浙江大学

ZHEJIANG UNIVERSITY

浙江大学软件学院优秀大学生夏令营

调研报告

姓　　名 张宏远

选　　题 任务 1+子任务 2

目录

一、子任务 2 3
2.1 3D Gaussian 原理学习 3
2.1.1 基础 3D Gaussian 概念 3
2.1.2 维度从 3D 转换到 2D 6
2.1.3 颜色在 3D Gaussian 中的处理 8
2.1.4 论文中 3D Gaussian 实现步骤 9
2.1.4.1 表示步骤 9
2.1.4.2 渲染步骤 10
2.1.4.3 优化步骤 11
2.1.5 论文中 3D Gaussian 存在的问题 11
2.2 Street Gaussians 论文详读 12
2.2.1 Street Gaussians 关键技术 12
2.2.1.1 背景表示 12
2.2.1.2 物体表示 13
2.2.1.3 组合渲染 15
2.2.1.4 训练步骤 15
2.2.1.5 损失函数 16
2.2.2 当前存在的问题 17
2.2.3 潜在的改进方向 17
二、附录 20
三、引用 22

一、子任务 2

任务要求：当前，基于神经渲染的街景三维重建算法主要聚焦于两种主要方法：基于 NeRF 的算法和基于 3D Gaussian 的算法。请选择其中一种方式，深入了解 NeRF 或 3D Gaussian 的原理，并进一步选取一篇基于该方式的街景三维重建工作的论文，并详细阐述其关键技术、当前存在的问题以及潜在的改进方向。论文建议来自《中国计算机学会推荐国际学术会议和期刊目录》A 类论文或《清华大学新版计算机学科推荐学术会议和期刊列表》A 类论文。

2.1 3D Gaussian 原理学习

经过多方调研，我发现街景三维重建这种大场景比较适合 3D Gaussion 来实现，虽然 NeRF 使用在大场景领域的应用也有很多，但是 NeRF 强在新视图合成，可以渲染出未知视角，导出三维模型只是副产品而不是 nerf 的主要任务，所以在算法选择方面我选择了 3D Gaussian 的算法进行学习。学习的过程中我参考了 B 站视频，以及 *3D Gaussian Splatting for Real-Time Radiance Field Rendering* [1] 这篇论文，论文的链接为(https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/)。

2.1.1 基础 3D Gaussian 概念

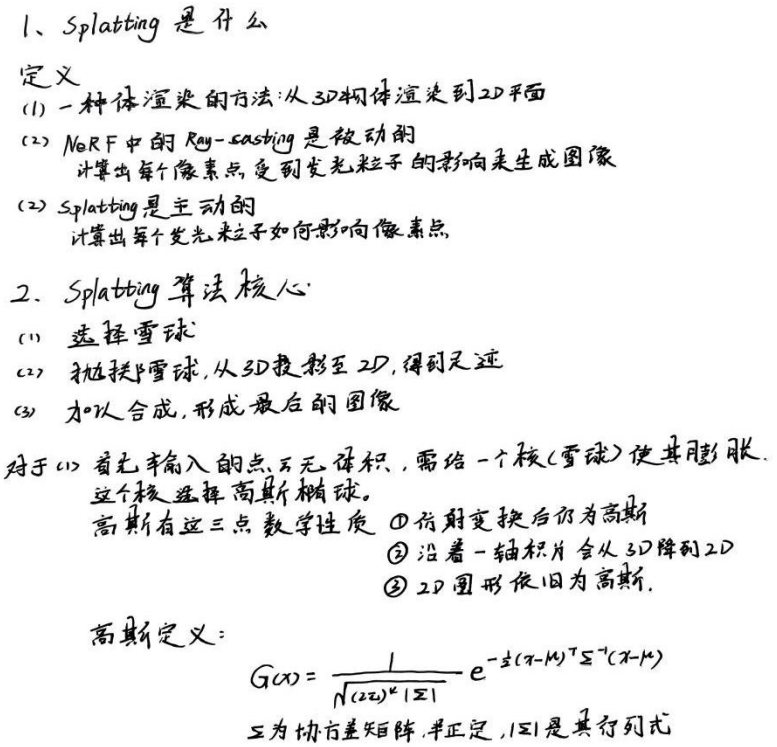

图 1 笔记一

在论文中给出的 3D Gaussian 的形状在 3D 视角是椭球，投射到 2D 视角是椭圆，如图 2 所示，在对高斯分布进行梳理推导后证明了其分布确为椭球推导证明过程在笔记二图 3 中给出。

3D Gaussians

图 2 论文中的 3D Gaussian 形状

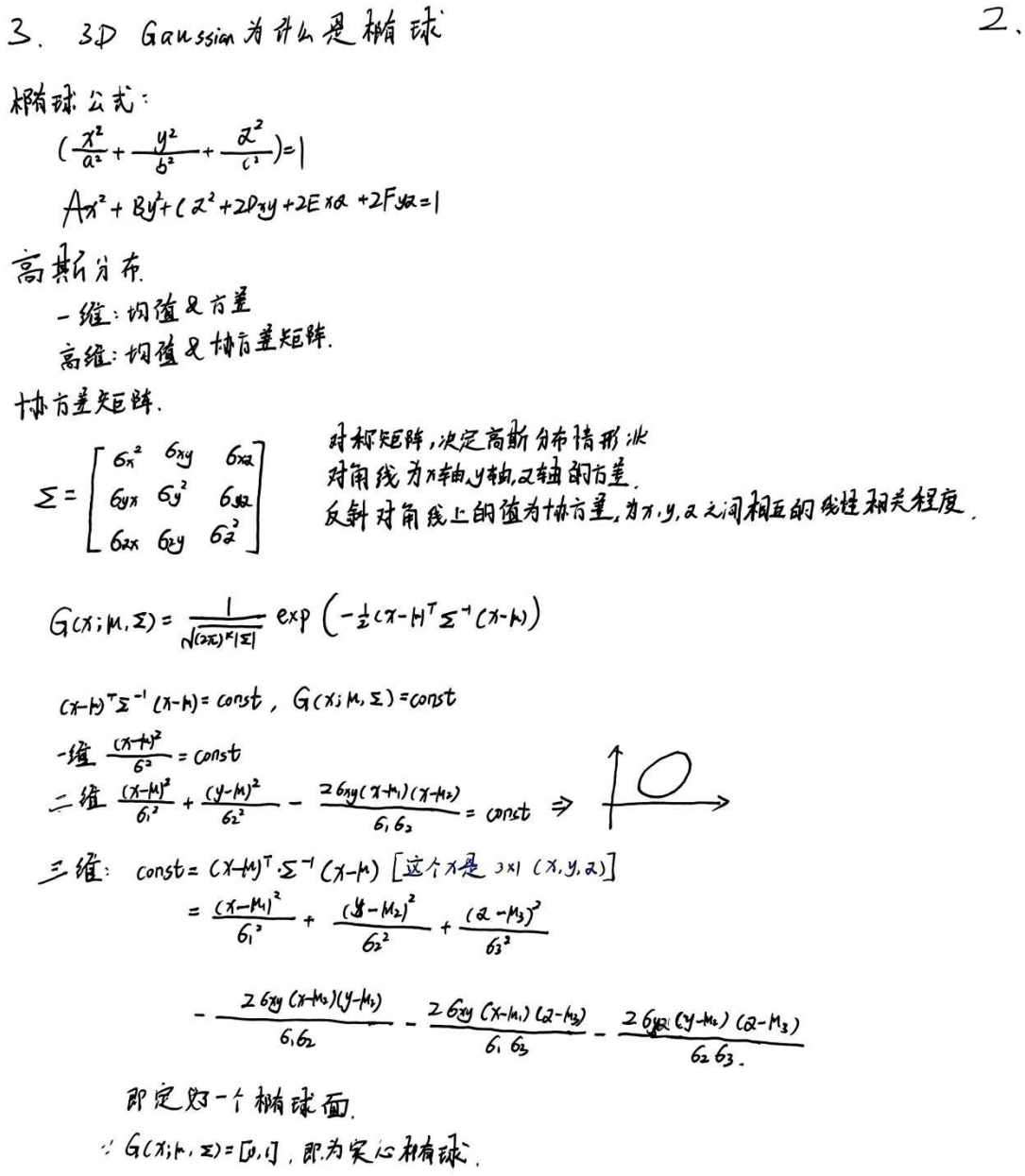

图 3 笔记二

图 5 笔记三中对各向异性和各项同行的概念进行了解释，并推导出了论文中较为核心的一条公式，如图 4 所示，具体的计算代码在附录的代码 4 中给出。

Given a scaling matrix S and rotation matrix R, we can find the corresponding Σ:

