- Author: Diantao Tu
- Date: 2024-02-17 13:51:03

相机位姿格式

这一段内容只关于colmap的相机位姿格式。已知colmap保存的相机位姿是 `world to camera` 即 T_{cw}.
在 `read_extrinsics_text(path_to_model_file)` 函数中, 读取了相机的位姿数据, 并且保存为 `Image` 类型的数据.

```
cam_extrinsics = read_extrinsics_text(cameras_extrinsic_file)
```

在 `readColmapCameras` 函数中, 对图像信息进行了进一步处理, 这里要注意, 相机的旋转被转换成了旋转矩阵的形式, 并且进行了转置. 也就是说此时的旋转矩阵是 R_{wc}, 但平移仍然是 t_{cw}.

```
def readColmapCameras(cam_extrinsics, cam_intrinsics, images_folder):
    R = np.transpose(qvec2rotmat(extr.qvec))
    T = np.array(extr.tvec)
    cam_info = CameraInfo(R=R, T=T)

cam_infos_unsorted = readColmapCameras(cam_extrinsics, cam_intrinsics, images_folder)
```

所以最终的 `Scene` 类中, `scene_info` 变量中保存的相机位姿是 R_{wc} 和 t_{cw}.

```
scene_info = sceneLoadTypeCallbacks["Colmap"](args.source_path, args.images, args.eval)
```

然后根据 `scene_info` 中的数据, 会生成多尺度的图像(默认只有一个原始尺度)用于后续的训练.

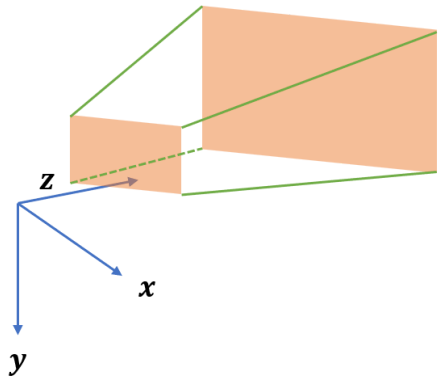
```
print("Loading Training Cameras")
self.train_cameras[resolution_scale] = cameraList_from_camInfos(scene_info.train_cameras,
resolution_scale, args)
print("Loading Test Cameras")
self.test_cameras[resolution_scale] = cameraList_from_camInfos(scene_info.test_cameras,
resolution_scale, args)
```

透视投影矩阵

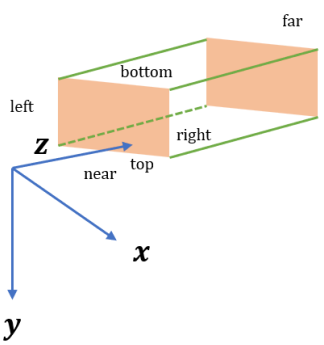
以下的推导过程借鉴了这篇文章以及这篇文章. 这两篇文章是计算机图形学相关的, 其中涉及了不同坐标系之间的变换(例如左手坐标系, 坐标轴方向变化)等等. 但 Gaussian Splatting 中的透视矩阵没有这么复杂. 以下为详细的推导过程.

三个坐标系

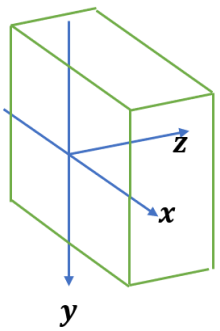
投影矩阵涉及三个坐标系, 他们的可视化如下图所示



观测空间



正交空间



标准设备坐标

- 观测空间: 可以看作是相机坐标系, 也就是物体原始所在的三维空间. 这个空间是一个截锥(frustum). 这是因为我们考虑的是照相机, 它的视野范围可以看作是一个棱锥, 其中顶点在光心. 成像就是把棱锥进行了截断, 因此得到了截锥. 理论上这个截锥的深度(沿着Z轴)应该是无穷远的, 因为相机可以看到无穷远点. 但在实际应用中, 我们会设置一个范围 $[near, far]$, 并且只考虑深度位于该范围的物体.
- 正交空间: 把观测空间通过正交变换得到的空间, 这个空间里物体的距离(深度)是与观测空间一样的, 但是物体的大小和形状是不一样的. 具有近大远小的特点. 从物理上看, 就是把截锥映射成了一个长方体, 把截锥的底面变成与顶面大小一样. 这种变换下, 靠近底面的物体会被压缩, 也就是较远的物体会变小. 最终形成了近大远小的效果.
- 标准设备坐标(NDC, Normalized Device Coordinates): 这个坐标系是一个标准化的坐标系, 其中物体的坐标范围是 $[-1, 1]$, 这个坐标系是为了方便计算机进行图形渲染而设计的. 在 3DGS 中, 我们只考虑 $z > 0$的场景, 因此这里得到的 NDC 实际范围是 $x \in [-1, 1], y \in [-1, 1], z \in [0, 1]$.

透视投影机矩阵就是把观测空间中的物体坐标变换到标准设备坐标系中的矩阵. 这个矩阵本身并不涉及从三维投影到二维的过程.

变量定义

这里我们定义六个变量, 代表了整个场景的边界

- $near$ 简称为 n, 指的是截锥的顶面的 z 轴坐标
- far 简称为 f, 指的是截锥的底面的 z 轴坐标
- $left$ 简称为 l, 指的是截锥的顶面的左边界的 x 轴坐标
- $right$ 简称为 r, 指的是截锥的顶面的右边界 x 轴坐标
- top 简称为 t, 指的是截锥的顶面的上边界的 y 轴坐标
- $bottom$ 简称为 b, 指的是截锥的顶面的下边界的 y 轴坐标

根据以上的定义, 正交空间下的长方体的边界就是 $x \in [l, r], y \in [b, t], z \in [n, f]$. 要注意的是, y 轴是向下的. 为了保证 $top > bottom$, 所以在上图里,下方是 top, 上方是 $bottom$.

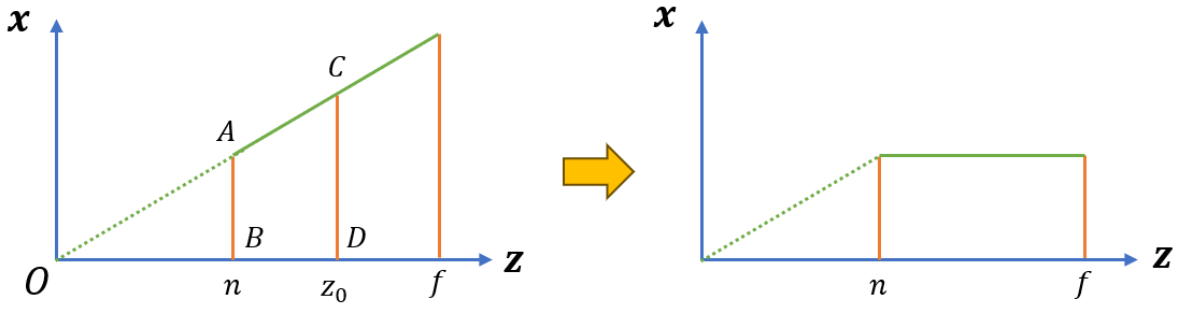
观测空间->正交空间

从观测空间变换到正交空间下, 使用的是一个压缩矩阵, 即截锥变成了长方体. 这个矩阵的形式如下

$$M_{orth} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{n+f}{n} & -f \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix}$$

推导过程

首先关注对 x, y 的变换, 如下图所示.



这里用 x 的压缩作为举例. 当 $z = n$ 时, 不需要压缩, 当 $z = f$ 时, 需要最大程度的压缩. 因此可以看出对 x 的压缩程度是与 z 相关的, 而非一个固定的常数. 这一点体现在压缩矩阵 M_{orth} 中就是矩阵的第一行为 $[1, 0, 0, 0]$. 同理, 对 y 的压缩也是与 z 相关的, 体现在矩阵的第二行为 $[0, 1, 0, 0]$.

$$M_{orth} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ . & . & . & . \\ . & . & . & . \end{bmatrix}$$

那么为了实现对 x, y 的压缩, 就要在齐次坐标的最后一维上做文章. 因为齐次坐标最终的变化是 $[x, y, z, w] \to [x/w, y/w, z/w, 1]$. 只要让 w 与 z 相关, 就可以实现对 x, y 的压缩. 因此矩阵的第四行要与 z 相关. 从上面的图片中可以发现, ΔOAB 与 ΔOCD 是相似的, 因此可以得到 $AB = \frac{n}{z_0} CD$. 这里的 AB 代表着最终的 x 的坐标, CD 代表着观测空间中的 x 的坐标. 所以只需要对坐标乘以 n/z 即可. 这一点体现在矩阵的第四行为 $[0, 0, 1/n, 0]$.

$$M_{orth} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ . & . & . & . \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix}$$

此时需要确定对 z 的变换. 从正交变换的定义可以看出, 对 z 的变换是与 x, y 无关的, 而且变换前后的 z 是一样的. 因此可以确定矩阵的第三行为 $[0, 0, M, N]$.

$$M_{orth} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & M & N \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ Mz + N \\ z/n \end{bmatrix} \to \begin{bmatrix} \frac{xn}{z} \\ \frac{yn}{z} \\ \frac{(Mz+N)n}{z} \\ 1 \end{bmatrix}$$

当 $z = n$ 时, $Mn + N = n$, 当 $z = f$ 时, $Mf + N = f$, 因此可以得到 $M = \frac{n+f}{n}, N = -f$. 最终的矩阵为

$$M_{orth} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{n+f}{n} & -f \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix}$$

正交空间->标准设备坐标

这个变换本质上就是一个各向异性的缩放+平移, 把一个长方体变成了另一个长方体. 首先看平移变换. 这个变换把x, y变换到了以原点为中心, 把z变换到了以$(f - n)/2$为中心.

$$\begin{cases} x \in [l, r] \\ y \in [b, t] \\ z \in [n, f] \end{cases} \Longrightarrow \begin{cases} x \in [-\frac{r-l}{2}, \frac{r-l}{2}] \\ y \in [-\frac{t-b}{2}, \frac{t-b}{2}] \\ z \in [0, f-n] \end{cases}$$

该平移变换对应的矩阵为

$$M_{trans} = \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

接下来是缩放变换. 这个变换是一个各向异性的缩放, 使得x, y的范围变成了$[-1, 1]$, z 的范围变成了$[0, 1]$.

$$M_{scale} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{1}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

透视投影矩阵

把以上三个变换组合, 就得到了最终的结果

$$M_{persp} = M_{scale}M_{trans}M_{orth} = \begin{bmatrix} \frac{2}{r-l} & 0 & -\frac{r+l}{n(r-l)} & 0 \\ 0 & \frac{2}{t-b} & -\frac{t+b}{n(t-b)} & 0 \\ 0 & 0 & \frac{f}{n(f-n)} & -\frac{f}{f-n} \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix} \Longrightarrow \begin{bmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f}{f-n} & -\frac{fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

在3DGS的原代码中, 这个矩阵是通过以下的代码生成的

```
P = torch.zeros(4, 4)
z_sign = 1.0
P[0, 0] = 2.0 * znear / (right - left)
P[1, 1] = 2.0 * znear / (top - bottom)
P[0, 2] = (right + left) / (right - left)
P[1, 2] = (top + bottom) / (top - bottom)
P[3, 2] = z_sign
P[2, 2] = z_sign * zfar / (zfar - znear)
P[2, 3] = -(zfar * znear) / (zfar - znear)
```

其对应的矩阵为

$$M_{persp} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f}{f-n} & -\frac{fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

与正常推导的相比, 第三列的前两个数字差了一个负号. 暂时不知道是我推导错了, 还是源代码写错了. 这篇文章的推导结果与我相同, 所以我认为可能是源代码写错了. 但是这个负号对最终的结果并没有影响, 因为 $r = -l$, 所以这两项其实都是0.

雅可比矩阵计算

在把三维高斯椭球投影到二维时, 需要计算投影变换的雅可比矩阵. 其代码如下

```
float3 t = transformPoint4x3(mean, viewmatrix); // 高斯均值在相机坐标系下的坐标

const float limx = 1.3f * tan_fovx;
const float limy = 1.3f * tan_fovy;
const float txtz = t.x / t.z;
const float tytz = t.y / t.z;
t.x = min(limx, max(-limx, txtz)) * t.z;
t.y = min(limy, max(-limy, tytz)) * t.z;

glm::mat3 J = glm::mat3(
focal_x / t.z, 0.0f, -(focal_x * t.x) / (t.z * t.z),
0.0f, focal_y / t.z, -(focal_y * t.y) / (t.z * t.z),
0, 0, 0);
```

投影变换

这里的"投影变换"与上一节的 `观测空间->正交空间` 中所使用的压缩矩阵 M_{orth} 是非常类似的. 这里的投影变换是将空间点变换到 `Ray Space` 下, 而 M_{orth} 则是变换到正交空间下. 这两个空间有着相似的性质:从光心发出的射线会变为平行线. 也就是把视锥体压缩成了长方体. 但两者的区别在于 M_{orth} 是作用于空间的指定区域内的($x \in [l, r], y \in [b, t], z \in [n, f]$), 而变换到 `Ray Space` 是对所有区域都可以. 另外, 变换到 `Ray Space` 下并不会保证深度不变(即z在变换前后一样). 关于 `Ray Space` 的更详细解释可以见 EWA Splatting(2002).
从观测空间变换到 `Ray Space` 对应的变化为

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Longrightarrow \begin{bmatrix} x/z \\ y/z \\ \sqrt{x^2+y^2+z^2} \end{bmatrix}$$

相机模型

在三维视觉中, 我们使用相机的内参 K 将三维点从相机坐标系变换到图像坐标系下, 然后再除以第三维 z 得到像素坐标.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad K \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} xf_x + zc_x \\ yf_y + zc_y \\ z \end{bmatrix} \Longrightarrow \begin{bmatrix} xf_x/z + c_x \\ yf_y/z + c_y \\ 1 \end{bmatrix}$$

上一节的投影变换是作用于图像坐标系的, 也就是说相机坐标系下的三维点要先通过内参变换到图像坐标系 $[xf_x + c_x, yf_y + c_y, z]^T$, 然后才能使用投影变换到 `Ray Space` 下.

