Chapter 1

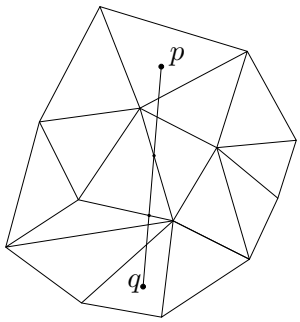
Algorithm for R between two arbitrary points in a mesh

To measure the Dirichlet energy, we need to calculate the rotation coefficient between to arbitrary points q and p that do not necessarily lie within the same tetrahedron. Since the metric and curl is different in each tet, we need to be able to efficiently determine all tets that get intersected by the straight line from q to p, and use the correct metric for each corresponding line segment. The calculation for the coefficient then works in the following way:

Algorithm 1 Rotation coefficient R between q and p

Input (q, p)
    LINESEGMENTS $\leftarrow$ *tetFinder*(q, p) //returns all tets intersected by the line $p\vec{q}$ with the line segments within them
    $R \leftarrow \mathrm{Id}$
    for each SEGMENT **in** LINESEGMENTS
        $R \leftarrow R \cdot$ *calcCoeff*(SEGMENT)
return R

The missing component here is how do we efficiently find all tetrahedra that get intersected. One possibility would be to use ray-triangle intersection and test against the whole mesh, but this is not practical, as we have have local information that we can exploit.



Mesh

Algorithm 2 Byzantine Leader-Based Epoch-Change (process p_i).

7: **State**
8: *lastts* $\leftarrow 0$: most recently started epoch
9: *nextts* $\leftarrow 0$: timestamp of the next epoch
10: *newepoch* $\leftarrow [\perp]^n$: list of NEWEPOCH messages

11: **upon event** *complain*(p_ℓ) **such that** $p_\ell = leader(lastts)$ **do**
12: **if** *nextts* $=$ *lastts* **then**
13: *nextts* $\leftarrow$ *lastts* $+ 1$
14: send message $[\text{NEWEPOCH}, nextts]$ to all $p_j \in \mathcal{P}$

15: **upon** receiving a message $[\text{NEWEPOCH}, ts]$ from p_j **such that** $ts = lastts + 1$ **do**
16: *newepoch*$[j] \leftarrow$ NEWEPOCH

17: **upon exists** *ts* **such that** $\{p_j \in \mathcal{P} |\ newepoch[j] = ts\} \in \mathcal{K}_i$ **and** *nextts* $=$ *lastts* **do**
18: *nextts* $\leftarrow$ *lastts* $+ 1$
19: send message $[\text{NEWEPOCH}, nextts]$ to all $p_j \in \mathcal{P}$

20: **upon exists** *ts* **such that** $\{p_j \in \mathcal{P} |\ newepoch[j] = ts\} \in \mathcal{Q}_i$ **and** *nextts* $>$ *lastts* **do**
21: *lastts* $\leftarrow$ *nextts*
22: *newepoch* $\leftarrow [\perp]^n$
23: **output** *startepoch*$(lastts, leader(lastts))$