VNU Journal of Science: Comp. Science & Com. Eng. Vol. 41, No. 2 (2025) 29–48

VNU Journal of Science:
Computer Science and Communication Engineering

Journal homepage: http://www.jcsce.vnu.edu.vn/index.php/jcsce

Original Article

An Efficient Spatial Grid Method for View Synthesis and Hidden Surface Reconstruction

Ma Thi Chau[1]*, Dao Viet Anh[2]

[1] *VNU University of Engineering and Technology, Hanoi, 144 Xuan Thuy, Cau Giay, Hanoi, Vietnam*

Received 23rd April 2025;
Revised 30th July 2025; Accepted 10th December 2025

Abstract: The article introduces a solution integrating ray casting in spatial grids with multi-layer perceptron models to synthesize novel views and reconstruct implicit surfaces. This method employs grid-based querying to enhance geometric reconstruction and overall model performance. It starts with 2D images captured from various viewpoints, which are transformed into new 3D views. The process begins with camera calibration to derive camera matrices. Rays are then extracted, including their origins, direction vectors, colors, and a specified region of interest in a spatial grid for 3D reconstruction. A ray-space projection technique samples space elements and queries signed distance function values representing surface geometry, along with colors from features stored at each spatial grid node. Density is computed based on signed distance values, and the ray color is synthesized from all elements along the ray, weighted by density and color. A Neural Radiance Fields model combined with a spherical Gaussian function models areas outside the region of interest to improve representation in unbounded scenes. Tested on the BlendedMVS and DTU datasets, the method demonstrates high performance in novel view synthesis and implicit surface reconstruction, achieving enhanced efficiency without compromising quality.

Keywords: View synthesis, hidden surfaces, NeRF, NeuS.

1. Introduction

3D technology is essential across various fields, including entertainment, healthcare, engineering, and education. However, 3D reconstruction faces challenges. It often requires significant investment in expensive hardware, making it inaccessible for budget-limited organizations. Additionally, the reconstruction process is complex, involving multiple stages from data collection to image recreation, which increases time and costs, limiting adoption in small and medium-sized industries.

A modern approach to 3D reconstruction from 2D images combines machine learning with traditional volumetric rendering techniques,

*Corresponding author.
E-mail address: chaumt@vnu.edu.vn
https://doi.org/10.25073/2588-1086/vnucsce.4909

exemplified by Neural Radiance Fields (NeRF) [1]. NeRF employs deep neural networks to simulate light and density in a 3D scene, reconstructing images from a set of 2D images taken from various views. Its key advantage is the ability to recreate images with high detail and quality, effectively handling complex surfaces and reflective lighting. However, NeRF's high computational demands present challenges for hardware-constrained systems, and it still struggles with accurately reconstructing object shapes. In contrast, Neural Implicit Surface Reconstruction (NeuS) [2] uses a deep learning model to not only recreate lighting but also simulate hidden surfaces of objects in 3D space based on the signed distance function (SDF). This enhances shape recovery, particularly for objects with surfaces not visible from certain views. NeuS can capture intricate surface details and hidden 3D structures, yielding improved results in scene reconstruction from 2D images. Nevertheless, NeuS also requires significant computational resources and long training times, which may limit its application in fast-paced environments. Additionally, solutions utilizing spatial grids have emerged as crucial for enhancing computational efficiency in 3D reconstruction models. Spatial grids effectively partition 3D space into smaller cells, facilitating intuitive modeling of objects and scenes. Techniques like Direct Voxel Grid Optimization (DVGO) [3] and Plenoxel harness spatial grid flexibility to accelerate 3D reconstruction without sacrificing quality. Despite these advancements, DVGO and voxel-based methods still face challenges related to memory optimization and processing capabilities in complex spatial scenarios or when high resolution is required.

In this article, a method for view synthesis and hidden surface reconstruction using spatial grid and two Multi-Layer Perceptron (MLP) models of color and signed distance function (sdf) is proposed. Our goal is to recover new views and 3D information from 2D images captured from multiple viewpoints while reducing the time and maintaining the quality. The central idea is to employ the benefits of grid-based querying alongside implicit surfaces to enhance geometric reconstruction and model performance.

The article is structured as follows: Section 2 reviews foundational knowledge relevant to the method's development. Section 3 details the proposed method for view synthesis and hidden surface reconstruction using spatial grid. Section 4 presents the practical implementation of the method and demonstrates its effectiveness and performance. Finally, Section 5 concludes the article.

2. Background and Related Works

2.1. Background

Volume rendering and Volume ray casting: Volume rendering is a key technique in computer graphics [4, 5] used to create images from 3D data, typically represented as scalar fields or volumetric data. Unlike traditional rendering methods, volume rendering effectively simulates complex internal structures, capturing details such as transparency and refraction. In view synthesis, this technique generates images from various viewpoints based on the same 3D data, eliminating the need for additional data collection and enhancing the accuracy of reproducing viewpoints from multiple input images.

Volume ray casting [6] is a widely used technique for processing three-dimensional volumetric data. It works alongside splatting and texture-based rendering, excelling at producing high-resolution images with excellent quality while simulating reflection, scattering, and refraction. However, it requires significant computational resources and time. The volume rendering process using ray casting involves four main steps: (i) Ray casting, (ii) Ray sampling, (iii) Shading, and (iv) Color compositing. Ray casting is the process of projecting a ray from the camera center through a pixel I_i on the image and into the volumetric space, traveling through the entire volume and exiting it. A ray r in space is characterized by its origin at the camera center o and its direction d:

$$\mathbf{r} : \mathbf{x}(t) = \mathbf{o} + t\mathbf{d} \tag{1}$$

where, t is the parameter of the line equation, satisfying the condition $t \geq 0$. Only consider points that lie in front of the origin o are considerd.

In addition to ray casting, ray tracing [4] and ray marching [7, 8] are fundamental methods for simulating light movement in three-dimensional space. Both methods used in volume ray casting model light with rays but differ in how they compute intersection points and sampling. Ray tracing represents a light ray as a straight line in 3D space and solves intersection problems by verifying geometric conditions. In volumetric environments, this complexity increases due to optical properties like density and absorption, often requiring approximations or optimizations, such as voxel division to limit checks. In contrast, ray marching samples along the light ray by taking small incremental steps through space, rather than finding exact intersection points.

$$\mathbf{x}_{i+1} = \mathbf{x}_i + s\mathbf{d} \tag{2}$$

where x_i represents the current position on the ray at step i, and s is the step size in the direction of the ray d. The ray marching method does not require precise intersection calculations; instead, it checks environmental information at the sampled points x_i.

Ray casting, ray tracing, and ray marching all emit rays from a source and process them along their paths, but they serve different purposes and are suited to various problems. Ray tracing provides high accuracy for well-defined objects but requires significant computational resources due to complex intersection equations. Ray marching is more versatile and effective in heterogeneous environments with continuous optical properties, but it needs careful management of parameters like step size and convergence threshold to maintain accuracy. Ray casting is the simplest and least resource-intensive method, often preferred in modern rendering algorithms, especially with advancements in machine learning.

Monte Carlo sampling [8] is a computational method based on probability theory used to solve complex integral problems through random sampling. It simulates light phenomena in volumetric environments by approximating calculations related to light and materials. Rather than using impractical analytical methods in heterogeneous environments, Monte Carlo sampling employs strategically distributed sample points to average the light function's value, approximating the integral. Each sample point evaluates light interactions influenced by absorption, emission, and scattering, with results combined to simulate global lighting effects. This method is particularly effective in environments with varying optical properties, making it ideal for applications like medical imaging and physical simulations. Monte Carlo sampling approximates the integral value by randomly selecting N points $x_1, x_2, \ldots, x_N$ from the domain D and calculates the light:

$$I = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i) \tag{3}$$

To enhance computational efficiency, this method often employs an appropriate probability distribution $p(x)$ for selecting sample points. In this context, the integral is approximated as follows:

$$I = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)} \tag{4}$$

The method, referred to as importance sampling, reduces variance by focusing more sample points in areas that significantly contribute to the integral. In volume ray casting, it serves as a crucial tool for managing complex environments with high precision, all while avoiding an unnecessary increase in the number of sample points.

