RBE 549: Project 2
Buildings built in minutes - SfM and NeRF

Deepak Harshal Nagle
Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: dnagle@wpi.edu
Telephone: (774) 519-8335

Irakli Grigolia
Computer Science
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: igrigolia@wpi.edu
Telephone: (508) 373-3402

***Abstract*—This project aims to apply computer vision techniques to reconstruct 3D scenes and estimate the camera poses using a method called Structure from Motion (SfM). SfM is a technique that utilizes a series of 2D images to reconstruct the 3D structure of a scene and produce point cloud-based 3D models. The principle of stereoscopic photogrammetry is used in SfM to calculate the relative 3D poses of an object from stereo pairs by using triangulation methods.**

PHASE 1: CLASSICAL APPROACH

A. *Estimating Fundamental Matrix*

In stereo geometry, two camera poses are subject to an epipolar constraint. This means that if a 3D point is projected onto one of the camera poses, its corresponding projection on the other pose must lie on a line. The relationship between the two projections is captured by the Fundamental matrix.

The Fundamental matrix represents a system of linear equations with 9 unknowns, which can be solved using Singular Value Decomposition (SVD). After solving the system, the rank constraint is enforced by setting the last singular value to zero and recomputing the Fundamental matrix.

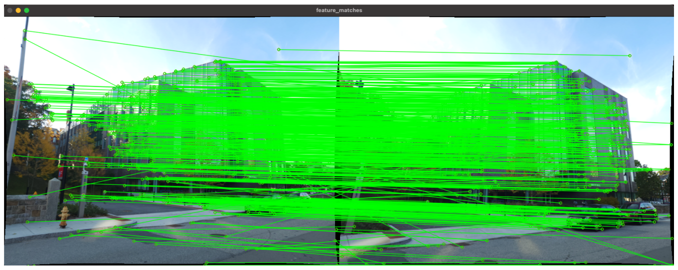

Fig. 1. Matching Features

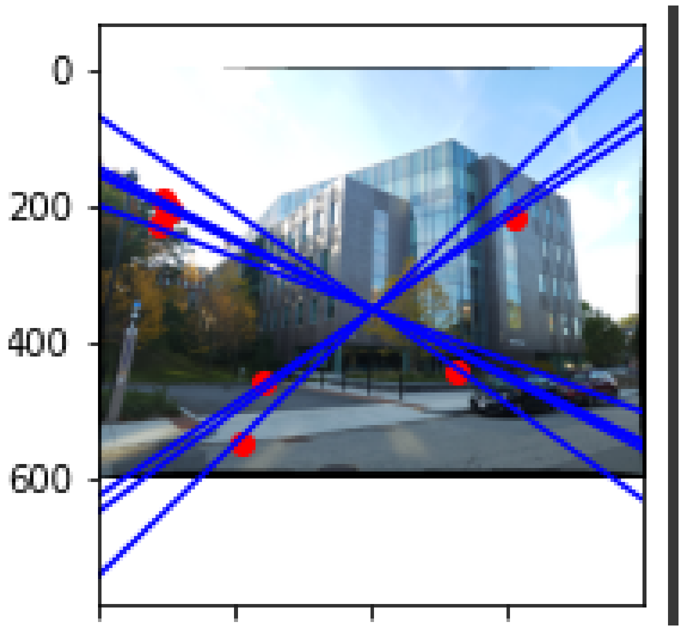

Fig. 2. Epipolar Lines

B. *RANSAC*

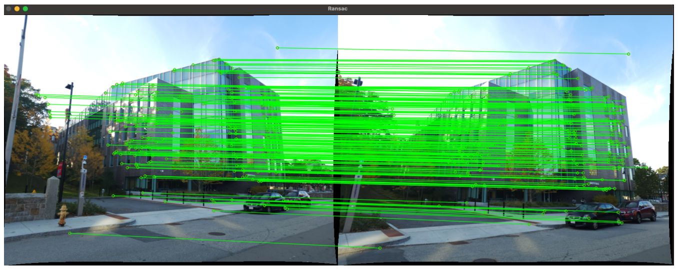

Fig. 3. Feature Matching After Ransac

RANSAC is a robust algorithm for outlier detection and removal, which is commonly used in computer vision to refine feature correspondences. It works by repeatedly sampling subsets of the data and estimating a model from each sample. The best model is then selected based on the number of inliers it has, and these inliers are used to re-estimate the model. This process is repeated until a satisfactory solution is found. In this case, RANSAC was used to filter out incorrect matches and improve the accuracy of the feature correspondences.

