3D Gaussian Splatting for real time Radiance fields Rendering

→ traditional 3D scene (meshes, point clouds).
→ Neef → extensive training & rendering

3D Gaussian Repres

↳ preserves volumetric radiance fields prop.
↳ Enable efficient training.
↳ real time - rendering at → 30 FPS.

Key Innovation

↳ 3D gaussian repres. derived from SFM points.
↳ Adaptive density control & anisotropic Covariance optimization
↳ Fast tile based splatting renderer for R.T Rendering.

Point Based Rendering (PBR) → 3D objects are rendered as a points. rather that triangles or Volumetric grid (voxel)

directly rendering point samples without requiring explicit inform. like edges or faces.

(depth maps, Lider).

- Can result in holes or aliasing
- Harder to process
- need Specilized technique likes Splatting for Smooth results

- point cloud → surfel
  - 3D pos (x,y,z)
  - Color (R,G,B)
  - Normal vector (optional)
  - opacity (α)
  - Spherical Harmonics (SH) Coefficients (for view dependent effects)
- These points can be obtained from
  - SFM, MVS, LiDAR, NeRF,

Point rendering techinique.

↳ point sprites → each 3D point is treated as 2D texture (sprite) (simple & fast) but suffer from holes, at sparse regions)

↳ splat based rendering
uses elliptical splats (disc) instead of single points. Blends overlapping splats to fill gaps and create a smooth appearence.

↳ Deferred shading.
→ First renders point attributes into a buffer.
→ Then applies shading & lighting in a second phase.

↳ Adaptive Density Control

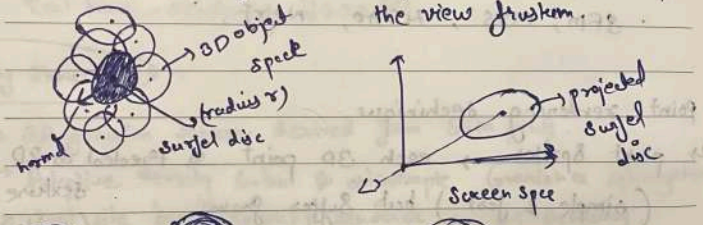
Visibility and occulsion handling

↳ since points do not have explicit connectivity, rendering require depth based techniques.

① Depth Sorting → sort points based on their depth for correct rendering

② α - Blending (opacity Blending) → used in GS for smooth, compact

③ Screen space Culling → removes points outside the view frustum.



Point Based Rendering Eq^n

$$C = \sum_{i=1}^{N} T_i \alpha_i c_i \qquad \alpha_i = 1 - \exp(-\sigma_i \delta_i)$$ is the opacity of each points.

Blending accumulates Color along the ray,

similar to Nerf's ray marching.

anisotropic Gaussian → allowing elliptical splats for better approx. Complex Scene structure

Covariance (Σ) → Gated Gaussian shape.

Spherical Harmonics (SH) → model color & lighting.

Objective → High Quality Novel view synthesis

goals → generate new views of a 3D Scene from a sparse set of images.

Input: SFM with no Surface Normals

Challenge: Need a representation that is both differentiable and fast for rendering.

Why 3D GS? they can be optimized using gradient descent (differentiable)

→ They can be easily projected onto a 2D-Scene
→ They do not require surface normals, which are hard to estimate from SFM data.

adam optimize for gradient updates.

$$G(x) = e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

μ → mean 3D pos of Gaussian

Σ → Covariance Matrix (defines shape & Spread)

Why no Surface Normals?

→ prev. point based methods used small 2D Circles with Normals

→ SFM points are too sparse, making it difficult to estimate accurate Normals.

→ Normals are too Noisy, making optimization unstable.

→ 3D G.S represents geometry without needing normals.

Projection of 3D G. to 2D for Rendering.

① viewing transform matrix W (world Coordinate to Camera Coordinate)

② Covariance Matrix Σ' in Camera Space

$\Sigma' = JW\Sigma W^T J^T$ → Jacobian Matrix (approx the perspective transform)

↙ by removing third Column & row

2×2 (Covar. Matrix)

*

Optimization the Covar. Matrix Σ

↳ without opti, the Gaussian would be isotropic (spherical)

& would not adapt to Complex Geometry of objects.

Covariance Matrix must be +ve semi-definite

direct optim. of C.M is risky due to 6 indepe parameters.

The paper factorize CM into

① S (scaling Matrix) (avoids Numerical Instability)

② R (Rotation Matrix)

$\Sigma = RSS^TR^T$ → ensure stability during optimization.

→ Gradient descent can sometimes produce invalid Σ matrices, causes rendering artifacts or crashes.

Tile Based Rasterization →

- ~~Sorting~~
- prev. Methods had hard limit on the no of splats that could recieve gradient descent.

* Sorting & Blending is Computationally extensive

Tile based rendering → 16 × 16 tiles ← Screen

(Gaussian Sorted) process per tile

→ cull Gaussian outside the view frustum. (99% Gs)

→ Radix Sort → sort Gs in depth order. (front to back)

GPU (fast sorting)

Gaussian are blended using (α)

If incorrect sorting occur further Gs may occur in front of closer ones.

Efficient Gradient Computation (Backward pass)

→ prev method store long lists of blended points per pixel,

Instead of storing, the method reuses the sorted gaussian from the forward pass. (Memory efficient)

→ accumulates color until α reaches 1, stop early reducing computation. (Efficient α blending)

Limitations: artifacts in poorly observed region.