Color compositing determines the final color of each ray as it passes through a volumetric medium. This method accumulates color values along the ray, using colors from the shading of sampled elements while simulating light absorption, emission, and transmission through the medium. Color compositing is typically achieved through continuous integration or approximated in a discrete form to manage lighting effects effectively. Mathematically, it involves solving optical integrals along the light ray's path, where the ray is sampled at specific points for approximation. The discretized formula for color compositing can be expressed as follows:

$$C \approx \sum_{i=0}^{N} c_i T_i \Delta t_i \tag{5}$$

where c_i is the color at the i_{th} sample point, T_i represents the transmitted value at that sample point, and Δt_i is the distance between consecutive sample points along the ray. The propagation function at the i_{th} sample point is also approximated using the discrete formula:

$$T_i = \prod_{j=0}^{i-1} \exp\left(-\tau_j \Delta t_j\right) \tag{6}$$

where τ_i is the absorption coefficient at the j_{th} sample point. This formula indicates that the transmitted value at each point depends on the cumulative absorption values accumulated along the ray's path. A key consideration in the discrete color compositing technique is the choice of sample distance Δt_i. This distance must be small enough to accurately simulate variations in the volumetric medium. If Δt_i is too large, important spatial details, such as regions with high absorption density, may be missed, resulting in inaccurate outcomes. Conversely, if Δt_i is too small, the number of samples that need to be computed increases significantly, which can degrade the overall performance of the algorithm. A common enhancement in color compositing is to actively accumulate emission and absorption intensities, rather than calculating each step independently. This can be accomplished by updating the accumulation directly at each sample computation step, for example:

$$C_{i+1} = C_i + c_i T_i \left(1 - \exp\left(-\tau_j \Delta t_j\right)\right) \tag{7}$$

where C_i represents the accumulated color value up to the i_{th} sample point. This formula facilitates efficient color compositing by reducing the number of necessary calculations, utilizing the values that have already been accumulated from previous steps.

2.2. Related Works

The Structure-from-Motion (SfM) Algorithm [9] enables the recovery of three-dimensional structures in a computer-generated space by utilizing the movement and acquisition of two-dimensional images of those structures. This fundamental problem in computer vision aims to create a 3D model of a scene from sets of 2D images captured from different views. The process involves several steps, including feature extraction, feature matching, geometric reconstruction, and global optimization, to ensure an accurate and consistent model. The initial step in the SfM algorithm is to extract features from the input images. These features typically include prominent points such as corners, edges, or high-contrast areas, as they are easily identifiable and can be matched across images. Common algorithms for this step are SIFT (Scale-Invariant Feature Transform) [10], SURF (Speeded-Up Robust Features) [11], and ORB (Oriented FAST and Rotated BRIEF) [12]. After feature extraction, the next step is to match features between pairs of images. This involves comparing feature vectors to find pairs of points with the highest similarity, often measured using

Euclidean distance. Pairs with a distance below a defined threshold are considered matches. However, due to potential noise or outliers in the data, the RANSAC (Random Sample Consensus) algorithm [13] is frequently used to eliminate incorrect matches. To ensure accurate reconstruction, it is crucial to identify image pairs with strong feature correspondences and sufficiently different viewing angles, thereby enhancing the stability of subsequent processing steps. Evaluation criteria for image pairs can be based on the number of valid matches or their spatial distribution across the images. The fundamental matrix F represents the geometric relationship between two images taken from different angles and is calculated from the matched point pairs, satisfying the epipolar equation, where x and x' are the coordinates of a point in the first and second images, respectively. The Eight-Point Algorithm or enhanced nonlinear methods are commonly used to compute F. The essential matrix E is derived from the fundamental matrix F. Then, it can be decomposed into the intrinsic matrix K and the extrinsic matrix $[R|t]$, where R is the rotation matrix and t is the translation vector. The intrinsic matrix K describes the internal parameters of the camera, such as focal length f, pixel size, and optical center. The extrinsic matrix P is composed of the rotation matrix R and the translation vector t, which together describe the camera's position and orientation in world space. Once the intrinsic and extrinsic matrices are established, the 3D coordinates of points in space can be calculated using triangulation. Let P_1 and P_2 be the projection matrices of two cameras, and x_1 and x_2 be the coordinates of a point in the corresponding images. The 3D coordinates X are computed by solving the equation:

$$A\mathbf{X} = \mathbf{0} \tag{8}$$

where A is constructed from P_1, P_2, x_1 and x_2. This system is typically solved using the Least Squares method. The final step in the process is global optimization through Bundle Adjustment, which aims to refine both the 3D coordinates of the points and the camera parameters to minimize the overall projection error:

$$\text{Minimize} \quad \sum_{i,j} \|\mathbf{x}_{ij} - \pi(P_j, \mathbf{X}_i)\|^2 \tag{9}$$

Where, $\mathbf{x}_{ij}$ is the observed pixel position, $\pi(P_j, \mathbf{X}_i)$ is the predicted position through the camera model, and $\|\cdot\|$ denotes the Euclidean norm. Common optimization techniques include Levenberg-Marquardt and gradient descent.

The original NeRF [1] represents a significant advancement in 3D rendering technology, utilizing a fully connected deep neural network to model a continuous function of 3D scenes. This model takes a 3D point and a viewing direction as inputs and produces corresponding color and density values as outputs, effectively transforming spatial data into visual representations. NeRF models shadowing, along with scattering, refraction, and reflection phenomena, using a function that maps 5D input - comprising the spatial coordinates of a point and the viewing direction - to RGB color and density.

$$F(x, y, z, \theta, \phi) \rightarrow (R, G, B, \sigma) \tag{10}$$

where, (x, y, z) represents the spatial coordinates, (θ, ϕ) is the view direction, (R, G, B) is the output color, and σ is the density at any point in space. The neural network employed is a fully connected network, also known as a Multi layer perception (MLP), which approximates the shadowing model along with the properties of scattering, refraction, and reflection phenomena. It is trained on a dataset of 2D images captured from multiple angles of the scene. Through this training, NeRF can reconstruct complex scenes with high levels of detail and accuracy, effectively handling challenges such as shading, reflections, and scattering that traditional rendering methods often struggle to address. A key aspect of the NeRF method is its rendering technique, which is similar to the volume rendering method discussed in the foundational knowledge section. Rays are emitted from the camera's viewpoint into the scene, with color and density sampled continuously along these rays. The neural network generates color and density values at every point in 3D space. Finally, the cumulative color along the ray is calculated using the discrete Monte Carlo propagation. Although discretized volume rendering introduces discrete errors, it remains effective for rendering complex scenes and is widely employed in practical implementations of NeRF. The optimization process in NeRF is enhanced by position embedding techniques, which improve the model's capacity to represent high-frequency details. This process transforms the input coordinates and viewing direction into a higher-dimensional space, enabling the neural network to capture fine details more effectively. The position encoding for a vector x is defined as follows:

$$\mathrm{PE}(x) = \Big(\sin(2^0\pi x), \cos(2^0\pi x), \dots, \sin(2^{L-1}\pi x), \cos(2^{L-1}\pi x)\Big) \tag{11}$$

Where L represents the number of frequency levels. The transform enhances the model's ability to learn spatial variation.

The training process for NeRF utilizes gradient descent algorithms to adjust the neural network's weights, minimizing the mean squared error between the colors generated by the model and the observed colors in the training images. This iterative optimization of the loss function allows the neural network to accurately map the 5D input to color and density values, producing realistic images of the scene.

