SplatAD: Real-Time Lidar and Camera Rendering with 3D Gaussian Splatting for Autonomous Driving

Supplementary Material

In the supplementary material, we provide implementation details for our method and baselines, evaluation details, and additional results. In Appendix A, we describe the details of the lidar rendering. In Appendix B, we describe our training setup more closely and provide hyperparameters. In Appendix C, we describe the process of how we generate point clouds with our baselines. In Appendix D, we provide additional details and explanations of our rolling shutter modeling. In Appendix E, we give additional qualitative examples from our method and baselines. Last, in Appendix F we provide evaluation details.

A. Lidar rendering details

A.1. Lidar tiling

The tiling for lidar rasterization is done such that each tile has the same number of lidar points to rasterize. Horizontally, we define each tile to cover N_ϕ lidar points, corresponding to $N_\phi \cdot \text{res}_\phi$ degrees, where res_ϕ is the azimuth resolution. Vertically, each tile covers N_ω elevation channels, where the elevation channels are defined in lidar specifications. We set the elevation tile boundaries to lie between the elevation channels. Thus, each lidar tile can rasterize $N_\phi \cdot N_\omega$ lidar points. We set $N_\phi = 32$ and $N_\omega = 8$ so that $N_\phi \cdot N_\omega = 256$, analogous to the $16 \times 16 = 256$ pixels in the image tiles. Further, a lidar point cloud corresponds to $M_\phi = \lceil 360° / (N_\phi \cdot \text{res}_\phi) \rceil$ tiles horizontally, and $M_\omega = N_{\text{beams}} / N_\omega$ vertically, where N_{beams} is the number of lidar beams/channels.

As described in Sec. 3.3, to find intersections between all Gaussians and tiles, each Gaussian is defined by an AABB $[(\phi_{\text{low}}, \omega_{\text{low}}), (\phi_{\text{high}}, \omega_{\text{high}})]$ centered around its 2D mean in spherical coordinates $\boldsymbol{\mu}^S$. The extent of the AABB corresponds to the projected $3-\sigma$ size of the Gaussian, which further is expanded by acknowledging its velocity. For each Gaussian, we convert these spherical coordinates to tile coordinates, *e.g.*, $(2, 3)$ denotes the tile that is second horizontally and third vertically. This yields an AABB in tile coordinates $[(\Phi_{\text{low}}, \Omega_{\text{low}}), (\Phi_{\text{high}}, \Omega_{\text{high}})]$. By computing the lower limits inclusively, and the upper limits exclusively, the area of the AABB expressed in tile coordinates corresponds to the number of intersections for that Gaussian.

When finding the azimuth tile coordinates, we must account for the wrapping of angles. Although the 2D means of Gaussians are bound to $[0°, 360°)$, their AABB coordinates are not. In addition, the last azimuth tile might extend beyond $360°$ as $\phi_{\text{max}} = M_\phi \cdot N_\phi \cdot \text{res}_\phi \geq 360°$. When wrapping the angles, this can create some overlap between the first and last column of tiles. Since this overlapping area has no corresponding lidar points for the last column of tiles (they are already taken care of by the first column of tiles), it should not be considered for intersections. Thus, the lower limit is found as

$$\Phi_{\text{low}} = \begin{cases} \left\lfloor \frac{\phi_{\text{low}}}{N_\phi \cdot \text{res}_\phi} \right\rfloor & \text{if } \phi_{\text{low}} \geq 0, \\ \left\lfloor \frac{(\phi_{\text{low}} + 360) - \phi_{\text{max}}}{N_\phi \cdot \text{res}_\phi} \right\rfloor & \text{otherwise,} \end{cases} \tag{14}$$

and upper limit as

$$\Phi_{\text{high}} = \begin{cases} \left\lceil \frac{\phi_{\text{high}}}{N_\phi \cdot \text{res}_\phi} \right\rceil & \text{if } \phi_{\text{high}} \leq 360, \\ \left\lceil \frac{\phi_{\text{high}} \pmod{360}}{N_\phi \cdot \text{res}_\phi} \right\rceil + M_\phi & \text{otherwise.} \end{cases} \tag{15}$$

