Supplementary Material
Racing in Volume with Flow Ensembles

1 FastFlowGS: Additional Details

1.1 Summary of method

The overview of our method is presented in Alg. 1. FastFlowGS represents a dynamic scene as a composition of multiple Gaussian layers rendered together by alpha blending in depth order: a static background modeled once and updated only in appearance, a dynamic foreground updated geometrically at each frame, and in outdoor environments, a distant sky layer modeled by a skybox. At the core of FastFlowGS is a variational formulation of per-frame Gaussian position initialization. We treat the true 3D position of each Gaussian as an optimization variable and seek the sequence of positions across time that minimizes a quadratic cost combining two terms: 1) a temporal dynamics term that penalizes deviations from the previous frame's position scaled by accumulated uncertainty, and 2) a multi-source measurement term that penalizes disagreement with each independent tracker estimates scaled by their triangulation covariance. The closed-form minimizer of this objective is a precision-weighted fusion that subsumes both the Kalman temporal prior and multi-resolution tracker fusion as special cases, and it produces a calibrated per-Gaussian posterior covariance that downstream processes can exploit directly. The practical motivation is efficiency: at frame $t+1$, the Gaussian representation from frame t already approximates the scene geometry closely. A strong geometric initialization that relocates Gaussians to their frame $t+1$ positions reduces the required finetuning to N_{dyn} iterations rather than the thousands needed when optimizing from scratch, enabling immersive streaming at interactive rates.

1.2 Additional experiments with competing methods

We report full metrics (PSNR, MPSNR) for the low-memory experiments outlined in Sec. 5.2 of the main text and Tab. 4.

The motivation is that this setting now lets these methods distinguish between foreground and background, which helps them attempt corrections over time, instead of just losing the dynamic Gaussians that fall outside the segmentation mask and are never recovered.

FastFlowGS sometimes loses its advantage in MPSNR, which we justify by a limitation of the metric itself. Fig. 1 shows qualitative comparisons of our method and the two highest-ranked competing methods in terms of MPSNR (QUEEN and TrackerSplat). The competing methods' blurriness produces cars with (on-average) closer texture to the groundtruth, but lacking detail or sharp features.

