CMPT 985 - 3D Vision

Homework 3

Neural Field Reconstruction

Traven Blaney - 301539045

Due date: April 26th

Introduction

This homework assignment involved developing a neural network to reconstruct 3D objects from occupancy field samples. The task entailed constructing a model comprising a 3D feature grid and a multilayer perceptron (MLP). The primary goal was to train a single level-of-detail (LOD) dense grid model, a multiple LOD dense grid model, and a hash grid model to minimize the discrepancy between the predicted occupancy fields and the actual (ground truth) occupancy fields. The purpose of this reconstruction exercise is to accurately produce a plausible surface reconstruction that captures the overall shape of the object, given a set of occupancy field samples.

Training

The training setup hyperparameters for the three different models included in this study are summarized below:

```
{
    "lr": 0.01,
    "epochs": 100,
    "current_obj": "bunny.obj",
    "current_obj_path": "processed/bunny.obj",
    "resolution": 128,
    "batch_size": 4096,
    "weight_decay": 0.00001,
    "lr_decay_step": 20,
    "lr_decay_gamma": 0.1,
    "grid_type": "dense",
    "grid_feature_dimension": 16,
    "mlp_width": 64,
    "num_mlp_layers": 2,
    "base_lod": 8,
    "num_lods": 1
}
```


Figure 1: Training hyperparameters for single LOD model


```
{
    "lr": 0.01,
    "epochs": 100,
    "current_obj": "bunny.obj",
    "current_obj_path": "processed/bunny.obj",
    "resolution": 128,
    "batch_size": 4096,
    "weight_decay": 0.00001,
    "lr_decay_step": 20,
    "lr_decay_gamma": 0.1,
    "grid_type": "dense",
    "grid_feature_dimension": 16,
    "mlp_width": 64,
    "num_mlp_layers": 2,
    "base_lod": 6,
    "num_lods": 3
}
```


Figure 2: Training hyperparameters for multiple (3) LOD model

```
{
    "lr": 0.01,
    "epochs": 100,
    "current_obj": "bunny.obj",
    "current_obj_path": "processed/bunny.obj",
    "resolution": 128,
    "batch_size": 4096,
    "weight_decay": 0.00001,
    "lr_decay_step": 20,
    "lr_decay_gamma": 0.1,
    "grid_type": "hash",
    "grid_feature_dimension": 4,
    "mlp_width": 256,
    "num_mlp_layers": 10,
    "base_lod": 4,
    "num_lods": 6
}
```


Figure 3: Training hyperparameters for hash model

As for the training times, the HashGrid was significantly faster compared to the other models, averaging ~11 batch/s. This is compared to the single LOD model at ~3.4 batch/s, and the multiple LOD model at ~3 batch/s. The model state that corresponded to the 'best loss', or smallest loss value, was saved to disk and used in the reconstruction portion of this study.

Loss values typically decreased rather quickly within the first 5 or so epochs, then plateaued with incremental and gradual decreases as it converged to the optimum. This was expected behavior because of the use of a learning rate scheduler (StepLR). Initial loss values for the majority of models was pretty consistent at ~0.5.

Table 1: Final loss values recorded

Mesh	Model	Best Loss	Epoch
bunny.obj	Single LOD	0.276	79
^	Multiple LOD	**0.058**	54
^	Hash	0.354	36
column.obj	Single LOD	0.192	5
^	Multiple LOD	**0.109**	70
^	Hash	0.324	58
dragon_original.obj	Single LOD	0.159	47
^	Multiple LOD	**0.071**	85

^	Hash	0.366	48
serapis.obj	Single LOD	0.241	19
^	Multiple LOD	**0.073**	64
^	Hash	0.362	63
utah_teapot.obj	Single LOD	0.170	40
^	Multiple LOD	**0.052**	23
^	Hash	0.331	73

Results

The result metrics used to evaluate were the Chamfer and Hausdorff distances.

- Chamfer Distance: This metric measures the average of the nearest point distances between every point in one set to the closest point in the other set and vice versa. It's often used in 3D shape comparisons because it provides a measure of how much one shape needs to be altered to resemble another shape.
- Hausdorff Distance: This metric identifies the single longest distance from a point in one set to the closest point in the other set and vice versa, effectively capturing the worst-case scenario. It measures the maximum distance of a set to the nearest point in the other set, providing a stringent measure of similarity. It's particularly sensitive to outliers and useful in applications where the maximum deviation is more critical than the average deviation.

