Let's Make a Splan: Risk-Aware Trajectory Optimization in a Normalized Gaussian Splat

Jonathan Michaux*, Seth Isaacson*, Challen Enninful Adu, Adam Li, Rahul Kashyap Swayampakula, Parker Ewen, Sean Rice, Katherine A. Skinner, and Ram Vasudevan

***Abstract*—Neural Radiance Fields and Gaussian Splatting have transformed the field of computer vision by enabling photo-realistic representation of complex scenes. Despite this success, they have seen only limited use in real-world robotics tasks such as trajectory optimization. Two key factors have contributed to this limited success. First, it is challenging to reason about collisions in radiance models. Second, it is difficult to perform inference of radiance models fast enough for real-time trajectory synthesis. This paper addresses these challenges by proposing SPLANNING, a risk-aware trajectory optimizer that operates in a Gaussian Splatting model. This paper first derives a method for rigorously upper-bounding the probability of collision between a robot and a radiance field. Second, this paper introduces a normalized reformulation of Gaussian Splatting that enables the efficient computation of the collision bound in a Gaussian Splat. Third, a method is presented to optimize trajectories while avoiding collisions with a scene represented by a Gaussian Splat. Experiments demonstrate that SPLANNING outperforms state-of-the-art methods in generating collision-free trajectories in highly cluttered environments. The proposed system is also tested on a real-world robot manipulator. A project page is available at https://roahmlab.github.io/splanning.**

Fig. 1: SPLANNING constructs risk-aware trajectories in a Gaussian Splatting map in real-time. The top image shows a real-world scene that a 7DOF serial manipulator must plan through, starting at the blue configuration (left) and ending at the green configuration (right). The scene is represented as a normalized 3D Gaussian Splat. Then, in real-time, SPLANNING solves an optimization problem that constrains the probability that the robot's forward occupancy (bottom center, purple) collides with the scene.

I. Introduction

For a robot to safely navigate its environment, it must understand the scene geometry it operates within. This understanding must include a detailed model of the scene and a method to reason about collisions with the environment. Radiance field representations, such as Neural Radiance Fields (NeRFs [1]) and Gaussian Splatting [2], have recently emerged as powerful methods for building detailed models of the scene. A radiance field is a five-dimensional function that maps a 3D point and viewing direction to an RGB color and volume rendering opacity. This function is then integrated along camera rays to approximate the image formation process. NeRFs use neural networks to learn the parameters of a radiance field, while Gaussian Splatting models use a set of 3D Gaussian functions. Over the past several years, radiance field representations have marked a paradigm shift in computer vision, with wide-ranging impacts on scene reconstruction [3, 4], novel view synthesis [1], 3D tracking [5, 6], and more.

The robotics research community has begun trying to integrate these models for robotic tasks such as localization [7], mapping [8, 9], and navigation [10]. A key strength of radiance models for robotics applications is that they continuously represent the environment. This differs from conventional discrete representations in robotics, such as point clouds and occupancy grids. However, reasoning rigorously about collisions in these continuous environments is challenging. While existing planners offer practical solutions to this problem, such as discretizing the robot body [10], discretizing the map before planning [11], or treating the confidence ellipsoids of the Gaussians in a Gaussian Splat as obstacles [12], work remains to fully exploit the continuous nature of radiance field models.

This paper extends the literature on motion planning in radiance fields by proposing a real-time, receding-horizon trajectory optimization algorithm called SPLANNING. This paper's key contributions are: 1) A rigorous definition and derivation of rigid body collision within a radiance field model, starting directly from the rendering equation; 2) a computationally efficient approach to upper-bound the probability of collision within a Gaussian Splatting model; 3) a re-formulation of Gaussian Splatting that normalizes the 3D Gaussians to ensure

*Denotes equal contribution.

Jonathan Michaux, Seth Isaacson, Challen Enninful Adu, Adam Li, Rahul Kashyap Swayampakula, Parker Ewen, Sean Rice, Katherine A. Skinner and Ram Vasudevan are with the Department of Robotics, University of Michigan, Ann Arbor, MI 48109. {jmichaux, sethgi, enninful, adamli, rahulswa, pewen, seanrice, kskin, ramv}@umich.edu.

This work was funded by MCity, University of Michigan.

the correctness of the collision probabilities; 4) a novel risk-aware trajectory planner for robot manipulators. Simulation and hardware experiments illustrate that the risk-aware planner solves challenging tasks in real-time.

Relationship to Prior Work: SPLANNING builds upon prior work entitled Safe Planning for Articulated Robots Using Reachability-based Obstacle Avoidance With Spheres (SPARROWS) [13]. The prior work develops a trajectory optimization algorithm that leverages a novel sphere-based reachable set that overapproximates the swept volume of a serial robot manipulator. At runtime, SPARROWS uses this representation to enforce collision-avoidance constraints with obstacles of known geometry. In contrast, the present work introduces a novel chance constraint to facilitate planning in scenes with arbitrary geometry modeled by radiance fields.

II. Related Work

SPLANNING is an algorithm combining trajectory optimization, reachability analysis for robot safety, and radiance fields for 3D scene representation. We discuss the relevant literature here.

A. Trajectory Optimization

To generate safe motion plans, state-of-the-art trajectory optimizers such as CHOMP [14], TrajOpt [15], MPOT [16], and cuRobo [17] model the robot or the environment with simple geometric primitives such as spheres [18, 19], ellipsoids [20], capsules [21, 22], or convex polygons and perform collision-checking along a given trajectory at discrete time instances. CHOMP represents the robot as a collection of discrete spheres and avoids collisions using a signed distance field to maintain a safety margin with the environment. TrajOpt uses the support mapping of convex shapes to represent the environment obstacles and the robot. Then, the signed distance (positive distance and penetration depth) between two convex shapes is computed by the Gilbert-Johnson-Keerthi [23] and Expanding Polytope Algorithms [24]. MPOT represents the robot geometry and environment obstacles as a collection of spheres and implements a collision-avoidance cost using an occupancy map [16]. Using a gradient-free approach, MPOT optimizes a batch of smooth trajectories and selects the one with the lowest cost. Similarly, a recent method called cuRobo [17] represents the robot as a collection of spheres and solves multiple trajectory optimization problems in parallel on the GPU to identify a collision-free path. The major drawback of these approaches is that the resulting trajectories are not guaranteed to be safe, as collision avoidance is enforced only as a soft penalty in the cost function. Furthermore, these approaches require explicit representations of scene geometry, such as zonotopes or 3D occupancy grids, which can be challenging to construct in real-world robotics applications. In contrast, SPLANNING enforces collision avoidance as a constraint and operates directly in a learned Gaussian Splatting model.

B. Reachability Analysis

Reachability-based Trajectory Design (RTD) [25] is a recent approach to real-time motion planning that generates provably safe trajectories in a receding-horizon fashion. At runtime, RTD constructs reachable sets that overapproximate all possible robot configurations corresponding to a pre-specified continuum of parameterized trajectories. RTD then solves a nonlinear optimization problem to select a feasible trajectory such that the robot's motion is guaranteed to be collision-free. If a feasible trajectory is not found, RTD brings the robot safely to a stop by executing a fail-safe braking maneuver. Unlike traditional trajectory optimization methods, RTD constructs reachable sets such that obstacle-avoidance constraints are satisfied in *continuous-time*. Recent extensions of RTD have demonstrated real-time, certifiably-safe motion planning for robotic arms [13, 26–29] and mobile robots [30, 31]. Probabilistic extensions of RTD have also been proposed [32], which provide safety guarantees in uncertain environments. However, these reachability-based methods assume ground-truth knowledge of obstacle geometry or that probability density functions on the locations of obstacles are provided. SPLANNING extends the literature on reachability analysis by presenting a reachability-based planner that operates in a radiance field model.

C. Radiance Fields

Neural Radiance Fields (NeRFs) were first introduced to address the problem of novel view synthesis [1]. NeRFs and their variants use neural networks to estimate the radiance emitted by a scene point when viewed from a given direction. Since their introduction, NeRFs have found many use cases in robotics [33] including navigation [10, 34], pose estimation [7, 35], manipulation [36], and SLAM [8, 9]. NeRF-Nav [10], an early effort toward safe planning in radiance fields, approximates the robot as a set of points and then avoids collisions by integrating the NeRF density along the path traced by each point. A later work, CATNIPS [11], instead presents a framework for relating NeRFs to Poisson Point Processes. CATNIPS then introduces a method to convert a NeRF to an occupancy grid to plan robot motions.

While NeRFs use a neural network to model image formation, Gaussian Splatting methods learn a similar representation using Gaussian basis functions [37]. Recently, Kerbl et al. introduced 3D Gaussian Splatting (3DGS) [2], which learns the parameters of un-normalized 3D Gaussians via gradient descent. 3DGS offers high-speed rendering and a training method compatible with modern Graphics Processing Units (GPUs). However, similar to NeRFs, only limited efforts have been made towards using 3DGS for real-time motion planning. A recent work called Splat-Nav [12] leverages the fact that 3DGS only renders Gaussians where the 99% confidence interval overlaps with the camera's view frustum. This allows Splat-Nav to ignore all other Gaussians while generating motion plans within safe corridors that avoid collisions with the 99% confidence intervals. Still, Splat-Nav uses sampling-based methods to check collisions. In addition, Splat-Nav's collision checking is not differentiable. In contrast, this paper

Fig. 2: SPLANNING optimizes trajectories in a Normalized 3D Gaussian Splat given a start configuration (blue) and goal configuration (green). Offline, a Normalized 3D Gaussian Splat is constructed to represent the scene geometry. Then, during online planning, a constraint bounds the probability that the Spherical Forward Occupancy (purple) intersects with the scene, as represented by a Normalized 3D Gaussian Splat.

proposes a novel re-formulation of 3DGS enabling collision checking that is computationally simple and fully differentiable, allowing it to be incorporated into a trajectory optimizer and gradient-based learning methods. Notably, we demonstrate that our collision avoidance representation is more accurate than existing methods.

III. Proposed Method Overview

As illustrated in Fig. 2, SPLANNING computes risk-aware trajectories using a visual scene representation. The key insight behind SPLANNING is the combination of reachability analysis with a novel Normalized 3D Gaussian Splat. This combination allows SPLANNING to constrain the probability of collision between the robot's reachable set and the scene.

A. Notation

We briefly outline the notation used throughout the rest of the document.

1) Sets and Functions: $n_A \in \mathbb{N}$ represents an arbitrary constant, while $N_A = \{1, \ldots, n_A\} \subset \mathbb{N}$ is a set. Capital Roman letters X denote sets, and $co(X)$ denote the convex hull of X. $f(x)$ denotes a function of x, while $f[g]$ denotes a functional of the function g.

2) Probabilities: A probability space is denoted $(\Omega, \mathcal{F}, \mathcal{P})$ with sample space Ω, event space $\mathcal{F}$, and probability measure $\mathcal{P}$. A random variable is denoted $\mathbf{X} : \Omega \to S$ for some measurable space S. For a random event E, the probability that E occurs is denoted $\mathrm{P}(E)$.

3) Robot Descriptions: $q(t;k)$ and $\dot{q}(t,k)$ represent a robot's time-dependent configuration and velocity, respectively. $k \in K \subset \mathbb{R}^{n_k}$ parameterizes a continuum of trajectories, and $t \in T \subset \mathbb{R}$ specifies the time horizon. The forward occupancy of a robot while following a trajectory $q(t;k)$ is denoted $\mathrm{FO}(q(t;k))$.

B. Overview of Optimization

At runtime, SPLANNING computes probabilistically safe trajectories by solving an optimization problem that limits the probability of the robot's forward occupancy intersecting the environment. SPLANNING tries to solve the following optimization problem in a receding-horizon manner:

$$\min_{k \in K} \quad \texttt{cost}(k) \tag{1}$$

$$q(t;k) \in [q^-_{j,\text{lim}}, q^+_{j,\text{lim}}] \qquad \forall t \in T \tag{2}$$

$$\dot{q}(t;k) \in [\dot{q}^-_{j,\text{lim}}, \dot{q}^+_{j,\text{lim}}] \qquad \forall t \in T \tag{3}$$

$$\mathrm{P}\left(\mathrm{FO}(q(t;k)) \cap \mathscr{E} \neq \emptyset\right) < \beta \qquad \forall t \in T \tag{4}$$

The cost function (1) is a user-defined objective, such as bringing the robot close to a desired goal. Input constraints enforce limits on joint positions (2) and velocities (3). Finally, (4) is a safety constraint ensuring that the probability of the robot's forward occupancy $\mathrm{FO}(q(t;k))$ intersecting the environment $\mathscr{E}$ stays below the risk threshold β.

The rest of this paper is organized as follows. Section IV details the representation of the robot and outlines how the forward occupancy of the robot is overapproximated. Section V provides a detailed description of SPLANNING, including the Normalized 3D Gaussian Splatting used to represent the scene and a method for overapproximating the risk constraint in (4). Section VI evaluates SPLANNING both in simulation and on hardware, and Section VII concludes and discusses avenues for future work.

IV. Robot Representation

This section describes the construction of a sphere-based safety representation for a serial robotic manipulator. Section IV-A summarizes the kinematics of the robotic arm. Section IV-B then introduces the arm occupancy, which is the volume occupied by the arm in the environment. Lastly, Thm. 3 establishes the existence of the robot's Spherical Forward Occupancy, a sphere-based reachable set representation used to construct SPLANNING's novel chance constraint in Section V-C.