Some of the feature matches that we were given were a bit off. To refine these correspondences, RANSAC was applied with the Fundamental matrix as the underlying model.

C. *Estimating Essential Matrix from Fundamental Matrix*

The Essential matrix builds upon the concepts of the Fundamental matrix, by extending the relationship from the image view to the camera view. This allows for the establishment of a physical and geometrical constraint in the relative poses of the two cameras in the stereo setup. In other words, it provides a means of obtaining information about the spatial relationship between the two cameras, beyond just their projection onto the image plane.

This additional information can be used to recover the 3D structure of the scene, as well as the camera poses, by triangulating matched points in the two views. The Essential matrix is also useful in solving the stereo correspondence problem, which involves finding the corresponding points in each image pair.

Overall, the Essential matrix is a powerful tool for stereo vision applications, providing critical information for 3D scene reconstruction and other computer vision tasks.

D. *Estimating Camera Pose from Essential Matrix*

We utilized SVD and some mathematical techniques, as stated in the problem description, to decompose the Essential matrix into rotation and translation matrices. Additionally, we took measures to ensure that the derived rotation matrix was indeed a rotation matrix by performing SVD cleanup.

This process of decomposing the Essential matrix is crucial for estimating the relative pose of two cameras observing a scene. By obtaining the rotation and translation matrices, we can determine the position and orientation of one camera with respect to the other, allowing us to triangulate 3D points from corresponding 2D image points. This is an essential step in 3D computer vision and finds application in various fields such as robotics, augmented reality, and autonomous vehicles.

E. *Linear Triangulation*

Linear Triangulation is the process of finding the 3D coordinates of world points by using feature correspondences, camera matrix, and camera poses. Each 2D-3D correspondence provides two equations for each camera projection. Ideally, the intersection of the two lines formed by a 3D point and its projections in the two images should give us the triangulated 3D point. However, in practice, the lines are usually non-coplanar, making an exact solution impossible. Instead, we find the best approximation to the solution by solving a system of linear equations using SVD. The scale parameters in the equations are eliminated by exploiting the fact that a vector's cross-product with itself is zero. To ensure the correct solution, a chirality check is performed, which ensures that the points in front of the camera have positive Z-coordinates.

