Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXVI, pp.792-809
Sergio A. Elaskar, Elvio A. Pilotta, Germán A. Torres (Eds.)
Córdoba, Argentina, Octubre 2007

PARTICLE-TRACE VISUALIZATION TECHNIQUE

Mario A. Storti, Gustavo A. Ríos Rodríguez, Norberto N. Nigro, Lisandro D. Dalcín and Rodrigo R. Paz

Centro Internacional de Métodos Computacionales en Ingeniería (CIMEC), INTEC(CONICET-UNL), Güemes 3450, (S3000GLN), Santa Fe, Argentina, mstorti@intec.unl.edu.ar,
http://www.cimec.org.ar/mstorti

Keywords: Visualization, particle trace, computational fluid dynamics

Abstract. Particle tracing is a visualization technique that gives to the observer the simultaneous perception of direction and velocity of the flow. An efficient algorithm is presented allowing the computation of particle trajectories for unsteady flows in moving meshes, including also the computation of skin-friction particles, i.e. particles bound to the body skin. Visualization with the OpenDX package, as well as the parallel implementation in SMP architectures is also described. Discussion of visualization parameters, as particle density, and frame rate in order to avoid coherence effect is discussed.

1 INTRODUCTION

Many visualization techniques are available for the presentation of results in CFD (Computational Fluid Dynamics). Colormaps of some scalar quantities, like magnitude of velocity vector, vorticity, helicity or Q-criterion, are usually shown. However they do not show directly the direction of the flow, at least for low Reynolds number flows. For large Reynolds number flows, eddies are created at shear layers and transported downstream by the fluid, giving indirectly the perception of flow direction. Among the techniques that visualize directly the direction of the flow are the plot of vector as arrows and streamlines or streaklines. However these techniques hardly visualize simultaneously flow direction and magnitude. One of the few techniques that both visualize the direction and magnitude of the flow is particle-tracing (Forssell and Cohen, 1995; Kenwright and Lane, 1996; Steinman, 2000). In this technique, massless particles are introduced in the flow in some points and their movement are tracked by solving ODE's. One disadvantage of the method is that it can be visualized statically, i.e. it only makes sense to use particle-tracing in animations. A related technique is based on textures (van Wijk, 2002; Jobard et al., 2002), it can be more efficient that the presented particle method, but is restricted to 2D, or at most a slice of 3D flows.

In this work some advances in the utilization of the particle trace visualization technique are presented. First, an algorithm for the efficient computation of particle trajectories, specially in moving meshes used when moving boundaries are present, for instance for fluid-structure-interaction or free-surface problems, is presented. Friction particles are particle trajectories that remain attached at the surface and are driven by the viscous traction, much in the same way that friction streamlines are related to free streamlines. Computation and visualization of them is also discussed. Particles can be colored either with a certain scalar magnitude or either a random color may be assigned accordingly to each particle. In the last case, this can help in keeping visual decoherence of the particles, when very large quantities of particles are traced.

Several animations of complex flows are presented. In this examples the animations are computed with the free software **`traceprt`** developed at CIMEC and available elsewhere. This code generates particle trajectories for subsequent visualization with OpenDX.

2 PARTICLE TRACING ALGORITHMS

2.1 Particle tracing on fixed meshes

The algorithm is basically to inject particles at some specified instants and then follow the trajectory of each particle throughout the domain until one of the following conditions are reached

- The particle exits the domain.
- It reaches some kind of invalid position, for instance it reaches a point where the velocity is null and then no further advancing is possible.
- The particle is eliminated because of some visualization strategy, for instance it reaches a region that is not of interest, or the number of particles exceeds a prefixed maximum number.

If the particle was injected at time/position $(\mathbf{x}_0, t^0)$ then we should solve the trajectory by integrating the following ODE for $\mathbf{x}(t)$

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{v}(\mathbf{x}, t), \\ \mathbf{x} &= \mathbf{x}_0, \quad \text{at } t = t^0. \end{aligned} \tag{1}$$

Of course, this is somehow discretized, so that only the positions $\mathbf{x}^n$ at time *"frames"* $t^n = n\Delta t$ are computed. This time step is independent to that one used for the computation of the velocity field itself. It is assumed that some kind of discrete representation for the velocity field is available, in particular we use in general a velocity field constant per element, and interpolated linearly in time between two frames, but more precise representations can be easily introduced. Also the domain Ω is assumed to be tesselated

$$\Omega = \bigcup_{e \in [0, n_{\text{elem}})} \Omega_e \tag{2}$$

Assuming first that the velocity field is constant in time the algorithm is basically as follows. The element e such that $\mathbf{x}_0 \in \Omega_e$ is found. The particle is then inside that element at time t^0. Then a starting position *inside* the element is known and the *exiting* time and position are computed. If the exiting time is greater than the target time, then the position at the target time is interpolated, otherwise the trajectory is continued on the element that shares the face that contains the exiting point with the current element. The algorithm is summarized as follows

Given $(x,t)^0$, computes $(x,t)^j$ for $j = 1, n$, as approximate points on the particle trajectory.
Find element e such that $\mathbf{x}^0 \in \Omega_e$.
for $j = 0$, to $n-1$ **do**
  { **Loop over frames, advance particle from** t^j **to** t^{j+1} }
  $k = 0$, $(\mathbf{x},t)^{\text{in}} = (\mathbf{x},t)^j$
  loop
    { **Advance particle from** t^j **to** t^{j+1} }
    { **traversing as many elements as needed** }
    From $(\mathbf{x},t)^{\text{in}}$, element velocity $\mathbf{v}_e$, and element geometry Ω_e compute the exiting time and position $(\mathbf{x},t)^{\text{out}}$
    if $t^{\text{out}} \leq t^{\text{in}} + \text{tol}$ **then**
      Mark particle as dead.
      break
    else if $t^{\text{out}} \geq t^{j+1}$ **then**
      $\mathbf{x}^{j+1} = \dfrac{t^{j+1} - t^{\text{in}}}{t^{\text{out}} - t^{\text{in}}}(\mathbf{x}^{\text{out}} - \mathbf{x}^{\text{in}})$
      break
    else
      e = element that shares with e the face containing $\mathbf{x}^{\text{out}}$
      $(\mathbf{x},t)^{\text{in}} = (\mathbf{x},t)^{\text{out}}$
    end if
  end loop
end for

Some notes:

- In section §2.4 a method to compute the exiting time t^{out} efficiently will be presented.
- Condition $t^{\text{out}} \leq t^{\text{in}} + \text{tol}$ may happen if, when exiting some element e (see figure 2) by face ab, it is found that in the following element e' the velocity field $\mathbf{v}_{e'}$ is exiting from the same side ab. In this case, the exiting time from e' is the same as the inlet time $t^{\text{out}} = t^{\text{in}}$, so that. Note that this kind of velocity field is not admissible if the continuity equation is imposed in a strong form. So, in a finite element context, where the continuity equation is imposed in a weak form, it is expected that this kind of situation could happen, but with a low probability. In this case the particle is said to be *"lost"*.

