Project 3
Buildings built in minutes - SfM and NeRF
used 11 late days

Aadhya Puttur, Alex Chiluisa

I. Introduction

In this paper we will be constructing 3D coordinates and scenes from 2D images. In phase 1 we will be implementing Structure from Motion (SfM), which is reconstructing a 3D scene and simultaneously obtaining the camera poses of a monocular camera w.r.t. of the given scene. We do this by obtaining images of different view points of one scene and finding coordinates that correspond to one another in each image using SIFT. Then we estimate camera poses using math from epipolar geometry in order to estimate the 3D coordinates from the 2d image points. We then use many optimization techniques to minimize the error from the estimation.

Write what we will be doing in phase 2

II. Structure from Motion Data

To start predicting camera poses of ever camera pose, we start with structure from motion with 2 views. In Fig 1,

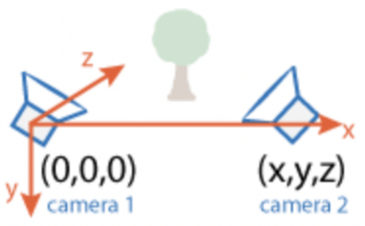


Fig. 1. Structure from Motion 2 views, (MathWorks)

we can see that structure from two stationary cameras, one camera will be considered as the origin. Camera 2 references camera 1's coordinates as the origin to determine its own coordinates by some arbitrary scale. The epipolar geometry is the intrinsic parameters which is known (K matrix), and the relative pose. The pre-steps to perform structure from motion is to first have a set of images to work with. These images are already distortion corrected and resized to 800 x 600 px. SIFT key-points and descriptors are used since SfM heavily relies on good feature matching and key point matching to find image correspondences. We have the rgb

Fig. 2. Images of Unity Hall used to perform SfM

values of each feature in each txt file corresponding to each of the 5 images, along with the coordinates of the image key-points of the corresponding features.

III. Fundamental Matrix

We use epipolar geometry to find the pose of the camera relative to the first camera

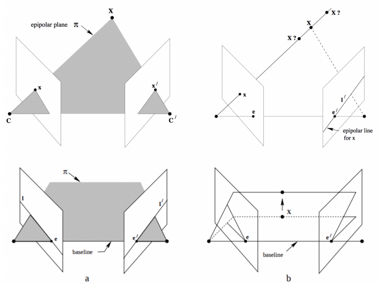


Fig. 3. Epipolar Geometry

The epipole is the point of intersection of the line joining the camera centers with the image plane, Fig 19. We apply this epipolar constraint to find a relationship of the correspondences of two images. The Fundamental matrix which is size by 3 x 3, it has 7 degrees of freedom. It's a rank of 2. We first find two column vectors in the plane that correspond such as x and $x^{'}$. The fundamental matrix is represented by the letter F. Fig 4 represents the relationship

$$\begin{bmatrix} x_i' & y_i' & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0$$

Fig. 4. $x'^T F x = 0$

between these column vectors. Every perfect correspondence between two column vectors of two images that face the same feature vector, equals zero every time.The null spaces of F are the epipole e and e′. This epipole line is passing through x and x′ respectively and is a 3 component vector.

A. Chiluisa is with the Department of Robotics Engineering, Worcester Polytechnic Institute, Worcester, MA 01609, USA (e-mail: ajchiluisa@wpi.edu)

A. Puttur is with the Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA 01609, USA (e-mail: aputtur@wpi.edu)

Therefore if we find all the epipolar lines of each image meaning a line from every x and x$'$ to e and e$'$ respectively, all lines will be passing through e and e$'$ Fig 4. We set up a homogeneous linear system of 9 unknowns and we are solving for m correspondences where $x_i'^T F x_i = 0$ and $i = 1, 2, ...m$. Although, we talked about how the fundamental matrix can be used, what specifically is it and how can it be calculated? We use our K, our intrinsic camera parameters to find F. We take the pinhole equation. We take this equation

$$Z \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = K \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Fig. 5. Pinhole Equation

