mathematics

MDPI

Article

Numerical Investigation of Freely Falling Objects Using Direct-Forcing Immersed Boundary Method

Cheng-Shu You [1], Ming-Jyh Chern [2], Dedy Zulhidayat Noor [3] and Tzyy-Leng Horng [1,*]

[1] Department of Applied Mathematics, Feng Chia University, Taichung 40724, Taiwan; csyou@mail.fcu.edu.tw
[2] Department of Mechanical Engineering, National Taiwan University of Science and Technology, Taipei 106, Taiwan; mjchern@mail.ntust.edu.tw
[3] Department of Mechanical Engineering, Institute Teknologi Sepuluh Nopember, Surabaya 60111, Indonesia; zulnoor@me.its.ac.id
* Correspondence: tlhorng@fcu.edu.tw

Received: 29 July 2020; Accepted: 15 September 2020; Published: 18 September 2020

Abstract: The fluid-structure interaction of solid objects freely falling in a Newtonian fluid was investigated numerically by direct-forcing immersed boundary (DFIB) method. The Navier–Stokes equations are coupled with equations of motion through virtual force to describe the motion of solid objects. Here, we rigorously derived the equations of motion by taking control-volume integration of momentum equation. The method was validated by a popular numerical test example describing the 2D flow caused by the free fall of a circular disk inside a tank of fluid, as well as 3D experimental measurements in the sedimentation of a sphere. Then, we demonstrated the method by a few more 2D sedimentation examples: (1) free fall of two tandem circular disks showing drafting, kissing and tumbling phenomena; (2) sedimentation of multiple circular disks; (3) free fall of a regular triangle, in which the rotation of solid object is significant; (4) free fall of a dropping ellipse to mimic the falling of a leaf. In the last example, we found rich falling patterns exhibiting fluttering, tumbling, and chaotic falling.

Keywords: fluid-structure interaction; direct-forcing immersed boundary method; equation of motion; circular disk sedimentation; tandem circular disks sedimentation; multiple circular disks sedimentation; falling triangle; falling ellipse

1. Introduction

The falling of light objects, such as feathers or leaves in fluids, driven by gravity and hydrodynamic force, poses an intractability in fluid dynamics. The embedded nonlinearity often results in complicated falling patterns and chaotic trajectories. Several experimental and numerical works have been conducted to study the extensive dynamics of freely falling objects. Experiments conducted by Belmonte et al. [1] and Mahadevan et al. [2] focused on thin flat strips and tumbling cards falling through a fluid in a vertical cell. Through experiments and numerical simulations of 2D incompressible Navier–Stokes equations, Pesavento and Wang [3] analyzed the aerodynamics of a freely falling ellipse, a setup akin to a leaf or business card falling in air.

The investigation into the unsteady aerodynamics of fluttering and tumbling plates formed the core work of Andersen et al. [4]. By varying the thickness-to-width ratio and the dimensionless moment of inertia, they were able to capture the transitions among fluttering, tumbling, and steady fall for the free fall of a plate. Jin and Xu [5] studied the unsteady aerodynamics associated with freely falling elliptical and rectangular plates both experimentally and numerically. In their discovery, they showed that the difference in trajectories between elliptical and rectangular plates was a nuance. The distinction, attributable to the geometry of plates, was observed in the angular velocity, in which

the rectangular plate was found to rotate at a much slower rate than the elliptical one. Though the falling of solids with a slender shape, like ellipse and rectangular plate exhibits richer falling patterns, here, we managed to study this sedimentation hydrodynamics staring from a circular disk to a triangle and, finally, to an ellipse by decreasing the symmetry in shape using our long-term developing direct-forcing immersed boundary (DFIB) method [6–8]. To avoid repetition, the fundamental idea of DFIB method, together with its benchmark testing and error analysis, is referred to Reference [6–8]. Besides free falling of a single solid object, sedimentation of multiple solid particles has long been an interesting and fundamental subject bearing many significant industrial applications, such as paper making, specialty chemicals, petroleum, bioengineering, pharmaceuticals, biomass gasification, and combustion. Even today, it is still under extensive and intensive study theoretically, numerically, and experimentally [9–14]. This motivated us to study sedimentation of multiple circular disks here in order to understand interaction dynamics among solids, besides fluid-solid interaction, during settling.

The rest of this paper is organized as follows. The basic idea of DFIB approach for solving fluid-structure-interaction (FSI) problems is described in Section 2 with the coupled equations of motion for a freely falling solid object rigorously derived by control-volume integration of momentum equations. In Section 3, we validated DFIB method by a popular numerical test example describing the 2D flow caused by the free fall of a circular disk inside a tank of fluid [15], as well as 3D experimental sedimentation measurements of a sphere [16]. In Section 4, sedimentation of two circular disks in tandem and multiple circular disks in arrays, showing drafting, kissing, and tumbling phenomena, was computed to demonstrate our DFIB method. Section 5 shows the computation of free fall of a regular triangle, where the rotation is as significant as displacement in the motion of solid due to the breaking of rotational symmetry possessed by a circular disk. The falling pattern starts to show minor fluttering and tumbling behaviors. The rotational effect is further studied by the free fall of a dropping ellipse to mimic the falling of a leaf. We found its falling patterns exhibiting fluttering, tumbling, and chaotic falling, which are particularly sensitive to the aspect ratio of ellipse and density ratio between solid and fluid. Finally, a conclusion is given at end.

