related works

Gaussian-lic

key steps:

1. initialization of gaussian map
   The photo-realistic Gaussian map is bootstrapped with all 3D points of the first received frame.
2. expansion the gaussian map
   Every frame after the first usually captures the geometry and appearance of the newly observed areas. However, LiDAR points from different frames may contain duplicate information. We take every fifth frame as a keyframe. When a keyframe is received, we first merge all points received in these five frames into a point cloud. To avoid redundancy, we then render O from the current keyframe image view according to α-blending.
3. optimize the gaussian map
   Once the current received frame is a keyframe, we randomly select K keyframes from all keyframes to optimize the Gaussian map, avoiding the catastrophic forgetting problems and maintaining the geometric consistency of the global map. We randomly shuffle the selected K keyframes and iterate through each of them to optimize the map by minimizing re-rendering loss:

$$L = (1-\lambda)\left\|E[C]-\bar{C}\right\|_1 + \lambda L_{D-SSIM}$$

CUDA multi thread backpropagation

1. perform per-Gaussian back-propagation and assign CUDA threads for all Gaussians in a tile, increasing the parallelism while decreasing collisions.
2. For faster training of the Gaussianmap representation, sparse Adamwas applied to only update Gaussians in the current camera frustum.
3. remove slim gaussian balls

GS-LIVO

1. The lio will do the IESKF to get the state estimation
2. the gaussian initialize the gaussian balls according to the current image and lidar point cloud at t he estimated state
3. the gaussian balls will be put into the global gaussian map (octree similar with the voxel map) to filter out the useless and invalid points
4. the filtered gaussian balls in current frame will be put into the sliding window and be optimized
5. the lio state estimation and the optimized gaussian balls and image in current frame will be put into the vio part to do the state estimation

3DGS-LM -> only to improve the 3d gaussian, not the slam

We utilize our LM implementation in the second stage of3DGS optimization (see Fig. 2). Before that, we use the ADAM optimizer to obtain an initialization of the Gaussian parameters. It is also possible to use the LM optimizer from the beginning, however this does not bring any additional speed-up (see Fig. 4). In the beginning of optimization, gradient descent makes rapid progress by optimizing the Gaussians from a single image per iteration. In contrast, we sample many images in every LM iteration, which makes every iteration more time-consuming. This additional compute overhead is especially helpful to converge to optimal Gaussian parameters quicker (see Fig. 1 left). Splitting the method in two stages also allows us to complete the densification of the Gaussians before employing the LM optimizer, which simplifies the implementation.

DashGaussian

这篇文章《DashGaussian: Optimizing 3D Gaussian Splatting in 200 Seconds》提出了一种名为DashGaussian的方法，旨在加速3D高斯泼溅（3D Gaussian Splatting, 3DGS）的优化过程，同时保持渲染质量。以下是其核心原理的详细解释：

---

1. 问题背景

3D高斯泼溅（3DGS）是一种用于新视角合成的技术，通过优化一组3D高斯基元来表示场景。然而，3DGS的优化过程通常需要较高的计算成本，尤其是在高分辨率渲染和高斯基元数量较多时。现有的加速方法（如剪枝冗余基元或优化渲染管线）往往以牺牲渲

染质量为代价。

2. 核心思想

DashGaussian的核心思想是通过动态调度优化复杂度（包括渲染分辨率和高斯基元数量）来合理分配计算资源，从而在不降低渲染质量的前提下加速优化。具体来说：

- **动态渲染分辨率**：在优化初期使用低分辨率图像，逐渐过渡到高分辨率，以减少计算量。
- **基元数量调度**：根据渲染分辨率动态调整高斯基元的数量，避免过度密集化。

---

3. 技术原理

3.1 优化复杂度的定义

优化复杂度由两个主要因素决定：

1. **渲染分辨率**：分辨率越高，计算量越大。
2. **高斯基元数量**：基元越多，计算负担越重。
   DashGaussian通过动态调整这两个因素来降低整体优化复杂度。

