Learning View Selection for 3D Scenes

Yifan Sun[1,2] Qixing Huang[1] Dun-Yu Hsiao[2] Li Guan[2] Gang Hua[2]
[1]The University of Texas at Austin [2]Wormpex AI Research
{yifan,huangqx}@cs.utexas.edu {dunyu.hsiao, li.guan}@bianlifeng.com ganghua@gmail.com

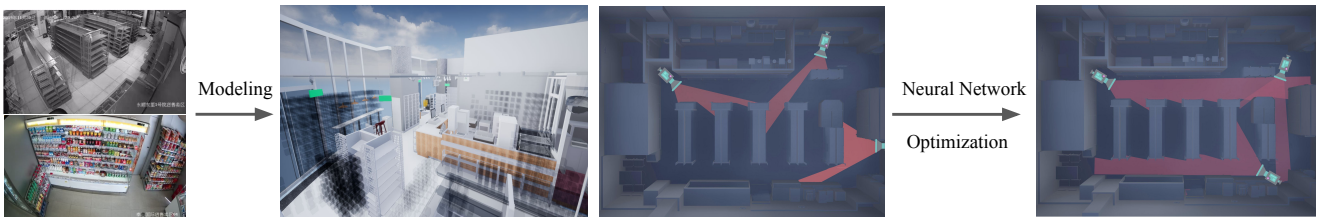


Figure 1: We present a method that optimizes max-coverage camera placement based on learning. Given the real scene model with the areas of interest (marked as black blocks) that require to be covered by a set of cameras, we learn a continuous scoring function and apply continuous optimization on camera poses to improve the global coverage score. In this figure, the red area in the top view of a scene indicates that the coverage increases after optimization.

Abstract

Efficient 3D space sampling to represent an underlying 3D object/scene is essential for 3D vision, robotics, and beyond. A standard approach is to explicitly sample a dense collection of views and formulate it as a view selection problem, or, more generally, a set cover problem. In this paper, we introduce a novel approach that avoids dense view sampling. The key idea is to learn a view prediction network and a trainable aggregation module that takes the predicted views as input and outputs an approximation of their generic scores (e.g., surface coverage, viewing angle from surface normals). This methodology allows us to turn the set cover problem (or multi-view representation optimization) into a continuous optimization problem. We then explain how to effectively solve the induced optimization problem using continuation, i.e., aggregating a hierarchy of smoothed scoring modules. Experimental results show that our approach arrives at similar or better solutions with about 10 x speed up in running time, comparing with the standard methods.

1. Introduction

Computing the multi-view representation of an underlying 3D object/scene is a fundamental problem in 3D vision and beyond. The desired output shall use a small set of views to maximize a pre-defined scoring function that involves objectives such as coverage of the underlying scene and viewing constraints among salient regions. A standard approach is first to densely sample a set of views and then formulate a generalized set cover problem, e.g., by optimally selecting a subset of views to maximize the scoring function. However, this approach usually incurs significant computational overheads due to the cost of rendering and solving the induced optimization problem.

In this paper, we introduce a novel approach that avoids explicitly sampling a dense set of views. Our approach is motivated by training view prediction networks that take a small collection of views as input and outputs predicted views that correspond to other camera poses. Specifically, our approach approximates the scoring function using a neural network that combines a view prediction network and a scoring module.

The view prediction takes a 3D model and a camera pose as input and outputs a concise representation under that view. The scoring module then takes the view predictions under multiple camera poses as input and outputs an approximation of the pre-defined score. The entire network is trained end-to-end. This approach allows us to turn the difficult multi-view representation optimization problem into a simple continuous optimization problem. In particular, our approach avoids explicitly sampling rendered images (See Figure 1).

Our approach is motivated by the fact that although a rendered image may contain rich geometric details of the underlying scene, the visibility pattern, which affects the coverage problem's results, has low-dimensional structures. A 3D scene usually consists of primitive geometric shapes, and contours of shapes (e.g., sharp edges of box-shape objects) mostly capture the visibility patterns. Recent advances in representations of 3D models and neural network models make it possible to learn representations of visibility patterns

to approximate the scoring function's loss surface.

Specifically, our approach represents a 3D scene using a volumetric representation. The view prediction network combines a scene encoder, a camera transformation encoder, and a coverage estimator decoder. We also design a separate scoring module, which outputs the coverage score for a specific camera configuration. The resulting network allows us to optimize an initial camera configuration by continuously optimizing the camera configurations. This procedure is efficient and effective.

Our learning-to-optimize formulation made two major contributions: (1) we turn a discretized multiple-view selection problem into a continuous optimization problem by training a deep neural network that approximates the coverage scoring function; (2) we provide the solver to find the optimal camera poses given the formulation with much less computational cost. We have evaluated our approach on a benchmark dataset through greedy sampling over the multi-view camera arrays in 339 high-resolution models of real scenes. The evaluation results show that we achieve similar coverage results comparing to greedy methods on dense sampling grids with about 10× speed. Our method also outperforms other fast baseline methods within similar time budgets.

2. Related Works

The problem of optimizing a set of views to cover a 3D model is related to a variety of problems in computer vision, computer graphics, robotics, and combinatorics optimization. In this section, we review relevant works in camera placement, path planning, and view prediction, which are most relevant to this paper's context and the proposed approach.

2.1. Camera Placement

The camera placement problem originates from the "Art Gallery" problem [7] in computational geometry. Given a 2D map, it aims to design an optimal static surveillance system to completely cover the map. Directional cameras with a pyramidal field of view and omnidirectional cameras that can capture a spherical area around are two standard camera models for this problem.

Many works on optimizing camera placement project a 3D scene to 2D polygons when computing the covered area for simplicity [13, 24, 25, 29, 12], where the field of views are reduced to triangles and circles, and check the feasibility of camera placement back in the 3D scene. With the increment of computing power and availability of realistic 3D models, Beker et al. [4] extended the coverage problem to the real 3D setting, where the height of objects and their places are evolved. The works in [2, 1] further generalized camera placement optimization methods for 2D maps to compact 3D space.