Table 2: Bunny reconstruction results

Mesh	Type	Resolution	Chamfer distance	Hausdorff distance
bunny.obj	one_lod	64	**0.01519175721**	**0.2500628153**
bunny.obj	one_lod	128	0.01562858079	0.2563410681
bunny.obj	one_lod	256	0.01873554495	0.2568706379
bunny.obj	m_lod	64	0.005800271795	**0.2373532429**
bunny.obj	m_lod	128	**0.003507745584**	0.2503851505
bunny.obj	m_lod	256	0.003869496536	0.2502060643
bunny.obj	hash	64	**0.03809925503**	**1.141345247**
bunny.obj	hash	128	0.1015446695	1.323788424
bunny.obj	hash	256	0.1429994862	1.32859881

Single LOD:

The Chamfer distance increased for higher resolutions consistently, as did the Hausdorff distance, suggesting finer details were lost at higher resolutions due to the fixed nature of the single LOD grid.

M LOD:

The Hausdorff distance is best at 64 resolution, compared to the Chamfer distance being best at 128 resolution (although it is very similar to the 256 scenario).

Hash:

The Chamfer distance increased for higher resolutions consistently, as did the Hausdorff distance. The Hausdorff distance is significantly worse for the hash model compared to the other models, indicating the presence of outliers. This could mean that certain local areas or aspects of the data processed with this method deviate significantly from their counterparts in the ground truth set, leading to a distortion that is captured as a high Hausdorff distance - this trend remains consistent for all mesh objects, and is easily visualized.

Table 3: Column reconstruction results

column.obj	one_lod	64	0.01223029132	**0.07267592373**
column.obj	one_lod	128	**0.00952460416**	0.08167877497
column.obj	one_lod	256	0.009960277079	0.08180757332
column.obj	m_lod	64	0.005726882263	0.03157373677
column.obj	m_lod	128	0.003057077364	0.01933368156
column.obj	m_lod	256	**0.002273081278**	**0.01511215553**
column.obj	hash	64	**0.07112445692**	**1.209984444**
column.obj	hash	128	0.2010652268	1.217159811
column.obj	hash	256	0.2743150547	1.239309882

Single LOD:

For the one_lod model, the Chamfer distance is best at the 128 resolution, and for the Hausdorff at 64.

M LOD:

For the m_lod model, the Hausdorff and Chamfer distance is best at 256 resolution.

Hash:

The Chamfer distance increased for higher resolutions consistently for the hash model, as did the Hausdorff distance.

Table 4: Dragon reconstruction results

dragon_original.obj	one_lod	64	0.01065876991	0.1007488612
dragon_original.obj	one_lod	128	**0.009105041673**	**0.09670238947**
dragon_original.obj	one_lod	256	0.009325529953	0.09783375878
dragon_original.obj	m_lod	64	0.004866619813	0.0704408877
dragon_original.obj	m_lod	128	0.00252088015	0.0558024109
dragon_original.obj	m_lod	256	**0.001915062138**	**0.0553626198**
dragon_original.obj	hash	64	**0.1389435904**	**1.271204968**
dragon_original.obj	hash	128	0.2055844275	1.280492152
dragon_original.obj	hash	256	0.2236133617	1.284470186

Single LOD:

For the one_lod model, the Chamfer and Hausdorff distance is best at the 128 resolution. This is likely due to the inability for the one_lod model to capture finer details, which becomes more pronounced at the 256 resolution.

M LOD:

For the m_lod model, the Hausdorff and Chamfer distance is best at 256 resolution.

Hash:

For the hash model, the Hausdorff and Chamfer distance is best at 64 resolution.

Table 5: Serapis reconstruction results

serapis.obj	one_lod	64	**0.02154018899**	**0.2969154786**
serapis.obj	one_lod	128	0.02510398893	0.3008915985
serapis.obj	one_lod	256	0.0280223755	0.3030059076
serapis.obj	m_lod	64	0.00847581704	0.2904314556
serapis.obj	m_lod	128	**0.004040524833**	**0.2639017568**
serapis.obj	m_lod	256	0.005885194032	0.2882738921
serapis.obj	hash	64	**0.05897269194**	**1.196874298**
serapis.obj	hash	128	0.130665132	1.197268896
serapis.obj	hash	256	0.1610292619	1.197059581

Single LOD:

For the one_lod model, the Chamfer and Hausdorff distance is best at the 64 resolution. This is likely due to the inability for the one_lod model to capture finer details, which becomes

more pronounced at the 128 and 256 resolution, for a mesh object which contains a high level of finer resolution regions.

M LOD:

Both the Chamfer and Hausdorff distance in this case is better for the m_lod model at 128 resolution (compared to 256), which is a slight deviation from the aforementioned trends. This is likely due to the complexity of this model in the face region, which proves highly difficult for even a m_lod model to reproduce accurately.

