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
Current System (Laptop)

- Let’s try the repo instructions ➔ https://github.com/graphdeco-inria/gaussian-splatting
- Windows 11
- CUDA 11.7
- Conda 4.9.2
- Microsoft Visual Studio 2019
Prologue to the Release

• We were not the first to release code for our own paper(!)

• [https://github.com/wanmeihuali/taichi_3d_gaussian_splatting](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)

• 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
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%
Connected by ('127.0.0.1', 56832) [09/07 18:34:09]
Training progress: 16%
Connected by ('127.0.0.1', 56841) [09/07 18:37:16]
Training progress: 18%

| 0/30000 [00:00<?, ?it/s] |
| 4720/30000 [03:07<16:49, 25.03it/s, Loss=0.0658754] |
| 5360/30000 [03:51<25:08, 16.33it/s, Loss=0.0748449] |
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
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

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
by Manuel Allinger

45.0 MB

Excavator
by Manuel Allinger

82.7 MB

420 Purize(c) Trabant
by Manuel Allinger

62.8 MB

Nike Next
by Alex Carlier

12.0 MB
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

\[ I(x) = \sum_{i} \alpha_i(x)c_i \prod_{j} 1 - a_j(x), \quad \alpha = oG(x), \quad G(x) = e^{-0.5(x-\mu)^T\Sigma^{-1}(x-\mu)} \]

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

- We can convert Gaussian Splatting to triangle rasterization
3D Gaussian Splatting & Alpha Blending

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

1. Vertex Shader
2. Geometry Shader
3. Fragment Shader
4. Blending
3D Gaussian Splatting & Alpha Blending

\[ I(x) = \sum_i \alpha_i(x)c_i \prod_j 1 - \alpha_j(x), \quad \alpha = oG(x), \quad G(x) = e^{-0.5(x-\mu)^T\Sigma^{-1}(x-\mu)} \]

\[ \alpha_1 + (1 - \alpha_1)\alpha_2 \]

\(<\text{clear to background RGB, 0}>\)

\(\text{glBlendFunc(ONE\_MINUS\_DST\_ALPHA, ONE)}\)
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>

\[ \text{glBlendFunc}(\text{ONE\_MINUS\_DST\_ALPHA}, \text{ONE}) \]

\[ A_1 = 0, c_1 = (1 - 0) \alpha_1 \square + \square \]

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

\[ 1 - A_{n+1} = (1 - \alpha_n)(1 - A_n) = 1 - (\alpha_n(1 - A_n) + A_n) \] ☺
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?

https://creativecommons.org/licenses/by-sa/2.5/
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 much 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


© Aras Pranckevičius
Overview of Research on 3DGS Compression

• Some of the following papers are preprints

• Depending on the targeted venues, reviewing may be double-blind

• We thus omit author names for papers pending a final accept
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
   \[ G_{S_j} = \sum_{i=1}^{MHW} \mathbb{1}(G(X_j), r_i) \cdot \sigma_j \cdot \gamma(\Sigma_j) \]
   
   2. Avoid high storage usage of last SH band
      - Teacher-student distillation by optimizing \( \mathcal{L}_{\text{distill}} = \frac{1}{HW} \sum_{i=1}^{HW} \| C_{\text{teacher}}(r_i) - C_{\text{student}}(r_i) \|_2^2 \)
      - Use pseudo views for better coverage

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

Volume, avoid excessive focus on large background Gaussians

opacity
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

N Gaussians → Masked Gaussians → Slight distortion → R-VQ for Scale and Rotation → View-dependent Color

Hash Grids → View Direction → Tiny MLP → Color → Project & Rasterize

18.03.2024
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

x1.5  x2.5
Reducing the Memory Footprint of 3D Gaussian Splatting (I3D ’24)

1. Pruning based on coverage of 3D regions and discernible detail

2. Includes variable SH assignment and distillation

3. K-means cluster properties + Codebook + Quantization (16-bit)

4. Evaluation against concurrent work, mobile prototype
Revised Training for Compact 3DGS Scenes

1. Pruning (start-to-end)  
   +0.03db / 2.37×

2. SH Assignment (halfway)  
   -0.14db / 8.0×

3. Codebook (after)  
   -0.21db / 27.4×
Reducing the Memory Footprint of 3D Gaussian Splatting

submission n°1
## Which Method to Pick? Synergies?

<table>
<thead>
<tr>
<th>Dataset Method/Metric</th>
<th>Mip-NeRF360</th>
<th>Deep Blending</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SSIM</td>
<td>PSNR</td>
</tr>
<tr>
<td>Ours</td>
<td>0.809</td>
<td>27.10</td>
</tr>
<tr>
<td>Low</td>
<td>0.811</td>
<td>27.22</td>
</tr>
<tr>
<td>EAGLES [15]</td>
<td>0.808</td>
<td>27.16</td>
</tr>
<tr>
<td>Compact3DGSS [19]</td>
<td>0.798</td>
<td>27.08</td>
</tr>
<tr>
<td>Compact3D [24]</td>
<td>0.808</td>
<td>27.16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset Method/Metric</th>
<th>Mip-NeRF360 No Hidden</th>
<th>Tanks&amp;Temple</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SSIM</td>
<td>PSNR</td>
</tr>
<tr>
<td>Ours</td>
<td>0.864</td>
<td>28.58</td>
</tr>
<tr>
<td>Low</td>
<td>0.866</td>
<td>28.73</td>
</tr>
<tr>
<td>EAGLES [15]</td>
<td>0.866</td>
<td>28.69</td>
</tr>
<tr>
<td>Compact3DGSS [19]</td>
<td>0.856</td>
<td>28.60</td>
</tr>
<tr>
<td>Compact3D [24]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LightGaussian [13]</td>
<td>0.858</td>
<td>28.46</td>
</tr>
</tbody>
</table>
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