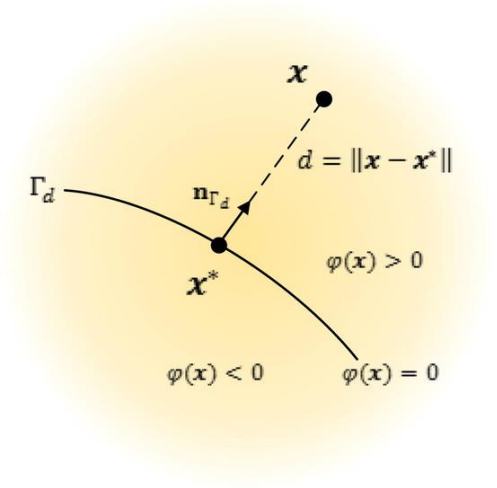
Rendering Using Implicit Neural Representation of Geometry with Ray Marching

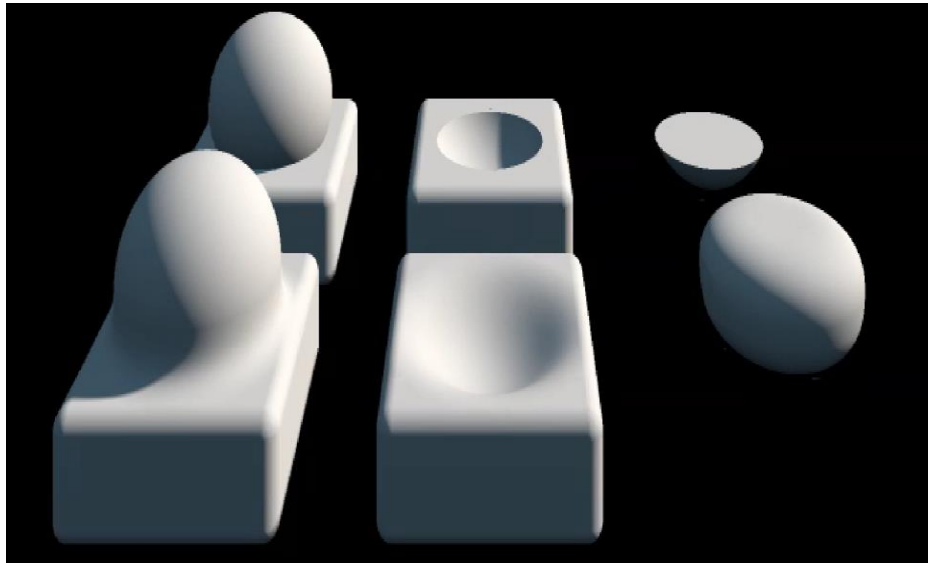
Purva Kulkarni
Dilip Subbain G
Medha Rudra
Reethu Vattikunta
Ninad Bhagwat

Background

- A Signed Distance Function (SDF) is a mathematical representation that encodes the shortest distance from any point in space to a surface. The "signed" part indicates whether the point is inside or outside the surface: distances are positive outside the surface and negative inside.



- **Use Cases of SDFs in Graphics and Other Industries:**
  1. **3D Rendering**:
     1. SDFs are widely used in ray-marching techniques to render smooth surfaces.
     2. SDFs enable smooth blending between different shapes and objects, making it easy to combine models directly in the mathematical representation, allowing for more dynamic and procedural modeling.
  2. **Physics Simulations**:
     1. SDFs are useful for collision detection by determining the closest points on an object's surface.
     2. With SDFs, simulating soft or deformable objects becomes easier since the distance function can be adjusted dynamically to reflect the changing shape of an object over time, ensuring accurate collision detection and response.



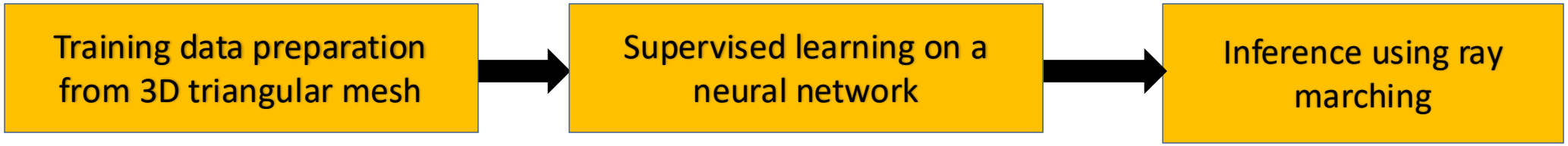
- **Traditional approach to convert a triangular mesh to SDF**
  - **Discretization using voxelization:**
    - A query point and the mesh are approximated as voxels in an infinite grid and the distance to the surface voxel is computed
    - Entails a high computational cost

Mesh to SDF conversion pipeline

- We propose a novel method to encode 3D meshes as a neural network that computes the minimum distance from any point in space to the surface of the mesh. Our approach significantly improves performance by leveraging the power of neural networks to represent the signed distance function (SDF).