Hash:

For the hash model, the Hausdorff and Chamfer distance is best at 64 resolution.

Table 6: Teapot reconstruction results

utah_teapot.obj	one_lod	64	**0.01860423001**	**0.2061330952**
utah_teapot.obj	one_lod	128	0.02179673912	0.2068172134
utah_teapot.obj	one_lod	256	0.02495320691	0.2070928053
utah_teapot.obj	m_lod	64	0.004233970146	**0.03906958977**
utah_teapot.obj	m_lod	128	0.002445141818	0.0792859462
utah_teapot.obj	m_lod	256	**0.001584508494**	0.1461351329
utah_teapot.obj	hash	64	**0.03820090256**	**1.125810772**
utah_teapot.obj	hash	128	0.0738827983	1.269764859
utah_teapot.obj	hash	256	0.1385403783	1.318165792

Single LOD:

For the one_lod model, the Chamfer and Hausdorff distance is best at the 64 resolution.

M LOD:

The Hausdorff distance is better for the m_lod model at 64 resolution, which is likely due to the introduction of outliers when sampled at a higher resolution. The Chamfer distance is still best at 256, supporting the notion that outliers were introduced at the higher resolution sampling.

Hash:

For the hash model, the Hausdorff and Chamfer distance is best at 64 resolution.

Summary of observations:

Through the comparison of Chamfer and Hausdorff distances across various OBJ files using different reconstruction methods and resolutions, it becomes evident that the m_lod

method consistently delivers superior geometric fidelity, both overall and locally, particularly at higher resolutions. The one_lod method tends to fall short in capturing sufficient detail at high resolutions. At higher resolutions, the reconstruction process becomes more sensitive to data complexity. A single LOD model, which might be optimized for a certain level of detail (64 or 128 resolution), may not effectively manage the increased complexity, resulting in inaccurate or distorted reconstructions at higher resolution. Meanwhile, Hash methods struggle with preserving details and suffer from false occupancy positives (outliers). These findings suggest that for applications demanding precise geometric reconstruction, selecting high-resolution methods that match the required accuracy is crucial - such as the m_lod model. Additionally, Hash methods might need further optimization and careful consideration when processing specific geometries, but could be better suited for large volumes where minimizing memory expenditures is vital.

Visualizations:

All reconstructions were imported and rendered (EEVEE render engine) in Blender for visualization purposes, using a Python script. This Blender file was included in the submission of this project, in the Github repository, as were the images.

Figure 4: Bunny reconstructions, m_lod model, ordered by resolution (64 → 128 → 256)

Figure 5: Bunny reconstructions, one_lod, ordered by resolution (64 → 128 → 256)

Figure 6: Bunny reconstructions, hash model, ordered by resolution (64 → 128 → 256)

Figure 7: Column reconstructions, m_lod model, ordered by resolution (64 → 128 → 256)

Figure 8: Column reconstructions, one_lod model, ordered by resolution (64 → 128 → 256)

Figure 9: Column reconstructions, hash model, ordered by resolution (64 → 128 → 256)

Figure 10: Dragon reconstructions, m_lod model, ordered by resolution (64 → 128 → 256)

Figure 11: Dragon reconstructions, one_lod model, ordered by resolution (64 → 128 → 256)

Figure 12: Dragon reconstructions, hash model, ordered by resolution (64 → 128 → 256)

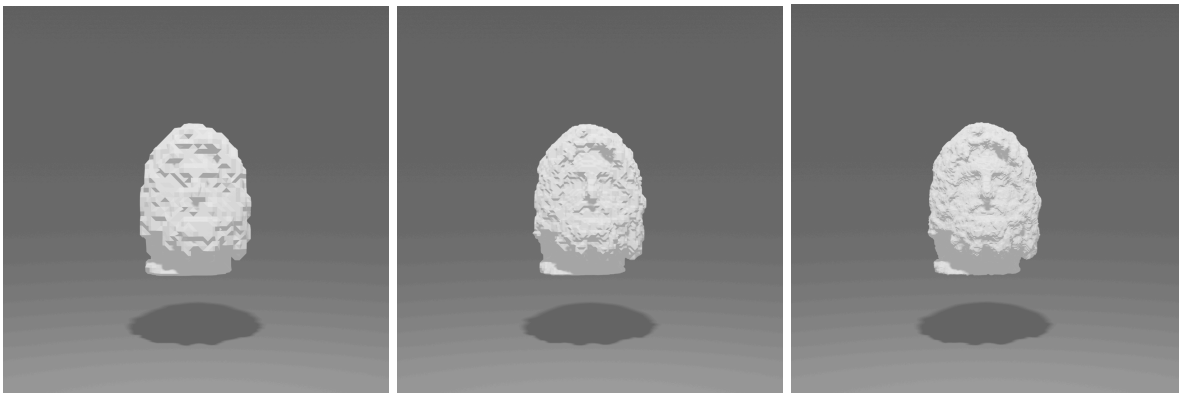

Figure 13: Serapis reconstructions, m_lod model, ordered by resolution (64 → 128 → 256)

