3D Gaussian Splatting for Real-Time Radiance Field Rendering

more → Mip-NeRF360 : another splatting 3D real time renderer.
than ←
better than → Instant NGP →

Radiance Field methods : view synthesis of scenes captured with multiple photos or video is. Issues: They require NN that are costly to train & render. This is prominent with Nerf. It takes a "long" time to train when changing view. Currently no method can achieve real-time display rates (1080p, 30 fps).

1. Introduction

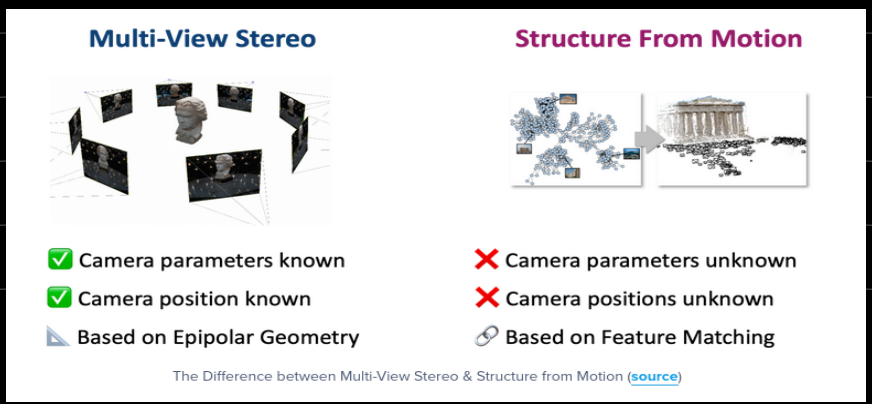
① The scene is represented w 3D Gaussians to preserve properties. Avoids unnecessary computation in empty spaces.
② Interleaved optimization/density control of the 3D Gaussians, optimizing anisotropic covariance to achieve accurate repr. of the scene.
③ fast visibility-aware rendering algorithm that supports anisotropic splatting, accelerates training, allows for real-time rendering

Meshes & points are used for 3D scene building. Issues: Costly for the calculations. Nerf optimizes MLPs using volumetric ray-marching. Other ways: voxel, hash grids or points. Costly and results in noise.

3D Gaussians: Rich based splatting ensures real time rendering

Nerf requires up to 48 h of constant training to achieve. The fast — but-lower-quality — radiance field methods can achieve interactive rendering times (≈ 10–15 fps), but fall short of real-time rendering at high res.

Structure-from-Motion vs Multi-View Stereo



(9/10/'25)

https://pyimagesearch.com/2024/10/14/photogrammetry-explained-from-multi-view-stereo-to-structure-from-motion/

Multi-View Stereo uses Epipolar Geometry to calculate the different positions. Epipolar geometry is the geometry of stereo vision. When 2 cameras view a distinct 3D scene from 2 points. From 2D to 3D is perspective projection. Looking at a cube, you'll see a square. https://en.wikipedia.org/wiki/Epipolar_geometry (9/10'25)

Point clouds are the xyz coordinates of a data point. A lot of these points together form a cloud representing all the data. These data points come from for example LiDARs, heat, reflectivity, ring/channel, color, range, class, velocity---
https://www.thinkautonomous.ai/blog/point-clouds/
(9/10'25)

The 3D Gaussian Splatting model only requires SfM points as input. 3D Gaussians are an excellent choice because they be rasterized differently by projecting them to 2D, and applying α-blending using an equivalent formation as NeRF. 2nd component is optimization of some parameters and removing some 3D gaussians during optimization. 3rd component is using fast GPU sorting alg. and is inspired by tile-based rasterization.

2.1 Traditional Scene Reconstruction & Rendering

Structure-from-Motion (SfM) enabled an entire domain where collection of photos could be used to synthesize novel views. SfM estimates sparse point clouds during camera calibration. Multi-view Stereo (MVS) also exists.
All these methods re-project and blend the input images into the novel view camera, and use geometry to guide the projection. They suffer from not being able to reconstruct non-Lambertian surfaces or "over-reconstruction" when MVS generates inexistent geometry. Recent Neural Rendering alg. reduce the cost of storing all images on GPU, outperforming these methods.

