Visualization of Advection-Diffusion in Unsteady Fluid Flow

G. K. Karch, F. Sadlo, D. Weiskopf, C.-D. Munz, T. Ertl

Appendix

This is supplementary material to the paper "Visualization of Advection-Diffusion in Unsteady Fluid Flow". We provide a pseudo code for the GPU implementation of the method presented in the paper, and point out important implementation aspects.

Input: $\overline{\phi}(t)$, polynomial order o_p

Output: $\overline{\phi}(t + \delta t)$

1: **for** each dimension **do**
2:   **for all** thread blocks in parallel **do**
3:     **for all** threads in parellel **do** {thread corresponds to cell i}
4:       $\tilde{\overline{\phi}}_i(\tau = t) = \overline{\phi_i}(t)$
5:       $v_{left} := -v(x_{i-\frac{1}{2}})$ if $v(x_{i-\frac{1}{2}}) < 0$, otherwise $v_{left} := 0$
6:       $v_{right} := v(x_{i+\frac{1}{2}})$ if $v(x_{i+\frac{1}{2}}) > 0$, otherwise $v_{right} := 0$
7:       **for** $k = 0$ **to** o_p **do** {for each reconstruction polynomial}
8:         **for** $j = 1$ **to** o_p **do** {for each coefficient except w_0^k}
9:           compute w_j^k from Eq. 14
10:         **end for**
11:         $\sigma_k = \sum_{m=0}^{o_p-1}(w_{m+1}^k \cdot \sum_{n=0}^{o_p-1}(\Sigma_{m,n} \cdot w_{n+1}^k))$
12:         $\tilde{\omega}_k = \lambda_k/(\epsilon + \sigma_k)^4$ {Eq. 15 to 16}
13:         ω_k:=normalize($\tilde{\omega}_k$)
14:       **end for**
15:       compute ϕ_i^{WENO} and its coefficients $w_j^{i,WENO}$ from Eq. 15
16:       **if** active diffusion **then**
17:         store $w_j^{k=o_p/2}$ {coefficients from central polynomial}
18:       **end if**
19:       $\overline{f}_{i-}^l = v_{left} \cdot \phi_i^{WENO}(x_{i-\frac{1}{2}})$, $\overline{f}_{i-}^r = v_{right} \cdot \phi_i^{WENO}(x_{i+\frac{1}{2}})$
20:       **if** active diffusion **then**
21:         $\overline{d}_{i-}^l = D_\phi \sum_{j=1}^{o_p}(w_j^{k=o_p/2} \cdot (x_{i-\frac{1}{2}})^{j-1})/2$
22:         $\overline{d}_{i-}^r = -D_\phi \sum_{j=1}^{o_p}(w_j^{k=o_p/2} \cdot (x_{i+\frac{1}{2}})^{j-1})/2$
23:       **else**
24:         $\overline{d}_{i-}^l = \overline{d}_{i-}^r = 0$
25:       **end if**
26:       **for** $j = 0$ **to** $n - 1$ **do** {loop over prediction steps}
27:         $\tilde{\overline{\phi}}_i(\tau + \delta t/n) = \tilde{\overline{\phi}}_i(\tau) + w_1^{i,WENO} \cdot (v_{i-\frac{1}{2}} + v_{i+\frac{1}{2}})/2 \cdot \delta t/n$ {Eq. 22}
28:         perform steps 7 to 25
29:         accumulate $\overline{f}_{i-}^l$, $\overline{f}_{i-}^r$, $\overline{d}_{i-}^l$, and $\overline{d}_{i-}^r$
30:         $\tau := \tau + \delta t/n$
31:       **end for**
32:       $\overline{f}_{i+} = \overline{f}_{(i+1)-}^l + \overline{f}_{(i-1)-}^r$, $\overline{d}_{i+} = \overline{d}_{(i+1)-}^l + \overline{d}_{(i-1)-}^r$
33:       $\delta\overline{\phi}_i = (\overline{f}_{i+} - \overline{f}_{i-}^l - \overline{f}_{i-}^r + \overline{d}_{i+} - \overline{d}_{i-}^l - \overline{d}_{i-}^r)\delta t$
34:       **return** $\overline{\phi}_i(t + \delta t) = \overline{\phi}_i(t) + \delta\overline{\phi}_i$
35:     **end for** all threads
36:   **end for** all thread blocks
37: **end for** each dimension

Notes

- All indices are 0-based (e.g., first vector element is indexed with 0).
- The following variables are stored in shared memory (i.e., fast GPU memory at block scope): $\overline{\phi}$, $\tilde{\overline{\phi}}$, ω_k, $w_j^{i,WENO}$, $\overline{f}_{i-}^{l}$ together with $\overline{d}_{i-}^{l}$, and $\overline{f}_{i-}^{r}$ with $\overline{d}_{i-}^{r}$. Whereas we have to store fluxes at the right and left face separately, the fluxes at a given face (e.g., $\overline{f}_{i-}^{r}$ and $\overline{d}_{i-}^{r}$) are combined.
- Line 5 and 6:
  Velocity $v(x_{i-\frac{1}{2}})$ is trilinearly interpolated in space and linearly interpolated in time between $v^{t_n}(x_{i-\frac{1}{2}})$ and $v^{t_{n+1}}(x_{i-\frac{1}{2}})$, where v^{t_n} and $v^{t_{n+1}}$ ($t_n \leq t$, $t+\delta t \leq t_{n+1}$) are two consecutive steps of the simulated velocity field, stored in 3D texture. $v^t(x_{i+\frac{1}{2}})$ is computed accordingly. For each cell we consider only outflow velocity directly, i.e., velocity which is used for calculation of $\overline{f}_{i-}^{l}$ and $\overline{f}_{i-}^{r}$. The upper-script l and r are used to differentiate between fluxes on the right and left face—see comment on lines 19 and 21.
- Line 9:
  The matrices $\mathbf{L}_k$ (one for each reconstruction polynomial ϕ_k), $dim(\mathbf{L}_k) = (o_p+1, o_p+1)$ are stored in the GPU constant memory. In order to compute coefficients w_j^k at a given cell, values ϕ from neighboring cells must be available for each thread in a block. These values are therefore stored in shared memory. The coefficient w_0^k is not needed as it does not influence the oscillation indicator σ.
- Line 11:
  The oscillation matrix Σ, $dim(\Sigma) = (o_p, o_p)$ is precomputed and stored in GPU constant memory.
- Line 19 and 21:
  Only fluxes that go outside a given cell i are computed explicitly. $\overline{f}_{i+} + \overline{d}_{i+}$ are evaluated from $\overline{f}_{(i+1)-}^{l} + \overline{d}_{(i+1)-}^{l}$ and $\overline{f}_{(i-1)-}^{r} + \overline{d}_{(i-1)-}^{r}$, that is, from outflux at left face from cell $i+1$ and outflux at right face from cell $i-1$. In order to access these values from a given cell i they are stored in GPU shared memory.