3.2 频率引导的分辨率调度

- **频率理论**：图像的高频分量对应细节信息，低频分量对应整体结构。低分辨率图像缺少高频分量，因此优化初期可以用低分辨率图像拟合低频信息，后期再用高分辨率图像补充高频细节。
- **分辨率调度**：
  - 通过计算训练视图的频率能量（公式2），动态决定何时从低分辨率切换到高分辨率。
  - 使用分数函数（公式3）分配优化步骤，确保高分辨率阶段仅在必要时使用。

3.3 分辨率引导的基元调度

- **基元数量与分辨率的关系**：高分辨率需要更多基元来描述细节，而低分辨率时基元过多会导致计算冗余。
- **基元增长曲线**：
  - 基元数量随优化步骤动态调整（公式4），初期增长缓慢，中期加速，后期趋于稳定。
  - 采用凹向上的增长曲线，减少早期计算量。
- **动量基元预算**：
  - 动态估计最终基元数量（公式5），避免依赖经验值。
  - 通过动量更新机制（类似物理中的动量）自适应调整基元数量。

3.4 抗锯齿与优化质量

- 低分辨率训练时，采用抗锯齿下采样避免误导优化。
- 通过渐进式增加分辨率，逐步拟合高频分量，避免3D走样问题。

---

4. 实现细节

1. **分辨率调制**：将连续的分辨率因子离散化，以简化计算。
2. **学习率调整**：在低分辨率阶段保持较高的学习率，高分辨率阶段逐步衰减。
3. **基元剪枝与分裂**：根据基元的梯度或渲染误差动态剪枝或分裂基元，控制基元数量。

GauSS-MI

This paper introduces Gaussian Splatting Shannon Mutual Information (GauSS-MI) as a metric for efficient next best view selection in high-visual fidelity active reconstruction.

这篇文章《GauSS-MI: Gaussian Splatting Shannon Mutual Information for Active 3D Reconstruction》提出了一种基于高斯泼溅（3D Gaussian Splatting, 3DGS）和香农互信息（Shannon Mutual Information, SMI）的主动三维重建系统，旨在通过量化视觉不确定性并选择最优视角来提高重建效率和视觉质量。以下是其核心实现原理的详细介绍：

1. 问题背景与挑战

- **主动三维重建**：在未知环境中，机器人需要动态选择下一个最佳视角（Next-Best-View, NBV）以高效完成高质量的三维重建。
- **现有问题**：传统方法主要关注几何完整性（如体素覆盖），而忽略视觉质量的不确定性量化，导致重建结果在纹理和细节上表现不佳。
- **新需求**：需要一种能够实时评估视觉信息增益的指标，并指导机器人选择信息量最大的视角。

2. 核心方法：GauSS-MI

（1）概率模型构建

- **高斯泼溅表示**：场景由多个3D高斯椭球体（Gaussians）表示，每个高斯包含几何（位置、协方差）和光学属性（颜色、透明度）。
- **可靠性概率**：为每个高斯定义一个二元随机变量 r，表示其渲染的可靠性（可靠或不可靠），初始化为无先验信息（$P(r) = 0.5$）。
- **逆传感器模型**：通过当前观测与渲染图像的损失（颜色和深度差异）更新高斯的可靠性概率。损失越小，可靠性越高：

$$P(r^i|Z_k) = \frac{1}{(\lambda_L L_k)^{\lambda_T T^{[i]}} + 1}$$

其中 L_k 是损失图像，$T^{[i]}$ 是高斯的累积透射率（调节更新强度）。

（2）香农互信息（GauSS-MI）

- **目标**：量化从新视角 z_k 中获取的预期信息增益，以最小化地图的不确定性（条件熵）。
- **互信息定义**：

$$I(r; z_k) = H(r) - H(r \mid z_k)$$

最大化互信息等价于最小化条件熵。

- **高效计算**：通过解析推导，将互信息分解为像素级信息增益的累加：

$$I(r; z) = \sum_{j=1}^{n_z} P(z^{[j]}|M^{[j]}) \sum_{i \in N^{[j]}} -T^{[i]} log(P(r^{[i]}))$$