Given Φ_{low} and Φ_{high}, the number of tiles covered horizontally is $\Phi_{\text{high}} - \Phi_{\text{low}}$. As for the image-case, Gaussians are duplicated for each intersection and associated with a unique identifier based on the intersecting tile and the depth of the Gaussian. This unique identifier is used for global sorting. However, before creating this identifier, the azimuth tile coordinates of intersections are wrapped to $[0, M_\phi)$

$$\Phi_{\text{low, wrapped}} = (\Phi_{\text{low}} + M_\phi) \pmod{M_\phi}, \tag{16}$$

$$\Phi_{\text{high, wrapped}} = (\Phi_{\text{high}} + M_\phi) \pmod{M_\phi}. \tag{17}$$

For the elevation tile coordinates, we loop over the sorted elevation boundaries and set Ω_{low} to the last boundary that is smaller than ω_{low} and Ω_{high} to the last boundary that is larger than ω_{high}.

A.2. Lidar points to rasterization points

During training and evaluation, we provide the azimuth and elevation values used for rasterization based on the collected lidar data. We begin by removing any ego-motion compensation by expressing the lidar points' coordinates relative to the sensor pose at the time of capture. For this, we assume a linear motion during the lidar scan's capture. Next, for each lidar point $\mathbf{x} = [x, y, z]^\text{T}$, we convert it to spherical coordinates

$$\mathbf{x}^S = \begin{bmatrix} \phi \\ \omega \\ r \end{bmatrix} = \begin{bmatrix} \arctan2(y, x) \\ \arcsin(z/r) \\ \sqrt{x^2 + y^2 + z^2} \end{bmatrix}. \tag{18}$$

Each point is then mapped to a single tile, using the same approach as for the Gaussians, while assuming that the extent of each lidar point is zero.

In some cases, this process can assign slightly more than 256 lidar points to a tile. This occurs, for instance, if the

linear motion assumption for the ego-motion removal is violated. During training, we shuffle points for each iteration and discard any points beyond 256. For evaluation, we run multiple rasterization passes and concatenate the results.

For tiles with less than 256 points, we still spawn 256 threads for rasterization. Similar to how threads that have reached sufficient accumulation only help loading new batches of Gaussians into shared memory, we use threads without assigned lidar points for the same purpose.

B. Training details

In this section, we present details of our model and how we train it.

Optimization: All parameters of our model are optimized jointly for 30,000 steps, using the Adam [17] optimizer. Learning rates for the different parameters are reported in Tab. 6, and are scheduled using an exponential decay scheduler when applicable. Following [14], we start the optimization with images 4 times smaller than the original resolution and upsample with a factor of 2 after 3,000 and 6,000 steps. Like [35], we optimize the pose parameters of all actor trajectories but do not adjust the camera or lidar poses. The same approach is applied to the baselines.

Initialization: Gaussians are initialized from a mix of lidar points and random points. We use a maximum of 2M lidar points for the static world and add 500 points, drawn randomly within the box, for each actor. In addition to the lidar points, we also initialize 60,000 Gaussians from random points. Half of these points are sampled uniformly within the lidar range. The other half is created from uniformly sampled directions and distances sampled linearly in inverse distance beyond the lidar range, up to a maximum distance of 10 kilometers. Gaussians created from lidar points are initialized with the color retrieved from projecting the point into the closest image, while Gaussians created from random points are initialized with random colors. The scale of a Gaussian is initialized as 20% of the average distance to its three nearest neighbors, and the opacity is initialized to 0.5.