- With this technique, we achieve **distance computation from any point to a mesh surface in an average of 0.008 secs as tested on models with triangle count ranging from 1k to 500k**, a significant improvement over traditional SDF methods, which typically require more computationally intensive processes to determine distances from a point to the mesh surface.

- This opens up new possibilities for real-time applications in 3D rendering, simulations, and interactive environments, where high-speed distance calculations are critical. By using a neural network to implicitly encode the mesh geometry, we not only reduce the computational burden but also increase the scalability of complex models and scenes.



Data Preparation

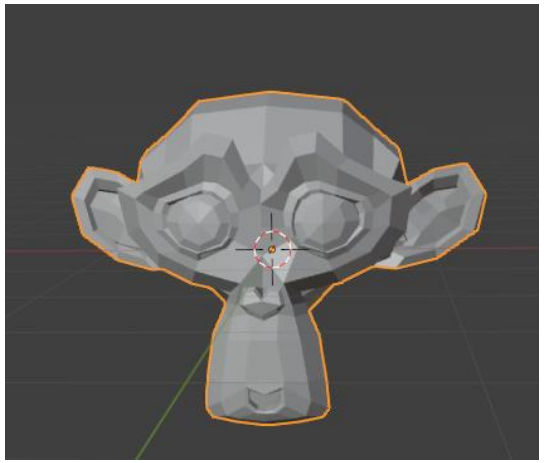
Aim:

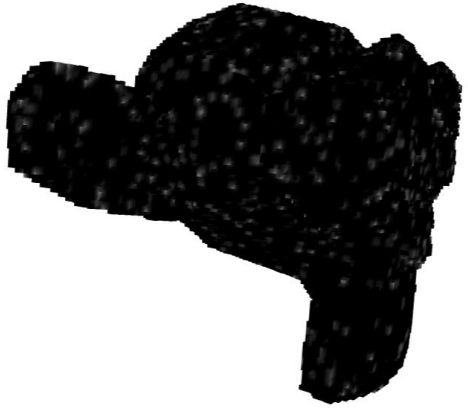
To generate tuples of the form (x,y,z) along with their corresponding distances (d) for supervised learning purposes.

1. Generate binary voxelized grid from 3D mesh
2. Compute point-distance tuples for training

Data Preparation
Generate the binary voxelized grid:

- Scale the triangular mesh to fit within the range of -1 to 1.
- Create a voxel grid with a resolution that matches the desired model accuracy.
- For each triangle in the mesh, check the centroid to determine if it occupies a voxel.
- Construct a binary grid where voxels inside the mesh are marked as 1, and those outside the mesh are marked as 0.
- The **time complexity** for generating the binary grid is **O(n), where n = #triangles.**