其中 $P(z^{[j]})$ 是传感器测量先验（基于亮度噪声模型），$log(P(r^{[i]}))$ 反映高斯的信息增益。

（3）实时性能优化

- **并行化**：利用CUDA实现高斯投影和互信息计算的并行化，复杂度为 $O(N_pN_gN_c)$，其中 N_p 是像素数，N_g 是高斯数，N_c 是候选视角数。
- **对比优势**：相比Fisher信息矩阵方法（复杂度随观测数量增长），GauSS-MI的计算量稳定，适合实时应用。

3. 系统架构总览

系统分为两大模块：**在线3DGS重建模块**和**主动规划模块**，通过共享内存实现实时数据交互。

1. 在线3DGS重建模块

(1) 实时高斯泼溅地图构建

- **初始化**：首帧RGB-D数据通过SFM（如COLMAP）或SLAM（如Gaussian Splatting SLAM）初始化高斯椭球体，属性包括：
  - 几何参数：均值 $\mu_\omega^{[i]}$（世界坐标系位置）、协方差 $\Sigma_W^{[i]}$（各向异性形状）。
  - 光学参数：球谐系数（颜色 $c^{[i]}$）、透明度 $\alpha^{[i]}$。
- **动态扩展**：
  - 新观测的深度图通过**反投影**生成候选高斯，剔除冗余（基于重叠度阈值）。
  - 使用**梯度下降**在线优化高斯参数（每帧迭代10-20次），损失函数为RGB-D的L1损失 + SSIM：

$$L_{total} = \lambda_{rgb} \parallel C - \hat{C} \parallel + \lambda_{depth} \parallel D - \hat{D} \parallel + \lambda_{ssim} SSIM(C, \hat{C})$$

(2) 概率地图更新（Algorithm 1）

- **输入**：当前帧的RGB-D图像 $\hat{C}_k, \hat{D}_k$ 和相机位姿 σ_k。
- **关键步骤**：
  1. **渲染与损失计算**：
     - 通过3DGS渲染当前视角的预测图像 C, D。

- 计算逐像素损失

$$L_k = \lambda_c \parallel C - \hat{C}_k \parallel + (1-\lambda_c) \parallel D - \hat{D}_k \parallel$$

2. **概率反向投影**：
   - 对每个像素 j，沿射线排序高斯 $N^{[j]}$，计算累积透射率 $T^{[i]}$（公式3）。
   - 更新高斯的log odds（公式10）：

$$l_{1:k}^{[i]} = -\lambda_T T^{[i]} log(\lambda_L L_k^{[j]}) + l_{1:k-1}^{[i]}$$

3. **概率转换**：通过sigmoid函数将log odds转为概率

$$P(r^{[i]}) = \frac{1}{1+e^{-l[i]}}$$

- **优化细节**：
  - 使用CUDA并行化射线追踪和概率更新，每个线程处理一条射线（像素）。
  - 内存管理：采用哈希表（如InstantNGP的网格结构）加速高斯查询。

2. 主动规划模块

(1) 候选视角生成（Viewpoint Primitive Library）

- **动作空间设计**（公式21）：
  - 四旋翼的简化动作参数 α=$[v_{xy}, v_z, \omega_z]$，离散化为：
    $V_{xy} = 0, 0.5, 1.0\, m/s, V_z = -0.2, 0, 0.2\, m/s, \Omega_z = -30^\circ, 0^\circ, 30^\circ$
  - **前向传播**（公式22）：给定当前状态 $\sigma_0 = [x_0, y_0, z_0, \psi_0]^T$，生成终点状态 σ_f（位置和偏航角）。
- **运动基元（Motion Primitive）**：
  - 采用**最小Snap轨迹**（附录B）连接 σ_0 和 σ_f，闭式解为7次多项式：

