CS231A
Computer Vision: From 3D Reconstruction to Recognition

Gaussian Splatting for Novel View Synthesis
The problem of novel view synthesis

Inputs: sparsely sampled images of scene

Outputs: new views of same scene
(rendered by our method)

Rendering (Graphics): Given 3D Scene + Camera parameters, yield images
Inverse Graphics: Given Images, Infer Camera Poses & 3D Scene!

Images → 3D Scene → Camera Poses

How to get camera poses?

Can assume we know the camera poses.
Differentiable Rendering

Scene Representation → Renderer → Rendered Images

Differentiable Rendering

Differentiable Rendering

Scene Representation $\rightarrow$ Differentiable Renderer $\rightarrow$ Optimization via SGD $\rightarrow$ Rendered Images $\rightarrow$ GT Images

Differentiable Rendering

Given an observable variable (pixel colors), we will build a differentiable forward model that we then use to estimate unobserved (latent) variables (geometry, appearance)!

Ways to Render

Surface rendering

Volume rendering
Volume rendering equation

\[ I(D) = I_0 T(0) + \int_0^D c(s) \rho(s) T(s) \, ds \]

**pixel color at coordinates D**

**radiance density**

**transparency**

\[ T(s) = \exp \left( - \int_s^D \rho(t) \, dt \right) \]

**Silvio Savarese & Jeannette Bohg**  **Lecture 17**
Represent a scene as a continuous 5D function

\[(x, y, z, \theta, \phi) \rightarrow F_\Omega \rightarrow (r, g, b, \sigma)\]

- **Spatial location**
- **Viewing direction**
- **Output color**
- **Output density**

- Fully-connected neural network
  - 9 layers
  - 256 channels

No need to instantiate Volume representation
Generate views with traditional volume rendering

Generate views with traditional volume rendering

Rendering model for ray $r(t) = o + td$:

$$C \approx \sum_{i=1}^{N} T_i \alpha_i c_i$$

$t$ = point along ray
$C$ = Color of Pixel
$c$ = color of point

How much light is blocked earlier along ray:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

Transparency

How much light is contributed by ray segment $i$:

$$\alpha_i = 1 - e^{-\sigma_i \delta t_i}$$

Function of segment length $\delta t_i$ and volume density $\sigma$

Optimize with gradient descent on rendering loss

$$\min_{\Omega} \sum_{i} \| \text{render}^{(i)}(F_{\Omega}) - I^{(i)}_{gt} \|^2$$

Training network to reproduce all input views of the scene

The problem of novel view synthesis

Inputs: sparsely sampled images of scene

Outputs: new views of same scene (rendered by our method)

Vision-Only Navigation

MPC controller

Real World or Simulator

Camera Images

Predicated Images

Estimator

NeRF

State

Trajectory Optimizer

Control Actions
Plenoxels: Radiance Fields …
[Yu et al. 2022]
Direct Voxel Grid Optimization
[Sun et al. 2021]

InstantNGP: Instant Neural …
[Müller et al. 2022]

PointNeRF: Point-based Neural …
[Xu et al. 2022]

Efficient Geometry-aware 3D…
[Chan et al. 2022]
TensorRF: Tensor Radiance Fields
[Chen & Xu et al. 2022]
Hybrid Multi-Scale Grid, HashMap, Neural Field

- Instant-NGP: 9.0
- MipNeRF360: 0.07

FPS

Slide adopted from 6.5980 – ML for Inverse Graphics – Vincent Sitzmann
3D Gaussian Splatting for Real-Time Radiance Field Rendering

BERNHARD KERBL*, Inria, Université Côte d’Azur, France
GEORGIOS KOPANAS*, Inria, Université Côte d’Azur, France
THOMAS LEIMKÜHLER, Max-Planck-Institut für Informatik, Germany
GEORGE DRETTAKIS, Inria, Université Côte d’Azur, France