A. Arm Kinematics

Given a compact time interval $T \subset \mathbb{R}$, we define a trajectory for the robot's configuration as $q : T \to Q \subset \mathbb{R}^{n_q}$ and a trajectory for the velocity as $\dot{q} : T \to \mathbb{R}^{n_q}$. We restate an assumption [13, Ass. 4] about the robot model:

Assumption 1. *The robot operates in a three-dimensional workspace, denoted $W_s \subset \mathbb{R}^3$, such that $W_s \subset W$ where W denotes the world frame. There exists a reference frame called*

the base frame, denoted the 0th frame, that indicates the origin of the robot's kinematic chain. We assume the robot's base frame coincides with the origin of the world frame. The robot is fully actuated and composed of only revolute joints, where the jth joint actuates the robot's j^{th} link. The robot's j^{th} joint has position and velocity limits given by $q_j(t) \in [q^-_{j,\text{lim}}, q^+_{j,\text{lim}}]$ and $\dot{q}_j(t) \in [\dot{q}^-_{j,\text{lim}}, \dot{q}^+_{j,\text{lim}}]$ for all $t \in T$, respectively. The robot's input is given by $u : T \to \mathbb{R}^{n_q}$.

We also assume that the robot's j^{th} reference frame $\{\hat{x}_j, \hat{y}_j, \hat{z}_j\}$ is attached to the robot's j^{th} revolute joint, and that $\hat{z}_j = [0, 0, 1]^\top$ points in direction of the jth joint's axis of rotation. Then $\text{FK}_j : Q \to \mathbb{R}^{4\times 4}$ maps the robot's time-dependent configuration to the pose of the jth joint in the world frame:

$$\text{FK}_j(q(t)) = \begin{bmatrix} R_j(q(t)) & p_j(q(t)) \\ 0 & 1 \end{bmatrix}, \tag{5}$$

where

$$p_j(q(t)) = \sum_{l=1}^{j} R_l(q(t)) p_l^{l-1}, \tag{6}$$

and

$$R_j(q(t)) := R_j^0(q(t)) = \prod_{l=1}^{j} R_l^{l-1}(q_l(t)) \tag{7}$$

are the position and orientation of frame j with respect to world frame W, respectively.

For simplicity, Assum. 1 only specifies revolute joints to simplify the description of the forward kinematics; the methods described in this paper readily extend to other joint types. We also ignore any uncertainty in the robot's dynamics, and instead assume that one could apply an inverse dynamics controller [38] to track any trajectory of the robot perfectly. As a result, this work focuses exclusively on modeling the kinematic behavior of the robotic arm.

B. Arm Occupancy

In this subsection, we define the *forward occupancy* as the volume occupied by the arm in the workspace W_s. Let $L_j \subset W_s \subset \mathbb{R}^3$ denote the volume occupied by the robot's j^{th} link with respect to the jth reference frame. Then the forward occupancy of link j is the map $\text{FO}_j : Q \to \mathcal{P}(W_s)$ defined as

$$\text{FO}_j(q(t)) = p_j(q(t)) \oplus R_j(q(t)) L_j, \tag{8}$$

where $p_j(q(t))$ and $R_j(q(t))$ specify the pose of the jth joint, and $R_j(q(t))L_j$ is the rotated volume of link j. The volume occupied by the entire arm in the workspace is then defined by the map $\text{FO} : Q \to W_s$ such that

$$\text{FO}(q(t)) = \bigcup_{j=1}^{n_q} \text{FO}_j(q(t)) \subset W_s. \tag{9}$$

Because the geometry of any of the robot's links may be arbitrarily complex, we restate an assumption [13, Ass. 5] that simplifies the construction of an overapproximation to the forward occupancy:

Assumption 2. *Given a robot configuration $q(t)$ and any $j \in \{1, \ldots, n_q\}$, there exists a ball with center $p_j(q(t))$ and radius r_j that overapproximates the volume occupied by the jth joint in W_s. We further assume that link volume L_j is a subset of the tapered capsule formed by the convex hull of the balls overapproximating the jth and $(j+1)^{th}$ joints.*

Following Assum. 2, we now define the ball $S_j(q(t))$ overapproximating the volume occupied by the jth joint as

$$S_j(q(t)) = B_2(p_j(q(t)), r_j) \tag{10}$$

and the tapered capsule $TC_j(q(t))$ overapproximating the jth link as

$$TC_j(q(t)) = co\Big(S_j(q(t)) \cup S_{j+1}(q(t))\Big). \tag{11}$$

Then, the volumes occupied by the jth link (8) and the entire arm (9) is overapproximated by

$$\text{FO}_j(q(t)) \subset TC_j(q(t)) \tag{12}$$

and

$$\text{FO}(q(t)) \subset \bigcup_{j=1}^{n_q} TC_j(q(t)) \subset W_s, \tag{13}$$

respectively.

For convenience, the notation $\text{FO}(q(T))$ denotes the forward occupancy over an entire time interval T. $\text{FO}(q(T))$ is also called the *reachable set* of the robot. Notably, the arm is *collision-free* over the time interval T if $\text{FO}(q(T))$ does not intersect with the environment.

To facilitate the exposition of our approach, we summarize the construction of the robot's safety representation in the following theorem:

Theorem 3. *Given a serial manipulator with $n_q \in \mathbb{N}$ revolute joints and a time partition T of a finite set of intervals T_i (i.e., $T = \cup_{i=1}^{n_t} T_i$), the swept volume corresponding to the robot's motion over T is overapproximated by a collection of L_2 balls in $\mathbb{R}^3$, which we call the Spherical Forward Occupancy ($\mathcal{SFO}$) [13] defined as*

$$\mathcal{SFO} = \cup_{j=1}^{n_q} \cup_{i=1}^{n_t} \cup_{m=1}^{n_S} S_{j,i,m}(q(T_i; k)), \tag{14}$$

where each $S_{j,i,m}(q(T_i; k))$ is an L_2 ball in $\mathbb{R}^3$, $n_S \in \mathbb{N}$ is a parameter that specifies the number of closed balls overapproximating each of the robot's links, and k is a trajectory parameter that characterizes the motion of the robot over T.

Note that one can explicitly construct an $\mathcal{SFO}$ that satisfies this assumption using the approach described in [13], which we summarize for convenience in Appendix B.

V. Detailed Description of SPLANNING

This section provides a detailed overview of SPLANNING, a novel approach for generating risk-aware motion plans in cluttered scenes represented as radiance fields using Gaussian basis functions. Section V-A provides a brief overview of radiance fields and Gaussian Splatting. Section V-B then describes how to bound the probability of collision between a ball in $\mathbb{R}^3$ and a learned radiance field represented by Gaussian Splats. Section V-C presents a closed-form upper-bound on the aforementioned probability of collision, and Section V-D discusses how to leverage this result as a computationally-tractable chance constraint for online trajectory optimization.

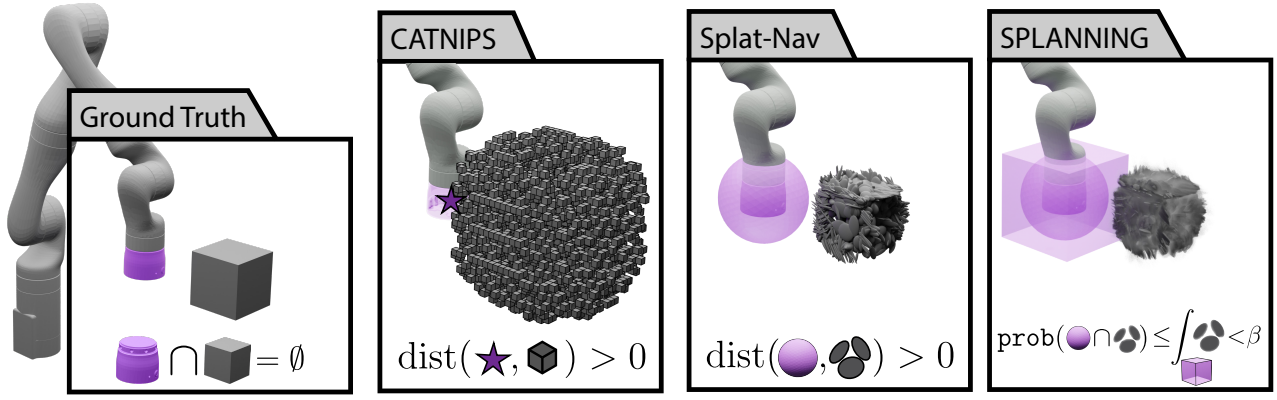

Fig. 3: The collision constraint representation of CATNIPS, Splat-Nav, and SPLANNING are compared. CATNIPS transforms a NeRF into a 3D occupancy grid by relating it to a Poisson Point Process; after convolving this grid with a robot kernel, the center of the robot is checked for collision with the grid. Splat-Nav deterministically evaluates whether the confidence ellipsoids of each Gaussian intersect with a spherical robot. Finally, SPLANNING integrates a Normalized 3D Gaussian Splat over an overapproximation of the robot arm to form a risk constraint.

A. 3D Scene Representation

This section introduces the Normalized 3D Gaussian Splatting model used to represent the environment.

1) Volume Rendering: A radiance field is a 5-dimensional function $\mathcal{L} : (x, d) \mapsto (r, g, b, \sigma)$ that maps a point $x \in \mathbb{R}^3$ and viewing direction $d \in \mathbb{S}^2$ to r, g, b colors and a volume density σ [39]. This work neglects color as it does not impact the collision probability. Further, the density σ does not depend on the view direction. As a result, we simplify the description of the radiance model by estimating the density function $\sigma : \mathbb{R}^3 \to \mathbb{R}$.

We define a ray $\varphi : \mathbb{R} \to \mathbb{R}^3$ by $\varphi(t) = o_\varphi + t v_\varphi$, where $o_\varphi \in \mathbb{R}^3$ is the ray origin and $v_\varphi \in \mathbb{S}^2$ is the unit direction vector. From the density function σ, we may compute the probability that a particle travels along $\varphi(t)$ from $t = a$ to $t = b$ without collision using the transmittance function $\mathcal{T}_a^b$, as derived in [39]:

$$\mathcal{T}_a^b[\varphi] = \exp\left(-\int_a^b \sigma(\varphi(t))dt\right). \tag{15}$$

Equivalently, we may define $C_a^b[\varphi]$ as the random event describing a particle colliding while traveling along φ from a to b, where $\mathrm{P}(C_a^b[\varphi]) = 1 - \mathcal{T}_a^b[\varphi]$ denotes the probability that a collision occurs.

2) Normalized Gaussian Splatting: Traditional splatting approaches represent the density function σ using basis functions such as 3D Gaussian functions [37]. This allows the integral in (15) to be computed by transforming each Gaussian to each ray's coordinate system and then analytically marginalizing the depth dimension. The result is a set of 2D Gaussian functions on the image plane that are queried and blended to form the image. 3DGS [2] proposed a variation of the traditional splatting formulation that uses un-normalized 3D Gaussian functions to represent a scene. A consequence of using un-normalized Gaussians is that the 3D to 2D transformation performed during rasterization *projects* the 3D Gaussians onto the image plane rather than integrating the 3D Gaussians. Because the 2D Gaussians are not constructed from integrating 3D Gaussians along rays, the 3D Gaussians cannot be interpreted as a basis set for σ.

In contrast, the method we introduce for collision evaluation (Section V-B) relies on using the 3D Gaussians as a basis for σ. Hence, one of our key contributions is to re-formulate 3DGS using $n_G \in \mathbb{N}$ normalized 3D Gaussians such that σ can be used to compute the probability of collision of a particle traveling along φ. In particular,

$$\begin{aligned}\mathrm{P}(C_a^b[\varphi]) &= 1 - \mathcal{T}_a^b[\varphi] \\ &= 1 - \exp\left(-\int_a^b \sigma(\varphi(t))dt\right) \\ &= 1 - \exp\left[\sum_{n=1}^{n_G} \frac{w_n}{\sqrt{(2\pi)^3 |\Sigma_n|}} \cdot \right. \\ &\qquad \left. \int_a^b \exp\left(-\frac{1}{2}\|\varphi(t) - \mu_n\|^2_{\Sigma_n^{-1}}\right) dt\right],\end{aligned} \tag{16}$$

where $w_n \in \mathbb{R}^+$ is a weight parameter and $G_n : \mathbb{R}^3 \to \mathbb{R}$ gives the normalized Gaussian density with mean $\mu_n \in \mathbb{R}^3$ and covariance matrix $\Sigma_n \in \mathbb{R}^{3\times 3}$. That is,

$$G_n(x) = \frac{1}{\sqrt{(2\pi)^3 |\Sigma_n|}} \exp\left(-\frac{1}{2}(x - \mu_n)^T \Sigma_n^{-1} (x - \mu_n)\right). \tag{17}$$

Additionally, existing 3D Gaussian Splatting implementations apply a low-pass filter to the 2D Gaussian functions on the image plane by convolving the projected 2D Gaussians with an isotropic 2D Gaussian with a covariance of 0.3 pixels. This low-pass filter reduces artifacts in the rendered images. We omit this step from the normalized 3DGS rendering procedure because it has an effect that cannot be reasoned about in 3D. This would undermine the validity of our collision avoidance constraint.

Instead, we apply an analogous filter in 3D Gaussians by convolving the 3D Gaussians in world coordinates with an isotropic 3D Gaussian with a small covariance of $1e - 6$. Because this operation is applied to the 3D Gaussians, it does not impact the ability of the collision constraint to use the 3D Gaussians to model σ. The implementation details describing the updated Gaussian Splat training process are provided in Appendix E.

