



3D Gaussian Splatting

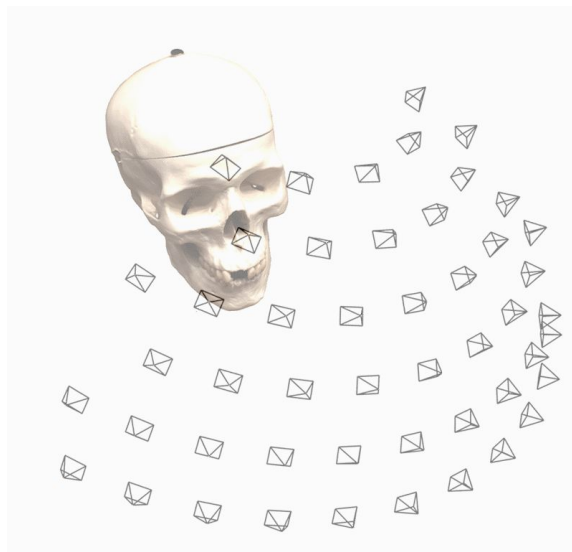
Outils Maths - 27 Mai 2024

Gaspard Thévenon

Introduction

Introduction - Image based rendering

figures from IDR - github

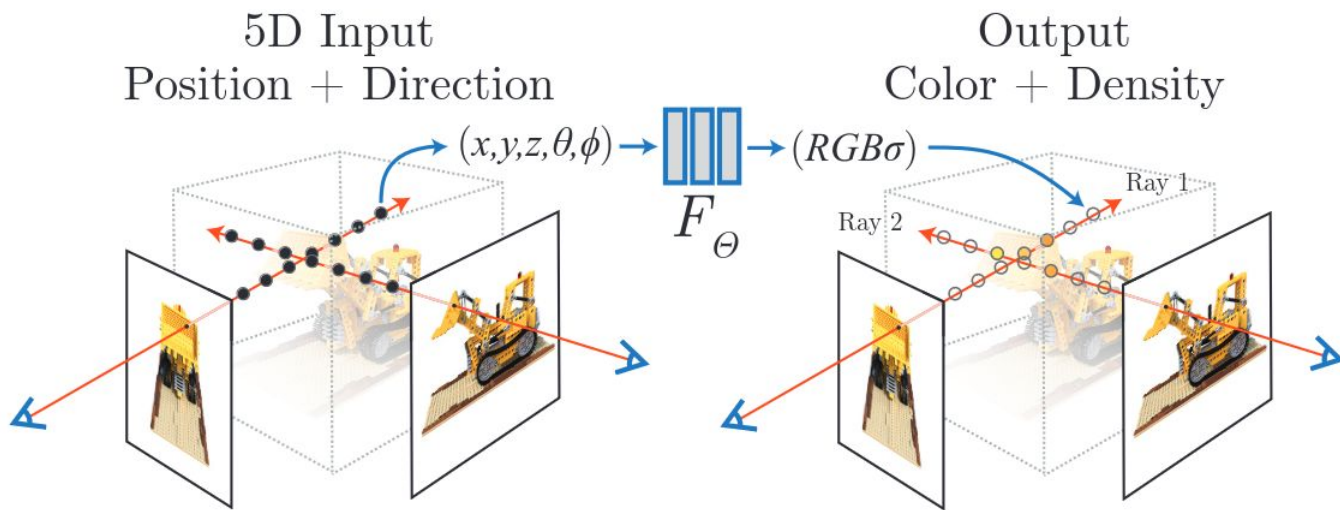


Training views



Novel view synthesis

Introduction - NeRFs



> In August 2020, groundbreaking step in novel view synthesis

> Uses a Multi-Layer Perceptron to encode an implicit position/view function for color and density

> Rendering uses ray marching, sampling the function at each point

figure from Mildenhall et al [2]

Introduction - 3D Gaussian Splatting

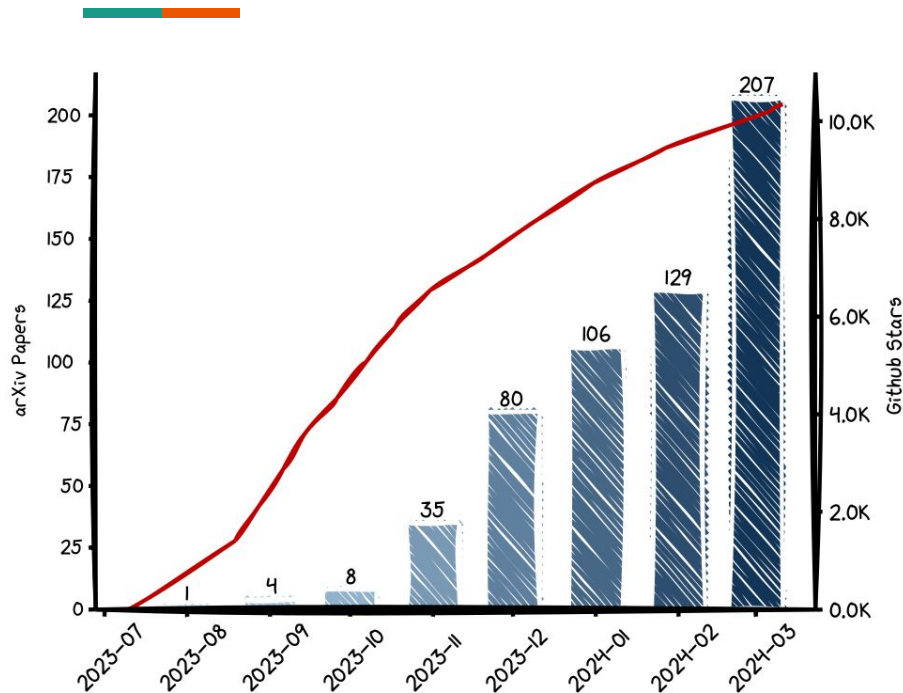
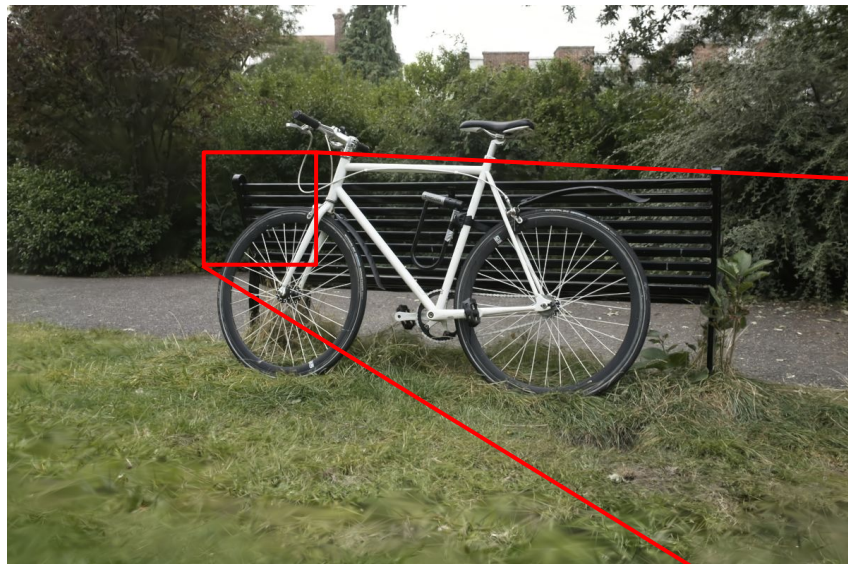