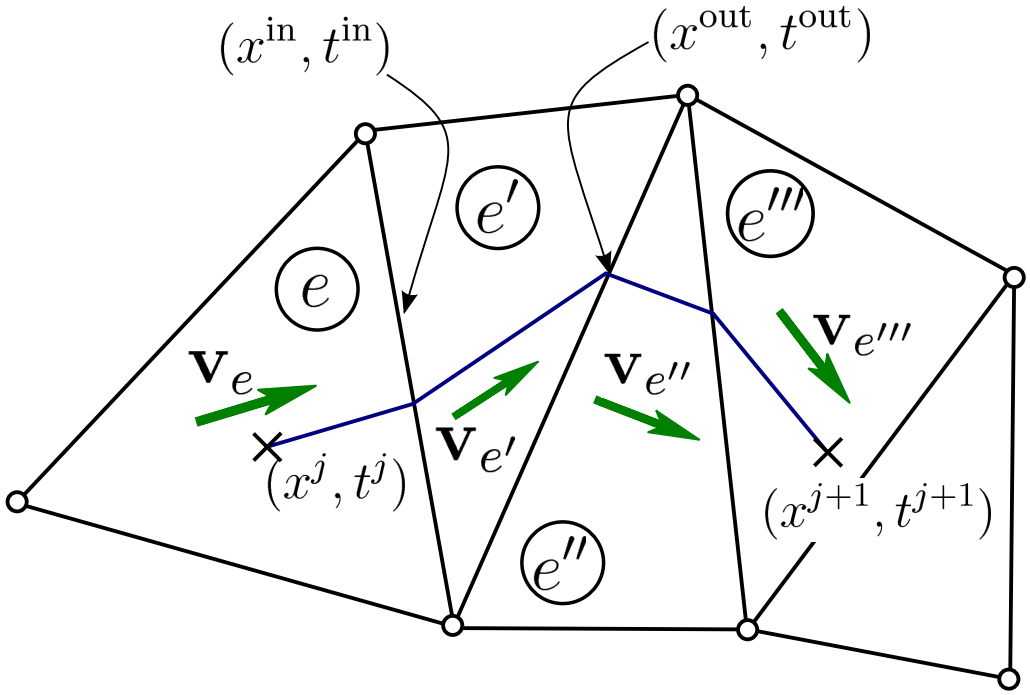

Figure 1: Tracing particle trajectory from t^j to t^{j+1}

2.2 Computing area coordinates for an arbitrary point

Given a point $\mathbf{x}_0 \in \Omega_e$, where Ω_e is a simplicial element, the following algorithm computes its area (or volume in 3D) coordinates (or FEM linear interpolation functions) N_j. Inside the element

$$\mathbf{x} = \sum_{j=1}^{n_{\text{el}}} N_j(\mathbf{x})\, \mathbf{x}_j, \tag{3}$$

where $\mathbf{x}_j$ are the node coordinates. The following restrictions must be imposed

$$1 = \sum_{j=1}^{n_{\text{el}}} N_j(\mathbf{x}). \tag{4}$$

As the N_j are linear

$$N_j(\mathbf{x}) = a_j \mathbf{x} + b_j. \tag{5}$$

Specializing at the nodal points a set of n_{el} equations of the form

$$\mathbf{I} = \mathbf{AC}, \tag{6}$$

are obtained, where

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_0 & 1 \\ \mathbf{a}_1 & 1 \\ \vdots & \vdots \end{bmatrix}, \text{ and} \qquad \mathbf{C} = \begin{bmatrix} \mathbf{x}_0 & \mathbf{x}_1 & \dots \\ 1 & 1 & \dots \end{bmatrix}. \tag{7}$$

So $\mathbf{A}$ can be computed as $\mathbf{A} = \mathbf{C}^{-1}$. Once $\mathbf{A}$ is obtained the area coordinates for a given point can be computed as

$$\mathbf{N}(\mathbf{x}) = \mathbf{A} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}, \tag{8}$$

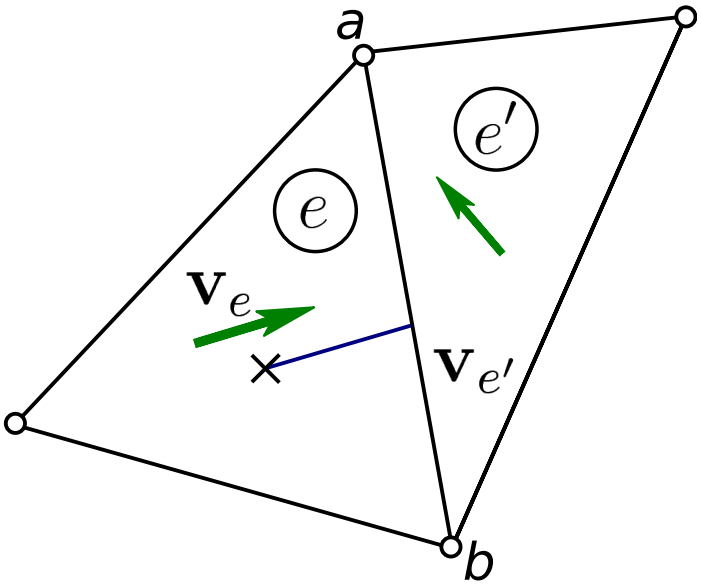

Figure 2: Tracing particle trajectory from t^j to t^{j+1}. Entering element e' finds a velocity that is exiting from the entering face.

and their gradients (which are constant) are the first n_{dim} columns of $\mathbf{A}$, where n_{dim} is the number if spatial dimensions.

$$\mathbf{A} = \begin{bmatrix} \nabla N_1 & * \\ \nabla N_2 & * \\ \vdots & \vdots \end{bmatrix}. \tag{9}$$

2.3 Finding the initial element

When injecting a particle at a certain position $\mathbf{x}_0$, the first task is to find the simplicial element e such that $\mathbf{x}_0 \in \Omega_e$. A first, exhaustive approach is to test for all elements, compute the area coordinates as described in section §2.2 and check the values of them. If all the area coordinates are in range $[0, 1]$, then $\mathbf{x}_0 \in \Omega_e$. However, this algorithm is in the worst and average case $O(n_{\text{elem}})$.

In order to reduce the cost of this task, a fictitious trajectory can be started from the center $\mathbf{x}_e$ of any element e using the algorithm described in section §2.1, with a dummy velocity field defined by $\mathbf{v} = \mathbf{x}_0 - \mathbf{x}_e$. This fictitious trajectory should eventually reach point $\mathbf{x}_0$. The last element traversed while following this trajectory should be the element where $\mathbf{x}_0$ belongs. In this way, the cost would be $O(\text{number of traversed elements}) \propto {n_{\text{elem}}}^{1/n_{\text{dim}}}$, which is significantly lower than n_{elem}, specially in higher dimensions. However, this algorithm can fail if the domain is non-convex, and then the fictitious trajectory is not fully contained in the domain.

Finally, this can be fixed in most cases by constructing an octree (quadtree in 2D) with the points at the center of the elements and taking a starting point not a random element, but the element whose center is closer to the starting point $\mathbf{x}_0$. In this way, the cost is reduced to $O(\log n_{\text{elem}})$ for computing the closest element center with the octree algorithm and then to follow a (probably very short, i.e. $O(1)$) trajectory to the starting point. This also has the benefit of reducing the probability of not reaching the starting element for non-convex domains.

For the implementation in **`traceprt`** the package ANN (Arya et al., 1998; Arya and Mount, 2006), a library written in C++, which supports data structures and algorithms for both exact and approximate nearest neighbor searching in arbitrarily high dimensions, has been used. In

the nearest neighbor problem a set of data points in d-dimensional space is given. These points are preprocessed into a data structure, so that given any query point q, the nearest or generally k nearest points of P to q can be reported efficiently. The library implements a number of different data structures, based on *kd*-trees and box-decomposition trees.