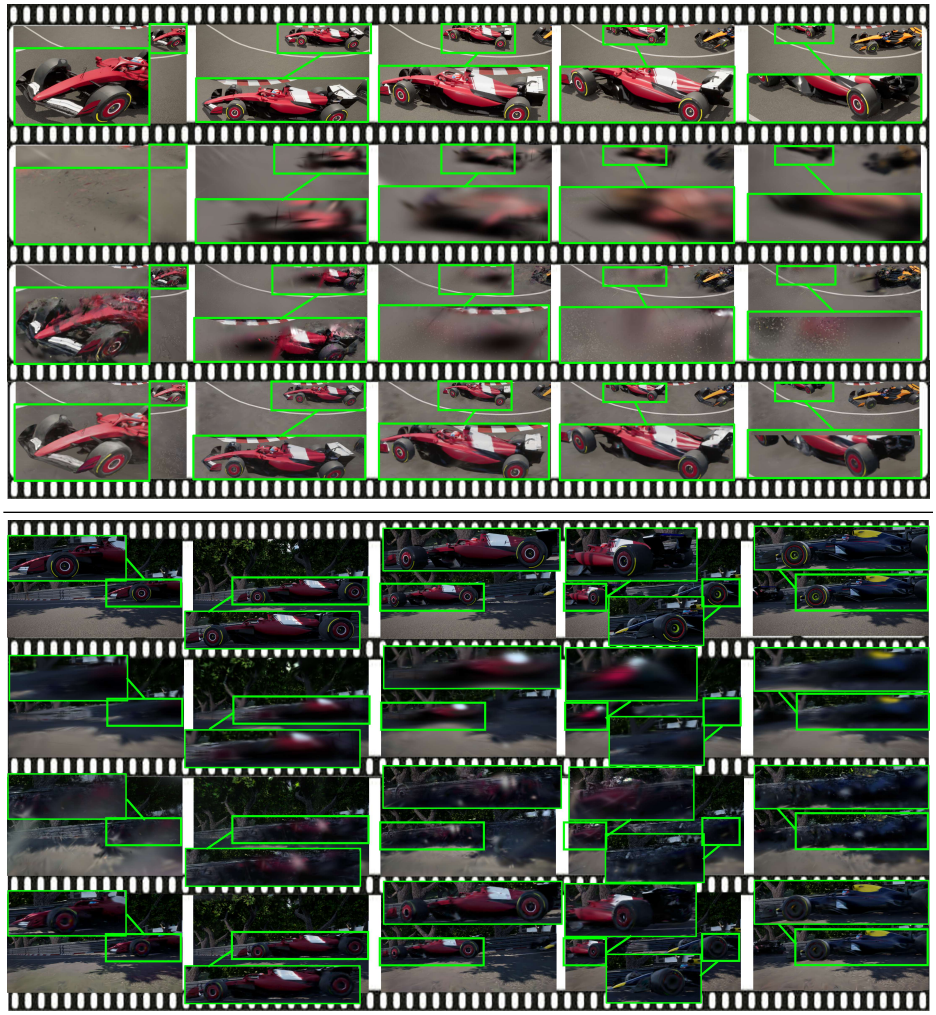

Table 1: Qualitative comparison on consecutive frames on the best performing methods: First row is `Ground truth`, second row is `QUEEN`, third row is `TrackerSplat`, fourth row is `FastFlowGS`

Table 2: Qualitative results with fewer primitives in background: FastFlowGS achieves comparable quality with enhanced convergence speed (Time is in seconds)

	VMAF	PSNR	MPSNR	Time	VE	PE
D-3DGS	37.11	18.58	16.8	731.67	0.05	0.03
HiCoM	1.06	10.18	10.61	25.83	0.04	0.40
QUEEN	9.57	15.13	15.07	40.83	0.39	0.40
ReConGs	1.01	10.47	10.29	39.00	0.03	0.28
Trackersplat	4.78	15.53	18.78	74.83	0.07	0.22
Ours	51.24	23.03	17.59	22.83	2.27	1.02

In contrast, our method produces visually better images, at the cost of some (on-average) noisier texture. MPSNR prefers blur because the squared error penalizes sharp deviations heavily and a single sharp edge misalignment hurts MSE more than global blur. As such, though MPSNR is sometimes higher for other methods, we claim FastFlowGS is able to produce better reconstructions.

Overall, FastFlowGS is typically either best or second-best in all metrics, being able to reconstruct scenes with a larger number of Gaussians, faster, and better quality.

1.3 Timing breakdown of FastFlowGS

Table 3 reports the per-frame timing breakdown for FastFlowGS on Fairmont (variation 3). Static background, dynamic foreground, and skybox optimize on separate GPUs. We report the wall-clock time which equals the slowest lane.

Within the dynamic lane, the initialization pipeline (Voronoi interpolation through fusion) adds less than 1 s and finetuning dominates at ∼90% of the lane cost. On Monaco4D, the static SH update is the overall bottleneck due to the large scene scale. However on CMU Panoptic, static and dynamic finish in roughly equal time. Tracks are precomputed offline. LightGlue (feature tracks) takes ∼40 ms/view, GMFlow (dense tracks) takes ∼65 ms/view and DOT (point tracks+dense) takes ∼0.9 s/frame amortized over 8-frame windows in general. We use LightGlue+GMFlow on Monaco4D and LightGlue+DOT in CMU-Panoptic. All baselines similarly exclude equivalent preprocessing from their reported times (TrackerSplat excludes point tracks, D-3DGS excludes segmentation masks, QUEEN excluded depth estimation).

While pre-processing is dominated by the point tracker, it runs in parallel across frames and views. The practical bottleneck is **sequential** finetuning, which FastFlowGS reduces from 738 s to 9.5 s/frame (**78**×). Notably, TrackerSplat uses the same point tracker yet requires 72 s for finetuning (7.6× more), confirming that the reduction stems from our multi-resolution fusion, not from the external priors themselves.