Fig 5, and expand it by getting the world points using the pinhole equation for both images. We combine these world point equations from the two images We take the equation that $xEx' = 0$ where E is the essential matrix. $E = T_x R$ where R is the orientation of $x^{'}$ camera in x's frame. It is the 3 x 3 rotation matrix between the two cameras. T is the position of x's camera in x' frame. We take these two correspondence equations of the world point and feed it into $xEx' = 0$ and receive the following.

$$[x', y', 1] = K'^{-T} E K^{-1} \begin{bmatrix} x_{cm} \\ y_{cm} \\ 1 \end{bmatrix}$$

Fig. 6. The Fundamental matrix derivation is represented

We remove the Z because it is not zero it has to be greater than 0 because of the chierality constraint (depth positivity) From Fig 6, we had mentioned K are the intrinsic parameters but why are there 2 K's? Well since one camera is referencing another, the first camera will be represented as the origin camera where it's intrinsic parameters are 0s. In Fig 6, the fundamental matrix is represented as $K'^{-T} E K^{-1}$.

This is the Fundamental matrix that has 7 degrees of freedom because 9 unknowns (3 x 3) - 1 for Scale - 1 for Rank 2 = 7. However we are going to first ignore that it is rank 2 for now and estimate 8 parameters and then enforce the constraint of it being rank 2. Very similar to homography where each point corresponded to two constraints however where it is not similar in the F matrix each point contributes to only one constraint as the epipolar constraint is a scalar equation. This is also known as the eight-point algorithm. We summarize calculating the fundamental matrix below.

1) find x and x' correspondences using SIFT for all sets of images
2) Use the first two images to start calculating structure from motion
3) choose 8 random point correspondences to calculate F by stacking them Fig. 7
4) Find F through Ax =0 Fig. 7
5) Get the Singular Value Decomposition(SVD) of the output
6) F is the last column of V corresponding to least singular value
7) Take the SVD again
8) Enforce rank 2 by making the second output from SVD (SVD(F) = U, S, V) S to be 0 in order to zero out the last singular value.
9) Multiply U, S, and V to get the Fundamental matrix

$$\begin{bmatrix} x_1x_1' & x_1y_1' & x_1' & y_1x_1' & y_1y_1' & y_1' & x_1' & y_1' & 1 \\ ... & ... & .. & ... & ... & ... & ... & ... & ... \\ x_mx_m' & x_my_m' & x_m' & y_mx_m' & y_my_m' & y_m' & x_m' & y_m' & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix}$$

Fig. 7. Ax = 0, calculating the Fundamental matrix equals 0

Just to further explain step 7 and 8 where our output is U, S, V. U is an orthonormal basis thing, D is a diagonal matrix of singular values, and V is orthonormal of the transpose. Although we can't just go with any random 8 points to calculate a Fundamental matrix. We need to pick the 8 points that gives us a Fundamental matrix where the correspondence of the first and second image point with the fundamental matrix gives us zero $x'^T F x = 0$. Although since the correspondences were computed SIFT, we are bound to get noise and the relationship would always exactly be 0. Wr match outlier rejection by performing RANSAC. In Fig. 8

```
n=0;
for $i$ = $1{:}M$ do
    // Choose 8 correspondences, $\hat{x}_1$ and $\hat{x}_2$ randomly
    F = EstimateFundmentalMatrix($\hat{x}_1$, $\hat{x}_2$);
    $\mathcal{S} = \emptyset$;
    for $j$ = $1{:}N$ do
        if $| x_{2j}^T F x_{1j} | < \epsilon$ then
            $\mathcal{S} = \mathcal{S} \cup \{j\}$
        end
    end
    if $n < | \mathcal{S} |$ then
        $n = | \mathcal{S} |$;
        $\mathcal{S}_{in} = \mathcal{S}$
    end
end
```

Fig. 8. Algoritm for Get Inliers RANSAC Fundamental Matrix

we use the RANSAC algorithm to obtain a better estimate of the fundamental matrix by choosing the fundamental matrix with the maximum inliers. We then recalculate the Fundamental matrix with the inliers.