2.4 Computing the trajectory of a particle inside an element

Assuming that a starting point $\mathbf{x}_0 \in \Omega_e$ at time t^0 is available, and the velocity in the element is $\mathbf{v}_e$ the temporal derivatives of the area coordinates N_k along the particle trajectory are computed as

$$\frac{\mathrm{d}N_k}{\mathrm{d}t} = \nabla N_k \cdot \mathbf{v}_e. \tag{10}$$

As the particle must exit through a plane, some area coordinate will be null at the exit point. Conversely, if some area coordinate is null at a point on the trajectory the particle is exiting the element, so that the exit time t^j can be computed as

$$t^{\mathrm{out}} - t^{\mathrm{in}} = \min_k \frac{-N_k}{(\mathrm{d}N_k/\mathrm{d}t)}. \tag{11}$$

Note that, as the N_k are positive, and some derivative must be negative, only the minimum over those j with negative derivative must be taken into account. The exit position is then computed simply as

$$\mathbf{x}^{\mathrm{out}} = \mathbf{x}^{\mathrm{in}} + \mathbf{v}_e(t^{\mathrm{out}} - t^{\mathrm{in}}). \tag{12}$$

2.5 Parallel implementation

The algorithm can be easily implemented in parallel with threads in a SMP architecture. The most consuming CPU time stage is the advancing of particles. One thread is started in each processor, and a global index controlled by a semaphore points to the next particle to be advanced. The algebra is computed with the FastMat2 matrix algebra library that is included in the PETSC-FEM package (Storti et al., 2007; Sonzogni et al., 2000, 2002). This library has a caching strategy that allows faster execution while allowing multi-indices and other advanced features, however it is reentrant, so that execution in a threaded environment is safe. In an Intel Core Duo T2050 @1.60GHz processor (two cores), the processing time for advancing 25,000 particles 50 frames was, 33.5secs (26.8 secs/Mprtcls/frame) with 1 core and 18.2secs (14.4 secs/Mprtcls/frame) with two cores. This represents and efficiency of 92%. The implementation uses the Native POSIX Threads Library (NPTL) included in the Fedora Core 6 distribution (Glibc 2.5.3, Gcc 4.1.1) (Butenhof, 1997; GNU Project, 2007). Implementation for distributed memory environments with the MPI is under way.

3 SKIN-FRICTION PARTICLES

It is interesting to trace particles on a surface along the *"skin-friction lines"* (Monson et al., 1993; Tobak and Peake, 1982). Skin-friction lines are the limit of streamlines as the body skin is approached. First, a vector field tangent to the body skin must be computed, for instance by taking the viscous traction on the surface. In this way, the skin-friction lines represent the streamlines for small particles that lie on the surface and are dragged around the skin by the friction of the fluid. They can be determined experimentally in wind tunnel models by distributing a thin layer of oil on the body, and allowing the oil to be redistributed under the viscous traction of the fluid, also known as oil-streak technique. *"Skin-friction particles"* are to normal particles the same as skin-friction lines are to streamlines.

3.1 Surface traction field

First a per element traction field that is parallel to the body skin is determined. In general this can be computed as the tangential component $\mathbf{t}_{\|}$ of the surface traction field $\mathbf{t}$, i.e.

$$\begin{aligned} \mathbf{t} &= \boldsymbol{\tau} \cdot \hat{\mathbf{n}}, \\ \mathbf{t}_{\|} &= \mathbf{t} - (\mathbf{t} \cdot \hat{\mathbf{n}})\, \hat{\mathbf{n}}, \end{aligned} \tag{13}$$

where $\boldsymbol{\tau}$ is the deviatoric stress tensor. This in turn, can be computed from the velocity field as

$$\tau_{ij} = \mu(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}). \tag{14}$$

so that,

$$\begin{aligned} t_i &= \tau_{ij} n_j \\ &= \mu(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}) n_j. \end{aligned} \tag{15}$$

If the surface is a plane (for instance in the case of a simplicial element), then the last term can be put as

$$\mu \frac{\partial u_j}{\partial x_i} n_j = \mu \frac{\partial}{\partial x_i}(u_j n_j) = 0, \tag{16}$$

since $u_j n_j \equiv 0$ due to the non-slip condition. So that, a traction-like field computed with the simplified expression

$$\tilde{t_i} = \frac{\partial u_i}{\partial x_j} n_j, \tag{17}$$

can be used, where, in addition μ has been assumed to be constant.

Finally, if slip or wall-law condition are enforced, then they provide directly with a velocity field on the surface. In the last case, it must be taken into account that the velocity is known at a certain distance y_{wall} of the surface. Normally one can use the universal law of the wall in order to compute the traction

$$\frac{u}{u^*} = f\left(\frac{y_{\text{wall}} u^*}{\nu}\right). \tag{18}$$

As u is known, this equation can be solved for u^* and then the wall friction is obtained from

$$\tau_w = \rho (u^*)^2. \tag{19}$$

The direction of the traction is assumed to be parallel to velocity, i.e.

$$\mathbf{t} = \tau_w \frac{\mathbf{u}}{u}. \tag{20}$$

3.2 Tracking particles on surfaces

The algorithm for computing area coordinates must be somewhat modified. For any point $\mathbf{x}'$ in the surface

$$\begin{aligned} \mathbf{x}' &= \sum_{j=1}^{n_{\text{el}}} N_j(\mathbf{x})\, \mathbf{x}_j = \mathbf{BN}, \\ 1 &= \sum_{j=1}^{n_{\text{el}}} N_j(\mathbf{x}), \end{aligned} \tag{21}$$

where

$$\mathbf{B} = \left[\begin{array}{ccc} \mathbf{x}_1 & \mathbf{x}_2 & \dots \end{array} \right]. \tag{22}$$

Given a point $\mathbf{x}$ that is not on the surface, projection on the surface $\mathbf{x}^*$ must be computed. This is the point that has minimum distance to the plane, i.e.

$$\begin{aligned} \mathbf{x}^* &= \sum_j N_j^* \mathbf{x}^j = \mathbf{B}\mathbf{N}^*, \\ \{\mathbf{N}^*, \lambda^*\} &= \underset{\mathbf{N},\lambda}{\operatorname{argmin}} \left\{ \tfrac{1}{2} \|\mathbf{B}\mathbf{N} - \mathbf{x}\|^2 + \lambda \sum \mathbf{N}_j \right\}. \end{aligned} \tag{23}$$

The last term in the functional is a Lagrange multiplier term in order to enforce the restriction. Expanding the quadratic term the following set of equations is obtained

$$\left[\begin{array}{cc} \mathbf{B}^T\mathbf{B} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} \end{array} \right] \left[\begin{array}{c} \mathbf{N}^* \\ \lambda^* \end{array} \right] = \left[\begin{array}{c} \mathbf{B}^T\mathbf{x} \\ 1 \end{array} \right]. \tag{24}$$

It can be shown that in this way $\mathbf{x} - \mathbf{x}^* = \mathbf{x} - \mathbf{B}\mathbf{N}$ is orthogonal to the vectors tangential to the plane spanned by the vectors $\{\mathbf{x}_j\}$. Effectively, any vector in the plane $\mathbf{w}$ is a combination of the form

$$\mathbf{w} = \sum_j \alpha_j \mathbf{x}_j = \boldsymbol{\alpha}^T \mathbf{B}, \quad \text{with} \quad \sum_j \alpha_j = \boldsymbol{\alpha}^T \mathbf{1} = 1. \tag{25}$$

Let $\mathbf{w}_1 = \mathbf{B}\boldsymbol{\alpha}_1$, $\mathbf{w}_2 = \mathbf{B}\boldsymbol{\alpha}_2$ two vectors in that plane. Then from the first equation of (24)

$$\boldsymbol{\alpha}_j^T (\mathbf{B}^T\mathbf{B}\mathbf{N}^* + \lambda \mathbf{1} - \mathbf{B}^T\mathbf{x}) = 0, \tag{26}$$

so that

$$\mathbf{w}_j(\mathbf{B}\mathbf{N} - \mathbf{x}) + \lambda = 0, \tag{27}$$

and substracting the equation for $\mathbf{w}_2$ from that one for $\mathbf{w}_1$ the following expression is obtained

$$(\mathbf{w}_2 - \mathbf{w}_1)(\mathbf{B}\mathbf{N} - \mathbf{x}) = 0. \tag{28}$$

As $\mathbf{w}_2 - \mathbf{w}_1$ represents any vector tangential to the plane, it means that $\mathbf{x}^* - \mathbf{x} = \mathbf{B}\mathbf{N} - \mathbf{x}$ is orthogonal to the plane.