figure from Chen et al [3]

- > Paper [1] in August 2023
- > Had a huge success in the last months
- > Combines rendering quality, low training times, and explicit representation
- > Scenes are represented as hundreds of thousands (or millions) of ellipsoids

3D Gaussian Splatting

3D Gaussian Splatting - Scene representation



A scene is composed of colored points, each one influencing a zone around it according to a 3D Gaussian field.

Image below: Gaussians fields are rendered here as opaque ellipsoids for visualization.



figure from Kerbl et al [1]

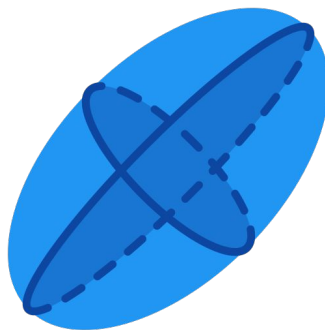
3D Gaussian Splatting - Scene representation



3D GS - Parameters

- Position (mean) p
- Covariance matrix Σ
- Opacity α
- Features: view-dependent RGB,
via spherical harmonics
 $l_{0,0}, l_{1,-1}, l_{1,0}, l_{1,1}, \dots$

$$G(x) = \alpha \exp \left(-\frac{1}{2} (x - p)^T \Sigma^{-1} (x - p) \right)$$



3D GS - Parameters

- Position (mean) p

- Covariance matrix $\Sigma = RSS^T R^T$

- Opacity α

logit space

- Features: view-dependent RGB,
via spherical harmonics

$$l_{0,0}, l_{1,-1}, l_{1,0}, l_{1,1}, \dots$$

Rotation matrix, stored as
a quaternion q

Scaling matrix, stored as 3
values in log space

Rasterization - Gaussian kernels

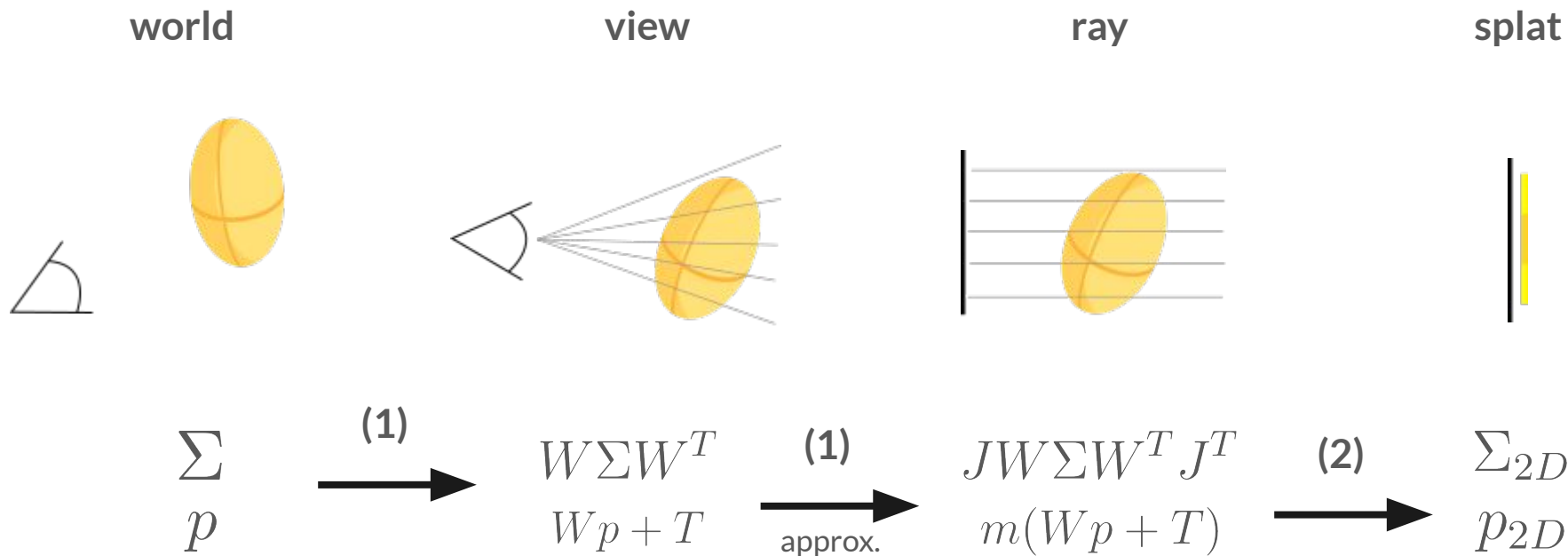
Described by Zwicker et al [7]


$$G_{\Sigma,p}(x) = \frac{1}{2\pi|\Sigma^{-1}|} \exp\left(-\frac{1}{2} * (x - p)^T \Sigma^{-1} (x - p)\right)$$

Apply affine transformation $\Phi(x) = Mx + b$: **(1)** $\Phi(G_{\Sigma,p}(x)) = \frac{1}{|M^{-1}|} G_{M\Sigma M^T, \Phi(p)}(\Phi(x))$

