3DV 2024

3D Gaussian Splatting

Splatting in Practice

Bernhard Kerbl

3DGS in Practice – Overview

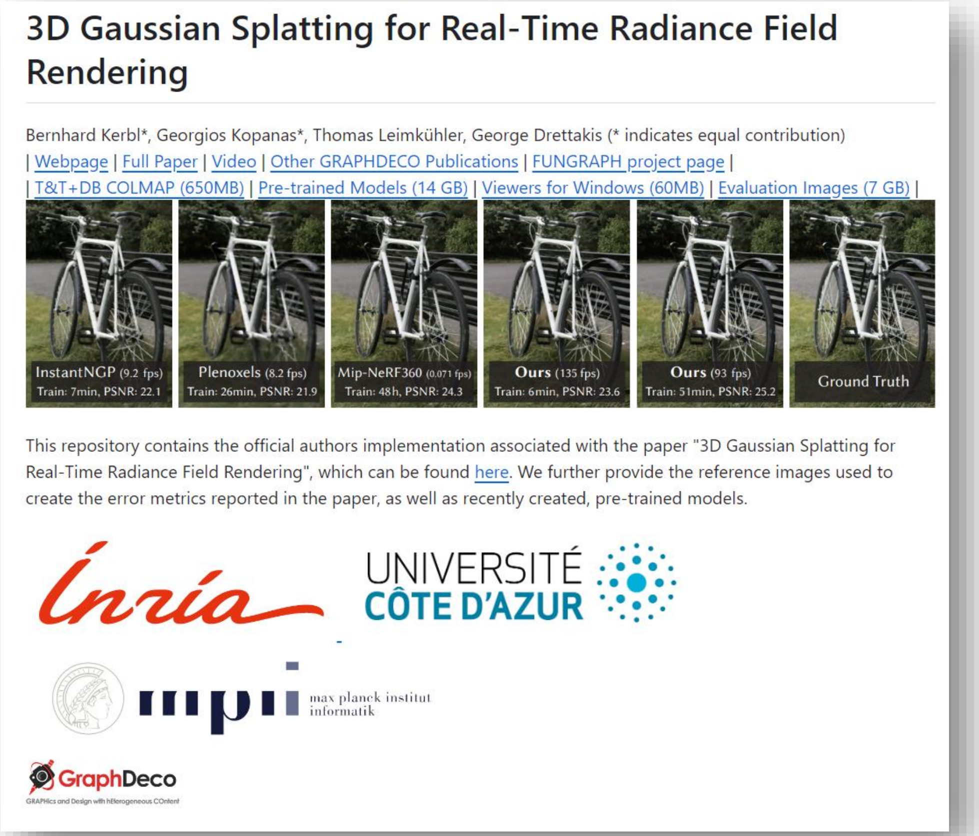
1. Running the GraphDeco code
2. 3DGS Everywhere Else
3. 3DGS Rendering with Graphics Pipelines
4. Reducing the Size of 3DGS Models

Your host for today

Current System (Laptop)

- Windows 11
- CUDA 11.7
- Conda 4.9.2
- Microsoft Visual Studio 2019

- Let's try the repo instructions →



https://github.com/graphdeco-inria/gaussian-splatting

Prologue to the Release

- We were not the first to release code for our own paper(!)
- https://github.com/wanmeihuali/taichi_3d_gaussian_splatting
- Prototype based on preprint, uses **Taichi Lang** for implementation
- Same method, entirely different design, clearly a clean-room feat

Running the GraphDeco Code

Starting from Scratch

3D Gaussian Splatting Ecosystem

- GraphDeco repository contains
  1. Input Datasets
  2. Pytorch scripts (Python)
  3. Submodules (C++/CUDA) – recursive!
     - Extensions for training
     - **SIBR** Viewers (optional)

SIBR_viewers @ 4ae964a	Bumped sibr viewers	5 months ago
arguments	Random background flag added	5 months ago
assets	Added LR example	8 months ago
gaussian_renderer	Typo in gaussian_renderer.__init__.py (#68)	8 months ago
lpipsPyTorch	Initial commit	9 months ago
scene	Bump rasterizer, faster text loading	6 months ago
submodules	Bump rasterizer, faster text loading	6 months ago
utils	Provide --data_device option to put data on CPU to save VR...	8 months ago
.gitignore	Initial commit	9 months ago
.gitmodules	Update .gitmodules to include .git for SIBR_viewers	7 months ago