在3DGS中, 像素坐标的原点位于图像的中心, 而不是三维视觉中常用的左上角. 所以此时的 $c_x = c_y = 0$. 从相机坐标系到图像坐标系的变换为

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Longrightarrow \begin{bmatrix} xf_x \\ yf_y \\ z \end{bmatrix}$$

将以上两种变换组合起来, 就得到了 3DGS 代码中的变换, 即

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Longrightarrow \begin{bmatrix} xf_x/z \\ yf_y/z \\ \sqrt{(xf_x)^2 + (yf_y)^2 + z^2} \end{bmatrix}$$

假设输入的是 t, 变换之后的结果为 $\phi(t)$, 那么计算 雅可比矩阵

$$\mathbf{J} = \frac{\partial \phi(t)}{\partial t} = \begin{bmatrix} \frac{\partial \phi_x}{\partial x} & \frac{\partial \phi_x}{\partial y} & \frac{\partial \phi_x}{\partial z} \\ \frac{\partial \phi_y}{\partial x} & \frac{\partial \phi_y}{\partial y} & \frac{\partial \phi_y}{\partial z} \\ \frac{\partial \phi_z}{\partial x} & \frac{\partial \phi_z}{\partial y} & \frac{\partial \phi_z}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{f_x}{z} & 0 & -\frac{xf_x}{z^2} \\ 0 & \frac{f_y}{z} & -\frac{yf_y}{z^2} \\ {xf_x}^2 l & {yf_y}^2 l & zl \end{bmatrix} \quad l = \frac{1}{\sqrt{(xf_x)^2 + (yf_y)^2 + z^2}}$$

2D协方差矩阵计算

在把三维高斯椭球投影到二维后, 得到了2D的协方差矩阵, 其可以看作是一个二维的椭球. 这里把它进行了近似, 使得它变成了一个圆. 这个元的半径是椭球的长轴. 其计算方法是:得到2D协方差矩阵后, 计算该矩阵的最大特征值, 然后对它开根号, 即得到了椭球的长轴. 但同时为了保证椭球在 3δ 区间内, 所以还需要乘以一个系数 3. 具体的代码如下

```
// Compute 2D screen-space covariance matrix
float3 cov = computeCov2D(p_orig, focal_x, focal_y, tan_fovx, tan_fovy, cov3D, viewmatrix);

// Invert covariance (EWA algorithm)
float det = (cov.x * cov.z - cov.y * cov.y);
if (det == 0.0f)
  return;

// Compute extent in screen space (by finding eigenvalues of
// 2D covariance matrix). Use extent to compute a bounding rectangle
// of screen-space tiles that this Gaussian overlaps with. Quit if
// rectangle covers 0 tiles.
float mid = 0.5f * (cov.x + cov.z);
float lambda1 = mid + sqrt(max(0.1f, mid * mid - det));
float lambda2 = mid - sqrt(max(0.1f, mid * mid - det));
float my_radius = ceil(3.f * sqrt(max(lambda1, lambda2)));
```

其背后的数学原理如下.
对于一个二维的协方差矩阵A, 我们想要计算它的特征值, 就是要求解 $Ax = \lambda x$, 其中 λ 是特征值, x 是特征向量. 其特征值的解法是求解 $det(A - \lambda I) = 0$, 其中 I 是单位矩阵.

$$A = \begin{bmatrix} x & y \\ y & z \end{bmatrix} \quad A - \lambda I = \begin{bmatrix} x-\lambda & y \\ y & z-\lambda \end{bmatrix}$$

因此有

$$det(A-\lambda I) = (x-\lambda)(z-\lambda) - y^2 = \lambda^2 - (x+z)\lambda + xz - y^2$$

$$\lambda_1 = \frac{x+z}{2} + \sqrt{\left(\frac{x+z}{2}\right)^2 - (xz-y^2)}$$

$$\lambda_2 = \frac{x+z}{2} - \sqrt{\left(\frac{x+z}{2}\right)^2 - (xz-y^2)}$$

而且值得注意的是, $xy - y^2$ 正是原本的协方差矩阵的行列式.

```
float det_inv = 1.f / det;
float3 conic = { cov.z * det_inv, -cov.y * det_inv, cov.x * det_inv };
```

两行代码则是计算了协方差矩阵的逆矩阵. 这是因为协方差矩阵是一个二维矩阵, 对于一个二维矩阵, 它的逆矩阵可以表示为

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad A^{-1} = \frac{1}{det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

GLM 库的矩阵乘法

使用 GLM 库进行矩阵乘法时, 要把乘法反过来写. 比如我们要计算 $C = A * B$, 在GLM中就要写成 $C = B * A$. 这里具体的缘故我暂时无法理解, 但应该与以下两个因素相关:

- GLM 是一个列主序的库, 也就是说它的矩阵是列向量的集合, 而不是行向量的集合.
- GLM 在进行矩阵乘法的时候, 会把左侧的矩阵看作列向量, 右侧的矩阵看作行向量. 这恰好与我们的习惯相反.

接下来举例子来理解这个问题. 其中会使用 $\times$ 表示 GLM 的矩阵乘法, $\cdot$ 表示普通的矩阵乘法(Eigen, Pytorch).
也就是说 $A \times B = B \cdot A$

例子1: 计算高斯在三维空间中的变换

```
glm::mat3 M = S * R;
// Compute 3D world covariance matrix Sigma
glm::mat3 Sigma = glm::transpose(M) * M;
```

上面这段代码转换为可以理解的数学公式为

$$M = S \times R = R \cdot S$$

$$\Sigma = M^T \times M = M \cdot M^T = (R \cdot S) \cdot (R \cdot S)^T = R \cdot S \cdot S^T \cdot R^T$$

这个公式与 3DGS 中的公式(6) 是一致的.

例子2: 计算高斯在二维上的投影

```
// viewmatrix 本身存储的是 T_cw.transpose(), 这里是取出了它的前三行三列, 即 R_cw.transpose()
// 然后再进行了转置, 得到了 R_cw
glm::mat3 W = glm::mat3(
viewmatrix[0], viewmatrix[4], viewmatrix[8],
viewmatrix[1], viewmatrix[5], viewmatrix[9],
viewmatrix[2], viewmatrix[6], viewmatrix[10]);

glm::mat3 T = W * J;

glm::mat3 Vrk = glm::mat3(
cov3D[0], cov3D[1], cov3D[2],
```

```
cov3D[1], cov3D[3], cov3D[4],
cov3D[2], cov3D[4], cov3D[5]);

glm::mat3 cov = glm::transpose(T) * glm::transpose(Vrk) * T;
```

这段代码中, `Vrk`是对称的, 因此是否转置并不影响. 把以上代码转换为数学公式为

$$
\begin{aligned}
T &= W \times J = J \cdot W \\
cov &= T^T \times Vrk \times T \\
&= T \cdot Vrk \cdot T^T \\
&= J \cdot W \cdot Vrk \cdot W^T \cdot J^T
\end{aligned}
$$

这个公式与 3DGS 中的公式(5) 是一致的.

反向传播

矩阵乘法与梯度反向传播的规律

这是在写完整个反向传播过程后, 我自己总结的一个公式. 我觉得这个公式(规律)应该有个名字, 但我暂时没有找到.
对于矩阵乘法 $Y = AX$ 来说, 假设已经知道了 Loss 对 Y 的梯度 $\frac{\partial Loss}{\partial Y}$, 那么 Loss 对 X, A 的梯度是

$$
\begin{aligned}
\frac{\partial Loss}{\partial X} &= A^T \frac{\partial Loss}{\partial Y} \\
\frac{\partial Loss}{\partial A} &= \frac{\partial Loss}{\partial Y} X^T
\end{aligned}
$$

这里的 $\partial Loss / \partial Y$ 是一个矩阵, 其中的每一个元素都是Loss对 Y中该元素的梯度. 举个例子, 其中 Y 是一个 2×4的矩阵, A 是一个 2×3的矩阵, X 是一个 3×4的矩阵.

$$
\frac{\partial L}{\partial Y} = \begin{bmatrix} \frac{\partial L}{\partial y_{11}} & \frac{\partial L}{\partial y_{12}} & \frac{\partial L}{\partial y_{13}} & \frac{\partial L}{\partial y_{14}} \\ \frac{\partial L}{\partial y_{21}} & \frac{\partial L}{\partial y_{22}} & \frac{\partial L}{\partial y_{23}} & \frac{\partial L}{\partial y_{24}} \end{bmatrix} \quad A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \quad X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{bmatrix}
$$

根据以上规律, 可以得到

$$
\frac{\partial L}{\partial X} = A^T \frac{\partial L}{\partial Y} = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial y_{11}} & \frac{\partial L}{\partial y_{12}} & \frac{\partial L}{\partial y_{13}} & \frac{\partial L}{\partial y_{14}} \\ \frac{\partial L}{\partial y_{21}} & \frac{\partial L}{\partial y_{22}} & \frac{\partial L}{\partial y_{23}} & \frac{\partial L}{\partial y_{24}} \end{bmatrix}
$$

$$
\frac{\partial L}{\partial A} = \frac{\partial L}{\partial Y} X^T = \begin{bmatrix} \frac{\partial L}{\partial y_{11}} & \frac{\partial L}{\partial y_{12}} & \frac{\partial L}{\partial y_{13}} & \frac{\partial L}{\partial y_{14}} \\ \frac{\partial L}{\partial y_{21}} & \frac{\partial L}{\partial y_{22}} & \frac{\partial L}{\partial y_{23}} & \frac{\partial L}{\partial y_{24}} \end{bmatrix} \begin{bmatrix} x_{11} & x_{21} & x_{31} \\ x_{12} & x_{22} & x_{32} \\ x_{13} & x_{23} & x_{33} \\ x_{14} & x_{24} & x_{34} \end{bmatrix}
$$

PyTorch 反向传播基础

在Pytorch的反向传播中, 会使用到 `torch.autograd.Function` 类, 在这个类中我们需要自定义 `forward` 与 `backward` 函数. 一个简单的例子如下

```
class MyFunction(torch.autograd.Function):
    @staticmethod
    def forward(ctx, input1, input2, input3):
        ctx.save_for_backward(input1, input2, input3)
        # 一系列各种操作(省略)
        return output1, output2

    @staticmethod
    def backward(ctx, grad_output1, grad_output2):
        input1, input2, input3 = ctx.saved_tensors
        # 一系列各种操作(省略)
        return grad_input1, grad_input2, grad_input3
```

从这个例子中可以看到, `forward` 函数的输入是 `input1, input2, input3`, 输出是 `output1, output2`. 那么在 `backward` 函数中, 输入的就应该是最终的Loss对 `output1, output2` 的梯度, 即 `grad_output1, grad_output2`. `backward` 函数的输出就是最终的Loss对 `input1, input2, input3` 的梯度, 即 `grad_input1, grad_input2, grad_input3`. 也就是说, `forward` 函数的输出的数量应该与 `backward` 函数的输入的数量相同, 反之亦然.

接下来详细解释3DGS中反向传播的计算方法. `forward` 的输出是每个像素的颜色, 则根据以上定义可知, `backward` 的输入是最终的Loss对每个像素的颜色的梯度.

α-Blending 公式

在3DGS中, 会使用到 α-Blending 公式, 也就是NeRF中的体渲染公式, 其公式如下

$$C_f = C_0\alpha_0 + C_1\alpha_1(1-\alpha_0) + C_2\alpha_2(1-\alpha_0)(1-\alpha_1) + \cdots + C_n\alpha_n\prod_{i=0}^{n-1}(1-\alpha_i)$$

其中 C_f 是每个像素的最终颜色, C_i 是每个高斯的颜色, α_i 是每个高斯的透明度.
通过这个公式, 可以得到最终的颜色相对于每个高斯的颜色的梯度计算公式

$$\frac{\partial C_f}{\partial C_i} = \alpha_i\prod_{j=0}^{i-1}(1-\alpha_j)$$

最终颜色相对于透明度的计算则会更复杂一些

$$\begin{aligned}
\frac{\partial C_f}{\partial \alpha_n} &= C_n\prod_{i=0}^{n-1}(1-\alpha_i) \\
\frac{\partial C_f}{\partial \alpha_{n-1}} &= C_{n-1}\prod_{i=0}^{n-2}(1-\alpha_i) - C_n\alpha_n\prod_{i=0}^{n-2}(1-\alpha_i) = (C_{n-1}-C_n\alpha_n)\prod_{i=0}^{n-2}(1-\alpha_i) \\
\frac{\partial C_f}{\partial \alpha_{n-2}} &= C_{n-2}\prod_{i=0}^{n-3}(1-\alpha_i) - C_{n-1}\alpha_{n-1}\prod_{i=0}^{n-3}(1-\alpha_i) + C_n\alpha_n\alpha_{n-1}\prod_{i=0}^{n-3}(1-\alpha_i) \\
&= [C_{n-2} - C_{n-1}\alpha_{n-1} - C_n\alpha_n(1-\alpha_{n-1})]\prod_{i=0}^{n-3}(1-\alpha_i) \\
\frac{\partial C_f}{\partial \alpha_{n-3}} &= [C_{n-3} - C_{n-2}\alpha_{n-2} - C_{n-1}\alpha_{n-1}(1-\alpha_{n-2}) - C_n\alpha_n(1-\alpha_{n-1})(1-\alpha_{n-2})]\prod_{i=0}^{n-3}(1-\alpha_i)
\end{aligned}$$

Loss相对于高斯颜色的梯度

在反向传播中, 输入的是 Loss 相对于每个像素的颜色的梯度, 也就是 $\frac{\partial L}{\partial C_f}$. 我们想要计算 Loss 相对于每个高斯的颜色的梯度, 也就是 $\frac{\partial L}{\partial C_i}$. 根据链式法则, 有

$$\frac{\partial L}{\partial C_i} = \frac{\partial L}{\partial C_f}\frac{\partial C_f}{\partial C_i} = \frac{\partial L}{\partial C_f}\alpha_i\prod_{j=0}^{i-1}(1-\alpha_j)$$

这个公式体现在代码中则是如下

```
const float G = exp(power);
const float alpha = min(0.99f, con_o.w * G);
T = T / (1.f - alpha);
const float dchannel_dcolor = alpha * T;
for (int ch = 0; ch < C; ch++)
{
    const float dL_dchannel = dL_dpixel[ch];
    atomicAdd(&(dL_dcolors[global_id * C + ch]), dchannel_dcolor * dL_dchannel);
}
```

要注意的是, `T` 的初始值是 $\prod_{i=0}^{n}(1-\alpha_i)$, 所以 `T` 要首先除以 $1-\alpha$ 才能得到 $\prod_{j=0}^{i-1}(1-\alpha_j)$.