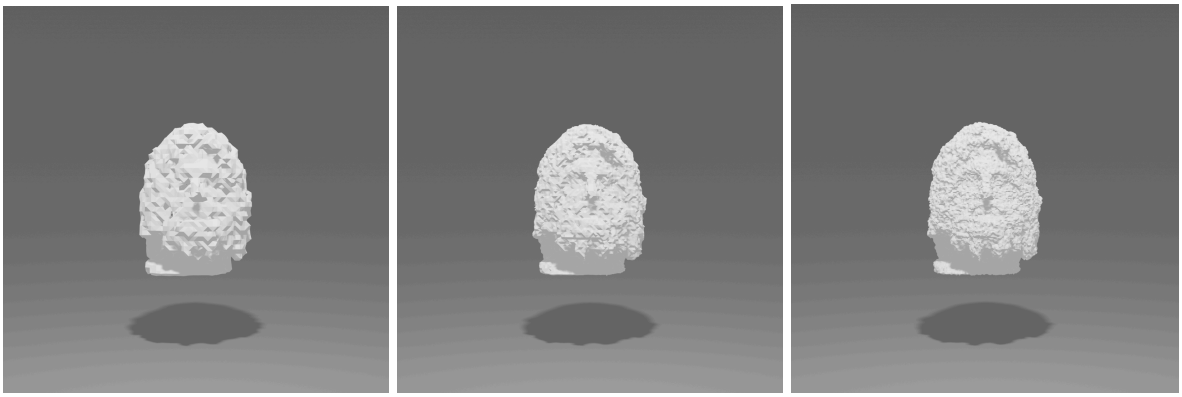

Figure 14: Serapis reconstructions, one_lod model, ordered by resolution (64 → 128 → 256)

Figure 15: Serapis reconstructions, hash model, ordered by resolution (64 → 128 → 256)

Figure 16: Teapot reconstructions, m_lod model, ordered by resolution (64 → 128 → 256)

Figure 17: Teapot reconstructions, one_lod model, ordered by resolution (64 → 128 → 256)

Figure 18: Teapot reconstructions, hash model, ordered by resolution (64 → 128 → 256)

Discussion

Single LOD vs. Multiple LOD Models:

Single LOD Limitations: A model operating with a single level of detail (LOD) must employ a fixed grid resolution. This resolution needs to be optimally selected to balance between capturing sufficient detail and maintaining manageable computational demands. If the resolution is set too low, the model fails to capture finer details; conversely, if set too high, it leads to increased computational overhead, potentially making training and processing unfeasible. Typically, one such resolution is adequate for large-scale features but falls short in representing finer structural details accurately. This is represented through the single LOD model consistently performing best when sampled and reconstructed using a 64, or 128 resolution grid, but performs slightly worse with higher resolutions due to the presence of finer details becoming lost

Multiple LOD Advantages: In contrast, multiple LOD models excel by incorporating various grid resolutions that handle different scales of detail concurrently. This capability allows these models to allocate resources more efficiently—applying higher resolutions selectively to areas requiring fine detail and lower resolutions where less detail suffices. Such adaptability is especially crucial for complex models, where precision across scales can dramatically enhance the outcome.

Conclusions:

The comparative analysis between single and multiple LOD models clearly favors the latter in terms of both performance and functional adaptability. Multiple LOD systems provide a dynamic framework that adjusts resolution based on the detail requirement of different areas within the modeled space, leading to more accurate and efficient representations. This adaptability is particularly beneficial in complex environments where detail levels vary significantly across the space, ensuring that each area is represented with an appropriate level of detail to optimize both accuracy and computational efficiency.