A standard approach to compute visibility maps of scenes is to discretize them as voxels [10]. The cameras are sampled from a spatial grid. Under this formulation, coverage maximization is turned into a set cover problem (SCP), where a set consists of voxels visible from one camera sample. Both integer programming and greedy methods are applied for solving SCP.

Fan *et al*. [11] conducted a comparison on both optimality and time efficiency among different approximating methods. Although greedy methods prove to be a good approximation both theoretically [8, 34] and in practice with polynomial complexity, we can only afford to compute visibility between each camera-voxel pair with a coarse discretization. Zhao *et al*.[37] proposed the idea of an iteratively refined dynamic uniform grid for camera sampling to guarantee feasibility, and we improve this technique for accelerating visibility computation.

2.2. Path Planning

The problem of path planning is closely related to structure from motion (SfM), where the path's length is a vital factor of the utility function for 3D scanning and reconstruction. Early works mainly focus on scanning a single 3D object with prior knowledge about its size and approximate position. The next best view (NextBest) strategy [3] is applied to maximize the coverage of the 3D object with minimum numbers of views for the expectation of path length to be small.

As NextBest is determined based on partial temporal observation of the objective, the two highlighted heuristic methods, voxel occupancy [9, 21] and occlusion edge [22, 23, 28] make the simplest predictions in geometry on the unobserved part to proceed. Therefore steps taken are conservative to restrict viewpoints close to the surface of the objective of scanning.

Sequeira and Goncalves [32] extended the occlusion methods to the scanning of large indoor scenes and allowed the view space to have high degrees of freedom. The definition of NBV requires the utility of every novel view to be updated after each step. Therefore, only local path planning is considered, and there is no guarantee on either the number of views required for coverage or the overall path length.

When a coarse 3D model of the underlying object or scene is available, one can hierarchically solve the view sampling problem. [11, 30] first perform sampling on coarse grids to build a rough model of the objective. These methods still solve a set cover problem at the refinement phase to choose the view candidates on a uniform fine grid. As a result, this refinement phase can be very time-consuming.

Hepp *et al.* [14] applied a static and non-uniform grid sampling for view candidates based on their distance to the occupied voxels. While all these methods produce high-quality 3D reconstructions, they are designed for one-time applications and have to restart even with a simple perturbation, such as moving one object in the scene.

Moreover, to build a surveillance system, occupied voxels and unoccupied ones may become the hotspot, which dramatically increases the number of pairs for visibility computation. After determining the camera views, connecting

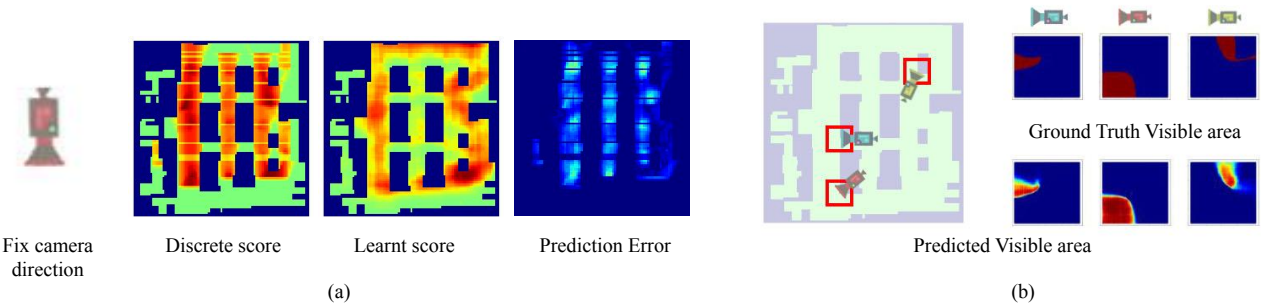


Figure 2: This figure illustrates the basic idea of our approach, which converts combinatorial optimization into continuous optimization. We predict the ground-truth score with its learned approximation, given a discretized scene with a camera pose. (a) shows the heat map of ground-truth scoring function in different positions with fixed view direction (a projection of a 6D heat map) and its continuous approximation from learning. We fix the direction parallel to walls and obstacles to make views and the scoring function easy to visualize. (b) shows three examples of ground-truth visible areas and the predicted ones (middle-left, bottom-left, and top-right in order), cropped around cameras' positions.

them using the shortest cycle solves the so-called traveling salesman problem (or TSP), which is another NP-hard problem. Many approximating algorithms [5, 19] have been designed to solve TSP efficiently, and we do not focus on this sub-problem in the paper.

2.3. View Synthesis

Our view prediction network is motivated by recent successes in leveraging deep neural networks to model image translation [15, 38] and neural rendering [35, 36, 20, 18, 33]. However, these works have focused on computing dense pixel-wise outputs. In our early experiments, we found that such dense outputs are not only time-consuming to compute, but they also lead to rather non-smooth objective functions that are hard to optimize. In contrast, we focus on designing an intermediate representation suitable for approximating the scoring function of the set cover problem.

3. Approach

This section describes the technical details of our approach. Section 3.1 presents the problem statement and an overview of the proposed approach. Section 3.2 to Section 3.6 elaborate on the details of each component.

3.1. Problem Statement and Approach Overview

Problem statement. The input to our approach consists of a 3D scene $S = (\mathscr{V}, \mathscr{F})$, represented as a triangular mesh and a set of entities $\mathscr{E} \subset \mathscr{F}$ for coverage (which are represented as a subset of triangular faces). In real applications, the entities encode interest regions (e.g., products in a supermarket). We also assume we have a pre-defined scoring function $s(e, C) \in \mathbb{R}$ that takes an entity $e \in \mathscr{E}$ and the transformation of a camera C as input, and outputs a score. Our goal is to place a minimal number of cameras so that a user-specified fraction of the entities are covered (e.g., 90%).

