2D Gaussian Splatting for Geometrically Accurate Radiance Fields

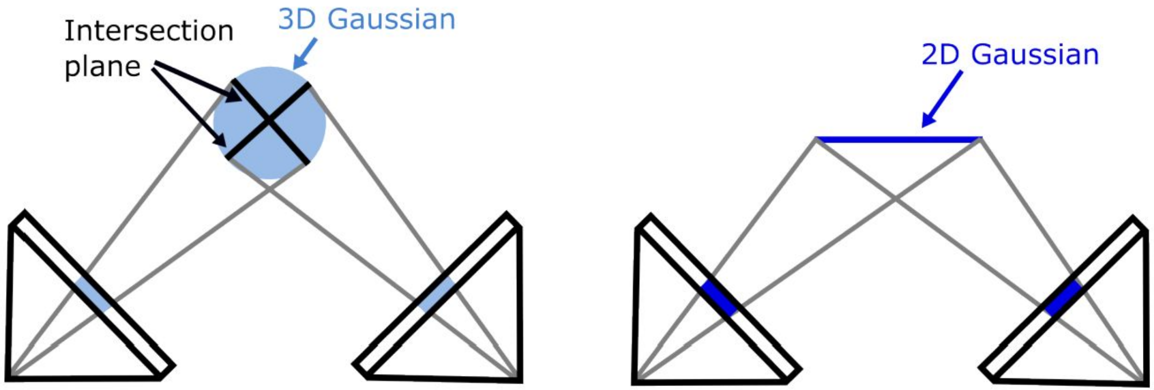
Reviewer: Esteban Wirth
Archeologist: Esteban Wirth
Hacker: Leonardo Mendonça
PhD Student: Leonardo Mendonça

Reviewer Section

Esteban Wirth

Objectives of the paper

- Create a model that accurately describes the surface of objects in a scene by using only pictures of the scene from different angles and minimal restrictions.
- Improve depth consistency from the 3DGS model by introducing explicit ray-splat intersections over 2D Gaussians;
- Create these models in reasonable time and accuracy.



Model: 3DGS to 2DGS

The mathematical model is based on the 3D Gaussian Splatting

$$\mathcal{G}(\mathbf{p}) = \exp(-\frac{1}{2}(\mathbf{p} - \mathbf{p}_k)^\top \Sigma^{-1} (\mathbf{p} - \mathbf{p}_k)) \tag{1}$$

$$\Sigma' = \mathbf{J}\mathbf{W}\Sigma\mathbf{W}^\top\mathbf{J}^\top$$

$$\mathbf{c}(\mathbf{x}) = \sum_{k=1}^{K} \mathbf{c}_k \, \alpha_k \, \mathcal{G}_k^{2D}(\mathbf{x}) \prod_{j=1}^{k-1} (1 - \alpha_j \, \mathcal{G}_j^{2D}(\mathbf{x})) \tag{3}$$

The model adapts the 3DGS model by eliminating the third row and column of the adapted covariance matrix.

Unfortunately this is not justified or explained. The reasons why, or if, this works are not presented in the paper.

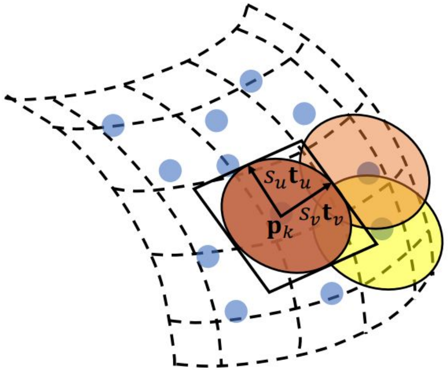
Model: Multiview reconstruction of 2D Gaussian

$$P(u, v) = \mathbf{p}_k + s_u \mathbf{t}_u u + s_v \mathbf{t}_v v = \mathbf{H}(u, v, 1, 1)^{\mathrm{T}} \tag{4}$$

$$\text{where } \mathbf{H} = \begin{bmatrix} s_u \mathbf{t}_u & s_v \mathbf{t}_v & \mathbf{0} & \mathbf{p}_k \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{RS} & \mathbf{p}_k \\ \mathbf{0} & 1 \end{bmatrix} \tag{5}$$