Loss相对于高斯透明度(α)的梯度

根据链式法则, 同样可以得到 Loss 相对于每个高斯的透明度的梯度.

$$\frac{\partial L}{\partial \alpha_i} = \frac{\partial L}{\partial C_f}\frac{\partial C_f}{\partial \alpha_i}$$

但由于 $\frac{\partial C_f}{\partial \alpha_i}$ 的计算比较复杂, 所以体现在代码中也较为难以理解.
可以将其进行分解, 我们先省略 $\frac{\partial L}{\partial C_f}$, 只关注 $\frac{\partial C_f}{\partial \alpha_i}$.

$$\frac{\partial C_f}{\partial \alpha_n} = A_n \prod_{i=0}^{n-1}(1-\alpha_i) \qquad A_n = C_n$$

$$\frac{\partial C_f}{\partial \alpha_{n-1}} = A_{n-1} \prod_{i=0}^{n-2}(1-\alpha_i) \qquad A_{n-1} = C_{n-1} - C_n\alpha_n$$

$$\frac{\partial C_f}{\partial \alpha_{n-2}} = A_{n-2} \prod_{i=0}^{n-3}(1-\alpha_i) \qquad A_{n-2} = C_{n-2} - C_{n-1}\alpha_{n-1} - C_n\alpha_n(1-\alpha_{n-1})$$

$$\frac{\partial C_f}{\partial \alpha_{n-3}} = A_{n-3} \prod_{i=0}^{n-3}(1-\alpha_i) \qquad A_{n-3} = C_{n-3} - C_{n-2}\alpha_{n-2} - C_{n-1}\alpha_{n-1}(1-\alpha_{n-2}) - C_n\alpha_n(1-\alpha_{n-1})(1-\alpha_{n-2})$$

通过观察, 可以发现相邻的两项 A_i, A_{i-1} 之间有着相似的关系. 只需要把 A_i 中除了第一项以外的部分提取出来, 然后把这一部分乘以 $(1-\alpha_i)$ 即可得到 A_{i-1} 的最后部分. 至于 A_{i-1} 的前两项, 则是固定为 C_{i-1} 和 $C_i\alpha_i$.
代码中的实现则是如下

```
const float G = exp(power);
const float alpha = min(0.99f, con_o.w * G);
T = T / (1.f - alpha);
const float dchannel_dcolor = alpha * T;
float dL_dalpha = 0.0f;
for (int ch = 0; ch < C; ch++)
{
    const float c = collected_colors[ch * BLOCK_SIZE + j];
    // Update last color (to be used in the next iteration)
    accum_rec[ch] = last_alpha * last_color[ch] + (1.f - last_alpha) * accum_rec[ch];
    last_color[ch] = c;

    const float dL_dchannel = dL_dpixel[ch];
    dL_dalpha += (c - accum_rec[ch]) * dL_dchannel;
}
dL_dalpha *= T;
// Update last alpha (to be used in the next iteration)
last_alpha = alpha;
// Account for the background
float bg_dot_dpixel = 0;
for (int i = 0; i < C; i++)
    bg_dot_dpixel += bg_color[i] * dL_dpixel[i];
dL_dalpha += (-T_final / (1.f - alpha)) * bg_dot_dpixel;

atomicAdd(&(dL_dopacity[global_id]), G * dL_dalpha);
```

上面这一段代码的核心在于 `accum_rec` 变量的计算. `(1.f - last_alpha) * accum_rec` 就是把上一次结果中的后面部分提取出来, 然后乘以 $(1-\alpha)$, 接着 `last_alpha * last_color` 就是 $C_i\alpha_i$.
可以看到代码最终计算的梯度是 `G * dL_dalpha`, 这里额外有一个 `G`, 是因为我们之前推导的过程中使用的符号 α_i 指的是**每个高斯在当前像素的透明度**, 而非**每个高斯本身的透明度**. 而我们要求梯度传导到高斯本身的透明度上. 假设高斯本身的透明度是 α_i', 则有 $\alpha_i' = G\alpha_i$. 那么在链式法则里, 公式就变成了

$$\frac{\partial L}{\partial \alpha_i'} = \frac{\partial L}{\partial \alpha_i}\frac{\partial \alpha_i}{\partial \alpha_i'} = G\frac{\partial L}{\partial \alpha_i}$$

另外, 这段代码中还考虑了背景的影响. 实际渲染过程中, 最终的颜色是两部分的叠加, 一部分是高斯的体渲染, 另一部分是背景颜色. 即

$$C_f = C_{gs} + C_{bg}\prod_{i=0}^{n}(1-\alpha_i)$$

刚才的推导过程中都省略了 C_{bg} 这一项(实际代码中, $C_{bg}=0$). 所以在计算 $\frac{\partial L}{\partial \alpha_i}$ 的时候, 实际有两项, 第一项是我们一直推导的, 第二项是 $\frac{\partial C_{bg}}{\partial \alpha_i}$.

$$\frac{\partial C_{bg}}{\partial \alpha_i} = -C_{bg}\prod_{j=0}^{i-1}(1-\alpha_j)\cdot\prod_{j=i+1}^{n}(1-\alpha_j)$$

$$\frac{\partial L}{\partial \alpha_i} = \frac{\partial L}{\partial C_f}\frac{\partial C_f}{\partial \alpha_i} = \frac{\partial L}{\partial C_f}\left(\frac{\partial C_{gs}}{\partial \alpha_i} + \frac{\partial C_{bg}}{\partial \alpha_i}\right)$$

所以, 把以上都总结起来, 就得到了最终反向传播到高斯透明度的计算公式.

$$\frac{\partial L}{\partial \alpha_i'} = \frac{\partial L}{\partial C_f}\frac{\partial C_f}{\partial \alpha_i}\frac{\partial \alpha_i}{\partial \alpha_i'} = \frac{\partial L}{\partial C_f}\left(\frac{\partial C_{gs}}{\partial \alpha_i} + \frac{\partial C_{bg}}{\partial \alpha_i}\right)\frac{\partial \alpha_i}{\partial \alpha_i'}$$

Loss相对于高斯均值(二维)的梯度

高斯的均值整体变换过程为 `世界坐标系` → `相机坐标系` → `NDC坐标系` → `像素坐标系`. 其中 `世界坐标系` → `相机坐标系` 的变换矩阵是相机外参, 在代码中表示为 `viewmatrix`.
`相机坐标系` → `NDC坐标系` 的变换矩阵是投影矩阵, 已经在本文档的开头推导过.
`NDC坐标系` → `像素坐标系` 的变换非常简单, 就是把 $x \in [-1,1]$ 映射到 $[0,W]$, $y \in [-1,1]$ 映射到 $[0,H]$. 即

$$\begin{aligned} x_{pixel} &= \frac{x_{NDC}+1}{2}W \\ y_{pixel} &= \frac{y_{NDC}+1}{2}H \end{aligned} \qquad \Longrightarrow \frac{\partial x_{pixel}}{\partial x_{NDC}} = \frac{W}{2} \quad \frac{\partial y_{pixel}}{\partial y_{NDC}} = \frac{H}{2}$$

`NDC` 坐标也可以看作是二维的, 所以这一节的推导是计算 Loss 相对于**NDC坐标下**的高斯均值的梯度.

在体渲染公式中, 并没有直接出现高斯的均值. 但是在计算每个高斯在当前像素的透明度时, 使用了高斯的均值与像素之间的距离作为权重, 即上一节中出现的 G. 其具体公式如下

$$G = \exp\left(-\frac{1}{2}\left(ad_x^2 + cd_y^2\right) - bd_xd_y\right)$$

$$d_x = x - x_{pixel} \quad d_y = y - y_{pixel}$$

$$\alpha_i = G\cdot\alpha_i'$$

其中, x, y 是当前像素的位置, x_{pixel}, y_{pixel} 是高斯的均值在像素坐标系下的位置(为了简洁省略了代表第 i 个高斯的下标). a, b, c 是高斯的协方差矩阵的**逆矩阵**的三个元素. 即

$$\mathrm{cov}_{2D}^{-1} = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

这里 G 的表达式其实就是二维高斯分布, $G = \exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$, 把相应元素带入即可得到以上的表达式.
根据这个公式即可使用链式法则计算 Loss 相对于高斯均值的梯度.

$$\frac{\partial L}{\partial x_{pixel}} = \frac{\partial L}{\partial \alpha_i}\frac{\partial \alpha_i}{\partial x_{pixel}} = \frac{\partial L}{\partial \alpha_i}\frac{\partial \alpha_i}{\partial G}\frac{\partial G}{\partial x_{pixel}} = \frac{\partial L}{\partial \alpha_i}\alpha_i'\frac{\partial G}{\partial x_{pixel}}$$

$$\frac{\partial L}{\partial y_{pixel}} = \frac{\partial L}{\partial \alpha_i}\frac{\partial \alpha_i}{\partial y_{pixel}} = \frac{\partial L}{\partial \alpha_i}\alpha_i'\frac{\partial G}{\partial y_{pixel}}$$

我们在之前已经计算好了 $\frac{\partial L}{\partial \alpha_i}$, α_i' 是高斯本身的透明度, 属于已知量. 所以现在要计算的就是 $\frac{\partial G}{\partial x_{pixel}}$, $\frac{\partial G}{\partial y_{pixel}}$. 根据 G 的表达式, 可以得到

$$\frac{\partial G}{\partial x_{pixel}} = \frac{\partial G}{\partial d_x}\frac{\partial d_x}{\partial x_{pixel}} = G\left(-ad_x - bd_y\right)$$
$$\frac{\partial G}{\partial y_{pixel}} = G\left(-bd_x - cd_y\right)$$

最后, 即可计算出 Loss 相对于高斯均值(NDC坐标)的梯度.

$$\frac{\partial L}{\partial x_{NDC}} = \frac{\partial L}{\partial G}\frac{\partial G}{\partial x_{pixel}}\frac{\partial x_{pixel}}{\partial x_{NDC}} = \frac{\partial L}{\partial \alpha_i}\alpha_i' G(-ad_x - bd_y)\frac{W}{2}$$
$$\frac{\partial L}{\partial y_{NDC}} = \frac{\partial L}{\partial G}\frac{\partial G}{\partial y_{pixel}}\frac{\partial y_{pixel}}{\partial y_{NDC}} = \frac{\partial L}{\partial \alpha_i}\alpha_i' G(-bd_x - cd_y)\frac{H}{2}$$

这个公式体现在代码中则是如下

```
const float ddelx_dx = 0.5 * W;
const float ddely_dy = 0.5 * H;
const float G = exp(power);
const float dL_dG = con_o.w * dL_dalpha;
const float gdx = G * d.x;
const float gdy = G * d.y;
const float dG_ddelx = -gdx * con_o.x - gdy * con_o.y;
const float dG_ddely = -gdy * con_o.z - gdx * con_o.y;

// Update gradients w.r.t. 2D mean position of the Gaussian
atomicAdd(&dL_dmean2D[global_id].x, dL_dG * dG_ddelx * ddelx_dx);
atomicAdd(&dL_dmean2D[global_id].y, dL_dG * dG_ddely * ddely_dy);
```

Loss相对于高斯协方差矩阵(二维)的梯度

根据上面 G 的表达式, 可以得到Loss相对于高斯协方差矩阵(a, b, c)的梯度.
高斯协方差矩阵的整体变换过程是 `Ray Space(3D)` → `二维(a,b,c)` → `二维取逆(d,e,f)`.
所以在梯度反向传播的时候, 要先传播到 (d, e, f), 然后再传播到 (a, b, c). 下面首先计算 Loss 相对于 (d, e, f) 的梯度.

$$\frac{\partial G}{\partial d} = -\frac{1}{2}Gd_x^2 \quad \frac{\partial G}{\partial e} = -Gd_xd_y \quad \frac{\partial G}{\partial f} = -\frac{1}{2}Gd_y^2$$

把该公式与链式法则结合, 即可得到

$$\frac{\partial L}{\partial d} = \frac{\partial L}{\partial G}\frac{\partial G}{\partial d} = \frac{\partial L}{\partial \alpha_i}\alpha_i' G\left(-\frac{1}{2}Gd_x^2\right)$$
$$\frac{\partial L}{\partial e} = \frac{\partial L}{\partial G}\frac{\partial G}{\partial e} = \frac{\partial L}{\partial \alpha_i}\alpha_i' G\left(-Gd_xd_y\right)$$
$$\frac{\partial L}{\partial f} = \frac{\partial L}{\partial G}\frac{\partial G}{\partial f} = \frac{\partial L}{\partial \alpha_i}\alpha_i' G\left(-\frac{1}{2}Gd_y^2\right)$$

这个公式体现在代码中则是如下

```
const float dL_dG = con_o.w * dL_dalpha;
const float gdx = G * d.x;
const float gdy = G * d.y;
// Update gradients w.r.t. 2D covariance (2x2 matrix, symmetric)
atomicAdd(&dL_dconic2D[global_id].x, -0.5f * gdx * d.x * dL_dG);    // 对d的梯度
atomicAdd(&dL_dconic2D[global_id].y, -0.5f * gdx * d.y * dL_dG);    // 对e的梯度
atomicAdd(&dL_dconic2D[global_id].w, -0.5f * gdy * d.y * dL_dG);    // 对f的梯度
```

可以发现代码中关于 e 的梯度是错误的, 额外多乘了 $1/2$. 但是在后续的代码中又对这一项额外乘了一个 2, 所以最终结果是正确的.

接下来计算 Loss 相对于 (a, b, c) 的梯度. 根据矩阵求逆的公式

$$\text{cov}_{2D} = \begin{bmatrix} a & b \\ b & c \end{bmatrix} \qquad \text{cov}_{2D}^{-1} = \frac{1}{det(\text{cov}_{2D})}\begin{bmatrix} c & -b \\ -b & a \end{bmatrix} = \begin{bmatrix} d & e \\ e & f \end{bmatrix}$$

其中, $det(\text{cov}_{2D})$ 是 cov_{2D} 的行列式, 其计算公式是 $det(\text{cov}_{2D}) = ac - b^2$. 带入逆矩阵的公式, 可以得到

$$d = \frac{c}{ac-b^2} \quad e = -\frac{b}{ac-b^2} \quad f = \frac{a}{ac-b^2}$$

然后分别计算 d, e, f 对 a, b, c 的梯度, 即可得到

$$\begin{aligned}
&\frac{\partial d}{\partial a} = -\frac{c^2}{(ac-b^2)^2} \quad \frac{\partial d}{\partial b} = \frac{2bc}{(ac-b^2)^2} \quad \frac{\partial d}{\partial c} = -\frac{b^2}{(ac-b^2)^2} \\
&\frac{\partial e}{\partial a} = \frac{bc}{(ac-b^2)^2} \quad \frac{\partial e}{\partial b} = -\frac{ac+b^2}{(ac-b^2)^2} \quad \frac{\partial e}{\partial c} = \frac{ab}{(ac-b^2)^2} \\
&\frac{\partial f}{\partial a} = -\frac{b^2}{(ac-b^2)^2} \quad \frac{\partial f}{\partial b} = \frac{2ab}{(ac-b^2)^2} \quad \frac{\partial f}{\partial c} = -\frac{a^2}{(ac-b^2)^2}
\end{aligned}$$

最终即可得到 Loss 相对于 (a, b, c) 的梯度.

$$\begin{aligned}
\frac{\partial L}{\partial a} &= \frac{\partial L}{\partial d}\frac{\partial d}{\partial a} + \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} + \frac{\partial L}{\partial f}\frac{\partial f}{\partial a} = \frac{1}{ac-b^2}\left(-c^2\frac{\partial L}{\partial d} + bc\frac{\partial L}{\partial e} - b^2\frac{\partial L}{\partial f}\right) \\
\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial d}\frac{\partial d}{\partial b} + \frac{\partial L}{\partial e}\frac{\partial e}{\partial b} + \frac{\partial L}{\partial f}\frac{\partial f}{\partial b} = \frac{1}{ac-b^2}\left(2bc\frac{\partial L}{\partial d} - (ac+b^2)\frac{\partial L}{\partial e} + 2ab\frac{\partial L}{\partial f}\right) \\
\frac{\partial L}{\partial c} &= \frac{\partial L}{\partial d}\frac{\partial d}{\partial c} + \frac{\partial L}{\partial e}\frac{\partial e}{\partial c} + \frac{\partial L}{\partial f}\frac{\partial f}{\partial c} = \frac{1}{ac-b^2}\left(-b^2\frac{\partial L}{\partial d} + ab\frac{\partial L}{\partial e} - a^2\frac{\partial L}{\partial f}\right)
\end{aligned}$$

这个公式体现在代码中则是如下

```
float a = cov2D[0][0] += 0.3f;
float b = cov2D[0][1];
float c = cov2D[1][1] += 0.3f;

float denom = a * c - b * b;    // 矩阵的行列式
float dL_da = 0, dL_db = 0, dL_dc = 0;
float denom2inv = 1.0f / ((denom * denom) + 0.0000001f);
dL_da = denom2inv * (-c * c * dL_dconic.x + 2 * b * c * dL_dconic.y + (denom - a * c) *
dL_dconic.z);
dL_dc = denom2inv * (-a * a * dL_dconic.z + 2 * a * b * dL_dconic.y + (denom - a * c) *
dL_dconic.x);
dL_db = denom2inv * 2 * (b * c * dL_dconic.x - (denom + 2 * b * b) * dL_dconic.y + a * b *
dL_dconic.z);
```

代码中的 `dL_dconic.y` 就是 $\partial L/\partial e$, 可以发现代码与公式相比, 在这一项上有额外的乘以 2. 这是因为上面在计算 $\partial L/\partial e$ 的时候, 额外多乘了 $1/2$, 所以在这里要额外乘以 2 才能抵消.