2.2 Neural Rendering & Radiance Fields

CNN for final rendering frequently results in temporal flickering. CNNs were also used for blending weights or texture-space solutions.

Deep learning was used with volumetric ray-marching were subsequently proposed building continuous differentiable density fields to represent geometry. Rendering using volumetric ray marching has significant cost due to the large number of samples required to query the volume. Neural Radiance Fields (NeRFs) introduced important sampling and positional encoding, but used a large MLP negatively affecting speed. Mip-NeRF360 is the state-of-the-art model that improved quality, but training & rendering times are extremely high. Instant NGP which uses a hash grid and an occupancy grid to accelerate computation and a smaller MLP to represent density and appearance. Plenoxels uses sparse voxel grid to interpolate a continuous density field and are able to forgo NN. Both rely on Spherical Harmonics. the former to represent directional effects directly, the latter to encode its inputs to the color network. (+) good results. (-) issues with representing empty space and image quality is limited by the choice of the structured grid used for acceleration and rendering speed is hindered by the need to query many samples for a given ray-marching step. 3D Gaussians can achieve better results without neural components.

2.3 Point-Based Rendering & Radiance Fields

Point Sample rendering rasterizes an unstructured set of point types of graphics APIs or parallel software rasterizations on the GPU. Point Sampling suffers from holes, causes aliasing, and is strictly discontinuous. Solution: "Splatting" with a larger extent than the pixel. e.g. circular, elliptic disc, ellipsoids, or surfels. Other differentiable point-based techniques exist by augmenting points with neural features and rendering using a CNN. They still need MVS. Point-based α-blending and NeRF-style volumetric rendering share essentially the same image formation model.

Different from NeRF? The image formation model is the same. However the rendering alg. is very different. NeRFs are continuous representation implicitly representing empty/occupied space; expensive random sampling is req. to find the samples with consequent noise and computational expenses. In contrast, points are unstructured, discrete representation that is flexible enough to allow creation, destruction, and displacement of geometry similar to NeRF. This is achieved by minimizing opacity and positions, while avoiding the shortcomings of a full volumetric representation.

Intermezzo: What is α-blending?

Not needed to know

Formula: $C(r) = \int_0^\infty T(t)\,\sigma(t)\,c(t)\,dt$, where $\sigma(t)$ = density (opacity) at a point; $c(t)$ = RGB color at a point; and accumulated transmittance (how much light passes through) $T(t) = \exp\left(-\int_0^t \sigma(s)\,ds\right)$

Alpha bleeding happens when the model predicts nonzero density (σ) in regions where it shouldn't be causing faint "foggy" colors outside object boundries, halos or ghosting artifacts near edges, background colors being contaminated by nearby surfaces.

Why this happens? Density regularization is weak (not penalizing small, spurious densities outside surfaces). Positional encoding aliasing or sampling noise (causes fuzzy boundries). Volume rendering's cumulative nature.

How is this mitigated? Density regularization/sparsity loss (e.g., penalizing nonzero densities). Empty space skipping or occupancy grids (so rays don't accumulate unnecessary density). Background regularization (forcing known background colors to be clean), better sampling strategies near surfaces.

(13/10/'25)

https://openaccess.thecvf.com/content/CVPR2022/papers/Niemeyer_RegNeRF_Regularizing_Neural_Radiance_Fields_for_View_Synthesis_From_Sparse_CVPR_2022_paper.pdf

RegNeRF is widely cited as an early robust attempt to suppress artifacts that are essentially equivalent to what people mean by "alpha bleeding."