Approach overview. As illustrated in Figure 2, the key idea of our approach is to formulate the view selection problem as solving a continuous optimization problem. Traditional approaches typically apply combinatorial optimization to solve the induced set cover problem. However, such approaches require very dense sampling of the high-dimensional camera configuration space. Our approach addresses this issue by directly learning a continuous scoring function that takes a collection of cameras as input and outputs a score to represent coverage. Instead of learning the overall coverage with the entire configuration of cameras, which enlarges the latent space drastically when the number of cameras increases, this is done by learning a visibility function using a deep neural network and then concatenate the visibility function to form the scoring function. This continuous function enables us to optimize camera poses by using powerful continuous optimization tools. Our approach consists of five components, which are highlighted below.

1. Scoring function modeling. Although our goal is to minimize the number of cameras, directly formulating this objective does not lead to optimization problems that are easy to solve (due to the non-continuity nature of the camera count). Our approach addresses this issue by formulating a surrogate function, which is easier to approximate using neural networks.

2. Visibility module. The visibility module replaces the costly rendering operation with a network that takes a 3D scene and a camera pose as input and outputs a visibility map. A key advantage of this module is that one can easily extract gradients of the visibility map concerning the camera pose beside the visibility map. Such gradients are critical for optimizing camera poses.

3. Scoring function module. The module takes n views as input and outputs their score. This scoring module and the visibility module are learned together so that the scoring module's output approximates the scores obtained from solving costly combinatorial optimization.

4. Optimization. The optimization phase consists of a continuous optimization component and a discrete optimization component. The continuous component fixes the number

of cameras and applies BFGS [26] to optimize the camera poses, obtaining a score for each camera count. The discrete component determines the optimal number of cameras through a binary search.

5. Hyper-parameter optimization. The last component performs hyper-parameter optimization to maximize the performance of our approach. This step involves using the finite-difference method to compute gradients of hyper-parameter to minimize the average number of cameras required on a small validation set.

3.2. Scoring Function Modeling

This section presents a surrogate function for optimizing a collection of cameras. We will use this function to generate the training data for learning the scoring network. Specifically, consider n cameras $\mathscr{C} = \{C_1, \cdots, C_n\}$. Let $w_{ie} = w(e, C_i) \in \{0,1\}, \forall 1 \leq i \leq n, e \in \mathscr{E}$ be the indicator that specifies whether the entity e is visible from the camera C_i or not. Note that w_{ie} depends on the orientation $R_i \in SO(3)$ of the camera C_i and its field-of-view. With this setup, we define the scoring function $s(C_1, \cdots, C_n) \in \mathbb{R}$ that measures the coverage of $\mathscr{E}$ using $\mathscr{C}$ as

$$s := \sum_{e \in \mathscr{E}} \left(\max_{1 \leq i \leq n} w_{ie} s(e, C_i) \right) - \lambda n \tag{1}$$

where λ is the trade-off parameter between ensuring the quality of the coverage and using a minimal number of cameras. $s(e, C_i)$ is the scoring function to be introduced next. Note that although our goal is to ensure a user-specified coverage rate, we found the formulation of (1), which mimics a Lagrangian form, is easy to optimize.

We proceed to model $s(e, C_i)$. Instead of using a binary function, our goal is to model $s(e, C_i)$ so that it depends on the angle between the viewing direction and the face geometry. This leads to a smooth scoring function that is easy to optimize and approximate. Specifically, let t_i be the location of the camera C_i. With o_e and n_e we denote the center and normal of the face associated with entity e, respectively. Let θ_{ij} be the angle between the viewing direction $t_i - o_e$ and the face normal n_e. With this setup, we rewrite the scoring function s as

$$s := \sum_{e \in \mathscr{E}} \max_{1 \leq i \leq n} \left(w_{ie} \cdot (\cos^{\alpha}(\theta_{ie}) + \delta_{\max}) - \delta_{\max} \right) - \lambda n \tag{2}$$

where $\delta_{\max}$ is a hyper-parameter that ensures all the views are covered. α is a hyper-parameter that determines the quality function. Note that we will optimize all the hyper-parameters in the final phase of our approach (See Section 3.6).

(2) is difficult to optimize because it is a highly non-smooth (due to w_{ie}) and non-convex objective function of the camera poses. One strategy to optimize such a rather complex objective function is to discretize the solution space by placing a dense set of camera pose samples and select a subset of samples to optimize (2). This strategy usually leads to solving a variant of the set cover problem [31]. However, the downside of this approach is that the solution space is only discrete. To obtain a sufficiently accurate solution, we have to place a dense set of view samples, which incur a high computational cost during testing time.

Our approach is to regress a neural network to predict the score from a camera collection $\mathscr{C} = \{C_1, \cdots, C_n\}$. This scoring function is learned from a sample set of camera configurations and their corresponding scores. The learned scoring function allows us to solve continuous optimization problems to optimize the camera poses. Note that in contrast to solving the problem of predicting a distribution over camera collections (in which high-density regions correspond to camera collections with high scores), our approach, which integrates predicted visibility maps, is more interpretable.

3.3. View Prediction Network

This section introduces the technical details of the view prediction network, which approximates the coverage function of the scene S from one camera C. We first describe how we encode a 3D scene. We then discuss the network structure and the training procedure.

Scene encoding. The input scene S is discretized by a 3D grid (We used $85 \times 85 \times 7$ in our experiment). Let G_S denote the 3D tensor that shares the same size as the grid, with each cell counting the number of enclosed entities in that cell. To encode the underlying scene (which dictates the visibility information), we add a mask channel M_S to the grid, i.e., each cell is associated with a 0-1 indicator, specifying whether some objects in the scene occupy this block or not. Specifically, $M_S(i, j, k) = 0$ if cell (i, j, k) is occupied and 1 otherwise. Note that the non-zero cells of G_S and M_S may not be identical, as the entity set $\mathscr{E}$ of S can be a subset of its faces.