Integrate over z axis: **(2)** $\int_{\mathbb{R}} G_{\Sigma,p}(x, y, z) dz = G_{\Sigma[:2,:2], p[:2]}(x, y)$

Rasterization - Splatting



Rasterization - Splatting - Computation of J



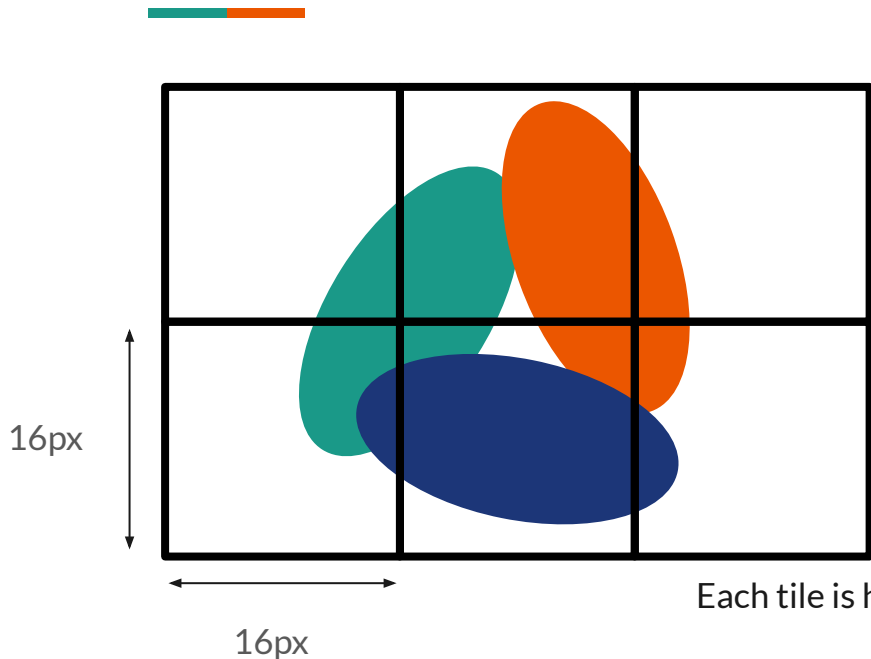
Projection transformation:

$$m(u) = \begin{pmatrix} u_0/u_2 \\ u_1/u_2 \\ ||(u_0, u_1, u_2)|| \end{pmatrix}$$

First-order affine approximation at p:

$$m(u) = m(p) + J(u - p), J = \begin{pmatrix} 1/u_2 & 0 & -u_0/u_2^2 \\ 0 & 1/u_2 & -u_1/u_2^2 \\ u_0/l & u_1/l & u_2/l \end{pmatrix}, l = ||(u_0, u_1, u_2)||$$

Rasterization - Blending



Each tile is handled by a CUDA kernel as a thread block

Buffer of intersections between gaussians and tiles, sorted (by tile ID, then Depth)

Tile 1	Depth
Tile 2	Depth
Tile 2	Depth
Tile 3	Depth
Tile 4	Depth
Tile 4	Depth

...

For each pixel, intersecting ellipses are blended together (by depth order):

$$c(x) = \sum_{k=0}^N \alpha_k(x) c_k \prod_{i=0}^{k-1} (1 - \alpha_i(x))$$

3DGS - Rasterizing

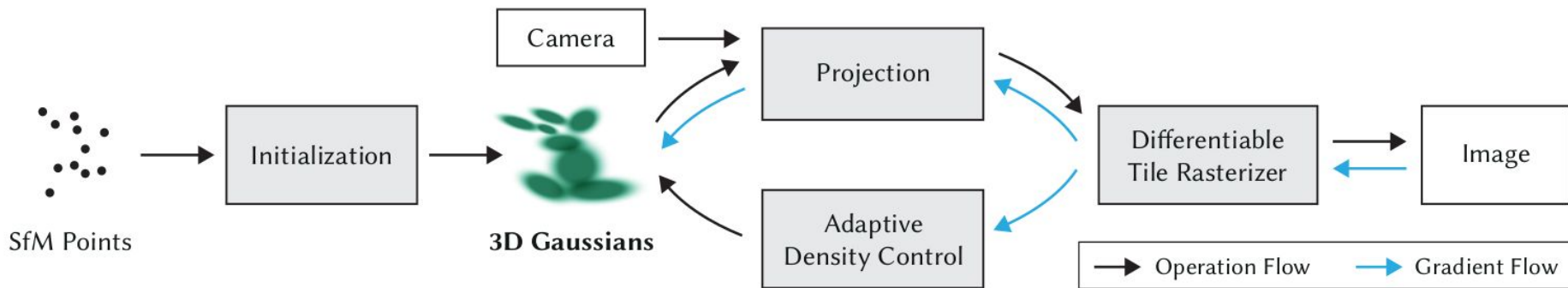
~ 350k Gaussians

video from Kerbl et al [1]



Optimization - Overview

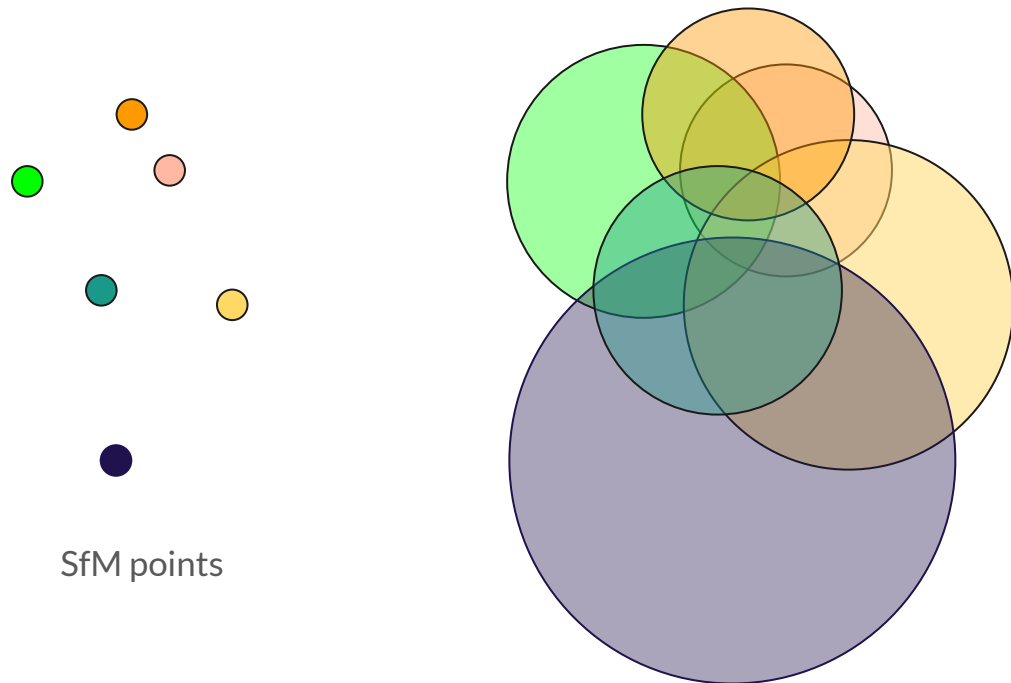
figure from Kerbl et al [1]