$$\Sigma = RSS^T R^T \tag{6}$$

图 4 3D Gaussian 协方差矩阵的分解

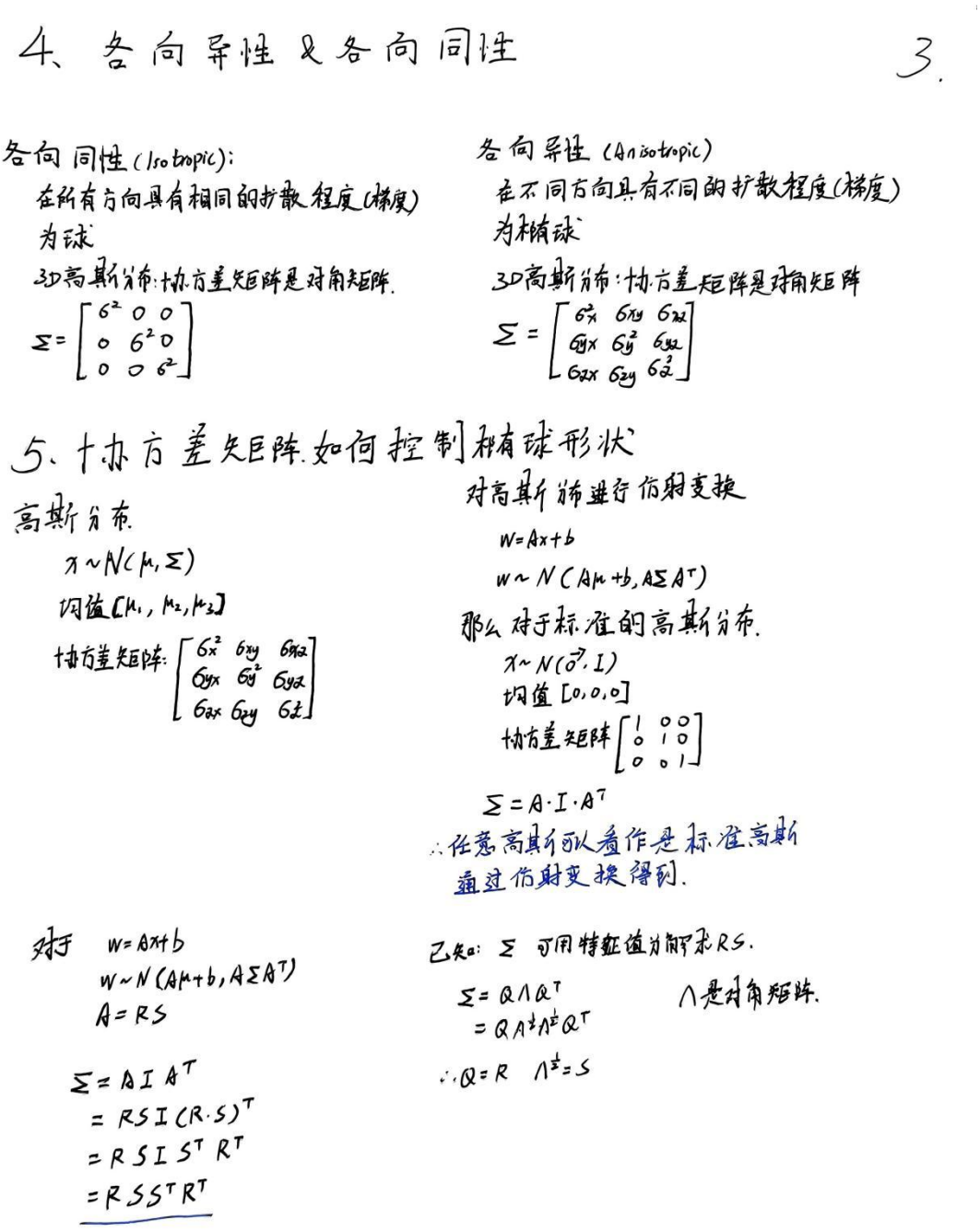

图 5 笔记三

2.1.2 维度从 3D 转换到 2D

在计算机图形学中，变化的关键概念包括观测变换、投影变换、视口变换和光栅化。学习笔记如图 6 所示。观测变换将场景中的对象从世界坐标系转换到摄像机坐标系；投影变换将三维坐标转换为二维坐标，并且不论原始形状的长宽比如何，都会被压缩成一个正方体，直到视口变换才将比例恢复；视口变换则将标准化的设备坐标转换为屏幕坐标，适应屏幕的分辨率和比例；光栅化则是将物体绘制到屏幕上的过程，通过采样将连续的几何形状转变为离散的像素点。例如，mesh 中的一个面片是连续的，但屏幕是离散的，为了在屏幕上展示它，需要对其进行离散化处理。