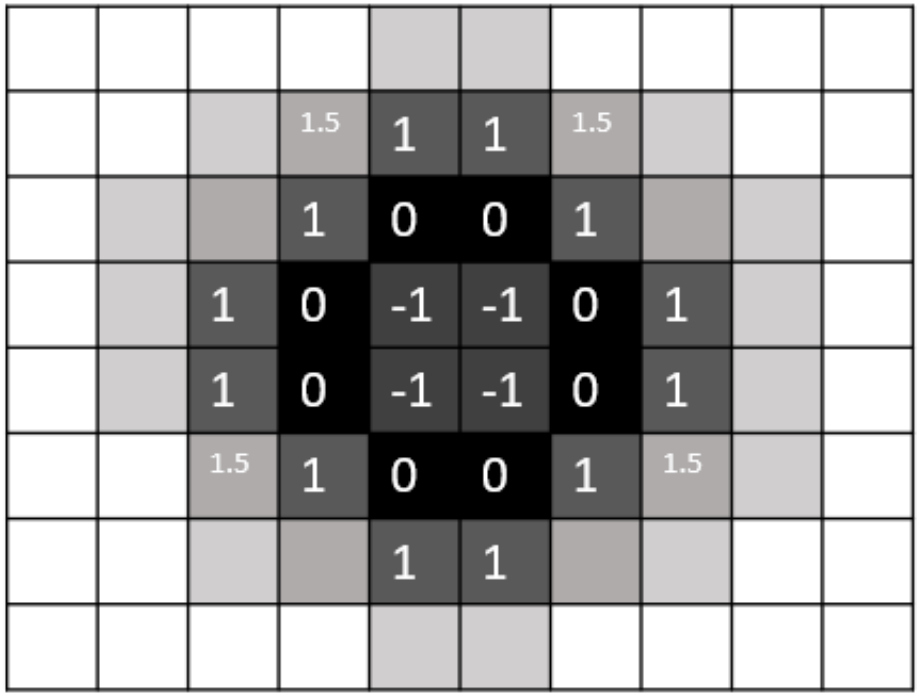




Data Preparation

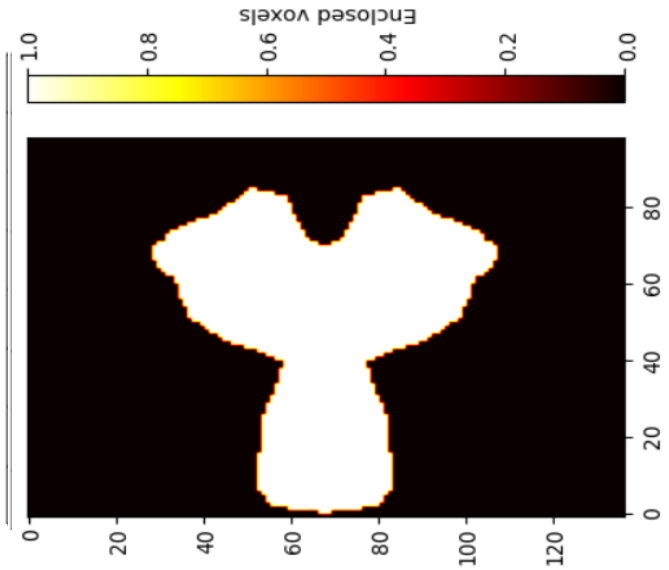
Compute Point-Distance Tuples for Training:

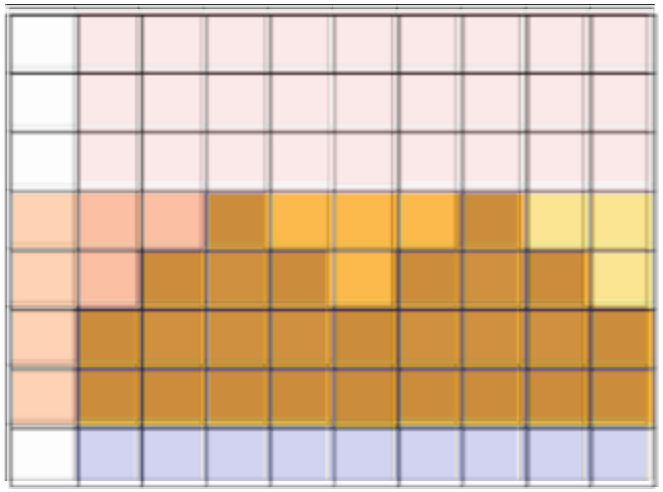
1. Distance Calculation with BFS:
    1. Start with the surface voxels, marking their distances as 0. Use the centroids of these surface voxels as points and push them into a queue.
    2. Perform BFS by popping voxels from the queue, calculating distances for neighboring voxels layer by layer.
    3. For each processed voxel, store the signed distance function (SDF) value and the nearest surface voxel.
    4. For each neighboring voxel, check the nearest surface voxel associated with adjacent voxels, and assign the surface voxel that provides the smallest distance to the current voxel.



Data Preparation
Compute Point-Distance Tuples for Training:

2. Assign Negative Distances for Internal Voxels:
   1. To assign negative distances for voxels inside the surface, perform a grid scan in six directions: x, y, z, -x, -y, and -z.
   2. In each direction, mark all voxels that appear after the first detected surface voxel with a value of 1.
   3. Take the cumulative sum across all six directions, and use a mask to identify the enclosed voxels. Voxels with a cumulative value of 6 are considered to be inside the surface, and their distances are assigned as negative.