- Run training: `python train.py -s <path to COLMAP dataset>`

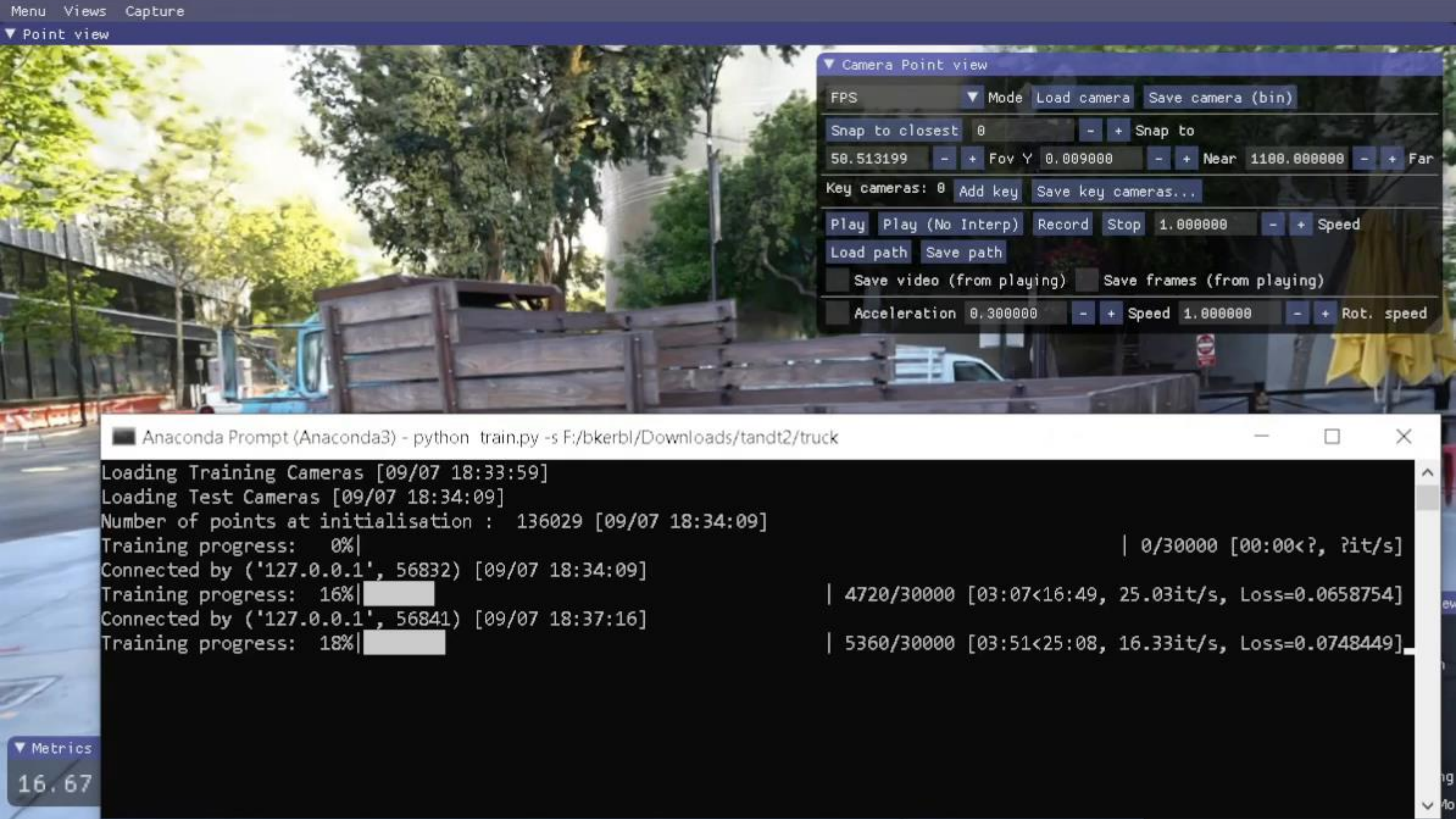
- Generates trained model in **custom** .ply format

(Remote) Training Client

- Thin client: `train.py` renders current state, transfers image
  - TCP/IP protocol, remote possible
  - Server/host can define IP/Port for connection

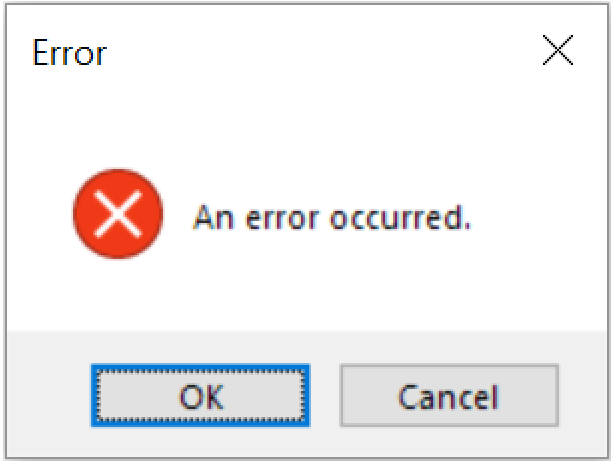
- Build source or use pre-built `SIBR_remoteGaussian_app.exe`

- Several scenarios possible, just run the executable if local


Menu Views Capture
Point view
Camera Point view
FPS
Mode
Load camera
Save camera (bin)
Snap to closest
0
Snap to
50.513199
Fov Y
0.009000
Near
1100.000000
Far
Key cameras: 0
Add key
Save key cameras...
Play
Play (No Interp)
Record
Stop
1.000000
Speed
Load path
Save path
Save video (from playing)
Save frames (from playing)
Acceleration
0.300000
Speed
1.000000
Rot. speed
Anaconda Prompt (Anaconda3) - python train.py -s F:/bkerbl/Downloads/tandt2/truck
Loading Training Cameras [09/07 18:33:59]
Loading Test Cameras [09/07 18:34:09]
Number of points at initialisation : 136029 [09/07 18:34:09]
Training progress: 0%| | 0/30000 [00:00<?, ?it/s]
Connected by ('127.0.0.1', 56832) [09/07 18:34:09]
Training progress: 16%| | 4720/30000 [03:07<16:49, 25.03it/s, Loss=0.0658754]
Connected by ('127.0.0.1', 56841) [09/07 18:37:16]
Training progress: 18%| | 5360/30000 [03:51<25:08, 16.33it/s, Loss=0.0748449]
Metrics
16.67

Unexpected Difficulties

- “cl.exe not found”
  - Try without the suggested `SET DISTUTILS_USE_SDK=1`, or put MSVC on the Path
- “Illegal memory access”
  - Appears to occur preferably on RTX 40xx or Ubuntu
- No multi-GPU support
  - We simply didn’t have multi-GPU workstations to develop/test on
- No direct batch training support
  - But possible manually, simply `backward` multiple times before `step`



Real-Time Gaussian Viewer

- Standalone renderer, uses CUDA + OpenGL Interop (if it can)
- Reads `.ply` files generated by `train.py`
- Several convenience features
  - Visualize Gaussians as ellipsoids
  - Crop scene to region of interest
  - Display ground truth image with each camera