B. Bounding the Probability of Collision

SPLANNING enforces safety by ensuring the probability of collision between the robot and the scene is below a given risk threshold. This subsection derives a method for bounding the probability of collision between a ball in $\mathbb{R}^3$ and a scene represented as a radiance field. We make the following assumption about the Normalized 3D Gaussian Splat's representation of the scene:

Assumption 4. *The normalized 3D Gaussian Splat is assumed to have converged such that the 3D Gaussians form a valid basis for the volume density σ. Consequently, the transmittance, as defined in (15), is accurately computed. Furthermore, we assume that the transmittance, which represents the probability of a particle colliding while traveling along a ray, is equivalent to the probability that an infinitesimal segment of a rigid body experiences a collision when traveling along the same ray.*

Without loss of generality, suppose $S = B_2(0, \rho)$ is a closed L_2 ball centered at the origin with radius ρ and whose boundary is denoted ∂S. This assumption is made without loss of generality because one can apply a frame transformation to any arbitrary ball in $\mathbb{R}^3$ to shift its center to the origin. Let $C(S)$ denote the random event that the ball S collides with the environment. Then, we seek to compute the probability that $C(S)$ occurs, which we denote $\mathrm{P}(C(S))$. We model this as the probability that a ray randomly cast from the center of S experiences a collision before reaching ∂S.

Formally, let $(\Omega_{\mathbf{V}}, \mathcal{F}_{\mathbf{V}}, \mathcal{P}_{\mathbf{V}})$ be a probability space, and let $\mathbf{V} : \Omega_{\mathbf{V}} \to \mathbb{S}^2$ be a random variable representing the direction of a randomly-cast ray originating at the center of S. In particular, let $\mathbf{V}$ be uniformly distributed on $\mathbb{S}^2$ under $\mathcal{P}_{\mathbf{V}}$. Define the random ray $\mathbf{\Phi}$ as:

$$\mathbf{\Phi}(t, \omega_{\mathbf{V}}) = t\mathbf{V}(\omega_{\mathbf{V}}), \quad \omega_{\mathbf{V}} \in \Omega_{\mathbf{V}}. \tag{18}$$

The transmittance of $\mathbf{\Phi}(t, \omega_{\mathbf{V}})$ from $t = 0$ to $t = \rho$ is given by

$$\mathcal{T}_0^{\rho}(\omega_{\mathbf{V}}) = \exp\left(-\int_0^{\rho} \sigma(\mathbf{\Phi}(t, \omega_{\mathbf{V}}))dt\right), \quad \omega_{\mathbf{V}} \in \Omega_{\mathbf{V}}. \tag{19}$$

Note that $\mathcal{T}_0^{\rho}(\omega_{\mathbf{V}})$ depends on $\mathbf{\Phi}(t, \omega_{\mathbf{V}})$, which in turn depends on $\mathbf{V}(\omega_{\mathbf{V}})$. Hence, $\mathcal{T}_0^{\rho} : \Omega_{\mathbf{V}} \to \mathbb{R}$ is also a random variable on $(\Omega_{\mathbf{V}}, \mathcal{F}_{\mathbf{V}}, \mathcal{P}_{\mathbf{V}})$. The probability the random ray collides on the interval from $t = 0$ to $t = \rho$ is computed by $1 - \mathcal{T}_0^{\rho}(\omega_{\mathbf{V}})$. Finally, the probability that the collision risk for the randomly-cast ray exceeds a risk threshold $\alpha \in \mathbb{R}^+$ is given by

$$\mathrm{P}(C(S)) = \mathrm{P}\left(1 - \mathcal{T}_0^{\rho}(\omega_{\mathbf{V}}) \geq \alpha\right), \quad \omega_{\mathbf{V}} \in \Omega_{\mathbf{V}}. \tag{20}$$

The following provides an upper bound on $\mathrm{P}(C(S))$:

Theorem 5. *Consider, without loss of generality, a ball $S = B_2(0, \rho)$ of radius ρ that is centered at the origin. Let $\alpha \in \mathbb{R}^+$ denote the risk threshold defined in (20) for the probability of collision between a ray and the environment. Then, the probability that the ball S collides with the environment is bounded above by*

$$\mathrm{P}(C(S)) \leq \frac{1}{\alpha}\left[1 - \exp\left(-\frac{1}{4\pi}\int_S \sigma(x)dx\right)\right] \tag{21}$$

where the integral denotes a volume integral over the sphere $\mathcal{S}$.

A proof is provided in Appendix C-A.

C. Evaluating the Collision Bound

Computing the exact integral in (21) is non-trivial. While [40, Thm 3.3] provides a method for computing the integral of Gaussian functions over spheres, it depends on an infinite series that is difficult to evaluate in practice. Therefore, we derive a closed-form expression for computing an upper bound for the probability of collision in Theorem 5.

Theorem 6. *Let $S = B_2(0, \rho)$ be a closed ball of radius ρ that is centered at the origin. Suppose the density function $\sigma : \mathbb{R}^3 \to \mathbb{R}$ is represented by a set of $n_G \in \mathbb{N}$ normalized Gaussian basis functions $\{G_n\}_{n=1}^{n_G}$ with means $\{\mu_n\}_{n=1}^{n_G}$ and covariance matrices $\{\Sigma_n\}_{n=1}^{n_G}$, each with Eigendecomposition given by $\Sigma_n = R_n \Lambda_n R_n^T$. Finally, let the diagonal elements of Λ_n be denoted as $\lambda_{n,1}, \lambda_{n,2}, \lambda_{n,3}$ and represent the eigenvalues of Σ_n. Then*

$$\int_S \sigma(x)dx \leq \mathcal{H}(S) \tag{22}$$

where

$$\mathcal{H}(S) = \sum_{n=1}^{n_G} \eta_n' w_n \cdot \prod_{\ell=1}^{3}\left[\sqrt{\frac{\pi\lambda_{n,\ell}'}{2}}\left(\mathrm{erf}\left(\frac{\rho - \mu_{n,\ell}'}{\sqrt{2\lambda_{n,\ell}'}}\right) - \mathrm{erf}\left(\frac{\rho + \mu_{n,\ell}'}{\sqrt{2\lambda_{n,\ell}'}}\right)\right)\right]. \tag{23}$$

Above, erf *denotes the error function and μ_n', λ_n', and η_n' correspond to the mean, eigenvalues, and normalization constant of the normalized Gaussian G_n' obtained by rotating G_n by R_n^T. That is, G_n' has mean $\mu_n' = R_n^T \mu_n$ and covariance $\Sigma_n' = R_n^T \Sigma_n R_n$. Furthermore,*

$$\mathrm{P}(C(S)) \leq \frac{1}{\alpha}\left[1 - \exp\left(-\frac{1}{4\pi}\int_S \sigma(x)dx\right)\right] \tag{24}$$

$$\leq \frac{1}{\alpha}\left[1 - \exp\left(-\frac{1}{4\pi}\mathcal{H}(S)\right)\right]. \tag{25}$$

Thus, we may constrain the risk of collision by enforcing

$$\left[1 - \exp\left(-\frac{1}{4\pi}\mathcal{H}(S)\right)\right] < \alpha \cdot \beta \tag{26}$$

for a given risk threshold $\beta \in (0, 1]$.

A proof is provided in Appendix C-B. Note that, in practice, we set $\alpha = \beta$ for simplicity due to their mutual dependence introduced in (26).

D. Risk-Aware Motion Planning

The purpose of SPLANNING is to generate risk-aware motion plans in cluttered environments in a receding-horizon fashion. Prior to planning, a normalized Gaussian Splatting representation of the scene is constructed. At every planning

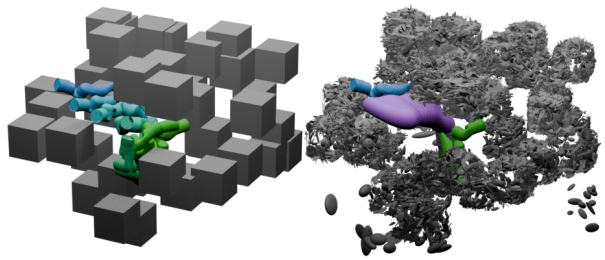

Fig. 4: SPLANNING generates safe trajectories in densely cluttered environments in simulation. The left panel shows discrete time steps of sequential trajectories that brings the arm from the start configuration (blue) safely to the goal configuration (green). The right panel shows an intermediate planning step where the Spherical Forward Occupancy (purple) avoids the obstacles represented by a Normalized 3D Gaussian Splat.

iteration, the robot is given $t_p \leq 0.5$ seconds to find a feasible trajectory by solving

$$\min_{k \in K} \quad \texttt{cost}(k) \qquad \texttt{(Splanning-Opt)} \tag{27}$$

$$\text{s.t.} \quad q_j(T_i;k) \subseteq [q^-_{j,\text{lim}}, q^+_{j,\text{lim}}] \qquad \forall (i,j) \in N_t \times N_q \tag{28}$$

$$\dot{q}_j(T_i;k) \subseteq [\dot{q}^-_{j,\text{lim}}, \dot{q}^+_{j,\text{lim}}] \qquad \forall (i,j) \in N_t \times N_q \tag{29}$$

$$\sum_{\substack{(j,m)\in \\ N_q \times N_s}} 1 - \exp\left(-\frac{\mathcal{H}(S_{j,i,m}(q_j(T_i;k))}{4\pi}\right) < \alpha \cdot \beta \quad \forall i \in N_t \tag{30}$$

where $k \in K$ is the trajectory parameter (App. B-A) and $\texttt{cost}(k)$ is a task-specific cost function. The robot's reachable set $\{S_{j,i,m}\}_{m=1}^{n_S}$ is a function of the robot's position (and hence trajectory parameter) and is computed repeatedly while numerically solving (Splanning-Opt). Safety is enforced using the novel collision-avoidance constraint (30). If a solution is not found, the robot executes a braking maneuver using the trajectory parameter found in the previous planning iteration. Since SPLANNING's collision-avoidance constraints are differentiable, analytical constraint gradients are provided to ensure real-time motion planning. For simplicity, expressions for kinematics and dynamics constraints were not included in (Splanning-Opt). However, [27] and [13] provide detailed explanations for including such constraints.

VI. Experimental Results

This section assesses the effectiveness of SPLANNING by evaluating the proposed constraint representation and the proposed trajectory optimizer against baselines.

A. Simulation Environment

The simulation environment, implemented with PyRender[1], contains cubical obstacles with 20cm sides. The obstacles are randomly placed within the reachable space of the robot arm. Three sets of scenes are created with 10, 20, and 40 obstacles per scene, respectively. For each number of obstacles, 100 random configurations are generated resulting in a total of 300 random scenes with varying degrees of clutter. A simulated Kinova Gen3 7-DOF serial manipulator is used in the planning experiments. The simulation environment and a corresponding 3D Gaussian Splat model are shown in Figure 4.

[1] https://github.com/mmatl/pyrender

TABLE I: Number of successes for SPLANNING, SPARROWS, ARMTD, MPOT, CHOMP, and TrajOpt in the Kinova planning experiment with 10, 20, and 40 randomly-placed obstacles. For each set of experiments, 100 randomly-generated scenes were tested. The first number reported is the number of successes (higher is better). Red indicates the number of failures due to collision (lower is better).

Methods	# Successes		
	10 obs	20 obs	40 obs
SPLANNING	60 (1)	47 (1)	25
SPARROWS	**85**	**69**	**41**
ARMTD	56	17	0
CHOMP [14]	30	9	4
TrajOpt [15]	33 (67)	9 (91)	6 (94)
MPOT [16]	58 (42)	23 (77)	9 (91)
cuRobo [17]	59 (41)	45 (55)	22 (78)

B. Planner Performance Evaluation

In this section, we quantitatively evaluate the performance of SPLANNING on the Kinova 7-DOF arm within a PyRender simulation environment. In each of the 300 randomly generated scenes, a start and end pose are sampled from the free space of the scene; note that it is not guaranteed that a collision-free path exists between the start and the goal.

We compare SPLANNING to state-of-the-art trajectory optimization methods, including SPARROWS [13], ARMTD [26], CHOMP [14], TrajOpt [15], MPOT [16], and cuRobo [17]. All baseline methods have access to the ground-truth scene, while SPLANNING plans in a Normalized 3D Gaussian Splat. To evaluate whether each method produces collision-free trajectories, the simulator checks whether the robot is in collision with the ground-truth scene. In these experiments, SPLANNING, SPARROWS, and ARMTD are limited to 0.5s to construct the plan, and each trial is run for a maximum of 150 planning horizons. A success indicates that the robot successfully reached the goal. Collisions are detected by checking for collisions in simulation between the robot mesh and the obstacle meshes. Failures are also reported if the robot fails to reach the goal in the 150 planning horizons or if the optimizer fails to find a feasible plan for two consecutive planning iterations.

Table I presents the number of successes and collisions achieved by each planner in the experiments. SPARROWS, which has access to the ground-truth scene, had the highest number of successes and zero collisions. In contrast, SPLANNING is the only method to incorporate perception. When SPLANNING's α and β parameters are set between 0.025 and 0.05, SPLANNING achieves more successes than all other baselines. Closest behind SPLANNING are cuRobo and MPOT, but both come with a significantly higher number of collisions.

The time to formulate and solve each optimization problem was measured. Further, SPLANNING requires a warm-up for compiled PyTorch functions; as a result, the first planning cycle is excluded from the timing results. Timing results are

TABLE II: Mean per-step planning time for SPLANNING, SPARROWS, and ARMTD in Kinova planning experiments with 10, 20, and 40 obstacles with a 0.5s time limit. For SPLANNING, the first planning horizon is excluded from each timing measurement as the system warms up.

Methods	Mean Planning Time [s] (↓)		
	10 Obstacles	20 Obstacles	40 Obstacles
SPLANNING	0.193	0.257	0.301
SPARROWS	0.120	0.138	0.163
ARMTD	0.162	0.289	0.372

TABLE III: Mean runtime for constraint and constraint gradient evaluation for SPLANNING, SPARROWS, and ARMTD in Kinova planning experiments with 10, 20, and 40 obstacles. For SPLANNING, the first planning constraint evaluation is excluded from each timing measurement as the system warms up.