IV. Essential Matrix

The Essential matrix is another 3 x 3 matrix, but with some additional properties that relates the corresponding points assuming that the cameras obeys the pinhole model (unlike F). Previously, we observed the Essential matrix in Fig. 6 where F was represented as $F = K'^{-T}EK^{-1}$ although we did not use the intrinsic parameters or the Essential matrix to calculate F but only the correspondences. Now we can change the equation to get E where $E = K'^{T}FK$ or $E = T_xR$. This is where two random images in the world can be related by one equation where translation and rotation are coupled. We use this to disambiguate the two values from each other because the essential goal is to get the pose of the camera. Therefore we get the essential matrix using the intrinsic parameters and the optimized Fundamental matrix from the previous step.

V. Camera Poses

The camera pose consists of 6 degrees-of-freedom (DOF) Rotation (Roll, Pitch, Yaw) and Translation (X, Y, Z) of the camera with respect to the world.

$$E = UDV^T = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

Fig. 9. Singular Value Decomposition of Essential Matrix

The D has a eigen values of 1 (diagonal elements) and the last diagonal value is 0 because it has a rank 2 and it is not full rank. These values are 1 because we are using normalized camera coordinates when we are doing the essential matrix computation. This differs from Fundamental matrix because F is in pixel coordinates. We know from the Essential Matrix that $E = T_xR$ and we need T and R to get the camera pose. The left null space of E is the epipole of the second image or x' where $T^TE = T^T(t_xR) = 0$. Therefore, the translation vector is the third column of U when you take the SVD of E. We get two solutions for U because the sign can be flipped, U and -U. U is a orthogonal matrix, the translation vector has to be orthonormal to the first two columns of U so $T_x = [u_1xu_2]$

$$E = T_xR = U \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} U^TR = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

Fig. 10. Rotation and Translation

$$R = U \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T or.U \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T$$

Fig. 11. $USV^TTwosolutionsforS$

The translation and rotational vector is calculated in Fig. 10 where the rotation vector R is a SO(3) matrix. We then get rotation matrix from Fig. 11 and T = U_3 or T = $-U_3$. Therefore, there are 4 solutions:

1) $C_1 = U(:,3)andR_1 = UWV^T$
2) $C_2 = -U(:,3)andR_2 = UWV^T$
3) $C_3 = U(:,3)andR_3 = UW^TV^T$
4) $C_4 = -U(:,3)andR_4 = UW^TV^T$

since R has two solutions and T has two solutions We cannot have 4 different solutions, one of the solutions has to be valid.

VI. Triangulation

In triangulation, we are measuring 2 missing distances with a known angle. We can use the cheirality condition to select the best camera pose out of the four possible camera poses. We do this by triangulating 3D points given two camera poses. We will first use linear triangulation to get an initial estimate of the 3D world points. Given two camera poses, (C1,R1) and (C2,R2), and correspondences, x1 $\longleftrightarrow$ x2, we will triangulate 3D points using linear least square. Our first camera pose will always be the origin point so C1 and R1 will be default values for the reference camera.

$$P = KR[I_{3x3} - C]$$

Fig. 12. Camera Pose representation

$$\begin{bmatrix} x \\ 1 \end{bmatrix}_X P_x \begin{bmatrix} X \\ 1 \end{bmatrix} = 0$$

Fig. 13. Pinhole Projection Equation

The camera pose can be represented Fig. 12. This is the projection matrix. K represents the camera intrinsic parameters, R is the 3x3 rotational matrix, I is the identity matrix, and C is the 2 translation solutions that we discussed in part V. From this projection matrix we can use the relation where if 2 vectors are equal their cross product is 0.

