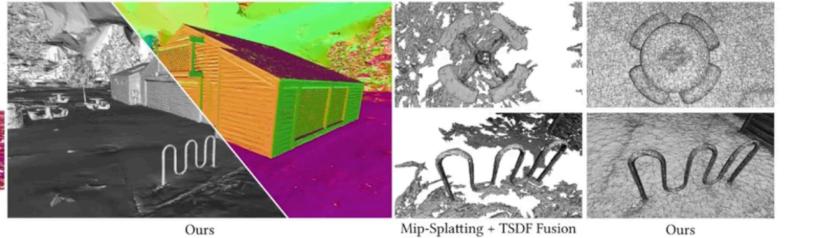
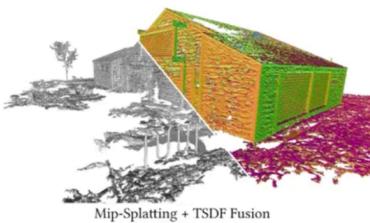


GOF

Gaussian Opacity Fields: Efficient and Compact Surface Reconstruction in Unbounded Scenes

一、核心见解与创新点：



SuGaR:

提取表面点，中心点渲染深度图估计法向量->泊松重建
问题：忽略高斯的不透明度和尺度，深度图不准确

2DGs:

使用交点渲染深度图->tsdf
问题：tsdf薄片形状建模不好，无边界不能建模，复杂形状网格太多太慢

- 高斯不透明场：从3d高斯中定义不透明场，由射线上分布最大值作为交点，按照交点定义不透明度，取所有射线中最小值得到不透明场值
- Mesh生成：利用不透明度场，改进网络提取、简化和细化
- 训练过程中用射线近似高斯的法线

GOF

Gaussian Opacity Fields: Efficient and Compact Surface Reconstruction in Unbounded Scenes 2024年4月 arxiv

一、不透明场的定义

(1) 从3维高斯中定义射线交点：

世界坐标系：

相机中心点： \mathbf{o} 射线方向： \mathbf{r}
射线上一点： $\mathbf{x} = \mathbf{o} + t\mathbf{r}$

高斯坐标系：

$$\begin{aligned}\mathbf{o}_g &= (\mathbf{R}_k(\mathbf{o} - \mathbf{p}_k)) \odot \mathbf{s}_k^{-1} \\ \mathbf{r}_g &= \mathbf{R}_k \mathbf{r} \odot \mathbf{s}_k^{-1} \\ \mathbf{x}_g &= \mathbf{o}_g + t \mathbf{r}_g\end{aligned}$$

高斯坐标系内沿射线方向上的分布是一维高斯分布：

$$G_k^{1D}(t) = G_k(\mathbf{x}_g) = e^{-\frac{1}{2} \mathbf{x}_g^T \mathbf{x}_g}$$

高斯分布的最大值
定义为交点：

$$t^* = -\frac{B}{A}$$

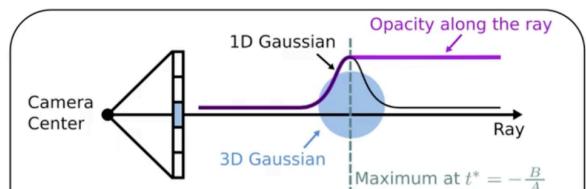
$$A = \mathbf{r}_g^T \mathbf{r}_g \text{ and } B = \mathbf{o}_g^T \mathbf{r}_g.$$

由交点确定贡献

$$\mathcal{E}(G_k, \mathbf{o}, \mathbf{r}) = G_k^{1D}(t^*)$$

颜色渲染：

$$c(\mathbf{o}, \mathbf{r}) = \sum_{k=1}^K c_k \alpha_k \mathcal{E}(G_k, \mathbf{o}, \mathbf{r}) \prod_{j=1}^{k-1} (1 - \alpha_j \mathcal{E}(G_j, \mathbf{o}, \mathbf{r}))$$



