Towards Efficient Neural Rendering

Sungjoo Yoo

Computing Memory Architecture Lab.
Computer Science and Engineering
Seoul National University
Agenda

• Introduction to neural rendering
• Problem: high computation cost
• Efficient model #1: voxel grid
• Efficient model #2: mesh
• Summary and prospect
Photo-Realistic Experience on Mobile Devices

• Entertainment (games, sports, movies, music performance, ...)
• Productivity (virtual office, meeting room, ...)

[https://www.apple.com/apple-vision-pro/]

[Spatial VR meeting app]
How to Obtain Virtual Objects/Scenes from Images?

• Recently (in 2020), neural rendering (aka neural radiance field, NeRF) was proposed
• It enables us to train a neural network \( (F_\theta) \), with captured images, to generate novel view images

\[ \text{NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis} \]
MLP for Radiance Field (Color and Density)

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis
Volume Rendering: Alpha Composition

Transmittance = ray survival prob. on 1 ... i-1

\[
\hat{C}(r) = \sum_{i=1}^{N} T_i (1 - \exp(-\sigma_i \delta_i)) c_i, \quad \text{where } T_i = \exp \left( -\sum_{j=1}^{i-1} \sigma_j \delta_j \right)
\]

\( r(t) = o + td \)

probability of surface

\( w_i \), weight of sample point \( i \)
3D Models Trained from Images
Agenda

• Introduction to neural rendering

• **Problem: high computation cost**

• Efficient model #1: voxel grid

• Efficient model #2: mesh

• Summary and prospect
High Compute Cost of NeRF

• Assume a VR headset with 1440x1600 pixels, 90Hz
  • At each pixel, we run 8-layer MLP of 256 hidden dimension on 256 sample points
  • Total # multiplications = 1440x1600 x 90 x 256 x 256 x 256 x 7 = 2.4 x 10^{16} = 24 Peta multiplications/second
  • ~ 1000X more compute than 50 TOPS (of future mobile NPU) is needed for real time

• Methods for compute cost reduction
  • Voxel grid to exploit pre-computation
  • Mesh to exploit existing graphics pipeline
  • Light field network to exploit existing CNN acceleration
Agenda

• Introduction to neural rendering
• Problem: high computation cost
• Efficient method #1: voxel grid
• Efficient method #2: mesh
• Summary and prospect
PlenOctree (ICCV 21)
Train NeRF, Build a Voxel Grid and Fine-tune It

- NeRF-SH, \( f(x) \)
- Color, \( c(d; k) \)

\[
f(x) = (k, \sigma) \quad \text{where} \quad k = (k^m_\ell)^m_{-\ell \leq m \leq \ell}
\]

\[
c(d; k) = S \left( \sum_{\ell=0}^{\ell_{\max}} \sum_{m=-\ell}^{\ell} k^m_\ell Y^m_\ell(d) \right)
\]

\( S: \text{sigmoid} \)

(a) NeRF-SH Training

(b) Conversion to PlenOctree
>1000X Faster than NeRF

• Low compute cost/sample point
  • Voxel grid contains pre-computed values
  • Rendering requires simple computation on the pre-computed values

• Small # sample points/ray
  • Sparsity to skip compute on empty space
  • Visibility to skip compute on unseen space, e.g., interior of object