- Gaussian parameters are optimized through back-propagation
- They are initialized with SfM Points
- One camera is chosen at each iteration, and the rendered image is compared to the groundtruth

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}}$$

Optimization - Initialization

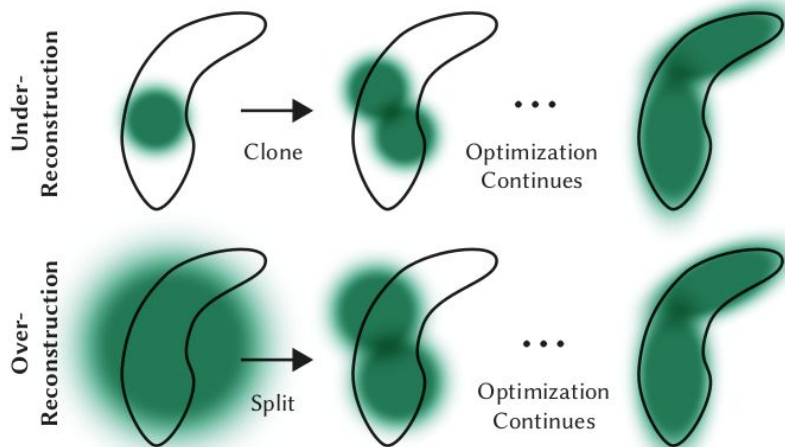


- > One Gaussian per point
- > Isotropic kernel
- > Radius: mean distance to the closest three points

Optimization - Density control

Adding Gaussians

If there are not enough Gaussians to render details, new ones are added. Every T iterations, Gaussians with positional gradients above a threshold are split or cloned.



Removing Gaussians

> Every T iterations, Gaussians that are too large are removed ; Gaussians with opacity less than a threshold are removed

> Every K iterations, opacities are reset to a low value, in order to remove flying outliers

*Typical values in original implementation:
 $T = 100$, $K = 3000$,
after warm-up phase*

3DGS - Results



Dataset	Mip-NeRF360						Tanks&Temples						Deep Blending					
Method Metric	SSIM [↑]	PSNR [↑]	LPIPS [↓]	Train	FPS	Mem	SSIM [↑]	PSNR [↑]	LPIPS [↓]	Train	FPS	Mem	SSIM [↑]	PSNR [↑]	LPIPS [↓]	Train	FPS	Mem
Plenoxels	0.626	23.08	0.463	25m49s	6.79	2.1GB	0.719	21.08	0.379	25m5s	13.0	2.3GB	0.795	23.06	0.510	27m49s	11.2	2.7GB
INGP-Base	0.671	25.30	0.371	5m37s	11.7	13MB	0.723	21.72	0.330	5m26s	17.1	13MB	0.797	23.62	0.423	6m31s	3.26	13MB
INGP-Big	0.699	25.59	0.331	7m30s	9.43	48MB	0.745	21.92	0.305	6m59s	14.4	48MB	0.817	24.96	0.390	8m	2.79	48MB
M-NeRF360	0.792 [†]	27.69 [†]	0.237 [†]	48h	0.06	8.6MB	0.759	22.22	0.257	48h	0.14	8.6MB	0.901	29.40	0.245	48h	0.09	8.6MB
Ours-7K	0.770	25.60	0.279	6m25s	160	523MB	0.767	21.20	0.280	6m55s	197	270MB	0.875	27.78	0.317	4m35s	172	386MB
Ours-30K	0.815	27.21	0.214	41m33s	134	734MB	0.841	23.14	0.183	26m54s	154	411MB	0.903	29.41	0.243	36m2s	137	676MB

Improvements and Applications

Applications



- > Reconstruction of dynamic scenes [21, 22, 23]
- > Head animation [24, 25, 33]
- > AI-Generated Content [26, 27, 30, 34]
- > SLAM, Autonomous Driving, Medical imaging, ...

Applications - Reconstruction of dynamic scenes



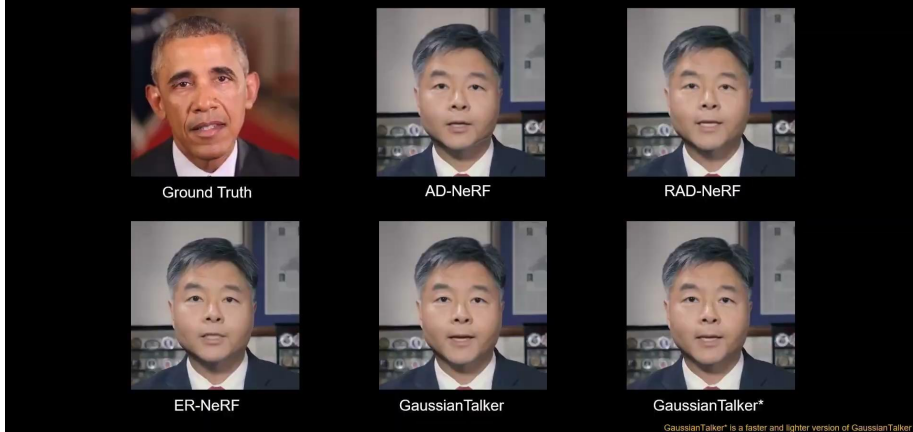
video from Yang et al [21]