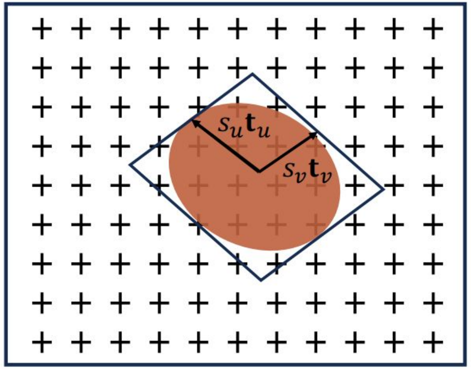
$$\mathcal{G}(\mathbf{u}) = \exp\left(-\frac{u^2 + v^2}{2}\right)$$

Tangent frame (u,v)



2D Gaussian Splat
in object space

Image frame (x,y)



2D Gaussian Splat
in image space

Model: Ray-splat intersection

$$\mathbf{x} = (xz, yz, z, 1)^{\mathrm{T}} = \mathbf{W}P(u, v) = \mathbf{W}\mathbf{H}(u, v, 1, 1)^{\mathrm{T}} \quad (7)$$

$$\mathbf{h}_u = (\mathbf{W}\mathbf{H})^{\mathrm{T}}\mathbf{h}_x \quad \mathbf{h}_v = (\mathbf{W}\mathbf{H})^{\mathrm{T}}\mathbf{h}_y \quad (8)$$

$$\mathbf{h}_u \cdot (u, v, 1, 1)^{\mathrm{T}} = \mathbf{h}_v \cdot (u, v, 1, 1)^{\mathrm{T}} = 0 \quad (9)$$

$$u(\mathbf{x}) = \frac{\mathbf{h}_u^2\mathbf{h}_v^4 - \mathbf{h}_u^4\mathbf{h}_v^2}{\mathbf{h}_u^1\mathbf{h}_v^2 - \mathbf{h}_u^2\mathbf{h}_v^1} \qquad v(\mathbf{x}) = \frac{\mathbf{h}_u^4\mathbf{h}_v^1 - \mathbf{h}_u^1\mathbf{h}_v^4}{\mathbf{h}_u^1\mathbf{h}_v^2 - \mathbf{h}_u^2\mathbf{h}_v^1} \quad (10)$$

With hx = (-1,0,0,x) and hy = (0,-1,0,y)

Model: Degenerate Solutions and Rasterization

$$\hat{\mathcal{G}}(\mathbf{x}) = \max\left\{\mathcal{G}(\mathbf{u}(\mathbf{x})), \mathcal{G}(\frac{\mathbf{x}-\mathbf{c}}{\sigma})\right\} \tag{11}$$

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1} \mathbf{c}_i \, \alpha_i \, \hat{\mathcal{G}}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1} (1 - \alpha_j \, \hat{\mathcal{G}}_j(\mathbf{u}(\mathbf{x}))) \tag{12}$$

There is an abuse of notation.

It is unclear what definition of G is being used in each case of the maximum

Training

$$\mathcal{L}_d = \sum_{i,j} \omega_i \omega_j |z_i - z_j|$$

$$\omega_i = \alpha_i \hat{\mathcal{G}}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1}(1 - \alpha_j \hat{\mathcal{G}}_j(\mathbf{u}(\mathbf{x})))$$

$$\mathcal{L}_n = \sum_i \omega_i (1 - \mathbf{n}_i^{\mathrm{T}} \mathbf{N})$$

$$\mathbf{N}(x, y) = \frac{\nabla_x \mathbf{p}_s \times \nabla_y \mathbf{p}_s}{|\nabla_x \mathbf{p}_s \times \nabla_y \mathbf{p}_s|}$$

$$\mathcal{L} = \mathcal{L}_c + \alpha \mathcal{L}_d + \beta \mathcal{L}_n$$

Where Lc is a color loss-function and is taken from 3DGS

Ld is a depth distortion loss-function

Ln is a normal consistency loss-function

Training

Input (A) w/o. NC (B) w/o. DD Full Model

Experiments and Results: DTU Dataset

Table 1. Quantitative comparison on the DTU Dataset [Jensen et al. 2014]. Our 2DGS achieves the highest reconstruction accuracy among other methods and provides 100× speed up compared to the SDF based baselines.