Recall the pinhole equation, although this time using the same equation we will produce the pinhole projection equation Fig. 13. We perform the pinhole projection equation for both the first and second image. We then stack these equations in a 6 x 6 matrix. We then get the SVD of this matrix and the world point X is in the matrix V. We perform the calculation to get world point for every single pose of the corresponding two images. The rationale behind this is that we want to find what pose is the right pose. We perform this by using the chierality constraint.

The chierality constraint or the depth positivity constraint enforces that we choose the pose that has the maximum points in-front of both the cameras. We want the pose shown in Fig. 14 where the points are in front of the camera. WE do this by performing the chierality constraint equation Fig. 15. r is the third row of R (the rotation matrix), X is the world location, and C is the camera translation vector. We perform the chierality constraint for every capital X. The pose that outputs the maximum number of points is the correct pose. We perform this in DisambiguateCameraPose.py.

Once we get our camera pose, the X world points for the camera pose, as well as the image 1 and image 2 correspondences, we want to minimize the error for the X world points. This reason behind is because we have

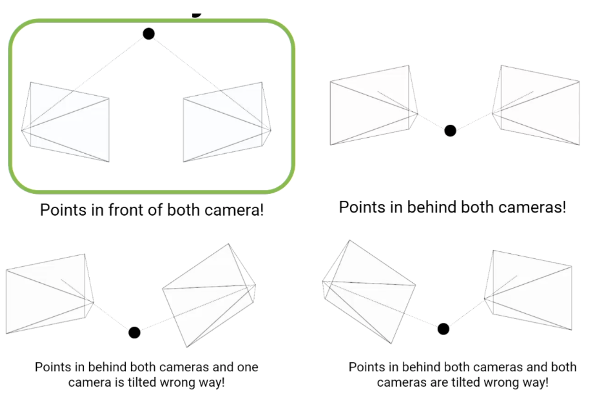

Points in front of both camera!
Points in behind both cameras!
Points in behind both cameras and one camera is tilted wrong way!
Points in behind both cameras and both cameras are tilted wrong way!

Fig. 14. Chierality, 4 different poses

$$r_3^T(X - C) > 0$$

Fig. 15. Chierality Constraint

only been minimizing error using algebraic error through using SVDs and minimizing the error of Ax-b. The problem with this is because algebraic error does not have geometric meaning as it does not really make sense physically. To counter this we will perform geometric error.

$$e_{geom} = ||\hat{x} - x||_2^2$$

$$e_{geom} = (\frac{P_1X}{P_3X} - x)^2 + (\frac{P_2X}{P_3X} - y)^2$$

Fig. 16. geometric error of re-projection and image coordinate

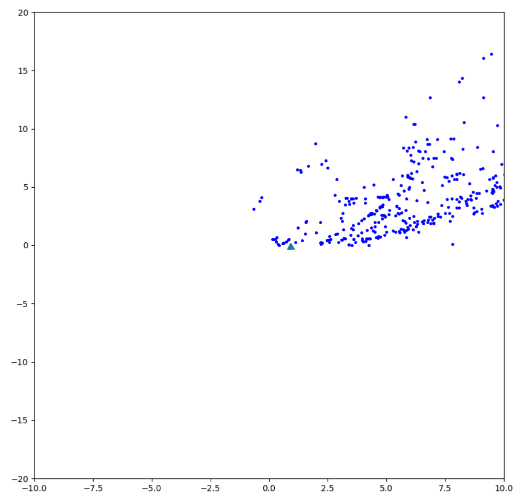
$$argmin_x||\frac{E_{geom,x}}{E_{geom,x'}}||_2^2$$