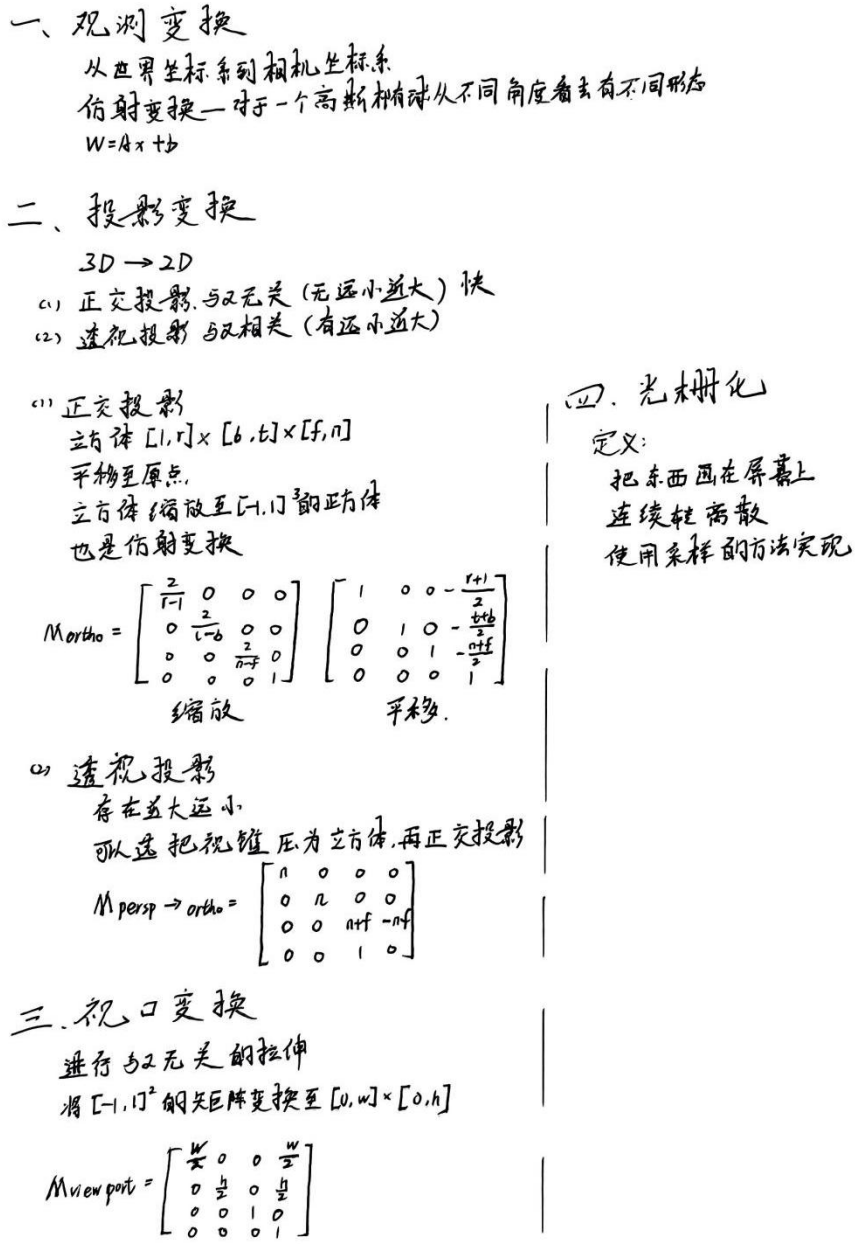

图 6 笔记四

对于光栅化的实例代码在附录的代码 5 中给出。演示如图 7 所示，其是将一个四边形渲染到一个二维图像上，进行光栅化操作。

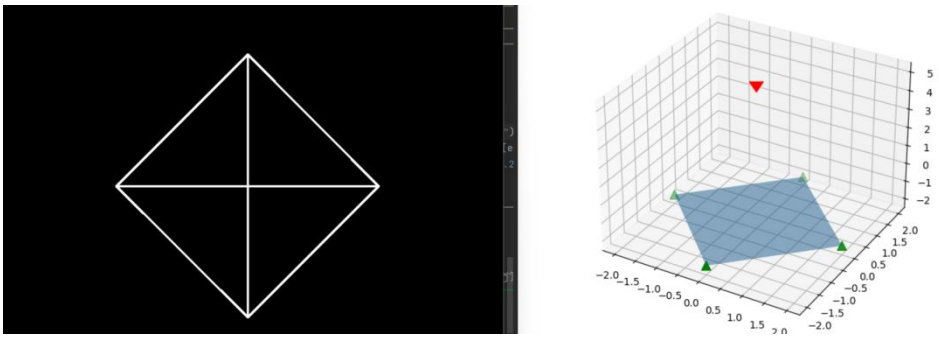

图 7 光栅化渲染四边形

介绍完计算机图形学中的概念，我这边还需要回到 3D Gaussian 中理解其是如何变换，不同点在于变换的过程中引入了雅可比矩阵的概念，对协方差进行处理，其他概念都在图 8 中给出。

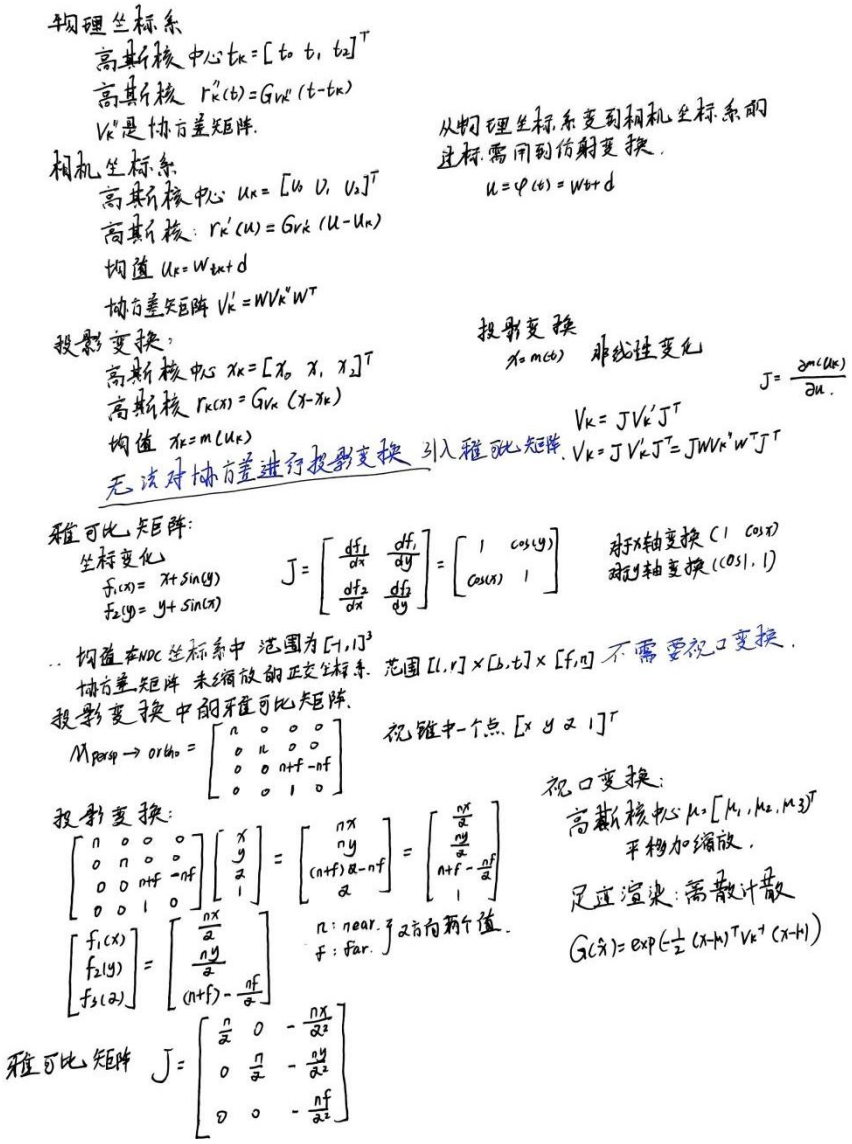

图 8 笔记五

2.1.3 颜色在 3D Gaussian 中的处理

上述内容解决了创造“雪球”，和抛“雪球”的过程，但是在这个过程中“雪球”始终只有白色这一种颜色，而想要准确的表达物品的特征需要很多颜色信息的介入，图 10 就是与色彩相关的内容，其引入了基函数和球谐函数的概念，3D Gaussian 在色彩合成的方法上与 NeRF 类似，在论文中公式如图 9 所示。但是由于 splatting 没有找粒子的过程所以整体速度要较快于 NeRF。

A typical neural point-based approach (e.g., [Kopanas et al. 2022, 2021]) computes the color C of a pixel by blending $\mathcal{N}$ ordered points overlapping the pixel:

$$C = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \tag{3}$$

图 9 论文中计算颜色公式

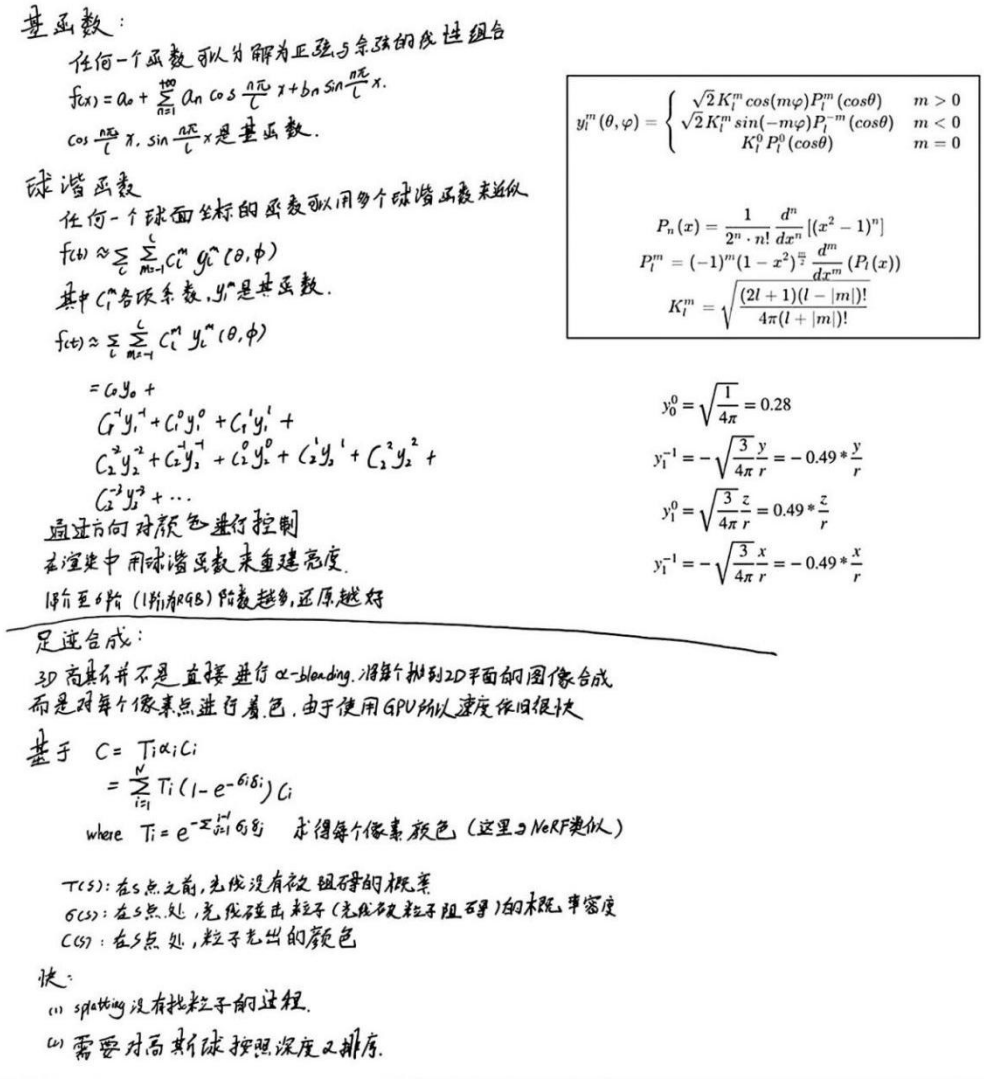

图 10 笔记六

2.1.4 论文中 3D Gaussian 实现步骤

上述学习内容中，我已对 3D Gaussian 技术要点进行了详细总结。接下来，我将对 3D Gaussian Splatting 这篇论文进行阐述。这篇论文主要分为三个部分，如图 11 所示。接下来，我将对每个部分进行详细介绍。

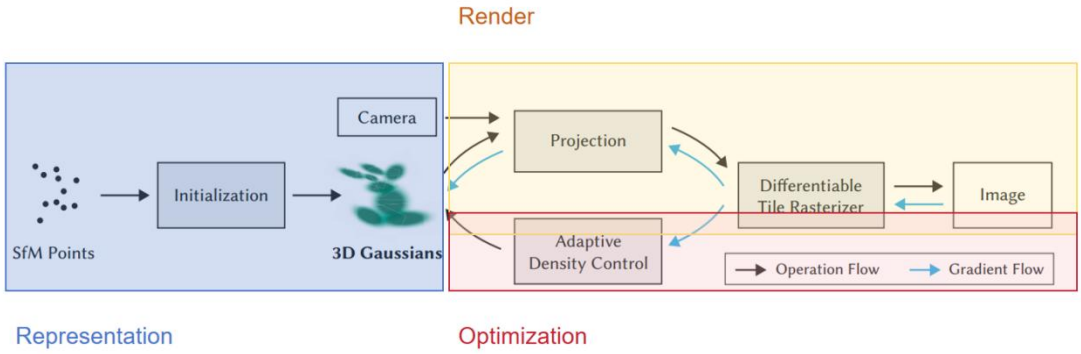

图 11 3D Gaussian Splatting 三个步骤

2.1.4.1 表示步骤

3D Gaussian Splatting 中将几何形状建模为一组不需要法线的三维高斯分布。其高斯分布由定义在世界坐标系中的完整三维协方差矩阵 Σ 以及中心点（均值）定义，具体公式如公式 1 所示。

$$G(x) = e^{-\frac{1}{2}(x)^T \Sigma^{-1}(x)}$$

公式 1 Gaussians 公式

这里提出的 Gaussian 是各向异性的，其各个维度的分量是不同的，其参数有中心点坐标、协方差矩阵、不透明度以及球谐函数，具体维度如图 12 所示。

Gaussian representation parameters:

- point (mean) μ, [3×1]
- covariance matrix Σ, [3×3]
- opacity α, [1×1]
- spherical harmonics, [3×9]

图 12 3D Gaussian 参数

在 3D Gaussian Splatting 中不会直接去优化协方差矩阵，而是将协方差矩阵分解为一个缩放矩阵 S 和旋转矩阵 R。如公式 2 所示，其具体的推导过程也在图 5 中给出。

$$\Sigma = RSS^T R^T$$

公式 2 协方差矩阵

2.1.4.2 渲染步骤

这一部分是 3D Gaussian Splatting 中较难的部分，在论文中，作者给出了其算法如图 13 所示，对于其中的具体步骤如下所示。

1、根据当前相机的位姿，删除一些当前不可见的 Gaussian。

2、把 3D Gaussian 投影，将其转变成 2D Gaussian，具体做法为给定观看变换矩阵 W，在相机坐标下的协方差矩阵 Σ'如公式 3 所示，其中 J 是投影变换的仿射近似的雅可比矩阵。

$$\Sigma' = JW\,\Sigma\,W^T J^T$$

公式 3 Σ'公式

3、在得到 2D Gaussian 后会将屏幕分割为 16*16 的 tiles，并记录每个 Gaussian 包含了那些 tile。并以 Gaussian 与 tile 的匹配对来作为单位。

4、根据这些匹配对来进行排序，Gaussian 将按照每个 tile 来进行排列。这样就可以对 tile 中的任意一个像素，找到对其有影响的 Gaussian。

5、通过对每个像素的进行着色，将 Gaussian 渲染出来，得到最终的 RGB 颜色。

Algorithm 2 GPU software rasterization of 3D Gaussians

w, h: width and height of the image to rasterize
M, S: Gaussian means and covariances in world space
C, A: Gaussian colors and opacities
V: view configuration of current camera

```
function RASTERIZE(w, h, M, S, C, A, V)
    CullGaussian(p, V)                          ▷ Frustum Culling
    M', S' ← ScreenspaceGaussians(M, S, V)      ▷ Transform
    T ← CreateTiles(w, h)
    L, K ← DuplicateWithKeys(M', T)             ▷ Indices and Keys
    SortByKeys(K, L)                            ▷ Globally Sort
    R ← IdentifyTileRanges(T, K)
    I ← 0                                       ▷ Init Canvas
    for all Tiles t in I do
        for all Pixels i in t do
            r ← GetTileRange(R, t)
            I[i] ← BlendInOrder(i, L, r, K, M', S', C, A)
        end for
    end for
     return I
end function
```

图 13 3D Gaussian 渲染算法

2.1.4.3 优化步骤

3D Gaussian Splatting 在优化过程中主要提出了两种策略，以处理图 14 中显示的两种情况。当一个高斯分布在二维平面上具有较大梯度时，存在两种可能的情况。第一种情况是欠重建（under reconstruction），通常出现在细节丰富的区域。对此，策略是复制一个相同大小的高斯分布，让这两个高斯分布分别表示场景中的不同区域。

第二种情况是过重建（over reconstruction），通常出现在初始点云较少的场景中。在细节较多的区域，这里会用一个较大的单独高斯分布来表示。这时，需要将这个较大的高斯分布分裂为两个较小的高斯分布，分裂的比例为 1/1.6，这是 3D Gaussian Splatting 团队通过多次实验确定的经验值。然后，这两个较小的高斯分布分别表示该区域中的不同部分。