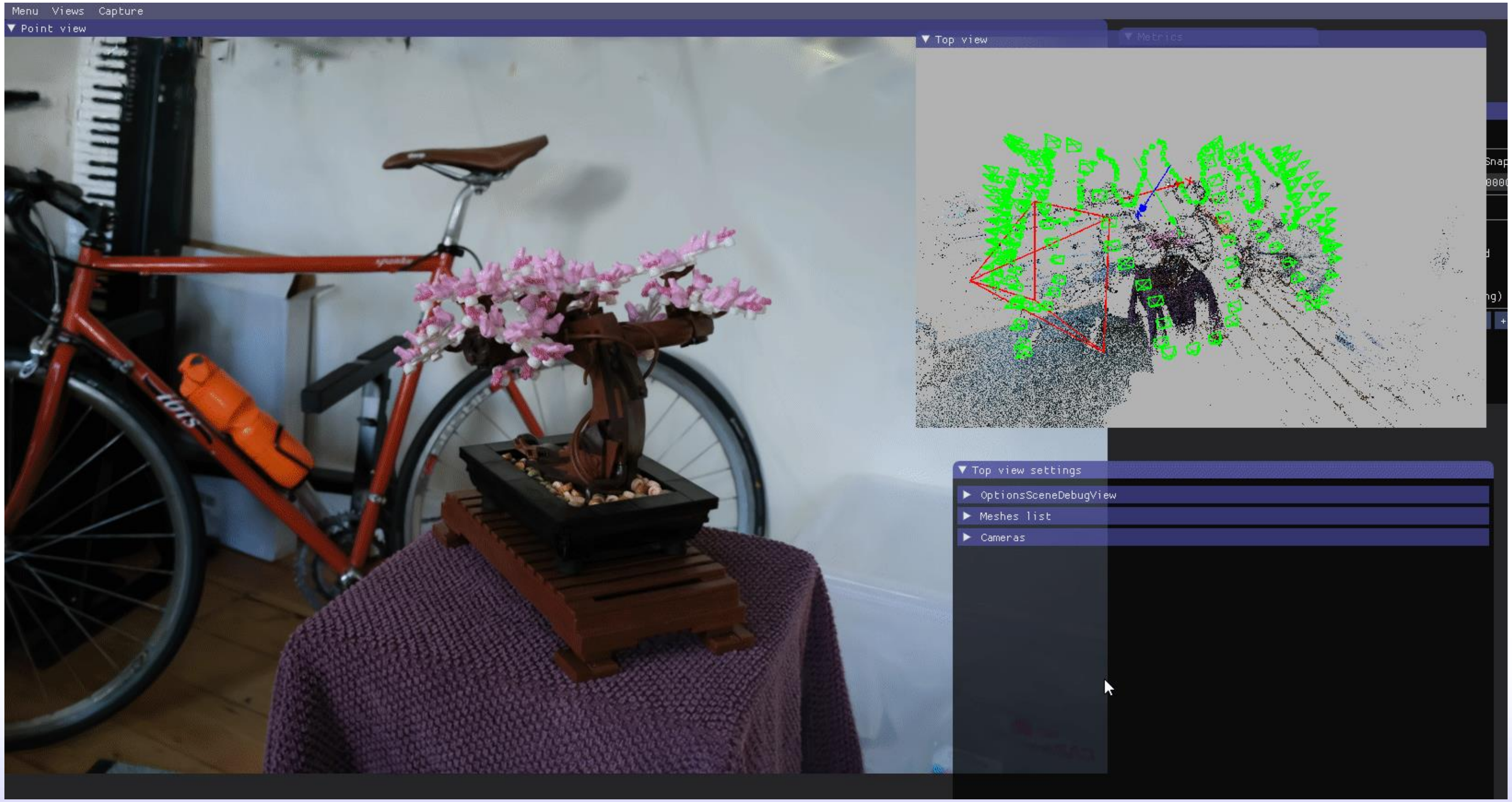

Camera Point view
FPS
Mode
Load camera
Save camera (bin)
Snap to closest
105
Snap to
47.136211
Fov Y
0.010000
Near
1000.000000
Key cameras: 0
Add key
Save key cameras...
Play
Play (No Interp)
Record
Stop
1.000000
Speed
Load path
Save path
Save video (from playing)
Save frames (from playing)
Acceleration
0.300000
Speed
1.000000
Rot. spe
3D Gaussians
Splats
Render Mode
1.000
Scaling Modifier
Metrics
31.47 (31.78 ms)

Rendering & Evaluation

- `render.py` for producing renderings of training / test set
- `metrics.py` for running relevant error metrics on renderings
- `full_eval.py` to replicate the paper's full quantitative evaluation
  - Includes training, rendering and metrics computation
  - At current state, takes about 6 – 8 hours on an RTX 3090

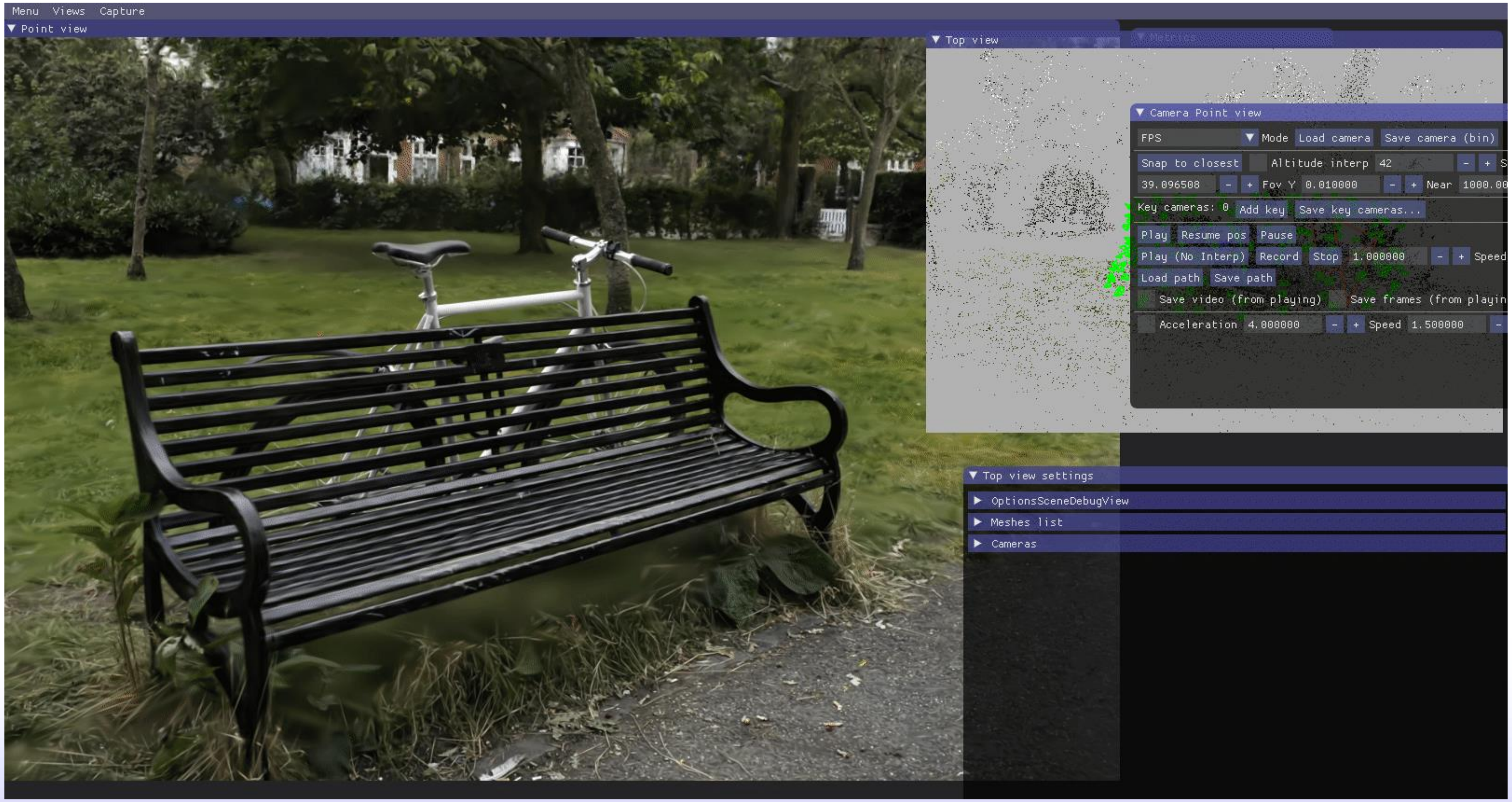
GraphDeco 3DGS Roadmap (Engineer)