Fig. 17. optimization problem

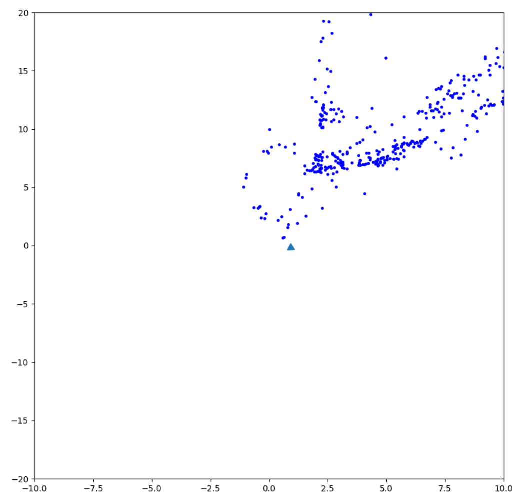
We perform geometric error by using our estimated capital X and re-projecting it back into the image. This is called nonlinear triangulation because we have some estimate of where the re-projection might be but we want to get a better estimate to have a better prediction of the world point. However our re-projection definitely will not be perfect because we take the forward projection and inverse projection. The use the projection form of the geometric error equation Fig. 16. We use this nonlinear equation to minimize. Refer to Fig. 12 for the projection matrix where one is identity and X is homogenized.

In nonlinear triangulation we perform the optimization problem shown in 17. We want to figure out the world point value of X such that error of both the images for all the points is minimized. We take the world point and re-project it on both images and we know what the pixel correspondences and we want to minimize the pixel error and we do this for every single point on the image.

We do this easily by using $scipy.optimize.leastsquares$ and the parameter that we are optimizing is the world point X.



(a)



(b)

Fig. 18. Linear Triangulation (a) vs Nonlinear Triangulation (b) of camera pose and 3D points projection

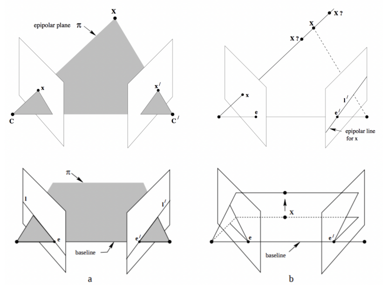


Fig. 19. Epipolar Geometry

VII. Perspective-n-Points

We have been talking about structure from motion of 2 views so far although now how do we incorporate more than 2 views? Now that we have done 2 views the rest of the images have some estimate of reference for their world X points.

Recall that we have the 2d to 3D correspondences of two images and now we want to get the camera poses for n images.

Now that we have estimated the camera pose of the new image, now we need to reject outliers in a linear PnP using RANSAC. We do this by measuring the re-projection error and we are trying to find the camera pose R, C that has a minimum re-projection error on the inlier set. Recall in part VI. we had to find the best camera pose and Fig. 8 where we used RANSAC to find the best inlier set. Similarly with the nth image and its image coordinate correspondences along with our estimated world points, we must find the correct camera pose for the nth image.

```
$n = 0$
for $i$ = 1:$M$ do
    // Choose 6 correspondences, $\hat{X}$ and $\hat{x}$, randomly
    [C R] = LinearPnP($\hat{X}$, $\hat{x}$, K);
    $\mathcal{S} = \emptyset$;
    for $j$ = 1:$N$ do
        // Measure Reprojection error
        $e = \left(u - \frac{P_1^T \tilde{X}}{P_3^T \tilde{X}}\right)^2 + \left(v - \frac{P_2^T \tilde{X}}{P_3^T \tilde{X}}\right)^2$;
        if $e < \epsilon_r$ then
            $\mathcal{S} = \mathcal{S} \cup \{j\}$
        end
    end
    if $n < |\mathcal{S}|$ then
        $n = |\mathcal{S}|$;
        $\mathcal{S}_{in} = \mathcal{S}$
    end
end
```

Fig. 20. Perspective-n-Points RANSAC

$$\lambda x = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Fig. 21. Linear Pinhole Camera Projection