Loss相对于协方差矩阵(三维)的梯度

回想高斯协方差矩阵的变换过程: 从 `相机坐标系` → `图像坐标系` → `Ray Space` → `二维`

我们之前的推导已经把梯度推导到了二维, 现在需要把梯度推导到三维. 正向的计算过程是

$$\text{cov}_{3D} = \begin{bmatrix} \sigma_0 & \sigma_1 & \sigma_2 \\ \sigma_1 & \sigma_3 & \sigma_4 \\ \sigma_2 & \sigma_4 & \sigma_5 \end{bmatrix} \qquad \text{cov}_{2D} = J \cdot W \cdot \text{cov}_{3D} \cdot W^T \cdot J^T{}_{:2,:2} = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

在这一节中, 只推导相对于三维协方差矩阵的梯度, 不关心相对于 J, W的梯度. 所以我们令 $T = J \cdot W$

$$T = \begin{bmatrix} t_0 & t_1 & t_2 \\ t_3 & t_4 & t_5 \\ t_6 & t_7 & t_8 \end{bmatrix}$$

然后计算 $T \cdot \text{cov}_{3D} \cdot T^T$, 最终可以得到

$$\begin{aligned}
a &= \sigma_0 t_0^2 + 2\sigma_1 t_0 t_1 + 2\sigma_2 t_0 t_2 + \sigma_3 t_1^2 + 2\sigma_4 t_1 t_2 + \sigma_5 t_2^2 \\
b &= \sigma_0 t_0 t_3 + \sigma_1(t_0 t_4 + t_1 t_3) + \sigma_2(t_0 t_5 + t_2 t_3) + \sigma_3 t_1 t_4 + \sigma_4(t_1 t_5 + t_2 t_4) + \sigma_5 t_2 t_5 \\
c &= \sigma_0 t_3^2 + 2\sigma_1 t_3 t_4 + 2\sigma_2 t_3 t_5 + \sigma_3 t_4^2 + 2\sigma_4 t_4 t_5 + \sigma_5 t_5^2
\end{aligned}$$

接着计算 a, b, c 对 $\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5$ 的梯度, 即可得到

$$
\begin{aligned}
&\frac{\partial a}{\partial \sigma_0} = t_0^2 \quad \frac{\partial a}{\partial \sigma_1} = 2t_0t_1 \quad \frac{\partial a}{\partial \sigma_2} = 2t_0t_2 \quad \frac{\partial a}{\partial \sigma_3} = t_1^2 \quad \frac{\partial a}{\partial \sigma_4} = 2t_1t_2 \quad \frac{\partial a}{\partial \sigma_5} = t_2^2 \\
&\frac{\partial b}{\partial \sigma_0} = t_0t_3 \quad \frac{\partial b}{\partial \sigma_1} = t_0t_4 + t_1t_3 \quad \frac{\partial b}{\partial \sigma_2} = t_0t_5 + t_2t_3 \quad \frac{\partial b}{\partial \sigma_3} = t_1t_4 \quad \frac{\partial b}{\partial \sigma_4} = t_1t_5 + t_2t_4 \quad \frac{\partial b}{\partial \sigma_5} = t_2t_5 \\
&\frac{\partial c}{\partial \sigma_0} = t_3^2 \quad \frac{\partial c}{\partial \sigma_1} = 2t_3t_4 \quad \frac{\partial c}{\partial \sigma_2} = 2t_3t_5 \quad \frac{\partial c}{\partial \sigma_3} = t_4^2 \quad \frac{\partial c}{\partial \sigma_4} = 2t_4t_5 \quad \frac{\partial c}{\partial \sigma_5} = t_5^2
\end{aligned}
$$

最终即可得到 Loss 相对于 $\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5$ 的梯度.

$$
\begin{aligned}
\frac{\partial L}{\partial \sigma_0} &= \frac{\partial L}{\partial a}\frac{\partial a}{\partial \sigma_0} + \frac{\partial L}{\partial b}\frac{\partial b}{\partial \sigma_0} + \frac{\partial L}{\partial c}\frac{\partial c}{\partial \sigma_0} = \frac{\partial L}{\partial a}t_0^2 + \frac{\partial L}{\partial b}t_0t_3 + \frac{\partial L}{\partial c}t_3^2 \\
\frac{\partial L}{\partial \sigma_1} &= \frac{\partial L}{\partial a}2t_0t_1 + \frac{\partial L}{\partial b}(t_0t_4 + t_1t_3) + \frac{\partial L}{\partial c}2t_3t_4 \\
\frac{\partial L}{\partial \sigma_2} &= \frac{\partial L}{\partial a}2t_0t_2 + \frac{\partial L}{\partial b}(t_0t_5 + t_2t_3) + \frac{\partial L}{\partial c}2t_3t_5 \\
\frac{\partial L}{\partial \sigma_3} &= \frac{\partial L}{\partial a}t_1^2 + \frac{\partial L}{\partial b}t_1t_4 + \frac{\partial L}{\partial c}t_4^2 \\
\frac{\partial L}{\partial \sigma_4} &= \frac{\partial L}{\partial a}2t_1t_2 + \frac{\partial L}{\partial b}(t_1t_5 + t_2t_4) + \frac{\partial L}{\partial c}2t_4t_5 \\
\frac{\partial L}{\partial \sigma_5} &= \frac{\partial L}{\partial a}t_2^2 + \frac{\partial L}{\partial b}t_2t_5 + \frac{\partial L}{\partial c}t_5^2
\end{aligned}
$$

这个公式体现在代码中则是如下

```
// Gradients of loss L w.r.t. each 3D covariance matrix (Vrk) entry,
// given gradients w.r.t. 2D covariance matrix (diagonal).
// cov2D = transpose(T) * transpose(Vrk) * T;
dL_dcov[6 * idx + 0] = (T[0][0] * T[0][0] * dL_da + T[0][0] * T[1][0] * dL_db + T[1][0] *
T[1][0] * dL_dc); // s0
dL_dcov[6 * idx + 3] = (T[0][1] * T[0][1] * dL_da + T[0][1] * T[1][1] * dL_db + T[1][1] *
T[1][1] * dL_dc); // s3
dL_dcov[6 * idx + 5] = (T[0][2] * T[0][2] * dL_da + T[0][2] * T[1][2] * dL_db + T[1][2] *
T[1][2] * dL_dc); // s5

// Gradients of loss L w.r.t. each 3D covariance matrix (Vrk) entry,
// given gradients w.r.t. 2D covariance matrix (off-diagonal).
// Off-diagonal elements appear twice --> double the gradient.
// cov2D = transpose(T) * transpose(Vrk) * T;
dL_dcov[6 * idx + 1] = 2 * T[0][0] * T[0][1] * dL_da + (T[0][0] * T[1][1] + T[0][1] * T[1]
[0]) * dL_db + 2 * T[1][0] * T[1][1] * dL_dc;
dL_dcov[6 * idx + 2] = 2 * T[0][0] * T[0][2] * dL_da + (T[0][0] * T[1][2] + T[0][2] * T[1]
[0]) * dL_db + 2 * T[1][0] * T[1][2] * dL_dc;
dL_dcov[6 * idx + 4] = 2 * T[0][2] * T[0][1] * dL_da + (T[0][1] * T[1][2] + T[0][2] * T[1]
[1]) * dL_db + 2 * T[1][1] * T[1][2] * dL_dc;
```

Loss相对于高斯均值(三维)的梯度

路径1:3D协方差到2D协方差投影

高斯在从 `相机坐标系` → `图像坐标系` → `Ray Space` 时使用了一个雅可比矩阵 J, 该矩阵的具体推导在上文已经给出.

$$
J = \begin{bmatrix}
\frac{f_x}{z} & 0 & -\frac{xf_x}{z^2} \\
0 & \frac{f_y}{z} & -\frac{yf_y}{z^2} \\
xf_x{}^2l & yf_y{}^2l & zl
\end{bmatrix} \quad l = \frac{1}{\sqrt{(xf_x)^2 + (yf_y)^2 + z^2}}
$$

可以看出, J 是与高斯的均值相关的. 所以可以通过 J 来计算 Loss 相对于高斯均值的梯度. 同样使用上面的表示方法, $T = J \cdot W$, 那么有

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial T}\frac{\partial T}{\partial J}\frac{\partial J}{\partial x} \quad \frac{\partial L}{\partial y} = \frac{\partial L}{\partial T}\frac{\partial T}{\partial J}\frac{\partial J}{\partial y} \quad \frac{\partial L}{\partial z} = \frac{\partial L}{\partial T}\frac{\partial T}{\partial J}\frac{\partial J}{\partial z}$$

首先来计算 $\partial L/\partial T$. 由于二维高斯的协方差是三维高斯左上角的4个数字, 所以实际上 T 的最后一行是没用的. 因此我们只需要计算前两行的梯度.

$$\begin{aligned}
&\frac{\partial a}{t_0} = 2t_0\sigma_0 + 2t_1\sigma_1 + 2t_2\sigma_2 \quad \frac{\partial b}{t_0} = t_3\sigma_0 + t_4\sigma_1 + t_5\sigma_2 \quad \frac{\partial c}{t_0} = 0\\
&\frac{\partial a}{t_1} = 2t_0\sigma_1 + 2t_1\sigma_3 + 2t_2\sigma_4 \quad \frac{\partial b}{t_1} = t_3\sigma_1 + t_4\sigma_3 + t_5\sigma_4 \quad \frac{\partial c}{t_1} = 0\\
&\frac{\partial a}{t_2} = 2t_0\sigma_2 + 2t_1\sigma_4 + 2t_2\sigma_5 \quad \frac{\partial b}{t_2} = t_3\sigma_2 + t_4\sigma_4 + t_5\sigma_5 \quad \frac{\partial c}{t_2} = 0\\
&\frac{\partial a}{t_3} = 0 \quad \frac{\partial b}{t_3} = t_0\sigma_0 + t_1\sigma_1 + t_2\sigma_2 \quad \frac{\partial c}{t_3} = 2t_3\sigma_0 + 2t_4\sigma_1 + 2t_5\sigma_2\\
&\frac{\partial a}{t_4} = 0 \quad \frac{\partial b}{t_4} = t_0\sigma_1 + t_1\sigma_3 + t_2\sigma_4 \quad \frac{\partial c}{t_4} = 2t_3\sigma_1 + 2t_4\sigma_3 + 2t_5\sigma_4\\
&\frac{\partial a}{t_5} = 0 \quad \frac{\partial b}{t_5} = t_0\sigma_2 + t_1\sigma_4 + t_2\sigma_5 \quad \frac{\partial c}{t_5} = 2t_3\sigma_2 + 2t_4\sigma_4 + 2t_5\sigma_5
\end{aligned}$$

这一段的公式体现在代码中则是如下

```
// Gradients of loss w.r.t. upper 2x3 portion of intermediate matrix T
// cov2D = transpose(T) * transpose(Vrk) * T;
float dL_dT00 = 2 * (T[0][0] * Vrk[0][0] + T[0][1] * Vrk[0][1] + T[0][2] * Vrk[0][2]) * dL_da
+
    (T[1][0] * Vrk[0][0] + T[1][1] * Vrk[0][1] + T[1][2] * Vrk[0][2]) * dL_db;
float dL_dT01 = 2 * (T[0][0] * Vrk[1][0] + T[0][1] * Vrk[1][1] + T[0][2] * Vrk[1][2]) * dL_da
+
    (T[1][0] * Vrk[1][0] + T[1][1] * Vrk[1][1] + T[1][2] * Vrk[1][2]) * dL_db;
float dL_dT02 = 2 * (T[0][0] * Vrk[2][0] + T[0][1] * Vrk[2][1] + T[0][2] * Vrk[2][2]) * dL_da
+
    (T[1][0] * Vrk[2][0] + T[1][1] * Vrk[2][1] + T[1][2] * Vrk[2][2]) * dL_db;
float dL_dT10 = 2 * (T[1][0] * Vrk[0][0] + T[1][1] * Vrk[0][1] + T[1][2] * Vrk[0][2]) * dL_dc
+
    (T[0][0] * Vrk[0][0] + T[0][1] * Vrk[0][1] + T[0][2] * Vrk[0][2]) * dL_db;
float dL_dT11 = 2 * (T[1][0] * Vrk[1][0] + T[1][1] * Vrk[1][1] + T[1][2] * Vrk[1][2]) * dL_dc
+
    (T[0][0] * Vrk[1][0] + T[0][1] * Vrk[1][1] + T[0][2] * Vrk[1][2]) * dL_db;
float dL_dT12 = 2 * (T[1][0] * Vrk[2][0] + T[1][1] * Vrk[2][1] + T[1][2] * Vrk[2][2]) * dL_dc
+
    (T[0][0] * Vrk[2][0] + T[0][1] * Vrk[2][1] + T[0][2] * Vrk[2][2]) * dL_db;
```

然后计算 $\partial T/\partial J$. 同样, 这里的 J 我们仍然只关心前两行

$$J = \begin{bmatrix} j_0 & j_1 & j_2 \\ j_3 & j_4 & j_5 \\ \cdot & \cdot & \cdot \end{bmatrix} \qquad W = \begin{bmatrix} w_0 & w_1 & w_2 \\ w_3 & w_4 & w_5 \\ w_6 & w_7 & w_8 \end{bmatrix}$$