- Improved TopView



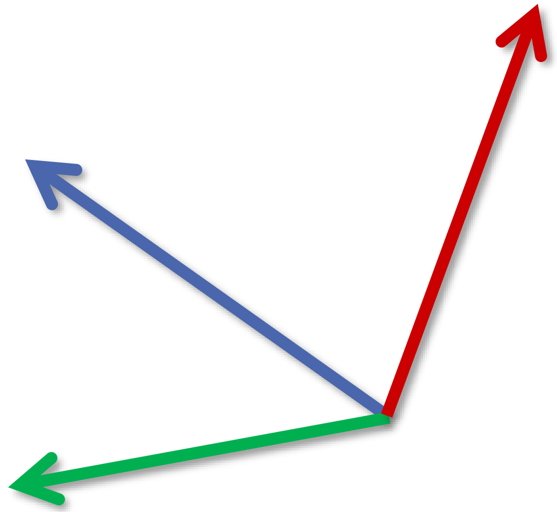
GraphDeco 3DGS Roadmap (Engineer)

- Improved TopView
- On-demand Images Overlay



GraphDeco 3DGS Roadmap (Engineer)

- Improved TopView
- On-demand
  Images Overlay
- Altitude Interpolation and Locking



GraphDeco 3DGS Roadmap (Engineer)

- Improved TopView

- On-demand
  Images Overlay

- Altitude Interpolation and Locking

- VR support via OpenXR

3DGS Everywhere Else

Services, Plug-Ins, Other Implementations

Keeping Track of the 3DGS Space

Janusch Patas
@janusch_patas

Jonathan Stephens
@jonstephens85

Radiance Fields
@RadianceFields

https://github.com/MrNeRF/awesome-3D-gaussian-splatting (curated list of publications)

https://radiancefields.com (news related to radiance fields)

3DGS Adaptations (not exhaustive!)

LUMA AI

(Thanks to Aras Pranckevičius)

https://gsplat.tech (Thanks to Jakub Červený)

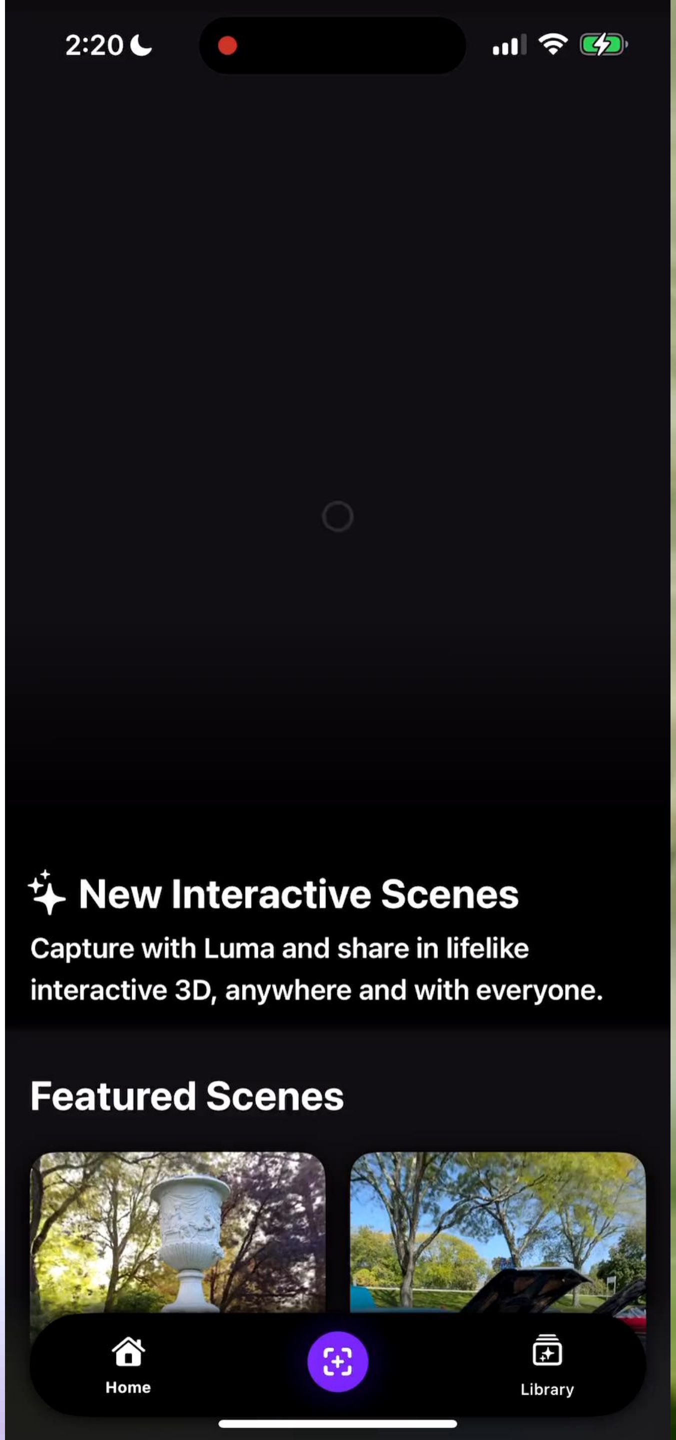
nerfstudio

(E.g., through Luma AI Plugin)