Table 3: Per-frame timing breakdown. Times in seconds. *Offline* stages are run once per sequence (precomputed). For our method, *online* stages are parallelized into independent lanes executing on separate GPUs; wall-clock time equals the slowest lane.

Component	Parallelism	Ours (Monaco4D)	Dynamic-3DGS	TrackerSplat	Ours (CMU Panoptic)
Offline pre-processing					
Feature tracker	views × trackers	0.031	-	-	0.031
Point tracker	views × trackers	727.38	-	727.38	727.38
Optical flow	views	0.035	-	-	0.035
Segmentation	views	0.015	0.015	0.015	0.015
Pre-proc. total		727.461	0.015	727.395	727.461
Online — Dynamic lane					
Trackers (load)	views × trackers	<0.1	-	-	<0.1
Voronoi interpolation	views × trackers	0.5	-	-	0.5
Cross-level disagreement	views	<0.1	-	-	<0.1
Triangulation	per tracker	0.2	-	-	0.2
Fusion	-	0.15	-	-	0.15
Initialization	-	0.85	-	-	-
Finetuning	-	∼9.5	738	72	∼3.0
Dynamic total		∼10.5	738	72	∼4.0
Online — Static and Skybox lanes (ours only)					
SH finetuning (500 iter.)	-	∼25	-	-	∼1.0
Texture optimization	-	∼1.0	-	-	-
Wall-clock online (max of lanes)		∼25	738	72	∼4.0
Wall-clock total (+ offline)		∼752	738.015	799.595	∼731

Table 4: Qualitative results with fewer primitives in background: Per sequence FastFlowGS achieves comparable quality with enhanced convergence speed (Time is measured in seconds). QUEEN ran out-of-memory for Sequence Main Straight, noted with –

Sequence	Metrics	D-3DGS	HiCoM	QUEEN	ReConGs	Trackersplat	FastFlowGS
Fairmont	VMAF	40.47	0.89	5.39	0.71	2.72	46.89
	PSNR	20.49	10.35	21.73	10.27	15.31	24.5
	MPSNR	17.49	10.75	23.54	10.31	15.86	19.78
	Time	738	27	61	48	72	25
	VE	0.05	0.03	0.09	0.01	0.04	1.88
	PE	0.03	0.38	0.36	0.21	0.21	0.98
Main Straight	VMAF	35.83	1.09	–	1.21	0.78	50.1
	PSNR	17.34	9.82	–	9.62	14.57	21.97
	MPSNR	15.99	9.79	–	9.81	16.99	18.91
	Time	724	24	–	48	95	21
	VE	0.05	0.05	–	0.03	0.01	2.39
	PE	0.02	0.41	–	0.2	0.15	1.05
Rascasse	VMAF	38.73	2.01	12.67	1.66	9.67	53.84
	PSNR	17.43	10.64	12.02	10.76	16.83	23
	MPSNR	15.11	11.14	9.62	11.19	21.34	17.53
	Time	729	25	23	35	70	23
	VE	0.05	0.08	0.58	0.05	0.14	2.34
	PE	0.02	0.43	0.55	0.31	0.24	1
Tunnel	VMAF	32.11	0.25	0.39	0.36	7.15	48.9
	PSNR	17.76	10.39	20.15	10.56	15.57	21.83
	MPSNR	16.81	11.62	23.35	10.11	19.91	16.92
	Time	732	24	69	39	94	19
	VE	0.04	0.01	0.01	0.01	0.08	2.57
	PE	0.02	0.43	0.29	0.27	0.17	1.15
Uphill	VMAF	41.17	0.97	0.219	0.89	1.59	55.84
	PSNR	18.66	10.22	13.73	11.41	15.64	22.28
	MPSNR	17.92	10.19	9.71	10.72	20.92	20.97
	Time	740	29	70	39	60	27
	VE	0.06	0.03	0	0.02	0.03	2.07
	PE	0.03	0.35	0.2	0.29	0.26	0.83
Pool	VMAF	34.36	1.16	38.75	1.24	6.77	51.87
	PSNR	19.82	9.64	23.16	10.19	15.24	24.6
	MPSNR	17.47	10.19	24.22	9.62	17.64	11.43
	Time	727	26	23	25	58	22
	VE	0.05	0.04	1.68	0.05	0.12	2.36
	PE	0.03	0.37	1.01	0.41	0.26	1.12