We use PnP RANSAC Fig. 20 algorithm to estimate the best camera pose by choosing the maximum number of points in front of the error under a given re-projection error threshold. We choose 6 correspondences only because we are using a trivial version of PnP. We take these random 6 correspondences and feed them into LinearPnP. Recall that linear PnP minimizes algebraic error and nonlinear PnP minimizes geometric error. In linear PnP we get the projection values for small x and y. There are 11 number of unknowns in this equation because there are 12 values (3 x 4 matrix) - 1 (scale) = 11 actual unknowns. It is a 2 x 12 matrix for one point and we do this for n number of points. We perform the same procedure like we did for Part V. and we get the SVD of this matrix. Through this process of obtaining the rotation and translation vectors from P, we get the camera pose of the nth image using the intrinsic parameters K. A linear least squares system that relates the 3D and 2D points can be solved for (t,R) where $t = R^t C$.

Non-linear refinement of PnP minimizes the geometric error because due to the divisions and projection, the system is non-linear. Our goal is to minimize geometric error between measurement and projected 3D point.

$$argmin_{q,C} = \sum (x - \frac{P_{1,i}^T X_i}{P_{3,i}^T X_i})^2 + (y - \frac{P_{2,i}^T X_i}{P_{3,i}^T X_i})^2$$

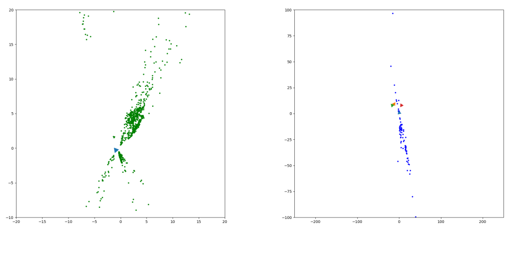
Fig. 22. Optimization problem Nonlinear PnP

Recall when we performed nonlinear triangulation in part VI although now we are estimating the camera pose of the new camera Fig. 22. We already have a map established from the first two images and we are basically trying to estimate the third image's camera pose with respect to that known map. Again like we did for non-linear triangulation we are using $scipy.optimize.leastsquares$ to obtain this minimization.

In Fig. 22 we are minimizing q so we went from a rotational matrix to a quartonion space. When we represent rotation as quartonian q it is much better to optimize because when we try to do jacobians later on it is much more efficient as it converges better.

VIII. Bundle Adjustment

Now that got poses and World point for the rest of n images and minimized the error for it, we can optimize even further by doing bundle adjustment. We have 4 cameras poses and we need to adjust the camera poses and 3D projection points based on the real image pixel coordinates.



(a) (b)

Fig. 23. 3D world points with camera poses, image 1 is a close up

We first create a visibility matrix in order to see which points are visible to each camera. We find the relationship between a camera and point, by constructing a I×J binary matrix, V where Vij is one if the jth point is visible from the ith camera and zero otherwise. Then with the given initialized camera poses and 3D points, we refine them by minimizing re-projection error using bundle adjustment Fig. 24. We specifically minimize the C, R in quartonian space, and X the world points.

$$\min_{\{C_i,q_i\}_{i=1}^{i},\{X\}_{j=1}^{J}} \sum_{i=1}^{I} \sum_{j=1}^{J} V_{ij} \left(\left(u^j - \frac{P_1^{jT}\tilde{X}}{P_3^{jT}\tilde{X}} \right)^2 + \left(v^j - \frac{P_2^{jT}\tilde{X}}{P_3^{jT}\tilde{X}} \right)^2 \right) \text{ where } V_{ij} \text{ is the visibility matrix.}$$