One of the primary limitations of NeRF is its significant computational and time demands during optimization and deployment. For a digital image sized 500×500 pixels, and averaging 192 samples per ray through space raycasting, NeRF must recompute the MLP model for each sampled element. This means that during the processing of a single image, NeRF runs the MLP model up to $500 \times 500 \times 192 = 48,000,000$ times. Consequently, NeRF's speed is considerably reduced compared to traditional methods. Additionally, NeRF's MLP model has a two-stage structure. The first stage processes the spatial coordinates $\mathbf{x}(t)$ to compute the density σ and the features $\mathbf{F}_{\text{feat}}$, while the second stage predicts color based on $\mathbf{F}_{\text{feat}}$ and the viewing direction $\mathbf{d}$. As a result, NeRF requires substantial computational resources to optimize each stage. For each optimization process on a 2D input image dataset, NeRF typically takes between 20 to 40 hours to achieve optimal results, presenting a significant barrier to its practical application. Another challenge with NeRF is its ability to reconstruct geometry, particularly in accurately recovering the surfaces of objects in the scene. This limitation partly stems from NeRF's design, which primarily focuses on synthesizing new viewpoints. While simple solutions, such as setting density thresholds to define object surfaces, can be used, these methods often introduce unwanted noise, leading to fragmented and flawed object geometries.

The NeRF++ model [14] further refines the original NeRF framework by integrating additional features such as spatially varying reflectance and implementing auxiliary tasks to enhance training efficiency. The primary objective of NeRF++ is to improve the quality of generated images while ensuring robustness and operational efficiency. This model represents a critical step forward in the pursuit of high-quality visual rendering.

αSurf [15] represents a significant advancement in implicit surface reconstruction, particularly for semi-transparent and thin objects. Its innovative approach to decoupling geometry and opacity offers enhanced accuracy and robustness, making it a valuable contribution to the field. The authors introduce method designed specifically to handle the challenges posed by semi-transparent and thin objects. It emphasizes the separation of geometric and opacity information, leading to more accurate reconstructions. αSurf effectively separates geometric information from opacity, allowing for more nuanced representations of thin and semi-transparent materials. The method utilizes implicit surfaces, which provide flexibility and robustness in reconstructing complex shapes. By focusing on semi-transparent objects, αSurf addresses a significant gap in existing reconstruction techniques that often overlook these materials. The decoupling of geometry and opacity results in more precise reconstructions, particularly for challenging materials. The use of implicit surfaces enhances the robustness of the method against noise and irregularities in the data. The approach is applicable to a wide range of objects, from delicate fabrics to glass-like materials, expanding its utility in various fields. However, the method may incur higher computational costs, especially in real-time applications, due to the intricate processing of opacity and geometry. In addition, high-quality input data is essential for achieving optimal results, which may not always be available in practical scenarios.

Unisurf [16] represents a significant advancement in multi-view reconstruction, effectively merging neural implicit surfaces and radiance fields. Unisurf enhances both accuracy and efficiency, paving the way for future developments in 3D scene representation. The method offers improved representation and rendering of complex 3D scenes. Unisurf combines neural implicit surfaces and radiance fields, allowing for seamless integration of shape and appearance information. The framework ensures consistency across multiple views, enabling more accurate and coherent reconstructions. By leveraging implicit representations, Unisurf reduces the computational burden typically associated with traditional rendering techniques. The integration of implicit surfaces allows for more precise shape representations, improving the quality of the reconstructed scenes. The framework can adapt to various types of scenes, from simple objects to complex environments. Unisurf shows promise for real-time applications, making it suitable for interactive graphics and virtual reality. However, the unified approach may introduce challenges in training, requiring careful tuning of parameters to achieve optimal results. High-quality input data is crucial for effective reconstruction, which may limit its applicability in certain situations.

DVGO [3], developed by Cheng Sun and colleagues, represents a significant advancement aimed at overcoming the performance and computational time limitations of the NeRF model. DVGO takes a novel approach by replacing the fully connected MLP network in NeRF with a voxel grid interpolation method, enabling it to learn continuous spatial representations. This approach effectively reduces the required computational resources and accelerates processing speed while preserving the ability to reproduce details in three-dimensional space. DVGO enhances NeRF by prioritizing super-fast convergence in radiance field reconstruction. The authors introduce a DVGO method that streamlines the convergence process during NeRF training. The training paradigm is centered around optimizing the neural network using rendering loss, which ensures effective learning. DVGO is well-suited for a range of applications, including 3D modeling, computer graphics, and real-time rendering, where rapid and accurate results are essential. DVGO structures the space using two distinct voxel grids, each designed to store different types of information for the image reconstruction process. The first voxel grid is utilized to store the density values σ at each leaf node. The density at a spatial element is computed through trilinear interpolation based on the values stored in the grid, followed by the application of an activation function to ensure the data's usability. The second voxel grid in DVGO stores features related to light and color information at each leaf node. Similar to the density grid, these features are interpolated in three dimensions to compute values at the spatial elements. However, rather than directly using the interpolated values to determine color, DVGO combines them with the position $x(t)$ and the viewing direction d as input to a shallow MLP network. This MLP network is responsible for calculating the RGB color at each element. The final color of the ray is synthesized from the elements along the ray using an accumulation method asking to NeRF. Storing information on a voxel grid provides substantial advantages in terms of resource efficiency and computation time. Rather than running the MLP network for each element, DVGO requires only simple inference from the values stored at the leaf nodes of the grid, significantly reducing the number of computations needed. Moreover, by employing a shallow MLP instead of a fully connected MLP like in NeRF, the color inference process at each element is accelerated. Additionally, DVGO separates the calculations of density and color into two independent voxel grids. This approach not only minimizes dependencies between the data but also eliminates the need for a two-stage model as used in NeRF. Consequently, DVGO can directly optimize the stored values at each node of the voxel grid without relying on a pretrained MLP. However, like NeRF, DVGO is primarily focused on synthesizing novel views, which limits its effectiveness in recovering geometry.

Learning neural implicit surfaces by volume rendering for multi-view reconstruction (***NeuS***) [2], developed by Peng Wang et.al., is an advanced method aimed at improving geometric reconstruction capabilities in novel view synthesis models, specifically targeting the limitations of NeRF. Rather than directly learning the density values at each point in space, as NeRF does, NeuS employs an indirect approach through the SDF. This method enables NeuS to represent implicit surfaces more efficiently by leveraging the properties of the SDF to determine weights at spatial points, thus recovering detailed object geometry. The surface of the object is defined at the point where the SDF equals zero, $\mathrm{sdf}(\mathbf{x}(t^*)) = 0$, where $\mathbf{x}(t^*)$ represents the intersection of the ray with the surface. One of the key contributions of NeuS is the development of the weight function $w(t)$, which describes the influence of a point on the ray on the synthesized color, based on the SDF value. The weight $w(t)$ achieves its maximum at the surfaces, specifically at the position $t*$, and can be represented as a bell-shaped distribution. Common functions of this type include the Gaussian distribution, Laplace distribution, and the derivative of the Sigmoid function. In NeuS, a normalized form of the derivative of the Sigmoid function is employed to compute the weights. However, NeuS also needs to address the challenge of prioritizing surfaces that are closer to the camera when a ray intersects multiple surfaces. This is in line with the near-far law, which dictates that the influence of nearby surfaces should be emphasized more than that of distant surfaces. NeuS continues to employ a fully connected MLP, like NeRF, for computing color values, while the SDF values are learned through an optimization process. To ensure that the reconstructed surfaces are not noisy or exhibit significant artifacts, NeuS applies the Eikonal constraint, which guarantees that the SDF values conform to the properties of a distance function, with gradients maintaining a magnitude close to 1. This approach contributes to smoother and more accurate reconstructed surfaces. Despite these advancements in surface reconstruction capabilities, NeuS has some limitations. First, it still relies on NeRF's spatial rendering solution, which involves using a fully connected MLP, resulting in high computational resource and time demands. Second, in cases of low-density surfaces, such as mirrors or less textured areas, the weight $w(t)$ often fails to reach a clearly maximal value. This leads to biased learning of the SDF values, which do not accurately represent the necessary sign changes of the function, making it difficult for NeuS to reconstruct these surfaces accurately.