We use another six channels P_C (each channel has the same dimension as G_S and M_S) to encode the camera pose. The first three channels store the relative position of each cell to the camera C. The second three channels encode the absolute location of each cell in the scene coordinate system. We found that such a dense encoding facilitates direct interactions between the scene geometry and the camera pose and improves the learned visibility map's generability. In the following, we denote the predicted view $\tilde{V}(C, S)$ from camera C as

$$\tilde{V}(C, S) := V_{\theta_v}(G_S, M_S, P_C, C).$$

Note that $\tilde{V}(C, S)$ is also a grid with the same dimension as G_S and M_S. In the following, we introduce the network design and the training procedure.

Network Structure. As illustrated in Figure 3 admits an encoding-decoding pipeline. The encoding module extracts features (e.g., shape, pose, spacial interaction between objects) from the queried scene with the given camera transformation. This module has six 3D convolutions and three fully connected layers. The decoder outputs a visibility

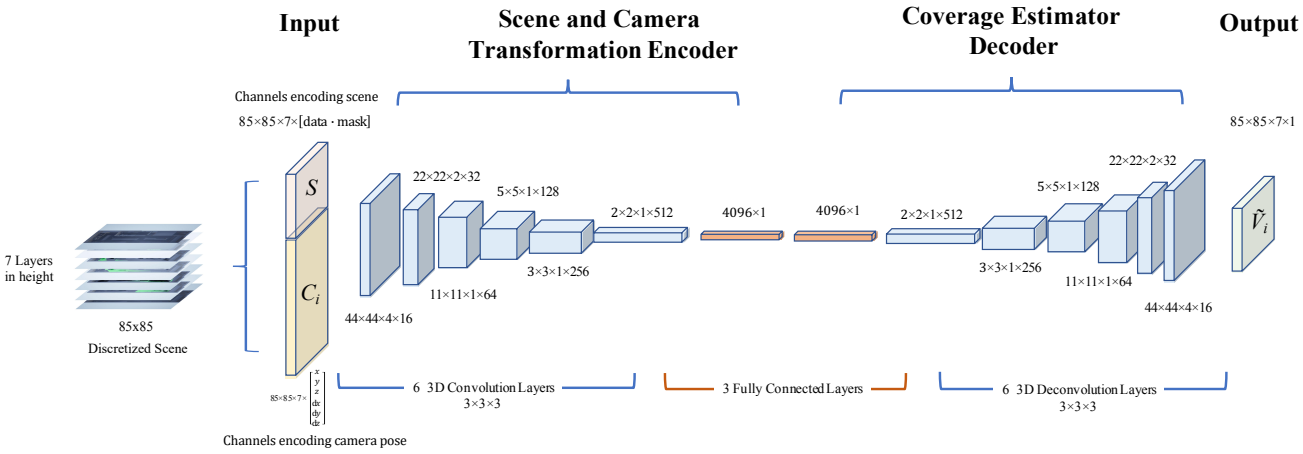


Figure 3: Our view prediction network consists of an encoding module and a decoding module. Each module has six 3D convolution or deconvolution layers. They are bridged together by three fully connected layers. The input is a dense 3D tensor that encodes scene geometry and an input camera pose.

map from a combination of the encoded inputs above. The convolution layers of the decoding module share the same network as the encoder. Moreover, A rectified linear unit follows the 3D convolution, fully connected, and 3D upsampling/Deconvolution layers.

Learning a single camera view. To find an optimal set of parameters θ_v for the view prediction network, we seek to minimize the following loss function:

$$\sum_{<C,S,V^\star>\in\mathscr{D}} \left\| (\tilde{V}(C,S) - V^\star) \circ \left(\mu(G_S+1)\circ M_S - 1\right) \right\|_{\mathscr{F}}^2$$

where $\mathscr{D}$ is the set of training data tuples, $V^\star$ is the ground truth for supervision, and $\circ$ denotes the element-wise product; $\|\cdot\|_{\mathscr{F}}$ is the tensor Frobenius norm. μ is a hyper-parameter that adjusts the weights of three types of block: blocks that are occupied, blocks that are not occupied and contain no entity for detection, and blocks that contain at least one entity for detection.

3.4. Scoring Module

The scoring module is designed to approximate the scoring function (1). It is separated from the view prediction network as an individual module because computing gradients of the outside "max" in (1) is relatively inexpensive for a discretized scene with a small camera budget. This module's input is the set of views from view prediction network $\tilde{V}_1, \tilde{V}_2, ..., \tilde{V}_n$.

Specifically, we query the view prediction network with scene S and camera transformations $C_1, C_2, ..., C_n$ and collect all the outputs to be the input of the scoring module, rearranging them in a 4-dimensional tensor with n channels. We pass the 4-dimensional tensor to a max-pooling layer along the channels and take the summary of all entries as the final estimation of the scoring function. We define the output g of the scoring module as

$$g(\tilde{V}_1, \tilde{V}_2, ..., \tilde{V}_n) = \sum_{i,j,k} \left(\max_{1\le l\le n} \tilde{V}_l(i,j,k) \right) \cdot M_S(i,j,k) - \lambda n$$

where (i,j,k) denotes the value of the (i,j,k)-th entry.

3.5. Optimization

This section discusses how to utilize the learned scoring network to solve the view selection for each new scene. This problem involves the optimization of two variables: the number of cameras and camera locations. In the following, we describe how to solve the continuous problem of placing the cameras while fixing the number of cameras. We then introduce how to optimize the number of cameras.