Loss hyperparameters: Our model is optimized by minimizing Eq. (13) with $\lambda_r = 0.8$, $\lambda_{\text{depth}} = 0.1$, $\lambda_{\text{los}} = 0.1$, $\lambda_{\text{intens}} = 1.0$, and $\lambda_{\text{raydrop}} = 0.1$. Except for λ_r, for which we use the same value as in [14], all hyperparameters are set heuristically. The MCMC loss in Eq. (13) is adapted from [16] and consists of an opacity regularization term and a scale regularization term

$$\lambda_{\text{MCMC}}\mathcal{L}_{\text{MCMC}} = \lambda_o \sum_i |o_i| + \lambda_\Sigma \sum_{ij} \left| \sqrt{\text{eig}_j(\Sigma_i)} \right|, \quad (19)$$

where we use $\lambda_{\text{o}} = 0.005$ and $\lambda_\Sigma = 0.001$. The line-of-sight loss for Gaussians intersecting a lidar point p is implemented as

$$\mathcal{L}_{\text{los},p} = \sum_{r_i < r_p - \epsilon} \alpha_i, \quad (20)$$

Table 6. Learning rates (LR) for the different parameter groups. Learning rate scheduling is done using exponential decay.

Parameters	Initial LR	Final LR	Warm-up steps
Means	1.6e-6	1.6e-6	0
Features	2.5e-3	2.5e-3	0
Opacities	5.0e-2	5.0e-2	0
Scales	5.0e-3	5.0e-3	0
Quaternions	1.0e-3	1.0e-3	0
Sensor vel. linear	1.0e-3	1.0e-6	1000
Sensor vel. angular	2.0e-4	1.0e-7	1000
Cam. time to center	2.0e-4	1.0e-7	10000
Actor trajectories	1.0e-3	1.0e-4	2500
Sensor embeddings	1.0e-3	1.0e-3	500

where r_i is the range of Gaussian i, r_p is the range of the lidar point p and $\epsilon = 0.8$.

Densification strategy: We use the MCMC strategy introduced in [16] with the same hyperparameters and the maximum number of Gaussians set to 5M.

Features: In addition to the three color channels, our Gaussians have associated features of dimension 13. We give the sensor-specific embeddings a size of 8. The small CNN used for decoding view-dependent effects consists of two residual blocks with a hidden dimension of 32 and kernel size 3, before a final linear layer. The MLP used for decoding lidar intensity and ray drop probability is also lightweight, consisting of only 2 layers and a hidden dimension of 32.

C. Baseline details

The three considered 3DGS-based baselines, PVG [6], Street Gaussians [43], and OmniRe [7] are all implemented in the open-source repository `drivestudio` [8]. We modify the codebase to enable point cloud rendering by, as described in [7], projecting lidar points into depth images, fetching their depth, and projecting them into 3D.

To generate a point cloud, we place 6 virtual cameras, each with a horizontal FOV of $60°$, in the lidar origin. The cameras are rotated such that they collectively cover $360°$. The focal length is set to the median horizontal focal length of each dataset. This ensures that depth images are rendered at a similar resolution as the models were trained on.

Next, the six depth images are rendered. Here, depth refers to the α-blended z coordinate of Gaussians in camera coordinates. For each lidar point, we project it into the depth image and bilinearly interpolate the nearby pixel values. The z-depths are then converted to ranges t by dividing their values by the cosine of the angle between the direction of the lidar point and the direction of the corresponding cameras z-axis. The points are placed in 3D using the true lidar points origins $\mathbf{o}$ and direction $\mathbf{d}$ as $\mathbf{o} + \mathbf{d}t$.