Plenoxels [17] presents an approach to rendering 3D scenes using a method that diverges from traditional neural network-based techniques. The authors introduce a technique that utilizes voxel grids to represent 3D scenes, eliminating the need for complex neural networks. This method aims to improve efficiency and simplify the rendering process while maintaining high-quality visuals. Plenoxels ultilize voxel grids, allowing for faster access and manipulation of 3D data compared to neural networks. The method demonstrates significant improvements in rendering speed, making real-time applications more feasible. Despite the simplified architecture, Plenoxels achieve high-quality outputs that are competitive with state-of-the-art neural rendering methods. The approach is more computationally efficient, which is crucial for applications in real-time rendering and interactive applications. By avoiding the complexity of neural networks, the method is more accessible for implementation and understanding. The technique shows robustness in various lighting and scene conditions, providing consistent results. However, the method may struggle with extremely large or complex environments. The resolution of the voxel grid can impact the detail of the rendered images, potentially leading to limitations in highly detailed scenes.

3. Spatial Grid-Based Synthesizing New Views and Reconstructing Hidden Surfaces

3.1. Proposal

An enhanced method, including two MLP models, is proposed (Fig. 1) to tackle the challenge of synthesizing novel viewpoints and reconstructing hidden surfaces in 3D space, utilizing a spatial grid. This method focuses on optimizing the reproduction of 3D scenes from a collection of 2D images captured from different views. The system's input consists of 2D images along with information about the camera positions and relevant parameters. The input data can be either masked or unmasked; for masked images, the model targets synthesizing new viewpoints and recovering hidden surfaces within the masked areas. Conversely, for unmasked images, the model must reconstruct both the background space behind the object and the scene within the RoI. The RoI refers to the area of the object's image. In the case of masked images, it specifically denotes the portion of the image that corresponds to the mask. The training process results in a spatial grid that retains features at the nodes, alongside two MLP models: MLP_{sdf} and $\text{MLP}_{\text{color}}$.

MLP_{sdf} consists of five hidden layers, each with 128 neurons and ReLU activation functions. The output is a scalar value representing the signed distance function (SDF). $\text{MLP}_{\text{color}}$ also contains five hidden layers with 128 neurons per layer. It outputs three RGB color values, normalized using the sigmoid activation function. The inputs to both models include positional encodings of spatial coordinates (with frequency level $L = 10$), the view direction (also encoded), gradients from the SDF field (only for $\text{MLP}_{\text{color}}$), and features interpolated from the spatial grid (denoted f_sdf-color). Both models are trained using the Adam optimizer with a learning rate of 5×10^{-4} and a batch size of 8192. This configuration enables fast training and accurate SDF and color prediction.

The model's output includes the synthesized novel viewpoint and the hidden surface represented by the sdf values. Processing starts with camera calibration using the SfM algorithm to analyze and extract key information from input data, including extrinsic and intrinsic

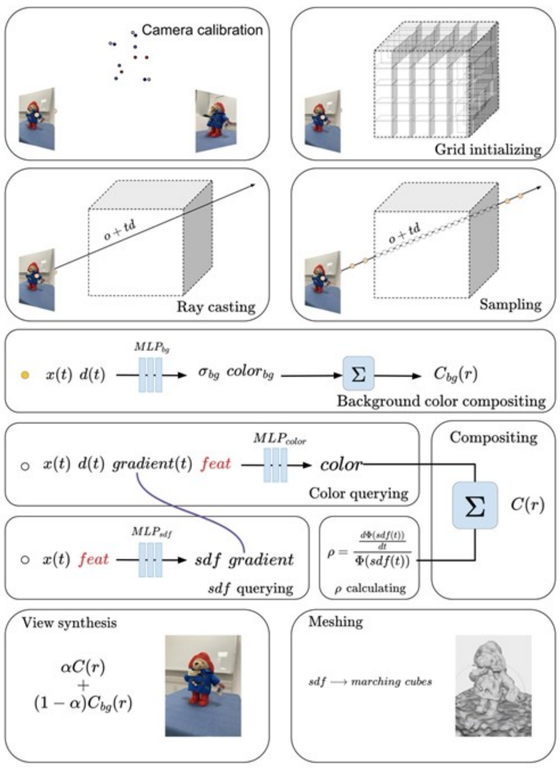

Figure 1. Our proposal.

matrices. These two matrices are essential for projecting from the 2D image space into 3D space, ensuring high accuracy in scene reconstruction. After the camera calibration step, the collected data is input into the main model for training. This phase involves optimizing the model's parameters, such as the density values and characteristics of the voxel grid, using deep learning algorithms. The voxel grid acts as the representation space, storing information about the density and properties of each spatial element. The spatial grid is updated by two MLP models: MLP_{sdf} and MLP_{color}. Rays are casted from the images into space using a ray casting algorithm. Points along these rays are then sampled using the Monte Carlo method. Information at each point is interpolated on the grid and fed into the MLP models to predict key parameters, including density σ for background points, sdf for points within the RoI, and color for each point. Thesdf values are converted into density and transmittance α ensuring adherence to the physical principles of the environment. A cumulative propagation function is subsequently applied to synthesize color from the discrete points. The error between the synthesized color and the ground truth, along with constraints such as the Eikonal equation [18] for signed distance sdf and mask errors, is calculated and used to optimize the voxel grid and the MLP models. Once the optimization process is complete, the model can perform two primary tasks: novel view synthesis and hidden surface reconstruction. To generate new viewpoints, the model leverages the learned information to predict color and transparency at each point in space, then synthesizes rays at specific angles to produce new 2D images. Concurrently, the marching cubes algorithm is employed to construct a polygonal mesh from the voxel grid. This approach enables high-detail reconstruction of hidden surfaces of objects in three-dimensional space, enhancing representation and ensuring accurate reproduction of the objects.

3.2. *Camera Calibration*

The input is a set of images of the same scene captured from various viewpoints. Each 2D image contains information about lighting and color from a specific view, providing essential data for constructing a 3D scene. This dataset is vital for enabling the model to learn the characteristics of the 3D space from the 2D images. SfM algorithm analyzes the relationships between these images by identifying common feature points, thus reconstructing a 3D point cloud that represents the scene. During this process, SfM computes and outputs the extrinsic and intrinsic matrices for each image. SfM not only generates these matrices but also reconstructs a sparse 3D point cloud that captures the key features of the scene from the set of images. After acquiring the point cloud from SfM, the next step is to crop the data to identify the RoI. This process aims to eliminate unnecessary parts in the 3D space, concentrating on areas that contain important information or objects that need reconstruction. The RoI is defined based on the spatial bounds of the point cloud, utilizing spatial cropping algorithms such as Bounding Volume Hierarchy (BVH) or Octree. This approach helps narrow the processing space, reducing computational resource requirements while improving the accuracy of hidden surface reconstruction. Data regarding the intrinsic matrix K and extrinsic matrix P obtained from SfM is then used to calculate the centers of the cameras in world space, as well as the direction d of each ray corresponding to each pixel. Specifically, the camera center o is determined from the extrinsic matrix:

$$o = -R^T t \tag{12}$$

The direction d of the rays is determined by the relationship between the coordinates of the image point on the image plane and the corresponding region in world space. For a pixel $p = (u, v)$, the ray direction vector is given by:

$$d = R^T K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{13}$$

Simultaneously, the correlations between the rays and the RoI are determined by back-projecting the pixels from the image plane into three-dimensional space. This approach focuses on the spatial areas that the rays from the camera traverse, minimizing the involvement of irrelevant regions. As a result, the necessary data is gathered efficiently, providing better support for the subsequent processing steps. The outcome of these processes is a collection of RoI, a 3D space that clearly defines the boundaries containing important features, along with a set of camera centers o and ray directions d projected into this space. This RoI is utilized to construct a spatial grid or voxel grid, offering clear guidance for the subsequent geometric reconstruction and new view synthesis steps. Together with the centers and directions of the rays, the RoI serves as a foundational element, bridging the information between the 2D and 3D spaces.

