test: reference to eq.(**??**).

This notes focuses on the effective implementation of Eq.(6.3.10). By default, refers to the derivation.pdf that comes with it.

You said that we should interpolate quantities containing $\delta(\phi - \phi_{\boldsymbol{r}-\boldsymbol{r}'})$, e.g., ψ^I_{sb} in Eq.(6.3.9), with many "rays" prior to the code implementation. By brute force, the CPU cost for each ray-tracing is $O(N_s)$ - already implemented. For N_s triangles and M directions, the CPU cost is $O(N_s^2 \times M)$. Further plugging the interpolated quantity into the first line of Eq.(6.3.10) necessitates $O(N_s^3 \times M)$ CPU time.

Analytically, as shown in Eq.(6.3.10), the delta functions could be integrated. The last line of Eq.(6.3.10) entails $O(N_s^3)$ CPU time. In Eq.(6.3.12), all but the following quantities are already pre-computed:

$$
\begin{aligned}
&e^{-\tau(\boldsymbol{r}_n - \boldsymbol{r}_{n'})}\\
&f(\phi_{\boldsymbol{r}_n - \boldsymbol{r}_{n'}} - \phi_{\boldsymbol{r}_{n'} - \boldsymbol{r}_{n''}})\\
&f_{g^2}(\phi_{\boldsymbol{r}_n - \boldsymbol{r}_{n'}} - \phi^I)
\end{aligned}
$$

The first term: $O(N_s^3)$ CPU and $O(N_s^2)$ memory. Ray tracing in τ added the extra $O(N_s)$.

The second term: the angles inside are pre-computed.

The third term: $O(N_s^2)$ CPU and memory.

Of course, if the spatial part, i.e., calculation pertain to N_s could be accelerated, the complexities decrease accordingly.

As a first try, may I try implementing Eq.(6.3.10) by Eq.(6.3.12)?