		24	37	40	55	63	65	69	83	97	105	106	110	114	118	122	Mean	Time
implicit	NeRF [Mildenhall et al. 2021]	1.90	1.60	1.85	0.58	2.28	1.27	1.47	1.67	2.05	1.07	0.88	2.53	1.06	1.15	0.96	1.49	> 12h
	VolSDF [Yariv et al. 2021]	1.14	1.26	0.81	0.49	1.25	0.70	0.72	1.29	1.18	0.70	0.66	1.08	0.42	0.61	0.55	0.86	>12h
	NeuS [Wang et al. 2021]	1.00	1.37	0.93	0.43	1.10	0.65	0.57	1.48	1.09	0.83	0.52	1.20	0.35	0.49	0.54	0.84	>12h
explicit	3DGS [Kerbl et al. 2023]	2.14	1.53	2.08	1.68	3.49	2.21	1.43	2.07	2.22	1.75	1.79	2.55	1.53	1.52	1.50	1.96	11.2 m
	SuGaR [Guédon and Lepetit 2023]	1.47	1.33	1.13	0.61	2.25	1.71	1.15	1.63	1.62	1.07	0.79	2.45	0.98	0.88	0.79	1.33	~ 1h
	2DGS-15k (Ours)	0.48	0.92	0.42	0.40	1.04	0.83	0.83	1.36	1.27	0.76	0.72	1.63	0.40	0.76	0.60	0.83	5.5 m
	2DGS-30k (Ours)	0.48	0.91	0.39	0.39	1.01	0.83	0.81	1.36	1.27	0.76	0.70	1.40	0.40	0.76	0.52	0.80	10.9 m

Experiments and Results: DTU Dataset

Table 3. Performance comparison between 2DGS (ours), 3DGS and SuGaR on the DTU dataset [Jensen et al. 2014]. We report the averaged chamfer distance, PSNR (training-set view), reconstruction time, and model size.

	CD ↓	PSNR ↑	Time ↓	MB (Storage) ↓
3DGS [Kerbl et al. 2023]	1.96	**35.76**	11.2 m	113
SuGaR [Guédon and Lepetit 2023]	1.33	34.57	~1 h	1247
2DGS-15k (Ours)	0.83	33.42	**5.5 m**	**52**
2DGS-30k (Ours)	**0.80**	34.52	10.9 m	**52**

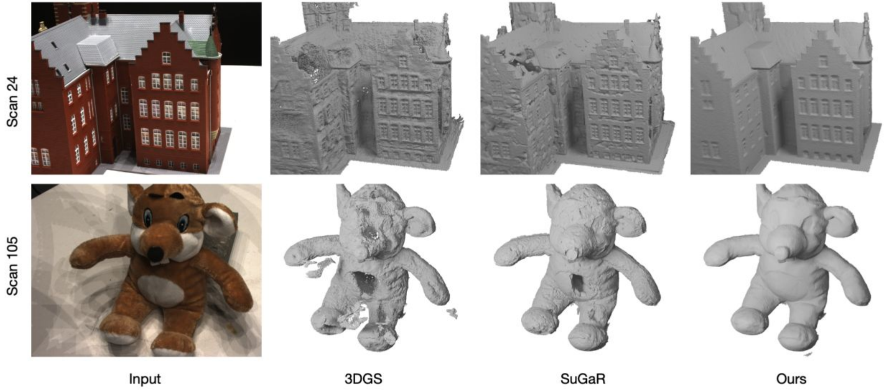


Fig. 5. Qualitative comparison on the DTU benchmark [Jensen et al. 2014]. Our 2DGS produces detailed and noise-free surfaces.

Experiments and Results: Tanks and Temples Dataset

Table 2. Quantitative results on the Tanks and Temples Dataset [Knapitsch et al. 2017]. We report the F1 score and training time.

	NeuS	Geo-Neus	Neurlangelo	SuGaR	3DGS	Ours
Barn	0.29	0.33	0.70	0.14	0.13	0.41
Caterpillar	0.29	0.26	0.36	0.16	0.08	0.23
Courthouse	0.17	0.12	0.28	0.08	0.09	0.16
Ignatius	0.83	0.72	0.89	0.33	0.04	0.51
Meetingroom	0.24	0.20	0.32	0.15	0.01	0.17
Truck	0.45	0.45	0.48	0.26	0.19	0.45
Mean	0.38	0.35	0.50	0.19	0.09	0.32
Time	>24h	>24h	>24h	>1h	14.3 m	15.5 m