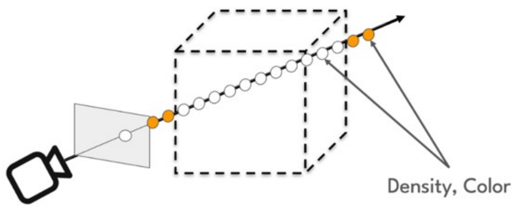

Figure 2. Ray casting.

3.3. Initializing The Spatial Grid and Performing Ray Casting

The spatial grid is utilized to represent values of interest, such as density, color, and features, at its leaf nodes, leveraging its efficiency. This results in a clear and visual structure that optimizes the process of querying information at any point in three-dimensional space through an interpolation function. The general formula for this interpolation function is expressed as follows:

$$interp(x, V) : (R^3, R^{C\times N_x\times N_y\times N_z}) \tag{14}$$

where x represents the position of the element to be queried, V denotes the spatial grid, C indicates the number of dimensions representing the value, and N_x, N_y,N_z, are the respective numbers of voxels along the x, y, and z dimensions. By default, three-dimensional linear interpolation is employed, ensuring high efficiency in computation and scalability.

A ray r in space is defined by the formula (1), which describes a straight line, where t is the parameter that determines the position of an arbitrary element $x(t)$ on the ray r (Fig. 2). To limit the space to be processed, the intersection of the ray with the space is calculated using the AABB (Axis-Aligned Bounding Box) intersection algorithm [19]. The values t_n and t_f represent the parameters at the two intersection points of the ray with the region of interest V. Once these values are known, the origin of the ray can be updated by replacing the camera center o with a new origin o', calculated using the following formula:

$$o' = o + t_n d \tag{15}$$

This restricts the parameter t to the range $0 \leq t \leq t_f - t_n$, optimizing sampling in space and concentrating on the region of actual interest. This approach not only reduces the volume of data that needs to be processed but also improves performance and accuracy when reconstructing the 3D scene.

The important sampling strategy of the proposed method is designed to balance minimizing computational costs with ensuring accuracy in spatial simulation. This approach emphasizes covering the entire region of interest while also focusing on detecting and sampling details in key areas. The process begins by uniformly sampling 64 points along each ray, with points evenly spaced in space. This initial distribution ensures comprehensive coverage, establishing a solid foundation for subsequent processing steps. Following the first step, the process continues with four consecutive rounds of focused sampling. In each round, the method queries the sdf values at the sampled points to identify areas with sign changes or significant gradients. These regions typically correspond to locations near hidden surfaces, where higher accuracy is essential during simulation. To enhance detail in these areas, an additional 16 sample points are selected and distributed around the marked regions. This approach ensures that regions with complex geometric features or potential surfaces are not overlooked during reconstruction. At the conclusion of the four iterations, the total number of sample points along each ray increases to 128, creating a denser point network that particularly targets high-importance areas. Each ray is defined by $t_i \in t_{i1}, t_{i2}, ..., t_{i128}$, with being the parameter on the t^{th} ray. For scenes with specific characteristics, such as images with infinitely extending backgrounds or environments with unrestricted depth, this strategy also incorporates an additional 32 sample points outside the main region of interest. The goal of this step is to ensure accuracy in synthesizing background colors while maintaining high quality in the overall spatial reconstruction. The combination of important sampling, and flexible processing for specific conditions makes this method both efficient and robust across various application contexts.

3.4. Background Synthesis

The space outside the RoI, known as the background space, will be processed separately using a simple MLP model to synthesize colors, ensuring both efficiency and accuracy in representation. In the background space, the MLP_{bg} model will be employed to simulate physical phenomena such as shadowing, scattering, refraction, and reflection (Fig. 3). The MLP_{bg} takes as input a 4-dimensional vector A' and maps from a six-dimensional space, which includes position and viewing direction information, to a four-dimensional space representing color and density. To minimize computational costs, the MLP_{bg} is designed as a single-stage model that directly predicts the density σ and the coefficient k of the Gaussian function.

$$\sigma, k = MLP_{bg}(x_{homo}) \tag{16}$$

In this context, x_{homo} refers to the coordinate normalized using homogeneous transformation. The color of each element in the background space is determined by the coefficient k, the basic components p, the parameter λ optimized during the learning process, and the viewing direction d

through the Gaussian function:

$$C(d) \approx \sum_{l=0}^{n} k_l G_l(d, p, \lambda) \tag{17}$$

With $G_l(d, p, \lambda) = e^{\lambda(d.p-1)}$. The function $G_l(d, p, \lambda)$ is a normalized Gaussian function, where $C(d)$ represents the color in the viewing direction d. The color in the background space is ultimately synthesized through an accumulation propagation function.

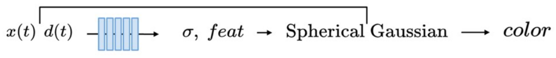

Figure 3. Back ground color and density.

3.5. New View Synthesis

Following the approach of the NeRF++ study, the RoI is encompassed within a sphere of radius $r = 1$. Within this area, the coordinates and properties of the rays are preserved and computed as usual. For elements located outside this spatial region, the coordinates will be transformed into homogeneous form to normalize the values to the range $[-1, 1]$. The MLP models (Fig. 4) consists of two primary components: MLP_{sdf}, which calculates the signed distance function, and MLP_{color} which determines the color at sampled points along the rays. The model's loss function comprises three main components: color loss, Eikonal loss, and mask loss. Each component adjusts the model's parameters to ensure accuracy and alignment with real-world data.

$$L = L_{color} + \lambda_1 L_{Eikonal} + \lambda_2 L_{mask} \tag{18}$$

where λ_1 and λ_2 are weighting factors, typically set to 0.1 to balance the contributions of each loss component. This formulation ensures that the model accurately reconstructs colors while maintaining geometric properties and clearly distinguishing the region of interest. *Color loss* is defined as the distance between the colors synthesized by the model along the rays and the actual colors in the image (ground truth). This distance is measured using the $L2$ norm (Euclidean distance) to ensure that the synthesized colors align closely with the real image. *Eikonal loss* ensures the geometric properties and consistency of the sdf. According to this principle, the magnitude of the gradient (normal vector) at every point in the SDF must equal 1. This condition helps ensure a continuous surface reconstruction and prevents the emergence of unwanted noise regions. *Mask loss* is calculated using a background separation mask, which classifies important areas in the image. Points marked as 1 in the mask represent the region of interest, where color contributes to the synthesized image, and the total weight on rays passing through this area sums to 1. Conversely, regions marked as 0 do not participate in the color synthesis process and are treated as transparent. The position encoding $PE(s)$ for a coordinate $x(t_i)$ or a viewing direction d is defined based on the formula (11). After encoding, the signed distance function value sdf_i and the gradient $gradient_i$ at the position $x(t_i)$ are computed. This is done by concatenating the position encoding at $x(t_i)$ with the feature retrieved from the spatial grid $feat_{sdf-color}$, and then passing it through the MLP_{sdf}. The output $gradient_i$ is a vector that serves as the surface normal at $x(t_i)$ while also determining the direction of variation of the sdf_i value.

$$sdf_i = MLP_{sdf}(stack(PE(x(t_i)), feats_{sdf-color})) \tag{19}$$

When utilizing the MLP_{sdf} model to predict the sdf value, the gradient of the sdf with respect to the input coordinates is also calculated. This gradient is crucial not only for defining the surface geometry to ensure smoothness, but it is also incorporated into the MLP_{color} model to compute color. The inputs to MLP_{color} consist of the position encoding $PE(x(t_i))$, the normal gradient $gradient_i$, the viewing direction d, and the feature $feats_{sdf-color}$. The computation process is as follows:

$$\begin{aligned} color_i'" = MLP_{color}(stack(PE(x(t_i)),\\ PE(d), gradient_i, feats_{sdf-color})) \end{aligned} \tag{20}$$

Finally, the Sigmoid activation function is applied to normalize the output value $color_i'$ to the range [0, 1], providing the color of the element at t_i.

$$color_i = Sigmoid(color_i') \tag{21}$$

This process not only guarantees accuracy in the computation of geometry and color but also optimizes the model for accurately simulating complex details in three-dimensional space. Incorporating the gradient at $x(t_i)$ as an additional parameter significantly improves the model's performance in simulating lighting phenomena such as scattering, refraction, and reflection, leading to more realistic results during the synthesis of new viewpoints.