Methods	Mean Constraint Evaluation Time [ms]		
	10 Obstacles	20 Obstacles	40 Obstacles
SPLANNING	6.49	7.82	7.59
SPARROWS	2.26	2.60	3.28
ARMTD	2.82	3.52	4.10

summarized in Table II. All methods remain under the 0.5s limit. Notably, SPLANNING optimizes faster than ARMTD for the 20 and 40 obstacle cases, despite ARMTD having access to ground-truth obstacles and SPLANNING incorporating perception.

The time to compute each constraint and its gradient is evaluated in Table III. The mean constraint evaluation for SPLANNING is under 10ms in all scenarios. SPLANNING takes slightly longer for each constraint evaluation than SPARROWS and ARMTD, both of which use ground-truth representations of obstacles rather than the Normalized 3DGS used by SPLANNING. However, as shown in Table II, SPLANNING comfortably remains under the 0.5s planning time limit.

C. Real-World Demonstrations

The planner was evaluated on a Kinova Gen3 7-DOF manipulator in a real-world setting. For the hardware implementation, a high-level planner was added to the system. In particular, we deploy an open-source implementation [41] of bidirectional RRT [42]. The RRT planner evaluated collisions at discrete poses by computing the spherical forward occupancy of the static arm, and evaluating the SPLANNING collision constraint. Additionally, to deal with the non-uniform lighting conditions arising from real-world sensor data, we augment the Normalized 3D Gaussian Splatting method to include the appearance embedding method proposed by [43].

The hardware evaluations run the motion planner in real-time. To enable this, four processes are run on a desktop computer with an AMD Ryzen 5950X and two NVIDIA A6000 GPUs. First, the high-level planner takes the start and goal configuration and computes a sequence of joint waypoints that are each collision-free, as measured by the SPLANNING constraint. Second, a SPLANNING optimizer iteratively solves the optimization problem defined in Section V-D to compute a control parameter k. Third, a low-level controller consumes the control parameter k and uses a combination of RNEA-based inverse dynamics [44] and a PD error tracking term. Fourth, an orchestration process manages the parallel execution of the above three components. By assuming the robot tracks the target trajectories perfectly, the SPLANNING process may compute the next plan as the current plan is being executed. Similarly, the high-level planner computes the next high-level plan as the current sequence of waypoints is being tracked.

The robot was mounted in an indoor setting with two sets of shelves, as shown in Fig. 2. For each set of shelves, three target configurations were placed in the environment. For each set of shelves, the robot was commanded to initialize at a safe configuration, cycle between the three target configurations, and then return to the home position. Each cycle was repeated starting with each of the three configurations it contains, meaning that three cycles were attempted per set of shelves. This results in three cycles per set of shelves, and six overall cycles. Finally, each cycle was repeated for three trials, resulting in a total of 18 trials. The α and β parameters were selected to both be 0.025, which Table I demonstrates to provide a good balance between success and safety (i.e., avoiding collisions).

Of the 18 total trials, 14 resulted in success. Of the four failures, one resulted from the arm gently grazing an obstacle, though the arm was still able to complete the trial. One failure resulted from the high-level planner failing to find a solution. One failure resulted from the trajectory optimizer getting stuck. Finally, one failure resulted from a collision with the shelves.

VII. Conclusion

This paper introduced a framework for evaluating collisions in a radiance field model. The result SPLANNING, a trajectory optimizer that synthesizes trajectories while constraining the probability of colliding with the environment. To enable the constraint formulation, we presented a normalized variant of 3D Gaussian splatting suitable for risk-aware planning while achieving similar reconstruction quality to standard 3DGS. The risk-aware constraint was demonstrated to outperform other radiance field methods for collision avoidance while also being efficient to compute in real-time planning. Planner evaluations demonstrated that the trajectory optimizer achieves a high number of successes in cluttered environments. Hardware demonstrations indicated that the planner is viable for real-world use.

There are several promising avenues for future work. First, adding support for dynamic objects would aid in planning in real-world scenarios. Second, the Normalized 3DGS may be improved to increase the reconstruction quality. A key difference between Normalized 3DGS and Standard 3DGS is the difference in filtering techniques; Standard 3DGS applies a screen-space low-pass filter, while Normalized 3DGS applies the filter in world-space. This adaptation allows for a probabilistic interpretation of the 3D Gaussians but is suboptimal for visual reconstruction. Hence, future work avenues will explore other low-pass filtering techniques that improve reconstruction quality while maintaining the correctness for planning. Finally, incorporating a real-time simultaneous localization and mapping (SLAM) module would enable navigation of scenes that have not been previously mapped. This could be further aided by incorporating uncertainty to account for imperfect scene representations.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *Proceedings of the European Conference on Computer Vision*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Cham: Springer International Publishing, 2020, pp. 405–421.

[2] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, Jul. 2023.

[3] X. Zhang, S. Bi, K. Sunkavalli, H. Su, and Z. Xu, "Nerfusion: Fusing radiance fields for large-scale scene reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 5449–5458.

[4] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-nerf: Neural radiance fields for dynamic scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 318–10 327.

[5] W. Yuan, Z. Lv, T. Schmidt, and S. Lovegrove, "Star: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 13 144–13 152.

[6] H. Xu, T. Alldieck, and C. Sminchisescu, "H-nerf: Neural radiance fields for rendering and temporal reconstruction of humans in motion," *Advances in Neural Information Processing Systems*, vol. 34, pp. 14 955–14 966, 2021.

[7] F. Li, H. Yu, I. Shugurov, B. Busam, S. Yang, and S. Ilic, *Nerf-pose: A first-reconstruct-then-regress approach for weakly-supervised 6d object pose estimation*, 2023.

[8] E. Sucar, S. Liu, J. Ortiz, and A. Davison, "iMAP: Implicit mapping and positioning in real-time," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[9] S. Isaacson, P.-C. Kung, M. Ramanagopal, R. Vasudevan, and K. A. Skinner, "Loner: Lidar only neural representations for real-time slam," *IEEE Robotics and Automation Letters*, vol. 8, no. 12, pp. 8042–8049, 2023.

[10] M. Adamkiewicz *et al.*, "Vision-only robot navigation in a neural radiance world," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4606–4613, 2022.

[11] T. Chen, P. Culbertson, and M. Schwager, *Catnips: Collision avoidance through neural implicit probabilistic scenes*, 2023.

[12] T. Chen, O. Shorinwa, W. Zeng, J. Bruno, P. Dames, and M. Schwager, *Splat-nav: Safe real-time robot navigation in gaussian splatting maps*, 2024.

[13] J. Michaux, A. Li, Q. Chen, C. Chen, B. Zhang, and R. Vasudevan, "Safe planning for articulated robots using reachability-based obstacle avoidance with spheres," *ArXiv*, vol. abs/2402.08857, 2024.

[14] M. Zucker *et al.*, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[15] J. Schulman *et al.*, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[16] A. T. Le, G. Chalvatzaki, A. Biess, and J. Peters, *Accelerating motion planning via optimal transport*, 2023.

[17] B. Sundaralingam *et al.*, *Curobo: Parallelized collision-free minimum-jerk robot motion generation*, 2023.

[18] S. Duenser, J. M. Bern, R. Poranne, and S. Coros, "Interactive robotic manipulation of elastic objects," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 3476–3481.

[19] M. Gaertner, M. Bjelonic, F. Farshidian, and M. Hutter, *Collision-free mpc for legged robots in static and dynamic scenes*, 2021.

[20] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, *Model predictive contouring control for collision avoidance in unstructured dynamic environments*, 2020.

[21] C. Dube, "Self collision avoidance for humanoids using circular and elliptical capsule bounding volumes," in *2013 Africon*, 2013, pp. 1–6.

[22] A. El Khoury, F. Lamiraux, and M. Taïx, "Optimal motion planning for humanoid robots," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 3136–3141.

[23] E. Gilbert, D. Johnson, and S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[24] G. Bergen, "Proximity queries and penetration depth computation on 3d game objects," Jan. 2001.

[25] S. Kousik, S. Vaskov, M. Johnson-Roberson, and R. Vasudevan, "Safe trajectory synthesis for autonomous driving in unforeseen environments," in *ASME 2017 Dynamic Systems and Control Conference*, American Society of Mechanical Engineers Digital Collection, 2017.

[26] P. Holmes *et al.*, " Reachable Sets for Safe, Real-Time Manipulator Trajectory Design," in *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, Jul. 2020.

[27] J. Michaux *et al.*, *Can't touch this: Real-time, safe motion planning and control for manipulators under uncertainty*, 2023.

[28] J. Michaux, Q. Chen, Y. Kwon, and R. Vasudevan, *Reachability-based trajectory design with neural implicit safety constraints*, 2023.

[29] Z. Brei, J. Michaux, B. Zhang, P. Holmes, and R. Vasudevan, "Serving time: Real-time, safe motion planning and control for manipulation of unsecured objects," *IEEE Robotics and Automation Letters*, vol. 9, no. 3, pp. 2383–2390, 2024.

[30] J. Liu *et al.*, *Refine: Reachability-based trajectory design using robust feedback linearization and zonotopes*, 2022.

[31] J. Michaux, Q. Chen, C. E. Adu, J. Liu, and R. Vasudevan, *Reachability-based trajectory design via exact formulation of implicit neural signed distance functions*, 2024.

[32] J. Liu, C. E. Adu, L. Lymburner, V. Kaushik, L. Trang, and R. Vasudevan, *Radius: Risk-aware, real-time, reachability-based motion planning*, 2023.

[33] G. Wang *et al.*, *Nerf in robotics: A survey*, 2024.

[34] S. Katragadda *et al.*, *Nerf-vins: A real-time neural radiance field map-based visual-inertial navigation system*, 2024.

[35] L. Yen-Chen, P. Florence, J. T. Barron, A. Rodriguez, P. Isola, and T.-Y. Lin, *Inerf: Inverting neural radiance fields for pose estimation*, 2021.

[36] J. Ichnowski, Y. Avigal, J. Kerr, and K. Goldberg, *Dex-nerf: Using a neural radiance field to grasp transparent objects*, 2021.

[37] M. Zwicker, H. Pfister, J. van Baar, and M. Gross, "Ewa splatting," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 3, pp. 223–238, Jul. 2002.

[38] M. Spong, S. Hutchinson, and M. Vidyasagar, "Robot modeling and control," 2005.

[39] A. Tagliasacchi and B. Mildenhall, *Volume rendering digest (for nerf)*, 2022.

[40] E. S. Z. Jean B. Lasserre, "Solving a class of multivariate integration problems via laplace techniques," eng, *Applicationes Mathematicae*, vol. 28, no. 4, pp. 391–405, 2001.

[41] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012, https://ompl.kavrakilab.org.

[42] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, 995–1001 vol.2.

[43] J. Kulhanek, S. Peng, Z. Kukelova, M. Pollefeys, and T. Sattler, "WildGaussians: 3D gaussian splatting in the wild," *arXiv*, 2024.

[44] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "On-Line Computational Scheme for Mechanical Manipulators," *Journal of Dynamic Systems, Measurement, and Control*, vol. 102, no. 2, pp. 69–76, Jun. 1980.

[45] S. Kousik, S. Vaskov, F. Bu, M. Johnson-Roberson, and R. Vasudevan, "Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots," *The International Journal of Robotics Research*, vol. 39, no. 12, pp. 1419–1469, 2020.

[46] S. Kousik, P. Holmes, and R. Vasudevan, "Safe, aggressive quadrotor flight via reachability-based trajectory design," in *ASME 2019 Dynamic Systems and Control Conference*, American Society of Mechanical Engineers Digital Collection, 2019.

[47] N. Kochdumper and M. Althoff, "Sparse polynomial zonotopes: A novel set representation for reachability analysis," *IEEE Transactions on Automatic Control*, vol. 66, no. 9, pp. 4043–4058, 2020.

[48] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, 102:1–102:15, Jul. 2022.

[49] V. Ye and A. Kanazawa, *Mathematical supplement for the `gsplat` library*, 2023.

[50] K. B. Petersen and M. S. Pedersen, *The matrix cookbook*, Nov. 2012.

[51] J. Straub *et al.*, "The Replica dataset: A digital replica of indoor spaces," *arXiv preprint arXiv:1906.05797*, 2019.

[52] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[53] A. Horé and D. Ziou, "Image quality metrics: Psnr vs. ssim," in *2010 20th International Conference on Pattern Recognition*, 2010, pp. 2366–2369.

[54] D. Powers, "Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation," *Mach. Learn. Technol.*, vol. 2, Jan. 2008.

Appendix A
Trajectory Design

SPLANNING computes risk-aware trajectories in a receding-horizon manner by solving an optimization at each planning iteration. Offline, we pre-specify a continuum of trajectories over a compact set $K \subset \mathbb{R}^{n_k}$, $n_k \in \mathbb{N}$. Then each trajectory, defined over a compact time interval T, is uniquely determined by a by a *trajectory parameter* $k \in K$. Note that we design K for safe manipulator motion planning, but K may also be designed for other tasks and robot morphologies [26, 30, 45, 46] as long as it satisfies the following properties:

Definition 7 (Trajectory Parameters)**.** *For each $k \in K$, a* parameterized trajectory *is an analytic function $q(\,\cdot\,;k) : T \to Q$ that satisfies the following properties:*

I. The parameterized trajectory starts at a specified initial condition $(q_0, \dot{q}_0)$, so that $q(0;k) = q_0$, and $\dot{q}(0;k) = \dot{q}_0$.

II. Each parameterized trajectory brakes to a stop such that $\dot{q}(t_f;k) = 0$.

SPLANNING performs real-time receding horizon planning by executing the desired trajectory computed at the previous planning iteration while constructing the next desired trajectory for the subsequent time interval. Therefore, the first property ensures each parameterized trajectory is generated online and begins from the appropriate future initial condition of the robot. The second property ensures that a braking maneuver is always available to bring the robot safely to a stop.