Continuous optimization of camera poses. The learned scoring function g can be combined with other potential functions for optimizing camera parameters. This paper considers a log-barrier function to ensure that all camera poses are feasible, i.e., they do not lie within the interior of the 3D model. Specifically, consider n camera poses $C_1, C_2, \cdots, C_n$, where n is fixed. We parameterize each camera pose C_i using a vector $v_i = (c_i; t_i)$, where $t_i \in \mathbb{R}^3$ and $R_i = \exp(c_i\times)$ encode the location and the orientation of C_i, respectively. Let $v = (v_1; \cdots; v_n) \in \mathbb{R}^{6n}$ collect all the camera variables. Define $g_n(v) := g(\hat{V}_1, \cdots, \hat{V}_n)$, we setup the following objective function to optimize the camera variables v:

$$\max_v \; g_n(v) - \gamma \sum_{i=1}^{n} \log(d_S(t_i)) \qquad (3)$$

where $d_S : \mathbb{R}^3 \to \mathbb{R}$ is the signed distance function associated with the scene model S; γ is a hyper-parameter in front of the log-barrier potential.

Starting from an initial camera configuration v_0, e.g., by solving the set cover problem using a coarse grid (See Section 4), we follow a standard continuation approach to optimize (3) (c.f. [26]). Specifically, we choose a sufficiently

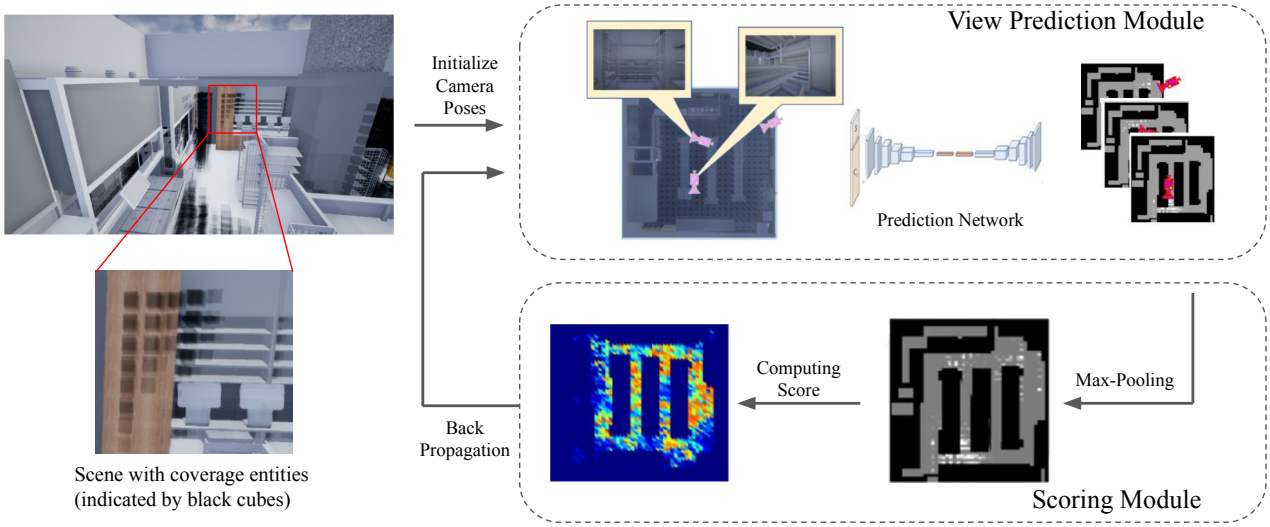


Figure 4: Optimization pipeline (**best viewed in color**). Our optimization process takes a scene with entities for coverage and initial camera poses as input and iteratively optimizes the scoring function. The view prediction module predicts the visibility of the scene for each camera view. The scoring module implements max-pooling over all predicted views and then compute the final score and back-propagate to update camera poses. In this pipeline we freeze the weights of the view prediction network.

large γ and gradually reduce its size by half $\gamma \leftarrow \frac{\gamma}{2}$ until γ is sufficiently small. With fixed γ, we apply BFGS [26] to solve (3). Note that although the objective function of (3) is non-convex, we found that the final solution is generally insensitive to the initialization, particularly when n is large.

Determine the number of cameras. There is a fundamental tradeoff between the number of cameras and the area of the coverage, i.e., a large n usually corresponds to a large coverage ratio. We determine the optimal number of cameras by performing a binary search, i.e., starting from a sufficiently large value for n, and find the smallest n so that the coverage ratio is above 90%. For our experiments under real scenes, the optimal value of n is around dozens.

3.6. Hyper-Parameter Optimization

The performance of our approach depends on the hyper-parameters, i.e., λ, α, $\delta_{\max}$, and μ. These hyper-parameters are used to regularize the underlying scoring function for optimization. Instead of choosing them manually, we optimize them together through a finite-difference approach. Specifically, given the current hyper-parameters, we sample a set of neighboring hyper-parameter configurations and run our approach to obtain the corresponding value for each hyper-parameter configuration (i.e., the average number of cameras needed on the training scenes). We use these samples to compute a numerical gradient (c.f. [26]) and use line-search to obtain the next hyper-parameter configuration.

4. Experimental Results

This section presents an experimental evaluation of the proposed approach. We begin by describing the problem setup in Section 4.1. We then analyze the experimental results in Section 4.2. Finally, we present an analysis of our approach.

4.1. Experimental Setup

Dataset. The experimental study utilizes a dataset consisting of 339 high-resolution 3D models reconstructed from real indoor scenes. Each model contains a set of marked entities, which are to be covered (Note that entity labels were unavailable for other datasets such as Matterport [6]). We split them into a training set with 300 models, a validation set with ten models, and a testing set with 29 models.

To train the view prediction network, we randomly place cameras in the training scenes and collect 500K views in total. The views are sampled from a uniform discretizing grid with random perturbation for both translations and rotations on the scene, with step angle $= 30^\circ$ and step translation $= 40$ centimeters. For each view, we used Unreal rendering Engine 4 [16], which is based on ray tracing. This procedure determines for each entity its score under the scoring function.