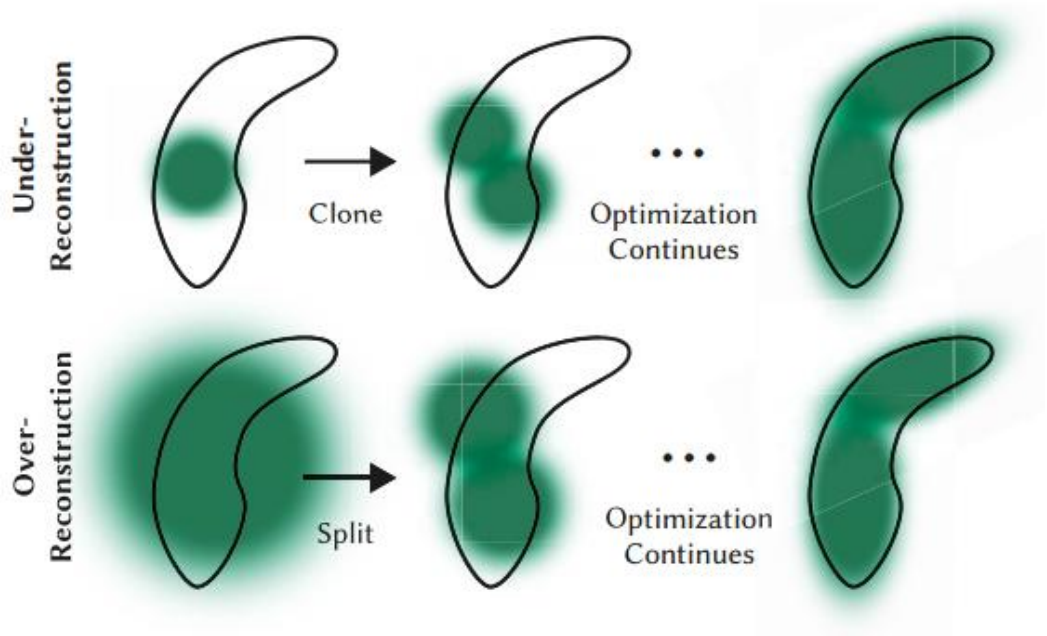

图 14 自适应高斯优化方案

2.1.5 论文中 3D Gaussian 存在的问题

3D Gaussian Splatting 在原文中无法处理动态物体。尽管已有一些其他论文对动态物体的处理做了一些工作，但这些方法一般是针对相对简单的动态物体进行处理，未涉及大场景中的运动轨迹。除此之外，在大场景下，Gaussian Splatting 需要良好的初始化点云。如果使用随机的初始化点云，可能会导致效果不佳。

2.2 Street Gaussians 论文详读

我选取了一篇基于 3D Gaussian 的街景三维重建论文 *Street Gaussians for Modeling Dynamic Urban Scenes* [2]，其论文地址(https://arxiv.org/abs/2401.01339)。

2.2.1 Street Gaussians 关键技术

Street Gaussians 这篇论文的目标是未来实现动态城市场景的实时渲染，其中实时渲染采用 3D Gaussian Splatting 来进行解决，动态效果使用 Scene Graph 技术来实现，而城市场景使用 LiDAR Prior 来生成高精度的点云，对模型进行一个比较好的初始化。

论文整体可以分为三个部分，分别为背景表示(Background Representation)、物体表示(Object Representation)和组合渲染(Compositional Rendering)。具体的处理流程如图 15 所示。接下来我将对这三个步骤进行分别的讲解。

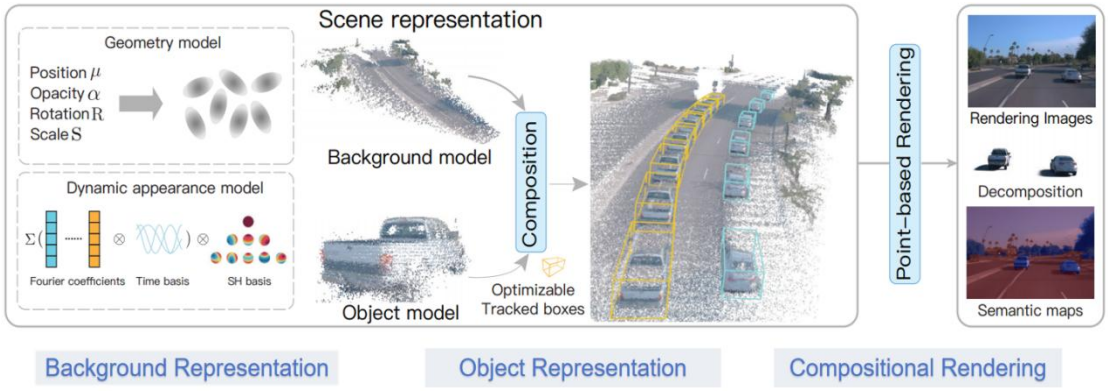

图 15 Street Gaussians 处理流程

2.2.1.1 背景表示

背景表示中主要有背景模式和天空模式，这里的处理方法基本上与 3D Gaussian Splatting 一致，不同点是在有二维语义的输入的情况下，会在每个 Gaussian 中都定义了可学习的 Semantic logits，语义图可以简单地通过将公式 4 中的颜色 c 更改为语义 logits β 来实现。

$$\mathbf{c}=\sum_{i\in N}\mathbf{c}_i\alpha_i\prod_{j=1}^{i-1}(1-\alpha_j),$$

公式 4 α-blending 公式

论文对于背景的主要贡献点在于初始化阶段，Street Gaussians 使用 LiDAR 点云来启动模型，并通过过滤掉不可见和运动的的点云来创建背景模型。LiDAR 点云的颜色信息是通过将点云投影到相应的图像平面上，并查询像素值来获得的。因为 LiDAR 点云在覆盖大范围区域时存在限制，因此 Street Gaussians 还结合了 SfM 点云来处理远距离区域的数据。

Street Gaussians 对天空模型的处理采用了一种特定的方法。首先，通过获取天空的掩模，将天空中的点云固定在一个球体上。在优化过程中，论文还对于天空高斯模型的位置进行了一定的限制，确保其保持在一个合理的范围内，不会产生不良的视觉效果或者干扰相机视野的情况。这种处理方法有效地控制了天空模型在场景优化中的位置和影响。

2.2.1.2 物体表示

在物体的表示中，每个对象被描述为一组可优化的履带式车辆姿态以及一个点云。这种表示方法能够有效地捕捉物体的姿态和形状，并通过优化过程生成如图 16 所示的高质量点云效果。对于那些远距离且点云信息较少的物体，论文采用了一种直接在边界框内初始化随机点云的方法来处理。

图 16 构建物体的初始化点云

在特定时刻 t，可以获取到每个边界框的姿态信息。对于每个物体的 Gaussian 模型，可以通过公式 5 这样的的变换，将物体从其局部坐标系转换到世界坐标系。在这个过程中，物体的位置、姿态等参数需要根据具体的坐标系变换规则进行调整，以确保高斯模型能够正确地反映物体在整个场景中的位置和方向。对于其他参数，如物体的形状、尺度等与坐标系无关，因此在转换过程中可以保持不变，只需适时地应用到新的坐标系中。

$$\boldsymbol{\mu}_w = \mathbf{R}_t \boldsymbol{\mu}_o + \mathbf{T}_t,$$
$$\mathbf{R}_w = \mathbf{R}_o \mathbf{R}_t^T,$$