From (24)

$$\left[\begin{array}{c} \mathbf{N} \\ \lambda \end{array} \right] = \mathbf{H}^{-1} \left[\begin{array}{c} \mathbf{B}^T\mathbf{x} \\ 0 \end{array} \right], \tag{29}$$

from where the expressions for computing the gradients of the N_j are obtained, i.e. they are the first n_{dim} columns of the r.h.s. in (29)

$$\mathbf{H}^{-1} \left[\begin{array}{c} \mathbf{B}^T\mathbf{x} \\ 0 \end{array} \right] = \left[\begin{array}{cc} \nabla N_1 & * \\ \nabla N_2 & * \\ \vdots & \vdots \end{array} \right]. \tag{30}$$

So, to trace particles on a surface the scheme proceeds in the same way as for the standard tracing algorithm, but while computing the total derivative of the area coordinates (10) the gradients of the area coordinates given by (30) should be used instead of (9). Note that as the gradients of the area coordinates are parallel to the plane, eq. (10) includes already a projection of $\mathbf{v}_e$ on the surface.

3.3 Velocity scale for skin-friction particles

Note that as described in section §3.1 the field generated has in general dimensions of traction. In order to convert it to velocity an equivalence parameter must be introduced. For instance consider a thin layer of oil of constant thickness h_{oil} and viscosity μ_{oil}. Then the velocity field $\mathbf{v}_{\text{oil}}$ would be

$$\mathbf{v}_{\text{oil}} = (h/\mu)_{\text{oil}}\mathbf{t} = \beta\mathbf{t}. \tag{31}$$

Note that in the context of flow visualization the parameter β can be chosen freely. If friction particles are visualized alone, i.e. not at the same time with normal ("fluid-volume") particles, then β acts simply as a scale for the surface velocity field. However, if also fluid-volume particles are visualized, then β controls the ration between fluid velocity and skin-friction velocity. So, β should be low enough to produce a skin flow that is slower than the volume fluid, but not so low so as to make it to look as almost at rest.

4 IMPLEMENTATION DETAILS

4.1 Coherence and frame rate limit

If too few frames of the particles are shown, then the position of the particle at the next frame can not be correlated by the observer to the previous ones and the sense of direction and speed are lost. This is clearly understood if it is assumed that the particles are arranged in a regular Cartesian grid of spacing h in all spatial directions. If the distance traversed by the particle in the frame step $v\Delta t$ is small with respect to h, then the observer can easily keep the identity of each particle (decoherence). This is more difficult as $v\Delta t$ approaches h, and in the limit of $v\Delta t = h$ complete coherence is produced. If $\mathbf{v}$ is parallel to a spatial axis (say x), then the position of particle (i, j, k) at time frame t^{n+a} is the same as the position of particle $(i + 1, j, k)$ at time t^n, so that the particles seem to be steady to the observer. If the flow direction is not aligned with an axis, then the particles move, but is difficult to the observer to determine the direction and speed of the flow, since many vectors $\mathbf{v}$ give the same particle positions. For instance a velocity such that $\mathbf{v} = (h/\Delta t + \epsilon, 0, 0)$, with ϵ small, is in fact mistaken as a slow velocity $\mathbf{v} = (\epsilon, 0, 0)$. This posses a limit on the frame rate

$$v\Delta t/h < \text{Co}_{\text{cr}}, \tag{32}$$

where $\text{Co}_{\text{cr}} = O(1)$ is some constant (probably depending on the ability of the observer). Note that the right hand side of (32) is similar to a Courant number, hence its name.

Note that, for a fixed velocity and time frame, this imposes a limit on the density of particles. If a higher density is desired, then lower time frames must be used, with the undesired effect of giving the impression of a slower flow. A key point in the particle tracing algorithm is to modify the basic technique in order to extend the above limit.

One first possibility is the use of randomly positioned particles. If the particles are randomly positioned, then the random spacing of the particles is perceived as larger scale tracer that helps the observer in tracing the individual particles. Another possibility is the use of colors, if each particle is given a color, then if m distinct colors are used, then $m^{1/n_{\text{dim}}}$ colors can be used in any spatial direction and coherence is produced when

$$v\Delta t/h < \text{Co}_{\text{cr}} m^{1/n_{\text{dim}}}, \tag{33}$$

effectively extending the coherence limit by a factor of $m^{1/n_{\text{dim}}}$. Of course, the number of colors m to effectively use there is not the that one given by the combination of software and hardware

used, since the combination of colors that are available currently is several magnitudes larger than the number of colors that an observer can distinguish in the context of particle tracing. Equation (33) is to be interpreted only as an indication of the effect of adding coloring to the particles.

4.2 Creation and destruction of particles

Particles are injected randomly in a certain region defined by the user with a certain non uniform probability. For instance, a common choice in exterior aerodynamics is to create particles near the skin of the body, or at a a certain region upstream of the body. The particles are injected with a certain temporal and spatial probability so as to produce a constant (in average) rate of particles Q_{inj} (measured in particles/sec). In the case of open flows, after a certain time the particles start exiting the domain. Also, some particles may be lost due to inaccuracies in the flow representation (see §2.1). In most cases, the particle population reaches a limit, and a balance (in average) of the form

$$Q_{\text{inj}} = Q_{\text{lost}} + Q_{\text{exit}}, \tag{34}$$

is obtained. In general, it is better to start the animation *after* the steady state population is reached.

In very long domains, typically used in external aerodynamics in the wake of the body, the time for particles to exit the domain may be high, and a lot of computational resources may be wasted in following particles that are in regions of low interest (far behind the body). In this case a strategy may be employed where the particles are *"killed by age"*, i.e. after a certain time τ_{kill} from their injection. Another possibility is to kill them wen they enter a certain region. The balance in this case is

$$Q_{\text{inj}} = Q_{\text{lost}} + Q_{\text{exit}} + Q_{\text{kill}}. \tag{35}$$

In closed flows, i.e. flows with no inlet/outlet Q_{exit} is null, so that the only possibility to reach a balance is either by Q_{lost} or Q_{kill}. If no killing by age is in effect, injection can only be balanced with Q_{lost}, but the rate at which particles are lost may be too low, specially if the velocity field is very well represented. As the number of lost particles is roughly proportional to the number of particles present $Q_{\text{lost}} \propto n_{\text{part}}$, this results in that the number of particles must be too high, and so the time to reach the steady state. This means that in this case also killing by age, should be necessary. If Q_{lost} is neglected, then the age limit should be given by

$$Q_{\text{inj}} \tau_{\text{kill}} = n_{\text{part}}. \tag{36}$$

In order to obtain a certain desired amount of particles in the steady state n_{part}, an infinite combination of Q_{inj} and τ_{kill} may be used. However a too small τ_{kill} would give very short particle trajectories and, conversely, a very high value would give a large initialization time to reach the steady state.

4.3 Visualization with OpenDX

The computed trajectories of the particles are visualized with the Open Data Explorer (OpenDX) program (Thompson, 2006). The particles are visualized using the *"Glyph"* module. Spheres give the better quality visualization, but are very expensive in CPU time and memory. Best visualizations have been obtained by reducing the quality to the minimum (the *"spiffy"* or *"diamond"* glyphs). In this way, visualization with $O(10^5 - 10^6)$ particles have been possible. A

color is attached to each particle by assigning a data component to the particle positions and then using a colormap. It has been proven useful to make the particles brighter than the body skin, for instance by controlling the specular, diffuse, and ambient properties through the *"Options"* module.

4.4 Hexahedral and wedge meshes

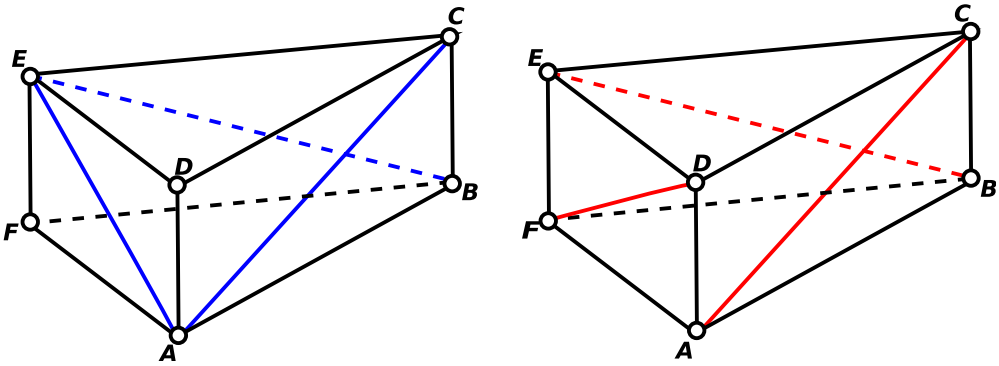

Figure 3: Right: admissible face splitting of a prism. Left: not admissible face splitting.

