair</i>				
<i>A Grey and Comfortable Chesterfield Chair</i>				
<i>An Eurofighter Typhoon Jet</i>				
<i>A Silver Airbus A320 Airplane</i>				
<i>A Blue Boeing 373 Airplane</i>				
<i>A Golden Eiffel Tower</i>				
<i>The Eiffel Tower made out of shiny Diamonds</i>				

Figure 17. More examples of geometric modification results using GeoDiffusion on engineering design objects from the geometry guidance dataset. The 3D objects are taken from Objaverse [8]. Visualized examples are from the geometry guidance dataset. The prompt for the image-to-image style transfer is shown in the left column.



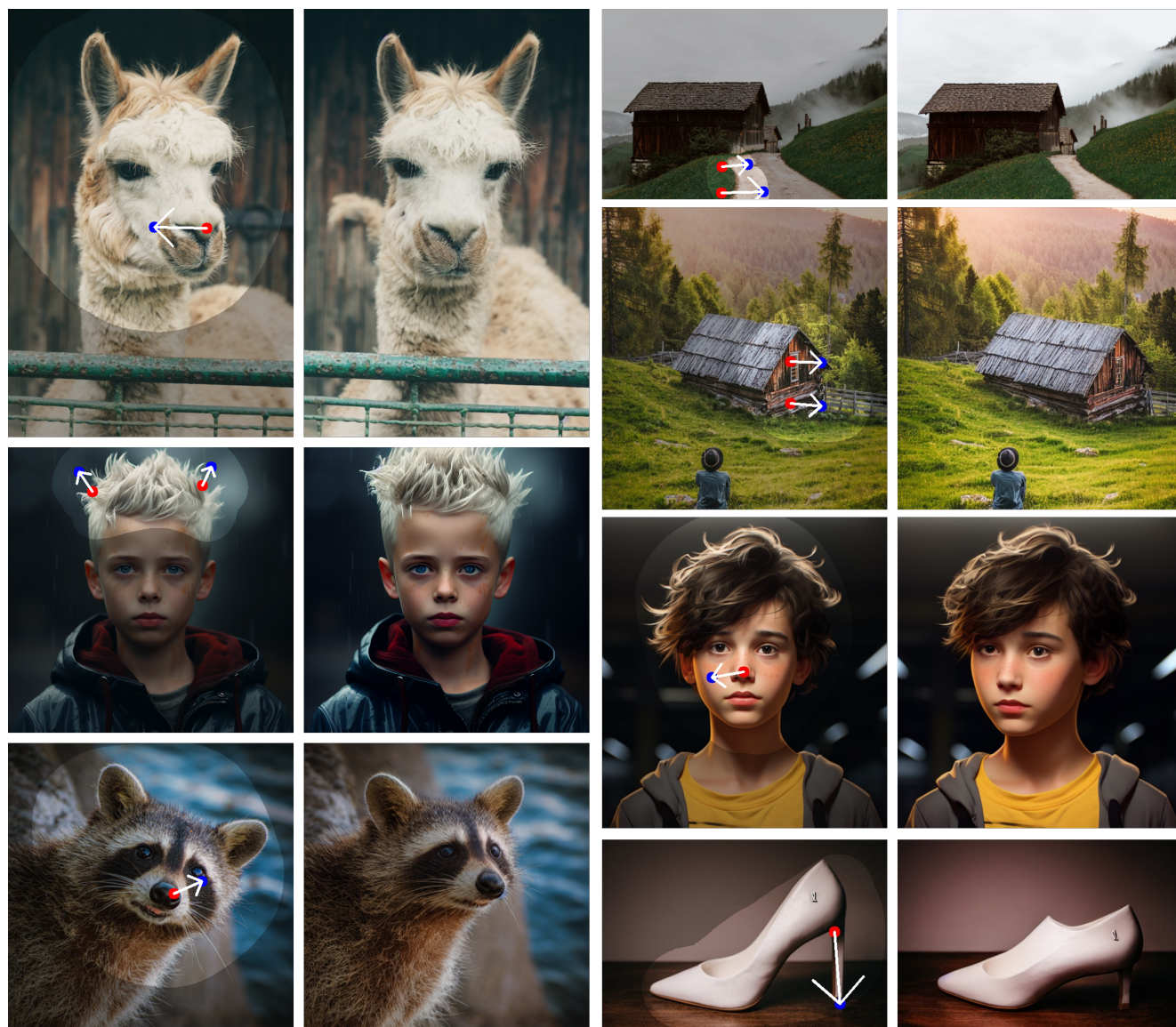


Figure 18. More examples of geometric modification results using GeoDrag on the DragBench dataset.