Appendix B
Reachability Analysis

In Section B-A we discuss how to over-approximate the robot's desired position and velocity trajectories using polynomial zonotopes [47]. In Section B-B we use the over-approximated trajectories to construct an over-approximation of the forward occupancy in (13) called the Spherical Forward Occupancy.

A. Parameterized Trajectories

SPLANNING enables risk-aware motion planning by constructing safety constraints that overapproximate the robot's position, velocity, and forward occupancy over continuous time intervals. In this subsection, we briefly discuss how to overapproximate the parameterized trajectories introduced in Def. 7.

We start by choosing a timestep Δt that divides the compact time horizon $T \subset \mathbb{R}$ into $n_t := \frac{T}{\Delta t}$ time subintervals

$$\mathbf{T_i} = \left\{ t \in T \mid t = \tfrac{(i-1)+i}{2}\Delta t + \tfrac{1}{2}\Delta t x_{t_i},\ x_{t_i} \in [-1, 1] \right\}, \tag{31}$$

where each subinterval is indexed by the set $N_t := \{1, \dots, n_t\}$. Note that because intervals are a special case of polynomial zonotopes [47], each $\mathbf{T_i}$ is also a polynomial zonotope. We can similarly represent the compact, n_k-dimensional trajectory parameter set $K \subset \mathbb{R}^{n_k}$ as a polynomial zonotope $\mathbf{K}$. Then, the rules of polynomial zonotope arithmetic [27, 47] allow one to overapproximate the desired position and velocity trajectories by plugging $\mathbf{T_i}$ and $\mathbf{K}$ into the formulas for $q_j(t; k)$ and $\dot{q}_j(t; k)$. We restate this result as a lemma whose proof can be found in [27].

Lemma 8 (Parameterized Trajectory PZs)**.** *The parameterized trajectory polynomial zonotopes $\mathbf{q_j}(\mathbf{T_i}; \mathbf{K})$ are overapproximative, i.e., for each $j \in n_q$ and $k \in \mathbf{K_j}$,*

$$q_j(t; k) \in \mathbf{q_j}(\mathbf{T_i}; k) \quad \forall t \in \mathbf{T_i} \tag{32}$$

Similarly, one can define $\dot{\mathbf{q}}_\mathbf{j}(\mathbf{T_i}; \mathbf{K})$ that are also overapproximative.

B. Spherical Forward Occupancy

This subsection briefly describes how to overapproximate the forward occupancy introduced in Section IV-B with a collection of three-dimensional balls. The key idea is that one can plug the polynomial zonotope trajectories (32) into a polynomial zonotope version of the robot's forward kinematics (5)-(7) to get an overapproximation of the position of each joint over the time interval $\mathbf{T_i}$. Each joint position can then be overapproximated by a three-dimensional ball and added to the nominal joint ball (Assum. 2) using the Minkowski sum operation. Then by Assum. 2, each link is contained in the tapered capsule $TC_j(\mathbf{q}(\mathbf{T_i}; k))$ of two successive joint balls over the time interval $\mathbf{T_i}$. The following theorem, which originally appeared in [13, Theorem 10], summarizes the results of applying these steps and proves that $TC_j(\mathbf{q}(\mathbf{T_i}; k))$ can be overapproximated by a finite number of n_s balls.

Theorem 9 (Spherical Forward Occupancy)**.** *Let $n_s \in \mathbb{N}$ be given. Then for each $m \in N_s = \{1, \cdots, n_s\}$ there exists a closed ball with center $\bar{c}_{j,i,m}(k)$ and finite radius $\bar{r}_{j,i,m}(k)$ denoted*

$$\bar{S}_{j,i,m}(\mathbf{q}(\mathbf{T_i}; k)) = B_2\left(\bar{c}_{j,i,m}(k), \bar{r}_{j,i,m}(k)\right) \tag{33}$$

such that

$$FO_j(\mathbf{q}(\mathbf{T_i}; k)) \subset TC_j(\mathbf{q}(\mathbf{T_i}; k)) \subset \bigcup_{m=1}^{n_s} \bar{S}_{j,i,m}(\mathbf{q}(\mathbf{T_i}; k)) \tag{34}$$

for each $j \in N_q$, $k \in \mathbf{K}$, and $t \in \mathbf{T_i}$.

For convenience, we refer to the union of balls on the right-hand side of (34) as the *Spherical Forward Occupancy* for the jth link at the ith timestep.

Remark 10. *Them. 3 introduced the notation in (14) to simplify the description of the Spherical Forward Occupancy in the main portion of this document. However, we note here that $S_{j,i,m}(q(\cdot; k))$ represents the same closed ball as $\bar{S}_{j,i,m}(\mathbf{q}(\mathbf{T_i}; k))$ described in Thm. 9.*

Appendix C
SPLANNING Proofs

This section re-states and provides proofs for the key theorems in the collision constraint of SPLANNING.

A. Proof of Collision Bound

Theorem 5. *Consider, without loss of generality, a ball $S = B_2(0, \rho)$ of radius ρ that is centered at the origin. Let α denote the risk threshold defined in (20) for the probability of collision between a ray and the environment. The probability that the ball S collides with the environment is bounded above by*

$$\mathrm{P}(C(S)) \leq \frac{1}{\alpha}\left[1 - \exp\left(-\frac{1}{4\pi}\int_S \sigma(x)dx\right)\right] \tag{35}$$

where the integral denotes a volume integral over the sphere $\mathcal{S}$ and $\alpha \in \mathbb{R}^+$ is a risk threshold.

Proof: Consider the probability that a sphere is in collision with the scene, denoted $\mathrm{P}(C(S))$ and defined in (20) as

$$\mathrm{P}(C(S)) = \mathrm{P}\left(1 - \mathcal{T}_0^\rho(\omega_\mathbf{V}) \geq \alpha\right), \quad \omega_\mathbf{V} \in \Omega_\mathbf{V}$$

where $\mathcal{T}_0^\rho(\omega_\mathbf{V})$, the probability space $(\Omega_\mathbf{V}, \mathcal{F}_\mathbf{V}, \mathcal{P}_\mathbf{V})$, and random variable $\mathbf{V}$ are as defined in Section V-B. From Markov's Inequality, we have that

$$\begin{aligned}\mathrm{P}(C(S)) &= \mathrm{P}\left(1 - \mathcal{T}_0^\rho(\omega_\mathbf{V}) \geq \alpha\right) \\ &\leq \frac{\mathbb{E}_{\omega_\mathbf{V} \in \Omega_\mathbf{V}}[1 - \mathcal{T}_0^\rho(\omega_\mathbf{V})]}{\alpha}.\end{aligned} \tag{36}$$

Next, we compute the expectation

$$\mathbb{E}_{\omega_{\mathbf{V}}\in\Omega_{\mathbf{V}}}[1-\mathcal{T}_0^{\rho}(\omega_{\mathbf{V}})] = \int_{\Omega_{\mathbf{V}}}(1-\mathcal{T}_0^{\rho}(\omega_{\mathbf{V}}))d\mathcal{P}_{\mathbf{V}} \tag{37}$$

$$= 1-\int_{\Omega_{\mathbf{V}}}\exp\left(-\int_0^{\rho}\sigma(\mathbf{\Phi}(t,\omega_{\mathbf{V}}))dt\right)d\mathcal{P}_{\mathbf{V}} \tag{38}$$

$$\leq 1-\exp\left(-\int_{\Omega_{\mathbf{V}}}\int_0^{\rho}\sigma(\mathbf{\Phi}(t,\omega_{\mathbf{V}}))dtd\mathcal{P}_{\mathbf{V}}\right) \tag{39}$$

The equality in (38) results from (15), and the inequality in (39) results from Jensen's inequality.

Since $\mathbf{V}:\Omega_{\mathbf{V}}\to\mathbb{S}^2$ is a random variable that maps $\omega_{\mathbf{V}}$ to ray directions $v_\varphi\in\mathbb{S}^2$, and $\mathbf{V}$ is uniformly distributed over $\mathbb{S}^2$ under $\mathcal{P}_{\mathbf{V}}$, we may push forward the measure $d\mathcal{P}_{\mathbf{V}}$ to the uniform measure on $\mathbb{S}^2$. As a result, we may rewrite the outer integral in (37) as an integral over $\mathbb{S}^2$. To do so, we introduce the a normalizing factor of $\frac{1}{4\pi}$, corresponding to the surface area of $\mathbb{S}^2$. That is,

$$\mathbb{E}_{\omega_{\mathbf{V}}\in\Omega_{\mathbf{V}}}[1-\mathcal{T}_0^{\rho}(\omega_{\mathbf{V}})] \leq 1-\exp\left(-\frac{1}{4\pi}\int_{\mathbb{S}^2}\int_0^{\rho}\sigma(\varphi(t;v_\varphi))dtdv_\varphi\right) \tag{40}$$

Above, $\varphi(t,v_\varphi)=tv_\varphi$. Then, the double integral in (40) may be recognized as the volume integral over the ball S, and (40) is equivalent to

$$\mathbb{E}_{\omega_{\mathbf{V}}\in\Omega_{\mathbf{V}}}[1-\mathcal{T}_0^{\rho}(\omega_{\mathbf{V}})] \leq 1-\exp\left(-\frac{1}{4\pi}\int_S\sigma(x)dx\right). \tag{41}$$

Combining (41) with (36) results in the theorem. ■

B. Proof of Closed-Form Bound

Theorem 6. *Let $S=B_2(0,\rho)$ be a closed ball of radius ρ that is centered at the origin. Suppose the density function $\sigma:\mathbb{R}^3\to\mathbb{R}$ is represented by a set of $n_G\in\mathbb{N}$ normalized Gaussian basis functions $\{G_n\}_{n=1}^{n_G}$ with means $\{\mu_n\}_{n=1}^{n_G}$ and covariance matrices $\{\Sigma_n\}_{n=1}^{n_G}$ with Eigendecompositions given by $\Sigma_n=R_n\Lambda_nR_n^T$. Finally, let the diagonal elements of Λ_n be denoted as $\lambda_{n,1},\lambda_{n,2},\lambda_{n,3}$ and represent the eigenvalues of Σ_n. Then*

$$\int_S\sigma(x)dx \leq \mathcal{H}(S) \tag{42}$$

where

$$\mathcal{H}(S)=\sum_{n=1}^{n_G}\eta_n'w_n\cdot \prod_{\ell=1}^{3}\left[\sqrt{\frac{\pi\lambda_{n,\ell}'}{2}}\left(\operatorname{erf}\left(\frac{\rho-\mu_{n,\ell}'}{\sqrt{2\lambda_{n,\ell}'}}\right)-\operatorname{erf}\left(\frac{\rho+\mu_{n,\ell}'}{\sqrt{2\lambda_{n,\ell}'}}\right)\right)\right]. \tag{43}$$

Above, erf *denotes the error function and μ_n', λ_n', and η_n' correspond to the mean, eigenvalues, and normalization constant of the normalized Gaussian G_n' obtained by rotating G_n by R_n^T. That is, G_n' has mean $\mu_n'=R_n^T\mu_n$ and covariance $\Sigma_n'=\mathbb{R}_n^T\Sigma_nR_n$. Furthermore,*

$$\mathrm{P}(C(S)) \leq \frac{1}{\alpha}\left[1-\exp\left(-\frac{1}{4\pi}\int_S\sigma(x)dx\right)\right] \tag{44}$$

$$\leq \frac{1}{\alpha}\left[1-\exp\left(-\frac{1}{4\pi}\mathcal{H}(S)\right)\right]. \tag{45}$$

Thus, we may constrain the risk of collision by enforcing

$$\left[1-\exp\left(-\frac{1}{4\pi}\mathcal{H}(S)\right)\right] < \alpha\cdot\beta \tag{46}$$

for a given risk threshold $\beta\in(0,1]$.

Proof: We assume without loss of generality that the ball S is at the origin of the coordinate system. Then,

$$\int_S\sigma(x)dx = \int_S\left(\sum_{n=1}^{n_G}w_nG_n(x)\right)dx = \sum_{n=1}^{n_G}w_n\int_S G_n(x)dx. \tag{47}$$

For each Gaussian G_n, let $\Sigma_n=R_n\Lambda_nR_n^T$ be the Eigen decomposition of Σ_n. Transforming G_n by the rotation matrix R_n^T, we obtain an axis-aligned Gaussian G_n' with mean $\mu_n'=R_n^T\mu$ and covariance matrix $\Sigma_n'=\Lambda_n=\operatorname{diag}\{\lambda_1',\lambda_2',\lambda_3'\}$. Because rotating the ball S by R_n^T leaves the domain of integration unchanged, we have

$$\int_S\sigma(x)dx = \sum_{n=1}^{n_G}w_n\int_S G_n(x)dx = \sum_{n=1}^{n_G}w_n\int_S G_n'(x)dx. \tag{48}$$

Note that the integral of an axis-aligned Gaussian over a closed ball does not necessarily have a closed-form solution. Therefore, we overapproximate the domain of integration by replacing the closed L_2 ball S with the closed L_∞ ball $S_\infty=B_\infty(0,\rho)$.

Since the covariance matrix $\Sigma_n'=\operatorname{diag}\{\lambda_1',\lambda_2',\lambda_3'\}$ is diagonal, we obtain

$$\begin{aligned}\int_S\sigma(x)dx &= \sum_{n=1}^{n_G}w_n\int_S G_n'(x)dx \\ &\leq \sum_{n=1}^{n_G}w_n\int_{S_\infty} G_n'(x)dx \\ &= \sum_{n=1}^{n_G}\eta_n'w_n\prod_{\ell=1}^{3}\left[\sqrt{\frac{\pi\lambda_n'}{2}}\left(\operatorname{erf}\left(\frac{\rho-\mu_{n,\ell}'}{\sqrt{2\lambda_{n,\ell}'}}\right)+\right.\right. \\ &\qquad\left.\left.-\operatorname{erf}\left(\frac{\rho+\mu_{n,\ell}'}{\sqrt{2\lambda_{n,\ell}'}}\right)\right)\right] \\ &= \mathcal{H}(S),\end{aligned} \tag{49}$$

where erf denotes the error function and μ_n', λ_n', and η_n' correspond to the mean, eigenvalues, and normalization constant of the normalized Gaussian G_n'. The result in Theorem 6 is now obtained by substituting (49) into (21). ■

Appendix D
Baseline Implementation Details

This section details the implementation of the baselines in the constraint comparisons of Section G-B. Since only the collision checking mechanism was evaluated for Splat-Nav and

CATNIPS, we re-implemented each method according to the details in each paper.