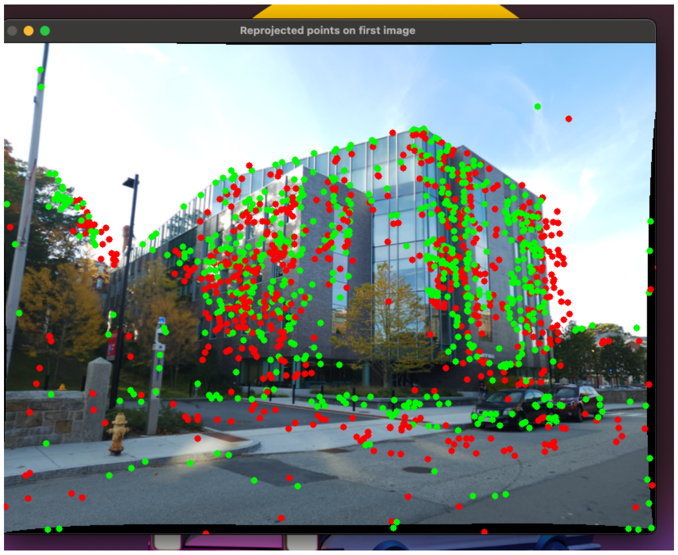

Fig. 4. Linear Triangulation Re-projection

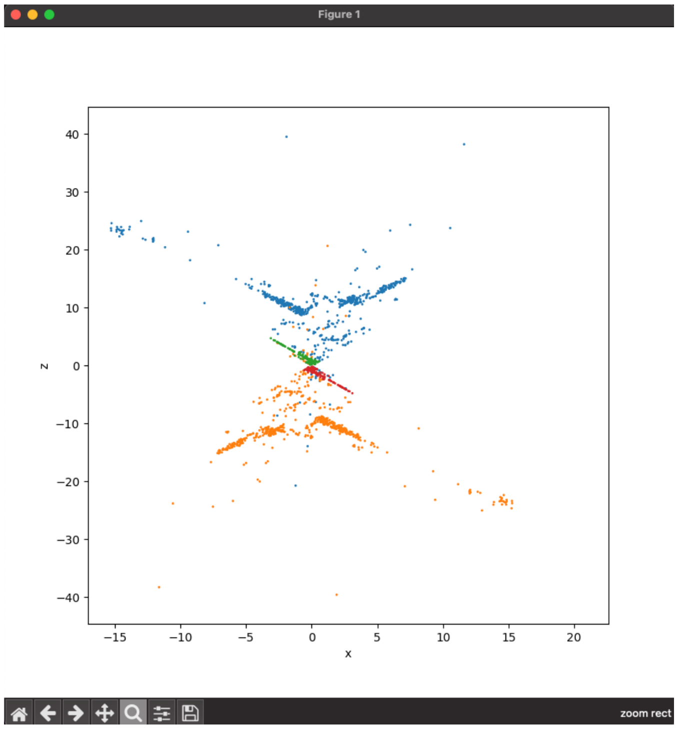

Fig. 5. Linear Triangulation

F. *Non-Linear Triangulation*

After obtaining the camera's actual pose, the next step is to minimize the error between the re-projected world point and the detected image point. While linear error reduction is effective, it may not be optimal for the 3D world where geometric error is more relevant. Therefore, we compute the non-linear error and aim to converge the re-projected point to obtain a more accurate world point. To achieve this, we utilized the least squares function from scipy.

It is worth noting that the non-linear triangulation approach provides more accurate results compared to the linear method. This is because the non-linear method considers the non-linearities in the camera's projection model, which the linear method does not account for. Additionally, the results of the non-linear method are more consistent with the actual 3D world geometry, making it a more reliable approach. Ultimately, accurate 3D reconstruction is essential for many applications, including virtual reality, augmented reality, and