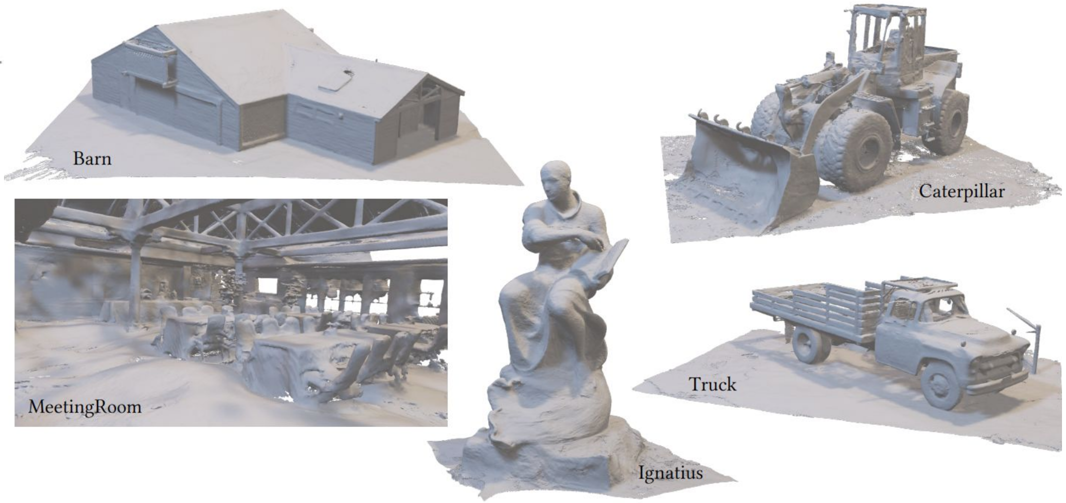
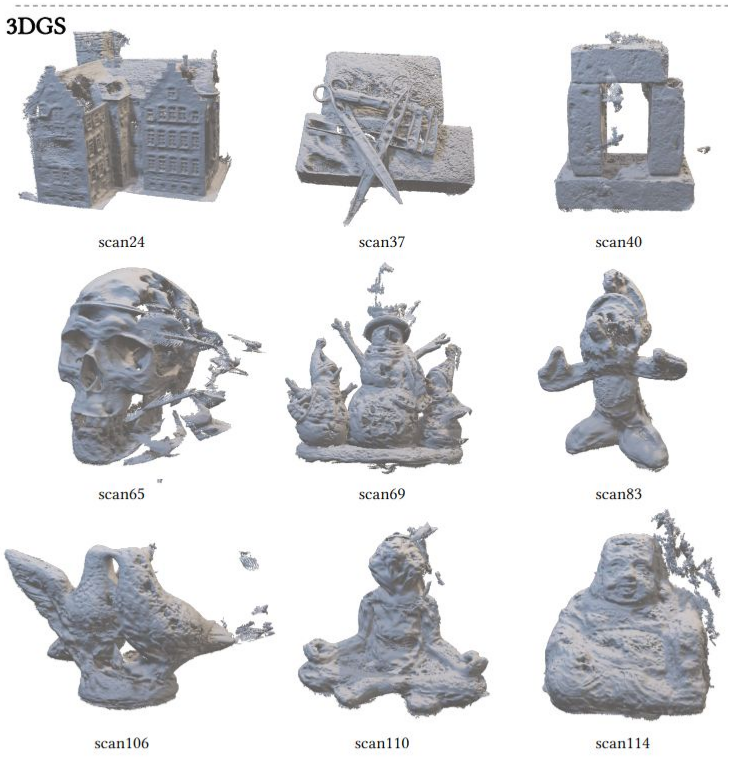


Fig. 10. Qualitative studies for the Tanks and Temples dataset [Knapitsch et al. 2017].

Experiments and Results: Depth maps 3DGS vs 2DGS



Conclusions

- The experiments gives positive results as compared to state of the art models.
- Lacks a quantitative experiment to show the insufficiency in modeling semi-transparent objects in comparison to other models particularly 3DGS.
- Mathematical model lacks justifications and explanations
- The model is implemented without discussing the specific choice of parameters which gives ambiguity as to how they managed to obtain them.
- The final product works but the math backing it is not well explained.
- It is recommended to re submit after revising the comments detailed above.

Archeologist Section

Esteban Wirth

Previous papers

Takes the initial model readapts it to have 2D Gaussians

Uses both the model and loss function of this paper as a basis.

Previous papers

Takes the concept of rendering over a surface with different ellipses.

Advances the field by giving a method of rendering surface with unknown geometry

Differentiable Surface Splatting for Point-based Geometry Processing

WANG YIFAN, ETH Zurich, Switzerland
FELICE SERENA, ETH Zurich, Switzerland
SHIHAO WU, ETH Zurich, Switzerland
CENGIZ ÖZTIRELI, Disney Research Zurich, Switzerland
OLGA SORKINE-HORNUNG, ETH Zurich, Switzerland