得到的梯度表达式为

$$\begin{aligned}
&\frac{\partial t_0}{\partial j_0} = w_0 \quad \frac{\partial t_1}{\partial j_0} = w_1 \quad \frac{\partial t_2}{\partial j_0} = w_2 \quad \frac{\partial t_3}{\partial j_0} = 0 \quad \frac{\partial t_4}{\partial j_0} = 0 \quad \frac{\partial t_5}{\partial j_0} = 0\\
&\frac{\partial t_0}{\partial j_1} = w_3 \quad \frac{\partial t_1}{\partial j_1} = w_4 \quad \frac{\partial t_2}{\partial j_1} = w_5 \quad \frac{\partial t_3}{\partial j_1} = 0 \quad \frac{\partial t_4}{\partial j_1} = 0 \quad \frac{\partial t_5}{\partial j_1} = 0\\
&\frac{\partial t_0}{\partial j_2} = w_6 \quad \frac{\partial t_1}{\partial j_2} = w_7 \quad \frac{\partial t_2}{\partial j_2} = w_8 \quad \frac{\partial t_3}{\partial j_2} = 0 \quad \frac{\partial t_4}{\partial j_2} = 0 \quad \frac{\partial t_5}{\partial j_2} = 0\\
&\frac{\partial t_0}{\partial j_3} = 0 \quad \frac{\partial t_1}{\partial j_3} = 0 \quad \frac{\partial t_2}{\partial j_3} = 0 \quad \frac{\partial t_3}{\partial j_3} = w_0 \quad \frac{\partial t_4}{\partial j_3} = w_1 \quad \frac{\partial t_5}{\partial j_3} = w_2\\
&\frac{\partial t_0}{\partial j_4} = 0 \quad \frac{\partial t_1}{\partial j_4} = 0 \quad \frac{\partial t_2}{\partial j_4} = 0 \quad \frac{\partial t_3}{\partial j_4} = w_3 \quad \frac{\partial t_4}{\partial j_4} = w_4 \quad \frac{\partial t_5}{\partial j_4} = w_5\\
&\frac{\partial t_0}{\partial j_5} = 0 \quad \frac{\partial t_1}{\partial j_5} = 0 \quad \frac{\partial t_2}{\partial j_5} = 0 \quad \frac{\partial t_3}{\partial j_5} = w_6 \quad \frac{\partial t_4}{\partial j_5} = w_7 \quad \frac{\partial t_5}{\partial j_5} = w_8
\end{aligned}$$

根据 J 的实际表达式, 我们知道 $j_1 = j_3 = 0$, 所以在使用链式法则的时候, 不需要计算这两个位置的梯度. 把$\partial L/\partial T$ 与 $\partial T/\partial J$ 结合, 即可得到 $\partial L/\partial J$. 其表达式为

$$
\begin{aligned}
\frac{\partial L}{\partial j_0} &= \frac{\partial L}{\partial t_0}\frac{\partial t_0}{\partial j_0} + \frac{\partial L}{\partial t_1}\frac{\partial t_1}{\partial j_0} + \frac{\partial L}{\partial t_2}\frac{\partial t_2}{\partial j_0} = \frac{\partial L}{\partial t_0}w_0 + \frac{\partial L}{\partial t_1}w_1 + \frac{\partial L}{\partial t_2}w_2 \\
\frac{\partial L}{\partial j_2} &= \frac{\partial L}{\partial t_0}w_6 + \frac{\partial L}{\partial t_1}w_7 + \frac{\partial L}{\partial t_2}w_8 \\
\frac{\partial L}{\partial j_4} &= \frac{\partial L}{\partial t_3}w_3 + \frac{\partial L}{\partial t_4}w_4 + \frac{\partial L}{\partial t_5}w_5 \\
\frac{\partial L}{\partial j_5} &= \frac{\partial L}{\partial t_3}w_6 + \frac{\partial L}{\partial t_4}w_7 + \frac{\partial L}{\partial t_5}w_8
\end{aligned}
$$

这个公式体现在代码中则是如下

```
// Gradients of loss w.r.t. upper 3x2 non-zero entries of Jacobian matrix
// T = W * J
float dL_dJ00 = W[0][0] * dL_dT00 + W[0][1] * dL_dT01 + W[0][2] * dL_dT02;
float dL_dJ02 = W[2][0] * dL_dT00 + W[2][1] * dL_dT01 + W[2][2] * dL_dT02;
float dL_dJ11 = W[1][0] * dL_dT10 + W[1][1] * dL_dT11 + W[1][2] * dL_dT12;
float dL_dJ12 = W[2][0] * dL_dT10 + W[2][1] * dL_dT11 + W[2][2] * dL_dT12;
```

根据雅可比矩阵的具体表达式, 我们可以得到

$$
j_0 = \frac{f_x}{z} \quad j_2 = -\frac{x f_x}{z^2} \quad j_4 = \frac{f_y}{z} \quad j_5 = -\frac{y f_y}{z^2}
$$

进而得到梯度的表达式

$$
\begin{aligned}
&\frac{\partial j_0}{\partial x} = 0 \quad \frac{\partial j_2}{\partial x} = -\frac{f_x}{z^2} \quad \frac{\partial j_4}{\partial x} = 0 \quad \frac{\partial j_5}{\partial x} = 0 \\
&\frac{\partial j_0}{\partial y} = 0 \quad \frac{\partial j_2}{\partial y} = 0 \quad \frac{\partial j_4}{\partial y} = 0 \quad \frac{\partial j_5}{\partial y} = -\frac{f_y}{z^2} \\
&\frac{\partial j_0}{\partial z} = -\frac{f_x}{z^2} \quad \frac{\partial j_2}{\partial z} = \frac{2x f_x}{z^3} \quad \frac{\partial j_4}{\partial z} = -\frac{f_y}{z^2} \quad \frac{\partial j_5}{\partial z} = \frac{2y f_y}{z^3}
\end{aligned}
$$

最终即可得到 Loss 相对于 x, y, z 的梯度.

$$
\begin{aligned}
\frac{\partial L}{\partial x} &= \frac{\partial L}{\partial j_0}\frac{\partial j_0}{\partial x} + \frac{\partial L}{\partial j_2}\frac{\partial j_2}{\partial x} + \frac{\partial L}{\partial j_4}\frac{\partial j_4}{\partial x} + \frac{\partial L}{\partial j_5}\frac{\partial j_5}{\partial x} = -\frac{\partial L}{\partial j_2}\frac{f_x}{z^2} \\
\frac{\partial L}{\partial y} &= -\frac{\partial L}{\partial j_5}\frac{f_y}{z^2} \\
\frac{\partial L}{\partial z} &= -\frac{\partial L}{\partial j_0}\frac{f_x}{z^2} + \frac{\partial L}{\partial j_2}\frac{2x f_x}{z^3} - \frac{\partial L}{\partial j_4}\frac{f_y}{z^2} + \frac{\partial L}{\partial j_5}\frac{2y f_y}{z^3}
\end{aligned}
$$

这个公式体现在代码中则是如下

```
float tz = 1.f / t.z;
float tz2 = tz * tz;
float tz3 = tz2 * tz;

// Gradients of loss w.r.t. transformed Gaussian mean t
float dL_dtx = x_grad_mul * -focal_x * tz2 * dL_dJ02;
float dL_dty = y_grad_mul * -focal_y * tz2 * dL_dJ12;
float dL_dtz = -focal_x * tz2 * dL_dJ00 - focal_y * tz2 * dL_dJ11 + (2 * focal_x * t.x) * tz3
* dL_dJ02 + (2 * focal_y * t.y) * tz3 * dL_dJ12;
```

这里的x, y, z 是相机坐标系下的坐标, 与高斯在世界坐标系之间的转换是通过相机旋转矩阵 R 得到的 (**注意这里与相机的平移 t 是无关的**). 所以还需要进一步把梯度传播到世界坐标系下. 假设

$$
R = \begin{bmatrix} r_0 & r_1 & r_2 \\ r_3 & r_4 & r_5 \\ r_6 & r_7 & r_8 \end{bmatrix} \qquad \begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}
$$

那么有

$$\begin{aligned}
\frac{\partial x}{\partial x_w} &= r_0 \quad \frac{\partial y}{\partial x_w} = r_1 \quad \frac{\partial z}{\partial x_w} = r_2 \\
\frac{\partial x}{\partial y_w} &= r_3 \quad \frac{\partial y}{\partial y_w} = r_4 \quad \frac{\partial z}{\partial y_w} = r_5 \\
\frac{\partial x}{\partial z_w} &= r_6 \quad \frac{\partial y}{\partial z_w} = r_7 \quad \frac{\partial z}{\partial z_w} = r_8
\end{aligned}$$

最终即可得到 Loss 相对于 x_w, y_w, z_w 的梯度.

$$\begin{aligned}
\frac{\partial L}{\partial x_w} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial x_w} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial x_w} + \frac{\partial L}{\partial z}\frac{\partial z}{\partial x_w} = \frac{\partial L}{\partial x}r_0 + \frac{\partial L}{\partial y}r_3 + \frac{\partial L}{\partial z}r_6 \\
\frac{\partial L}{\partial y_w} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial y_w} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial y_w} + \frac{\partial L}{\partial z}\frac{\partial z}{\partial y_w} = \frac{\partial L}{\partial x}r_1 + \frac{\partial L}{\partial y}r_4 + \frac{\partial L}{\partial z}r_7 \\
\frac{\partial L}{\partial z_w} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial z_w} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_w} + \frac{\partial L}{\partial z}\frac{\partial z}{\partial z_w} = \frac{\partial L}{\partial x}r_2 + \frac{\partial L}{\partial y}r_5 + \frac{\partial L}{\partial z}r_8
\end{aligned}$$

这个公式写成一个更简洁的形式, 即

$$\begin{bmatrix} \frac{\partial L}{\partial x_w} \\ \frac{\partial L}{\partial y_w} \\ \frac{\partial L}{\partial z_w} \end{bmatrix} = R^T \begin{bmatrix} \frac{\partial L}{\partial x} \\ \frac{\partial L}{\partial y} \\ \frac{\partial L}{\partial z} \end{bmatrix}$$

这个公式体现在代码中则是如下

```
// Account for transformation of mean to t
// t = transformPoint4x3(mean, view_matrix);
float3 dL_dmean = transformVec4x3Transpose({ dL_dtx, dL_dty, dL_dtz }, view_matrix);
```

路径2:3D均值到2D均值投影

高斯的均值投影过程为 世界坐标系 → 相机坐标系 → NDC坐标系 → 像素坐标系. 我们在前面的部分已经把梯度反向传播到了 NDC坐标系 下. 这一节将把梯度继续反向传播到 世界坐标系.
从世界坐标系变换到相机坐标系用的是外参 T, 为了不失一般性, 将 T 写成

$$T = \begin{bmatrix} t_0 & t_1 & t_2 & t_3 \\ t_4 & t_5 & t_6 & t_7 \\ t_8 & t_9 & t_{10} & t_{11} \\ t_{12} & t_{13} & t_{14} & t_{15} \end{bmatrix}$$

注意, 在3DGS中, T 保存的实际上是 T^T, 在进行变换的时候, 会把 T^T 再转置一次, 然后和世界坐标系下的坐标相乘. 但目前我们在严格按照源代码进行推导, 所以忽略掉"保存的是外参的转置"这个事实

根据变换公式, 可以得到

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w \end{bmatrix} = T^T \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} t_0 & t_4 & t_8 & t_{12} \\ t_1 & t_5 & t_9 & t_{13} \\ t_2 & t_6 & t_{10} & t_{14} \\ t_3 & t_7 & t_{11} & t_{15} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

$$x = \frac{x_c}{w} \quad y = \frac{y_c}{w} \quad z = \frac{z_c}{w}$$

把以上公式展开, 即可得到

$$\begin{aligned}
x &= \frac{t_0 x_w + t_4 y_w + t_8 z_w + t_{12}}{t_3 x_w + t_7 y_w + t_{11} z_w + t_{15}} \\
y &= \frac{t_1 x_w + t_5 y_w + t_9 z_w + t_{13}}{t_3 x_w + t_7 y_w + t_{11} z_w + t_{15}}
\end{aligned}$$

接着计算 x, y 对 x_w, y_w, z_w 的梯度, 即可得到

$$
\begin{aligned}
\frac{\partial x}{\partial x_w} &= \frac{t_0}{t_3x_w+t_7y_w+t_{11}z_w+t_{15}} - \frac{t_3(t_0x_w+t_4y_w+t_8z_w+t_{12})}{(t_3x_w+t_7y_w+t_{11}z_w+t_{15})^2} \\
\frac{\partial x}{\partial y_w} &= \frac{t_4}{t_3x_w+t_7y_w+t_{11}z_w+t_{15}} - \frac{t_7(t_1x_w+t_5y_w+t_9z_w+t_{13})}{(t_3x_w+t_7y_w+t_{11}z_w+t_{15})^2} \\
\frac{\partial x}{\partial z_w} &= \frac{t_8}{t_3x_w+t_7y_w+t_{11}z_w+t_{15}} - \frac{t_{11}(t_2x_w+t_6y_w+t_{10}z_w+t_{14})}{(t_3x_w+t_7y_w+t_{11}z_w+t_{15})^2} \\
\frac{\partial y}{\partial x_w} &= \frac{t_1}{t_3x_w+t_7y_w+t_{11}z_w+t_{15}} - \frac{t_3(t_1x_w+t_5y_w+t_9z_w+t_{13})}{(t_3x_w+t_7y_w+t_{11}z_w+t_{15})^2} \\
\frac{\partial y}{\partial y_w} &= \frac{t_5}{t_3x_w+t_7y_w+t_{11}z_w+t_{15}} - \frac{t_7(t_5x_w+t_5y_w+t_9z_w+t_{13})}{(t_3x_w+t_7y_w+t_{11}z_w+t_{15})^2} \\
\frac{\partial y}{\partial z_w} &= \frac{t_9}{t_3x_w+t_7y_w+t_{11}z_w+t_{15}} - \frac{t_{11}(t_2x_w+t_6y_w+t_{10}z_w+t_{14})}{(t_3x_w+t_7y_w+t_{11}z_w+t_{15})^2}
\end{aligned}
$$

以上公式比较复杂, 我们把其中相同的部分合并一下, 得到

$$
\begin{aligned}
m_w &= \frac{1}{t_3x_w+t_7y_w+t_{11}z_w+t_{15}} \\
\mathrm{mul}_x &= \frac{t_0x_w+t_4y_w+t_8z_w+t_{12}}{(t_3x_w+t_7y_w+t_{11}z_w+t_{15})^2} = (t_0x_w+t_4y_w+t_8z_w+t_{12})m_w^2 \\
\mathrm{mul}_y &= \frac{t_1x_w+t_5y_w+t_9z_w+t_{13}}{(t_3x_w+t_7y_w+t_{11}z_w+t_{15})^2} = (t_1x_w+t_5y_w+t_9z_w+t_{13})m_w^2 \\
\frac{\partial x}{\partial x_w} &= t_0m_w - t_3\mathrm{mul}_x \quad \frac{\partial x}{\partial y_w} = t_4m_w - t_7\mathrm{mul}_x \quad \frac{\partial x}{\partial z_w} = t_8m_w - t_{11}\mathrm{mul}_x \\
\frac{\partial y}{\partial x_w} &= t_1m_w - t_3\mathrm{mul}_y \quad \frac{\partial y}{\partial y_w} = t_5m_w - t_7\mathrm{mul}_y \quad \frac{\partial y}{\partial z_w} = t_9m_w - t_{11}\mathrm{mul}_y
\end{aligned}
$$

把这些公式代入链式法则, 即可得到 Loss 相对于 x_w, y_w, z_w 的梯度.

$$
\begin{aligned}
\frac{\partial L}{\partial x_w} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial x_w} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial x_w} = \frac{\partial L}{\partial x}(t_0m_w - t_3\mathrm{mul}_x) + \frac{\partial L}{\partial y}(t_1m_w - t_3\mathrm{mul}_y) \\
\frac{\partial L}{\partial y_w} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial y_w} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial y_w} = \frac{\partial L}{\partial x}(t_4m_w - t_7\mathrm{mul}_x) + \frac{\partial L}{\partial y}(t_5m_w - t_7\mathrm{mul}_y) \\
\frac{\partial L}{\partial z_w} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial z_w} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_w} = \frac{\partial L}{\partial x}(t_8m_w - t_{11}\mathrm{mul}_x) + \frac{\partial L}{\partial y}(t_9m_w - t_{11}\mathrm{mul}_y)
\end{aligned}
$$