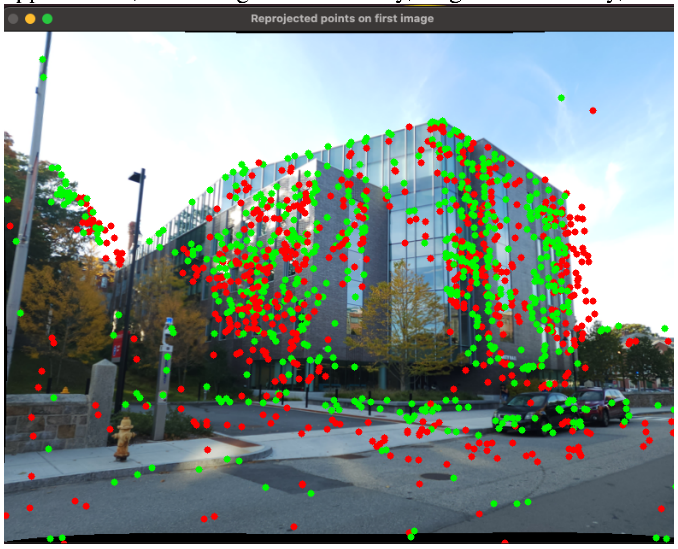

Fig. 6. Non-Linear Triangulation

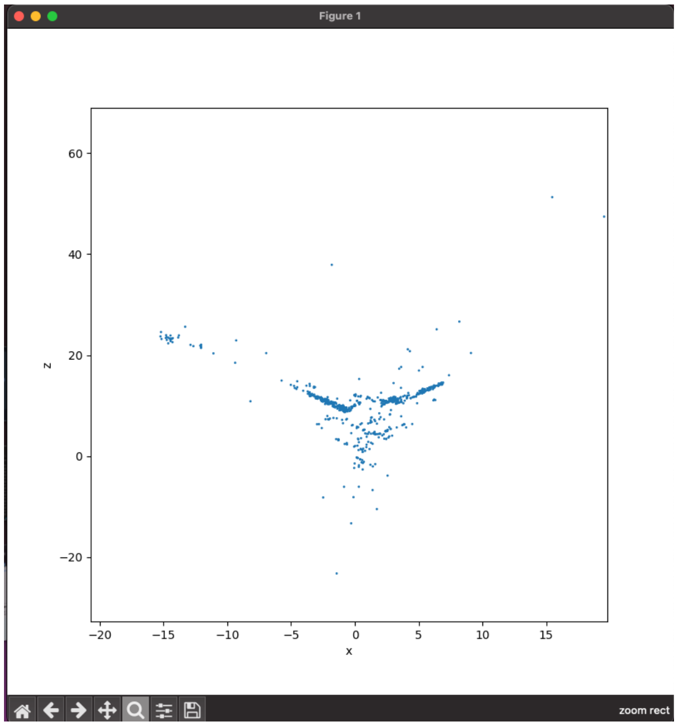

Fig. 7. Linear Triangulation

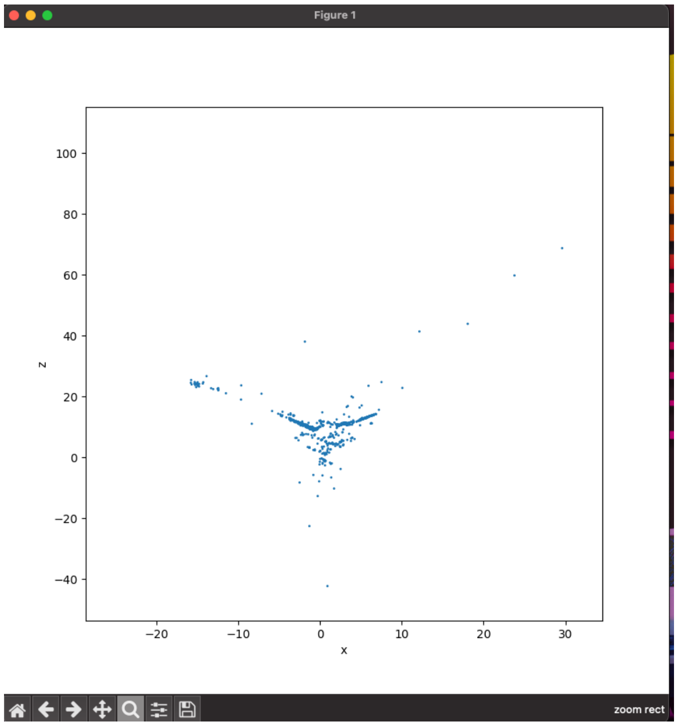

Fig. 8. Non-Linear Triangulation

G. Perspective-n-Point (PnP)

Once we obtained optimized world coordinates for two camera frames, we proceeded to estimate the pose of the other four frames using all the available images. To compute the camera poses, we employed the following methods:

1. Linear PnP: We solved a linear least squares equation with 12 correspondences or 6 point correspondences to calculate the new camera pose (R and T). We used the SVD to extract the R and T values from the last row of VT.

2. PnP RANSAC: PnP is susceptible to outliers, which can result in inaccurate camera poses. Therefore, if the reprojection error for the new camera is less than a given threshold epsilon, the points were added to the inlier set for a more robust camera pose estimation.

3.Nonlinear PnP: Similar to triangulation, we computed the geometric loss using least squares to further minimize the error and obtain a more accurate camera pose.

Using the nonlinear PnP method, we obtained the camera pose for each camera relative to the X points. We plotted these camera poses along with the triangulated points for the Unity hall and the other building dataset.

Additionally, the nonlinear PnP method was used to estimate the camera poses of all the remaining frames, with each frame being considered relative to the triangulated 3D points. This helped in ensuring a consistent and accurate reconstruction of the 3D scene.

H. Bundle Adjustment

After obtaining the camera poses and world coordinates, the next step is to refine them using Bundle Adjustment. This process involves optimizing the positions of both cameras and points simultaneously.

1.Visibility Matrix: The visibility matrix is constructed when reading the data files. As we read the "matching" files for each image, we build a matrix of all points (typically around 10,000 in Unity Hall files) and mark a 1 for each camera column if the point is visible from that camera. This results in an n x m size matrix for n points and m cameras.

2. Bundle Adjustment: The Scipy least square optimizer is used in Bundle Adjustment, much like in previous sections.The visibility matrix helps to reduce the computational requirements by specifying which jacobians need to be computed and which can be skipped for faster processing.

During Bundle Adjustment, we aim to minimize the re-projection error, which measures the difference between the observed image points and the projected 3D points. This error is minimized by adjusting the camera poses and 3D points until the difference between the observed and projected points is as small as possible.The least square optimization method is an iterative process that minimizes the error by making small adjustments to the camera poses and 3D points in each iteration. The process continues until the error is minimized to an acceptable level. In addition to reducing the re-projection error, Bundle Adjustment also has the advantage of improving the accuracy and consistency of the 3D reconstruction results. It can also help to remove any outliers that may be present in the initial reconstruction.

Overall, Bundle Adjustment is an essential step in refining the 3D reconstruction results and obtaining accurate and reliable camera poses and 3D coordinates.