It is common in exterior aerodynamics to add thin layers of structured elements to the body skin (Franck et al., 2004) in order to represent better the boundary layer. Even if the volume occupied by these layers is small, the computational requirements may be similar or higher than the rest of the mesh. Also important phenomena occurs in the vicinity of the body, and an accurate representation of the flow in this region is needed. The skin layers of elements are usually obtained as the cartesian product of the surface mesh and a one dimensional mesh in the normal direction to the skin. This generates layers of prism elements (also called *"wedges"*) if the surface mesh is composed of triangles, or hexahedral elements if it is composed of quadrangles. Since the algorithm, as presented so far, is valid for tetrahedra only, one possibility is to split the wedges and/or hexahedrals in tetrahedra. The other possibility is to extend the algorithm to these new kind of element geometries. However, OpenDX itself is unable to manage efficiently sets of elements with mixed geometry, so that the first alternative is chosen.

The splitting of a prism element is defined by first splitting the quadrangular faces in two triangles. Figure 3 shows a possible face splitting at right, inducing a splitting in three tetrahedra, namely $AEFB$, $ADCE$, and $BCEA$. Since each quadrangular face can be split in two ways, there are at most eight possible splittings. However, two out from these eight face splittings are not possible, for instance the splitting shown in figure 3 at the left does not give a feasible splitting in 3 tetrahedra. It can be shown that the two non-admissible splittings are those in which the *"slopes"* of the diagonals have all the same sign while traversing the edges of the triangular faces in a given sense. For instance, while traversing the edges of triangle FAB in the prism at the left in the sense $F \rightarrow A \rightarrow B \rightarrow F$, it can be verified that all the diagonals BE, FD, AC have a *"positive"* slope, while for the splitting at the left the signs are positive, negative, positive.

Of course a splitting in more tetrahedra is possible adding a node in the center of the prism and interpolating the physical values (for instance, velocity) there. However, only splittings that result in a minimum of tetrahedra and that do not need of interpolation are considered here, and this option is considered only as a last resource.

Once a splitting has been chosen for the faces of a given prism element e, the adjacent prisms

e' are constrained to split the shared faces in the same way as on the e side. For instance, in figure 4, face $ABCD$ in element e is split along diagonal AC, so that the same split must be chosen on face $A'B'C'D'$ of element e', as is shown on the left. Splitting of the face along diagonal $B'D'$ (as shown at the right) is invalid.

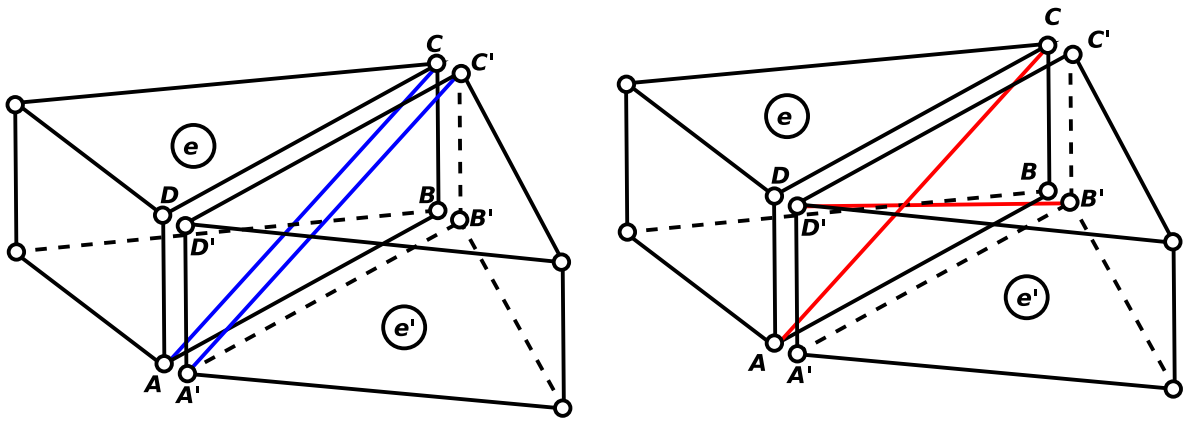

Figure 4: Left: Splitting from side e is compatible with that on the side of element e'. Right: Face splitting on elements e and e' is no compatible on the shared face.

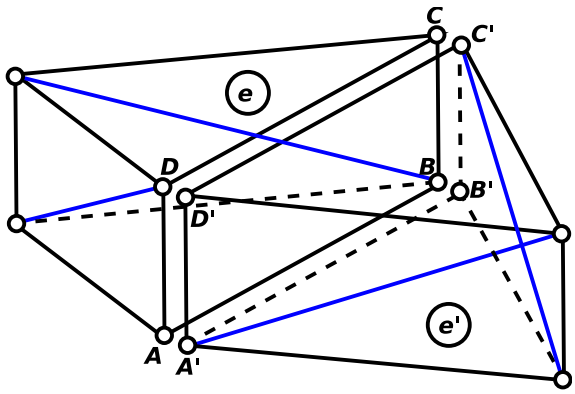

Figure 5: Splitting from the side of element e is incompatible with the splitting from side of element e'

Choosing a face splitting for all the quadrilateral faces in the mesh can be put as a very well known problem in computing science known as *"graph coloring"*. The problem is to color all the quadrilaterals of the faces with two colors (i.e. to assign one of the two possible splittings) in such a way that the three faces of each element are split in a compatible way, as described above. The algorithm is briefly as follows. Initially all faces have no color assigned. All the elements that have two colored (quadrilateral) faces are revised. If one of them has the same slope signs, then the remaining face is colored in the only possible way. If no element is in this condition the remaining elements are checked and a color is assigned to one of their faces. The elements are revised in decreasing order of the number of colored faces, i.e. first those with two colored faces (but not with the same sign, or it will be given a color in the previous step), then those with one colored face, then those with no colored faces. This process is repeated until all faces are assigned a color.

The algorithm may fail, for instance when revising element e (see figure 5) splitting along diagonal BD must be chosen, but this induces an invalid splitting on element e'. In practice this

condition has been never met. Note that the constraint on coloring is a very loose one, for each element only two out of eight possible colors are invalid. However, if this condition arises the solution is to add a center node, and split the prism in 6 tetrahedra.

The same basic technique can be applied *mutatis mutandis*, to hexahedral elements. However, it can be shown that from all the possible $2^6 = 64$ possible splittings of the faces, only two are admissible. So, the coloring problem is much more constrained, and invalid situations arise (but still are rare). This is again solved adding a center node, and splitting the hexahedral in twelve tetrahedra.

5 EXAMPLES

It is difficult, of course, to show the benefits of this technique in static figures, since its main advantages are realized only through animations. However, the following examples will show some of the problems described above.

The first example is the visualization of the flow around a car (the Volkswagen *Bora* model) (see figures 6 to 8). This is a simplified model of a racing car from the *Sport Team* squad, currently running in the Argentinian category TC2000. This simplified model takes into account some modifications of the racing car with respect to the roadgoing version, but has no rotating wheels, neither the rear aerofoil. The finite element mesh had 2,371,121 tetrahedra and 547,539 prisms (arranged in three layers of 182,513 elements each). After splitting the prisms, the mesh had 4,013,738 tetrahedra. Injection of particles has been performed uniformly in a layer 5cm thick around the car skin. The surface of the car is colored with vorticity. After reaching a steady state an average of approximmately 20,000 particles are present in the domain shown with the *diamond* type of glyph. Figure 7 shows visualization with *sphere* glyphs. In this case fewer particles can be visualized and the rate of injected particles has been adjusted so as to have an average number in the limit in the order of 4,000. Finally figure 8 shows friction particles in the back. Note the large recirculation zones where the particles spiral towards the center, where finally a large vortical structure detach. Results shown are for a velocity of 200 km/hr (Reynolds number 1.67×10^7).