Algorithm 1 FastFlowGS: Per-Frame Streaming Update ($t > 0$)

Require: Gaussians from frame $t-1$ with positions μ_i and covariances $\mathbf{P}_i$;
multi-view images $\{I_v\}_{v=1}^{V}$; foreground masks $\{M_v\}$; tracker set L
Ensure: Updated Gaussian positions μ_i and covariances $\boldsymbol{\Sigma}_i$
Project each μ_i into views; mark as dynamic if inside previous mask
// Estimate multi-resolution 2D motion fields
for each tracker $l \in L$ **in parallel do**
  Compute 2D flow $\mathbf{F}_l$ between frames $t-1$ and t for all views
  if l is sparse or medium **then**
    Voronoi-densify to full resolution; record distance confidence C_l^{vor}
  end if
end for
// Score cross-level agreement
for each tracker $l \in L$ **do**
  $C_l \leftarrow C_l^{\text{vor}} \times \exp(-\text{pairwise flow disagreement / mean flow magnitude})$
end for
// Associate Gaussians with pixels
Render previous Gaussians; record top-K indices and blending weights α per pixel
for each Gaussian i, view v, tracker l **do**
  Weight $w \leftarrow \alpha \times C_l \times \|\text{flow}\|$;
  displaced mean $\hat{x}^{2d} \leftarrow$ projected mean + α-weighted flow
end for
// Triangulate each tracker independently
for each tracker $l \in L$, each Gaussian i **in parallel do**
  Lift displaced 2D positions to viewing rays; discard views with motion nearly collinear to ray ($< 25^\circ$)
  Solve weighted least-squares for 3D position $\hat{\mu}_{i,l}$ and covariance $\hat{\boldsymbol{\Sigma}}_{i,l}$
end for
// Fuse tracker estimates with temporal prior (Kalman update)
$\mathbf{Q} \leftarrow q^2\mathbf{I}$, where $q =$ median 3D displacement across triangulated Gaussians
for each Gaussian i **do**
  $\mathbf{P}^- \leftarrow \mathbf{P}_i + \mathbf{Q}$ ▷ Predicted covariance
  $\boldsymbol{\Sigma}_i^{-1} \leftarrow (\mathbf{P}^-)^{-1} + \sum_l \hat{\boldsymbol{\Sigma}}_{i,l}^{-1}$ ▷ Fused precision
  $\mu_i \leftarrow \boldsymbol{\Sigma}_i\big[(\mathbf{P}^-)^{-1}\mu_i^{\text{prev}} + \sum_l \hat{\boldsymbol{\Sigma}}_{i,l}^{-1}\,\hat{\mu}_{i,l}\big]$
  if all tracker levels failed **then**
    $\mu_i \leftarrow \mu_i^{\text{prev}}$; $\boldsymbol{\Sigma}_i \leftarrow \mathbf{P}^-$
  end if
end for
// Geometric validation
for each Gaussian i **do**
  if μ_i projects inside mask in fewer than $V-2$ views **then**
    Mark as invalid
  end if
end for
// Finetuning and densification
Split or clone Gaussians with high $\text{tr}(\boldsymbol{\Sigma}_i)$
for Gaussian state **do**
  Dynamic: optimize all parameters (foreground-masked $\mathcal{L}_1$-SSIM) ▷ In parallel
  Static: optimize spherical harmonics only ▷ In parallel
end for
Composite static, dynamic, and skybox via alpha blending