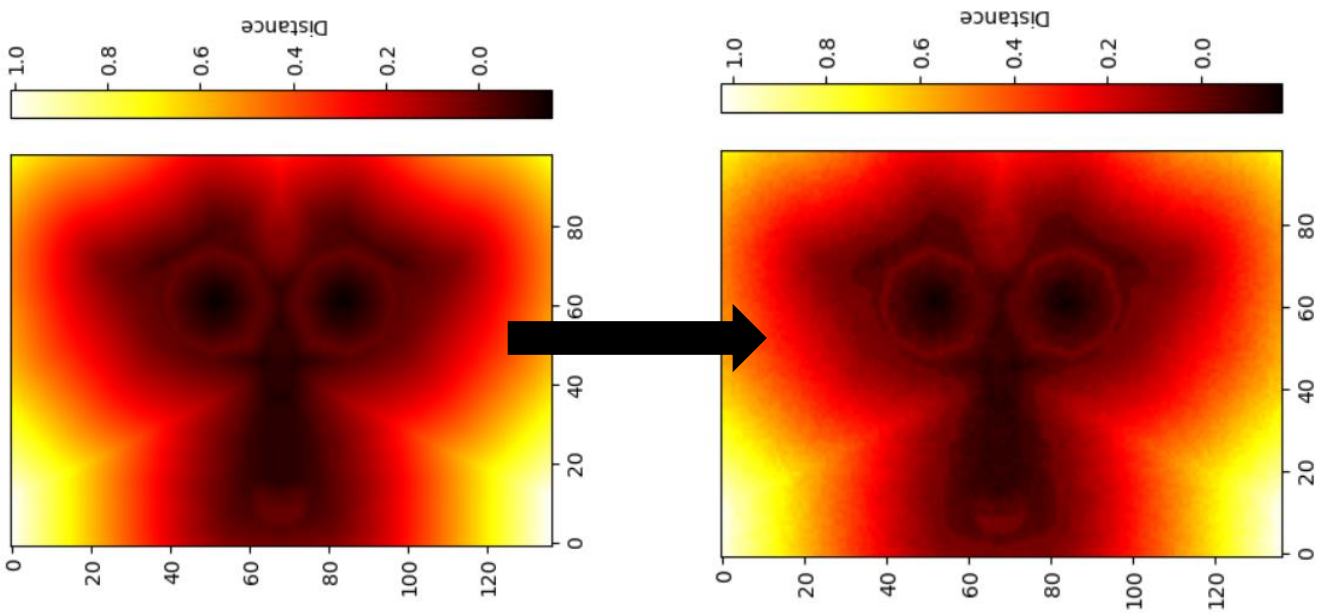




Data Preparation
Compute Point-Distance Tuples for Training:

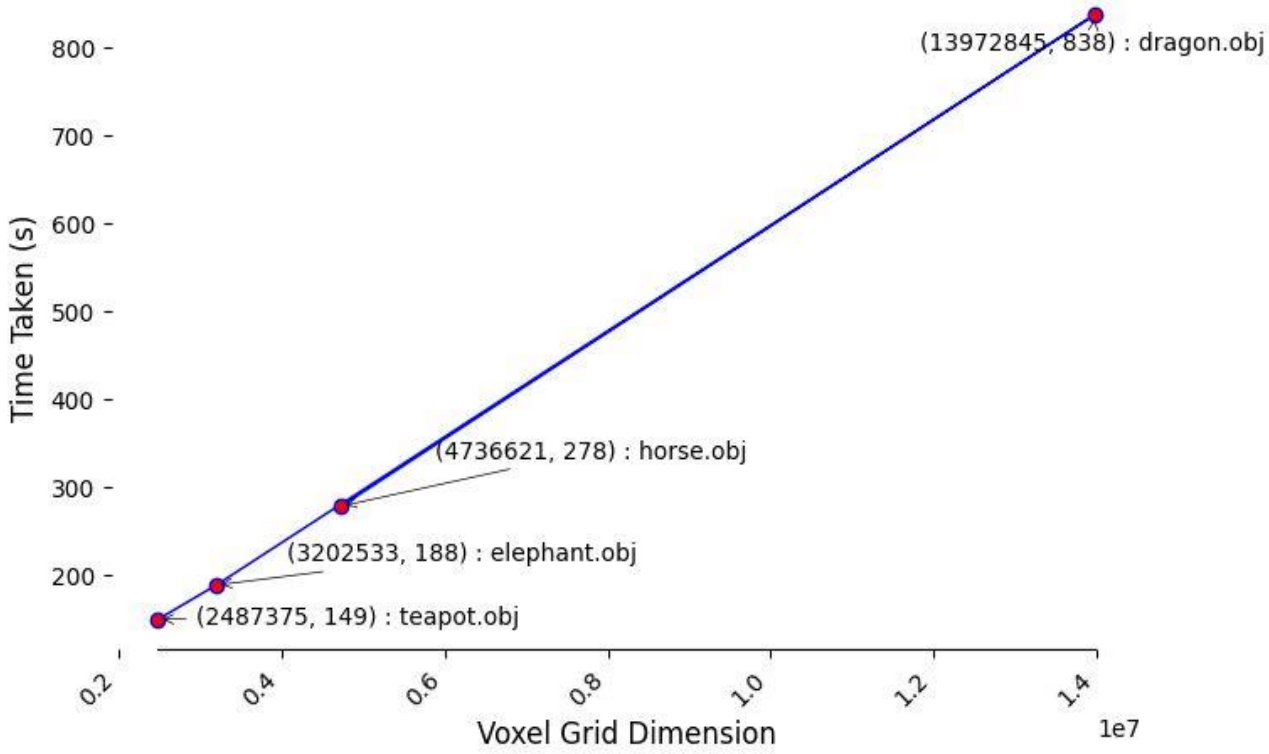
3. The time complexity of this distance computation is O(n), where n is the number of voxels.