D. Rolling shutter details

Our rolling shutter compensation is computed from approximated velocities in image space, referred to as pixel velocities. We assume the movement of a sensor C at time t to be modeled by a linear velocity $\mathbf{v}_C$ and an angular velocity $\boldsymbol{\omega}_C$, expressed in the sensor's coordinate system, as exemplified in Fig. 6. For a static Gaussian i, expressed in the coordinate system of this sensor, we can thus describe its velocity relative to the sensor as

$$\mathbf{v}^C_{i,\text{static}} = -(\boldsymbol{\omega}_C \times \boldsymbol{\mu}^C_i + \mathbf{v}_C), \tag{21}$$

where $\boldsymbol{\mu}^C_i$ is the mean of the Gaussian, expressed in the coordinate system of sensor C. Note that the sign of the velocity is negative because we are interested in the velocity of the Gaussian *relative* to the sensor.

If the Gaussian is associated with a dynamic actor, we must also consider the velocity contribution induced by the actor's movement. To this end, we introduce the "actor coordinate system", which is aligned with the actor's current pose but at a fixed location and rotation in the world coordinate system. For an actor modeled by a linear velocity $\mathbf{v}_{\text{act}}$ and an angular velocity $\boldsymbol{\omega}_{\text{act}}$, as exemplified in Fig. 6, we can describe the velocity of Gaussian i as

$$\mathbf{v}^{\text{act}}_{i,\text{dyn}} = \boldsymbol{\omega}_{\text{act}} \times \boldsymbol{\mu}^{\text{act}}_i + \mathbf{v}_{\text{act}}, \tag{22}$$

where $\boldsymbol{\mu}^{\text{act}}_i$ is the mean of the Gaussian, and all vectors are expressed in the actor coordinate system. Further, we can express this velocity in the coordinate system of sensor C using the composition of the actor-to-world and world-to-camera transforms, $T^{\text{act}\to C}$, as

$$\mathbf{v}^C_{i,\text{dyn}} = T^{\text{act}\to C}\mathbf{v}^{\text{act}}_{i,\text{dyn}}. \tag{23}$$

To account for both the velocity from the sensor and the velocity from the dynamic actor, we combine the two velocity sources. We add the velocity contributions together and obtain the complete velocity of Gaussian i relative to sensor C as

$$\begin{aligned}\mathbf{v}^C_i &= \mathbf{v}^C_{i,\text{static}} + \mathbf{v}^C_{i,\text{dyn}} \\ &= -(\boldsymbol{\omega}_C \times \boldsymbol{\mu}^C_i + \mathbf{v}_C) + T^{\text{act}\to C}(\boldsymbol{\omega}_{\text{act}} \times \boldsymbol{\mu}^{\text{act}}_i + \mathbf{v}_{\text{act}}).\end{aligned} \tag{24}$$

Here, the transformed velocity from the dynamic actor retains its sign as it is already relative to the sensor. Finally, using the derivation in [31] for the derivative of pixel coordinates with respect to camera motion, we obtain the pixel velocity for Gaussian i projected into sensor C by multiplying the result with the Jacobian of the projective transform

$$\mathbf{v}^I_i = \mathbf{J}^I(-\boldsymbol{\omega}_C \times \boldsymbol{\mu}^C_i - \mathbf{v}_C + T^{\text{act}\to C}(\boldsymbol{\omega}_{\text{act}} \times \boldsymbol{\mu}^{\text{act}}_i + \mathbf{v}_{\text{act}})). \tag{25}$$

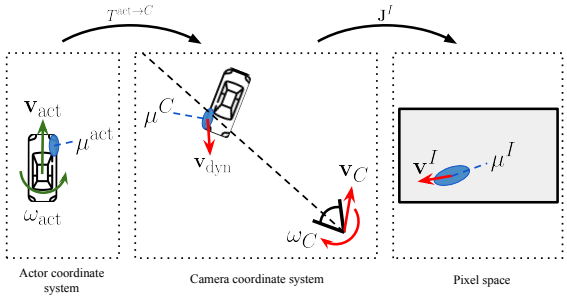


Figure 6. Visualization of an example of the components in the pixel velocity equation.