$$p^*(t) = \sum_{i=0}^{7} a_i t^i$$

- **运动代价** J：计算轨迹的总Snap能量 $J = \int_0^T \| s(t) \|^2 \, dt$，用于权衡信息增益与能耗。

(2) GauSS-MI计算（Algorithm 2）

- **输入**：候选视角 σ 和当前3DGS地图 G。
- **关键步骤**：
  1. **渲染预期图像**：
     - 从 σ 渲染颜色 $C^{[j]}$ 和亮度 $M^{[j]} = 0.299R + 0.587G + 0.114B$
     - 计算每个像素的信息增益 。
  2. **传感器先验建模**：
     - 基于相机噪声模型（Poisson-Gaussian [10]），计算 $P(z^{[j]} \mid M^{[j]})$。
  3. **互信息积分**：

$$I(\sigma) = \sum_{j=1}^{n_z} (z^{[j]} \mid M^{[j]}) \cdot f^{[j]}$$

- **加速策略**：
  - **多分辨率评估**：先低分辨率（1/4图像）粗选视角，再全分辨率精修。
  - **GPU并行**：每个CUDA Block处理一个候选视角。

(3) 最优视角选择

- **综合奖励**（公式23）：

$$R = w_I I(\sigma) - w_J J(\sigma_T)$$

  - 权重调参：w_I 和 w_J 根据场景动态调整（如狭窄环境增大 w_J 避免碰撞）。
- **安全约束**：

- 碰撞检测：通过3DGS地图的占据信息（如Alpha Shape）过滤不可行轨迹。
- 视野约束：确保目标视角在传感器FOV内（如D435的87°水平FOV）。

GS-LIVM

提出了一种基于高斯泼溅（Gaussian Splatting）的实时逼真多传感器（LiDAR-IMU-相机）建图框架，用于大规模无边界户外场景。以下是其核心原理的详细解释：

1. 问题背景与挑战

- **传统SLAM的局限性**：传统SLAM系统（如基于滤波或图优化的方法）主要依赖稀疏特征点，难以实现高质量的3D重建和逼真渲染。
- **神经场景表示的兴起**：NeRF和3D高斯泼溅（3DGS）等技术提升了场景表示的真实性，但在户外场景中面临以下挑战：
  - **数据稀疏性**：LiDAR点云分布不均匀（如多线旋转LiDAR或非重复扫描LiDAR）。
  - **实时性不足**：现有方法（如NeRF-SLAM或离线3DGS）优化时间长，无法满足实时需求。
  - **视角偏差**：传感器单向运动时，3D高斯优化会偏向相机视角，导致新视角渲染质量下降。

2. 核心方法

GS-LIVM通过以下技术解决上述问题：

2.1 多传感器紧耦合前端

- **状态估计**：采用LiDAR-IMU-视觉紧耦合里程计（如R³LIVE或FAST-LIVO）提供高精度的传感器位姿和点云坐标变换。
- **数据同步**：融合LiDAR（几何结构）、IMU（高频运动补偿）和相机（纹理信息）的数据。

2.2 体素级高斯过程回归（Voxel-GPR）

- **问题**：LiDAR点云稀疏且分布不均，直接用于3D高斯初始化会导致内存浪费和优化效率低。
- **解决方案**：
  1. **体素划分**：将空间划分为体素，对每个体素内的点云独立处理。

2. **高斯过程回归（GPR）**：
- 对每个体素内的点云，通过PCA确定主方向（值轴），其他方向为参数轴。
- 使用核函数（如RBF）建模点云分布，预测均匀分布的网格点（`P_α*`）及其协方差（`Σ_α*`）。
- 公式：

$$p(\mathbf{f}_* \mid \mathbf{x}_*, \mathbf{x}, \mathbf{f}) = \mathcal{N}(\mathbf{f}_* \mid \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$$

$$\boldsymbol{\mu}_* = \mathbf{K}_*^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{f}$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{K}_*$$

```
3. **并行加速**：利用CUDA并行处理多个体素，耗时小于30毫秒。
```