Fig. 1. Our method achieves real-time rendering of radiance fields with quality that equals the previous method with the best quality [Barron et al. 2022], while only requiring optimization times competitive with the fastest previous methods [Fridovich-Keil and Yu et al. 2022; Müller et al. 2022]. Key to this performance is a novel 3D Gaussian scene representation coupled with a real-time differentiable renderer, which offers significant speedup to both scene optimization and novel view synthesis. Note that for comparable training times to InstantNGP [Müller et al. 2022], we achieve similar quality to theirs; while this is the maximum quality they reach, by training for 51min we achieve state-of-the-art quality, even slightly better than Mip-NeRF360 [Barron et al. 2022].
Slide adopted from 6.5980 – ML for Inverse Graphics – Vincent Sitzmann
Neural Radiance Field: Parameterize Radiance Field densely, at every point in space

\[(x,y,z,\theta,\phi) \rightarrow \begin{array}{c} F \Theta \end{array} \rightarrow (RGB\sigma)\]
Key Idea: Parameterize Radiance Field sparsely, only where density is nonzero

3D Gaussian Blobs floating in Space

Mean

Key Idea: Parameterize Radiance Field sparsely, only where density is nonzero

3D Gaussian Blobs floating in Space

Slide adopted from 6.5980 – ML for Inverse Graphics – Vincent Sitzmann
Key Idea: Parameterize Radiance Field sparsely, only where density is nonzero

3D Gaussian Blobs floating in Space

\[ \sigma = 0 \]
Key Idea: Parameterize Radiance Field sparsely, only where density is nonzero

3D Gaussian Blobs floating in Space

\[ \sigma = 0 \]
\[ \sigma = 0.5 \]
\[ RGB = \]

Key Idea: Parameterize Radiance Field sparsely, only where density is nonzero

3D Gaussian Blobs floating in Space

$\sigma = 0$

$\sigma = 0.5$

$\sigma = 1$

$RGB =$

Slide adopted from 6.5980 – ML for Inverse Graphics – Vincent Sitzmann
Anisotropic Volumetric 3D Gaussians

Final Rendering

3D Gaussian Visualization
How to Render?

3D Gaussian Blobs floating in Space

$\sigma = 0$

$\sigma = 0.5$

$\sigma = 1$

$RGB = \star$

$RGB = \star$

$RGB = \star$

$RGB = \star$

Slide adopted from 6.8980 – ML for Inverse Graphics – Vincent Sitzmann
Same Volume Rendering Integral!

“Far Plane”

“Near Plane”

Camera

Slide adopted from 6.5980 – ML for Inverse Graphics – Vincent Sitzmann
Same Volume Rendering Integral!

Still sampling lots of empty space...

Slide adopted from 6.980 – ML for Inverse Graphics – Vincent Sitzmann
Same Volume Rendering Integral!

Stupid: we already know where the density will be nonzero!

Slide adopted from 6.980 – ML for Inverse Graphics – Vincent Sitzmann
Gaussians are closed under affine transforms, integration

\[ g_v(x - p) = \frac{1}{2\pi|V|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-p)^T V^{-1}(x-p)} \]

3D Covariance!
Gaussians are closed under affine transforms, integration

\[ G_V(x - p) = \frac{1}{2\pi|V|^{1/2}} e^{-\frac{1}{2} (x-p)^T V^{-1} (x-p)} \]

Affine mapping \( \Phi = Mx + p \) of coordinates (such as cam2world matrix!):

\[ G_V(\Phi^{-1}(u) - p) = \frac{1}{|M^{-1}|} G_{MVMT}(u - \Phi(p)) \]
Gaussians are closed under affine transforms, integration

\[ G_V(x - p) = \frac{1}{2\pi|V|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-p)^T V^{-1}(x-p)} \]

3D Covariance!

Affine mapping \( \Phi = Mx + p \) of coordinates (such as cam2world matrix!):

\[ G_V(\Phi^{-1}(u) - p) = \frac{1}{|M^{-1}|} G_{MVM^T}(u - \Phi(p)) \]

Integrate along axis:

\[ \int_{\mathbb{R}^3} G^3_V(x - p) \, dx_2 = G^2_V(\hat{x} - \hat{p}) \]

\[ V = \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix} \leftrightarrow \begin{pmatrix} a & b \\ b & d \end{pmatrix} = \hat{V} \]
Throwback: The Kalman Filter Algorithm

1: \textbf{Algorithm Kalman filter}($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2: \hspace{1em} $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3: \hspace{1em} $\bar{\Sigma}_t = A_t \Sigma_{t-1} A^T_t + R_t$
4: \hspace{1em} $K_t = \bar{\Sigma}_t C^T_t (C_t \bar{\Sigma}_t C^T_t + Q_t)^{-1}$
5: \hspace{1em} $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
6: \hspace{1em} $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
7: \hspace{1em} return $\mu_t, \Sigma_t$