4. To prevent the machine learning model from detecting artificial patterns, perturb the centroids slightly, ensuring that the points are not perfectly aligned in a grid.



Data Preparation
Time measurment

Model	Number of triangles in the mesh	Voxelized Grid dimensions	Time taken (in secs)
teapot.obj	6320	2487375	149
elephant.obj	49728	3202533	188
horse.obj	122075	4736621	278
dragon.obj	249882	13972845	839

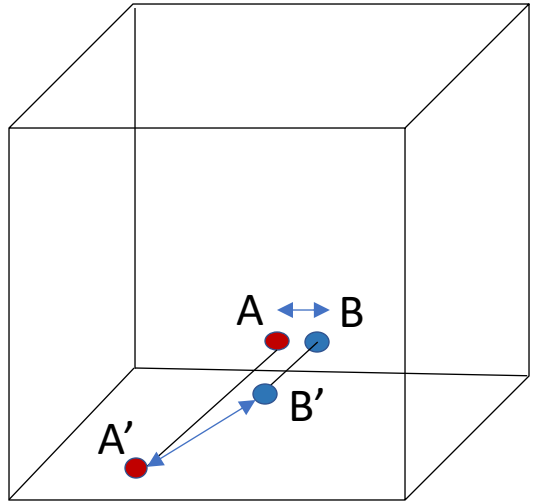


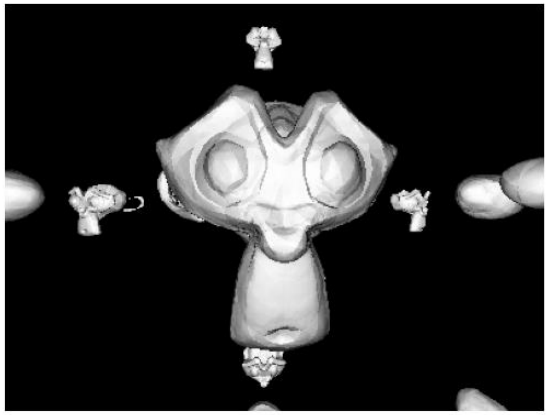
Training a neural network to represent equivalent SDF

- Model architecture
  - Input: 3D point,
  - Output: minimum distance to mesh surface
  - The input space needs to be expanded to a positional encoding, as presented in the NeRF paper.

    $p \rightarrow [p, \sin(2\pi p), \cos(2\pi p), \ldots, \sin(2^L \pi p), \cos(2^L \pi p)]$

    where $p \in$ [x, y, z]
  - We need to incorporate p into the encoded vector because the sine and cosine components are cyclic. As a result, a single encoded vector can map to multiple mappings in R^3





  - Artifact arising due to absence of p in the encoding vector.

Training a neural network to represent equivalent SDF

- Model architecture
  - Positional encoding is essential for capturing intricate details in a mesh, as it enables the model to capture high-frequency variations more effectively. This optimization allowed for a reduction in the model's complexity, shrinking the requirement from two hidden layers with 1024 neurons each to two hidden layers with just 512 neurons each.

2 hidden layers, 1024 neurons

2 hidden layers, 512 neurons,
Positional encoding

Training a neural network to represent equivalent SDF

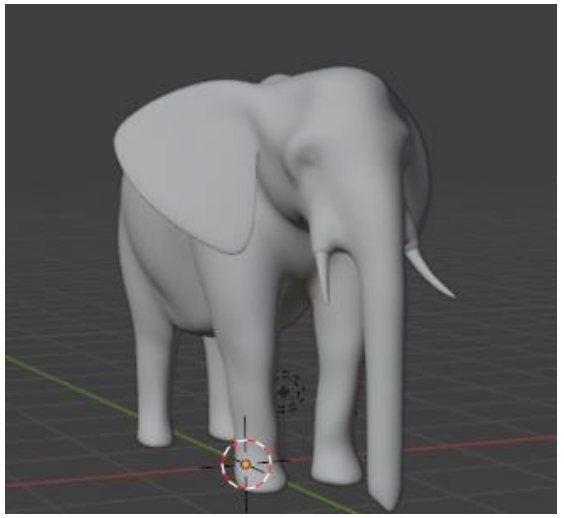
- Model architecture
  - The activation function we chose was Leaky ReLU, with a negative slope of 0.01.