E. Additional qualitative results

We provide additional qualitative comparisons for the NVS task between SplatAD and our baselines for nuScenes (Fig. 7), PandaSet (Fig. 8), and Argoverse 2 (Fig. 9). We omit OmniRe from the comparisons, as its renderings closely resemble Street Gaussians in most cases. Further, we show a qualitative example of our lidar rendering (Fig. 10), highlighting our method's ability to predict realistic intensity values.

F. Evaluation details

Here, we present the dataset-specific details of our evaluation. The same evaluation protocol is used for all datasets. For the NVS task, we adopt a 50% split, *i.e.*, using every other frame for training and the remaining frames for hold-out validation. In the reconstruction task, we train and evaluate using all frames and lidar scans.

PandaSet: We use the complete sensor rig of six cameras and one lidar when training and evaluating on PandaSet. We crop out the bottom 260 pixels from the back camera to remove views of the ego-vehicle. We choose the same 10 sequences as in [45] and [35]: `001, 011, 016, 028, 053, 063, 084, 106, 123, 158`.

Argoverse2: For Argoverse2, we use the seven ring cameras and both lidars. We crop out the bottom 250 pixels of the front center, rear left, and rear right cameras to remove views of the ego-vehicle. Again, we choose the same 10 sequences as [35]: `05fa5048-f355-3274-b565-c0ddc547b315, 0b86f508-5df9-4a46-bc59-5b9536dbde9f, 185d3943-dd15-397a-8b2e-69cd86628fb7, 25e5c600-36fe-3245-9cc0-40ef91620c22, 27be7d34-ecb4-377b-8477-ccfd7cf4d0bc, 280269f9-6111-311d-b351-ce9f63f88c81, 2f2321d2-7912-3567-a789-25e46a145bda, 3bffdcff-c3a7-38b6-a0f2-64196d130958, 44adf4c4-6064-362f-94d3-323ed42cfda9, 5589de60-1727-3e3f-9423-33437fc5da4b .`