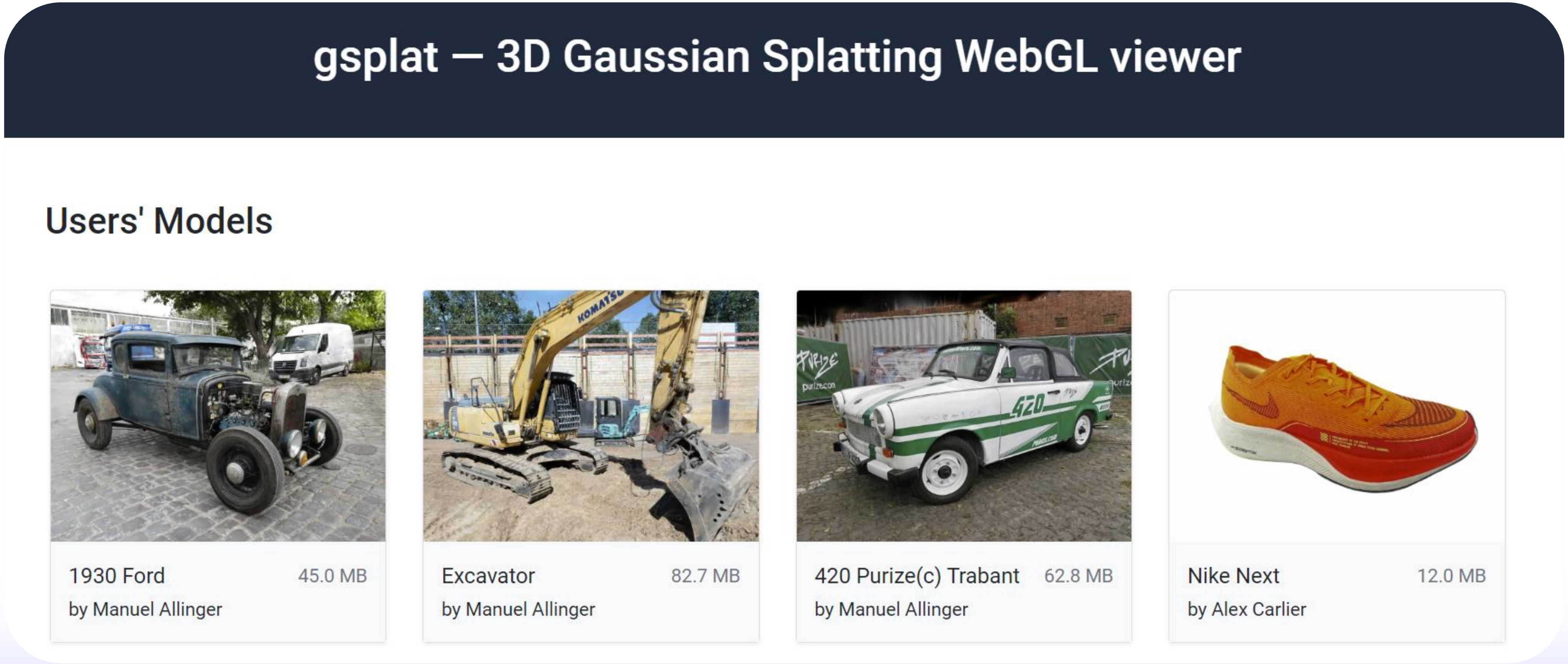
Luma

https://lumalabs.ai

- Interactive 3DGS Scenes
- App + Web Viewer
- Unreal Engine 5 plugin



gsplat.tech


gsplat — 3D Gaussian Splatting WebGL viewer
Users' Models
1930 Ford 45.0 MB
by Manuel Allinger
Excavator 82.7 MB
by Manuel Allinger
420 Purize(c) Trabant 62.8 MB
by Manuel Allinger
Nike Next 12.0 MB
by Alex Carlier

3DGS with Graphics Pipelines

"Why is everyone worrying about sorting?"

3D Gaussian Splatting & Alpha Blending

- Recap: Compositing Gaussians is a special variant of alpha blending

$$I(x) = \sum_i \alpha_i(x) c_i \prod_j^i 1 - a_j(x)\,, \qquad \alpha = oG(x), \qquad G(x) = e^{-0.5(x-\mu)^T {\Sigma'}^{-1}(x-\mu)}$$

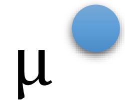
- Alpha blending is readily available in fixed-function triangle pipelines

- We can convert Gaussian Splatting to triangle rasterization

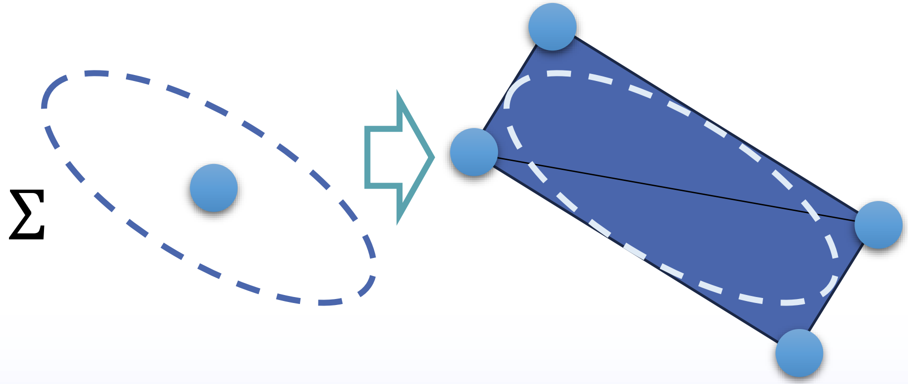
3D Gaussian Splatting & Alpha Blending

$$I(x) = \sum_i \alpha_i(x) c_i \prod_j^i 1 - a_j(x), \qquad \alpha = oG(x), \qquad G(x) = e^{-0.5(x-\mu)^T \Sigma'^{-1}(x-\mu)}$$