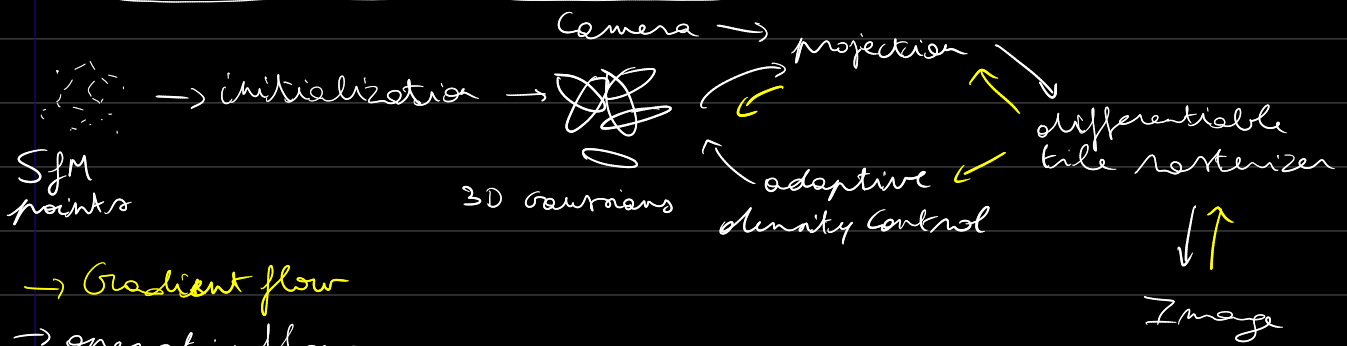
3. Overview

The input is a set of images of a static scene, together with corresponding cameras calibrated by SfM, which produces sparse point clouds as a side-effect. Using these points => create 3D Gaussians, defined by a position (mean), covariance matrix and opacity α, allow for flexible optimization regime. The color is represented using spherical harmonics. Based on these parameters 3D Gaussians can adaptively control the gaussian's density. The key to efficiency is tile-based rasterization, that allows for α-blending of anisotropic splats.

4. Differentiable 3D Gaussian Splatting

3D Gaussians are chosen since they are differentiable and can easily be projected to 2D splats allowing fast α-blending for rendering. The set of 3D Gaussians doesn't require normals. Optimization: gradient descent and update steps.

5. Optimization with adaptive density Control of 3D Gaussians



→ Gradient flow
→ operation flow

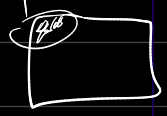
5.1 Optimization

Successive iterations of rendering + comparing to the real image in the training view. The optimization needs to create, move and destroy geometry. They use Stochastic Gradient Descent. Creating the rasterizations in the GPU is the only remaining bottleneck. Sigmoid activation function constrains data to $[0-1)$ and an exponential activation function for the covariance. Generate the gaussians with the location of the mean of the 3 closest data points.

5.2 Adaptive Control of Gaussians

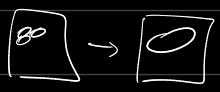
① Initial set of sparse points from SfM, then apply the method
② After warm-up, we densify every 100 iterations and remove Gaussians with low α value (more transparent)

only gaussian here

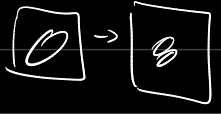


③ Handle "undereconstruction" and "overreconstruction"
↳ Optimization tries to handle this.

④ Small gaussians create a single big one →



Big gaussians create 2 smaller ones (by a factor of $\phi = 1.6$)



⑤ Cloning gaussians until they fit

6. Fast Differentiable Rasterizer for Gaussians

Tile-based rasterizer for Gaussian splats to pre-sort primitives for an entire image at a time, avoiding sorting pixel by pixel. Allows for fast back propagation, requires only constant overhead for each pixel.

① Split the screen 16×16 tiles
② Cull 3D Gaussians against the view frustum & each tile.
③ Only keep Gaussians with a 99% confidence interval.
④ Reject Gaussians at extreme positions i.e. outside the view.
⑤ Initiate each Gaussian according the the amount of tiles they overlap, and assign a key for each tile.
⑥ Sort these key fast using the GPU with Radix sort
(The α-blending is an approximation, but gets filled in during training and isn't noticable.
⑦ After sorting, you get a list of each tile by identifying the first & last depth sorted entry that splats to a given tile.
⑧ Each block first loads packets of Gaussians into shared memory and then for each given pixel, accumulates the color and α values by traversing front to back.
⑨ At regular intervals, tiles are queried, processing stops when all pixels have saturated (α goes until 1).
⑩ The only stopping criteria is α.

7. Implementation, Results & Evaluation (P8)

Using Python, CUDA, PyTorch... Warm up the computation using lower resolutions (4x smaller) and then upsample using twice after 250 and 500 iterations.

Error metrics of the model: PSNR, L-PIPS, and SSIM.
The rest of the paper talks about the results & discussions.
The results are good & the difference with Mip-NeRF360 is visible.