这个公式体现在代码中则是如下

```
float3 m = means[idx];

// Taking care of gradients from the screenspace points
float4 m_hom = transformPoint4x4(m, proj);
float m_w = 1.0f / (m_hom.w + 0.0000001f);

// Compute loss gradient w.r.t. 3D means due to gradients of 2D means
// from rendering procedure
glm::vec3 dL_dmean;
float mul1 = (proj[0] * m.x + proj[4] * m.y + proj[8] * m.z + proj[12]) * m_w * m_w;
float mul2 = (proj[1] * m.x + proj[5] * m.y + proj[9] * m.z + proj[13]) * m_w * m_w;
dL_dmean.x = (proj[0] * m_w - proj[3] * mul1) * dL_dmean2D[idx].x + (proj[1] * m_w - proj[3]
* mul2) * dL_dmean2D[idx].y;
dL_dmean.y = (proj[4] * m_w - proj[7] * mul1) * dL_dmean2D[idx].x + (proj[5] * m_w - proj[7]
* mul2) * dL_dmean2D[idx].y;
dL_dmean.z = (proj[8] * m_w - proj[11] * mul1) * dL_dmean2D[idx].x + (proj[9] * m_w -
proj[11] * mul2) * dL_dmean2D[idx].y;
dL_dmeans[idx] += dL_dmean;
```

路径3:颜色计算

每个高斯投影到图像上的颜色是根据球谐系数与视线方向得到的. 其中的视线方向是相机光心到高斯均值的方向. 因此, 颜色的梯度也会传播到高斯均值上. 暂时先忽略从球谐系数到颜色的计算, 只用一个函数来表达它. 即

$$C_i = f(\mathrm{SH}_i, \mathrm{dir}_i)$$

这样我们可以得到 Loss 对于视线方向的梯度. 再次强调, 这里不关心球谐系数的计算方法, 假设已经知道了 C_i 对于 dir_i 的梯度.

$$\frac{\partial L}{\partial \mathrm{dir}_i} = \frac{\partial L}{\partial C_i}\frac{\partial C_i}{\partial \mathrm{dir}_i}$$

那么, 接下来要计算 dir 对于高斯均值的梯度. 为了简洁, 省略了下标 i, 有

$$\mathrm{dir} = \frac{[x_w, y_w, z_w] - [c_x, c_y, c_z]}{\|[x_w, y_w, z_w] - [c_x, c_y, c_z]\|_2} = \frac{[x_w - c_x, y_w - c_y, z_w - c_z]}{\|[x_w - c_x, y_w - c_y, z_w - c_z]\|_2}$$

其中c_x, c_y, c_z代表相机的光心, 这其实就是一个归一化的向量. 因此, 可以得到

$$\begin{aligned}
l &= \|[x_w - c_x, y_w - c_y, z_w - c_z]\|_2^2 = (x_w - c_x)^2 + (y_w - c_y)^2 + (z_w - c_z)^2 \\
\frac{\partial \mathrm{dir}_x}{\partial x_w} &= \frac{l - (x_w - c_x)^2}{l^{3/2}} \quad \frac{\partial \mathrm{dir}_x}{\partial y_w} = -\frac{(x_w - c_x)(y_w - c_y)}{l^{3/2}} \quad \frac{\partial \mathrm{dir}_x}{\partial z_w} = -\frac{(x_w - c_x)(z_w - c_z)}{l^{3/2}} \\
\frac{\partial \mathrm{dir}_y}{\partial x_w} &= -\frac{(x_w - c_x)(y_w - c_y)}{l^{3/2}} \quad \frac{\partial \mathrm{dir}_y}{\partial y_w} = \frac{l - (y_w - c_y)^2}{l^{3/2}} \quad \frac{\partial \mathrm{dir}_y}{\partial z_w} = -\frac{(y_w - c_y)(z_w - c_z)}{l^{3/2}} \\
\frac{\partial \mathrm{dir}_z}{\partial x_w} &= -\frac{(x_w - c_x)(z_w - c_z)}{l^{3/2}} \quad \frac{\partial \mathrm{dir}_z}{\partial y_w} = -\frac{(y_w - c_y)(z_w - c_z)}{l^{3/2}} \quad \frac{\partial \mathrm{dir}_z}{\partial z_w} = \frac{l - (z_w - c_z)^2}{l^{3/2}}
\end{aligned}$$

把这些公式代入链式法则, 即可得到 Loss 相对于 x_w, y_w, z_w 的梯度.

$$\begin{aligned}
\frac{\partial L}{\partial x_w} &= \frac{\partial L}{\partial \mathrm{dir}_x}\frac{\partial \mathrm{dir}_x}{\partial x_w} + \frac{\partial L}{\partial \mathrm{dir}_y}\frac{\partial \mathrm{dir}_y}{\partial x_w} + \frac{\partial L}{\partial \mathrm{dir}_z}\frac{\partial \mathrm{dir}_z}{\partial x_w} \\
&= \frac{\partial L}{\partial \mathrm{dir}_x}\frac{l - (x_w - c_x)^2}{l^{3/2}} - \frac{\partial L}{\partial \mathrm{dir}_y}\frac{(x_w - c_x)(y_w - c_y)}{l^{3/2}} - \frac{\partial L}{\partial \mathrm{dir}_z}\frac{(x_w - c_x)(z_w - c_z)}{l^{3/2}} \\
\frac{\partial L}{\partial y_w} &= -\frac{\partial L}{\partial \mathrm{dir}_x}\frac{(x_w - c_x)(y_w - c_y)}{l^{3/2}} + \frac{\partial L}{\partial \mathrm{dir}_y}\frac{l - (y_w - c_y)^2}{l^{3/2}} - \frac{\partial L}{\partial \mathrm{dir}_z}\frac{(y_w - c_y)(z_w - c_z)}{l^{3/2}} \\
\frac{\partial L}{\partial z_w} &= -\frac{\partial L}{\partial \mathrm{dir}_x}\frac{(x_w - c_x)(z_w - c_z)}{l^{3/2}} - \frac{\partial L}{\partial \mathrm{dir}_y}\frac{(y_w - c_y)(z_w - c_z)}{l^{3/2}} + \frac{\partial L}{\partial \mathrm{dir}_z}\frac{l - (z_w - c_z)^2}{l^{3/2}}
\end{aligned}$$

这个公式体现在代码中则是如下

```
__forceinline__ __device__ float3 dnormvdv(float3 v, float3 dv)
{
    float sum2 = v.x * v.x + v.y * v.y + v.z * v.z;
    float invsum32 = 1.0f / sqrt(sum2 * sum2 * sum2);

    float3 dnormvdv;
    dnormvdv.x = ((+sum2 - v.x * v.x) * dv.x - v.y * v.x * dv.y - v.z * v.x * dv.z) *
invsum32;
    dnormvdv.y = (-v.x * v.y * dv.x + (sum2 - v.y * v.y) * dv.y - v.z * v.y * dv.z) *
invsum32;
    dnormvdv.z = (-v.x * v.z * dv.x - v.y * v.z * dv.y + (sum2 - v.z * v.z) * dv.z) *
invsum32;
    return dnormvdv;
}
float3 dL_dmean = dnormvdv(float3{ dir_orig.x, dir_orig.y, dir_orig.z }, float3{ dL_ddir.x,
dL_ddir.y, dL_ddir.z });
dL_dmeans[idx] += glm::vec3(dL_dmean.x, dL_dmean.y, dL_dmean.z);
```

Loss 相对于高斯旋转、尺度的梯度

在3DGS中, 高斯的协方差矩阵并不是直接保存的, 而是通过旋转矩阵 R 和尺度向量 S 得到的. 详情见原文中的公式(6). 因此, 我们需要计算 Loss 对于 R, S 的梯度. 这里的前向变换为

$$\text{cov}_{3D} = \begin{bmatrix} \sigma_0 & \sigma_1 & \sigma_2 \\ \sigma_1 & \sigma_3 & \sigma_4 \\ \sigma_2 & \sigma_4 & \sigma_5 \end{bmatrix} = M * M^T = R * S * S^T * R^T$$

第一步先来看 Loss 相对于 M 的梯度.

$$M = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & m_8 \end{bmatrix}$$

$$\text{cov}_{3D} = M * M^T = \begin{bmatrix} m_0^2 + m_1^2 + m_2^2 & m_0m_3 + m_1m_4 + m_2m_5 & m_0m_6 + m_1m_7 + m_2m_8 \\ m_0m_3 + m_1m_4 + m_2m_5 & m_3^2 + m_4^2 + m_5^2 & m_3m_6 + m_4m_7 + m_5m_8 \\ m_0m_6 + m_1m_7 + m_2m_8 & m_3m_6 + m_4m_7 + m_5m_8 & m_6^2 + m_7^2 + m_8^2 \end{bmatrix}$$

由此可以得到

$$\begin{aligned} \sigma_0 &= m_0^2 + m_1^2 + m_2^2 \\ \sigma_1 &= m_0m_3 + m_1m_4 + m_2m_5 \\ \sigma_2 &= m_0m_6 + m_1m_7 + m_2m_8 \\ \sigma_3 &= m_3^2 + m_4^2 + m_5^2 \\ \sigma_4 &= m_3m_6 + m_4m_7 + m_5m_8 \\ \sigma_5 &= m_6^2 + m_7^2 + m_8^2 \end{aligned}$$

计算偏导数, 可以得到

$$\begin{aligned} \frac{\partial \sigma_0}{\partial m_0} &= 2m_0 & \frac{\partial \sigma_1}{\partial m_0} &= m_3 & \frac{\partial \sigma_2}{\partial m_0} &= m_6 \\ \frac{\partial \sigma_0}{\partial m_1} &= 2m_1 & \frac{\partial \sigma_1}{\partial m_1} &= m_4 & \frac{\partial \sigma_2}{\partial m_1} &= m_7 \\ \frac{\partial \sigma_0}{\partial m_2} &= 2m_2 & \frac{\partial \sigma_1}{\partial m_2} &= m_5 & \frac{\partial \sigma_2}{\partial m_2} &= m_8 \\ \frac{\partial \sigma_1}{\partial m_3} &= m_0 & \frac{\partial \sigma_3}{\partial m_3} &= 2m_3 & \frac{\partial \sigma_4}{\partial m_3} &= m_6 \\ \frac{\partial \sigma_1}{\partial m_4} &= m_1 & \frac{\partial \sigma_3}{\partial m_4} &= 2m_4 & \frac{\partial \sigma_4}{\partial m_4} &= m_7 \\ \frac{\partial \sigma_1}{\partial m_5} &= m_2 & \frac{\partial \sigma_3}{\partial m_5} &= 2m_5 & \frac{\partial \sigma_4}{\partial m_5} &= m_8 \\ \frac{\partial \sigma_2}{\partial m_6} &= m_0 & \frac{\partial \sigma_4}{\partial m_6} &= m_3 & \frac{\partial \sigma_5}{\partial m_6} &= 2m_6 \\ \frac{\partial \sigma_2}{\partial m_7} &= m_1 & \frac{\partial \sigma_4}{\partial m_7} &= m_4 & \frac{\partial \sigma_5}{\partial m_7} &= 2m_7 \\ \frac{\partial \sigma_2}{\partial m_8} &= m_2 & \frac{\partial \sigma_4}{\partial m_8} &= m_5 & \frac{\partial \sigma_5}{\partial m_8} &= 2m_8 \end{aligned}$$

带入链式法则, 可以得到 Loss 相对于 M 的梯度.

$$
\begin{aligned}
\frac{\partial L}{\partial m_0} &= \frac{\partial L}{\partial \sigma_0}\frac{\partial \sigma_0}{\partial m_0} + \frac{\partial L}{\partial \sigma_1}\frac{\partial \sigma_1}{\partial m_0} + \frac{\partial L}{\partial \sigma_2}\frac{\partial \sigma_2}{\partial m_0} = 2m_0\frac{\partial L}{\partial \sigma_0} + m_3\frac{\partial L}{\partial \sigma_1} + m_6\frac{\partial L}{\partial \sigma_2} \\
\frac{\partial L}{\partial m_1} &= 2m_1\frac{\partial L}{\partial \sigma_0} + m_4\frac{\partial L}{\partial \sigma_1} + m_7\frac{\partial L}{\partial \sigma_2} \\
\frac{\partial L}{\partial m_2} &= 2m_2\frac{\partial L}{\partial \sigma_0} + m_5\frac{\partial L}{\partial \sigma_1} + m_8\frac{\partial L}{\partial \sigma_2} \\
\frac{\partial L}{\partial m_3} &= m_0\frac{\partial L}{\partial \sigma_1} + 2m_3\frac{\partial L}{\partial \sigma_3} + m_6\frac{\partial L}{\partial \sigma_4} \\
\frac{\partial L}{\partial m_4} &= m_1\frac{\partial L}{\partial \sigma_1} + 2m_4\frac{\partial L}{\partial \sigma_3} + m_7\frac{\partial L}{\partial \sigma_4} \\
\frac{\partial L}{\partial m_5} &= m_2\frac{\partial L}{\partial \sigma_1} + 2m_5\frac{\partial L}{\partial \sigma_3} + m_8\frac{\partial L}{\partial \sigma_4} \\
\frac{\partial L}{\partial m_6} &= m_0\frac{\partial L}{\partial \sigma_2} + m_3\frac{\partial L}{\partial \sigma_4} + 2m_6\frac{\partial L}{\partial \sigma_5} \\
\frac{\partial L}{\partial m_7} &= m_1\frac{\partial L}{\partial \sigma_2} + m_4\frac{\partial L}{\partial \sigma_4} + 2m_7\frac{\partial L}{\partial \sigma_5} \\
\frac{\partial L}{\partial m_8} &= m_2\frac{\partial L}{\partial \sigma_2} + m_5\frac{\partial L}{\partial \sigma_4} + 2m_8\frac{\partial L}{\partial \sigma_5}
\end{aligned}
$$

把上面的公式写成矩阵形式, 可以得到

$$
\begin{bmatrix}
\frac{\partial L}{\partial m_0} & \frac{\partial L}{\partial m_1} & \frac{\partial L}{\partial m_2} \\
\frac{\partial L}{\partial m_3} & \frac{\partial L}{\partial m_4} & \frac{\partial L}{\partial m_5} \\
\frac{\partial L}{\partial m_6} & \frac{\partial L}{\partial m_7} & \frac{\partial L}{\partial m_8}
\end{bmatrix}
=
\begin{bmatrix}
\frac{2\partial L}{\sigma_0} & \frac{\partial L}{\sigma_1} & \frac{\partial L}{\sigma_2} \\
\frac{\partial L}{\sigma_1} & \frac{2\partial L}{\sigma_3} & \frac{\partial L}{\sigma_4} \\
\frac{\partial L}{\sigma_2} & \frac{\partial L}{\sigma_4} & \frac{2\partial L}{\sigma_5}
\end{bmatrix}
\begin{bmatrix}
m_0 & m_1 & m_2 \\
m_3 & m_4 & m_5 \\
m_6 & m_7 & m_8
\end{bmatrix}
$$

