Instant Neural Graphics Primitives with a Multiresolution Hash Encoding

Thomas Müller  Alex Evans  Christoph Schied  Alexander Keller
NeRF

- NeRF Pros: simple representation, differentiable rendering model
- NeRF Cons: dumb brute force, insanely slow
- How can we improve the speed of volumetric rendering?
- smaller MLPs
  KiloNeRF: break up space into 163 or 323 voxels, each with its own set of (small) MLP weights

- direct voxel lookups
  Plenoxels: $512^3$ voxel grid with density and spherical harmonics

- Acorn: adaptive feature-grid with a lightweight MLP to decode
Instant Neural Graphics Primitives with a Multiresolution Hash Encoding

Thomas Müller  Alex Evans  Christoph Schied  Alexander Keller
The approach

1. For a given input coordinate $x$, we find the surrounding voxels at $L$ resolution levels and assign indices to their corners by hashing their integer coordinates

$$h(x) = \left( \sum_{i=1}^{d} x_i \pi_i \right) \mod T,$$

2. For all resulting corner indices, we look up the corresponding $F$-dimensional feature vectors from the hash tables

3. Linearly interpolate them according to the relative position of $x$ within the respective $l$-th voxel.

4. Concatenate + auxiliary inputs (the encoded view, etc.)

5. MLP
The approach

1. For a given input coordinate $x$, we find the surrounding voxels at $L$ resolution levels and assign indices to their corners by hashing their integer coordinates

   \[ h(x) = \left( \sum_{i=1}^{d} x_i \pi_i \right) \mod T, \]

2. For all resulting corner indices, we look up the corresponding $F$-dimensional feature vectors from the hash tables

3. Linearly interpolate them according to the relative position of $x$ within the respective $l$-th voxel.

4. Concatenate + auxiliary inputs (the encoded view, etc.)

5. MLP
The approach

1. For a given input coordinate $x$, we find the surrounding voxels at $L$ resolution levels and assign indices to their corners by hashing their integer coordinates.

2. For all resulting corner indices, we look up the corresponding $F$-dimensional feature vectors from the hash tables.

3. Linearly interpolate them according to the relative position of $x$ within the respective $l$-th voxel.

4. Concatenate + auxiliary inputs (the encoded view, etc.)

5. MLP
The approach

1. For a given input coordinate $x$, we find the surrounding voxels at $L$ resolution levels and assign indices to their corners by hashing their integer coordinates.

2. For all resulting corner indices, we look up the corresponding $F$-dimensional feature vectors from the hash tables.

3. Linearly interpolate them according to the relative position of $x$ within the respective $l$-th voxel.

4. Concatenate + auxiliary inputs (the encoded view, etc.)

5. MLP
The approach

1. For a given input coordinate $x$, we find the surrounding voxels at $L$ resolution levels and assign indices to their corners by hashing their integer coordinates.

2. For all resulting corner indices, we look up the corresponding $F$-dimensional feature vectors from the hash tables.

3. Linearly interpolate them according to the relative position of $x$ within the respective $l$-th voxel.

4. Concatenate + auxiliary inputs (the encoded view, etc.)

5. MLP
The approach

1. For a given input coordinate $x$, we find the surrounding voxels at $L$ resolution levels and assign indices to their corners by hashing their integer coordinates

2. For all resulting corner indices, we look up the corresponding $F$-dimensional feature vectors from the hash tables

3. Linearly interpolate them according to the relative position of $x$ within the respective $l$-th voxel.

4. Concatenate + auxiliary inputs (the encoded view, etc.)

5. MLP
Experiment results - reconstruction quality
Experiment results

| Hash (ours) | NGLOD | Hash (ours) | Frequency | Frequency | Hash (ours) | NGLOD | Hash (ours) |
|-------------|-------|-------------|-----------|-----------|-------------|-------|-------------|
| ![image](image1.png) | ![image](image2.png) | ![image](image3.png) | ![image](image4.png) | ![image](image5.png) | ![image](image6.png) | ![image](image7.png) | ![image](image8.png) |
| 17.8M (params) | 12.2M | 90.1k | 90.1k | 12.2M | 12.6M |
| 1:43 (mm:ss) | 1:06 | 3:18 | 5:27 | 1:46 | 1:38 |
| 0.9761 (IoU) | 0.9811 | 0.6509 | 0.9824 | 0.9998 | 0.9998 |