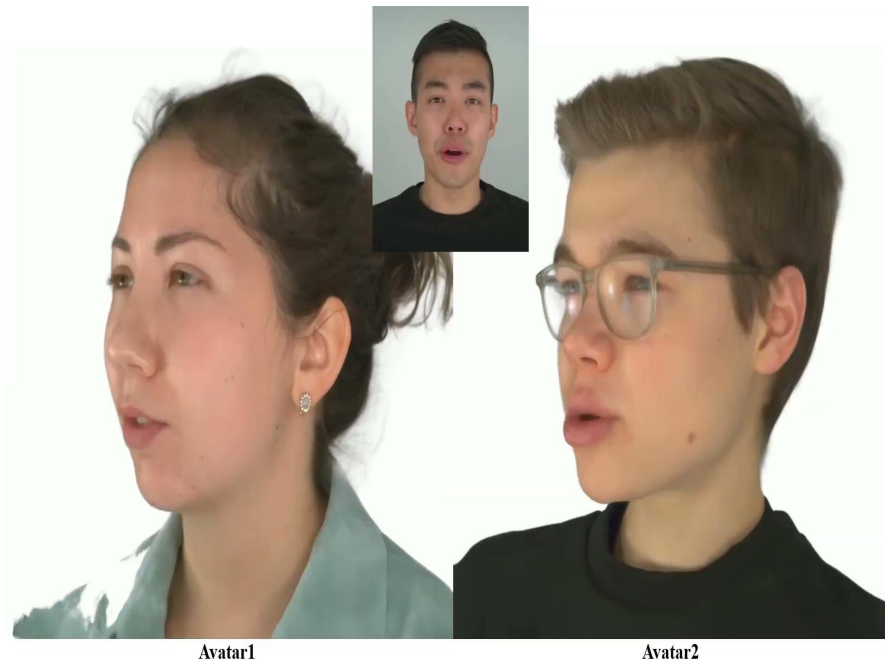
video from Yang et al [22]

Applications - Head animation

Cross-Driven Results Comparison



video from Cho et al [24]

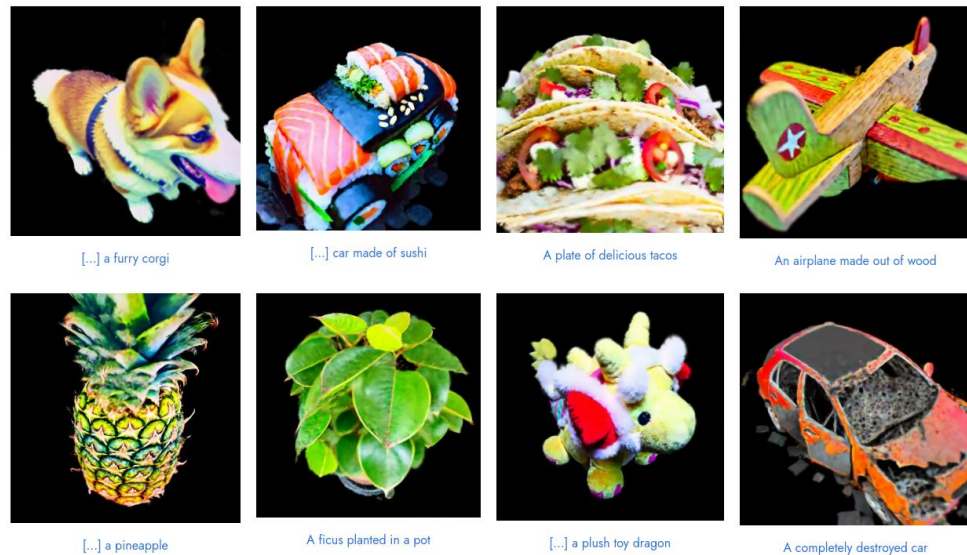


video from Xu et al [25]

Applications - AI-Generated Content



video from Tang et al [26]



figures from Chen et al [27]

Issues

- Memory inefficient: parameters are not compressed at all
- Despite optimization mechanisms, there are still “useless” gaussians and artifacts around hard edges
- No embedded geometry or lighting/material information

Dataset Method Metric	Mip-NeRF360					
	<i>SSIM</i> [↑]	<i>PSNR</i> [↑]	<i>LPIPS</i> [↓]	Train	FPS	Mem
Plenoxels	0.626	23.08	0.463	25m49s	6.79	2.1GB
INGP-Base	0.671	25.30	0.371	5m37s	11.7	13MB
INGP-Big	0.699	25.59	0.331	7m30s	9.43	48MB
M-NeRF360	0.792 [†]	27.69 [†]	0.237 [†]	48h	0.06	8.6MB
Ours-7K	0.770	25.60	0.279	6m25s	160	523MB
Ours-30K	0.815	27.21	0.214	41m33s	134	734MB



Domains of research around 3DGS improvement



Several domains of study around 3DGS since August 2023:

- Memory-efficiency: Reduce memory usage of parameters [17, 20, 32]
- Sparsity: Reduce number of Gaussians needed to represent the scene [17,18, 20, 29, 32]
- Improvement of the optimization routine:
 - > Progressive frequency regularization [19]
 - > Improvement of the adaptive density control heuristic [28, 31]
 - > Distributed optimization [35]
 - > Many others (cf surveys [3,13])

Memory footprint and Sparsity

- Sparsity: Typical idea, compute a score for each Gaussian, denoting its level of contribution to the scene
 - > Example [17] of score for Gaussian j :

$$s_j = \sum_{\text{view } p} \sum_{\text{pixel ray } r_j \in p} \mathbf{1}(\mathbf{G}_j, r_j) \alpha_j V_j$$

~ volume

(weighted) intersection Gaussian/ray

- Memory footprint: use vector quantization with a codebook, to store parameters for Gaussians
 - > [17]: only for SH coefficients, for low-contribution Gaussians...