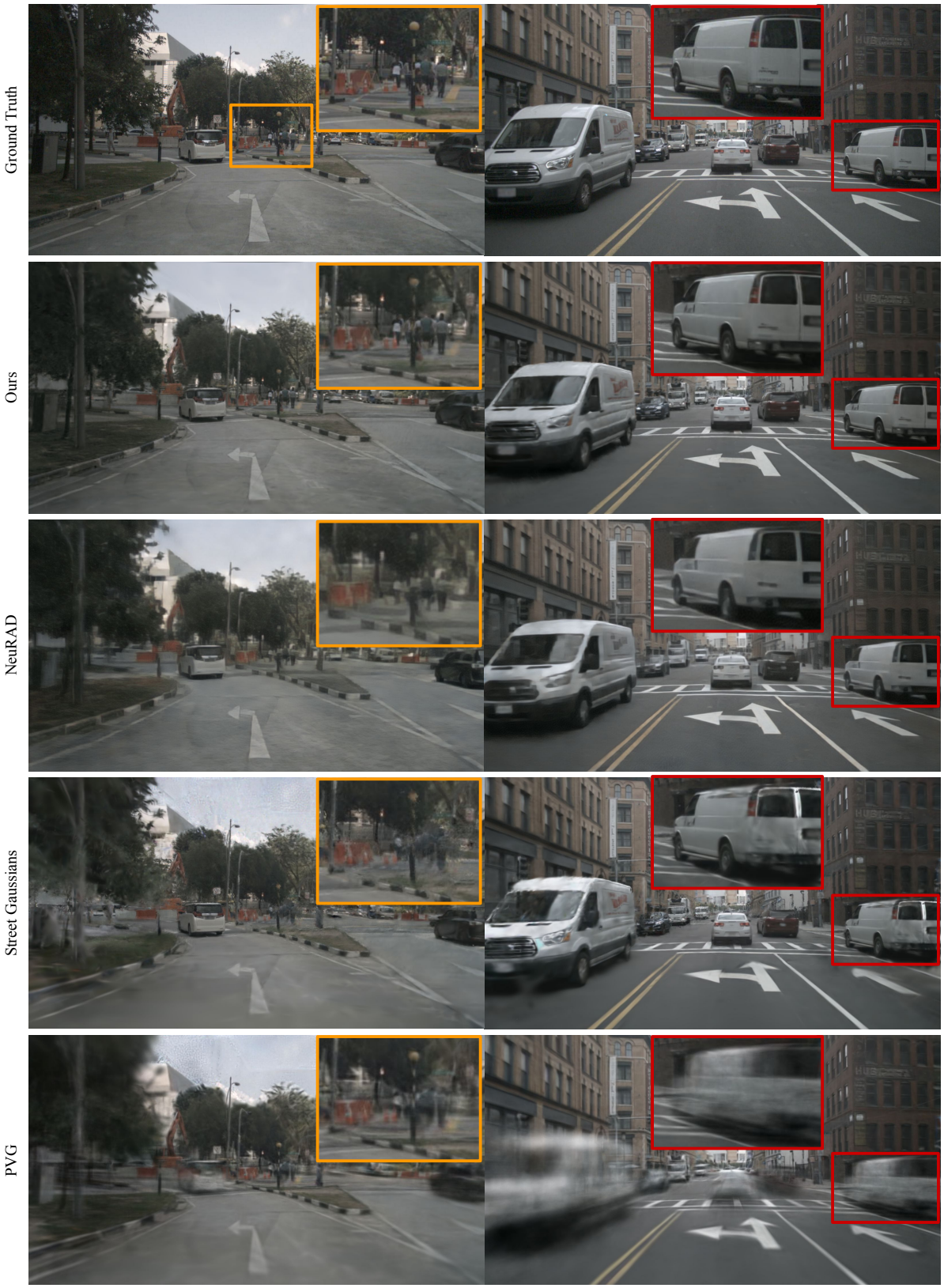


Figure 7. Qualitative NVS examples for nuScenes.

Figure 8. Qualitative NVS examples for PandaSet.