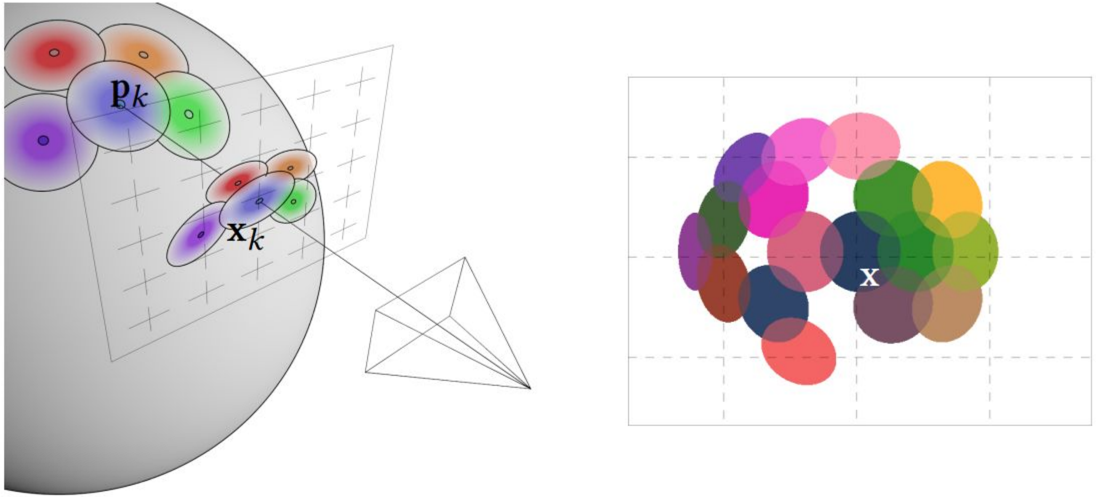


Fig. 2. Illustration of forward splatting using EWA [Zwicker et al. 2001]. A point in space $\mathbf{p}_k$ is rendered as an anisotropic ellipse centered at the projection point $\mathbf{x}_k$. The final pixel value $\mathbb{I}_\mathbf{x}$ at a pixel $\mathbf{x}$ in the image (shown on the right) is the normalized sum of all such ellipses overlapping at $\mathbf{x}$.

Next paper

Vidu4D: Single Generated Video to High-Fidelity 4D Reconstruction with Dynamic Gaussian Surfels

Yikai Wang[*1], Xinzhou Wang[*1,2,3], Zilong Chen[1,2], Zhengyi Wang[1,2], Fuchun Sun[1], Jun Zhu[†1,2]
[1]Department of Computer Science and Technology, BNRist Center, Tsinghua University
[2]ShengShu [3]College of Electronic and Information Engineering, Tongji University
yikaiw@outlook.com, wangxinzhou@tongji.edu.cn, dcszj@tsinghua.edu.cn