这个公式体现在代码中则是如下

```
// Convert per-element covariance loss gradients to matrix form
glm::mat3 dL_dSigma = glm::mat3(
  dL_dcov3D[0], 0.5f * dL_dcov3D[1], 0.5f * dL_dcov3D[2],
  0.5f * dL_dcov3D[1], dL_dcov3D[3], 0.5f * dL_dcov3D[4],
  0.5f * dL_dcov3D[2], 0.5f * dL_dcov3D[4], dL_dcov3D[5]
);

// Compute loss gradient w.r.t. matrix M
// dSigma_dM = 2 * M
glm::mat3 dL_dM = 2.0f * M * dL_dSigma;
```

接下来计算 Loss 相对于 S 的梯度. S 是由一个三维向量组成的对角矩阵, 即

$$
S = \begin{bmatrix} s_0 & 0 & 0 \\ 0 & s_1 & 0 \\ 0 & 0 & s_2 \end{bmatrix} \qquad R = \begin{bmatrix} r_0 & r_1 & r_2 \\ r_3 & r_4 & r_5 \\ r_6 & r_7 & r_8 \end{bmatrix}
$$

所以我们只需要计算 Loss 相对于对角线元素的梯度. 根据上文的矩阵乘法中的梯度计算公式, 可以得到

$$
\frac{\partial L}{\partial S} = R^T \frac{\partial L}{\partial M}
$$

把这个表达式展开, 即可得到

$$
\begin{aligned}
\frac{\partial L}{\partial s_0} &= r_0\frac{\partial L}{\partial m_0} + r_3\frac{\partial L}{\partial m_3} + r_6\frac{\partial L}{\partial m_6} \\
\frac{\partial L}{\partial s_1} &= r_1\frac{\partial L}{\partial m_1} + r_4\frac{\partial L}{\partial m_4} + r_7\frac{\partial L}{\partial m_7} \\
\frac{\partial L}{\partial s_2} &= r_2\frac{\partial L}{\partial m_2} + r_5\frac{\partial L}{\partial m_5} + r_8\frac{\partial L}{\partial m_8}
\end{aligned}
$$

这个公式体现在代码中则是如下

```
glm::mat3 Rt = glm::transpose(R);
glm::mat3 dL_dMt = glm::transpose(dL_dM);

// Gradients of loss w.r.t. scale
glm::vec3* dL_dscale = dL_dscales + idx;
dL_dscale->x = glm::dot(Rt[0], dL_dMt[0]);
dL_dscale->y = glm::dot(Rt[1], dL_dMt[1]);
dL_dscale->z = glm::dot(Rt[2], dL_dMt[2]);
```

根据同样的规律, 可以知道 Loss 相对于 R 的梯度为

$$\frac{\partial L}{\partial R} = \frac{\partial L}{\partial M} S^T = \begin{bmatrix} s_0 \frac{\partial L}{\partial m_0} & \frac{\partial L}{\partial m_1} & \frac{\partial L}{\partial m_2} \\ \frac{\partial L}{\partial m_3} & s_1 \frac{\partial L}{\partial m_4} & \frac{\partial L}{\partial m_5} \\ \frac{\partial L}{\partial m_6} & \frac{\partial L}{\partial m_7} & s_8 \frac{\partial L}{\partial m_8} \end{bmatrix}$$

旋转矩阵 R 是由四元数 $Q = [q_w, q_x, q_y, q_z]$ 得到的, 其公式为

$$R = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_x q_y - 2q_w q_z & 2q_x q_z + 2q_w q_y \\ 2q_x q_y + 2q_w q_z & 1 - 2q_x^2 - 2q_z^2 & 2q_y q_z - 2q_w q_x \\ 2q_x q_z - 2q_w q_y & 2q_y q_z + 2q_w q_x & 1 - 2q_x^2 - 2q_y^2 \end{bmatrix}$$

因此, 可以得到

$$\begin{matrix} \frac{\partial r_0}{\partial q_w} = 0 & \frac{\partial r_1}{\partial q_w} = -2q_z & \frac{\partial r_2}{\partial q_w} = 2q_y \\ \frac{\partial r_3}{\partial q_w} = 2q_z & \frac{\partial r_4}{\partial q_w} = 0 & \frac{\partial r_5}{\partial q_w} = -2q_x \\ \frac{\partial r_6}{\partial q_w} = -2q_y & \frac{\partial r_7}{\partial q_w} = 2q_x & \frac{\partial r_8}{\partial q_w} = 0 \end{matrix} \quad \Longrightarrow \quad \frac{\partial R}{\partial q_w} = \begin{bmatrix} 0 & -2q_z & 2q_y \\ 2q_z & 0 & -2q_x \\ -2q_y & 2q_x & 0 \end{bmatrix}$$

同理可以得到

$$\frac{\partial R}{\partial q_x} = \begin{bmatrix} 0 & 2q_y & 2q_z \\ 2q_y & -4q_x & -2q_w \\ 2q_z & 2q_w & -4q_x \end{bmatrix} \quad \frac{\partial R}{\partial q_y} = \begin{bmatrix} -4q_y & 2q_x & 2q_w \\ 2q_x & 0 & 2q_z \\ -2q_w & 2q_z & -4q_y \end{bmatrix} \quad \frac{\partial R}{\partial q_z} = \begin{bmatrix} -4q_z & -2q_w & 2q_x \\ 2q_w & -4q_z & 2q_y \\ 2q_x & 2q_y & 0 \end{bmatrix}$$

把这些公式代入链式法则, 即可得到 Loss 相对于 Q 的梯度.

$$\begin{aligned} \frac{\partial L}{\partial q_w} &= \frac{\partial L}{\partial R}\frac{\partial R}{\partial q_w} = -2q_z\frac{\partial L}{\partial r_1} + 2q_y\frac{\partial L}{\partial r_2} + 2q_z\frac{\partial L}{\partial r_3} - 2q_x\frac{\partial L}{\partial r_5} - 2q_y\frac{\partial L}{\partial r_6} + 2q_x\frac{\partial L}{\partial r_7} \\ &= 2q_z\left(\frac{\partial L}{\partial r_3} - \frac{\partial L}{\partial r_1}\right) + 2q_y\left(\frac{\partial L}{\partial r_2} - \frac{\partial L}{\partial r_6}\right) + 2q_x\left(\frac{\partial L}{\partial r_7} - \frac{\partial L}{\partial r_5}\right) \\ \frac{\partial L}{\partial q_x} &= \frac{\partial L}{\partial R}\frac{\partial R}{\partial q_x} = 2q_y\frac{\partial L}{\partial r_1} + 2q_z\frac{\partial L}{\partial r_2} + 2q_y\frac{\partial L}{\partial r_3} - 4q_x\frac{\partial L}{\partial r_4} - 2q_w\frac{\partial L}{\partial r_5} + 2q_z\frac{\partial L}{\partial r_6} + 2q_w\frac{\partial L}{\partial r_7} - 4q_x\frac{\partial L}{\partial r_8} \\ &= 2q_y\left(\frac{\partial L}{\partial r_1} + \frac{\partial L}{\partial r_3}\right) + 2q_z\left(\frac{\partial L}{\partial r_2} + \frac{\partial L}{\partial r_6}\right) - 4q_x\left(\frac{\partial L}{\partial r_4} + \frac{\partial L}{\partial r_8}\right) + 2q_w\left(\frac{\partial L}{\partial r_7} - \frac{\partial L}{\partial r_5}\right) \\ \frac{\partial L}{\partial q_y} &= \frac{\partial L}{\partial R}\frac{\partial R}{\partial q_y} = -4q_y\frac{\partial L}{\partial r_0} + 2q_x\frac{\partial L}{\partial r_1} + 2q_w\frac{\partial L}{\partial r_2} + 2q_x\frac{\partial L}{\partial r_3} + 2q_z\frac{\partial L}{\partial r_5} - 2q_w\frac{\partial L}{\partial r_6} + 2q_z\frac{\partial L}{\partial r_7} - 4q_y\frac{\partial L}{\partial r_8} \\ &= -4q_y\left(\frac{\partial L}{\partial r_0} + \frac{\partial L}{\partial r_8}\right) + 2q_x\left(\frac{\partial L}{\partial r_1} + \frac{\partial L}{\partial r_3}\right) + 2q_w\left(\frac{\partial L}{\partial r_2} - \frac{\partial L}{\partial r_6}\right) + 2q_z\left(\frac{\partial L}{\partial r_5} + \frac{\partial L}{\partial r_7}\right) \\ \frac{\partial L}{\partial q_z} &= \frac{\partial L}{\partial R}\frac{\partial R}{\partial q_z} = -4q_z\frac{\partial L}{\partial r_0} - 2q_w\frac{\partial L}{\partial r_1} + 2q_x\frac{\partial L}{\partial r_2} + 2q_w\frac{\partial L}{\partial r_3} - 4q_z\frac{\partial L}{\partial r_4} + 2q_y\frac{\partial L}{\partial r_5} + 2q_x\frac{\partial L}{\partial r_6} + 2q_y\frac{\partial L}{\partial r_7} \\ &= -4q_z\left(\frac{\partial L}{\partial r_0} + \frac{\partial L}{\partial r_4}\right) + 2q_w\left(\frac{\partial L}{\partial r_3} - \frac{\partial L}{\partial r_1}\right) + 2q_x\left(\frac{\partial L}{\partial r_2} + \frac{\partial L}{\partial r_6}\right) + 2q_y\left(\frac{\partial L}{\partial r_5} + \frac{\partial L}{\partial r_7}\right) \end{aligned}$$

以上公式体现在代码中则是如下

```
glm::vec4 q = rot;// / glm::length(rot);
float r = q.x;
float x = q.y;
```

```
float y = q.z;
float z = q.w;

dL_dMt[0] *= s.x;
dL_dMt[1] *= s.y;
dL_dMt[2] *= s.z;

// Gradients of loss w.r.t. normalized quaternion
glm::vec4 dL_dq;
dL_dq.x = 2 * z * (dL_dMt[0][1] - dL_dMt[1][0]) + 2 * y * (dL_dMt[2][0] - dL_dMt[0][2]) + 2 *
x * (dL_dMt[1][2] - dL_dMt[2][1]);
dL_dq.y = 2 * y * (dL_dMt[1][0] + dL_dMt[0][1]) + 2 * z * (dL_dMt[2][0] + dL_dMt[0][2]) + 2 *
r * (dL_dMt[1][2] - dL_dMt[2][1]) - 4 * x * (dL_dMt[2][2] + dL_dMt[1][1]);
dL_dq.z = 2 * x * (dL_dMt[1][0] + dL_dMt[0][1]) + 2 * r * (dL_dMt[2][0] - dL_dMt[0][2]) + 2 *
z * (dL_dMt[1][2] + dL_dMt[2][1]) - 4 * y * (dL_dMt[2][2] + dL_dMt[0][0]);
dL_dq.w = 2 * r * (dL_dMt[0][1] - dL_dMt[1][0]) + 2 * x * (dL_dMt[2][0] + dL_dMt[0][2]) + 2 *
y * (dL_dMt[1][2] + dL_dMt[2][1]) - 4 * z * (dL_dMt[1][1] + dL_dMt[0][0]);

// Gradients of loss w.r.t. unnormalized quaternion
float4* dL_drot = (float4*)(dL_drots + idx);
*dL_drot = float4{ dL_dq.x, dL_dq.y, dL_dq.z, dL_dq.w };
```

注意, 以上代码让人有些困惑的是, `x y z r` 与 `q.x q.y q.z q.w` 之间的对应关系. 这是因为一般常用的定义中, 四元数使用四个连续的内存来存储, 其中前三个位置是 `x y z` 分量, 最后一个位置是 `w` 分量. 但是在 3DGS 中, 作者使用了 `w x y z` 的顺序来存储四元数. 所以这会导致混乱. 同样, 最后的梯度中, `dL_dq.x` 对应的是 `q.w` 的梯度, `dL_dq.y` 对应的是 `q.x` 的梯度, 以此类推.

Loss相对于相机内参的梯度

这一段并非是原本的 3DGS 中包含的. 因为原始的 3DGS 假设了相机位姿, 内参都是固定的. 但为了扩展性, 可能需要优化内参. 因此这里给出Loss相对于相机内参的梯度.

路径1: 3D协方差到2D协方差投影

从3D协方差到2D协方差的投影过程中使用了雅可比矩阵J, 其中包含了相机的焦距(f_x, f_y). 因此Loss相对于雅可比的梯度可以回传到焦距上.
这里给出 J 对 f_x, f_y 的偏导数.

$$\frac{\partial j_0}{\partial f_x} = \frac{1}{z} \quad \frac{\partial j_2}{\partial f_x} = -\frac{x}{z^2} \quad \frac{\partial j_3}{\partial f_x} = 0 \quad \frac{\partial j_5}{\partial f_x} = 0$$
$$\frac{\partial j_0}{\partial f_y} = 0 \quad \frac{\partial j_2}{\partial f_y} = 0 \quad \frac{\partial j_3}{\partial f_y} = \frac{1}{z} \quad \frac{\partial j_5}{\partial f_y} = -\frac{y}{z^2}$$

通过以上公式以及链式法则, 可以得到 Loss 相对于焦距的梯度.

$$\frac{\partial L}{\partial f_x} = \frac{\partial L}{\partial J}\frac{\partial J}{\partial f_x} = \frac{\partial L}{\partial j_0}\frac{\partial j_0}{\partial f_x} + \frac{\partial L}{\partial j_2}\frac{\partial j_2}{\partial f_x} + \frac{\partial L}{\partial j_3}\frac{\partial j_3}{\partial f_x} + \frac{\partial L}{\partial j_5}\frac{\partial j_5}{\partial f_x} = \frac{1}{z}\frac{\partial L}{\partial j_0} - \frac{x}{z^2}\frac{\partial L}{\partial j_2}$$
$$\frac{\partial L}{\partial f_y} = \frac{\partial L}{\partial J}\frac{\partial J}{\partial f_y} = \frac{\partial L}{\partial j_0}\frac{\partial j_1}{\partial f_y} + \frac{\partial L}{\partial j_2}\frac{\partial j_2}{\partial f_y} + \frac{\partial L}{\partial j_3}\frac{\partial j_3}{\partial f_y} + \frac{\partial L}{\partial j_5}\frac{\partial j_5}{\partial f_y} = \frac{1}{z}\frac{\partial L}{\partial j_3} - \frac{y}{z^2}\frac{\partial L}{\partial j_5}$$

以上公式体现在代码中则是如下

```
float tz = 1.f / t.z;
float tz2 = tz * tz;
float dL_dfx = tz * dL_dJ00 - t.x * tz2 * dL_dJ02;
float dL_dfy = tz * dL_dJ11 - t.y * tz2 * dL_dJ12;
dL_dfocal[2 * idx + 0] = dL_dfx;
dL_dfocal[2 * idx + 1] = dL_dfy;
```