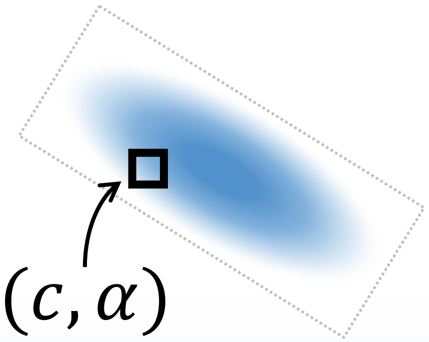
1. Vertex Shader



2. Geometry Shader



3. Fragment Shader



4. Blending

$\alpha_1 \square + (1 - \alpha_1)\alpha_2 \square$

3D Gaussian Splatting & Alpha Blending

$$I(x) = \sum_{i} \alpha_i(x) c_i \prod_{j}^{i} 1 - a_j(x), \qquad \alpha = oG(x), \qquad G(x) = e^{-0.5(x-\mu)^T \Sigma'^{-1}(x-\mu)}$$

$$\alpha_1 \square + (1-\alpha_1)\alpha_2 \square$$

```
<clear to background RGB, 0>
glBlendFunc(ONE_MINUS_DST_ALPHA, ONE)
```

3D Gaussian Splatting & Alpha Blending

RGB output of pixel shader pre-multiplied with alpha

$$\alpha_1 \square + (1-\alpha_1)\alpha_2 \square$$

```
<clear to background RGB, 0>
glBlendFunc(ONE_MINUS_DST_ALPHA, ONE)
```

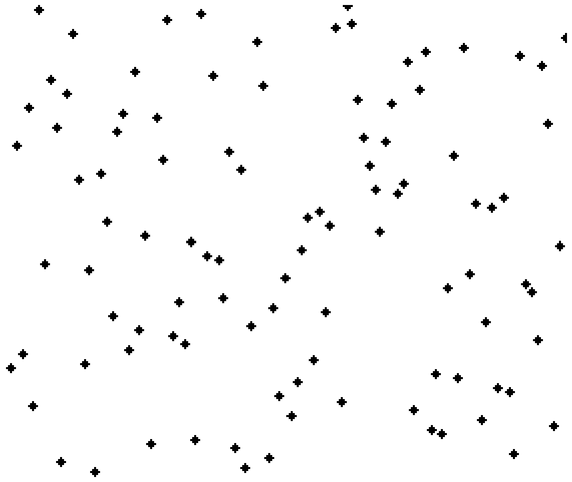
$$A_1 = 0, c_1 = (1-0)\alpha_1 \square + \blacksquare$$

$$A_2 = \alpha_1, c_2 = (1 - \alpha_1)\alpha_2 \square + \alpha_1 \square$$

$$1 - A_{n+1} = 1 - (\alpha_n(1-A_n) + A_n) = (1-\alpha_n)(1-A_n)$$ ☺

Sorting Gaussians

- Pipeline ensures "primitive order" of vertex indices
- But that order must be established first!
- Requires *sorting* of Gaussians for the current view
  - Millions of Gaussians: not hard, but also not trivial
  - Sort on **GPU**: fast, **requires compute shader support**
  - Sort on **CPU**: **slower** (adds index transfer), incremental or periodic?



Nuno Nogueira, Wikipedia, "Sorting Algorithm"

Reducing the Size of 3DGS

Taming the Gigabytes

Reducing the Size of 3DGS Scenes

- Is 3DGS a solution for fast, portable 3D viewing?
- Training speed ✅ Rendering speed ✅ Download speed ❌
- Generated `.ply` range from a few dozen MiB to more than one GiB
- Smaller than Plenoxels volumes, but <u>much</u> bigger than NeRF scenes

Analyzing the Storage Cost of a 3DGS Scene

- 59 x 4 bytes to represent a single Gaussian
- Millions of them!

Blog Posts by Aras Pranckevičius

Game Engine-oriented: reordering, texture encoding and palettes

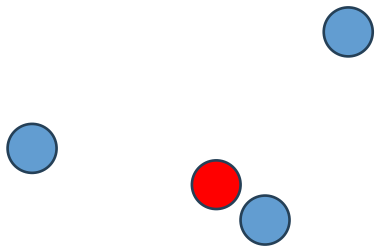
1. https://aras-p.info/blog/2023/09/13/Making-Gaussian-Splats-smaller/ (×12+)
2. https://aras-p.info/blog/2023/09/27/Making-Gaussian-Splats-more-smaller/

© Aras Pranckevičius

Compact3D (arXiv preprint)

- Consider Gaussians as a conglomerate of d-dimensional attributes

- Use k-means to learn a **quantized codebook** per attribute vector
    1. Start with k cluster means per attribute
    2. Store non-quantized attributes
    3. Quantize by snapping to closest d-dimensional mean
    4. Differentiably render 3D Gaussians
    5. Gradients for non-quantized attributes via straight-through estimator (STE)
    6. Update cluster positions, repeat



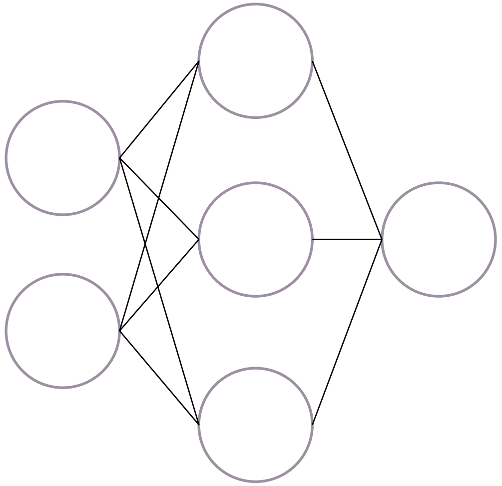
EAGLES (arXiv preprint)

1. Uses quantized latent integer vector $\mathbf{q} \in \mathbb{Z}^l$ for rotation, opacity, and SHs

2. MLPs to decode each $\mathbf{q}$ into attributes for rendering

3. Also employs STE to learn with quantization

4. Various training improvements
   - Progressive coarse-to-fine training
   - Tweak densification interval/threshold to maximize quality/Gaussian