Baseline approach. We employ three baseline approaches for experimental. The first baseline is the view-planner of [11] (or NextBest), which adopts a greedy method that searches for the Next-Best View through a discretized space of candidate locations for camera positioning. The second baseline is SubMOpt [30], which performs sub-modular optimization for view-planning. The third baseline employs simulated annealing to solve the induced combinatorial optimization problem.

Implementation details. Our network is built in the PyTorch framework [27]. We use Adam solver [17] with an initial learning rate of 0.002 and a decaying rate of 0.7 every 30 epochs. During testing, we generate the initial solution by applying the greedy approach [31] on a coarse grid with 1.2 meters in translation (translation along the up-right axis is 0) and 60° in rotation (rotation axis is along the up-right direction). All experiments were conducted on a machine with a

	S-Avg.	S-Var.	S-Per-Cam	Time(sec.)	#Cam (80%)	Time (80%)	#Cam (90%)	Time(90%)	Time(8.3%-Per-Came)
Sim. Anneal.	0.464	1.0×10^{-4}	0.046	565	-	-	-	-	-
NextBest [11]	0.711	6.7×10^{-5}	0.079	467	22.3	932	47.8	1784	-
SubMOpt [30]	0.691	1.6×10^{-4}	0.080	463	21.1	1023	40.4	1484	4873
Ours	0.724	2.5×10^{-5}	0.083	462	18.8	512	34.2	534	462

Table 1: Quantitative baseline comparisons. Columns 2 to 5 show the results with ten cameras. We kept the running time of each method close for fair comparisons of their performance on coverage score. The overall scores and scores per camera are given by the ratio between the entities covered by them and the total number of entities. Columns 6 and 7 show the average number of cameras and each approach's running time to achieve the coverage rate of 80%. Columns 8 and 9 show the average number of cameras and each approach's running time to achieve the coverage rate of 90%. Column 10 shows the average time for the baseline approaches to achieve the same performance as ours.

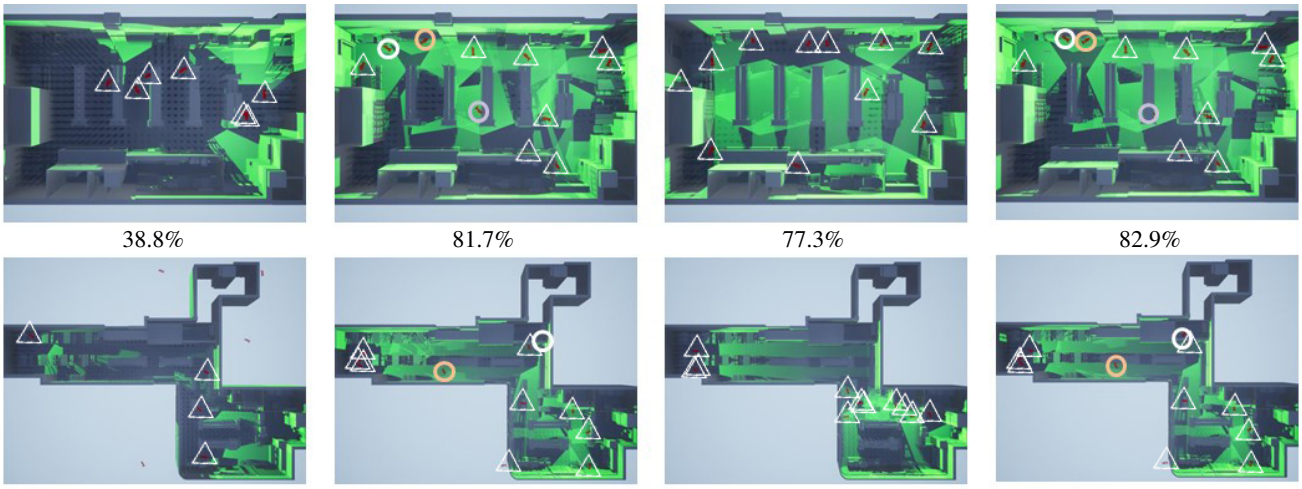


Figure 5: Qualitative evaluation of our method. Cameras are marked in red (visually enhanced with triangles and circles), and the green area indicates the covered regions from the camera configurations, with the overall coverage rates marked below. For configurations that are close to optimum, the visual difference in coverage becomes small. Since our method evolves coarse sampling, to compare with the NextBest baseline, we mark the cameras with noticeable movement in circles with different colors to make them more distinguishable.

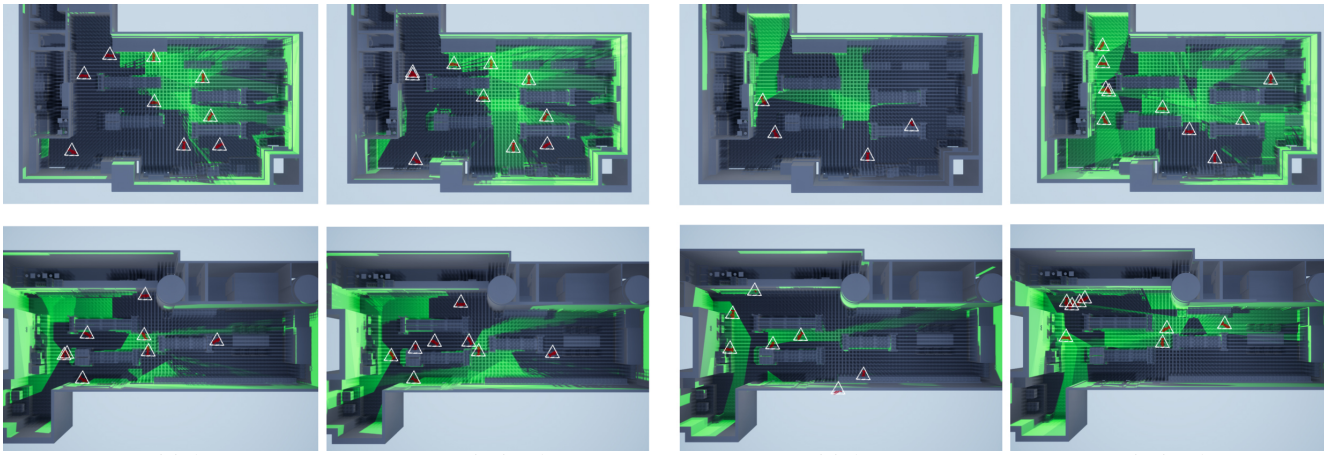
3.2GHz CPU, 16G main memory, and Geforce GTX1080 GPU.

