The new tracking algorithm in FOAM

Niklas Nordin

April 6, 2004

and blather...

Assuming we have a situation similar to the one in Figure 1, where a particle is located at position **a** and wants to go to **b**, we want to find the position,

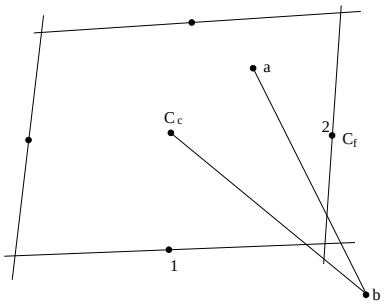

Figure 1: Particle in a cell moving from position **a** to **b**

p, at which the line $\mathbf{a} - \mathbf{b}$ intersects with face 2. Or, in other words, we want to find the smallest $\lambda_a \in [0, 1]$ which satisfy

$$(\mathbf{p} - \mathbf{C}_f) \cdot \mathbf{S}_f = 0, \quad \mathbf{p} = \mathbf{a} + \lambda_a(\mathbf{b} - \mathbf{a}) \qquad (1)$$

where $\mathbf{C}_f$ is the face centre and $\mathbf{S}_f$ is the face normal. This yields

$$\lambda_a = \frac{(\mathbf{C}_f - \mathbf{a}) \cdot \mathbf{S}_f}{(\mathbf{b} - \mathbf{a}) \cdot \mathbf{S}_f} \qquad (2)$$

Thus, we must calculate λ_a for all faces to find out which face it hits first and from Figure 1 we see that only two faces will result in a $\lambda_a \in [0, 1]$, face 1 and 2. When we've found λ_a we move it to **p**, change cell to the neighbouring one, update particle properties and continue the tracking. Had **b** been inside the cell all λ_a would have been either smaller than 0, or larger than 1.

This works very well if **a** is inside the cell, but what if it is outside. Assume we have something like in Figure 2, where the particle is at position **a**, which now is outside the cell, and wants to go to **b**. Using the algorithm above we will find that all λ_a are either smaller than 0 or larger than 1, and according to above, this is equivalent to the particle not changing cell. Hence, it will remain in the same cell, lost. Further, had position **b** been inside the cell, the algorithm would signal a face being hit and change cell, thus the particle would still be lost.

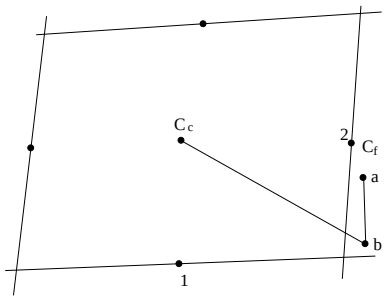

Figure 2: Particle not in a cell moving from position **a** to **b**

Instead of using **a** to find out if the partice will hit a face, we first note that if **b** is outside the cell we can track the particle from any position inside the cell and we will hit at least one face. We also note that it doesn't matter from which position we start the tracking, every path will cross the same faces. So instead of finding which face the particle will hit when moving from **a** to **b** we can, for instance, use the cell centre $\mathbf{C}_c$, and by doing that we only have to replace **a** with $\mathbf{C}_c$ in Equation (2).

$$\lambda_c = \frac{(\mathbf{C}_f - \mathbf{C}_c) \cdot \mathbf{S}_f}{(\mathbf{b} - \mathbf{C}_c) \cdot \mathbf{S}_f} \qquad (3)$$

Looking at Figure 1 again we see that using λ_c to find which face the particle will hit will still result in $\lambda_c \in [0, 1]$ for face 1 and 2, but to find out which face it will hit first we must still use λ_a, but now only on face 1 and 2. Had **b** been inside the cell all λ_c would have been either smaller than 0 or larger than 1 and it would not be necessary to calculate λ_a. So, depending on the outcome of λ_c, it is equally expensive or slightly more expensive than the original algorithm. The motivation for this will hopefully become clear when looking at Figure 2.

Instead of using λ_a, all which are either smaller than 0 or larger than 1, we now get $\lambda_c \in [0, 1]$ for face 2 and since position **a** is believed to be inside the cell it must thus also cross face 2. But when

calculating λ_a for face 2 we will get $\lambda_a < 0$, if the particle travels away from the cell and $\lambda_a > 1$ if it travels towards the cell and we move the particle to it's new position $\mathbf{p}$ using

$$\mathbf{p} = \mathbf{a} + \lambda(\mathbf{b} - \mathbf{a}), \quad \lambda = \min(1, \max(0, \lambda_a)) \quad (4)$$

Hence, if the particle is moving away from the cell, it will not move, only change cell and continue the tracking from the new cell.

The algorithm can thus be summarized as

- find all faces, F_i, where $\lambda_c \in [0, 1]$
- if number of $F_i == 0$

  move particle to it's end position

  else

  · find face $\mathcal{F}$ among F_i,

  for which λ_a is smallest

  · move particle to new position according to Equation (4)

  · set particle to belong to neighbouring cell for face $\mathcal{F}$

  end if

If the mesh is moving the only change will be in the calculation of λ, but this algorithm should still work provided that the particle is not too far outside the cell, i.e. the particle is more than one cell away. The time the particle spends in the individual cells will obviously not be correct though.

Concave cells

There is a possibility that the algorithm can end up in an infinite loop, or freeze the particle, if the cell is concave. If we consider Figure 3. When the particle moves from **a** to **b** it will hit and move to face f and change cell from 1 to 2, but since it's moving away from face g in cell 2, it will now change cell to 3 and again it's moving away from face h and will change cell to 1, where it will start all over again and we have an infinite loop...bummer.

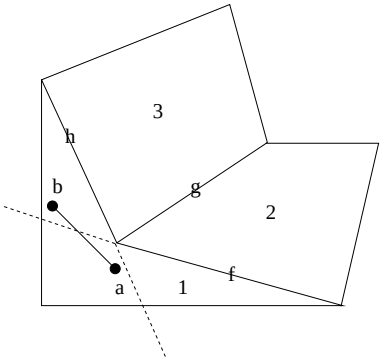

Figure 3: Particle moving in concave cell from **a** to **b**