公式 5 车辆的跟踪姿态转换公式

在点云中，每个点都被分配了一个三维高斯和一个语义逻辑标签和一个动态外观模型。这个动态外观模型（dynamic appearance model）是论文的另一个创新点，其是为了更准确地捕捉动态物体在不同时刻与环境的互动，这是球谐函数单独表示动态物体所不能有效处理的。因此，论文提出了一种结合时间基函数的方法，将时间信息融入球谐函数的系数中。具体而言，论文首先通过优化一组傅里叶系数来反映动态物体的光照变化。然后，利用逆傅里叶变换将光照信息传递给球谐函数的基函数，在公式 6 给出，从而得到最终对应的颜色信息。全部过程如图 17 所示。

$$z_{m,l} = \sum_{i=0}^{k-1} \boldsymbol{f}_i \cos\left(\frac{i\pi}{N_t} t\right).$$

公式 6 实值离散傅里叶反变换公式

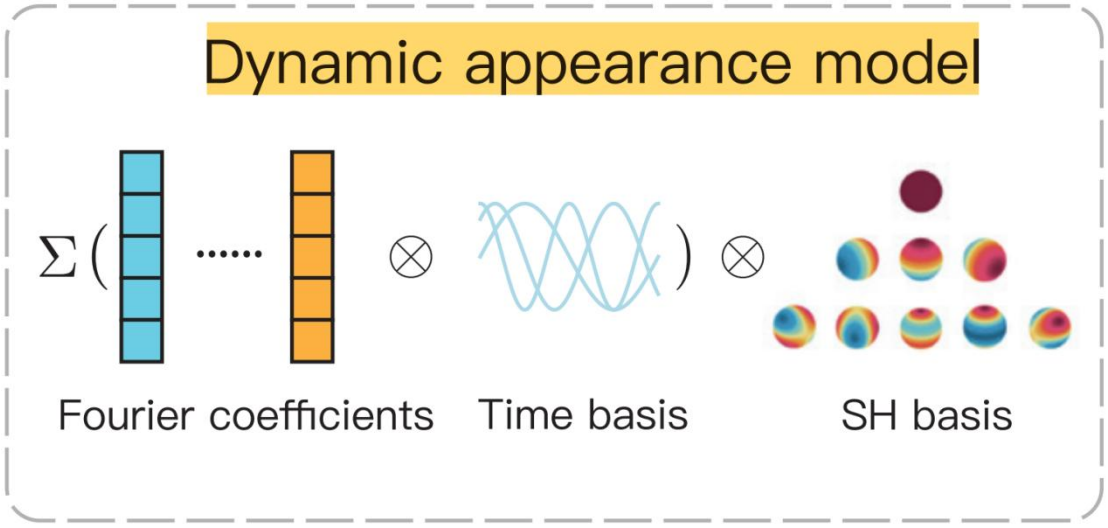

图 17 动态外观模型的实现过程

基于此方法构建的物体，光影效果更加光滑，而且没有显著的伪影。对比图如图 18 所示。

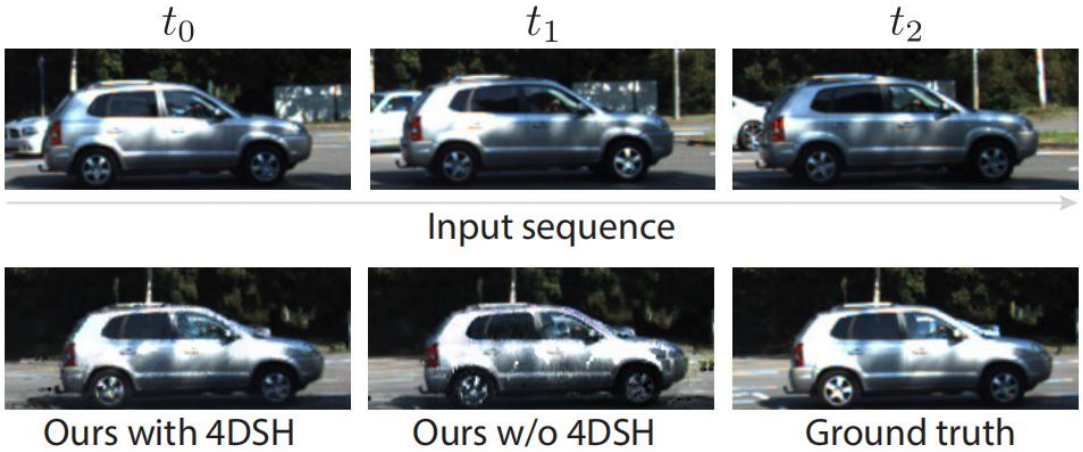

图 18 动态外观模型与其他模型的对比图

2.2.1.3 组合渲染

在组合渲染这一部分，由于物体、环境和天空的位置都以显式的 Gaussian 表示，因此可以简单地将它们的位置信息拼接起来。这样做之后，可以利用 3D Gaussian Splatting 中提到的方法将这些物体投影到 2D Gaussian 上。具体的流程如图 19 所示。

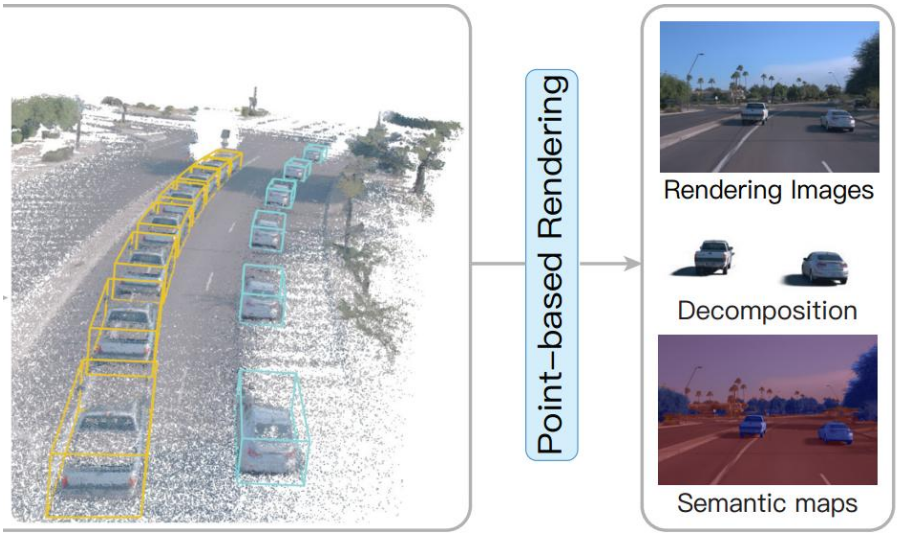

图 19 组合渲染流程

在将场景中的物体投影为 2D Gaussian 后，可以利用 alpha-blending 技术进行渲染。这种方法不仅可以用于颜色的渲染，还可以适用于场景中的语义信息和透明度等其他信息的渲染。如公式 7 所示。

$$\mathbf{C} = \sum_{i \in N} \mathbf{c}_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$$
$$\beta = \sum_{i \in N} \beta_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$$
$$\mathbf{O} = \sum_{i \in N} \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$$

公式 7 alpha-blending 渲染公式

2.2.1.4 训练步骤

论文中提出了一种 Tracking pose optimization 的训练方法，旨在应对模型处理过程中的噪声干扰问题。在实际应用中，由于传感器噪声等因素的影响，直接使用原始的姿态进行场景表示的优化可能会导致渲染质量的下降。为了解决这一问

题，Street Gaussians 这篇论文引入了一种创新的训练方法，即通过向每个变换矩阵添加可学习的变换，将跟踪的姿态视作可调节的参数。这种方法允许模型在训练中适应和学习不同噪声条件下的姿态变化，提高场景表示的稳定性和准确性。

$$\mathbf{R}_t' = \mathbf{R}_t \Delta \mathbf{R}_t,$$
$$\mathbf{T}_t' = \mathbf{T}_t + \Delta \mathbf{T}_t,$$

公式 8 跟踪姿态优化公式

公式中这些变换的梯度可以直接计算得到，无需依赖隐式函数或额外的中间过程。在反向传播过程中，这些梯度计算是直接且有效的，不需要进行任何额外的复杂计算步骤，这使得训练速度相对较快。通过这种方式优化后的效果有显著的提升，对比如表 1 所示。