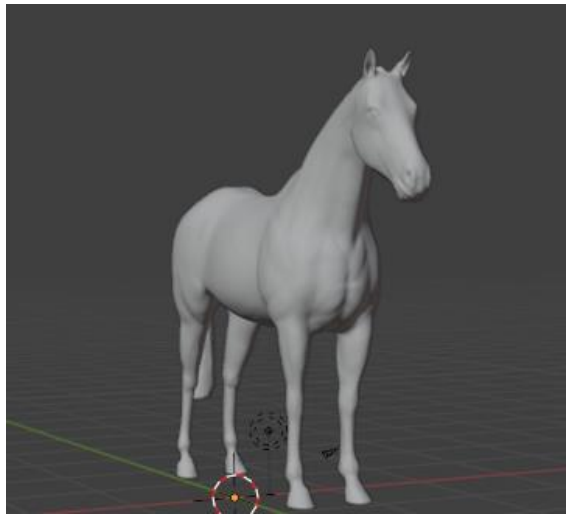
    Our decision to use Leaky ReLU instead of alternatives like ReLU, ELU, or smooth operators like Tanh was purely experimental. We observed better performance with this choice during testing.
  - The other model configurations are standard
    - Optimizer = Adam
    - Loss: MSE
  - No regularizers used as model fit both the training and test data well.