射线在高斯中的分布（高斯坐标系）

高斯坐标系内分布为三维标准正态分布

$$\mathbf{X}_g \sim N(\mathbf{0}, \mathbf{I}) \text{ 其中, 其中, } \mathbf{X}_g = (X_{g1}, X_{g2}, X_{g3})$$

$$\text{射线 } \mathbf{X}_g = \mathbf{o}_g + t \mathbf{r}_g$$

t 的分布 T 可以用 X 在 r_g 上的投影表示：

$$T \sim X_g \cdot r = r_{g1} X_{g1} + r_{g2} X_{g2} + r_{g3} X_{g3}$$

由正态分布性质得

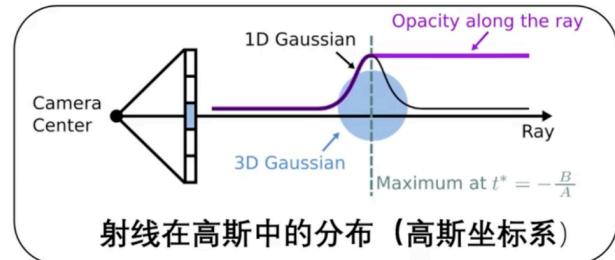
$$T \sim N(0, r_{g1}^2 + r_{g2}^2 + r_{g3}^2) = N(0, 1)$$

高斯坐标系内沿射线方向上的分布推导

一、不透明场的定义

(2) 由交点定义沿射线的不透明场:

$$O_k(\mathcal{G}_k, \mathbf{o}, \mathbf{r}, t) = \begin{cases} \mathcal{G}_k^{1D}(t) & \text{if } t \leq t^* \\ \mathcal{G}_k^{1D}(t^*) & \text{if } t > t^* \end{cases} \quad \mathcal{G}_k^{1D}(t) = \mathcal{G}_k(\mathbf{x}_g) = e^{-\frac{1}{2}\mathbf{x}_g^T \mathbf{x}_g}$$



沿射线的不透明度为

$$O(\mathbf{o}, \mathbf{r}, t) = \sum_{k=1}^K \alpha_k O_k(\mathcal{G}_k, \mathbf{o}, \mathbf{r}, t) \prod_{j=1}^{k-1} (1 - \alpha_j O_k(\mathcal{G}_k, \mathbf{o}, \mathbf{r}, t))$$

定义和射线无关的不透明场:
所有射线方向的最小值

$$O(\mathbf{x}) = \min_{(\mathbf{r}, t)} O(\mathbf{o}, \mathbf{r}, t)$$

$$d(p) = \sum_g \alpha_g \exp\left(-\frac{1}{2}(p - \mu_g)^T \Sigma_g^{-1}(p - \mu_g)\right)$$

SuGaR中的密度函数

GOF

Gaussian Opacity Fields: Efficient and Compact Surface Reconstruction in Unbounded Scenes 2024年4月 arxiv

二、正则项

(1) 深度分布 Depth Distortion:

$$\mathcal{L}_d = \sum_{i,j} \omega_i \omega_j |t_i - t_j|$$

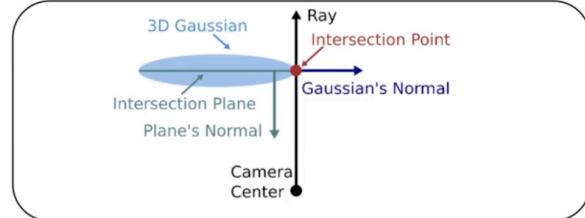
使得两个交点尽可能靠近

(2) 深度法线一致性 Normal Consistency:

$$\mathcal{L}_n = \sum_i \omega_i (1 - \mathbf{n}_i^\top \mathbf{N})$$

$$\mathbf{n}_i = -\mathbf{R}^T (\mathbf{r}_g \odot \mathbf{s}^{-1})$$

使用射线方向近似法线



```
// transform camera center and ray to gaussian's local coordinate system
// current center is zero
float3 cam_pos_local = {view2gaussian_j[12], view2gaussian_j[13], view2gaussian_j[14]};
float3 ray_local = transformPoint4x_without_t(ray_point, view2gaussian_j);

// scale the ray_local and cam_pos_local
double3 ray_local_scaled = {ray_local.x / scale_j.x, ray_local.y / scale_j.y, ray_local.z / scale_j.z};

// use ray_local_scaled is the normal direction
const float3 point_local_scaled_t = (-ray_local_scaled.x, -ray_local_scaled.y, -ray_local_scaled.z);

// here is the gradient at node_t
float3 point_normal_t = {point_local_scaled_t.x / scale_j.x, point_local_scaled_t.y / scale_j.y, point_local_scaled_t.z / scale_j.z};

float length = sqrt(point_normal_t.x * point_normal_t.x + point_normal_t.y * point_normal_t.y + point_normal_t.z * point_normal_t.z);
if (length < 0.001) {
    point_normal_t = {0, 0, 0};
} else {
    point_normal_t = point_normal_t / length;
}

point_normal_t = {point_normal_t.x / length, point_normal_t.y / length, point_normal_t.z / length};

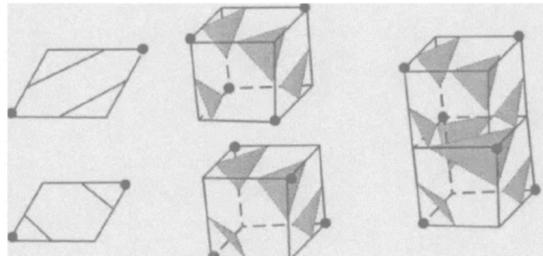
// transform to world space
float3 transformed_normal_t;
if (transformed_normal_t.x < 0) {
    transformed_normal_t = -transformed_normal_t;
}
transformed_normal_t.x = point_normal_t.x * vimagegaussian_1[1] + point_normal_t.y * vimagegaussian_1[2] + point_normal_t.z * vimagegaussian_1[3];
transformed_normal_t.y = point_normal_t.x * vimagegaussian_1[4] + point_normal_t.y * vimagegaussian_1[5] + point_normal_t.z * vimagegaussian_1[6];
transformed_normal_t.z = point_normal_t.x * vimagegaussian_1[7] + point_normal_t.y * vimagegaussian_1[8] + point_normal_t.z * vimagegaussian_1[9];
```

源码cuda代码中采用射线方向作为法向量

三、网格提取

(1) 对每个高斯定义一个3d框:

我们在每个高斯的3-sigma处生成一个3D边界框，其中中心具有最高的不透明度，而角落具有最低的不透明度。



Marching Cube 算法示意

(2) 使用3d框构建三角网格:

然后用3D边界框的中心和角落 建立四面体网格。使用CGAL库进行 Delaunay三角化，来构建四面体单元格。

- 改进1：快速求解不透明度，采用3dgs中分片渲染的形式加速计算
- 改进2：Marching Tetrahedra算法 采用线性插值来找到水平集，这里不透明场不符合线性关系，改用递增的二分查找

四、结果与总结

- 精度提高，而且背景质量好
- GOF是通用的不透明场定义，但是计算速度慢
- 法线一致性损失效果挺好 令人迷惑

	Precision ↑	Recall ↑	F-score ↑
A. Mip-Splatting w/ TSDF	0.15	0.25	0.16
B. Mip-Splatting w/ GOF	0.40	0.33	0.36
C. Ours w/o GOF	0.37	0.45	0.39
D. Ours w/o normal consistency	0.41	0.35	0.37
E. Ours w/o decoupled appearance	0.49	0.39	0.43
F. Ours	0.54	0.42	0.46