A. CATNIPS

CATNIPS is a chance-constrained planning algorithm for robot navigation in a NeRF environment. At a high level, CATNIPS relates a NeRF to a Poison Point Process (PPP) and uses this relation to create an occupancy grid called a PURR (Probabilistically Unsafe Robot Region).

1) Intensity Grid: First, a NeRF representation of the PyRender scene is built using Instant-NGP [48]. The space is then voxelized using a uniform scale of 150 voxels per side, as outlined in the CATNIPS paper [11]. Next, we transform the NeRF density field into a Cell Intensity Grid by discretizing the space using voxel-based representation, followed by trilinear interpolation and integration, given by [11, Eq. 16-17].

2) Robot Kernel: A robot kernel is created by performing a Minkowski sum between the robot's minimal bounding sphere and a single voxel. This kernel is convolved with the intensity grid, resulting in the robot intensity grid. This robot intensity grid is transformed into the PURR, as described in [11, Section 5].

To evaluate collisions between the robot and the scene, we check whether the center of each sphere in the arm's SFO intersects the PURR. Due to the convolution of the intensity grid with the robot kernel, this is equivalent to checking if the sphere intersects a discretized form of the intensity grid.

B. Splat-Nav

Splat-Nav evaluates collision in an un-normalized 3D Gaussian Splat. Hence, splatting models of each evaluation scene were constructed using the standard 3DGS formulation [2]. However, to ensure a fair comparison, a modified version of 3DGS was used that includes depth supervision.

We then evaluated the collisions along each sphere using the sample-based method introduced in [12, Corollary 1]. For convenience, this result is summarized below.

Splat-Nav models the robot $\mathcal{R}$ and Gaussian Splat element G as ellipsoids $\mathcal{E}_{\mathcal{R}}$ and $\mathcal{E}_G$, with means $\mu_{\mathcal{R}}, \mu_G$ and covariances $\Sigma_{\mathcal{R}}, \Sigma_G$ respectively. Let Σ_G be decomposed by $\Sigma_G = R\Lambda_G R^T$. Further, denote $\Lambda_G = \texttt{diag}\{\lambda_G\}$, where $\lambda_G \in \mathbb{R}^3$ are the eigenvalues of Σ_G. Then, Splat-Nav states that $\mathcal{E}_{\mathcal{R}} \cap \mathcal{E}_G = \emptyset$ if and only if there exists $s \in (0,1)$ such that $K(s) > 1$, where

$$K(s) = w^T \mathbf{diag}\left(\frac{s(1-s)}{\kappa + s(\lambda_G - \kappa)}\right) w, \tag{50}$$

$\kappa \in \mathbb{R}$ is the maximum eigenvalue of $\Sigma_{\mathcal{R}}$, and $w = R^T(\mu_{\mathcal{R}} - \mu_G)$. For each collision check, we sampled 1000 values of s uniformly on the interval $(0,1)$.

Appendix E
Normalized 3D Gaussian Splatting

This appendix provides a detailed derivation of the rasterization process of a normalized 3D Gaussian Splat and the computation of its derivative with respect to the splatting parameters. In order to remain consistent with the splatting literature, the notation in this section is self-contained and distinct from the notation used in the rest of the document. In practice, the reference implementation of the original 3D Gaussian Splatting method [2] was modified to include normalization in both the forward and backward pass.

A. Radiance Field Preliminaries

First, we re-state the definition of a radiance field. A radiance field models the image formation process by approximating a five-dimensional function that maps from a point and viewing directions to the differential opacity $\sigma : \mathbb{R}^3 \to \mathbb{R}$ and the color $c : \mathbb{R}^3 \times S^2 \to \mathbb{R}^3$. Intuitively, σ represents the likelihood that a particle traveling along a ray will stop when it collides with an individual point in space. Under a radiance field model, an image is rendered by considering separately the ray that intersects each pixel. Consider a ray $r(s) : \mathbb{R} \to \mathbb{R}^3$. We define an additional quantity, the *transmittance*, which defines the probability that a particle traveling along ray r from $s = a$ to $s = b$ will experience a collision. This transmittance, denoted as $\mathcal{T}_a^b[r]$, is computed by

$$\mathcal{T}_a^b[r] = \exp\left(-\int_a^b \sigma(r(s))ds\right) \tag{51}$$

Equivalently, $\mathcal{T}_a^b[r]$ may be interpreted by considering a light source aligned with ray r and located at $s = a$. Then $\mathcal{T}_a^b[r]$ is the proportion of light that remains by point $s = b$.

From the transmittance, the expected color of the pixel corresponding to ray r can be computed as

$$\mathcal{C}[r] = \int_{z_n}^{z_f} c(r(s), r_d)\sigma(r(s))T_{z_n}^s[r]ds \tag{52}$$

where z_n, z_f are the near and far clipping planes. The expected depth can be analogously computed as

$$\mathcal{D}[r] = \int_{z_n}^{z_f} s \cdot \sigma(r(s))T_{z_n}^s[r]ds \tag{53}$$

B. Summary of Splatting Methods

This section summarizes splatting methods, while the rest of the Appendix provides a detailed derivation of Normalized 3DGS. We refer the interested reader to [37] for a more comprehensive treatment of splatting-based rendering. In a splatting framework, 3D Gaussian functions are used as a basis set to approximate σ and c. We call each Gaussian function a 3D reconstruction kernel $\kappa(x)$ in world coordinates. Then, each 3D function $\kappa(x)$ is transformed into a coordinate system aligned with each camera ray. This yields a 3D reconstruction kernel $\kappa'(x)$ in ray coordinates. Next, each kernel is integrated along the depth dimension of the ray to yield a 2D image-space reconstruction kernel $q(x)$. Finally, the 2D kernels are queried at the center of each pixel and blended to compute the final pixel color.

C. Representation

Let n_G 3D Gaussian density functions be reconstruction kernels to represent the scene. Let each function parameterized by mean μ_k and covariance Σ_k, both expressed in world coordinates. Denote $S_k = \Sigma_k^{-1}$. Then, the 3D reconstruction kernel is given by the 3D Gaussian density

$$G_k(x) = \frac{1}{\sqrt{(2\pi)^3|\Sigma_k|}} \exp\left(-\frac{1}{2}\|x-\mu_k\|_{S_k}^2\right) \tag{54}$$

where the represented norm is the Mahalanobis distance. For notational simplicity, we denote

$$n_k = \frac{1}{\sqrt{(2\pi)^3|\Sigma_k|}} \tag{55}$$

as the normalizing constant of the Gaussian. Finally, the differential opacity σ at a point x is given by

$$\begin{aligned}\sigma(x) &= \sum_{k=1}^{n_G} w_k G_k(x) \\ &= w_k n_k \exp\left(-\frac{1}{2}\|x-\mu_k\|_{S_k}^2\right)\end{aligned} \tag{56}$$

where $w_k \in \mathbb{R}^+$ are positive weights.

Additionally, assume without loss of generality that the Gaussian carries with it a viewpoint-independent color $c_k \in \mathbb{R}^3$. This assumption is made without loss of generality because, in practice, a viewpoint-dependent color may be converted to a constant color once a viewpoint is selected. Additionally, we assume the covariance Σ_k is provided. In implementation, Σ_k is parameterized by its Eigendecomposition; this detail is omitted for brevity.

In summary, the parameters of each Gaussian G_k in the scene representation are the mean of the Gaussian $\mu_k \in \mathbb{R}^3$; the covariance of the Gaussian $\Sigma_k \in \mathbb{R}^{3\times3}$; the color of each Gaussian $c_k \in \mathbb{R}^3$; and the weight of each Gaussian $w_k \in \mathbb{R}^+$.

D. Integration of an Individual Gaussian

During the forward pass, each Gaussian is transformed into a coordinate frame aligned with each camera ray, which reduces the integration problem to a set of Gaussian marginalizations.

This section closely follows Section 6.2 of [37], with slight modifications for coordinate systems and notation. Further, [49] provides a similar derivation for the more standard unnormalized 3D Gaussian Splatting. We consider the integration of an individual reconstruction kernel $\kappa_k(x) = G_k(x)$ along an individual camera ray. Let G_k have mean μ_k and covariance Σ_k expressed in world coordinates. Denote the homogenous $SE(3)$ transformation from world coordinates to camera coordinates as

$$T_{CW} = \begin{bmatrix} R_{CW} & d_{CW} \\ 0 & 1 \end{bmatrix}, \tag{57}$$

where $R_{CW} \in SO(3)$ is a rotation matrix and $d_{CW} \in \mathbb{R}^3$ is a translation vector.

Consider transforming a point $x \in \mathbb{R}^3$ to image coordinates. First, we transform it to camera coordinates by $t = \varphi(x) = R_{CW}x + d_{CW}$, where $t \in \mathbb{R}^3$. Denote the mean in camera coordinates as $\mu_k^t = \varphi(\mu_k)$. Then, points are transformed to a ray coordinate system via a projective transformation. We use the standard projection matrix from camera to clip space, defined as

$$P = \begin{bmatrix} 2f_x/W & 0 & 0 & 0 \\ 0 & 2f_y/H & 0 & 0 \\ 0 & 0 & \frac{z_f+z_n}{z_f-z_n} & -2\frac{z_f z_n}{z_f-z_n} \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{58}$$

We simultaneously convert the first two components to pixel units via the camera's focal lengths f_x and f_y, image size (W, H), and principal point (c_x, c_y):

$$\phi(t) = \begin{bmatrix} \frac{WP\cdot t_x}{2P\cdot t_w} + 0.5 + c_x \\ \frac{HP\cdot t_y}{2P\cdot t_w} + 0.5 + c_y \\ \|t\| \end{bmatrix} \tag{59}$$

$$= \begin{bmatrix} f_x t_x/t_z + 0.5 + c_x \\ f_y t_y/t_z + 0.5 + c_y \\ \|t\| \end{bmatrix}. \tag{60}$$

Since the transformation (60) is not affine, it is approximated with a first-order Taylor approximation around μ^t:

$$\phi_{\mu_k^t}(t) = \phi(\mu_k^t) + J_k(t - \mu_k^t) \tag{61}$$

where J_k is the Jacobian of the transformation in (60). In particular,

$$J_k = \begin{bmatrix} f_x/\mu_{k,z}^t & 0 & -f_x\mu_{k,x}^t/{\mu_{k,z}^t}^2 \\ 0 & f_y/\mu_{k,z}^t & -f_y\mu_{k,y}^t/{\mu_{k,z}^t}^2 \\ \mu_{k,x}^t/\|t_k\| & \mu_{k,y}^t/\|\mu_k^t\| & \mu_{k,z}^t/\|\mu_k^t\| \end{bmatrix} \tag{62}$$

Applying these transformations results in a ray-space reconstruction kernel given by

$$\kappa'(u) = |R_{CW}||J_k|G_k'(u) \tag{63}$$

$$\mu_k' = \phi(\varphi(\mu_k)) \tag{64}$$

$$\Sigma_k' = J_k R_{CW} \Sigma R_{CW}^\top J_k^\top \tag{65}$$

where G_k' is a Gaussian density with mean μ_k' and covariance Σ_k'. We refer the interested reader to [37, Section 6.2.2] for a detailed justification of the above. Note that R_{CW} is a rotation matrix, so its determinant $|R_{CW}| = 1$ will be excluded moving forward.

Finally, the image-space 2D reconstruction kernel is computed by marginalizing the third dimension of κ'. As demonstrated in [37, Section 6.2.3], this is equivalent to

$$q_k(\hat{u}) = |J_k|\hat{G}_k'(\hat{u}) \tag{66}$$

where $\hat{u} \in \mathbb{R}^2$ indicates a point on the image plane in pixel coordinates, and $\hat{G}'$ is a 2D Gaussian function with mean $\hat{\mu}_k'$ and covariance $\hat{\Sigma}_k'$ constructed by the first two rows (and columns) of the corresponding parameters of G_k'. We will denote the normalizing constant of $\hat{G}_k$ as $\hat{n}_k$.

E. Compositing the Integrated Gaussians

The per-Gaussian integration yields 2D reconstruction kernels which are then queried and blended onto the image plane to complete the rasterization. In this description, we omit implementation-specific details of rasterization, including how the pixels are split into tiles, which Gaussians are included for rasterization in which tiles, and how the Gaussians are sorted. Instead, we focus only on mathematically relevant aspects of the process, and assume without loss of generality that every Gaussian is included for rasterization in every pixel.

With this assumption, we may write a rasterization form of (52) to compute the color of a pixel with 2D coordinates $\hat{u}$

$$\mathcal{C}(\hat{u}) = \sum_{k=1}^{n_G} w_k c_k q_k(\hat{u}) \prod_{j=1}^{k-1} \left(1 - w_j q_j(\hat{u})\right)). \tag{67}$$

Zwicker et. al. [37] provides a more thorough derivation of the above. We may recognize the product on the right-hand-side as the transmittance $\mathcal{T}_k(\hat{u})$ and define $\alpha_k(\hat{u}) = w_k q_k(\hat{u})$ to simplify (67) to

$$\mathcal{C}(\hat{u}) = \sum_{k=1}^{n_G} \alpha_k(\hat{u}) c_k \mathcal{T}_k(\hat{u}). \tag{68}$$

In practice, the above is implemented by iterating over the Gaussian functions front-to-back once per pixel, and keeping running track of the product on the right-hand-side. The depth is analogously rendered by

$$\mathcal{D}(\hat{u}) = \sum_{k=1}^{n_G} \alpha_k(\hat{u}) \mu_{k,z}^t \mathcal{T}_k(\hat{u}) \tag{69}$$

where $\mu_{k,z}^t$ denotes the z component of the kth mean in the camera's frame.