LightGaussian (arXiv preprint)

1. Prune Gaussians: compute a significance score $\mathrm{GS_j} = \sum_{i=1}^{MHW} \mathbb{1}(\mathrm{G}(\boldsymbol{X}_j), r_i) \cdot \sigma_j \cdot \gamma(\boldsymbol{\Sigma}_j)$

opacity

Volume, avoid excessive focus on large background Gaussians

2. Avoid high storage usage of last SH band
   - Teacher-student distillation by optimizing $\mathcal{L}_{\mathrm{distill}} = \frac{1}{HW} \sum_{i=1}^{HW} \| \boldsymbol{C}_{\mathrm{teacher}}(r_i) - \boldsymbol{C}_{\mathrm{student}}(r_i) \|_2^2$
   - Use pseudo views for better coverage

3. Compress per-Gaussian attributes
   - Encode positions <u>losslessly</u> with G-PCC (octree)
   - Optimize codebooks via k-means, weight Gaussians with scores from 1.

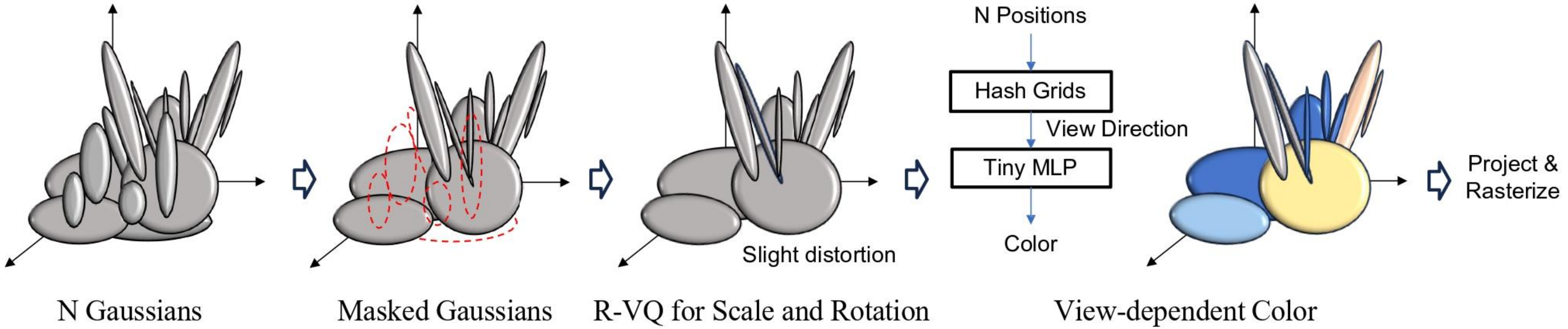
Compact 3D Gaussian Representation for Radiance Field (CVPR ‘24)

Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park

1. Learned masking parameter for binary masks via STE
2. Multiple stages of residual codebooks to fit Gaussian attributes
3. Hashgrids + MLP to represent view-dependent colors

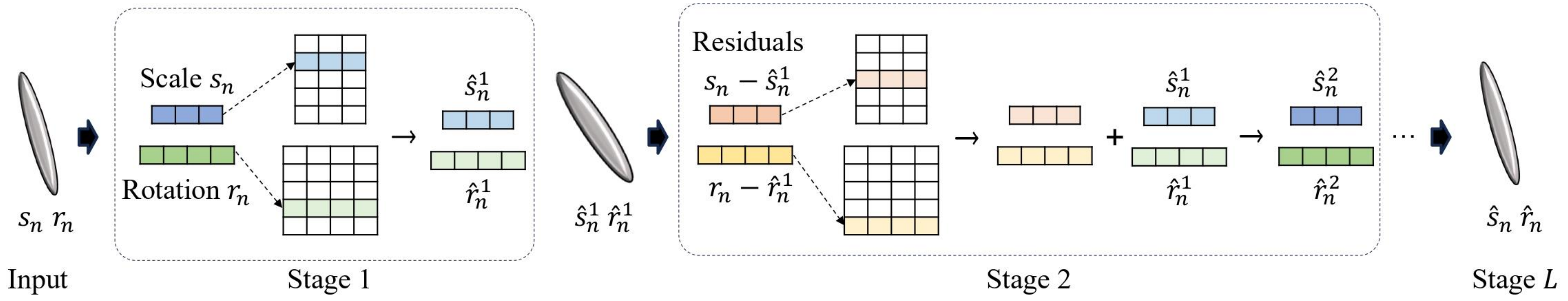
Compact 3D Gaussian Representation for Radiance Field (CVPR ‘24)

Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park



Compact 3D Gaussian Representation for Radiance Field (CVPR ‘24)

Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park

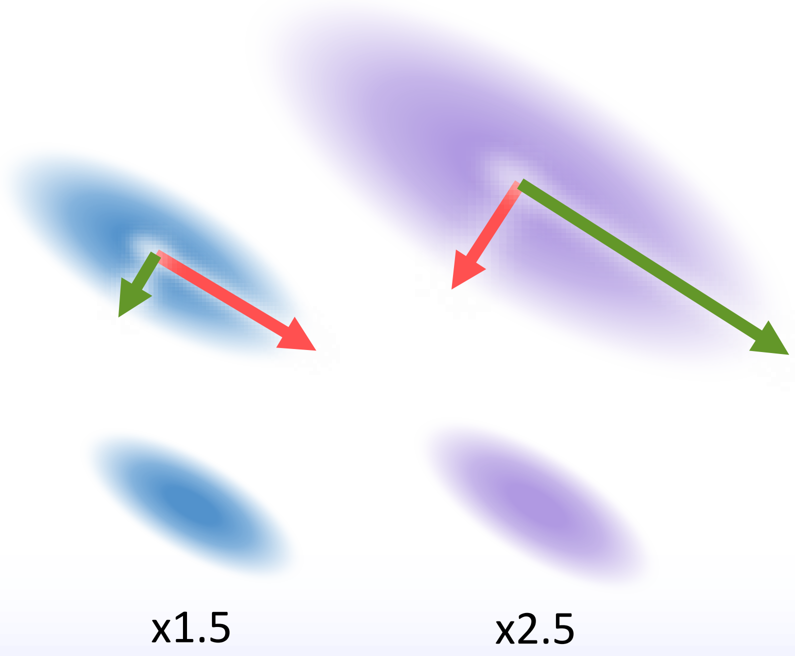


Compressed 3D Gaussian Splatting for Accelerated Novel View Synthesis (CVPR ‘24)

Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann

- Minimize Entropy and Treat Data
  - Addresses ambiguity of covariance representation
  - Factors out **scalar** scaling factor to minimize entropy
  - Sort by Morton order, run-length encode and deflate

- Graphics pipeline rendering for improved speed

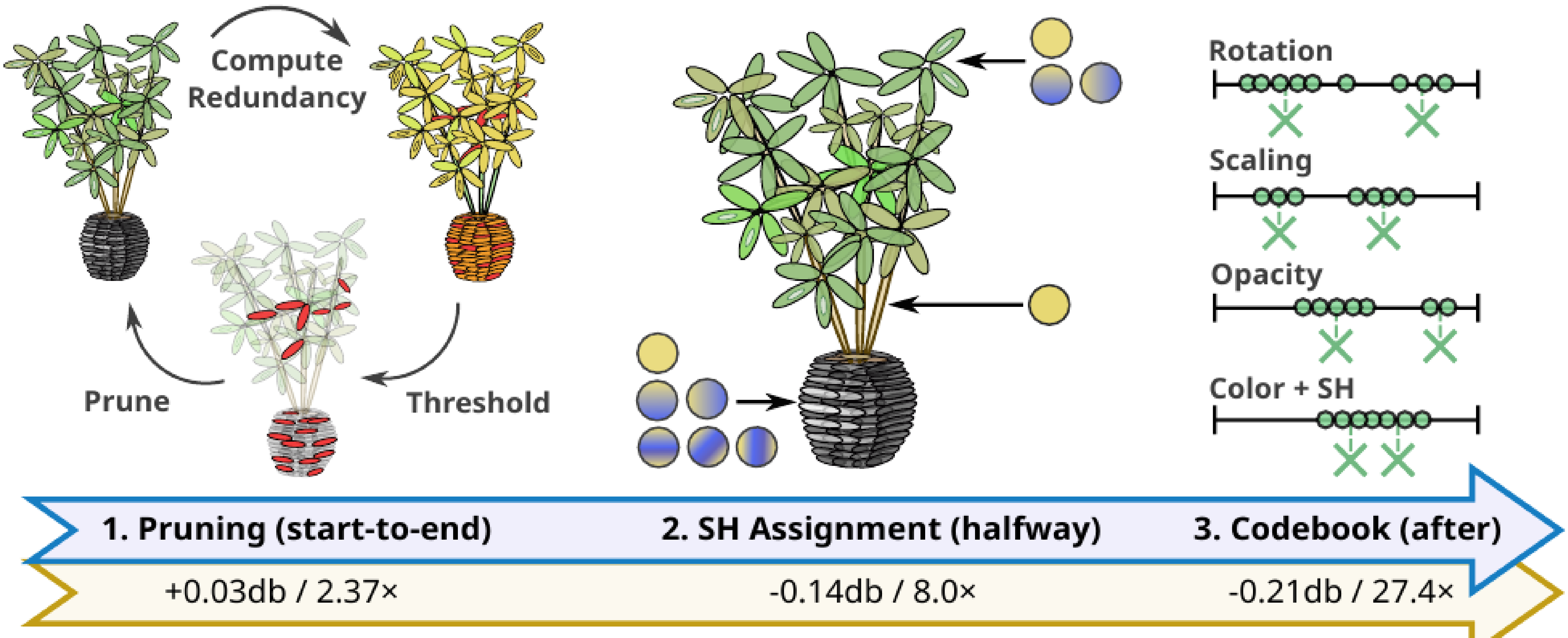



Compressed (Ours)
26.75 PSNR
46.6 MB
1.3 GB
27.18 PSNR
3D Gaussian Splatting

Reducing the Memory Footprint of 3D Gaussian Splatting (I3D '24)

1. Pruning based on coverage of 3D regions and discernible detail
2. Includes <u>variable</u> SH assignment and distillation
3. K-means cluster properties + Codebook + Quantization (16-bit)
4. Evaluation against concurrent work, <u>mobile prototype</u>

Revised Training for Compact 3DGS Scenes



Reducing the Memory Footprint of 3D Gaussian Splatting

submission n°1

Which Method to Pick? Synergies?

Dataset	Mip-NeRF360					Deep Blending				
Method/Metric	SSIM	PSNR	LPIPS	Train	Mem	SSIM	PSNR	LPIPS	Train	Mem
Ours	0.809	27.10	0.226	25m27s	29MB	0.902	29.63	0.249	22m4s	18MB
Low	0.811	27.22	0.224	25m22s	46MB	0.903	29.74	0.248	21m59s	35MB
EAGLES [15]	0.808	27.16	0.238	19m57s	68MB	0.910	29.91	0.245	17m24s	62MB
Compact3DGS [19]	0.798	27.08	0.247	33m6s	48MB	0.901	29.79	0.258	27m33s	43MB
Compact3D [24]	0.808	27.16	0.228	-	-	0.903	29.75	0.247	-	-

Dataset	Mip-NeRF360 No Hidden					Tanks&Temples				
Method/Metric	SSIM	PSNR	LPIPS	Train	Mem	SSIM	PSNR	LPIPS	Train	Mem
Ours	0.864	28.58	0.193	26m0s	27MB	0.840	23.57	0.188	14m0s	14MB
Low	0.866	28.73	0.190	25m52s	43MB	0.841	23.64	0.186	14m4s	21MB
EAGLES [15]	0.866	28.69	0.200	20m18s	67MB	0.835	23.41	0.200	9m48s	34MB
Compact3DGS [19]	0.856	28.60	0.209	33m1s	46MB	0.832	23.31	0.202	18m19s	39MB
Compact3D [24]	-	-	-	-	-	0.840	23.47	0.188	-	-
LightGaussian [13]	0.858	28.46	0.210	-	42MB	0.807	22.83	0.242	-	22MB

Looking for New Challenges

- Next 6 months: joining the **Human Sensing Lab** at Carnegie Mellon
- Looking for **faculty positions** for the time after
- Glad to hear about opportunities, preferably in Europe!
- Or just chat about 3DGS and potential follow-ups ☺

Questions?

Immer her damit