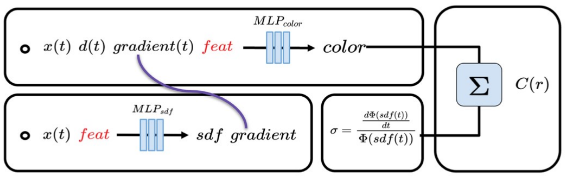
MLP_{color}
$x(t)$ $d(t)$ $gradient(t)$ $feat$
$color$
MLP_{sdf}
$x(t)$ $feat$
sdf $gradient$
$\sigma = \frac{\frac{d\Phi(sdf(t))}{dt}}{\Phi(sdf(t))}$
Σ
$C(r)$

Figure 4. RoI color and density.

3.6. Reconstructing Hidden Surfaces

After completing the model training process and the spatial grid, a detailed spatial representation of the object is obtained to be simulated. To convert the hidden surfaces in this representation into a polygonal mesh, the Marching Cubes algorithm is employed [20]. This algorithm facilitates the extraction of isosurfaces from scalar fields, transforming numerical values into a polygonal mesh that can be easily visualized on a computer. This process not only serves the purpose of visualization but also aids in the evaluation and analysis of the model. Specifically, during the evaluation of the polygonal mesh, the Marching Cubes algorithm on a grid with a resolution of 512 × 512 × 512 is applied using an isosurface threshold of 0. The nodes in this grid are assigned SDF values through linear interpolation queries on the spatial grid, as detailed in previous sections. The outcome of this process is a polygonal mesh that accurately represents the surfaces of the object in space.

4. Experiment and Evaluation

4.1. Datasets

The DTU dataset[1] is a key resource in 3D reconstruction research, collected in a controlled environment by scanning 124 diverse objects from multiple viewpoints with a precise camera and depth sensor. Its main advantages include high accuracy of depth information, which supports reliable evaluation of surface reconstruction models, as well as included lighting parameters and viewpoints for studying model performance under different conditions. However, because it was gathered in a controlled setting, the dataset may not fully capture real-world complexities like uneven lighting or unexpected obstacles.

The BlendedMVS[2] dataset is a synthetic resource designed for 3D reconstruction research, featuring 10 scenes of varying complexity. Each scene includes 20-50 images from different viewpoints, along with depth information and object masks that define areas of interest, improving surface reconstruction accuracy. A key advantage is its provision of both masked and unmasked versions, challenging models to distinguish objects from backgrounds without masks. This makes BlendedMVS valuable for evaluating algorithms under ideal and more realistic conditions where mask information may not be available.

[1]https://roboimagedata.compute.dtu.dk/? page_id=36
[2]https://github.com/YoYo000/BlendedMVS

4.2. Measurements

4.2.1. View Synthesis

Peak Signal-to-Noise Ratio (PSNR) is a widely used metric for assessing the quality of reconstruction in image and video compression. It measures the ratio between the maximum possible power of a signal (such as an image) and the power of corrupting noise that affects the fidelity of its representation.

$$PSNR = 10log_{10}(\frac{R^2}{MSE}) \tag{22}$$

Where, R is the maximum possible pixel value. MSE is the Mean Squared Error between the original and distorted images, calculated as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(I(i) - K(i))^2 \tag{23}$$

where, I is the original image, K is the degraded image, and N is the total number of pixels. Typical $PSNR$ values range from 20 to 50 dB, with higher values being better. A higher $PSNR$ value indicates better quality, meaning the reconstructed image is closer to the original.

Structural Similarity Index Measure (SSIM) is a metric used to assess the quality of images, particularly for comparing a reference image to a distorted one. SSIM measures similarity between two images based on three key components: Luminance, Contrast, and Structure.

$$SSIM = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{24}$$

Where, μ_x and μ_y are the mean luminance of the two images. σ_x and σ_y are the variance of the two images, σ_{xy} is the covariance between the two images. C_1 and C_2 are two small constants to avoid division by zero. SSIM values range from -1 to 1. A value of 1 indicates that the two images are identical. Values lower than 1 indicate differences between the images.

Learned Perceptual Image Patch Similarity (LPIPS) is a metric used to evaluate the perceptual similarity between images. LPIPS leverages deep learning to better align with human visual perception. LPIPS is designed to measure how similar two images appear to human observers, making it more relevant for applications in image generation and compression. It uses pre-trained CNNs to extract features from the images. The differences in these features are then used to compute a similarity score. LPIPS compares images in a learned feature space rather than the original pixel space. This allows it to capture higher-level perceptual differences. The LPIPS score is computed by extracting features from both images using a CNN, next calculating the Euclidean distance between the feature representations of the images, and then combining these distances with learned weights to produce a final similarity score. A lower LPIPS score indicates greater similarity between the images, while a higher score suggests they are more different.

4.2.2. Implicit Surface Reconstructing

Chamfer distance is a metric used primarily in the context of image processing and computer vision, particularly for evaluating the similarity between images or patterns. The Chamfer distance between two point sets, X and Y, is defined by the following formula.

$$C(X,Y) = \frac{1}{|X|}\sum_{x\in X}\min_{y\in Y}||x-y||^2 + \frac{1}{|Y|}\sum_{y\in Y}\min_{x\in X}||x-y||^2 \tag{25}$$

where $||x - y||^2$ denotes the Euclidean distance between points x and y, and $|X|$, $|Y|$ are the number of points in sets X and Y, respectively. The operation $min_{x\in X}|x-y|^2$ finds the shortest distance from a point x to any point in set Y.

4.3. Experiments

The camera calibration process in COLMAP starts with loading images and extracting SIFT features. After feature extraction, RANSAC is used to match features between images. From the matched points, the initial spatial structure is reconstructed by calculating the fundamental and essential matrices, which are then decomposed into extrinsic and intrinsic matrices. The output includes camera matrices and a point cloud, with RoI defined as a sphere of radius 300 around areas of high point cloud density.

Each feature in the spatial grid has specific dimensions, and the grid resolution is set to 100, creating a stable grid size. Grid node values initialize isosurface structures to aid in surface optimization and reduce noise. During coarse sampling, 64 points are initially sampled per ray, followed by four rounds of refinement that increase sampled points to 16 after each iteration. This ensures dense sampling in areas with varying sdf.

The input feature vector, $feats_{sdf-color}$, has 64 dimensions, augmented with positional encoding at frequency $L = 10$, resulting in a total input dimension. The model's hidden layers have input and output sizes of 128 dimensions, using ReLU activation before hidden layers. The output layer has a single dimension without nonlinear activation to maintain the sdf's sign.

Additionally, the model processes coordinate encoding at a frequency, view direction encoding, and the $feats_{sdf-color}$ vector. Thus, the model's input size is defined accordingly, with hidden layers again of 128 dimensions. The output layer is 3 dimensions, utilizing the Sigmoid activation function for RGB color channel values.