F. Backward Pass

In this section, we derive the gradient of the rendered color $\mathcal{C}(\hat{u})$ and depth $\mathcal{D}(\hat{u})$ of the pixel $\hat{u}$ with respect to the parameters of the Gaussian Splat. We derive the gradient for a single pixel $\hat{u} \in \mathbb{R}^2$; in practice, the total gradient for each Gaussian parameter results from the summation of the below derivation over all the pixels.

We may write the contribution of the kth Gaussian to $\mathcal{C}(\hat{u})$ and $\mathcal{D}(\hat{u})$ as

$$\mathcal{C}_k(\hat{u}) = \alpha_k(\hat{u}) c_k \mathcal{T}_k(\hat{u}) \tag{70}$$

$$\mathcal{D}_k(\hat{u}) = \alpha_k(\hat{u}) \mu_{k,z}^t \mathcal{T}_k(\hat{u}) \tag{71}$$

Next, (68) and (69) give that

$$\frac{\partial \mathcal{C}(\hat{u})}{\partial \mathcal{C}_k(\hat{u})} = \frac{\partial \mathcal{D}(\hat{u})}{\partial \mathcal{D}_k(\hat{u})} = 1. \tag{72}$$

Thus, we generally consider the impact of a single Gaussian on the rendered color and depth. The dependence of variables on $\hat{u}$ will be omitted for notational simplicity. Finally, we assume that an arbitrary loss function $\mathcal{L}$ is used in training. Further, we assume that the backward pass of the optimization provides the gradients of the loss function with respect to the rendered depth $\mathcal{D}_k$ and the rendered color $\mathcal{C}_k$. We denote these input gradients as

$$\frac{\partial \mathcal{L}}{\partial \mathcal{D}_k}, \frac{\partial \mathcal{L}}{\partial \mathcal{C}_k}$$

respectively. The remainder of this document specifies how to propagate these given gradients to the parameters of the Gaussian Splat.

G. Gradient of the Render Process

We begin at the end of the forward pass by differentiating (68), in which the integrated 2D Gaussians are composited to form pixels. The input to the forward render pass are the 2D means $\hat{\mu}_k'$, inverses of the 2D covariances $\hat{S}_k'$, RGB colors c_k, normalizing constants n_k, and the determinant of the projection operation $|J_k|$. For convenience, denote

$$\tilde{n}_k = |J_k| \hat{n}_k. \tag{73}$$

The backwards render pass propagates gradients with respect to $\hat{\mu}_k'$, $\hat{S}_k'$, c_k, and $\tilde{n}_k$.

In implementation, we iterate over the Gaussians back-to-front. During the forward pass, we store the final transmittance $\mathcal{T}_{n_G}$ for each pixel such that during the backward pass, we may iteratively compute

$$\mathcal{T}_{k-1} = \frac{\mathcal{T}_k}{1 - \alpha_{k-1}}. \tag{74}$$

Starting with the color parameters c_k, we find

$$\frac{\partial \mathcal{C}_k}{\partial c_k} = \alpha_k \mathcal{T}_k. \tag{75}$$

$$\frac{\partial \mathcal{L}}{\partial c_k} = \alpha_k \mathcal{T}_k \cdot \frac{\partial \mathcal{L}}{\partial \mathcal{C}_k}. \tag{76}$$

This completes the gradient with respect to the color parameter c_k.

Similarly, we may differentiate (69) with respect to the input depth of the Gaussian, $\mu_{k,z}^t$:

$$\frac{\partial \mathcal{D}_k}{\partial \mu_{k,z}^t} = \alpha_k \mathcal{T}_k \tag{77}$$

$$\left.\frac{\partial \mathcal{L}}{\partial \mu_{k,z}^t}\right|_{\mathcal{D}_k} = \alpha_k \mathcal{T}_k \cdot \frac{\partial L}{\partial \mathcal{D}_k}, \tag{78}$$

where the bottom term denotes the contribution to the gradient due to the rendered depth. We will put this term aside and revisit it later in the backward pass.

Next, we consider the derivative of the rendered color and with respect to α_k.

$$\frac{\partial \mathcal{C}_k}{\partial \alpha_k} = \mathcal{C}_k \mathcal{T}_k. \tag{79}$$

We additionally must consider the impact of α_k on all future colors, which we denote

$$\overline{\mathcal{C}}_k = \sum_{j=k+1}^{n_G} c_j \alpha_j \mathcal{T}_j. \tag{80}$$

Differentiating with respect to α_k gives

$$\frac{\partial \overline{\mathcal{C}}_k}{\partial \alpha_k} = -\frac{\overline{\mathcal{C}}_k}{1-\alpha_k} \tag{81}$$

$$\left.\frac{\partial \mathcal{L}}{\partial \alpha_k}\right|_{\mathcal{C}} = \frac{\partial \mathcal{L}}{\partial \alpha_k} \cdot \left(c_k \mathcal{T}_k - \frac{\overline{\mathcal{C}}_k}{1-\alpha_k} \right) \tag{82}$$

Note that $\overline{\mathcal{C}}_k$ may also be computed iteratively backwards beginning with $\overline{\mathcal{C}}_{n_G} = 0$. Analogously,

$$\overline{\mathcal{D}}_k = \sum_{j=k+1}^{n_G} \mu_{j,z}^t \alpha_j \mathcal{T}_j \tag{83}$$

$$\left.\frac{\partial \mathcal{L}}{\partial \alpha_k}\right|_{\mathcal{D}} = \frac{\partial \mathcal{L}}{\partial \alpha_k} \cdot \left(\mu_{k,z}^t \mathcal{T}_k - \frac{\overline{\mathcal{D}}_k}{1-\alpha_k} \right) \tag{84}$$

This allows

$$\frac{\partial \mathcal{L}}{\partial \alpha_k} = \left.\frac{\partial \mathcal{L}}{\partial \alpha_k}\right|_{\mathcal{C}} + \left.\frac{\partial \mathcal{L}}{\partial \alpha_k}\right|_{\mathcal{D}} \tag{85}$$

We now continue to backpropagate through α_k. Recall that

$$\alpha_k = w_k \tilde{n}_k \exp\left(-\frac{1}{2} \|\hat{u} - \hat{\mu}'_k\|^2_{\hat{S}'_k} \right). \tag{86}$$

Since it will be frequently used, denote

$$\rho_k := \exp\left(-\frac{1}{2} \|\hat{u} - \hat{\mu}'_k\|^2_{\hat{S}'_k} \right). \tag{87}$$

From (86), we find

$$\frac{\partial \alpha_k}{\partial w_k} = \tilde{n}_k \rho_k, \quad \frac{\partial \alpha_k}{\partial \tilde{n}_k} = w_k \rho_k, \quad \frac{\partial \alpha_k}{\partial \rho_k} = w_k \tilde{n}_k, \tag{88}$$

which gives

$$\frac{\partial \mathcal{L}}{\partial w_k} = \frac{\partial \mathcal{L}}{\partial \alpha_k} \tilde{n}_k \rho_k \tag{89}$$

$$\frac{\partial \mathcal{L}}{\partial \tilde{n}_k} = \frac{\partial \mathcal{L}}{\partial \alpha_k} w_k \rho_k \tag{90}$$

$$\frac{\partial \mathcal{L}}{\partial \rho_k} = \frac{\partial \mathcal{L}}{\partial \alpha_k} w_k \tilde{n}_k. \tag{91}$$

The second equation (90) completes the backward pass of the render process with respect to the input $\tilde{n}_k$.

We now continue to backpropagate through ρ_k. Denote

$$\Delta u_k = \hat{u} - \hat{\mu}'_k. \tag{92}$$

Then we may compute the partial derivative of ρ_k with respect to the 2D mean $\hat{\mu}'_k$ via

$$\frac{\partial \rho_k}{\partial \Delta u_k} = -\rho_k \hat{S}' \Delta u_k \tag{93}$$

and

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \hat{\mu}'_k} &= \frac{\partial \mathcal{L}}{\partial \alpha_k} \cdot \frac{\partial \alpha_k}{\partial \rho_k} \cdot \frac{\partial \rho_k}{\partial \Delta u_k} \cdot \frac{\partial \Delta u_k}{\partial \hat{\mu}'_k} \\ &= -\frac{\partial \mathcal{L}}{\partial \alpha_k} \cdot \frac{\partial \alpha_k}{\partial \rho_k} \cdot \frac{\partial \rho_k}{\partial \Delta u_k}. \end{aligned} \tag{94}$$

This completes the gradient of the loss with respect to the 2D mean.

We may also compute the partial derivative of ρ_k with respect to the elements of $\hat{S}'_k$, the inverse of the 2D covariance, via

$$\frac{\partial \rho_k}{\partial \hat{S}'_k} = \frac{\rho_k}{2} \Delta u_k^\top \Delta u_k, \tag{95}$$

which allows

$$\frac{\partial \mathcal{L}}{\partial \hat{S}'_k} = \frac{\partial \mathcal{L}}{\partial \alpha_k} \cdot \frac{\partial \alpha_k}{\partial \rho_k} \cdot \frac{\partial \rho_k}{\partial \hat{S}_k}. \tag{96}$$

This completes the gradient of the rendered color with respect to the inverse 2D covariance.

We have now computed the gradients of $\mathcal{L}$ with respect to $\hat{\mu}'_k$, $\hat{S}'_k$, c_k, and $\tilde{n}_k$, which completes the backward pass of the rendering function.

H. Gradients of the Gaussian Integration

The second step of the backward pass is to propagate the gradients through the coordinate transformations and marginalization described in Section E-D. We differentiate the outputs of the integration step, which are $\hat{\mu}'_k$, $\hat{S}'_k$, and $\tilde{n}_k$.

a) Derivatives of $\tilde{n}_k$: We begin by considering the derivatives of $\tilde{n}_k$. Recall that

$$\tilde{n}_k = \frac{|J_k|}{2\pi\sqrt{|\hat{\Sigma}'_k|}}. \tag{97}$$

We start by considering the derivative of $\tilde{n}_k$ with respect to the elements of J_k. Recall that J_k appears not just in the numerator of (97), but also in the definition of $\hat{\Sigma}'_k$. First, from [50, Eq. 49], we have the derivative of $\tilde{n}_k$ with respect to J_k due to the inclusion of $|J_k|$ is

$$\left.\frac{\partial \tilde{n}_k}{\partial J_k}\right|_{|J_k|} = \frac{|J_k| J_k^{-T}}{\sqrt{2\pi|\hat{\Sigma}'_k|}}. \tag{98}$$

We now pause dealing with J_k, and turn our attention to the $|\Sigma'_k|$ term in the denominator of (97). By the same identity from [50, p. 49],

$$\frac{\partial \tilde{n}_k}{\partial \hat{\Sigma}'_k} = \frac{|J_k| \left(\hat{\Sigma}'_k\right)^{-T}}{4\pi\sqrt{|\hat{\Sigma}'_k|}}. \tag{99}$$

Continuing,

$$\frac{\partial \tilde{n}_k}{\partial \Sigma'_k} = \begin{bmatrix} \frac{\partial \tilde{n}_k}{\partial \hat{\Sigma}'_k} & 0 \\ 0 & 0 \end{bmatrix} \tag{100}$$

$$\left.\frac{\partial \mathcal{L}}{\partial \Sigma'_k}\right|_{\tilde{n}_k} = \frac{\partial \mathcal{L}}{\partial \tilde{n}_k} \cdot \frac{\partial \tilde{n}_k}{\partial \Sigma'_k} \tag{101}$$

where the second term denotes the contribution to the partial derivative by $\tilde{n}_k$.

b) Derivatives of $\hat{S}'_k$: Given the derivative $\frac{\partial \mathcal{L}}{\partial \hat{S}'_k}$ computed in (96), we may repeatedly apply [50, Eq. 60] to find the partial derivatives with respect to $\hat{\Sigma}'_k = \left(\hat{S}'_k\right)^{-1}$. The results are omitted for brevity.