Algorithm 2 Gaussian-Weighted Voronoi Interpolation (adapted from DOT [3])

Require:
Sparse track source positions $\{\mathbf{p}_n^{\text{src}}\}_{n=1}^{N} \subset \mathbb{R}^2$ (pixels)
Target positions $\{\mathbf{p}_n^{\text{tgt}}\}_{n=1}^{N} \subset \mathbb{R}^2$ (pixels)
Target visibility $\{\alpha_n^{\text{tgt}}\}_{n=1}^{N} \subset [0,1]$
Image size $H \times W$
Number of neighbors K
Gaussian kernel scale σ
Ensure:
Dense flow $\mathbf{F} \in \mathbb{R}^{H\times W\times 2}$
Interpolated target visibility $\hat{\alpha} \in \mathbb{R}^{H\times W}$
Distance confidence $\mathbf{C}^{\text{vor}} \in \mathbb{R}^{H\times W}$
for each pixel $\mathbf{p} = (u, v)$ in $[0, W-1] \times [0, H-1]$ **do**
  Find K nearest source tracks by Euclidean distance in normalized coordinates: $\bar{\mathbf{p}} = \left(\frac{u}{W-1}, \frac{v}{H-1}\right)$, $\bar{\mathbf{p}}_n^{\text{src}} = \left(\frac{p_{n,x}^{\text{src}}}{W-1}, \frac{p_{n,y}^{\text{src}}}{H-1}\right)$
  Let $\{d_k^2\}_{k=1}^{K}$ be the squared distances to the K neighbors (sorted ascending)
  Gaussian weights: $w_k = \exp\!\left(-\,d_k^2 \,/\, 2\sigma^2\right)$
  Distance confidence: $\mathbf{C}^{\text{vor}}(\mathbf{p}) = w_1$ ▷ Nearest-neighbor weight
  Normalize: $\hat{w}_k = w_k \,/\, \sum_{j=1}^{K} w_j$
  Dense flow: $\mathbf{F}(\mathbf{p}) = \sum_{k=1}^{K} \hat{w}_k \left(\mathbf{p}_k^{\text{tgt}} - \mathbf{p}_k^{\text{src}}\right)$
  Target visibility: $\hat{\alpha}(\mathbf{p}) = \sum_{k=1}^{K} \hat{w}_k \, \alpha_k^{\text{tgt}}$
end for

1.4 Confidence-Weighted Voronoi Interpolation

Sparse and medium-density trackers produce correspondences at a subset of pixels. We densify each tracker's output to full resolution via Gaussian-weighted K-nearest-neighbor interpolation, adapted from DOT [3]. Algorithm 2 details the procedure, which jointly produces a dense flow field, interpolated target visibility, and a per-pixel distance confidence $\mathbf{C}^{\text{vor}}$.

This confidence enters the cross-level agreement and propagates into the triangulation weights, ensuring that pixels far from any observed track are down-weighted throughout the pipeline. We use $K = 8$ and $\sigma = 0.05$ in all experiments.

2 Monaco4D: Additional Details

This section provides the rendering and scene configuration details needed to reproduce Monaco4D. All sequences are built in Unreal Engine 5 [2] using path-traced rendering at real-world scale (1 Unreal Unit = 1 cm).

2.1 Camera Configuration

Intrinsics. All cameras use Unreal Engine's `CineCameraActor` with a 16:9 digital filmback, a fixed 12 mm prime lens at $f/2.8$, and no autofocus, yielding a horizontal field of view of approximately 89.4°. Intrinsic parameters are identical across every camera in the dataset. Hence, differences between viewpoints arise solely from position and orientation.

Initialization hemispheres. To provide a high-quality first-frame reconstruction for each sequence, we additionally place a dense hemispherical camera grid around each vehicle at $t = 0$ (Fig. 1). These views are used only for initialization and are excluded from evaluation.