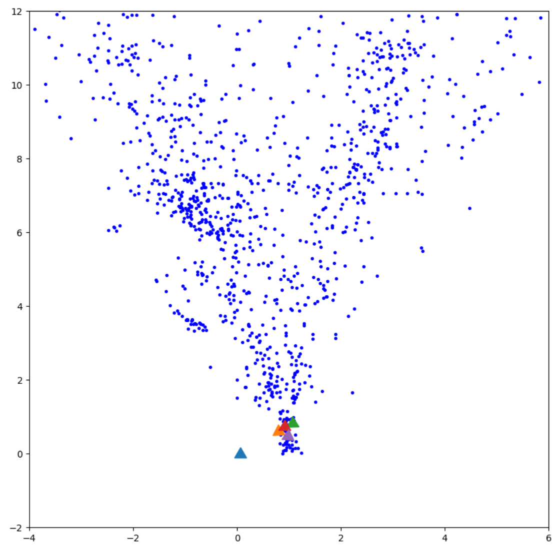

Fig. 9. Before Bundle Adjustment

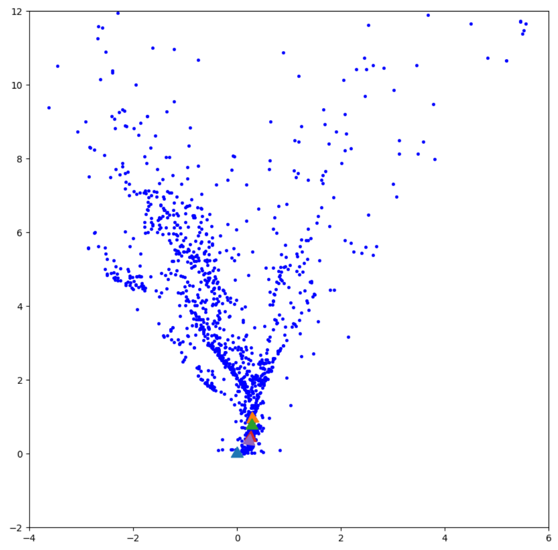

Fig. 10. After Bundle Adjustment

I. Phase 2: Deep Learning Approach

A. Introduction

NERF is an approach that lead to a new revolution in 3D scene representation, capable of synthesizing new views of complex scenes by optimizing the underlying continuous volumetric scene function with a sparse set of input views. This approach represents a scene using a fully connected network whose input is a single continuous 5D coordinate (spatial position: (x, y, z) and viewing direction: (θ, ψ)) and output is the volume density and the RGB pixel values at that viewing direction.

B. Input Data

Official Lego tiny NeRF data is used which is openly available on University of Berkeley website. In addition, x,y,z positions and θ, ψ directions are also included.

Even though NeRF is a deep-learning approach, it also involves classical approaches over the data before passing the input to the network.

C. Generation of Rays

Here, we use classical volume rendering technique where we consider any of the pixels in each image to be a ray in the 3D world.

First we use the conversion from pixel coordinates (u, v) to the normalized coordinates (X, Y, 1) with respect to the camera center.

The following equation describes a typical ray:

$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$

Where, '**o**' is the origin of the ray (the location of the pixel of the image with respect to the 3D world), '**d**' is the direction of the ray (unit vector from camera center to the image pixel), ’t’ is a parameter of the equation that is supposed to be continuous, however, it will be sampled at intervals to implement it in the program.

As we have the ray direction with respect to camera frame, we apply the rotation matrix corresponding to the rotation from camera to the world (obtained from the transformation matrix). This results in the ray direction obtained in the world coordinate. We further normalize it to a unit vector.

D. Point Sampling on the Ray

The points are sampled on the ray to obtain the values to be used as the input to our model. We have performed linear sampling, however, it works well with non-linear sampling as well. Also, we add a minimal noise is the sample positions so that the network is exposed to new data points thus, leading to better results.

E. Positional and Directional Encodings

If the coordinates of points are directly passed, the network gives very poor results, even if only a single image is passed. This is because the network has a tendency to learn specifically the low frequency features while ignoring higher frequencies. Thus, we use the sines and cosines of point coordinates (positional encodings) at different frequencies and pass them as the input to the network, instead of the point coordinates.

Similarly, instead of directional input, we utilize the sines and cosines of directional input at different frequencies, termed as Directional Encodings.

F. Volumetric Rendering

The outputs of the NeRF network are the RGB values and the volume density for the input location and the camera direction. The outputs from the network are fed into the volume rendering equation to obtain the color values at a given world point. The classical volume rendering equation is given below:

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

Where $\mathrm{T_i}$ are weights and $\mathrm{c_i}$ are colors.