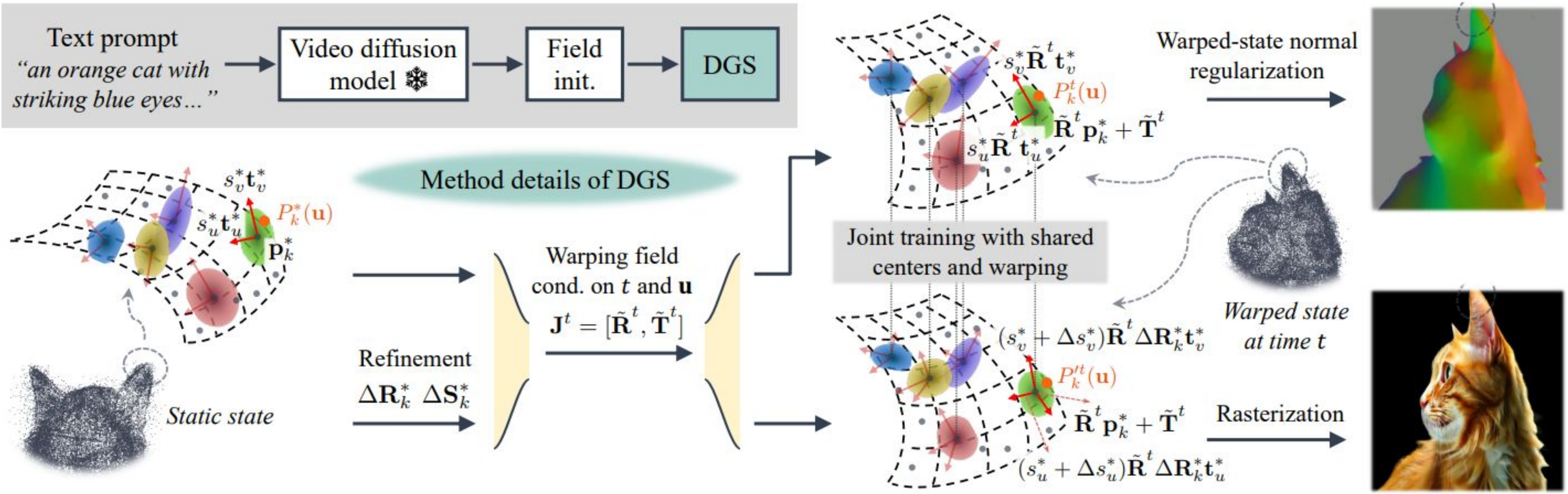
Use of 2DGS in Vidu4D: Rasterization

$$P_k^*(\mathbf{u}) = \mathbf{p}_k^* + s_u^*\mathbf{t}_u^* u + s_v^*\mathbf{t}_v^* v = [\mathbf{R}_k^*\mathbf{S}_k^* \quad \mathbf{p}_k^*]\,(u, v, 1, 1)^\top$$

$$\mathbf{c}(\bar{\mathbf{x}}) = \sum_k \mathbf{c}_k\, \alpha_k\, \mathcal{G}_k\big(\mathbf{u}(\bar{\mathbf{x}})\big) \prod_{j=1}^{k-1} \big(1 - \alpha_j\, \mathcal{G}_j\big(\mathbf{u}(\bar{\mathbf{x}})\big)\big)$$

$$\mathcal{L}_n = \sum_{k=1}^{K} \omega_k(1 - \mathbf{n}_k^\top \mathbf{N}^t), \quad \mathbf{N}^t(x, y) = \frac{\nabla_x \mathbf{p}^t \times \nabla_y \mathbf{p}^t}{|\nabla_x \mathbf{p}^t \times \nabla_y \mathbf{p}^t|}$$

Use of 2DGS in Vidu4D: Summary



Hacker Section

Leonardo Mendonça

Gaussian Geometry

Paper:

$$P(u, v) = \mathbf{p}_k + s_u \mathbf{t}_u u + s_v \mathbf{t}_v v = \mathbf{H}(u, v, 1, 1)^{\mathrm{T}} \tag{4}$$

$$\text{where}\, \mathbf{H} = \begin{bmatrix} s_u \mathbf{t}_u & s_v \mathbf{t}_v & \mathbf{0} & \mathbf{p}_k \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{RS} & \mathbf{p}_k \\ \mathbf{0} & 1 \end{bmatrix} \tag{5}$$

Code (gaussian_model.py):

```
def build_covariance_from_scaling_rotation(center, scaling, scaling_modifier, rotation):
    RS = build_scaling_rotation(torch.cat([scaling * scaling_modifier, torch.ones_like(scaling)], dim=-1), rotation).permute(0,2,1)
    trans = torch.zeros((center.shape[0], 4, 4), dtype=torch.float, device="cuda")
    trans[:,:3,:3] = RS
    trans[:, 3,:3] = center
    trans[:, 3, 3] = 1
    return trans
```

Gaussian Probability Distribution

Paper:

$$\mathcal{G}(\mathbf{u}) = \exp\left(-\frac{u^2 + v^2}{2}\right) \tag{6}$$

Code (gaussian_model.py):

```
        self.scaling_activation = torch.exp
        self.scaling_inverse_activation = torch.log

        self.covariance_activation = build_covariance_from_scaling_rotation
```

Gaussian Rendering

Paper:

$$u(\mathbf{x}) = \frac{\mathbf{h}_u^2\mathbf{h}_v^4 - \mathbf{h}_u^4\mathbf{h}_v^2}{\mathbf{h}_u^1\mathbf{h}_v^2 - \mathbf{h}_u^2\mathbf{h}_v^1} \qquad v(\mathbf{x}) = \frac{\mathbf{h}_u^4\mathbf{h}_v^1 - \mathbf{h}_u^1\mathbf{h}_v^4}{\mathbf{h}_u^1\mathbf{h}_v^2 - \mathbf{h}_u^2\mathbf{h}_v^1} \tag{10}$$

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1} \mathbf{c}_i\,\alpha_i\,\hat{\mathcal{G}}_i(\mathbf{u}(\mathbf{x}))\prod_{j=1}^{i-1}(1 - \alpha_j\,\hat{\mathcal{G}}_j(\mathbf{u}(\mathbf{x}))) \tag{12}$$

$$\hat{\mathcal{G}}(\mathbf{x}) = \max\left\{\mathcal{G}(\mathbf{u}(\mathbf{x})), \mathcal{G}(\frac{\mathbf{x}-\mathbf{c}}{\sigma})\right\} \tag{11}$$