Memory footprint and Sparsity

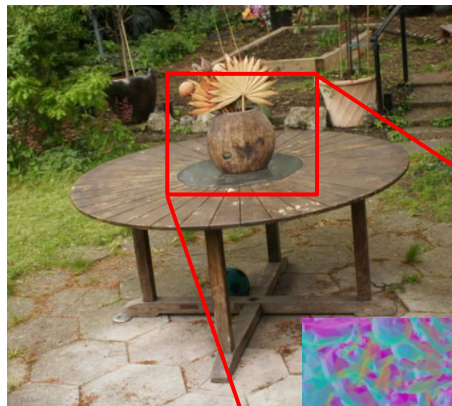
Methods	Mip-NeRF 360 Datasets					Tank and Temple Datasets				
	FPS↑	Size↓	PSNR↑	SSIM↑	LPIPS↓	FPS↑	Size↓	PSNR↑	SSIM↑	LPIPS↓
Plenoxels [67]	6.79	2.1 GB	23.08	0.626	0.463	11.2	2.3 GB	21.08	0.719	0.379
INGP-Big [46]	9.43	48MB	25.59	0.699	0.331	14.4	48MB	21.92	0.745	0.305
Mip-NeRF 360 [5]	0.06	8.6MB	27.69	0.792	0.237	0.14	8.6MB	22.22	0.759	0.257
VQ-DVGO [40]	4.65	63MB	24.23	0.636	0.393	-	-	-	-	-
3D-GS [32]	134	734MB	27.21	0.815	0.214	154	411MB	23.14	0.841	0.183
3D-GS* [32]	119	724MB	27.53	0.810	0.227	106	380MB	23.11	0.822	0.219
Ours	209	42MB	27.28	0.805	0.243	209	22MB	23.11	0.817	0.231

figure from Fan et al [17]

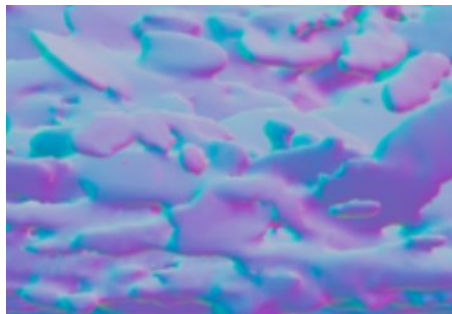
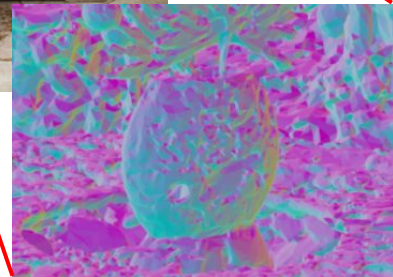
Surfaces and mesh reconstruction

Issues with traditional 3DGS

- > 3D Gaussians are optimized only with a visual error, and they don't offer a natural surface structure (i.e. normals).
- > Empirically, Gaussians centers and orientations rarely form a surface.



figures from Guédon et al [4]



Surface features - Overview

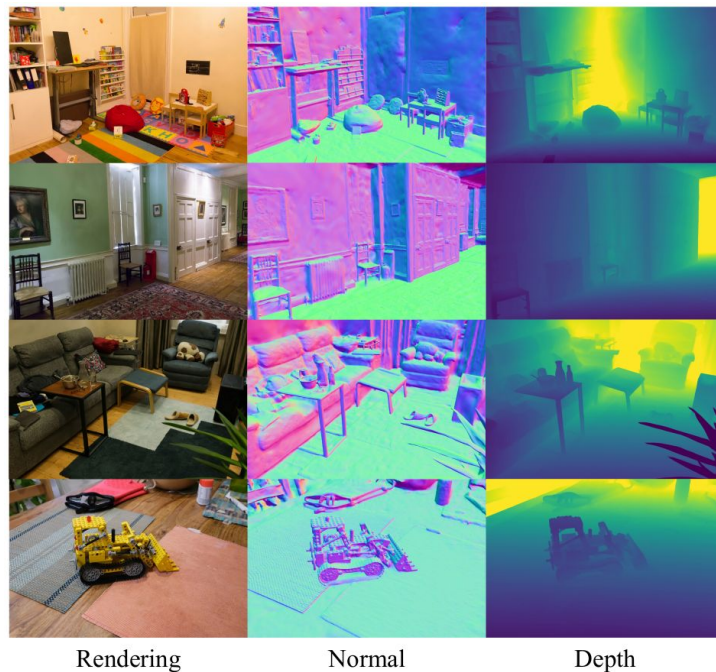
> General ideas: add (hard/soft) constraints on Gaussians to have “surface-like” parameters, like:

- one axis with very low scale (or hardcoded 0)
- 0/1 opacity

The normal is then defined as the smallest axis. The learning of normals is usually supervised [5,6,8] via pseudo-normals (computed with depth) or normal priors.

[4] optimizes an SDF instead.

Meshing usually involves Poisson reconstruction.



figures from Dai et al [8]

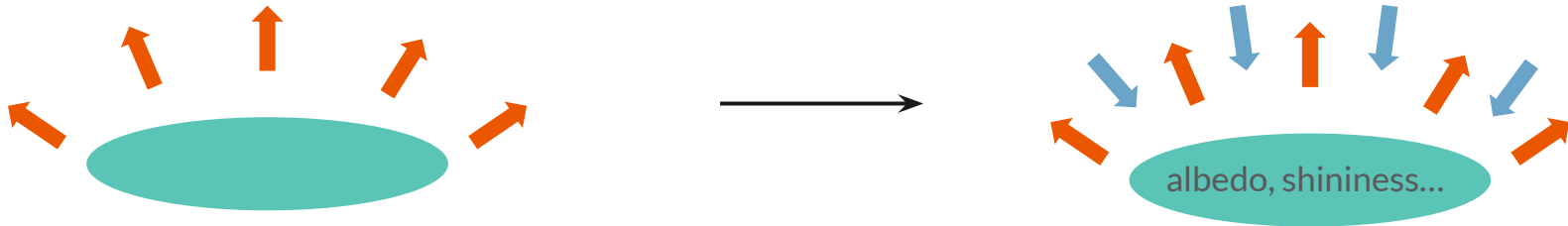
Intrinsic decomposition and Editing

Editing - Overview of existing work

Existing work on Gaussian splatting editing [9, 10, 11, 12] starts from the idea that:

3DGS only models outgoing light, for every splat. This merges all physical properties (incoming light, shininess, albedo, etc).