2.3 3D高斯的快速初始化

- **传统方法**：3D高斯的尺度和旋转参数需通过邻近点距离或随机初始化，收敛慢。
- **改进方法**：
  1. **基于VoxeI-GPR的初始化**：
  - 对每个体素的子网格（subgrid），用加权最小二乘拟合3D高斯的位置（`p_β`）和协方差矩阵（`Φ_β`）。
  - 公式：（权重`w_i^β = 1/Σ_*`，`Q`为点云相对于中心的偏移）。

$$\mathbf{p}^\beta = \frac{\sum_{i=1}^{n_r^2} (\mathcal{P}_{f*}^\alpha)_i^\beta \cdot w_i^\beta}{\sum_{i=1}^{n_r^2} w_i^\beta}; \Phi^\beta = \frac{\mathbf{Q}^\top \cdot \mathrm{diag}(w_1^\beta, \ldots, w_{n_r^2}^\beta) \cdot \mathbf{Q}}{\sum_{i=1}^{n_r^2} w_i^\beta}$$

```
    - 尺度和旋转直接由协方差矩阵分解得到（`S_β = diag(Φ_β)`，`R_β`初始为单位四元数）。
2. **颜色初始化**：通过相机外参将高斯中心投影到RGB图像，获取初始颜色（SH系数）。
```

2.4 迭代优化框架

- **地图扩展与协方差更新**：

- 体素分为四类（未探索/待处理/活跃/已收敛），仅对活跃体素迭代优化Voxel-GPR参数。
- 利用新观测的LiDAR点云更新协方差，直至收敛（方差小于阈值`η`）。

- **渲染与损失函数**：

1. **渲染模型**：
   - 使用3D高斯泼溅的光栅化渲染颜色（`C`）、深度（`D`）和轮廓（`S`）图像：

$$C = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$$

2. **损失函数**：

- **深度相似性损失（`L_d`）**：约束连续帧间的深度一致性。

$$\mathcal{L}_d = \left\| \mathbf{S}_{\mathcal{F}_{c+1}} \circ D_{\mathcal{F}_{c+1}} - \pi_{\mathcal{F}_{c+1}} \mathbf{T}_{\mathcal{F}_{c+1}} \mathbf{T}_{\mathcal{F}_c}^{-1} \pi_{\mathcal{F}_c}^{-1} (\mathbf{S}_{\mathcal{F}_c} \circ D_{\mathcal{F}_c}) \right\|_1$$

- **结构相似性损失（`L_p`）**：约束LiDAR点云与3D高斯的几何一致性。

$$\mathcal{L}_p = \frac{1}{n} \sum_{j=1}^{n} \min_{k \in \{1,\ldots,m\}} \left(\| (\mathbf{P}_f)_j - \mathbf{p}_k \|_2 - \bar{S}_k \right)$$

- **总损失**：结合光度损失（L1+SSIM）、深度和结构损失：

$$\mathcal{L} = (1 - \lambda_s) \| C - C_{gt} \|_1 + \lambda_s \mathcal{L}_{ssim} + \lambda_d \mathcal{L}_d + \lambda_p \mathcal{L}_p$$

VPGS-SLAM

VPGS-SLAM是一种创新的基于3D高斯泼溅(3D Gaussian Splatting)的大规模视觉SLAM系统，下面我将从技术细节层面深入解析其工作原理。

1. 体素化渐进式3D高斯表示系统

1.1 多分辨率体素-高斯混合数据结构

系统采用了一种分层的体素化表示方法：

- **多级体素划分**：根据相机距离动态调整体素大小，近处使用精细体素(如5cm)，远处使用粗糙体素(如20cm)
- **锚点机制**：每个体素中心初始化一个锚点，包含以下属性：
  - 32维特征向量f_a
  - 3维缩放因子l
  - k×3维偏移矩阵O (默认k=10)

$$V_i \in \mathbb{R}^{N\times 3} \quad \text{(体素中心坐标)}; x_i^a \in \mathbb{R}^3 \quad \text{(锚点位置)}; A_i = \{f_i^a \in \mathbb{R}^{32}, l_i \in \mathbb{R}^3, O_i \in \mathbb{R}^{k\times 3}\} \quad \text{(锚点属性)}$$