4.4. Evaluation

4.4.1. View Synthesis

The experiments on the BlendedMVS (Fig. 5) and DTU datasets were conducted without using masks, comparing our results to modern models such as NeRF [1], DVGO [3], Plenoxel [13](*Plens*), and NeuS [2]. The model was trained on a MIG 10G using an A100 GPU, demonstrating its efficiency in small configuration environments (10G VRAM). The model was evaluated using PSNR, SSIM, and LPIPS metrics after 2500 iterations (***i***), while also recording the loss value L during training. The PSNR results are presented in Table 1. The model converged quickly, achieving optimal results after 15 minutes of training, with a $PSNR$ of 27 dB after 15 minutes and 30 dB after 30 minutes, approaching the quality of current state-of-the-art models. Subjective visual assessments indicated that the synthesized images were of good quality, effectively distinguishing overall details; however, small details remained low quality or were lost in the synthesized images. Within the same timeframe, the convergence speed of our proposed method was nearly 20 times faster than that of NeRF and NeuS; the PSNR achieved in 30 minutes was comparable to NeRF trained for 15 hours and NeuS trained for 10 hours. However, the convergence speed was slower by a factor of 1.1 to 10 compared to the DVGO and Plenoxel methods, which utilize spatial grids. This slowdown is attributed to our method's calculation of indirect density through signed distance, where the optimization of the indirect density relies on the signed distance MLP_{sdf} model, significantly affecting convergence speed. Nevertheless, the background synthesis quality of the proposed method showed marked improvements over these other methods.

Table 1. PSNR ↑ on the BlendedMVS and DTU

S_{ID}	NeRF	DVGO	Plens	NeuS	Ours
Bear	26	30	26	26	29
Jade	22	28	25	22	29
Sculp	23	26	23	20	24
Stone	22	28	23	21	23
24	25	25	27	24	27
37	25	25	25	23	26
105	30	30	30	29	33
106	33	33	33	32	33
Time	20 hs	10 ms	11 ms	14 hs	40 ms
	300k ***i***	250k ***i***	250k ***i***	300k ***i***	100k ***i***

For the SSIM and LPIPS metrics, the results demonstrated significant performance without

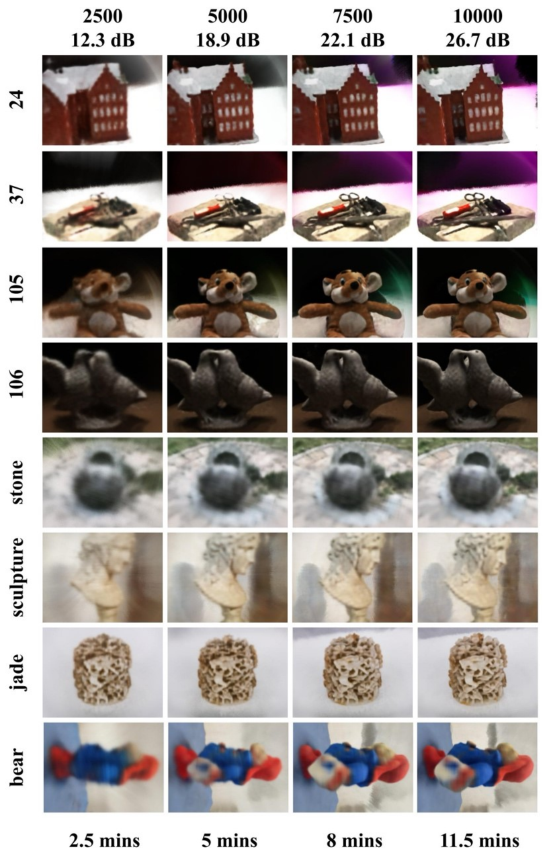

Figure 5. New view synthesis on DTU and BlendedMVS.

compromising quality, as detailed in Tables 2 and 3.

Table 2. SSIM ↑ on the BlendedMVS and DTU

S_{ID}	NeRF	DVGO	Plens	NeuS	Ours
Bear	0.15	0.03	0.05	0.11	0.04
Jade	0.26	0.08	0.11	0.27	0.08
Sculp	0.21	0.11	0.14	0.21	0.15
Stone	0.14	0.03	0.05	0.12	0.05
24	0.18	0.07	0.06	0.18	0.06
37	0.25	0.08	0.10	0.24	0.11
105	0.21	0.06	0.05	0.18	0.07
106	0.23	0.07	0.10	0.20	0.08

In the task of synthesizing novel views, performance on two datasets with and without masks is compared. The results indicate that using masks enhances training efficiency by 150% and execution efficiency by 110% on the BlendedMVS dataset, and by 160% and 130% respectively on the DTU dataset, as shown in Tables 4.

Table 3. LPIPS ↓ on the BlendedMVS and DTU

S_{ID}	NeRF	DVGO	Plens	NeuS	Ours
Bear	0.90	0.96	0.93	0.88	0.91
Jade	0.75	0.91	0.88	0.75	0.88
Sculp	0.80	0.88	0.87	0.79	0.86
Stone	0.86	0.93	0.92	0.86	0.90
24	0.75	0.85	0.80	0.72	0.89
37	0.79	0.88	0.89	0.78	0.88
105	0.82	0.89	0.84	0.80	0.89
106	0.87	0.91	0.92	0.83	0.93

Table 4. Performance comparison between with/without masks

S_{ID}	Training	time	Rendering	time
	Without	With	Without	With
Bear	10.10	15.10	20.31	18.10
Jade	10.15	15.11	20.13	18.05
Sculp	10.23	15.20	20.40	19.11
Stone	10.20	15.15	20.13	18.23
24	10.46	16.10	20.34	17.45
37	10.32	16.15	20.47	17.56
105	10.23	15.98	20.29	17.43
106	10.44	16.03	20.33	17.44

4.4.2. *Hidden Surface Reconstructing*

For the task of recovering hidden surfaces, the approach was evaluated using the DTU dataset without masks, as it provides realistic models essential for calculating the Chamfer distance metric. After every 10,000 iterations, the proposed method saves the model state and performs an evaluation using the Chamfer

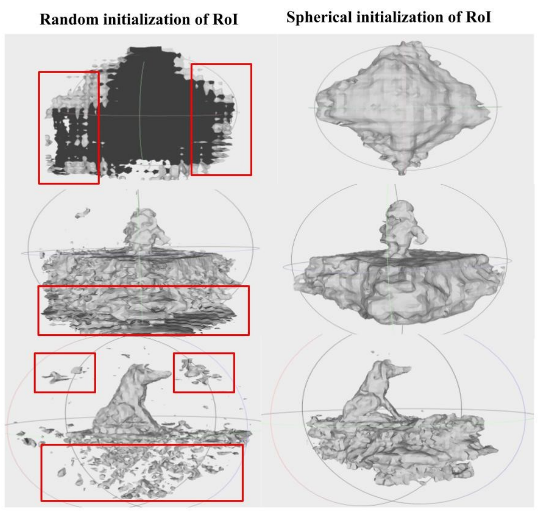

Figure 6. Comparison of Random and Spherical initialization of RoI.

distance on the realistic models from the DTU dataset. Using the spherical initialization of the region of interest, artifacts in the reconstructed 3D models are significantly reduced (Fig. 6). In the case of random initialization, the 3D model displays numerous noise surfaces during the first 10,000 iterations. These surfaces appear in a grid-like and random pattern, concentrating near the object and in regions with limited image information or detail during the optimization process. Furthermore, the signed distance field reveals many artifact values along the coordinate axes of the grid, resulting from errors in the grid and position encoding. Conversely, with spherical initialization prior to training, the level of noise in the initial 10,000 iterations is significantly reduced, leading to an improved convergence rate for the model. By orienting the parameters through spherical surface initialization, the training process becomes more stable, thereby minimizing the occurrence of artifacts in the signed distance field. The Chamfer distance of our method with modern models, including NeRF [1], α Surf [15], UNISURF [16], and the traditional COLMAP method are compared. The results are presented in Fig. 7 and Table 5. The model exhibited rapid optimization after just 15 minutes of training. It quickly adapted to the data, approximating the object to be reconstructed within the first 5,000 iterations. In the following iterations, the number of noise planes decreased significantly, resulting in smoother surfaces. The Chamfer distance measure recorded after 30 minutes of training was 1.75, which is comparable to the traditional method COLMAP, which takes 60 minutes. Consequently, within the same timeframe, the optimization speed of the proposed method is 14 to 20 times faster than NeuS. The results indicate that the proposed method significantly outperforms NeRF, UNISURF, and COLMAP, achieving quality on par with NeuS and α Surf.