路径2: 3D均值到2D均值投影

在高斯均值从3D投影到2D的过程中, 均值先使用了相机位姿变换到相机坐标系下, 然后使用透视投影矩阵变换到 NDC 坐标系下. 而透视投影矩阵是与相机的所有内参相关的. 因此Loss相对于透视投影矩阵的梯度可以回传到相机内参上. 在原版的3DGS中, 相机位姿与透视投影矩阵先进行了乘法, 得到了一个矩阵 T, 然后再与3D均值相乘.
所以, 这里我们只推导 Loss 相对于 T 的梯度, 至于 T 相对于相机内参的梯度, 则是交给了PyTorch的自动微分来计算. 这一节的所有符号请参考这里.
我们已知

$$
\begin{aligned}
x &= \frac{t_0 x_w + t_4 y_w + t_8 z_w + t_{12}}{t_3 x_w + t_7 y_w + t_{11} z_w + t_{15}} \\
y &= \frac{t_1 x_w + t_5 y_w + t_9 z_w + t_{13}}{t_3 x_w + t_7 y_w + t_{11} z_w + t_{15}}
\end{aligned}
$$

那么 x, y 相对于 T 的偏导数为

$$
\begin{aligned}
m_w &= \frac{1}{t_3 x_w + t_7 y_w + t_{11} z_w + t_{15}} \\
\mathrm{mul}_x &= \frac{t_0 x_w + t_4 y_w + t_8 z_w + t_{12}}{(t_3 x_w + t_7 y_w + t_{11} z_w + t_{15})^2} = (t_0 x_w + t_4 y_w + t_8 z_w + t_{12}) m_w^2 \\
\mathrm{mul}_y &= \frac{t_1 x_w + t_5 y_w + t_9 z_w + t_{13}}{(t_3 x_w + t_7 y_w + t_{11} z_w + t_{15})^2} = (t_1 x_w + t_5 y_w + t_9 z_w + t_{13}) m_w^2 \\
\frac{\partial x}{\partial T} &= \begin{bmatrix}
\frac{\partial x}{\partial t_0} & \frac{\partial x}{\partial t_1} & \frac{\partial x}{\partial t_2} & \frac{\partial x}{\partial t_3} \\
\frac{\partial x}{\partial t_4} & \frac{\partial x}{\partial t_5} & \frac{\partial x}{\partial t_6} & \frac{\partial x}{\partial t_7} \\
\frac{\partial x}{\partial t_8} & \frac{\partial x}{\partial t_9} & \frac{\partial x}{\partial t_{10}} & \frac{\partial x}{\partial t_{11}} \\
\frac{\partial x}{\partial t_{12}} & \frac{\partial x}{\partial t_{13}} & \frac{\partial x}{\partial t_{14}} & \frac{\partial x}{\partial t_{15}}
\end{bmatrix} = \begin{bmatrix}
x_w m_w & 0 & 0 & -x_w \mathrm{mul}_x \\
y_w m_w & 0 & 0 & -y_w \mathrm{mul}_x \\
z_w m_w & 0 & 0 & -z_w \mathrm{mul}_x \\
m_w & 0 & 0 & -\mathrm{mul}_x
\end{bmatrix} \\
\frac{\partial y}{\partial T} &= \begin{bmatrix}
0 & x_w m_w & 0 & -x_w \mathrm{mul}_y \\
0 & y_w m_w & 0 & -y_w \mathrm{mul}_y \\
0 & z_w m_w & 0 & -z_w \mathrm{mul}_y \\
0 & m_w & 0 & -\mathrm{mul}_y
\end{bmatrix}
\end{aligned}
$$

把以上公式代入链式法则, 可以得到 Loss 相对于 T 的梯度.

$$
\begin{aligned}
\frac{\partial L}{\partial t_0} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial t_0} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t_0} = x_w m_w \frac{\partial L}{\partial x} \\
\frac{\partial L}{\partial t_1} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial t_1} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t_1} = x_w m_w \frac{\partial L}{\partial y} \\
\frac{\partial L}{\partial t_3} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial t_3} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t_3} = -x_w \mathrm{mul}_x \frac{\partial L}{\partial x} - x_w \mathrm{mul}_y \frac{\partial L}{\partial y} \\
\frac{\partial L}{\partial t_4} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial t_4} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t_4} = y_w m_w \frac{\partial L}{\partial x} \\
\frac{\partial L}{\partial t_5} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial t_5} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t_5} = y_w m_w \frac{\partial L}{\partial y} \\
\frac{\partial L}{\partial t_7} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial t_7} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t_7} = -y_w \mathrm{mul}_x \frac{\partial L}{\partial x} - y_w \mathrm{mul}_y \frac{\partial L}{\partial y} \\
\frac{\partial L}{\partial t_8} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial t_8} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t_8} = z_w m_w \frac{\partial L}{\partial x} \\
\frac{\partial L}{\partial t_9} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial t_9} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t_9} = z_w m_w \frac{\partial L}{\partial y} \\
\frac{\partial L}{\partial t_{11}} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial t_{11}} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t_{11}} = -z_w \mathrm{mul}_x \frac{\partial L}{\partial x} - z_w \mathrm{mul}_y \frac{\partial L}{\partial y} \\
\frac{\partial L}{\partial t_{12}} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial t_{12}} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t_{12}} = m_w \frac{\partial L}{\partial x} \\
\frac{\partial L}{\partial t_{13}} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial t_{13}} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t_{13}} = m_w \frac{\partial L}{\partial y} \\
\frac{\partial L}{\partial t_{15}} &= \frac{\partial L}{\partial x}\frac{\partial x}{\partial t_{15}} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t_{15}} = -\mathrm{mul}_x \frac{\partial L}{\partial x} - \mathrm{mul}_y \frac{\partial L}{\partial y}
\end{aligned}
$$

Loss相对于相机外参的梯度

路径1：3D均值到2D均值投影

这条路径与上面的相机内参梯度是相同的推导。只要得到了 Loss 相对于矩阵 T 的梯度, 那么从 T 反传回内参和相机外参由 PyTorch 解决, 不需要再自己写.

路径2: 3D协方差到2D协方差投影

在计算 Loss 相对于高斯均值的梯度时, 有一条路径是 3D协方差到2D协方差的投影. 这条路径的前向计算是

$$\mathrm{cov}_{2D} = T\mathrm{cov}_{3D}T^T = JW\mathrm{cov}_{3D}W^TJ^T$$

其中的 W 其实就是相机的旋转矩阵 R_{cw}. 因此梯度可以通过这条路径传递到相机外参上. 我们在之前已经推导了 Loss 相对于中间矩阵 T 的梯度, 现在只需要把梯度传导到 W 即可. 根据矩阵乘法的梯度反向传播规律, 可以知道

$$\frac{\partial L}{\partial W} = J^T\frac{\partial L}{\partial T} = \begin{bmatrix} j_{00} & j_{10} & j_{20} \\ j_{01} & j_{11} & j_{21} \\ j_{02} & j_{12} & j_{22} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial t_0} & \frac{\partial L}{\partial t_1} & \frac{\partial L}{\partial t_2} \\ \frac{\partial L}{\partial t_4} & \frac{\partial L}{\partial t_5} & \frac{\partial L}{\partial t_6} \\ 0 & 0 & 0 \end{bmatrix}$$

把以上公式展开, 可以得到

$$\begin{aligned}
\frac{\partial L}{\partial w_0} &= j_{00}\frac{\partial L}{\partial t_0} + j_{10}\frac{\partial L}{\partial t_4} & \frac{\partial L}{\partial w_1} &= j_{00}\frac{\partial L}{\partial t_1} + j_{10}\frac{\partial L}{\partial t_5} & \frac{\partial L}{\partial w_2} &= j_{00}\frac{\partial L}{\partial t_2} + j_{10}\frac{\partial L}{\partial t_6} \\
\frac{\partial L}{\partial w_3} &= j_{01}\frac{\partial L}{\partial t_0} + j_{11}\frac{\partial L}{\partial t_4} & \frac{\partial L}{\partial w_4} &= j_{01}\frac{\partial L}{\partial t_1} + j_{11}\frac{\partial L}{\partial t_5} & \frac{\partial L}{\partial w_5} &= j_{01}\frac{\partial L}{\partial t_2} + j_{11}\frac{\partial L}{\partial t_6} \\
\frac{\partial L}{\partial w_6} &= j_{02}\frac{\partial L}{\partial t_0} + j_{12}\frac{\partial L}{\partial t_4} & \frac{\partial L}{\partial w_7} &= j_{02}\frac{\partial L}{\partial t_1} + j_{12}\frac{\partial L}{\partial t_5} & \frac{\partial L}{\partial w_8} &= j_{02}\frac{\partial L}{\partial t_2} + j_{12}\frac{\partial L}{\partial t_6}
\end{aligned}$$

另外, 雅可比矩阵 J中, 很多项为0, 只有 $j_{00}, j_{02}, j_{11}, j_{12}$ 不为0. 因此上面的公式可以简化为

$$\begin{aligned}
\frac{\partial L}{\partial w_0} &= j_{00}\frac{\partial L}{\partial t_0} & \frac{\partial L}{\partial w_1} &= j_{00}\frac{\partial L}{\partial t_1} & \frac{\partial L}{\partial w_2} &= j_{00}\frac{\partial L}{\partial t_2} \\
\frac{\partial L}{\partial w_3} &= j_{11}\frac{\partial L}{\partial t_4} & \frac{\partial L}{\partial w_4} &= j_{11}\frac{\partial L}{\partial t_5} & \frac{\partial L}{\partial w_5} &= j_{11}\frac{\partial L}{\partial t_6} \\
\frac{\partial L}{\partial w_6} &= j_{02}\frac{\partial L}{\partial t_0} + j_{12}\frac{\partial L}{\partial t_4} & \frac{\partial L}{\partial w_7} &= j_{02}\frac{\partial L}{\partial t_1} + j_{12}\frac{\partial L}{\partial t_5} & \frac{\partial L}{\partial w_8} &= j_{02}\frac{\partial L}{\partial t_2} + j_{12}\frac{\partial L}{\partial t_6}
\end{aligned}$$

其对应的代码如下

```
float dL_dw0 = J[0][0] * dL_dT00;
float dL_dw1 = J[0][0] * dL_dT01;
float dL_dw2 = J[0][0] * dL_dT02;
float dL_dw3 = J[1][1] * dL_dT10;
float dL_dw4 = J[1][1] * dL_dT11;
float dL_dw5 = J[1][1] * dL_dT12;
float dL_dw6 = J[0][2] * dL_dT00 + J[1][2] * dL_dT10;
float dL_dw7 = J[0][2] * dL_dT01 + J[1][2] * dL_dT11;
float dL_dw8 = J[0][2] * dL_dT02 + J[1][2] * dL_dT12;
```

高斯的投影过程

高斯的投影过程可以分成两个方面：均值的投影以及协方差的投影。
均值的投影过程是：

1. 将均值变换到相机坐标系下，使用相机的位姿 T_{cw}.
2. 根据相机焦距以及图像分辨率, 计算透视投影矩阵 M_{persp}. 该矩阵的具体推导在这里.
3. 使用M_{persp}把相机坐标系下的均值投影到 NDC 坐标下.
4. NDC坐标的前两维即看作2D坐标, 缩放到图像的长宽即可.

协方差的投影过程是:

1. 将协方差进行旋转, 使用相机的旋转 R_{cw}. 即 $R_{cw}\Sigma R_{cw}^T$, 论文中也表达为 $W\Sigma W^T$.
2. 将均值变换到相机坐标系下，使用相机的位姿 T_{cw}.
3. 根据均值的位置, 计算投影变换的近似, 即雅可比矩阵 J.
4. 用 J 对第一步得到的协方差进行变换, 得到 $\Sigma' = JW\Sigma W^T J^T$.
5. 取 Σ' 的前两行两列, 得到2D上的协方差矩阵 Σ_{2D}.

修改相机内参对3DGS的影响

在原版的3DGS中, 默认相机的参数是仅支持 f_x, f_y的,不支持主点. 也就是说默认了主点在图像中心. 这个假设等价于 $c_x = W/2, c_y = H/2$, 其中 W, H 是图像的宽高. 但有些情况下, 主点可能会偏离图像中心. 所以需要对3DGS进行修改, 使其支持主点. 并且可以用于后续相机参数的优化.

对雅可比矩阵 J 的影响

在原始3DGS中, 雅可比矩阵是由两部分复合而来的. 第一部分是相机的内参矩阵 K, 它把一个点 P 从相机坐标系变换到图像坐标系下. 第二部分是变换到 `Ray Space` 的投影矩阵. 我们将内参矩阵写为以下形式:

$$K = \begin{bmatrix} f_x & 0 & c_x - \frac{W}{2} \\ 0 & f_y & c_y - \frac{H}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

原始的3DGS中, 假设了 $c_x = W/2, c_y = H/2$, 所以 K 就变成了对角阵. 从相机坐标系到 `Ray Space` 的变换就可以写为

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Longrightarrow \begin{bmatrix} xf_x + z(c_x - \frac{W}{2}) \\ yf_y + z(c_y - \frac{H}{2}) \\ z \end{bmatrix} \Longrightarrow \begin{bmatrix} \frac{xf_x}{z} + c_x - \frac{W}{2} \\ \frac{yf_y}{z} + c_y - \frac{H}{2} \\ \sqrt{\left(xf_x + z(c_x - \frac{W}{2})\right)^2 + \left(yf_y + z(c_y - \frac{H}{2})\right)^2 + z^2} \end{bmatrix}$$

该雅可比矩阵为

$$J = \begin{bmatrix} \frac{f_x}{z} & 0 & -\frac{xf_x}{z^2} \\ 0 & \frac{f_y}{z} & -\frac{yf_y}{z^2} \\ \cdot & \cdot & \cdot \end{bmatrix}$$

省略了第三行, 因为第三行实际上是没用的. 可以看到, 这个雅可比矩阵是和主点无关的.

对透视投影矩阵的影响

在这一节中推导了 3DGS 中透视投影矩阵的形式. 其中它只考虑了焦距, 没有考虑主点. 因此需要在透视投影矩阵中加入主点的考虑. 考虑主点后, 透视投影矩阵的形式为

$$M_{persp} = \begin{bmatrix} \frac{2f_x}{W} & 0 & \frac{2c_x}{W} - 1 & 0 \\ 0 & \frac{2f_y}{H} & \frac{2c_y}{H} - 1 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

可以看到与原始的投影矩阵相比, 前两行发生了变化, 因为主点只影响 x, y 的坐标. 更具体的, 其实只是影响了前两行的第三列. 主对角线元素并没有受影响, 因为 $2n/(r-l) = 2f_x/W$.

以下给出该矩阵的详细推导过程. 我们首先考虑最基本的内参矩阵, f_x, f_y, c_x, c_y. 内参矩阵把三维点投影到图像上, 投影点的坐标是 $u = xf_x/z + c_x, v = yf_y/z + c_y$. 所以投影矩阵的前两行应该是

$$M = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

而 $xf_x/z + c_x$ 的范围是 $[0, W]$, 为了与源代码兼容, 我们需要把他变成 $[-1, 1]$. 所以要进行归一化, 即 $2\frac{xf_x/z+c_x}{W} - 1$. 对于 $yf_y/z + c_y$ 也是同样的处理. 所以最终的投影矩阵为

$$\begin{bmatrix} \frac{2}{W} & 0 & 0 & -1 \\ 0 & \frac{2}{H} & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} \frac{2f_x}{W} & 0 & \frac{2c_x}{W} - 1 & 0 \\ 0 & \frac{2f_y}{H} & \frac{2c_y}{H} - 1 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$