1.2 3D高斯生成过程

每个锚点生成k个3D高斯椭球体：

1. 位置计算：

$$\mu_i^m = x_i^a + O_i^m \cdot l_i \quad \text{for } m = 0, \ldots, k-1$$

2. 属性解码(通过2层MLP)：

$$\{f_i^a, \delta_i, d_i\} \rightarrow \{\alpha_i^m, q_i^m, s_i^m, c_i^m\}_{m=0}^{k-1}$$

   其中$\delta_i = ||x_i^a - x_i^c||_2$是锚点到相机的距离，d_i是方向向量

1.3 动态生长与修剪策略

生长条件：

- 当进入新区域时，在α累计值低于阈值或深度差异大的区域采样新锚点
- 当体素内高斯梯度∇g > τ_g时，在该体素生长新锚点

修剪条件：

- 定期检查锚点关联高斯的累计不透明度
- 删除长期不活跃(低α)的锚点及其高斯

2. 多子图管理与优化

2.1 子图初始化条件

采用双重阈值触发新子图创建：

- 距离阈值：室内d=0.5m，室外d=10m
- 旋转阈值：ω=50度
- 数学表达：

$$\text{new\_submap} = \begin{cases} 1, & \text{if } \Delta t > d \text{ or } \Delta R > \omega \\ 0, & \text{otherwise} \end{cases}$$

2.2 子图优化目标函数

包含四项损失：

$$L_m = L_c + L_{SSIM} + \lambda_d L_d + \lambda_{vol} L_{vol}$$

其中：

- 颜色损失：$L_c = ||\hat{I} - I||_1$
- SSIM损失：

$$L_{SSIM} = (1 - \lambda_{SSIM}) \cdot |\hat{I} - I|_1 + \lambda_{SSIM}(1 - SSIM(\hat{I}, I))$$

- 深度损失：$L_d = |\hat{D} - D|_1$
- 体积正则项：(Prod(·)计算向量各元素乘积，促进小高斯)

$$L_{vol} = \sum_{i=1}^{N_{ng}} Prod(s_i)$$

3. 2D-3D融合相机跟踪

3.1 两阶段优化流程

粗优化阶段：

- 仅使用光度误差最小化：

$$\{R,t\}_{coarse} = argmin_{R,t}(L_c + L_d)$$

- 自适应策略：当渲染质量指标Q_{25} < ζ时跳过此阶段
  精优化阶段：

1. 点云变换：

$$S = \{s_i = \{R_{i-1}, t_{i-1}\}\{R_{pred}, t_{pred}\}p | p \in P\}$$

2. 鲁棒ICP优化：

$$\Delta\{R,t\}_{est} = argmin_{R,t} \sum_{(s,q)\in C(\tau_t)} \rho(||\{R,t\}s - m||_2)$$

   使用Geman-McClure鲁棒核：

$$\rho(e) = \frac{e^2/2}{\sigma_t/3 + e^2}$$

3.2 哈希加速结构

使用空间哈希表实现高效最近邻搜索：

- 体素大小与子图分辨率一致

- 采用双重下采样策略保留原始点坐标
- 查询复杂度：平均O(1)

4. 闭环检测与全局优化

4.1 BEV闭环检测

1. 描述子提取：
   - 使用轻量级BEVPlace++网络
   - 生成256维全局描述向量
2. 相似度计算：当sim > 0.85且几何验证通过时触发闭环

$$sim(\alpha, \beta) = \frac{v_\alpha \cdot v_\beta}{||v_\alpha|| \cdot ||v_\beta||}$$

4.2 位姿图优化

构建包含两种约束的因子图：

1. 里程计约束：相邻帧间的相对位姿测量
2. 闭环约束：

$$E_{loop} = \lambda_{icp} E_{icp} + \lambda_{render} E_{render}$$

   其中E_{icp}来自体素ICP，E_{render}来自渲染光度误差

4.3 在线子图蒸馏

闭环触发时的融合流程：

1. 对齐子图坐标系
2. 重叠区域高斯合并：

$$G_{merged} = \frac{\sum w_i G_i}{\sum w_i}, \quad w_i = \alpha_i \cdot V_i$$

3. 非重叠区域保留原高斯
4. 联合优化更新全局地图