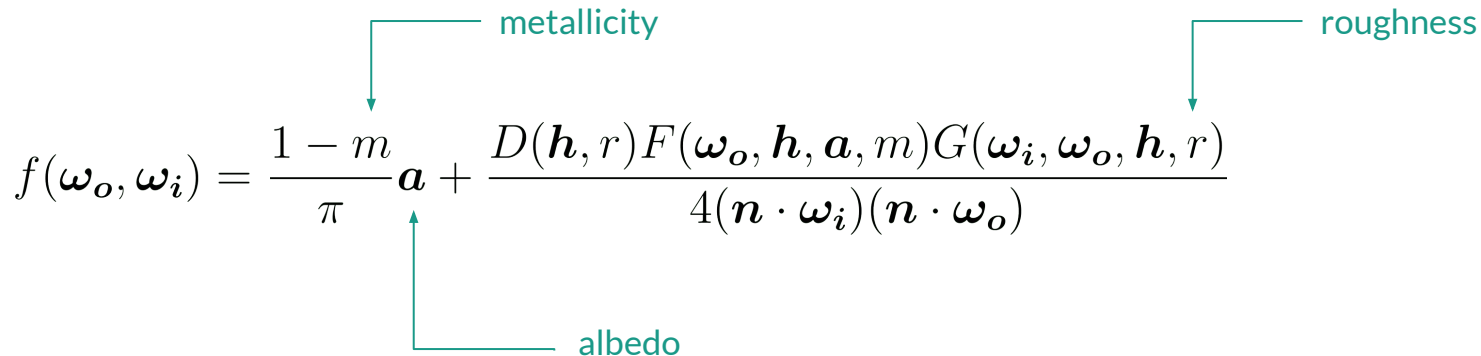
> Disantangle them by adding material/lighting parameters to Gaussians, which are learned during optimization.



Related to similar works on NeRFs [14, 15].

Material model - Microfacet BRDF

Simplified Disney BRDF model (Cook-Torrance):



The diagram shows the Cook-Torrance BRDF equation with three parameters annotated by teal arrows: 'metallicity' points to the m parameter in the first term, 'albedo' points to the a parameter in the first term, and 'roughness' points to the r parameter in the second term.

$$f(\omega_o, \omega_i) = \frac{1 - m}{\pi} a + \frac{D(\mathbf{h}, r) F(\omega_o, \mathbf{h}, a, m) G(\omega_i, \omega_o, \mathbf{h}, r)}{4(\mathbf{n} \cdot \omega_i)(\mathbf{n} \cdot \omega_o)}$$

3 parameters: **albedo**, **metallicity**, and **roughness**

Image-based lighting



[9, 10, 11, 12] use an envmap to model environmental light, which is global to the whole scene (usually represented as a cubemap or as spherical harmonic coefficients):

$$L_{global}(\omega_i)$$

Then, the incoming light is modeled (in [9, 10, 11]) as:

$$L_i(\omega_i) = V(\omega_i)L_{global}(\omega_i) + L_{local}(\omega_i)$$

IBL Efficiently - Split-sum approximation

Introduced by [16], approximates the specular part by:
with:

$$c_{\text{specular}} \approx I_{\text{light}}(\omega_o, \mathbf{n}, r) \cdot I_{\text{mat}}(\omega_o, \mathbf{n}, r, m)$$

$$I_{\text{light}}(\omega_o, \mathbf{n}, r) = \int_{\Omega} L_i(\omega_i) D(\omega_i, \omega_o, r) (\omega_i \cdot \mathbf{n}) d\omega_i \approx I_{\text{light}}(\mathbf{n}, \mathbf{n}, r)$$

$$I_{\text{mat}}(\omega_o, \mathbf{n}, r, m) = \int_{\Omega} \frac{DFG}{4(\omega_o \cdot \mathbf{n})} d\omega_i = F_0 \cdot \text{scale}(\omega_o \cdot \mathbf{h}, r) + \text{bias}(\omega_o \cdot \mathbf{h}, r)$$

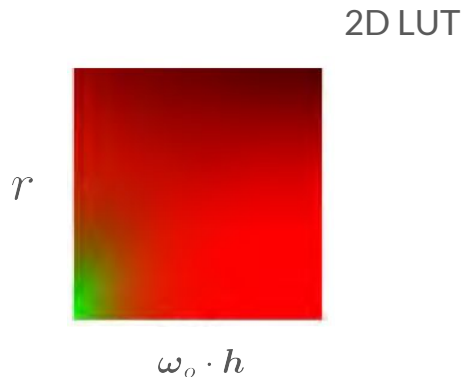


figure from Karis [16]

Editing - Overview of existing work



To summarize, usual parameters:

- Global envmap for environmental light
- Per Gaussian: albedo, roughness, normal, metallicity, visibility, local light (+ means, covariance ...)



Additional loss terms



Intrinsic decomposition is very ill-posed: there are a lot of ambiguities. So additional loss terms are present to supervise the parameters learning. Examples of losses:

White light:
$$\mathcal{L}_{light} = \sum_{i=1}^3 \left(3l_i - \sum_{j=1}^3 l_j \right)$$

+ terms to supervise normals
(pseudo-normals, prior normals, ...)

**Smooth parameters
change:**

$$\mathcal{L}_{smooth}(x) = \|\nabla x\| \exp(-\|\nabla I_{gt}\|)$$
$$x \in \{\mathbf{a}, r, m, d, \mathbf{n}\}$$

Results - Intrinsic decomposition



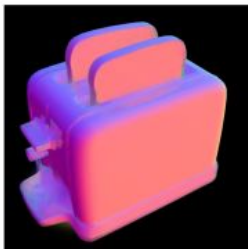
when it works



GT



Render



Normal



Metallic



Roughness



Specular



Diffuse

Results - Intrinsic decomposition

when it works less

GT

Render

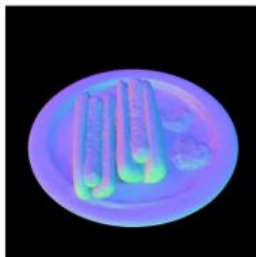
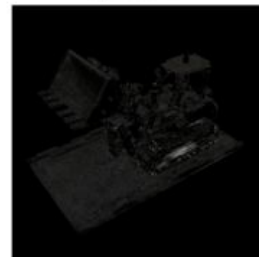
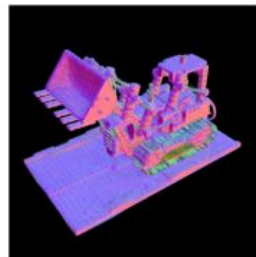
Albedo GT

Albedo

Normal

Diffuse

Specular



Results - Relighting



In these articles, “relighting” means changing the learned envmap by a new one before rendering.



figures from Shi et al [10]

Conclusion



Questions ?