<table>
<thead>
<tr>
<th>Tanks and Temples Dataset</th>
<th>best</th>
<th>second-best</th>
<th>FPS ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSNR ↑</td>
<td>SSIM ↑</td>
<td>LPIPS ↓</td>
<td></td>
</tr>
<tr>
<td>NeRF (original)</td>
<td>25.78</td>
<td>0.864</td>
<td>0.198</td>
</tr>
<tr>
<td>NeRF</td>
<td>27.94</td>
<td>0.904</td>
<td>0.168</td>
</tr>
<tr>
<td>SRN</td>
<td>24.10</td>
<td>0.847</td>
<td>0.251</td>
</tr>
<tr>
<td>Neural Volumes</td>
<td>23.70</td>
<td>0.834</td>
<td>0.260</td>
</tr>
<tr>
<td>NSVF</td>
<td>28.40</td>
<td>0.900</td>
<td>0.153</td>
</tr>
<tr>
<td>NeRF-SH</td>
<td>27.82</td>
<td>0.902</td>
<td>0.167</td>
</tr>
<tr>
<td>PlenOctree from NeRF-SH</td>
<td>27.34</td>
<td>0.897</td>
<td>0.170</td>
</tr>
<tr>
<td>PlenOctree after fine-tuning</td>
<td>27.99</td>
<td>0.917</td>
<td>0.131</td>
</tr>
</tbody>
</table>
Training Voxel Grid, From Scratch Without MLP, is Much Faster!

(a) NeRF-SH Training:~(x, y, z)→→σ→→Density→k
(θ, φ)→→k₁, k₂, k₃, k₄→→Σ→→Color
Spherical Harmonics

(b) Conversion to PlenOctree: Dense Samples → NeRF-SH → PlenOctree

NeRF-SH Training

C
Fine-tuning
∥c − ĝ∥²

Plenoxel: Radiance Fields without Neural Networks

NeRF

1 min
5 min
10 min

Training Image

a) Sparse Voxel Grid

b) Trilinear Interpolation

Σ
Spherical Harmonics

Ray Distance

Predicted Color

minimize \( L_{recon} + \lambda L_{TV} \)

Training Time (minutes)

0 10 20 30 40 50 60

100x Faster Convergence

PSNR

20 25 30 35
Agenda

• Introduction to neural rendering
• Problem: high computation cost
• Efficient method #1: voxel grid
• Efficient method #2: mesh
• Summary and prospect
How to Exploit Existing Fast Rendering Graphics Pipeline on GPU?

- Traditional rendering pipeline from mesh to pixel
- Directly training a mesh model with captured images
  - DIB-R++, MobileNeRF, ...
- Training NeRF and build a mesh from it (via marching cube)
  - Neural Duplex, Re-Rend, ...

[https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_on_the_web/Basic_theory]
Differentiable Renderer (DIB-R)

- CNN is trained to give mesh, texture (albedo) and light parameters (spherical harmonics approximation coefficients of environment map)
How to Learn Vertex Coordinates via SGD?

- Exploit the relationship btw pixel color and vertex coordinates \((x_1, y_1)\)

- Given \((x_1, y_1), (x_2, y_2), (x_3, y_3)\)

\[
x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 \\
y = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3
\]

\[
\begin{pmatrix}
\lambda_1 \\
\lambda_2 \\
\lambda_3
\end{pmatrix} = \frac{1}{2A} \begin{pmatrix}
x_2 y_3 - x_3 y_2 & y_2 - y_3 & x_3 - x_2 \\
x_3 y_1 - x_1 y_3 & y_3 - y_1 & x_1 - x_3 \\
x_1 y_2 - x_2 y_1 & y_1 - y_2 & x_2 - x_1
\end{pmatrix} \begin{pmatrix} 1 \\ x \\ y \end{pmatrix}
\]

- Color \(I_i\)

\[
I_i = w_0 u_0 + w_1 u_1 + w_2 u_2
\]

\[
w_k = \Omega_k(\vec{v}_0, \vec{v}_1, \vec{v}_2, \vec{p}_i), \quad k = 0, 1, 2.
\]
Differentiable Rasterization

• Baycentric interpolation of vertex attributes, e.g., color

\[ I_i = w_0 u_0 + w_1 u_1 + w_2 u_2 \]
\[ w_k = \Omega_k(\vec{v}_0, \vec{v}_1, \vec{v}_2, \vec{p}_i), \quad k = 0, 1, 2. \]