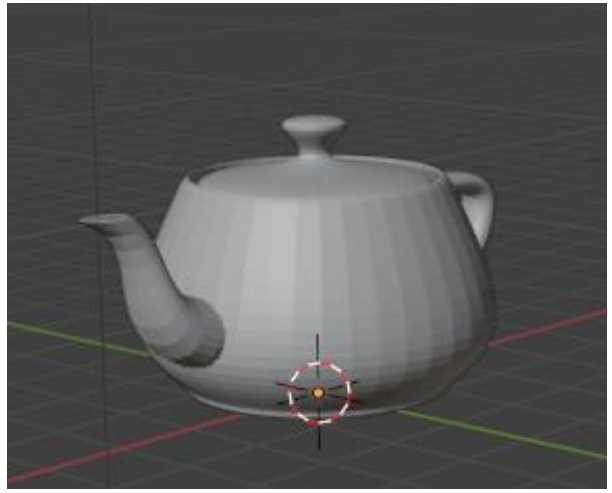
Inference

- Tested the accuracy of SDF generated from mesh using ray marching

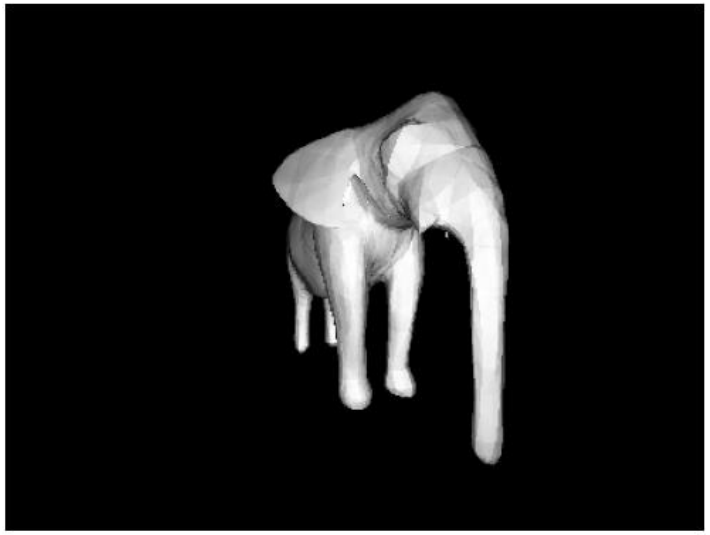
Ground truth

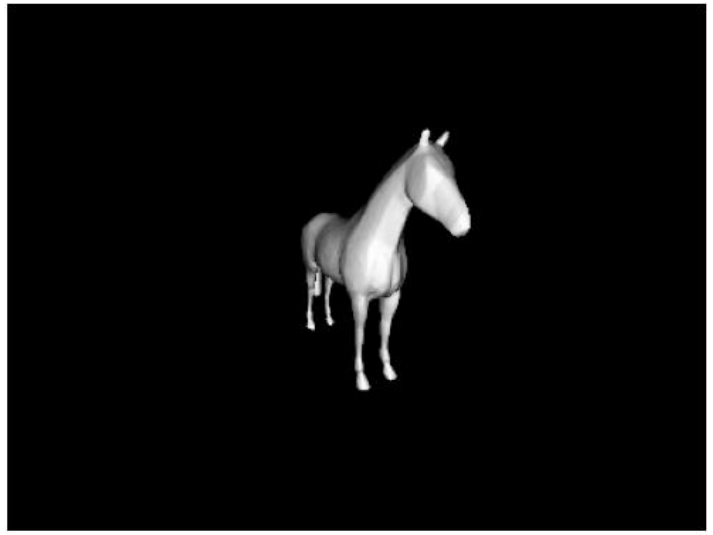


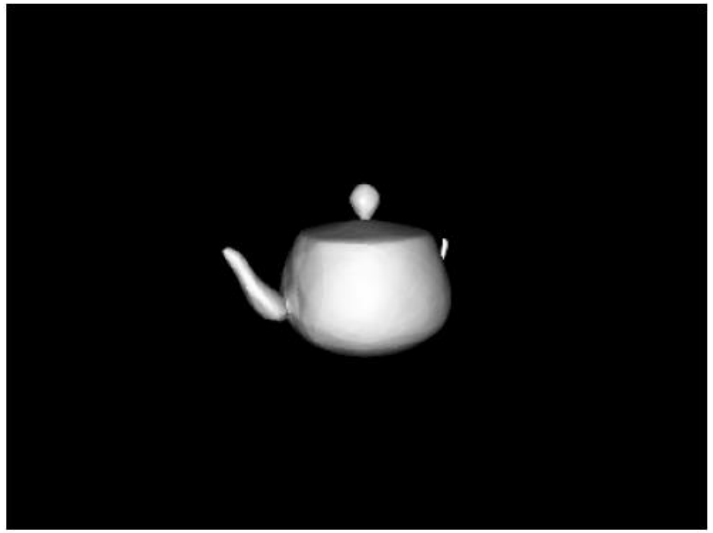




Neural SDFs



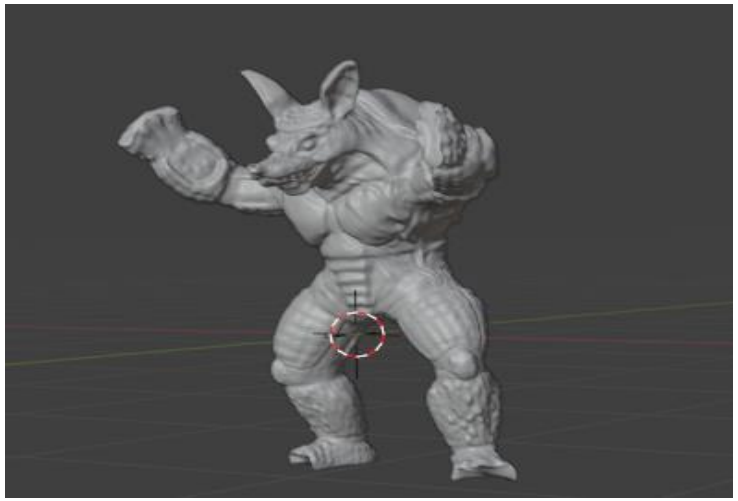


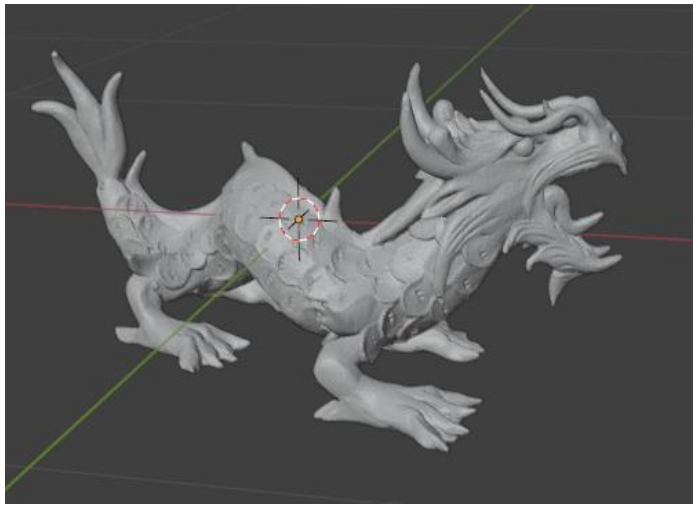


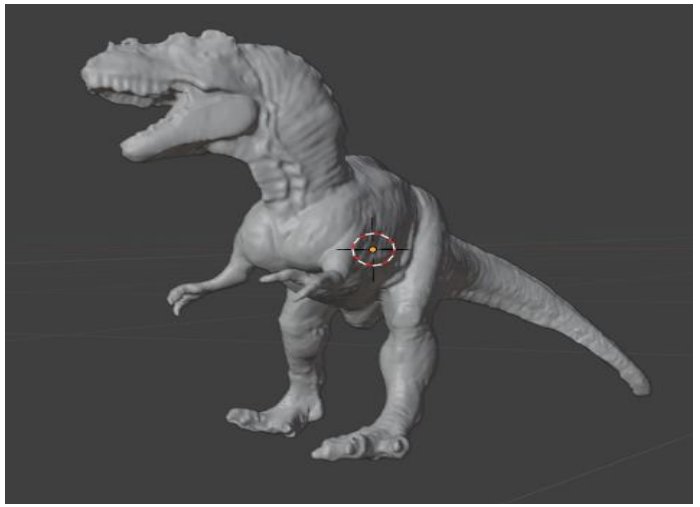
Inference

- Tested the accuracy of SDF generated from mesh using ray marching