The next example is the subsonic aerodynamics of a 155mm projectile for tube artillery named *"PACU"* (for *"Proyectil Argentino de Culote Hueco"*) developed by CITEFA (*Instituto de Investigaciones Científicas y Técnicas de las Fuerzas Armadas*, http://www.citefa.gov.ar) and being used currently in the Argentinian Army. There are two main factors to distinguish projectile aerodynamics from classic aerodynamics. The first of them is the fact that most projectiles have an axis or plane of symmetry, which implies also symmetric aerodynamic parameters. The second one is related to large spinning velocities that, for tube artillery, are from 5,000 to 10,000 RPM, generating aerodynamic effects which are only found in the aeroballistic area. Among these particular parameters are the force and torque produced by the Magnus effect. Although this force is relatively lower than the lift and can be ignored, the torque is critical for the projectile stability. Figures 9 and 10 show particle tracings around the projectile at an angle of attack of 5°. Particles are injected in small region upwind from the tip of the projectile, simulating the effect of injection of a jet of smoke. Again, vorticity is shown in colors on the body skin. Figure 10 shows the detail near the bottom of the projectile. Note the large concentration of particles in the inner part of the bottom, due to the recirculation. Finally, figure 11 shows skin-friction particles. Note the concentration of particles at the separation line, which is twisted in the sense of rotation of the projectile.

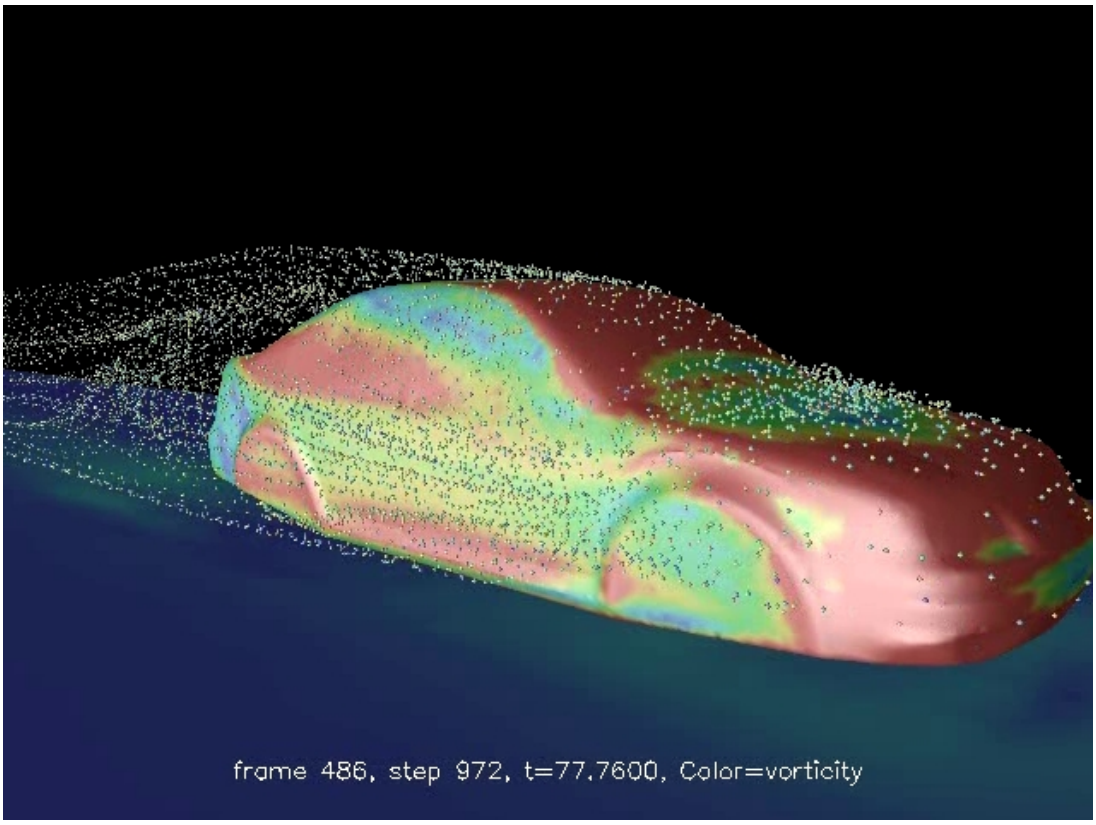

Figure 6: Flow around a racing car prototype. Diamond glyphs.

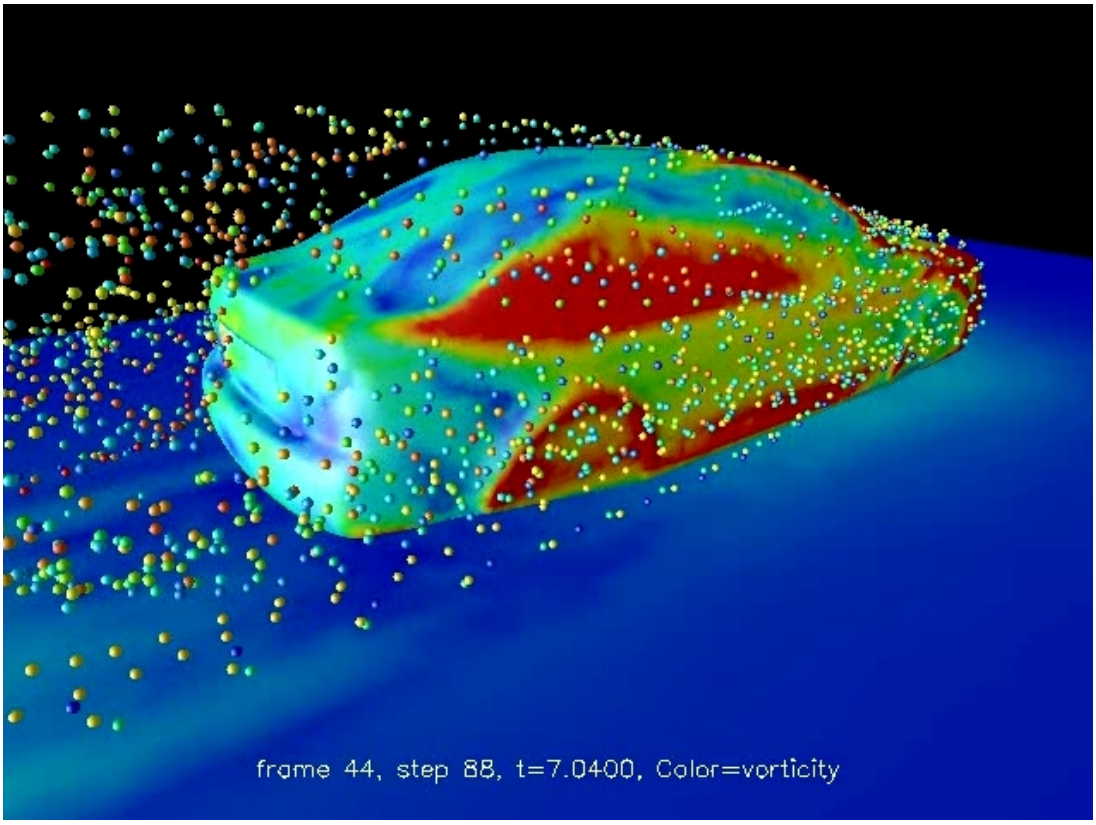

Figure 7: Flow around a racing car prototype. Sphere glyphs.