---

1: \textbf{Algorithm Bayes filter}($\text{bel}(x_{t-1}), u_t, z_t$):
2: \hspace{1em} for all $x_t$ do
3: \hspace{2em} $\text{bel}(x_t) = \int p(x_t | u_t, x_{t-1}) \text{bel}(x_{t-1}) \, dx$ \textbf{Predict Step}
4: \hspace{2em} $\text{bel}(x_t) = \eta p(z_t | x_t) \text{bel}(x_t)$ \textbf{Update Step}
5: \hspace{1em} endfor
6: \hspace{1em} return $\text{bel}(x_t)$
Instead: Rasterization
Instead: Rasterization

"Pixel Frustum"
“Cull” Gaussians with less than 99% confidence relative to view frustum

Slide adopted from 6.980 – ML for Inverse Graphics – Vincent Sitzmann
Step 1: Transform Gaussians into Camera Coordinates

Cam2world is affine mapping $\phi(x) = Wx + p$:

$$G_{V_k'}(\varphi^{-1}(u) - t_k) = \frac{1}{|W^{-1}|} G_{V_k'}(u - u_k) = r'_k(u)$$
Step 1: Transform Gaussians into Camera Coordinates

Cam2world is affine mapping $\phi(x) = Wx + p$:

$$G_{V_k'}(\varphi^{-1}(u) - t_k) = \frac{1}{|W^{-1}|}G_{V_k'}(u - u_k) = r_k'(u)$$

Projection $m(u)$ is not an affine mapping :/

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = m(u) = \begin{pmatrix} u_0 / u_2 \\ u_1 / u_2 \\ \| (u_0, u_1, u_2)^T \| \end{pmatrix}$$

$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = m^{-1}(x) = \begin{pmatrix} x_0 / l \cdot x_2 \\ x_1 / l \cdot x_2 \\ 1 / l \cdot x_2 \end{pmatrix}$$
Step 1: Transform Gaussians into Camera Coordinates

Cam2world is affine mapping \( \phi(x) = Wx + p \):

\[
G_{V_k'}(\phi^{-1}(u) - t_k) = \frac{1}{|W^{-1}|} G_{V_k'}(u - u_k) = r'_k(u)
\]

Projection \( m(u) \) is not an affine mapping :/

\[
\begin{pmatrix}
  x_0 \\
  x_1 \\
  x_2 
\end{pmatrix} = m(u) = \begin{pmatrix}
  u_0 / u_2 \\
  u_1 / u_2 \\
  ||(u_0, u_1, u_2)^T|| 
\end{pmatrix}
\]

\[
\begin{pmatrix}
  u_0 \\
  u_1 \\
  u_2 
\end{pmatrix} = m^{-1}(x) = \begin{pmatrix}
  x_0 / l \cdot x_2 \\
  x_1 / l \cdot x_2 \\
  1 / l \cdot x_2 
\end{pmatrix},
\]

But can approximate with first-order Taylor Expansion as:

\[
m_{u_k}(u) = x_k + J_{u_k} \cdot (u - u_k)
\]

\[
J_{u_k} = \frac{\partial m}{\partial u}(u_k)
\]

Slide adopted from 6.8980 – ML for Inverse Graphics – Vincent Sitzmann
Propagating a Gaussian through a Linear Model
Propagating a Gaussian through a Non-Linear Model
Linearizing the Non-Linear Model
Throwback: The Extended Kalman Filter Algorithm

Algorithm Extended Kalman filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

1. $\bar{\mu}_t = g(u_t, \mu_{t-1})$
2. $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
3. $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
4. $\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$
5. $\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$
6. return $\mu_t, \Sigma_t$

<table>
<thead>
<tr>
<th>state prediction (Line 2)</th>
<th>Kalman filter</th>
<th>EKF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_t \mu_{t-1} + B_t u_t$</td>
<td>$g(u_t, \mu_{t-1})$</td>
<td></td>
</tr>
<tr>
<td>$C_t \bar{\mu}_t$</td>
<td>$h(\bar{\mu}_t)$</td>
<td></td>
</tr>
</tbody>
</table>
Step 1: Transform Gaussians into Camera Coordinates