Fig. 24. Bundle Adjustment Optimization problem, Used Least Squares

Our structure from motion is complete. You can find the entire pipeline of the implementation in Fig. 25.

```
Data: Image Matches, $K$
Result: X, C, R
for all possible pair of images do
    // Reject outlier correspondences
    [x1, x2] = GetInliersRANSAC(x1, x2);
end
// For first two images
F = EstimateFundamentalMatrix(x1, x2);
E = EssentialMatrixFromFundamentalMatrix(F, K);
[Cset, Rset] = ExtractCameraPose(E);
// Perform linear triangulation
for $i$ = 1:4 do
    Xseti = LinearTriangulation(K, zeros(3,1), eye(3), Cseti, Rseti, x1, x2);
end
// Check cheirality condition
[C R] = DisambiguateCameraPose(Cset, Rset, Xset);
// Perform Non-linear triangulation
X = NonlinearTriangulation(K, zeros(3,1), eye(3), C, R, x1, x2, X0));
Cset = C, Rset = R;
// Register camera and add 3D points for the rest of images
for $i$=3:I do
    // Register the $i^{th}$ image using PnP.
    [Cnew Rnew] = PnPRANSAC(X, x, K);
    [Cnew Rnew] = NonlinearPnP(X, x, K, Cnew, Rnew);
    Cset = Cset∪Cnew, Rset = Rset∪Rnew;
    // Add new 3D points.
    Xnew = LinearTriangulation(K, C0, R0, Cnew, Rnew, x1, x2);
    Xnew = NonlinearTriangulation(K, C0, R0, Cnew, Rnew, x1, x2, X0);
    X = X∪Xnew;
    // Build Visibility Matrix.
    V = BuildVisibilityMatrix(traj);
    // Perform Bundle Adjustment.
    [Cset Rset X] = BundleAdjustment(Cset, Rset, X, K, traj, V);
end
```

Fig. 25. Algorithm Overview of SfM

IX. NeRF

NeRF is a method for constructing scenes through volumetric representation using images from different angles. NeRF utilizes a 5D coordinate system where $x = (x, y, z)$ and $d = (\theta, \phi)$ as input. Relying on light fields, it describes how light rays travel through x in every direction d. Figure 27 shows the rendering and training process. Then using positional encoding, it elevates the 3D coordinates of their scene into a higher dimensional frequency. Subsequently, it applies its hierarchical volume sampling to allow it to bias the sampling of the ray to encode as much information as it can regarding the object with the same samples.

A. Architecture

The overall architecture is shown in figure 26. NeRF relies on how its neural net takes the 5D coordinate inputs and generates a color (R, G, B) and density (γ) for each point. This 4D coordinate forms rays that intersect the object. By applying volume rending these values are composited into an image. Then the rendering loss of this image is calculated on a per-pixel basis. Using several viewing directions as inputs simplifies the generation of lighting fields. NeRF architecture is based on its neural net, and how its outcomes produce useful loss functions. By optimizing the network with the 2D image loss, we increase the accuracy of the network to predict images from several directions to generate a true shape representation of the object.

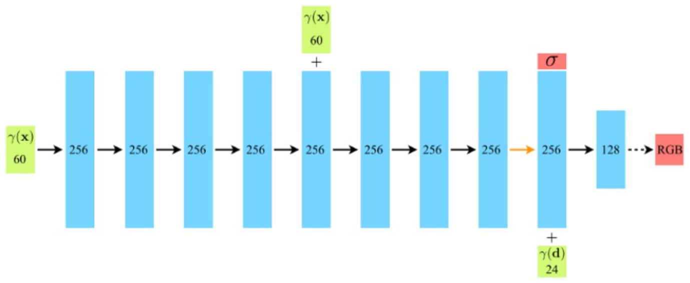


Fig. 26. The overall architecture of NeRF

B. Position Encoding

Employing position encoding on the coordinates before feeding the network, we can map the 3D coordinate into a higher dimensional space, enabling a better fitting of data with high-frequency variation leading to sharper models.