Evaluation protocol. We perform an experimental evaluation, both quantitatively and qualitatively. For quantitative evaluation, we compare the performance of our method and baseline approaches under the same running time upper bound. For qualitative evaluation, we provide a rendering of the scene to visually inspect the quality of the results. Specifically, we put a light source in the position of each camera with its emission cone the same geometry as the view cone of the camera.

4.2. Analysis of Results

Table 1 shows that our approach outperforms baseline approaches across different settings. The second to the fifth columns compare our approach and baseline approaches under ten cameras. We can see that with a similar running time, our approach outperforms all baseline approaches. Specifically, our approach achieves a coverage rate of 72.4%, which is 1.3% higher than the top-performing baseline NextBest [11] (See Figure 5 for a visual comparison). In terms of the covering ratio of one camera, our approach achieved an average ratio of 8.3%, which 0.3% higher than the top-performing baseline. Our approach, NextBest, and SubMOpt outperform simulated annealing by a salient margin. On the one hand, this indicates the hardness of view selection. On the other hand, it also shows the greedy approach's effectiveness for providing the initial solution.

Our approach separates from the baseline approaches when increasing the number of cameras to achieve higher coverage ratios (Note that simulated annealing did not converge in this case). For example, to achieve the coverage ratio of 80%, our approach only requires 18.8 cameras on average, while the top-performing baseline requires 22.1 cameras on average. When increasing the coverage ratio to 90%, our approach requires 34.2 cameras on average, while the top-performing baseline requires 40.4 cameras on average. In this regime, we can see the advantages of our approach and SubMopt that uses global optimization techniques. Another advantage of our approach comes from the



Initial-I. Optimized-I. Initial-II. Optimized-II.

Figure 6: This figure shows the effects of our approach under different initializations. Each sub-figure shows the camera poses, the covered regions are colored in green, and the uncovered regions are in gray. Every row shows a scene (top view). The first and third columns show two different random initial camera configurations of the same scene. The second and fourth columns show the corresponding optimized camera configurations. Cameras can be invalid and not marked initially and become valid after optimization. We can see that the coverage change drastically after optimization.

running time, which only slightly increases when adding more cameras. This is because the most time-consuming part of our approach is sampling a grid of initial cameras, and these initial samples are shared when applying our approach.

4.3. Analysis of Our Approach

Influence on camera initialization. Our experiments suggest our approach only requires a rather coarse grid to compute the initialization for our approach. Specifically, adding more samples to the initial grid (i.e., 1.2m spatial resolution and 60° angular resolution) does not increase the performance of our approach anymore. Figure 6 shows that our optimizer can significantly change the camera poses after optimization (See the reduction in uncovered regions, which are colored in gray). These results show that our approach only places modest constraints on the initial camera configuration and can be applied in different ways.

In contrast, the greedy approach and the next best viewer planner, which operate on a discrete set of samples, require much denser grids to achieve the same solution quality. In terms of timing, the running time of sub-modular optimization (the top-performing baseline) requires 4873 seconds to achieve the same performance as reported in Table 1. In other words, our approach leads to more than ten times speed up from the state-of-the-art in terms of running time.

Different optimization strategies. Besides the BFGS optimizer, we also tried other optimization strategies such as gradient descent with line search and the active set method [26]. The optimal solutions of different approaches remain similar. Such results suggest that local minimums of the learned continuous objective function are distinctive, which again demonstrates our approach's effectiveness.

Sensitivity of hyper-parameters. Note that our scoring function depends on a few hyper-parameters. Varying these hyper-parameters leads to variations of ± 0.6 #cameras on average (with the coverage threshold of 80%). In other words, optimizing these hyper-parameters is effective for our approach. On the other hand, such variations are not salient. We can understand from the fact that although the scoring function varies a lot, the locations of its local minimums may possess small variations. This shows the flexibility of our approach.

Scene resolution. We also tried using different resolutions to encode the visibility network. The performance of our approach remains similar. One explanation is that the visibility network is used to define a scoring function whose input dimension is much smaller than the number of cells, meaning a coarse grid is effective.

5. Conclusions

In this paper, we have introduced an approach for solving the multi-view coverage problem for 3D models. We have shown that it is possible to learn a neural network that takes a camera pose and a 3D model as input and outputs a feature representation of the visible area under that camera pose. The feature representations under multiple views can be aggregated to output an approximation of a pre-defined coverage score. This approach enables fast local refinement via gradient descent. We have demonstrated the usefulness of this approach in accuracy and efficiency.

Acknowledgement This work is supported in part by NSF IIS-2047677 and NSF IIS-1934932. The views presented in this paper are those of the authors and should not be interpreted as representing any funding agencies.