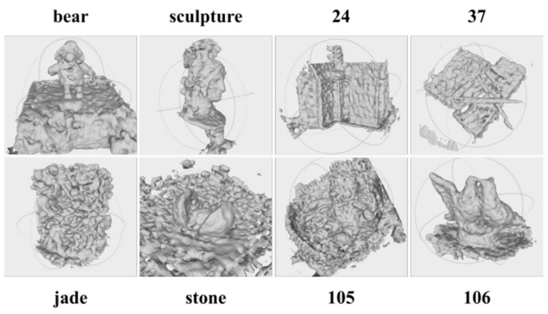

Figure 7. Implicit surface restoration on Blended MVS and DTU.

Table 5. Chamfer distance on DTU

S_{ID}	NeRF	Unis	NeuS	$\alpha Surf$	Ours
24	1.9	1.32	1.00	1.11	1.05
37	1.6	1.36	1.37	1.50	1.43
105	1.07	0.89	0.83	0.85	0.80
106	0.88	0.59	0.52	0.48	0.52
110	2.53	1.47	1.20	1.12	1.21
114	1.06	0.46	0.36	0.35	0.35
118	1.15	0.59	0.49	0.50	0.48
122	0.96	0.62	0.54	0.50	0.49

Visually, the surface quality of the proposed method is smooth like NeuS, surpassing α Surf, but is somewhat less effective than α Surf when handling thin or transparent surfaces.

5. Conclusions

The article introduces a novel method for synthesizing new views and reconstructing hidden surfaces. This approach enables the extraction of 3D information from a collection of 2D different view images. The problem addressed involves reconstructing a 3D model of a scene from these images, which are taken from different views.

The processing is divided into several stages, starting with the preprocessing of the 2D image data. A SfM framework, is utilized to analyze and extract information from the input images. After processing, camera matrices containing essential details like position, orientation, and optical parameters are obtained. From these matrices, the space is segmented into light rays, each defined by an origin, a direction vector, and a color. A RoI is also identified, focusing the 3D recovery process. These rays are projected into the space using ray casting, which determines the position and value of elements in the 3D environment. For each spatial element, the values of the sdf and color (c) are computed by querying features stored on the nodes of the spatial grid and shallow MLP models.

The resulting synthesis is used to compute the loss function and optimize the model. The method has been tested on datasets DTU and BlendedMVS, demonstrating high performance in both key tasks: synthesizing new views and reconstructing hidden surfaces. This approach significantly improves processing time and computational resource efficiency while maintaining high-quality results.

Acknowledgments

This work has been supported by VNU University of Engineering and Technology under project number CN24.14 ”Solution of 3D floor plan generation”.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, R. Ng, NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis (2020). arXiv:2003.08934.
URL https://arxiv.org/abs/2003.08934

[2] P. Wang, L. Liu, C. T. Yuan Liu, T. Komura, W. Wang, NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction (2021). arXiv:inarXiv preprint arXiv:2106.10689.

[3] S. C. Sun, M., Chen, H.T., Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction. (2022). arXiv:In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.pp. 5459–5469.

[4] J. Kajiya, B. V. Herzen, Ray Tracing Volume Densities (1984). arXiv:In ACM SIGGRAPH Computer Graphics: 18, pages 165–174.
URL https://doi.org/10.1145/964965.808594.

[5] N. L. Max, Optical Models for Direct Volume Rendering (1995). arXiv:). “Optical Models for Direct Volume Rendering”. In IEEE Trans. Vis. Comput. Graph.: 1, pages 99–108.

[6] K. Engel, M. Hadwiger, J. Kniss, C. Rezk-Salama, D. Weiskopf, In Real-Time Volume Graphics (2006). arXiv:A K Peters/CRC Press, pages 163–185. ISBN: 9781439864296.
URL https://doi.org/10.1201/b10629

[7] K. Z. et.al., Real-time Smoke Rendering using Compensated Ray Marchin (2008). arXiv:In ACM Trans. Graph.: 27.
URL https://doi.org/10.1145/1399504.1360635

[8] A. B. Owen, Monte Carlo Theory, Methods and Examples (2013).
URL https://artowen.su.domains/mc/.

[9] J. L. Schonberger, J.-M. Frahm, Structure-from-Motion Revisited (2016). arXiv:In Conference on Computer Vision and Pattern Recognition (CVPR).

[10] D. G. Lowe, Object Recognition from Local Scale-Invariant Features (1999). arXiv:In Proceedings of The Seventh IEEE International Conference on Computer Vision: volume 2. Ieee, pages 1150–1157.

[11] T. T. Herbert Bay, L. V. Gool, SURF: Speeded up Robust Features”. In Computer Vision (2006). arXiv:ECCV 2006: by editor Ales Leonardis, Horst

Bischof and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, pages 404–417. ISBN: 978-3-540-33833-8.

[12] E. Rublee, V. Rabaud, K. Konolige, G. Bradski, ORB: An Efficient Alternative to SIFT or SURF (2011). arXiv:In 2011 International Conference on Computer Vision: pages 2564–2571.
URL doi.org/10.1109/ICCV.2011.6126544

[13] M. A. Fischler, R. C. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography (1981). arXiv:In Commun. ACM: 24.6, pages 381–395. ISSN: 0001-0782.
URL https://doi.org/10.1145/358669.358692

[14] Zhang, K., Riegler, G., Snavely, N., Koltun, V, Nerf++: Analyzing and Improving Neural Radiance Fields. (2020). arXiv:. arXiv preprint arXiv:2010.07492.

[15] T. Wu, H. Liang, F. Zhong, G. Riegler, S. Vainer, J. Deng, α Surf: Implicit Surface Reconstruction for SemiTransparent and Thin Objects with Decoupled Geometry And Opacity (2023). arXiv:inarXiv preprint arXiv:2303.10083.

[16] M. Oechsle, S. Peng, A. Geiger, UNISURF: Unifying Neural Implicit Surfaces and Radiance Fields for Multi-View Reconstruction (2021). arXiv:In ICCV.

[17] A. Yu, S. Fridovich-Keil, M. Tancik, Q. Chen, A. K. Benjamin Recht, Plenoxels: Radiance Fields without Neural Networks (2022). arXiv:In CVPR.

[18] I. V. R. R. Z. Zhdanov, W. I. Fushchych, On The General Solution of The D'Alembert with a Nonlinear Eikonal Constraint And Its Applications (1995). arXiv:In Journal of Mathematical Physics: 36.12, pages 7109–7127.
URL https://doi.org/10.1063/1.531142.

[19] A. Majercik, C. Crassin, P. Shirley, M. McGuire, A Ray-Box Intersection Algorithm And Efficient Dynamic Voxel Rendering (2018). arXiv:inJournal of Computer Graphics Techniques (JCGT): 7.3, pages 66–81. ISSN: 2331-7418.
URL http://jcgt.org/published/0007/03/04/

[20] W. Lorensen, H. Cline, Marching Cubes: A High Resolution 3D Surface Construction Algorithm (1987). arXiv:In ACM SIGGRAPH Computer Graphics: 21.
URL https://doi.org/10.1145/37401.37422.