• Vertex coordinates, \( v \)'s and attributes (color), \( u \)'s can be learned via SGD

\[ \frac{\partial I_i}{\partial u_k} = w_k, \quad \frac{\partial I_i}{\partial \vec{v}_k} = \sum_{m=0}^{2} \frac{\partial I_i}{\partial w_m} \frac{\partial \Omega_m}{\partial \vec{v}_k}, \]

\[ \frac{\partial L}{\partial u_k} = \sum_{i=1}^{N} \frac{\partial L}{\partial I_i} \frac{\partial I_i}{\partial u_k}, \quad \frac{\partial L}{\partial \vec{v}_k} = \sum_{i=1}^{N} \frac{\partial L}{\partial I_i} \frac{\partial I_i}{\partial \vec{v}_k}. \]
MobileNeRF Directly Learns Mesh and Texture

- Exploit the existing graphics rendering pipeline
  - Mesh + texture + viewpoint $\rightarrow$ rasterization (feature) $\rightarrow$ shader (MLP) for color
- $\sim$50 frames/sec on iPhone Xs

(a) Triangle mesh  (b) Texture image (features and opacity)  (c) Feature image  (d) Final output
Training MobileNeRF

• Start with an initial grid mesh, optimize for vertex positions as well as feature.opacity/color MLPs

• Given a ray
  • Interpolation of texture feature on ray-mesh intersections
  • Interpolated feature $\rightarrow$ color on sample point $\rightarrow$ pixel color via alpha composition
MobileNeRF Runs 50 Frames/Sec on iPhoneXS
Given a Trained NeRF Model, Build a Mesh and Exploit Graphics Pipeline: Re-ReND Case

• Mesh (w/ texture) building via Marching Cube method
• At ray-triangle intersection, cheap color computation with texture (dot products of UVW and $\beta$)
• Up to ~1000 frames per second on NVIDIA GPU 3090
Agenda

• Introduction to neural rendering

• Problem: high computation cost

• Efficient method #1: voxel grid

• Efficient method #2: mesh

• Summary and prospect
Summary and Prospect

• >1000 neural rendering papers (arxiv) since the NeRF paper in 2020

• Drastic (>1000X) improvements of compute efficiency in only three years
  • Voxel grid to exploit pre-computation
  • Mesh to exploit existing graphics pipeline
  • Light field network to exploit CNN acceleration

• Prospect: Efficient models for real-time immersive photo-realism
  • Relighting, dynamic scene, large scale and large language model (LLM)
Towards Relighting:
Inverse Rendering on Voxel Grid (TensoIR)

- Can render images on new lighting
  - By separately learning material and shape
  - w.r.t. radiance (color) in NeRF

- Very slow rendering due to high compute cost in rendering eqn
  - Visibility
  - Indirect illumination
Dynamic Scene (aka. Free Viewpoint Video)
Large-Scale NeRF: Block NeRF

• Consists of small block-level NeRF models
• e.g., city center in San Francisco
Block-NeRF: Scalable Large Scene Neural View Synthesis
Multi-Modal 3D Model
e.g., Text Input based Object Localization

[LERF: Language Embedded Radiance Fields]
Summary and Prospect

• >1000 neural rendering papers since the NeRF paper in 2020
• Drastic (>1000X) improvements of compute efficiency in only three years
  • Voxel grid to exploit pre-computation
  • Mesh to exploit existing graphics pipeline
  • Light field network to exploit CNN acceleration
• Prospect: Efficient models for real-time immersive photo-realism
  • Relighting, dynamic scene, large scale and large language model (LLM)
  • >1000X more efficient models and implementations are needed
    • Compute cost, model size, training speed, …
Thank You!
Copyright Notice

This presentation in this publication was presented as a tinyML® Asia Technical Forum. The content reflects the opinion of the author(s) and their respective companies. The inclusion of presentations in this publication does not constitute an endorsement by tinyML Foundation or the sponsors.

There is no copyright protection claimed by this publication. However, each presentation is the work of the authors and their respective companies and may contain copyrighted material. As such, it is strongly encouraged that any use reflect proper acknowledgement to the appropriate source. Any questions regarding the use of any materials presented should be directed to the author(s) or their companies.

tinyML is a registered trademark of the tinyML Foundation.

www.tinyml.org