References

[1] A. H. Albahri and A. Hammad. A novel method for calculating camera coverage in buildings using bim. *Journal of Information Technology in Construction (ITcon)*, 22(2):16–33, 2017. 2

[2] K. A. Amriki and P. K. Atrey. Bus surveillance: how many and where cameras should be placed. *Multimedia tools and applications*, 71(3):1051–1085, 2014. 2

[3] J. E. Banta, Y. Zhien, X. Z. Wang, G. Zhang, M. Smith, and M. A. Abidi. Best-next-view algorithm for three-dimensional scene reconstruction using range images. In *Intelligent Robots and Computer Vision XIV: Algorithms, Techniques, Active Vision, and Materials Handling*, volume 2588, pages 418–430. International Society for Optics and Photonics, 1995. 2

[4] E. Becker, G. Guerra-Filho, and F. Makedon. Automatic sensor placement in a 3d volume. In *Proceedings of the 2nd International Conference on PErvasive Technologies Related to Assistive Environments*, page 36. ACM, 2009. 2

[5] V. Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985. 3

[6] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017. 6

[7] V. Chvatal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39–41, 1975. 2

[8] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979. 2

[9] C. Connolly. The determination of next best views. In *Robotics and automation. Proceedings. 1985 IEEE international conference on*, volume 2, pages 432–435. IEEE, 1985. 2

[10] D. Costa and L. A. Guedes. The coverage problem in video-based wireless sensor networks: A survey. *Sensors*, 10(9):8215–8247, 2010. 2

[11] X. Fan, L. Zhang, B. Brown, and S. Rusinkiewicz. Automated view and path planning for scalable multi-object 3d scanning. *ACM Transactions on Graphics (TOG)*, 35(6):239, 2016. 2, 6, 7

[12] Y.-G. Fu, J. Zhou, and L. Deng. Surveillance of a 2d plane area with 3d deployed cameras. *Sensors*, 14(2):1988–2011, 2014. 2

[13] J.-J. Gonzalez-Barbosa, T. García-Ramírez, J. Salas, J.-B. Hurtado-Ramos, et al. Optimal camera placement for total coverage. In *2009 IEEE International Conference on Robotics and Automation*, pages 844–848. IEEE, 2009. 2

[14] B. Hepp, M. Nießner, and O. Hilliges. Plan3d: Viewpoint and trajectory optimization for aerial multi-view stereo reconstruction. *ACM Transactions on Graphics (TOG)*, 38(1):4, 2018. 2

[15] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5967–5976, 2017. 3

[16] B. Karis and E. Games. Real shading in unreal engine 4. 6

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 6

[18] A. KT, P. Sakurikar, S. Saini, and P. J. Narayanan. A flexible neural renderer for material visualization. In *SIGGRAPH Asia 2019 Technical Briefs*, SA '19, page 83–86, New York, NY, USA, 2019. Association for Computing Machinery. 3

[19] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973. 3

[20] L. Liu, W. Xu, M. Zollhöfer, H. Kim, F. Bernard, M. Habermann, W. Wang, and C. Theobalt. Neural rendering and reenactment of human actor videos. *ACM Trans. Graph.*, 38(5):139:1–139:14, 2019. 3

[21] N. A. Massios, R. B. Fisher, et al. *A best next view selection algorithm incorporating a quality criterion*. Department of Artificial Intelligence, University of Edinburgh, 1998. 2

[22] J. Maver and R. Bajcsy. How to decide from the first view where to look next. 1990. 2

[23] J. Maver and R. Bajcsy. Occlusions as a guide for planning the next view. *Technical Reports (CIS)*, page 521, 1992. 2

[24] A. Mittal and L. S. Davis. Visibility analysis and sensor planning in dynamic environments. In *European conference on computer vision*, pages 175–189. Springer, 2004. 2

[25] A. T. Murray, K. Kim, J. W. Davis, R. Machiraju, and R. Parent. Coverage optimization to support security monitoring. *Computers, Environment and Urban Systems*, 31(2):133–147, 2007. 2

[26] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006. 4, 5, 6, 8

[27] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017. 6

[28] R. Pito. A solution to the next best view problem for automated surface acquisition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10):1016–1030, 1999. 2

[29] S. Ram, K. Ramakrishnan, P. Atrey, V. Singh, and M. Kankanhalli. A design methodology for selection and placement of sensors in multimedia surveillance systems. In *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*, pages 121–130. ACM, 2006. 2

[30] M. Roberts, D. Dey, A. Truong, S. Sinha, S. Shah, A. Kapoor, P. Hanrahan, and N. Joshi. Submodular trajectory optimization for aerial 3d scanning. In *International Conference on Computer Vision*, 2017. 2, 6, 7

[31] W. R. Scott, G. Roth, and J.-F. Rivest. View planning for automated three-dimensional object reconstruction and inspection. *ACM Comput. Surv.*, 35(1):64–96, Mar. 2003. 4, 6

[32] V. Sequeira and J. G. Gonçalves. 3d reality modelling: Photo-realistic 3d models of real world scenes. In *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*, pages 776–783. IEEE, 2002. 2

[33] A. Shysheya, E. Zakharov, K. Aliev, R. Bashirov, E. Burkov, K. Iskakov, A. Ivakhnenko, Y. Malkov, I. Pasechnik, D. Ulyanov, A. Vakhitov, and V. S. Lempitsky. Textured neural avatars. In *IEEE Conference on Computer Vision and*

Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019, pages 2387–2397, 2019. 3

[34] P. Slavık. A tight analysis of the greedy algorithm for set cover. *Journal of Algorithms*, 25(2):237–254, 1997. 2

[35] J. Thies, M. Zollhöfer, and M. Nieundefinedner. Deferred neural rendering: Image synthesis using neural textures. *ACM Trans. Graph.*, 38(4), July 2019. 3

[36] J. Tobin, W. Zaremba, and P. Abbeel. Geometry-aware neural rendering. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 11555–11565, 2019. 3

[37] J. Zhao and S. C. Sen-ching. Multi-camera surveillance with visual tagging and generic camera placement. In *2007 First ACM/IEEE International Conference on Distributed Smart Cameras*, pages 259–266. IEEE, 2007. 2

[38] J. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2242–2251, 2017. 3