Code (diff_surfel_rasterization/__init__.py):

```
num_rendered, color, depth, radii, geomBuffer, binningBuffer, imgBuffer = _C.rasterize_gaussians(*args)
```

- Rendering is done with a CUDA-optimized C++ script for faster GPU-based computing

Default Hyperparameters

```
class OptimizationParams(ParamGroup):  2 usages  hbb1
    def __init__(self, parser):  hbb1
        self.iterations = 30_000
        self.position_lr_init = 0.00016
        self.position_lr_final = 0.0000016
        self.position_lr_delay_mult = 0.01
        self.position_lr_max_steps = 30_000
        self.feature_lr = 0.0025
        self.opacity_lr = 0.05
        self.scaling_lr = 0.005
        self.rotation_lr = 0.001
        self.percent_dense = 0.01
        self.lambda_dssim = 0.2
        self.lambda_dist = 0.0
        self.lambda_normal = 0.05
        self.opacity_cull = 0.05

        self.densification_interval = 100
        self.opacity_reset_interval = 3000
        self.densify_from_iter = 500
        self.densify_until_iter = 15_000
        self.densify_grad_threshold = 0.0002
        super().__init__(parser, name: "Optimization Parameters")
```

Code: arguments__init.py

Densification and Adaptative Control of Gaussians

- The densification strategy is adapted from 3DGS [2], with mostly the same densification hyperparameters
- These values are given without justification in [2], and not all are mentioned explicitly in [1]
- Between the training iterations `densify_from_iter` (500) and `densify_until_iter` (15000), the model will periodically split or clone certain gaussians, depending on their scale
- After `opacity_reset_interval` (3000) epochs, the gaussians with opacity lower than `opacity_cull` (0,05) are removed, while the remaining ones have opacity reset to 0,01 (hardcoded)

Densification and Adaptative Control of Gaussians: Cloning

Code (gaussian_model.py):

```python
    def densify_and_clone(self, grads, grad_threshold, scene_extent):
        # Extract points that satisfy the gradient condition
        selected_pts_mask = torch.where(torch.norm(grads, dim=-1) >= grad_threshold, True, False)
        selected_pts_mask = torch.logical_and(selected_pts_mask,
                                              torch.max(self.get_scaling, dim=1).values <=
                                               self.percent_dense*scene_extent)

        new_xyz = self._xyz[selected_pts_mask]
        new_features_dc = self._features_dc[selected_pts_mask]
        new_features_rest = self._features_rest[selected_pts_mask]
        new_opacities = self._opacity[selected_pts_mask]
        new_scaling = self._scaling[selected_pts_mask]
        new_rotation = self._rotation[selected_pts_mask]

        self.densification_postfix(new_xyz, new_features_dc, new_features_rest, new_opacities,
         new_scaling, new_rotation)
```

Densification and Adaptative Control of Gaussians: Splitting

Code (gaussian_model.py):

```
def densify_and_split(self, grads, grad_threshold, scene_extent, N=2):  1 usage  hbb1
    n_init_points = self.get_xyz.shape[0]
    # Extract points that satisfy the gradient condition
    padded_grad = torch.zeros((n_init_points), device="cuda")
    padded_grad[:grads.shape[0]] = grads.squeeze()
    selected_pts_mask = torch.where(padded_grad >= grad_threshold, True, False)
    selected_pts_mask = torch.logical_and(selected_pts_mask,
                                          torch.max(self.get_scaling, dim=1).values > self.percent_dense*scene_extent)

    stds = self.get_scaling[selected_pts_mask].repeat(N,1)
    stds = torch.cat( tensors: [stds, 0 * torch.ones_like(stds[:,:1])], dim=-1)
    means = torch.zeros_like(stds)
    samples = torch.normal(mean=means, std=stds)
    rots = build_rotation(self._rotation[selected_pts_mask]).repeat(N,1,1)
    new_xyz = torch.bmm(rots, samples.unsqueeze(-1)).squeeze(-1) + self.get_xyz[selected_pts_mask].repeat(N, 1)
    new_scaling = self.scaling_inverse_activation(self.get_scaling[selected_pts_mask].repeat(N,1) / (0.8*N))
    new_rotation = self._rotation[selected_pts_mask].repeat(N,1)
    new_features_dc = self._features_dc[selected_pts_mask].repeat(N,1,1)
    new_features_rest = self._features_rest[selected_pts_mask].repeat(N,1,1)
    new_opacity = self._opacity[selected_pts_mask].repeat(N,1)
```