Fig. 1: Hemispherical initialization rig. At $t = 0$, a dense hemisphere of cameras surrounds each vehicle to bootstrap a high-quality first-frame Gaussian reconstruction. This provides the initial 3D representation that streaming methods then propagate forward in time. Hemisphere views are used only for initialization and are excluded from all evaluation metrics.

Trackside cameras. Cameras are distributed along the Monaco circuit along the guardrails, as shown in the track overview of Fig. 4. The number varies by track section: Main Straight (478), Uphill (342), Fairmont (232), Tunnel (198), Pool (135), and Rascasse (92). Frame counts vary per sequence with the duration of the captured motion.

Onboard cameras. Each vehicle carries seven onboard cameras that replicate standard Formula 1 broadcast mounting positions per FIA regulations [1]: forward, rear, lateral, and driver perspectives (Fig. 2). Onboard cameras share the same intrinsic configuration as trackside cameras.

Drone cameras. Three aerial trajectories per sequence simulate broadcast drone operation: subject tracking, rapid panning, and abrupt zoom changes. Their flight paths are overlaid on the track layout in Fig. 4 in color.

2.2 Environment and Track Geometry

The surrounding urban structures (buildings, grandstands, background elements) originate from a commercially available Monaco circuit asset (Fab). We reconstructed the track surface and several structural components from scratch to ensure accurate geometry, consistent topology, and full control over materials. The drivable track geometry and camera placements are metric (1 UE unit = 1 cm), so vehicle speeds, inter-camera baselines, and camera-to-track distances correspond to real-world values. Surrounding city structures are optimized for visual realism and occlusion fidelity rather than survey-grade geometric accuracy. Hence, while they preserve the characteristic layout and proportions of Monaco, they should not be treated as a precise urban reconstruction.

Table 5: Distribution of cameras: The number of cameras and frames per sequence, per variation. We also present the average number of views that are dynamic (have atleast 1 car) per frame

Sequence	Variation	Frames	Trackside Cameras	Frames with car per camera
Main Straight	V1	214	478	17
	V2	210	478	11
	V3	224	478	25
Rascasse	V1	147	92	28
	V2	147	92	52
Uphill	V1	197	342	20
	V2	233	342	49
Pool	V1	146	135	16
Tunnel	V1	320	198	19
	V2	335	198	32
Fairmont	V1	241	232	23
	V2	270	232	41
	V3	262	232	71

2.3 Weather and Illumination Layers

To generate environmental variability in the rendered dataset, we use the Ultra Dynamic Sky plugin in Unreal Engine. This plugin provides a procedural sky, lighting, cloud, fog, and weather system that can be controlled directly inside the scene. Instead of creating a single fixed lighting setup, we organize the environment into separate illumination and weather layers, each corresponding to a specific capture condition.

In our setup, we define individual layers for daytime illumination, night, foggy conditions, and rainy weather. Each layer stores the required scene configuration for that condition, including sky appearance, sun or moon contribution, atmospheric fog, cloud coverage, weather intensity, and the corresponding visual effects. This allows us to switch between different environmental states while keeping the geometry, camera setup, and scene layout unchanged.

We use the `daytime` layer to simulate clear or naturally lit outdoor conditions, with stronger directional illumination from the sun and brighter sky contribution. The `night` layer reduces the global illumination and changes the sky and atmospheric response to produce low-light captures. The `fog` layer increases atmospheric scattering and reduces scene visibility, creating images with lower contrast and stronger depth-dependent haze. The `rain` layer activates precipitation effects and modifies the overall appearance of the scene to represent wet weather conditions.