	Sequence A		
	w/o pose opt.	with pose opt.	with GT poses
MARS [55]	21.01	22.91	24.97
Ours	23.91	26.12	25.34
	Sequence B		
	w/o pose opt.	with pose opt.	with GT poses
MARS [55]	19.61	21.06	22.17
Ours	20.29	22.14	22.59

表 1 跟踪位姿优化的研究对比

2.2.1.5 损失函数

论文中使用三种损失函数来共同优化场景表示和跟踪姿态。Color loss：类似于 3D Gaussian 论文中的重建损失，这是渲染图像和观察图像之间的差异度量，包含 L1 loss 和 SSIM loss。Color Semantic loss：这是一个可选的损失项，如果提供了预测的二维语义作为输入，可以计算渲染语义和输入语义之间的 softmaxcross-entropy loss。Regularization loss：这是一个约束前景车辆的熵正则化项，目的是消除浮动点并增强分解效果。在论文中设定 λ 1 = 0.1， λ 2 = 0.1。具体的损失函数如图 20 所示。

Color loss $\mathcal{L}_{\text{color}} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}}$

Color Semantic loss(Optional) $\mathcal{L}_{sem} = -\sum_{l=1}^{L} \beta_l^* \log \beta_l$

Regularization loss $\mathcal{L}_{reg} = -\sum (O_{obj} log O_{obj} + (1-O_{obj}) log(1-O_{obj}))$

Final loss $\mathcal{L} = \mathcal{L}_{\text{color}} + \lambda_1 \mathcal{L}_{\text{sem}} + \lambda_2 \mathcal{L}_{\text{reg}}.$

图 20 损失函数

2.2.2 当前存在的问题

这篇论文和之前提到的关于 3D Gaussian 的研究一样，两者都存在依赖于高质量的初始点云数据来确保在复杂场景下的渲染效果。尽管提出通过提供精确的初始点云来规避模型的不足，这种方法本质上并未解决根本问题，只是通过优化输入数据来避免渲染缺陷，达到改善渲染效果的目的。然而，现实应用场景中，如缺少如雷达这样的高精度点云数据支持，模型的这一固有弱点将会暴露。

此外，该论文在建模过程中并不能区分镜面和非镜面区域，在镜面反射部分会产生不合理的三维高斯分布，严重影响重建，导致生成有缺陷的网格。

论文中虽然实现了对车辆的动态处理，但没有涉及行人等其他动态物体的处理。如果该技术期望在现实生活中获得广泛应用，对于行人等动态物体的处理是必须要考虑的。论文将天空、背景和动态车辆分开处理分开处理，这样生成的最终结果效果很好，但是如果能进一步对场景内的光照进行独立渲染，可能会进一步提高整体场景的真实感和视觉效果。尽管论文实现了动态城市街景的实时渲染，但该方法没有应用于整个城市的全景渲染，这个挑战也是值得尝试的。

2.2.3 潜在的改进方向

上述论文存在的问题都是可以改进的方向。此外，我还有一个动态车辆轨迹预测并构建未来时刻 3D 点云的想法。这个想法是基于车辆运动轨迹的状态较少，如果笼统来说总共有前进，左转，右转和后退四种状态，每种状态可能会有角度等因素的偏差，这里假设影响这些状态的公式都可以拟合出来，

在这个假设中，存在 $A_i(t_k)$、$B_i(t_k)$和 $C_i(t_k)$三种矩阵，其中 i=(0,1,2,3)分别代表一种车辆的状态(前进，左转，右转和后退)。因此共计 12 个关于时间变量 t 的的矩阵，这里需要有角度变量的介入，假设这里角度变量的判断与介入在另一步中进行。其中，$A_i(t_k)$和 $B_i(t_k)$矩阵是控制汽车位姿变化的矩阵，$C_i(t_k)$矩阵是结果矩阵，$X(t_k)$存储着当前车辆的位姿信息。在这个假设中，$A_i(t_k)$、$B_i(t_k)$和 $C_i(t_k)$三种矩阵都可以通过大量的车辆形势数据中的拟合出来，由相机或者其他传感器对汽车当前运动状态进行判断，选择不同的拟合矩阵进行预测。

在这个想法下，我只能保证公式的推导过程是正确的，但是公式的前提条件是否满足，假设是否成立，因为时间有限，我暂时无法证明。

图 21 是公式推导的过程，假设该公式的前提条件全部成立，可以看到最终推导出由当前汽车位姿信息得到下一时刻汽车位姿信息的预测公式。这里我给出了两种形式的推导过程一种是书写的形式，而另一种使用 Latex 编写，其内容都是相同的。

在离散时间维度下，存在

$$A(t_k)X(t_k)B(t_k)=C(t_k)$$

定义误差矩阵

$$Z(t_k)=A(t_k)X(t_k)B(t_k)-C(t_k)$$

对其进行向量化

$$vec(Z(t_k))=(B^T(t_k)\otimes A(t_k))vec(X(t_k))-vec(C(t_k))$$

为了简化，我这里定义如下：

$$\mathcal{Z}(t_k)=vec(Z(t_k))$$
$$x(t_k)=vec(X(t_k))$$
$$c(t_k)=vec(C(t_k))$$
$$M(t_k)=B^T(t_k)\otimes A(t_k)$$

$\therefore \mathcal{Z}(t_k)=M(t_k)x(t_k)-c(t_k)$

对于下一时刻误差公式

$$\mathcal{Z}(t_{k+1})=M(t_{k+1})x(t_{k+1})-c(t_k)$$

当 $t_k\to+\infty$，$\mathcal{Z}(t_k)\to 0$

假设 $\mathcal{Z}(t_{k+1})=w\mathcal{Z}(t_k)$ 这里的 $w>0$，是一个参数

$$\mathcal{Z}(t_{k+1})=\mathcal{Z}(t_k)+\dot{\mathcal{Z}}(t_k)(t_{k+1}-t_k)+\left(\frac{\partial\mathcal{Z}(t_k)}{\partial x(t_k)}\right)(x(t_{k+1})-x(t_k))+O(\tau^2)$$
$$=\mathcal{Z}(t_k)+\dot{\mathcal{Z}}(t_k)\tau+\left(\frac{\partial\mathcal{Z}(t_k)}{\partial x(t_k)}\right)(x(t_{k+1})-x(t_k))+O(\tau^2)$$

这里的 $\tau=t_{k+1}-t_k$，这里忽略 $O(\tau^2)$ 项.

$$\mathcal{Z}(t_{k+1})-\mathcal{Z}(t_k)$$
$$=(w-1)(M(t_k)\cdot x(t_k)-c(t_k))$$
$$=\tau\dot{\mathcal{Z}}(t_k)+\left(\frac{\partial\mathcal{Z}(t_k)}{\partial x(t_k)}\right)(x(t_{k+1})-x(t_k))$$

$\because$
$$\begin{cases}\dfrac{\partial\mathcal{Z}(t_k)}{\partial x(t_k)}=M(t_k)\\ \dot{\mathcal{Z}}=\dot{M}(t_k)x(t_k)-\dot{c}(t_k)\end{cases}$$

$\therefore$ 我们可以得到

$$M(t_k)vec(X(t_{k+1})-X(t_k))$$
$$=(w-1)(M(t_k)x(t_k)-c(t_k))-\tau(\dot{M}(t_k)x(t_k)-\dot{c}(t_k))$$