Densification and Adaptative Control of Gaussians: Opacity Reset

Code (gaussian_model.py):

```
    def reset_opacity(self):  1 usage  hbb1
        opacities_new = self.inverse_opacity_activation(torch.min(
         self.get_opacity, torch.ones_like(self.get_opacity)*0.01))
        optimizable_tensors = self.replace_tensor_to_optimizer(
         opacities_new,  name: "opacity")
        self._opacity = optimizable_tensors["opacity"]
```

Hidden Hyperparameters

- Maximum degree of spherical harmonics for anisotropic coloring in each gaussian
- Number of iterations to add new spherical harmonics (set to 1000, same as 3DGS)
- Number of iterations before applying depth distortion regularization (set to 3000, no explanation)
- Number of iterations before applying normal consistency regularization (set to 7000, no explanation)
- Minimum and maximum iteration where densification and pruning happen, as well as the gradient cutoff for densification and the opacity cutoff for pruning
- Opacity reset value (not only hidden, but hardcoded at 0,01)

Depth distortion regularization

- This regularization loss term seeks to minimize the distance between the depth of different gaussians intercepted by the same ray, therefore grouping the gaussians near the physical surface of the object
- The paper [2] suggests using weight parameter $\alpha=1000$ for bounded and $\alpha=100$ for unbounded scenes.
- However, in the authors' experiments, the value of α used changes from dataset to dataset. In the MipNeRF360 dataset, in particular, this regularization is not used at all
- In order to study the effects of depth distortion, we launched simulations of the MipNeRF360 Bonsai scene with resolution 390×260, with $\alpha=0$ (used in the tests) and $\alpha=1000$ (recommended for bounded scenes)

With depth distortion (*α=1000*)

Simulated

Ground truth

With depth distortion (α=*1000*)

Simulated

Ground truth

Without depth distortion (α=0)

Simulated

Ground truth

Without depth distortion ($\alpha=0$)

Simulated

Ground truth

Impact of resolution

- After training, we rendered the optimized scene both in the training resolution (390×260) and in a higher resolution of 1559×1039
- The quality of the reconstruction, as we shall see, is directly connected to the rendering resolution
- Since the results were very similar for the reconstructions with and without the depth distortion, we use here $\alpha=0$, the same value used by the authors in their evaluation of the MipNeRF360 dataset

Low resolution (390×260)

Simulated (α=0)

Ground truth

High resolution (1559×1039)

Simulated (α=0)

Ground truth

High resolution (1559×1039): Simulated, detailed view

PhD Student Section

Leonardo Mendonça

Project 1: 2D Half-Gaussian Splatting

- Attempt scene reconstruction replacing the 2D gaussians used in the paper with half-gaussians, following the distribution given below:

$$G(u,v) = \begin{cases} 2e^{-\frac{u^2+v^2}{2}}, \text{ if } v \geq 0 \\ 0, \text{ if } v < 0 \end{cases}$$

- This shape could be more suitable for expressing sharp edges, such as those found in many man-made objects
- Expected difficulty: Each gaussian has a non-differentiable line at *v=0*
- A smoothing function between the half-planes *v>0* and *v<0* will need to be defined (and perhaps trained) in order to allow for backpropagation

Project 1: 2D Half-Gaussian Splatting

- Many structures are characterized by sharp edges and straight angles
- In theory, the “sharp” half-gaussians could be used to represent such objects with fewer points than the full 2d gaussians

Project 2: Mixed 2-3D Gaussian Splatting

- The paper for 2DGS [1] reports difficulties in the reconstruction of translucent materials
- However, these materials can be accurately represented by volumetric (3D) gaussians, as seen in 3DGS [2] and EWA Splatting [3]
- Therefore, we propose a mixed approach, where 2D and 3D gaussians coexist and can be jointly optimized to accurately capture both opaque surfaces and translucent volumes

Project 2: Mixed 2-3D Gaussian Splatting

- Challenge: Every point in the initial point cloud should initialize to either a 2D or a 3D gaussian. How to decide the dimension of each gaussian beforehand?
- One also needs to consider whether each training hyperparameter (e.g. the densification threshold) should be the same for 2D and 3D
- Drawback: By mixing the gaussians, we lose 2DGS's ability to obtain surface normals for free
- As such, it is no longer possible to use normal consistency as a regularization strategy

Bibliography

1. Huang, B., Yu, Z., Chen, A., Geiger, A. and Gao, S., 2024, July. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH 2024 Conference Papers* (pp. 1-11).
2. Kerbl, B., Kopanas, G., Leimkühler, T. and Drettakis, G., 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.*, 42(4), pp.139-1.
3. Zwicker, M., Pfister, H., Van Baar, J. and Gross, M., 2002. EWA splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3), pp.223-238.