From this, it is trivial to apply the chain rule element-wise to obtain the partial derivatives of $\mathcal{L}$ with respect to $\hat{\Sigma}'_k$ due to $\hat{S}'_k$, which we denote

$$\left.\frac{\partial \mathcal{L}}{\partial \hat{\Sigma}'_k}\right|_{\hat{S}'_k}$$

Continuing, we observe

$$\left.\frac{\partial \mathcal{L}}{\partial \Sigma'_k}\right|_{\hat{S}'_k} = \begin{bmatrix} \left.\frac{\partial \mathcal{L}}{\partial \tilde{\Sigma}'_k}\right|_{\hat{S}'_k} & 0 \\ 0 & 0 \end{bmatrix}. \tag{102}$$

Finally, we arrive at the derivative of the loss $\mathcal{L}$ with respect to the transformed covariance Σ'_k

$$\frac{\partial \mathcal{L}}{\partial \Sigma'_k} = \left.\frac{\partial \mathcal{L}}{\partial \Sigma'_k}\right|_{\hat{S}'_k} + \left.\frac{\partial \mathcal{L}}{\partial \Sigma'_k}\right|_{\tilde{n}_k} \tag{103}$$

We conclude the derivatives with respect to Σ_k by transforming this back to the source coordinate system by

$$\frac{\partial \mathcal{L}}{\partial \Sigma_k} = R_{CW}^\top J_k^\top \frac{\partial \mathcal{L}}{\partial \Sigma'_k} J_k R_{CW}. \tag{104}$$

Now, we continue to backpropagate the gradients with respect to J_k. Denote $T_k = J_k R_{CW}$. Then

$$\frac{\partial \mathcal{L}}{\partial T_k} = \left(\frac{\partial \mathcal{L}}{\partial \Sigma'_k}^\top + \frac{\partial \mathcal{L}}{\partial \Sigma'_k} \right) T_k \Sigma_k \tag{105}$$

$$\frac{\partial T_k}{\partial J_k} = R_{CW}. \tag{106}$$

This brings us to

$$\frac{\partial \mathcal{L}}{\partial J_k} = \frac{\partial \mathcal{L}}{\partial \tilde{n}_k} \left.\frac{\partial \tilde{n}_k}{\partial J_k}\right|_{|J_k|} + \frac{\partial \mathcal{L}}{\partial T_k} \frac{\partial T_k}{\partial J_k} \tag{107}$$

where the first term is given by (90) and (98), and the second is given by (105) and (106). J_k depends on μ_k^t, so we continue to backpropagate.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mu_k^t} = & \left[\sum_{i=1}^3 \sum_{j=1}^3 \frac{\partial \mathcal{L}}{\partial J_{k,(i,j)}} \frac{\partial J_{k,(i,j)}}{\partial \mu_{k,n}^t} \right]_{n=\{1,2,3\}}^\top \\ & + \left[0 \quad 0 \quad \left.\frac{\partial \mathcal{L}}{\partial \mu_{k,z}^t}\right|_{\mathcal{D}_k} \right]^\top, \end{aligned} \tag{108}$$

where $J_{k,(i,j)}$ denotes the (i,j) entry of J_k. In practice, each term in (108) was computed symbolically, and the result is omitted here for brevity. Finally, we arrive at the gradient of $\mathcal{L}$ with respect to μ_k due to the contribution from J_k as

$$\left.\frac{\partial \mathcal{L}}{\partial \mu_k}\right|_{J_k} = R_{CW}^\top \frac{\partial \mathcal{L}}{\partial \mu_k^t} \tag{109}$$

c) Gradient of of $\hat{\mu}'_k$: The final term to differentiate is $\hat{\mu}'_k$. Immediately, we find that

$$\frac{\partial \hat{\mu}'_k}{\partial \mu_k} = \frac{\partial \hat{\mu}'_k}{\partial \mu'_k} \cdot \frac{\partial \mu'_k}{\partial \mu_k^t} \cdot \frac{\partial \mu_k^t}{\partial \mu_k} \tag{110}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot J_k \cdot R_{CW}. \tag{111}$$

This may be used to compute

$$\left.\frac{\partial \mathcal{L}}{\partial \mu_k}\right|_{\hat{\mu}'_k} = \frac{\partial \mathcal{L}}{\partial \hat{\mu}'_k} \cdot \frac{\partial \hat{\mu}'_k}{\partial \mu_k}. \tag{112}$$

This gives the final gradient,

$$\frac{\partial \mathcal{L}}{\partial \mu_k} = \left.\frac{\partial \mathcal{L}}{\partial \mu_k}\right|_{\hat{\mu}'_k} + \left.\frac{\partial \mathcal{L}}{\partial \mu_k}\right|_{J_k} \tag{113}$$

The above gradients were incorporated into a PyTorch automatic differentiation framework, based largely on the reference implementation provided by [2].

Appendix F
Implementation Details

The planning comparison experiments involving SPLANNING, SPARROWS, and ARMTD were conducted on a system with an AMD Ryzen 5950X @ 3.4GHz and dual NVIDIA RTX A6000 GPUs. The experiments involving cuRobo and MPOT were conducted on a system with an Intel Core i7-8700K CPU @ 3.70GHz and dual NVIDIA RTX A6000 GPUs. Baseline experiments for CHOMP and TrajOpt were performed on a machine with an Intel Core i9-12900H CPU @ 4.90GHz.

Appendix G
Additional Experiments

This section assesses the effectiveness of SPLANNING's by evaluating Normalized 3DGS reconstructions.

A. Normalized 3DGS Evaluation

We first evaluate the ability of the normalized 3DGS to reconstruct scenes compared to the original 3DGS formulation. We evaluate reconstruction quality on two common RGB-D datasets. First, the method was evaluated on Replica [51], which is a simulated dataset. Replica contains sequences of simulated images at 20Hz; these were decimated to 2Hz after which 1 in every 8 retained images was excluded from training for evaluation. Next, Normalized 3DGS was evaluated on TUM-RGBD [52], which is a real-world dataset. TUM-RGBD is a relatively difficult dataset for accurate 3D reconstruction due to low-resolution images and noisy depth data. TUM-RGBD sequences were downsampled to achieve an approximate total of 250 images per sequence. Again, 1 in every 8 images was excluded from training for evaluation. The reported metrics were only computed on the images excluded from training.

Visual reconstruction results, measured by SSIM and PSNR [53], are shown in Table IV. Experiments indicate that Normalized 3DGS approaches the visual reconstruction quality of standard 3DGS. This is a positive result, since the normalization adds significant complexity to the training process and is the only method allowing risk-aware planning.

Additionally, the geometric accuracy of each method is evaluated by measuring the difference between the rendered depth and ground-truth depth. For each image excluded from training, a depth image is rendered and compared to the ground truth. The Root-Mean-Square (RMS) error is calculated for each image and then averaged over all images to find the final metric. Similar to SSIM and PSNR results, Normalized 3DGS approaches the accuracy of standard 3DGS, as shown in Table V.

B. Collision Probability Evaluation

We first evaluate the performance of our constraint representation and compare it against existing methods that plan in radiance fields. In particular, we compare against CATNIPS [11], which represents the world using a Neural Radiance Field, and Splat-Nav [12], which represents the world

TABLE IV: Reconstruction evaluation on Replica and TUM-RGBD. Each method is evaluated at 7,000 and 30,000 iterations. Standard 3DGS slightly outperforms the normalized variant. However, the performance is comparable, and only Normalized 3DGS is suitable for motion planning. The **best** and second best result for each metric is annotated.

Dataset	Scene	Unnormalized, 7k		Unnormalized, 30k.		Normalized, 7k.		Normalized, 30k	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Replica	office0	40.56	0.976	**44.06**	**0.986**	40.61	0.977	42.14	0.979
	office1	40.44	0.969	**42.52**	**0.974**	40.22	0.971	40.39	0.964
	office2	32.80	0.934	**37.65**	**0.972**	32.64	0.934	35.80	0.963
	office3	33.92	0.941	**37.54**	**0.969**	33.84	0.943	36.68	0.961
	office4	35.33	0.950	**39.18**	**0.971**	35.15	0.951	38.53	0.968
	room0	33.99	0.946	**37.90**	**0.971**	34.20	0.949	36.33	0.961
	room1	34.57	0.947	**39.37**	**0.973**	34.00	0.946	37.57	0.967
	room2	36.02	0.956	**40.30**	**0.975**	35.93	0.959	39.01	0.970
TUM-RGBD	fr1/360	17.93	0.700	**20.14**	**0.717**	17.64	0.678	18.61	0.667
	fr1/desk	20.14	0.725	**21.29**	**0.758**	19.90	0.720	21.01	0.751
	fr1/desk2	19.56	0.720	**20.73**	**0.747**	19.31	0.717	20.44	0.744
	fr1/floor	22.05	0.662	**24.00**	**0.703**	21.84	0.660	23.65	0.697
	fr1/plant	18.73	0.655	**19.70**	**0.671**	18.41	0.647	19.19	0.661
	fr1/room	18.25	0.659	**19.46**	**0.689**	18.04	0.658	19.18	0.690
	fr1/teddy	18.32	0.639	**19.39**	**0.652**	18.10	0.636	18.88	0.648
	fr2/360_hemi	19.09	0.678	**20.16**	**0.686**	18.34	0.657	19.02	0.664
	fr2/coke	19.52	0.760	**21.19**	**0.774**	19.07	0.754	20.07	0.755
	fr2/dishes	22.68	0.809	**23.69**	**0.819**	22.23	0.801	23.24	0.811
	fr2/flowerbouquet	20.83	0.748	**22.17**	**0.762**	20.41	0.740	21.38	0.747

TABLE V: The RMSE depth error (m) is computed per evaluation image, then averaged over each image in the dataset. The standard 3DGS slightly outperforms the Normalized variant, but only Normalized 3DGS is suitable for risk-aware planning. Each method is evaluated at 7,000 and 30,000 iterations. The **best** and second best result is annotated.

	Scene	UnNorm 7k Its	UnNorm 30k Its	Norm 7k Its	Norm 30k Its
Replica	office0	0.022	**0.017**	0.028	0.034
	office1	0.015	**0.012**	0.020	0.034
	office2	0.028	**0.017**	0.046	0.029
	office3	0.032	**0.022**	0.051	0.035
	office4	0.038	**0.025**	0.062	0.038
	room0	0.030	**0.021**	0.040	0.044
	room1	0.022	**0.014**	0.032	0.031
	room2	0.036	**0.024**	0.045	0.034
TUM-RGBD	fr1/360	0.183	**0.174**	0.240	0.231
	fr1/desk	0.102	**0.101**	0.126	0.107
	fr1/desk2	0.136	**0.134**	0.162	0.140
	fr1/floor	0.050	**0.047**	0.057	0.052
	fr1/plant	0.276	**0.277**	0.313	0.296
	fr1/room	0.190	**0.174**	0.223	0.191
	fr1/teddy	0.268	**0.269**	0.292	0.272
	fr2/360_hemi	0.309	**0.350**	0.471	0.424
	fr2/coke	0.364	**0.359**	0.388	0.377
	fr2/dishes	0.178	**0.173**	0.230	0.202
	fr2/flowerbouquet	0.387	**0.369**	0.433	0.402

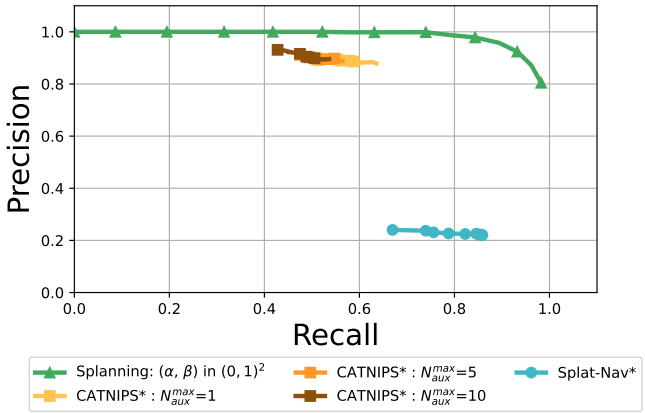

Fig. 5: Treating the collision constraints from SPLANNING, CATNIPS*, Splat-Nav* as a classifiers, where a positive indicates collision and negative indicates no collision, allows evaluating the respective precision-recall curves.

using an un-normalized 3D Gaussian Splat. Each method's collision representation is shown in Fig. 3. The NeRF representation for CATNIPS was trained using Instant-NGP [48], while the Splat-Nav scene representations were built using the original Gaussian Splatting method [2]. We treat each collision detection method as a classifier to determine whether a configuration will result in a collision. The performance of each method is measured by evaluating the precision and recall of the collision classifier [54]. A True Positive indicates the correct identification of a collision, while a True Negative indicates the correct identification of free space.

In each scene, the arm configurations are classified into three categories: Unsafe (in collision), Nearly Safe ($<$ 40cm to the nearest obstacle), and Safe ($>$ 40cm to the nearest obstacle). We randomly sample 30 configurations of the arm, which includes 10 configurations from each category to ensure an informative collection of samples. Since the CATNIPS and SPLANNING methods are described for spherical robots, we conduct our experiments by evaluating the collision over the individual spheres that comprise the robot's spherical forward occupancy. Further, to simplify the comparison with CATNIPS, we set each sphere to have the radius of the largest sphere in the SFO. Finally, CATNIPS and Splat-Nav are evaluated using our re-implementations of their respective constraints; as a result, we refer to their respective results as CATNIPS* and Splat-Nav*. Descriptions of the re-implementations are detailed in Appendix D.

We calculate the Precision-Recall values at various thresholds corresponding to the allowable collision risk. SPLANNING, CATNIPS, and Splat-Nav each express this maximum allowable risk differently. For SPLANNING, we vary the parameters α and β, in the range of (0,1), as discussed in Section V-B. For CATNIPS, we vary the collision probability σ and the maximum number of auxiliary particles

that may exist in the robot's volume, denoted N_{aux}^{max}. We refer the interested reader to [11] for a detailed explanation of these parameters. In our experiments, N_{aux}^{max} is set to $\{1, 5, 10\}$ while using V_{aux} as 10^{-6}, and σ is varied in the range of (0,1). Finally, Splat-Nav deterministically checks whether the robot collides with the 99% confidence ellipsoids, meaning the method does not emit a risk parameter. Instead, we evaluate the method across several level sets of the Gaussian functions, ranging from the 1-sigma ellipsoid to the 10-sigma ellipsoid.

We report Precision-Recall curves in Fig. 5. CATNIPS* achieves a high precision, indicating that most of the detected collisions were true collisions. However, the comparatively low recall indicates that unsafe configurations were classified as safe. Splat-Nav* achieves higher recall, indicating that the method is more conservative and will result in fewer collisions. However, the lower precision indicates the presence of false positives, which would cause a motion planner to behave overly conservatively. The SPLANNING constraint achieves higher precision and recall than either Splat-Nav* or CATNIPS*. Further, Fig. 5 shows that the thresholds α and β in SPLANNING allow for significant control of the behavior of the constraint, ranging from highly aggressive near the left to more conservative on the right.