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j) \tag{2}$$

$$\alpha_i = 1 - e^{-(\sigma_i \delta t_i)} \tag{3}$$

The transmittance till the given sample point is calculated using the density, followed by multiplying it with the RGB colors at that position to obtain the final RGB values in the image. Here is the exact formula for the same.

We perform the same process for all the sample points on all the rays, thus the name radiance field.

G. NeRF: Network

The actual network involves fully connected networks (Multi-Layer Perceptron) with the inputs being positional and directional encodings and the outputs being the density and the RGB values. The network involves skipped connection to obtain more accurate results. The network involves ReLU and Sigmoid activations.

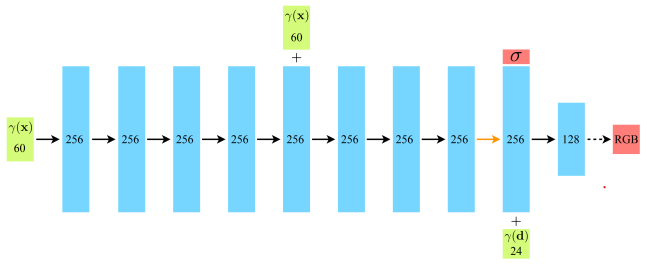

Fig. 11. NeRF architecture

However, we will only use a simple 5 layered network which takes both positional and directional encodings as input and the density and the RGB values as the output from the final layer, with ELU activations.

H. Network Parameters

For more accurate, we ran the network on the whole data, rather than splitting it into mini batches.

Input image size = 100x100

near distance = 2

far distance = 6

Depth samples per ray = 32

Learning rate = 0.005

Number of encoding function terms = 6

Loss function to be minimized: Log of mean-squared loss between predicted image (RGB values) and the actual image values

I. Results and Plots

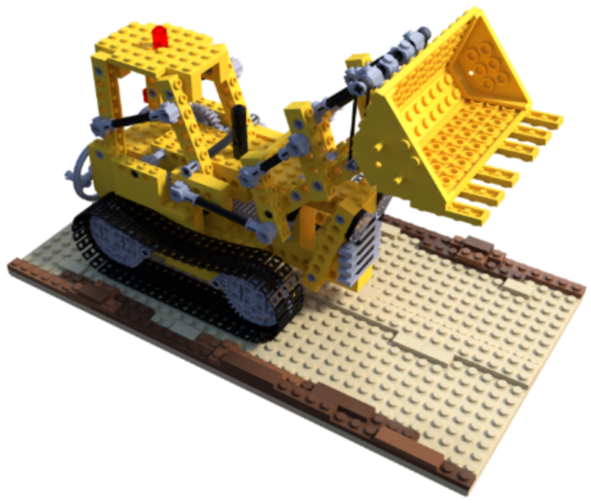

Fig. 12. Original Image

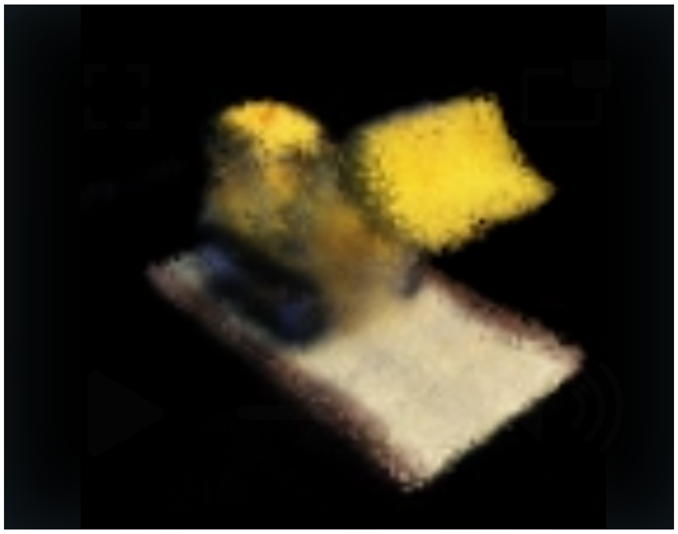

Fig. 13. Reconstructed 3D model

J. Conclusion

Considering the device constraints, the small sized network that we used and the relatively smaller sized images, we obtained good results and the 3-D model is visible. We created a the original NeRF architecure which works in the code, however, due to all these constrains, we used the smaller network as described earlier. If we were to use the actual network with complete network with the original images size, we could obtain sharp results closer to the actual one.

References

[1] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng (2020). NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. UC Berkeley, Google Research, UC San Diego.