| ![image](image9.png) | ![image](image10.png) | ![image](image11.png) | ![image](image12.png) | ![image](image13.png) | ![image](image14.png) | ![image](image15.png) | ![image](image16.png) |
| 8.8M (params) | 12.2M | 90.1k | 90.1k | 12.2M | 18.6M |
| 1:24 (mm:ss) | 1:11 | 3:30 | 3:04 | 0:58 | 1:37 |
| 0.9906 (IoU) | 0.9862 | 0.7389 | 0.2325 | 0.9646 | 0.9723 |
Test error over training time for varying hash table size $T$

- **Gigapixel image**
- **SDF**
- **NeRF**

Test error over training time for fixed values of feature dimensionality $F$

- **Gigapixel image: Tokyo**
- **Signed Distance Function: Cow**
- **Neural Radiance Field: LEGO**
Experiment results - runtime
Where does the speedup come from?

- factor of 10 from tiny-cuda-cnn - optimised CUDA kernels
- factor of 10~100 from smaller MLP due to better encoding
  - Combine many hash maps with cells of different resolutions

|                  | Mic  | Ficus | Chair  | Hotdog | Materials | Drums | Ship  | Lego  | avg.  |
|------------------|------|-------|--------|--------|-----------|-------|-------|-------|-------|
| Ours: Hash (1 s) | 26.09| 21.30 | 21.55  | 21.63  | 22.07     | 17.76 | 20.38 | 18.83 | 21.202|
| Ours: Hash (5 s) | 32.60| 30.35 | 30.77  | 33.42  | 26.60     | 23.84 | 26.38 | 30.13 | 29.261|
| Ours: Hash (15 s)| 34.76| 32.26 | 32.95  | 35.56  | 28.25     | 25.23 | 28.56 | 33.68 | 31.407|
| Ours: Hash (1 min)| 35.92| 33.05 | 34.34  | 36.78  | 29.33     | 25.82 | 30.20 | 35.63 | 32.635|
| Ours: Hash (5 min)| 36.22| 33.51 | 35.00  | 37.40  | 29.78     | 26.02 | 31.10 | 36.39 | 33.176|
| mip-NeRF (~hours)| 38.04| 33.19 | 37.14  | 39.31  | 32.56     | 27.02 | 33.08 | 35.74 | 34.510|
| NSVF (~hours)    | 34.27| 31.23 | 33.19  | 37.14  | 32.68     | 25.18 | 27.93 | 32.29 | 31.739|
| NeRF (~hours)    | 32.91| 30.13 | 33.00  | 36.18  | 29.62     | 25.01 | 28.65 | 32.54 | 31.005|
| Ours: Frequency (5 min) | 31.89| 28.74 | 31.02  | 34.86  | 28.93     | 24.18 | 28.06 | 32.77 | 30.056|
| Ours: Frequency (1 min) | 26.62| 24.72 | 28.51  | 32.61  | 26.36     | 21.33 | 24.32 | 28.88 | 26.669|
|                                      | Training speed | Rendering speed |
|--------------------------------------|----------------|-----------------|
| Original NeRF                        | 1-2 days       | 30 sec          |
| KiloNeRF, cached voxels              | 1-2 days       | 1/60 sec        |
| Learned voxels                       | 10-15 mins     | 1/15-1/2 sec    |
| Learned hash maps (Instant NGP)      | 5 sec - 5 mins | 1/60 sec        |
Hash Collision

- When the same feature vector is used for multiple spatial locations, you average gradients over all of them.
  - When only a small fraction of those locations have interesting things going on (e.g. not empty space), then that feature vector will mostly be used to represent the interesting stuff going on there, since gradients from that location will be largest.
Summary

• multiresolution hash encoding
  
  +

• Very small MLP (2-3 layers x 64 channels) decodes the trilinearly interpolated hash map features
  
  +

• optimized CUDA kernels