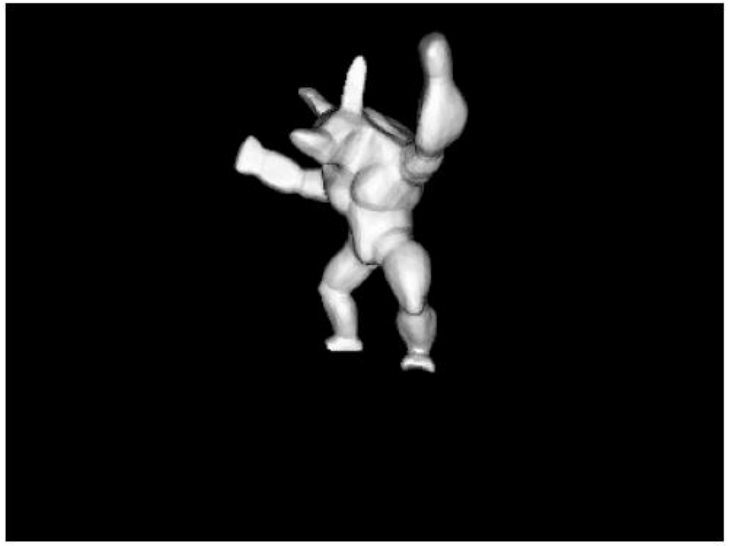
Ground truth

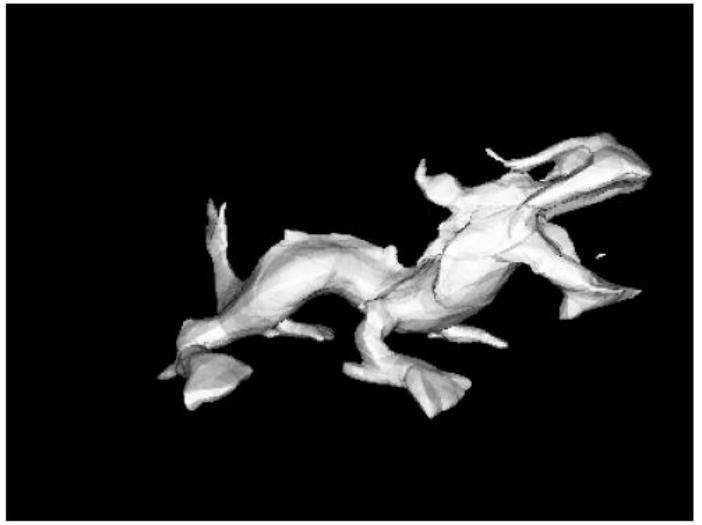


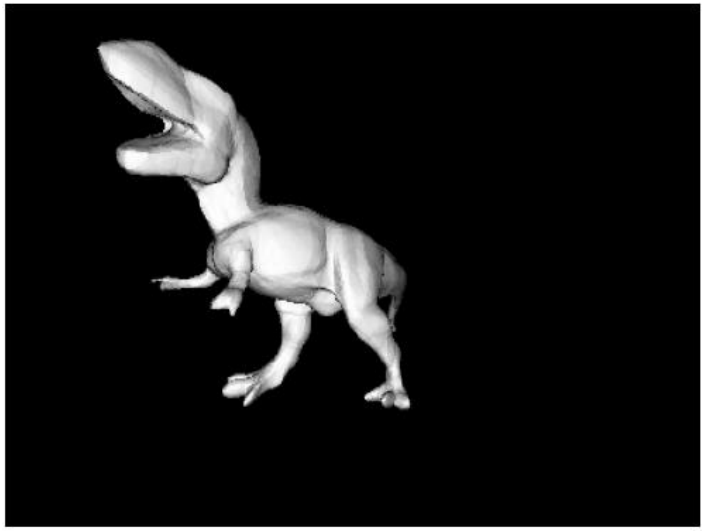




Neural SDFs







Inference

Model	Number of triangles in the mesh	Voxelized grid dimensions	Time taken to render (in secs)	
			Mesh	SDF
teapot.obj	6320	2487375	0.04	0.0075
elephant.obj	49728	3202533	0.25	0.005
horse.obj	122075	4736621	0.833	0.0057
dragon.obj	249882	13972845	3.494	0.0079