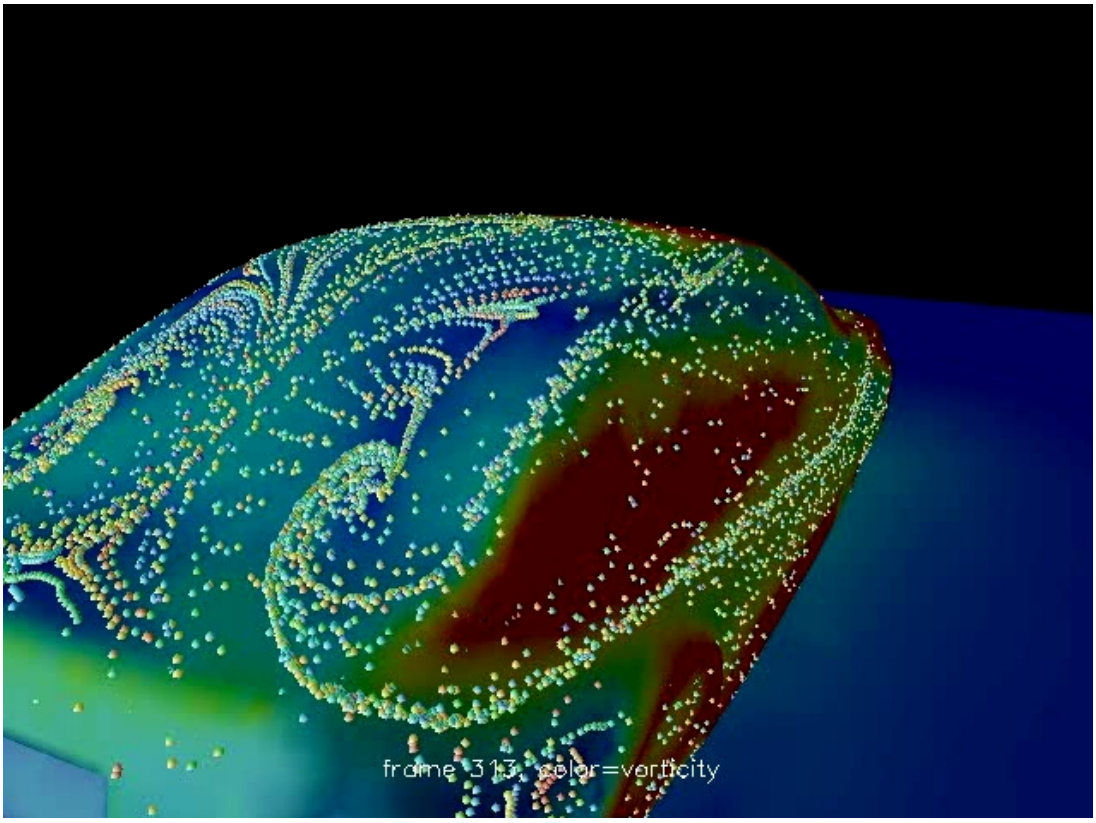

Figure 8: Flow around a racing car prototype. Skin-friction particles.

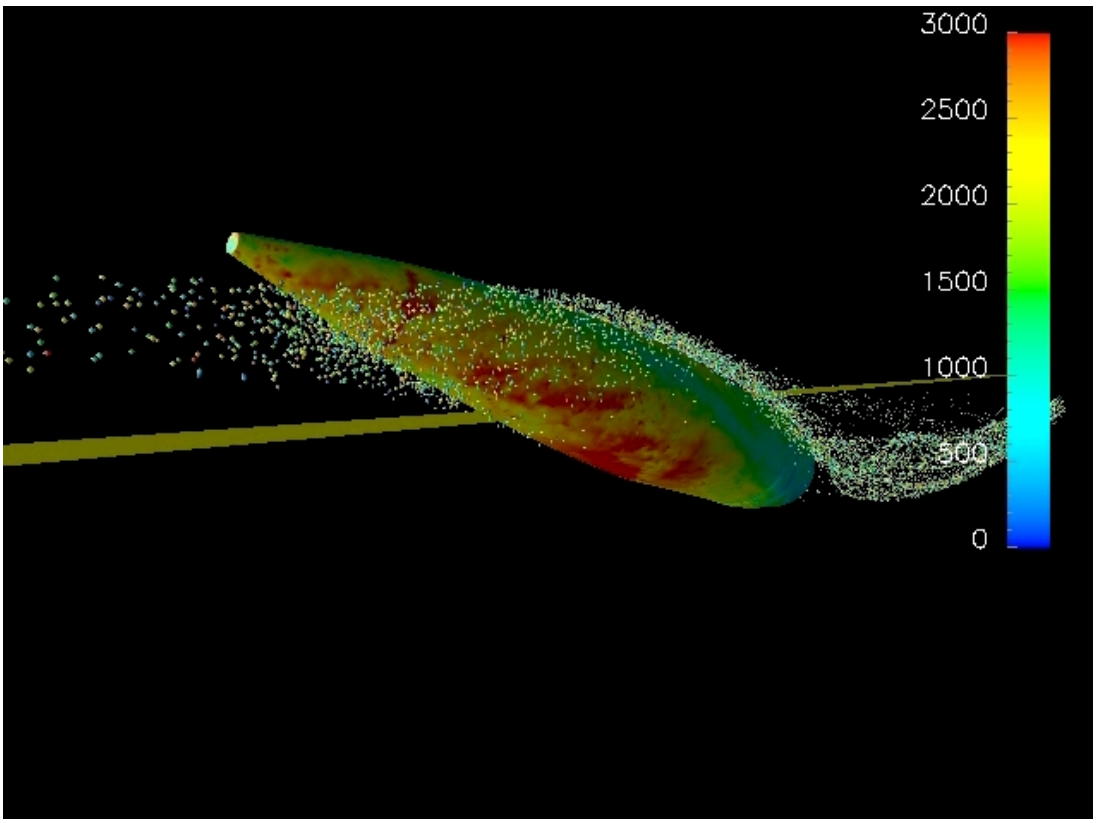

Figure 9: Particle tracings simulating a smoke jet.

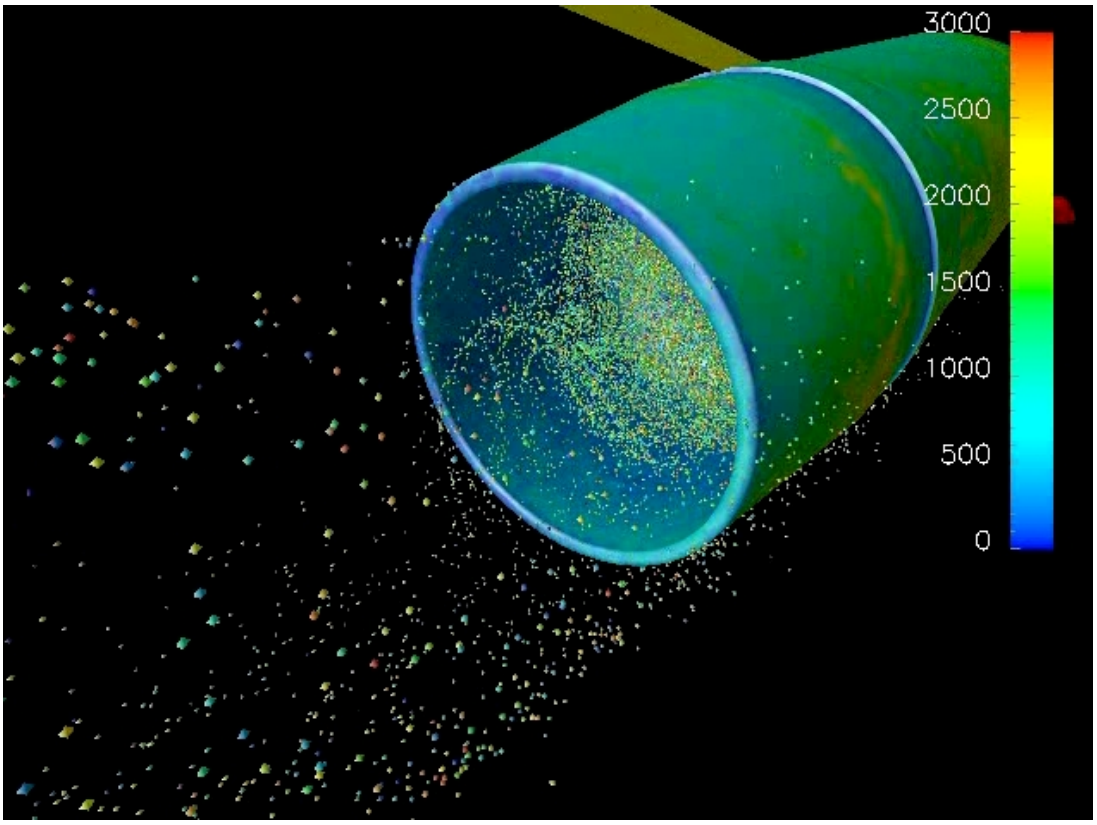

Figure 10: Particle tracings in the inner part of the bottom.