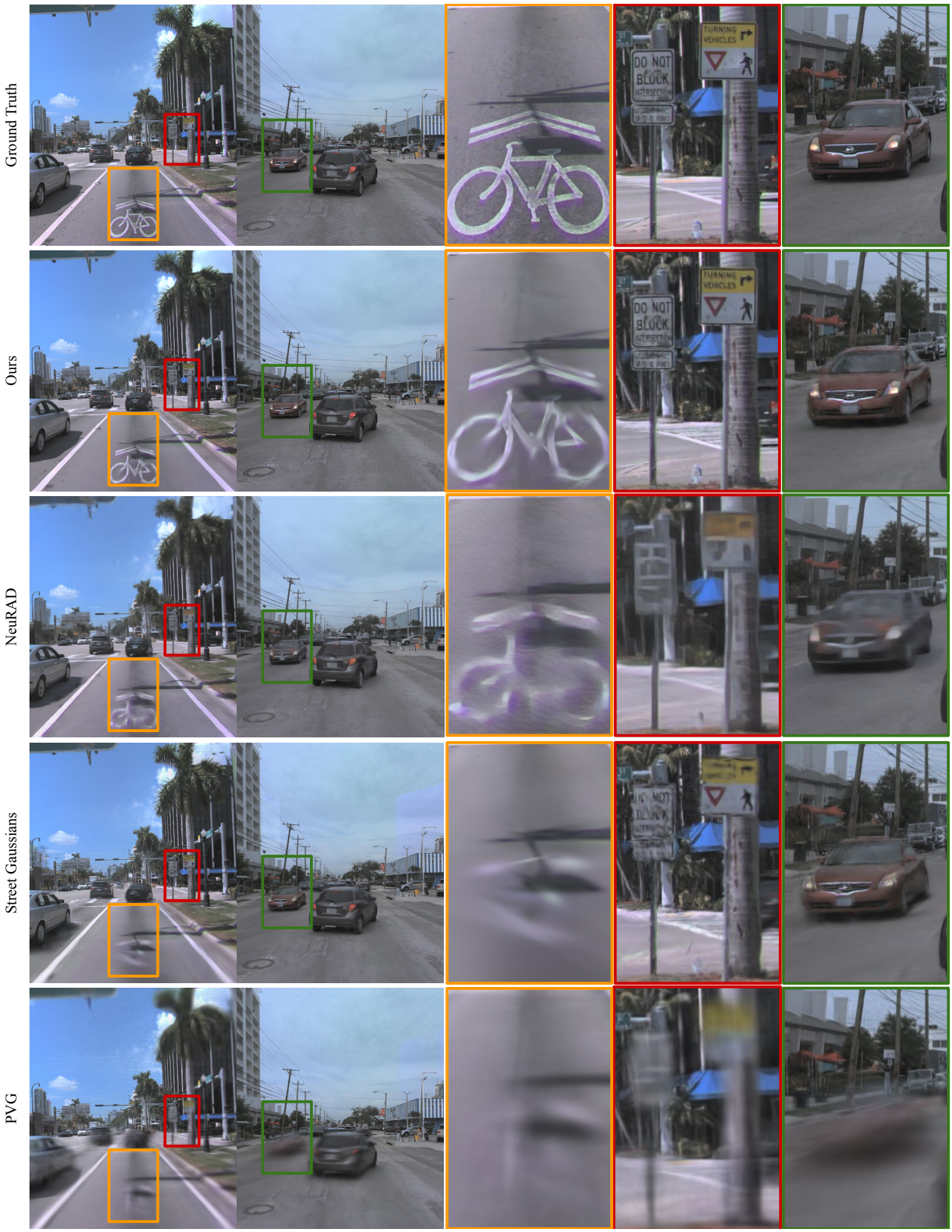
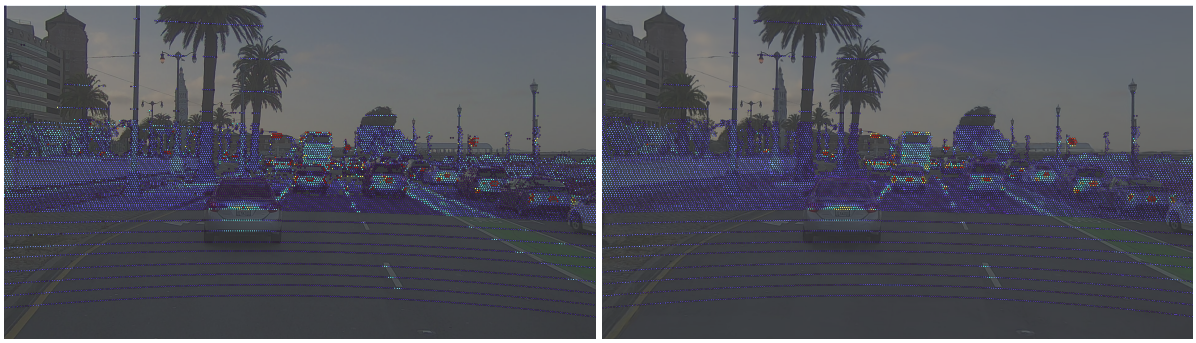


Figure 9. Qualitative NVS examples for Argoverse2.



Ground Truth Ours

Figure 10. Qualitative NVS example for our lidar intensity rendering. We illustrate a rendered point cloud painted with predicted intensity, projected into and overlaid on the corresponding RGB image. The RGB image has been made darker and more transparent to highlight the intensity colors.

nuScenes: We use all six available cameras and the top lidar on the nuScenes dataset. The bottom 80 pixels of the back camera are cropped to remove any views of the ego-vehicle. We select the following 10 sequences: `0039`, `0054`, `0061`, `0066`, `0104`, `0108`, `0122`, `0176`, `0180`, `0193`.