Comparison of HashGrid vs. DenseGrid:

Fundamental Differences in Data Handling:

HashGrid: The HashGrid method uses a sparse data structure where feature vectors are stored only at actively used hash locations. This approach ensures memory efficiency, particularly beneficial in large or sparse datasets. However, this also means that the hash table size is smaller relative to the total possible number of locations in a high-resolution space. This restriction can lead to information loss, especially if the hash function causes collisions or misses subtle variations in the data. In a hash-based system, collisions occur when multiple data points are mapped to the same hash bucket, leading to potential data loss or inaccuracies in the representation. These issues are evident throughout this study, as the HashGrid performs worse when reconstructed with a higher resolution grid. Interpolation errors might also be more pronounced in HashGrids due to the uneven distribution of data points in hash buckets. Inaccurate interpolation can lead to 'ghost' surfaces appearing where the algorithm incorrectly predicts the presence of a surface based on neighboring data points' influence, which might not actually be direct neighbors in terms of coordinates (this problem is typically mitigated in DenseGrids).

DenseGrid: In contrast, the DenseGrid allocates memory for a complete and uniform grid across the specified space, regardless of data sparsity. Each grid point holds a feature vector, allowing for more comprehensive data representation. The DenseGrid method is more likely to capture fine details due to the uniformity and density of the feature distribution. This notion functions even better in models that employ multiple LOD grids.

Impact of Feature Vector Dimensionality:

Feature Dimension in HashGrid: The choice of a smaller grid feature dimension in HashGrids (typically to maintain memory efficiency) can limit the amount of information each vector can represent. This is a trade-off for using less memory but can lead to a less detailed model representation, potentially impacting the accuracy of tasks such as surface reconstruction.

Feature Dimension in DenseGrid: DenseGrids can afford higher feature dimensions because they do not prioritize memory efficiency to the same extent. Because dense grids allocate memory for a complete grid irrespective of data sparsity, it's suitable to have richer feature vectors. This allows each point in the grid to hold more information, leading to richer and potentially more accurate representations.

Error Metrics and Performance:

Hausdorff Distance: The Hausdorff distance, a measure used to determine the maximum distance of a set to the nearest point in the other set, tends to be higher for HashGrids. This metric is particularly sensitive to outliers, which are more prevalent in reconstructions from

HashGrids due to the less detailed feature representation and potential hashing collisions - leading to false positives.

Conclusions:

The trade-offs between HashGrid and DenseGrid become particularly apparent when considering the balance between memory efficiency and accuracy. HashGrids, while advantageous in scenarios requiring efficient memory usage and capable of handling large, sparse datasets, can suffer from reduced accuracy and increased errors due to their sparse nature and lower feature dimensionality. DenseGrids, although more memory-intensive, provide a more accurate and detailed representation of space, making them suitable for applications where detail fidelity is critical. This difference is particularly exemplified in metrics like the Hausdorff distance, which highlights the greater propensity for errors and outliers in HashGrid-based reconstructions.

Resolution Changes in HashGrid and DenseGrid Models:

HashGrid Performance at High Resolutions:

Increased Collisions: At higher resolutions, HashGrid models tend to exhibit more pronounced errors. This issue arises because as the grid resolution increases, the likelihood of hash collisions also increases.

Sparse Data and Discretization: HashGrids handle data sparsely by discretizing it into hash buckets. At higher resolutions, the discretization process becomes more challenging because the grid needs to manage a greater number of data points within the limited capacity of the hash table. This limitation can exacerbate the effects of collisions and lead to a loss in detail where multiple distinct points might be treated as identical due to hashing to the same bucket.

DenseGrid Performance at Higher Resolutions:

Multiple LODs in DenseGrid: With DenseGrid, increasing the resolution generally enhances the results in scenarios employing multiple LODs. The inclusion of smaller resolution grids within the DenseGrid allows these models to capture and reconstruct finer details more effectively. As the resolution increases, these finer details become more pronounced and significant, aiding in a more precise and detailed reconstruction.

Single LOD Limitations: However, for DenseGrid models operating under a single LOD, higher resolutions can paradoxically worsen performance. This deterioration occurs because a single resolution must handle all details, from the largest to the most minute. Without the flexibility to adjust the grid's density based on the level of detail necessary, the single LOD model may struggle with the computational burden required to reconstruct excessively fine details. This can lead to inefficiencies and inaccuracies in areas where a coarser grid might have been sufficient.

Conclusions:

The impact of resolution changes in HashGrid and DenseGrid models highlights the strengths and weaknesses of each approach under different configurations. HashGrid models face challenges at higher resolutions primarily due to increased hash collisions and the limitations of discretizing a larger volume of data into a finite number of hash buckets. On the other hand, DenseGrid models benefit from multiple LODs at higher resolutions, as they can leverage finer grids to capture detailed features effectively. However, DenseGrid models with a single LOD can suffer from increased computational demands to capture an occupancy field accurately, and potential inaccuracies at higher resolutions, underscoring the advantages of using multiple LODs to handle varying levels of detail efficiently.