Cam2world is affine mapping $\phi(x) = \mathbf{W}x + \mathbf{p}$:

$$G_{V_k'}(\varphi^{-1}(\mathbf{u}) - t_k) = \frac{1}{|\mathbf{W}^{-1}|} G_{V_k'}(\mathbf{u} - \mathbf{u}_k) = r_k'(\mathbf{u})$$

Projection $m(\mathbf{u})$ is not an affine mapping :/

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = m(\mathbf{u}) = \begin{pmatrix} u_0 / u_2 \\ u_1 / u_2 \\ \|u_0, u_1, u_2\|^T \end{pmatrix}$$

$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} = m^{-1}(\mathbf{x}) = \begin{pmatrix} x_0 / l \cdot x_2 \\ x_1 / l \cdot x_2 \\ 1 / l \cdot x_2 \end{pmatrix},$$

But can approximate with first-order Taylor Expansion as:

$$m_{u_k}(\mathbf{u}) = x_k + \mathbf{J}_{u_k} \cdot (\mathbf{u} - \mathbf{u}_k) \quad \mathbf{J}_{u_k} = \frac{\partial m}{\partial \mathbf{u}}(\mathbf{u}_k)$$

Step 1: Transform Gaussians into Camera Coordinates

But can approximate with first-order Taylor Expansion as:

$$m_{u_k}(u) = x_k + J_{u_k} \cdot (u - u_k)$$

$$J_{u_k} = \frac{\partial m}{\partial u}(u_k)$$
Step 1: Transform Gaussians into Camera Coordinates

But can approximate with first-order Taylor Expansion as:

\[ m_{uk}(u) = x_k + J_{uk} \cdot (u - u_k) \quad J_{uk} = \frac{\partial m}{\partial u}(u_k) \]

Projected, 2D Gaussians are then:

\[ \frac{1}{|W^{-1}| |J^{-1}|} \mathcal{G}_{V_k}(x - x_k) \]

\[ V_k \quad = \quad J V_k' J^T \]
\[ = \quad J W V_k'' W^T J^T. \]
Step 1: Transform Gaussians into Camera Coordinates

But can approximate with first-order Taylor Expansion as:

\[ m_{u_k}(u) = x_k + J_{u_k} \cdot (u - u_k) \quad J_{u_k} = \frac{\partial m}{\partial u}(u_k) \]

Projected, 2D Gaussians are then:

\[ \frac{1}{|\mathbf{W}^{-1}||\mathbf{J}^{-1}|} \mathbf{G}_V \mathbf{k}(\mathbf{x} - \mathbf{x}_k) \]

\[ \mathbf{V}_k = \mathbf{J} \mathbf{V}_k' \mathbf{J}^T \]
\[ = \mathbf{J} \mathbf{W} \mathbf{V}_k'' \mathbf{W}^T \mathbf{J}^T. \]

Finally, can integrate along rays:

\[ q_k(\hat{x}) = \int_{\mathbb{R}} \frac{1}{|\mathbf{J}^{-1}||\mathbf{W}^{-1}|} \mathbf{G}_V \mathbf{k}(\hat{x} - \hat{x}_k, x_2 - x_{k2}) \, dx_2 \]
\[ = \frac{1}{|\mathbf{J}^{-1}||\mathbf{W}^{-1}|} \mathbf{G}_V \mathbf{k}(\hat{x} - \hat{x}_k). \]
Can compute volume rendering integral without ever sampling a single 3D point in space!

\[ C = \sum_{i \in N} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \]  

where \( c_i \) is the color of each point and \( \alpha_i \) is given by evaluating a 2D Gaussian with covariance \( \Sigma \) [Yifan et al. 2019] multiplied with a learned per-point opacity.
Projected 3D Gaussian makes 2D Gaussian!

Any problems for inverse graphics, though...?

\[ C = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \]

where \( c_i \) is the color of each point and \( \alpha_i \) is given by evaluating a 2D Gaussian with covariance \( \Sigma \) [Yifan et al. 2019] multiplied with a learned per-point opacity.
Problem: Local minima...
Fix 1: Start from SFM point cloud.
Fix 2: Heuristic *pruning* and *spawning* operations

Slide adopted from 6.980 – ML for Inverse Graphics – Vincent Sitzmann
Timelapse of the Optimization
(NeRF-Synthetic Dataset)
All interactive sessions are recorded at 1080p with an A6000
CS231A
Computer Vision: From 3D Reconstruction to Recognition

Next lecture:
Guest Lecture by Adam Harley on Visual Tracking