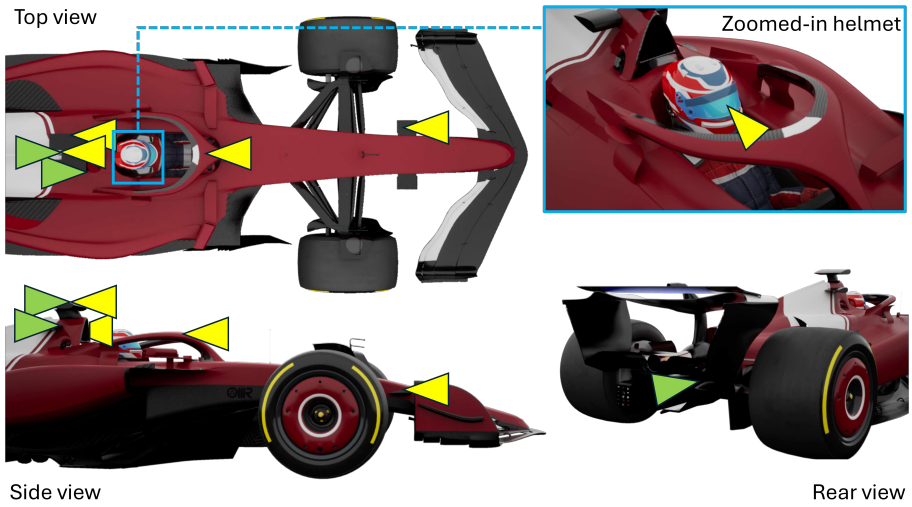

Fig. 2: Onboard camera placement. Four viewpoints of a single vehicle illustrating the seven per-car camera positions that follow FIA broadcast regulations [1]. triangles depict forward facing cameras while triangles depict rear facing cameras.

2.4 Vehicle Models and Animation

The Formula 1 car and driver models are based on commercially available assets, integrated into Unreal Engine with custom rigging (Fig. 3). The vehicle control rig incorporates physics-based suspension that produces dynamic body roll and pitch in response to track forces, realistic DRS flap actuation triggered at the correct circuit positions, and a steering linkage that constrains the driver's hands to the wheel so that driver pose follows steering input automatically. Head bobbing is driven by the suspension state, coupling driver motion to vehicle dynamics rather than relying on canned animation.

Speed calibration. Vehicles follow trajectories along the track at speeds calibrated against real Formula 1 segment times: simulated lap-segment durations fall within 90% of their real-world counterparts. The effective speed was reduced by roughly 10% so that each camera captures the vehicle across a larger number of frames per pass, increasing temporal coverage per observation without altering the spatial layout. Even at this reduced speed, peak inter-frame displacements reach 200–400 pixels for laterally oriented trackside cameras, placing the dataset well beyond the operating range of standard optical flow networks.

2.5 Rendering Configuration

Rendering uses the Unreal Engine Path Tracer with 64 spatial samples per pixel and 1 temporal sample. Standard anti-aliasing is disabled and temporal AA

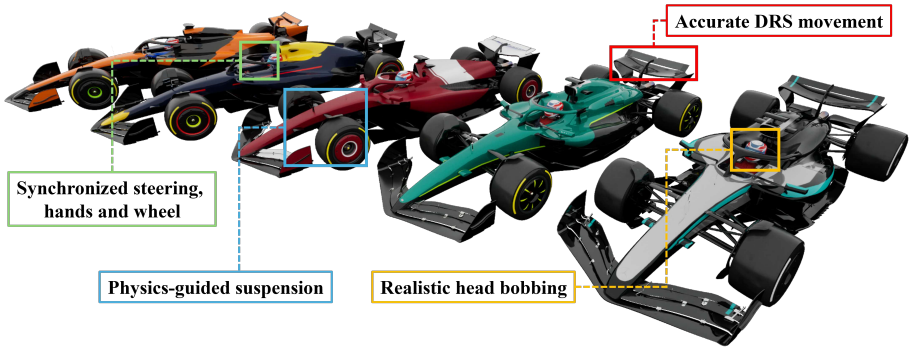

Fig. 3: Vehicle assets and dynamic rigging. Each vehicle integrates physics-based suspension, DRS flap actuation triggered at circuit-accurate positions, a steering linkage that constrains the driver's hands to the wheel, and head bobbing coupled to the suspension state. These mechanisms enforce that vehicle appearance changes frame-to-frame in response to track forces rather than following canned animation.

samples are set to 8. The path tracing denoiser is enabled with a 2-frame noise reduction window. Frames are exported as 8-bit PNG sequences at an effective render resolution of 150% screen percentage. Geometry culling for instanced static meshes is disabled such that all geometry remains visible to the path tracer.