Regularization could improve handling of unseen areas.

Training large scene requi > 20 GB GPU M.

↳ Point cloud compression technique could reduce memory usage.

Training start from 100K Random Gaussian → 6K-10K meaningful Gaus

↑ PSNR → image quality (better.) ↑ SSIM → measure similarity (better.) -1 to 1

↓ L-PIPS → (better.) Capture fine details.

Spherical Harmonics in 3D GS.

→ Instead of storing RGB Color (c) per Gaussian, SH encodes how the Color of the Gaussian changes on the viewing direction.

$$C(v) = \sum_{l=0}^{L} \sum_{m=-l}^{l} c_{lm} Y_l^m(v)$$

(c_{lm}: learned SH coeff of each gaussian; Y_l^m: SH basis func.)

Color as a func of viewing dire. (v)

provides efficient way to store directional appearance

Low SH are fast but smooth out fine details
$l = 0, 1, 2$

High SH ($l \geq 3+$) allows better reflection but increase memory usage.

GS-IR (Inverse Rendering)

Inverse Rendering → reconstruct 3D Scene prop from 2D image by estimating geometry, material prop & illumination.

$$L_o(x,v) = \int_{\Omega} L_i(x,l)\, f_r(l,v)\, (l \cdot n)\, dl$$

(hemisphere of all possible light dirn) → rendering eqn.

- $L_o(x,v)$: (out rad at x in direct v)
- $L_i(x,l)$: (incoming radiance at x from l)
- $f_r(l,v)$: BRDF
- $(l \cdot n)$: Surface normal

(1) in forward Rendering, we Compute $L_o(x,v)$ given Scene prop.

(2) In Inverse Rendering, we start with $L_o(x,v)$ (observed image)

and estimate (1) geometry
(2) Materials
(3) Lighting.

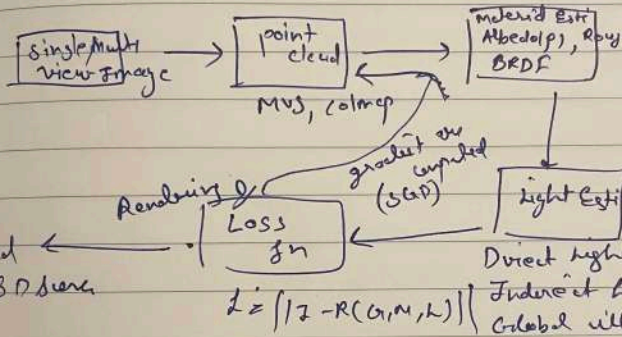
Inverse Rend is ill posed, multiple Combination of G, M, L can produce the same image.

We use diff. renderable rendering, optimization & DL Tech

$\hat{G}, \hat{M}, \hat{L} = \arg\min_{G,M,L} L(I, R(G,M,L))$

$\hat{}$ (estimated parameter)

Loss function (diff b/w Red & Rendered Image)



Nerf → Representing Scene Neural Radiance field for view synthesis

View synthesis → 2D images set → new arbitrary angle view

challenges ① Complex Geometry
② view dependent lighting
③ high resolution details.

Models a 5D func which

① Encodes a 3D point (x, y, z) & viewing direction (θ, ϕ).

② outputs:

① Color (r, g, b) Defines what color is seen at that point.

② Density (σ). How opaque the point is

This func^n is learned using a MLP (fully Connected) without Convolutions layer.

★★

Prior methods use discrete voxels grids or meshes

→ lot of memory to store H.R 3D scenes.

→ Meshes require pre-defined geometry (which isn't avail for real-world images)

Nerf Continous 5D Representation

→ no explicit geometry is needed

→ neural networks learns the underlying structure of the scene.

→ Scene is encoded in the networks weights. (Storage efficient)

Nerf works?

→ render an image from novel viewpoint

① Sample along Camera rays points

② Predict Color & density for each point.

Input (x, y, z, θ, ϕ) in MLP

output (r, g, b, σ) values

Scene Representation 5D Continous 3D spatial location (x, y, z)
2D view Dir (θ, ϕ)

3D coordinate → 8 FCL (256) Relu → σ (volume Density) (depends on x)
+ 256 Dimensional Feature Vector

Feature Vector + viewing direction → FCL (128 + Relu) → RGB (c) (Gbr)

Handling view-Dependent Effects

→ Non-Lambertian Surface

NeRFactor

Shape & reflectance of an object from multi-viewed Captured under one unknown illumination

Goal → to enable scene free viewpoint relighting, & material editing, by factorizing scene into albedo, light visibility & BRDF.

Use NeRF to estimate volumetric geometry, later distilled into surface representation,

① Geometry
② Spatially varying Reflectance
③ Light visibility
④ Environment lighting

Previous work:

NeRV → require multiple lighting conditions.

NeRF) doesn't model shadow & light visibility

PhySG → assumes non-spatially-varying reflectance

Nerfactor:

- uses learned BRDF instead of analytic BRDF
- works with single unknown illumination.

Method:

Input assumption

→ Starts with multi-view images of an object & their camera pose, captured under single unknown lighting conditions

→ Representing using Neural fields.

Each of these fields is parameterized by MLP

The MLP's are trained so that when the model "renders" the object from these neural fields, the output matches the observed input images.

→ Outputs

for every 3D point x on the object's surface

Nerfactor produces

1. Surface Normal (n)
2. Light visibility ($v(\omega_i)$)
3. Albedo (a)
4. Reflectance (z_{BRDF})