References



- [1] “3D Gaussian Splatting for Real-Time Radiance Field Rendering”, Kerbl, Kopanas, Leimkühler, Drettakis, 2023
- [2] “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”, Mildenhall, Srinivasan, Matthew Tancik, Barron, Ramamoorthi, Ng, 2020
- [3] “A Survey on 3D Gaussian Splatting”, Chen, Wang, 2024
- [4] “SuGaR: Surface-Aligned Gaussian Splatting for Efficient 3D Mesh Reconstruction and High-Quality Mesh Rendering”, Guédon, Lepetit, 2023
- [5] “2D Gaussian Splatting for Geometrically Accurate Radiance Fields”, Huang, Yu, Chen, Geiger, Gao, 2024
- [6] “DN-Splatter: Depth and Normal Priors for Gaussian Splatting and Meshing”, Turkulainen, Ren, Melekhov, Seiskari, Rahtu, Kannala, 2024
- [7] “EWA Splatting”, Zwicker, Pfister, van Baar, Gross, 2002
- [8] “High-quality Surface Reconstruction using Gaussian Surfels”, Dai, Xu, Xie, Liu, Wang, Xu, 2024
- [9] “Relightable 3D Gaussian: Real-time Point Cloud Relighting with BRDF Decomposition and Ray Tracing”, Gao, Gu, Lin, Zhu, Cao, Zhang, Yao, 2023
- [10] “GIR: 3D Gaussian Inverse Rendering for Relightable Scene Factorization”, Shi, Wu, Wu, Liu, Zhao, Feng, Liu, Zhang, Zhang, Zhou, Ding, Wang, 2023
- [11] “GS-IR: 3D Gaussian Splatting for Inverse Rendering”, Liang, Zhang, Feng, Shan, Jia, 2024
- [12] “DeferredGS: Decoupled and Editable Gaussian Splatting with Deferred Shading”, Wu, Sun, Lai, Ma, Kobbelt, Gao, 2024
- [13] “3D Gaussian Splatting as New Era: A Survey”, Fei, Xu, Zhang, Zhou, Yang, He, 2024
- [14] “NeILF: Neural Incident Light Field for Physically-based Material Estimation”, Yao, Zhang, Liu, Qu, Fang, McKinnon, Tsin, Quan, 2022
- [15] “NeILF++: Inter-reflectable Light Fields for Geometry and Material Estimation”, Zhang, Yao, Li, Liu, Fang, McKinnon, Tsin, Quan, 2023
- [16] “Real Shading in Unreal Engine 4”, Karis, 2013
- [17] “LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS”, Fan, Wang, Wen, Zhu, Xu, Wang, 2024
- [18] “GDGS: Gradient Domain Gaussian Splatting for Sparse Representation of Radiance Fields”, Gong, 2024
- [19] “FreGS: 3D Gaussian Splatting with Progressive Frequency Regularization”, Zhang, Zhan, Xu, Lu, Xing, 2024
- [20] “Compact 3D Gaussian Representation for Radiance Field”, Lee, Rho, Sun, Ko, Park, 2024
- [21] “Real-time Photorealistic Dynamic Scene Representation and Rendering with 4D Gaussian Splatting”, Yang, Yang, Pan, Zhang, 2024
- [22] “Deformable 3D Gaussians for High-Fidelity Monocular Dynamic Scene Reconstruction”, Yang, Gao, Zhou, Jiao, Zhang, Jin, 2024
- [23] “SC-GS: Sparse-Controlled Gaussian Splatting for Editable Dynamic Scenes”, Huan, Sun, Yang, Lyu, Cao, Qi, 2024

References



- [24] “GaussianTalker: Real-Time High-Fidelity Talking Head Synthesis with Audio-Driven 3D Gaussian Splatting”, Cho, Lee, Yoon, Hong, Ko, Ahn, Kim, 2024
- [25] “Gaussian Head Avatar: Ultra High-fidelity Head Avatar via Dynamic Gaussians”, Xu, Chen, Li, Zhang, Wang, Zhen, Liu, 2024
- [26] “DreamGaussian: Generative Gaussian Splatting for Efficient 3D Content Creation”, Tang, Ren, Zhou, Liu, Zheng, 2023
- [27] “Gsgen: Text-to-3D using Gaussian Splatting”, Chen, Wang, Wang, Liu, 2024
- [28] “A New Split Algorithm for 3D Gaussian Splatting”, Feng, Cao, Chen, Mu, Martin, Hu, 2024
- [29] “SUNDAE: Spectrally Pruned Gaussian Fields with Neural Compensation”, Yang, Zhu, Jiang, Ye, Chen, Zhang, Chen, Zhao, Zhao, 2024
- [30] “ART3D: 3D Gaussian Splatting for Text-Guided Artistic Scenes Generation”, Li, Tang, Huang, Li, 2024
- [31] “AtomGS: Atomizing Gaussian Splatting for High-Fidelity Radiance Field”, Liu, Xu, Hu, Chen, Feng, 2024
- [32] “Reducing the Memory Footprint of 3D Gaussian Splatting”, Papantonakis, Kopanas, Kerbl, Lanvin, Drettakis, 2024
- [33] “Gaussian Head & Shoulders: High Fidelity Neural Upper Body Avatars with Anchor Gaussian Guided Texture Warping”, Wu, Yang, Guo, Wan, Zhong, Oztireli, 2024
- [34] “GarmentDreamer: 3DGS Guided Garment Synthesis with Diverse Geometry and Texture Details”, Li, Li, Jiang, Xie, Gao, Wang, Yang, Jiang, 2024
- [35] “DoGaussian: Distributed-Oriented Gaussian Splatting for Large-Scale 3D Reconstruction Via Gaussian Consensus”, Chen, 2024