A 128-frame warm-up stage (both render and engine) precedes each sequence, to allow lighting and temporal state to converge before capture begins.

2.6 Lighting and Atmosphere

Illumination combines four physically motivated components:

Directional light. A directional light represents the sun, with intensity 10.0 and indirect lighting intensity 6.0. Volumetric scattering is enabled for correct light-atmosphere interaction.

Sky light. A `SkyLight` at intensity 1.0 provides diffuse environmental illumination captured from the atmospheric model. The lower hemisphere uses a solid color to prevent unrealistic below-ground contributions.

Atmospheric scattering. The `SkyAtmosphere` component models Rayleigh scattering (scale 0.0331) and Mie scattering (anisotropy 0.8) with additional absorption for subtle atmospheric color filtering.

Volumetric clouds. A cloud layer spans $5 - 10$ km altitude, with a tracing start distance of 350 km and maximum tracing distance of 50 km, introducing natural skylight variation and soft shadowing.

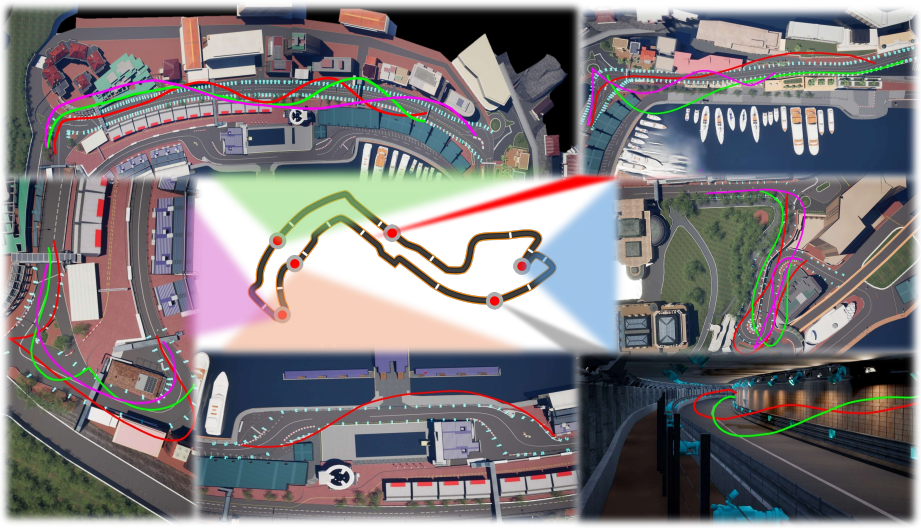

Fig. 4: Track layout, camera distribution, and drone trajectories. Center: the full Monaco circuit with the seven sequence segments highlighted. Insets: per-section camera placements viewed from above, showing the density and angular spread of trackside coverage. Colored paths overlaid on each segment trace drone trajectories.

2.7 Post-Processing and Motion Blur

A global Post Process Volume enables the ray tracing features required by the path tracer. No additional tone mapping, stylization, or image-space effects are applied. Motion blur is explicitly disabled (motion blur amount set to 0) so that each frame represents a sharp instantaneous capture.

2.8 Frame Rate

All sequences are rendered at a base rate of 30 FPS. Selected sequences are additionally rendered at 240 FPS to provide denser temporal sampling of vehicle motion, useful for tracking, reconstruction, and temporal interpolation tasks.

References

1. 2026 formula 1 technical regulations, `https://www.fia.com/sites/default/files/fia_2026_formula_1_technical_regulations_issue_8_-_2024-06-24.pdf`
2. Epic Games: Unreal engine, `https://www.unrealengine.com`
3. Le Moing, G., Ponce, J., Schmid, C.: Dense optical tracking: Connecting the dots. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 19187–19197 (2024)