C. Hierarchical Sampling

By creating two separate networks, coarse and fine networks we can sample around regions with higher impact in the final rendering avowing sampling-free spaces or occluded regions that do not contribute to the final result. Therefore, the coarse network initially evaluated the rays using stratified sampling, where the ray was divided into N equally spaced bins, and a sample was uniformly drawn from each bin. The outcomes of the coarse network are then used as input of the fine network to converge ray samples along key points in the volume which biases the samples to sections with relevant content. Then, the final render color for the ray is computed by evaluating the fine network

In this work, we implemented and verify the work described in [1]–[4]. Huge challenges arise at the beginning of the implementation. Understanding the concepts of some of the sections of the paper, such as how the volume rendering utilizes the neural net to be fed, was challenging to follow. Additionally, the complex math and its code were a drawback in the attempt to implement our system. NeRF significantly reduces the gap to recreate 3D scenarios from 2D images by introducing viewing directions and providing more accurate lighting features. NeRF can represent finer details than previous work.

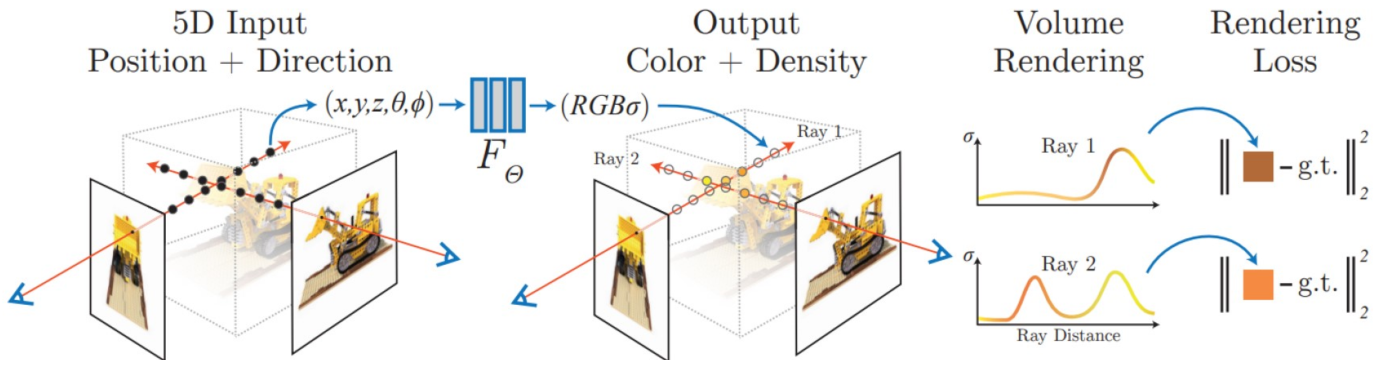


Fig. 27. The NeRF volume rendering and training process from the selection to the sampling points for every pixel in an image (left) through generation of individual pixel colors via volume rendering (right)

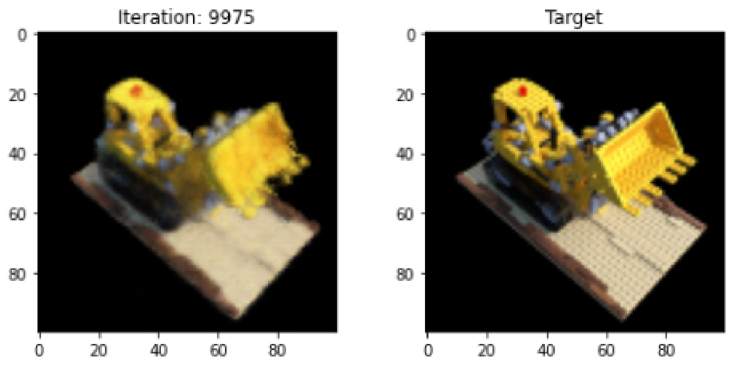


Fig. 28. Scene recreation and target lego

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.
[2] L. Yen-Chen, "Nerf-pytorch," https://github.com/yenchenlin/nerf-pytorch/, 2020.
[3] https://github.com/krrish94/nerf-pytorch.git.
[4] https://github.com/sakshikakde/SFM https://towardsdatascience.com/its-nerf-from-nothing-build-a-vanilla-nerf-with-pytorch-7846e4c45666.