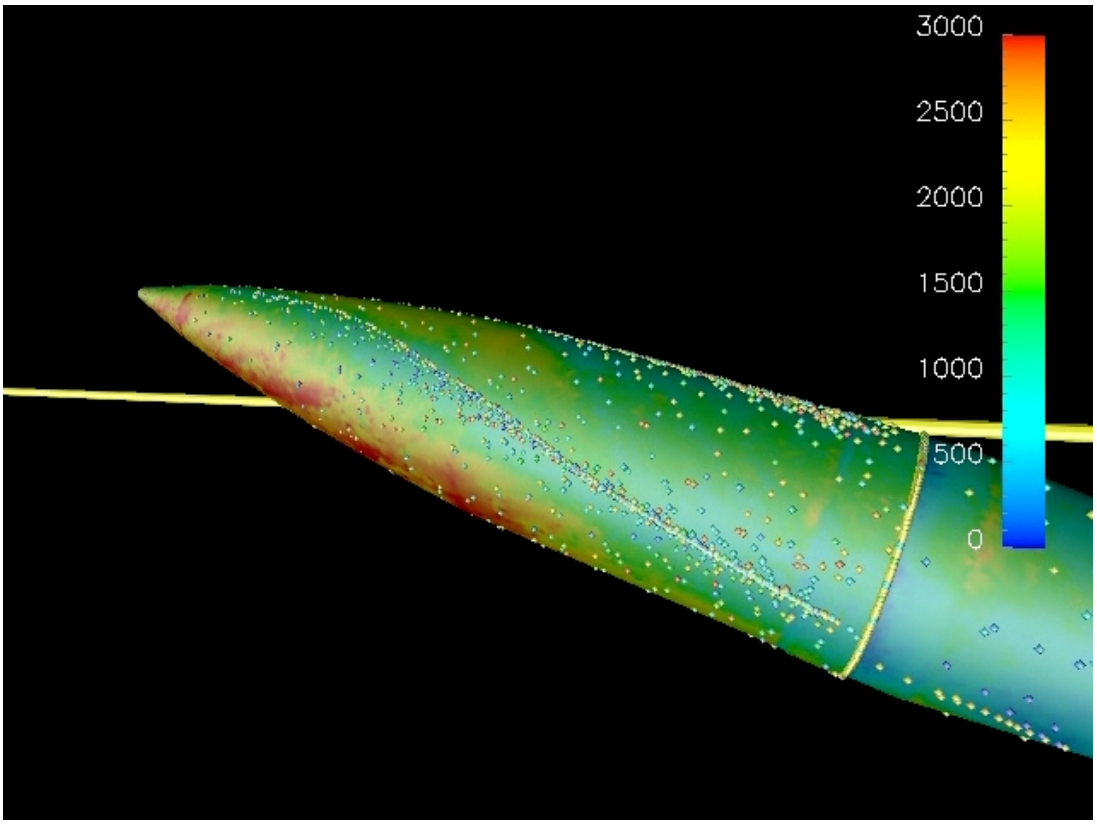

Figure 11: Skin-friction particles, note the concentration of particles at the separation line.

6 CONCLUSIONS

Particle tracing is a powerful visualization technique for CFD problems, giving to the user a simultaneous perception of direction and speed of the flow. The algorithm presented here allows the efficient computation of particle trajectories for unsteady flows in moving meshes. The algorithm can be applied also to the computation of skin-friction particle trajectories. Application to industrial problems has been presented.

Acknowledgment

This work has received financial support from Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET, Argentina, grants PIP 02552/00, PIP 5271/05), Universidad Nacional del Litoral (UNL, Argentina, grant CAI+D 2005-10-64) and Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT, Argentina, grants PICT 12-14573/2003, PME 209/2003). Also, the authors wish to thank *Ing. Fernando Bulla* from *Sport-Team Competición* for providing data for the racing car example, and *Ing. Javer Garibaldi* from *CITEFA* for providing data for the artillery projectile example. Authors made extensive use of freely distributed software as GNU/Linux OS, MPI, PETSc, GCC compilers, Octave, Open-DX, Perl, among many others. Also, many ideas from these packages have been inspirational to them.

REFERENCES

Arya S. and Mount D.M. *ANN: A Library for Approximate Nearest Neighbor Searching. Version 1.1.1.* Dept of Computer Science, University of Maryland, http://www.cs.umd.edu/~mount/ANN/, 2006.

Arya S., Mount D.M., Netanyahu N.S., Silverman R., and Wu A.Y. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

Butenhof D. *Programming with Posix Threads.* Addison-Wesley, 1997. ISBN 0201633922.

Forssell L.K. and Cohen S.D. Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Transactions on Visualization and Computer Graphics*, 01(2):133–141, 1995. ISSN 1077-2626. doi:http://doi.ieeecomputersociety.org/10.1109/2945.468406.

Franck G., Carazo F., Nigro N., Storti M., and D'Elía J. Simulación numérica del flujo alrededor del modelo de Ahmed para un ángulo de inclinación crítico. In *XIV Congreso sobre Métodos Numéricos y sus Aplicaciones, Bariloche, Argentina, Mecánica Computacional Vol. XXIII, 2189–2209.* 2004.

GNU Project. GNU C Library, http://www.gnu.org/software/libc/libc.html. 2007.

Jobard B., Erlebacher G., and Hussaini M.Y. Lagrangian-eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 08(3):211–222, 2002. ISSN 1077-2626. doi:http://doi.ieeecomputersociety.org/10.1109/TVCG.2002.1021575.

Kenwright D.N. and Lane D.A. Interactive time-dependent particle tracing using tetrahedral decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 02(2):120–129, 1996. ISSN 1077-2626. doi:http://doi.ieeecomputersociety.org/10.1109/2945.506224.

Monson D.J., Mateer G.G., and Menter F.R. Boundary-layer transition and global skin friction measurement with an oil-fringe imaging technique. *NASA STI/Recon Technical Report N*, 95:30224–+, 1993.

Sonzogni V., Yommi A., Nigro N., and Storti M. Cfd finite element parallel computations on a

beowulf cluster. In *ECCOMAS 2000*. 2000.

Sonzogni V., Yommi A., Nigro N., and Storti M. A parallel finite element program on a Beowulf Cluster. *Advances in Engineering Software*, 33:427–443, 2002.

Steinman D. Simulated pathline visualization of computed periodic blood flow patterns. *J. Biomech*, 33:623–628, 2000.

Storti M.A., Nigro N.M., Paz R.R., Dalcin L., and Lopez E. A general purpose, parallel, multi-physics FEM program. http://www.cimec.org.ar/petscfem. 2007.

Thompson D. OpenDx-Open Visualization Data Explorer versión 4.4.4, http://www.opendx.org/, http://en.wikipedia.org/wiki/IBM_OpenDX. 2006.

Tobak M. and Peake D.J. Topology of three-dimensional separated flows. *Annual Review of Fluid Mechanics*, 14:61–85, 1982.

van Wijk J.J. Image based flow visualization. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 745–754. ACM Press, New York, NY, USA, 2002. ISBN 1-58113-521-1. doi:http://doi.acm.org/10.1145/566570.566646.

((document-version "texstuff-1.0.13-27-g0d347b7")
(document-date "Mon Jul 30 14:28:12 2007 -0300")
(processed-date "Mon Jul 30 14:29:22 2007 -0300"))