SORT-FREE GAUSSIAN SPLATTING VIA WEIGHTED SUM RENDERING FREQUENT ASK QUESTIONS

1 DERIVATIVE

For the WSR rendering, the image can be rendered using

$$\mathbf{C} = \frac{\mathbf{c}_B w_B + \sum_{i=1}^{\mathcal{N}} \mathbf{c}_i \alpha_i w(d_i)}{w_B + \sum_{i=1}^{\mathcal{N}} \alpha_i w(d_i)}, \tag{1}$$

where $\mathbf{C}$ indicates the output image. $\mathbf{c}_B$ and w_B indicate the color and learnable weight of the background, respectively. d indicates the depth. $w(\cdot)$ indicates the learnable weight function.

We can calculate the derivatives of the learnable parameters w.r.t the loss $\mathcal{L}$. To save the calculation for the backward, the sum of weights w_s is saved during the forward step as

$$w_s = w_B + \sum_{i=1}^{\mathcal{N}} \alpha_i w(d_i), \tag{2}$$

where you might need to use atomic operations , e.g. `atomicAdd()` , as needed.

Then we can get the derivatives as follows,

$$\frac{\partial \mathcal{L}}{\partial \alpha_i} = w(d_i) \frac{\mathbf{c}_i - \mathbf{C}}{w_s} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{C}} \tag{3}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{c}_i} = \frac{\alpha_i w(d_i)}{w_s} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{C}} \tag{4}$$

$$\frac{\partial \mathcal{L}}{\partial w_B} = \frac{\mathbf{c}_B - \mathbf{C}}{w_s} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{C}} \tag{5}$$

$$\frac{\partial \mathcal{L}}{\partial w(d_i)} = \alpha_i \frac{\mathbf{c}_i - \mathbf{C}}{w_s} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{C}} \tag{6}$$

For the EXP-WSR, the weight is defined as

$$w(d_i) = \exp\left(-\sigma d_i^{\beta}\right), \quad i = 1, 2, \cdots, \mathcal{N}. \tag{7}$$

The derivatives can be calculated as follows

$$\frac{\partial \mathcal{L}}{\partial \sigma} = -\sum_{i=1}^{\mathcal{N}} w(d_i) d_i^{\beta} \cdot \frac{\partial \mathcal{L}}{\partial w(di)} \tag{8}$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = -\sum_{i=1}^{\mathcal{N}} w(d_i) \sigma ln(d_i) d_i^{\beta} \cdot \frac{\partial \mathcal{L}}{\partial w(di)} \tag{9}$$

For the LC-WSR, for the simplification, we re-wrote the $w(d_i)$ as

$$w(d_i) = \max(0, 1 - \sigma d_i)\, v_i, \quad i = 1, 2, \cdots, \mathcal{N}. \tag{10}$$

The derivatives can be calculated as follows

$$\frac{\partial \mathcal{L}}{\partial v_i} = \max(0, 1 - \sigma d_i) \cdot \frac{\partial \mathcal{L}}{\partial w(di)} \tag{11}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma} = \begin{cases} 0, & \text{if } 1 - \sigma d_i \leq 0 \\ -\sum_{i=1}^{\mathcal{N}} v_i d_i \cdot \frac{\partial \mathcal{L}}{\partial w(di)} & \text{otherwise} \end{cases} \tag{12}$$

2 TRICKS

Training Details
Training LC-WSR is challenging due to the ReLU-like weight function, which blocks gradient flow. To address this, we first train EXP-WSR and use its weights to initialize LC-WSR. For optimal PSNR performance, EXP-WSR itself is initialized using the weights from vanilla 3DGS.

Learning Rate Configuration
The learning rates are specified below as a reference. These values may require tuning based on your specific implementation:

- `sigma`: 0.000150
- `background weight`: 0.000010
- `beta`: 0.000001
- `weights`: 0.010000
- `opacity sh`: same as the learning rate for color sh

View-Dependent Opacity
Our implementation of view-dependent opacity closely follows the `sh_to_color` function, with a key difference: we omit the clamping step for opacity. Specifically, we extend the input data from 3 to 4 dimensions, using the additional channel to represent opacity. This approach is both simple to implement and efficient, as GPU hardware processes data in 4D (RGBA) by default. Thus, it incurs no additional computational or memory overhead.

In our code, the SH weights tensor has shape $N \times 16 \times 4$, where the first three channels correspond to RGB, and the last one is for opacity. We copy the $N \times 16 \times 3$ RGB weights directly from the official 3DGS implementation. For the opacity channel ($N \times 16 \times 1$), the first $N \times 1 \times 1$ entries are initialized using `RGB2SH(3dgs_opacity)`, while the remaining $N \times 15 \times 1$ entries are initialized to zero.

For reference, see the relevant implementation here:
`https://github.com/graphdeco-inria/gaussian-splatting/blob/54c035f7834b564019656c3e3fcc3646292f727d/utils/sh_utils.py#L114`