令 $H(t_k)=(w-1)(M(t_k)x(t_k)-c(t_k))-\tau(\dot{M}(t_k)x(t_k)-\dot{c}(t_k))$

$$M(t_k)vec(X(t_{k+1})-X(t_k))=H(t_k)$$

假设 $M(t_k)$ 可逆

$$vec(X(t_{k+1}))=vec(x(t_k))+M^{-1}(t_k)H(t_k)$$

In the discrete time framework, the following formula exists

$$A(t_k)X(t_k)B(t_k) = C(t_k) \tag{1}$$

When solving equation (1), we have designed a discrete time-varying error-monitoring matrix as shown below

$$Z(t_k) = A(t_k)X(t_k)B(t_k) - C(t_k) \tag{2}$$

For a more in-depth exploration, leveraging matrix linear operations and vectorization techniques, the discrete-form time-variant error-monitoring matrix (2) mentioned above can be reconfigured into a discrete time-varying error vector. It is crucial to emphasize that, on a fundamental level, these discrete time-varying error-monitoring matrix and error-monitoring vector are interchangeable. Consequently, we have

$$\text{vec}(Z(t_k)) = \sum_{i=0}^{L}(B^T(t_k) \otimes A(t_k))\text{vec}(X(t_k)) - \text{vec}(C(t_k)) \tag{3}$$

In order to make the derivation process more readable, we define

$$\begin{aligned} z(t_k) &= \text{vec}(Z(t_k)) \\ x(t_k) &= \text{vec}(X(t_k)) \\ c(t_k) &= \text{vec}(C(t_k)) \end{aligned}$$

and

$$M(t_k) = (B^T(t_k) \otimes A(t_k))$$

Then, the above equation (3) can be rewritten as

$$z(t_k) = M(t_k) \cdot x(t_k) - c(t_k) \tag{4}$$

In addition to the previously mentioned instant time t_k, it is imperative to define error functions at the subsequent instant time t_{k+1} as follows:

$$z(t_{k+1}) = M(t_{k+1}) \cdot x(t_{k+1}) - c(t_{k+1}) \tag{5}$$

When $t_k \to +\infty$, $z(t_k)$ approaches zero , we define $z(t_{k+1}) = \omega z(t_k)$, where $\omega > 0$ is a design parameter. Subsequently, one can derive the following result

$$z(t_{k+1}) - z(t_k) = \omega z(t_k) - z(t_k) \tag{6}$$

We define τ as the sampling gap with $\tau \in (0, 1)$. Then, the second-order Taylor expansion of $z(t_{k+1})$ with respect to $x(t_k)$ and t_k can be presented as

$$\begin{aligned} &z(t_{k+1}) \\ &= z(t_k) + \dot{z}(t_k)(t_{k+1} - t_k) \\ &+ (\partial z(t_k)/\partial x(t_k))(x(t_{k+1}) - x(t_k)) + O(\tau^2) \\ &= z(t_k) + \dot{z}(t_k)\tau \\ &+ (\partial z(t_k)/\partial x(t_k))(x(t_{k+1}) - x(t_k)) + O(\tau^2) \end{aligned} \tag{7}$$

By ignoring the $O(\tau^2)$ item in the above formula, we further have

$$\begin{aligned} &z(t_{k+1}) - z(t_k) \\ &= (\omega - 1)(M(t_k) \cdot x(t_k) - c(t_k)) \\ &= \tau\dot{z}(t_k) + (\partial z(t_k)/\partial x(t_k))(x(t_{k+1}) - x(t_k)) \end{aligned} \tag{8}$$

In equation (4), we consider the partial derivatives of t_k and $x(t_k)$, we have

$$\begin{cases} \dot{z}(t_k) & = \dot{M}(t_k)x(t_k) - \dot{c}(t_k) \\ \frac{\partial z(t_k)}{\partial x(t_k)} & = M(t_k) \end{cases} \tag{9}$$

Combining (8) and (9), we obtain

$$\begin{aligned} &(\omega - 1)(M(t_k)x(t_k) - c(t_k)) \\ &= \tau(\dot{M}(t_k)x(t_k) - \dot{c}(t_k)) + M(t_k)(x(t_{k+1}) - x(t_k)) \end{aligned} \tag{10}$$

To simplify the presentation, we denote that

$$\begin{aligned} &M(t_k)\text{vec}(X(t_{k+1}) - X(t_k)) = \\ &(\omega - 1)(M(t_k)\text{x}(t_k) - c(t_k)) - \tau(\dot{M}(t_k)\text{x}(t_k) - \dot{c}(t_k)) \end{aligned} \tag{11}$$

To enhance the clarity of the derivation process, we define

$$\begin{aligned} &H(t_k) = \\ &(\omega - 1)(M(t_k)\mathbf{x}(t_k) - c(t_k)) - \tau(\dot{M}(t_k)\mathbf{x}(t_k) - \dot{c}(t_k)) \end{aligned} \tag{12}$$

Then, we have

$$M(t_k)\text{vec}(\text{X}(t_{k+1}) - \text{X}(t_k)) = H(t_k) \tag{13}$$

We emphasize the non-singularity of matrix $M(t_k)$, ensuring the existence of matrix inversion within the temporal interval $[t_0, t_f]$. Thus, Equation (13) can be expressed more precisely.

$$\text{vec}(\text{X}(t_{k+1})) = \text{vec}(\text{X}(t_k)) + M^{-1}(t_k)H(t_k) \tag{14}$$

Then, the derivation process is completed.

图 21 预测过程推导

二、附录

以下代码均在 GitHub 上开源，开源地址如下。

(https://github.com/Metastarx/ZJU)

```
def computeCov3D(scale, mod, rot):
    # 创建缩放矩阵
    S = np.array(
        [[scale[0] * mod, 0, 0], [0, scale[1] * mod, 0], [0, 0, scale[2] * mod]]
    )

    # 归一化四元数以获得有效的旋转矩阵
    # 我们使用旋转矩阵
    R = rot

    # 计算 3D 世界的协方差矩阵 Sigma
    M = np.dot(R, S)
    cov3D = np.dot(M, M.T)

    return cov3D
```

代码 1 computeCov3D 函数

```
frame = create_canvas(700, 700)
angle = 0
eye = [0, 0, 5]
pts = [[2, 0, -2], [0, 2, -2], [-2, 0, -2]]
viewport = get_viewport_matrix(700, 700)

# get mvp matrix
mvp = get_model_matrix(angle)
mvp = np.dot(get_view_matrix(eye), mvp)
mvp = np.dot(get_proj_matrix(45, 1, 0.1, 50), mvp)  # 4x4

# loop points
pts_2d = []
for p in pts:
    p = np.array(p + [1])  # 3x1 -> 4x1
    p = np.dot(mvp, p)
    p /= p[3]
    # viewport
    p = np.dot(viewport, p)[:2]
```

```
        pts_2d.append([int(p[0]), int(p[1])])
    vis = 1
    if vis:
        # visualize 3d
        fig = plt.figure()
        pts = np.array(pts)
        x, y, z = pts[:, 0], pts[:, 1], pts[:, 2]
        ax = Axes3D(fig)
        ax.scatter(x, y, z, s=80, marker="^", c="g")
        ax.scatter([eye[0]], [eye[1]], [eye[2]], s=180, marker=7, c="r")
        ax.plot_trisurf(x, y, z, linewidth=0.2, antialiased=True, alpha=0.5)
        plt.show()

        # visualize 2d
        c = (255, 255, 255)
        for i in range(3):
            for j in range(i + 1, 3):
                cv2.line(frame, pts_2d[i], pts_2d[j], c, 2)
        cv2.imshow("screen", frame)
        cv2.waitKey(0)
```

代码 2 光栅化代码

三、引用

[1] Kerbl, B., Kopanas, G., Leimkühler, T., & Drettakis, G. (2023). 3D Gaussian Splatting for Real-Time Radiance Field Rendering. ACM Transactions on Graphics, 42(4).Yan, Y., Lin, H., Zhou, C., Wang, W., Sun, H., Zhan, K., ... Peng, S. (2024). Street Gaussians for Modeling Dynamic Urban Scenes. arXiv preprint arXiv:2401.01339.

[2] Yan, Y., Lin, H., Zhou, C., Wang, W., Sun, H., Zhan, K., ... Peng, S. (2024). Street Gaussians for Modeling Dynamic Urban Scenes.