2. Mathematical Model and Numerical Method

In the present work, we employed the DFIB approach developed in Reference [6–8] for all 2D and 3D FSI computations. A virtual force is added to the incompressible Navier–Stokes equations in order to accommodate the interaction between solid and fluid. A solid body, immersed in fluid, is identified by a volume-of-solid (VOS) function, η, which denotes the volume fraction of solid within a numerical cell. For a cell full of solid, η is equal to 1, while it becomes 0 for a fluid-filled cell, as shown in Figure 1. It would be fractional in a cut cell, consisting of both solid and fluid. With this notation, we describe DFIB method as follows.

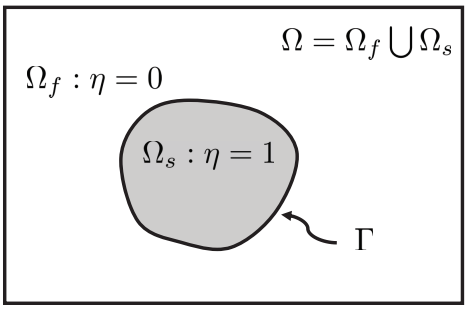


Figure 1. A schematic diagram of solid and fluid distinguished by a volume of solid (VOS) function η.

2.1. DFIB Method

The spirit of DFIB method is treating a rigid solid object immersed in fluid same as ambient fluid but with a prescribed velocity following equations of motion. This idea is achieved by applying an extra virtual force in momentum equations to enforce the fluid within the solid domain to follow

prescribed solid-object velocity [7,8]. The mechanics would then be totally equivalent to a rigid solid object interacting with its surrounding fluid. By doing so, we can avoid using traditional body-fitted methods, such as arbitrary Lagrangian-Eulerian (ALE) method [17–19], which is generally less easy to implement. More specifically, the incompressible Navier–Stokes equations under this framework are expressed as

$$\nabla \cdot \boldsymbol{u} = 0, \tag{1}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u}\boldsymbol{u}) = -\frac{1}{\rho_f}\nabla(p + \rho_f g z) + \nu \nabla^2 \boldsymbol{u} + \eta \boldsymbol{f}, \tag{2}$$

where $\boldsymbol{u}$, ρ_f, p, and ν are velocity, density, pressure, and kinematic viscosity of fluid; g is the gravitational acceleration. Note that, in Equation (2), the virtual force $\boldsymbol{f}$ is exerted only on the solid domain by volume-of-solid function η.

Traditionally, a time-splitting scheme together with pressure increment projection method, implemented by finite difference method in staggered grids, is used to advance velocity from nth to $(n+1)$th time level with $\boldsymbol{u}^{n+1}$ satisfying Equation (1). Here, in DFIB method [6–8], we advance velocity from $\boldsymbol{u}^n$ to the intermediate time level $\boldsymbol{u}^{**}$, instead of $\boldsymbol{u}^{n+1}$, with $\boldsymbol{u}^{**}$ satisfying Equation (1). This is because velocity in solid domain, Ω_s, has not yet satisfied prescribed solid object velocity. To do so, virtual force is recruited for the advancement of $\boldsymbol{u}^{**}$ to $\boldsymbol{u}^{n+1}$ such that $\boldsymbol{u}$ in Ω_s would comply with prescribed solid object velocity,

$$\frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^{**}}{\Delta t} = \eta \boldsymbol{f}^{n+1/2}. \tag{3}$$

Note that, with the virtual force accompanied by VOS, η, in Equation (3), we know this update of velocity to $\boldsymbol{u}^{n+1}$ would only be effective for solid domain. This leaves $\boldsymbol{u}^{n+1} = \boldsymbol{u}^{**}$, satisfying divergence-free condition, for fluid domain. With prescribed velocity for the solid object, denoted by $\boldsymbol{u}_s$, known in advance through equations of motion shown later, we can reciprocally calculate the virtual force by letting $\boldsymbol{u}^{n+1} = \boldsymbol{u}_s^{n+1}$ in Ω_s in Equation (3) and obtain

$$\eta \boldsymbol{f}^{n+1/2} = \frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^{**}}{\Delta t} = \frac{\boldsymbol{u}_s^{n+1} - \boldsymbol{u}^{**}}{\Delta t}. \tag{4}$$

Physically, the negative of volume integration of virtual force, $\eta \boldsymbol{f}$, would be the resultant force exerted on the solid object by fluid:

$$\boldsymbol{F} = -\iiint_{\Omega} \eta \rho_f \boldsymbol{f} \, dV. \tag{5}$$

This computation of resultant force would request complicated surface integration of pressure and viscous stress if we otherwise use traditional body-fitted methods.

2.2. Equations of Motion Governing the Movement of an Immersed Solid

The prescribed velocity at each solid cell $\boldsymbol{u}_s$ in Equation (4) has to follow the equations of motion governing the free-fall dynamics of solid objects. The motion of a solid object is tracked in a Lagrangian frame by linear and angular momentum equations, that delivers center-of-mass and angular velocities for solid object. That means we can decompose $\boldsymbol{u}_s$ at each solid cell into rigid-body translational and rotational components as follows.

$$\boldsymbol{u}_s = \boldsymbol{v}_s + \omega_s \times \boldsymbol{r}, \tag{6}$$

where $\boldsymbol{v}_s$ is the center-of-mass velocity of solid object, and ω_s is its angular velocity around a rotational axis through center of mass. $\boldsymbol{r}$ is the position vector of a solid cell with respect to center of mass. $\boldsymbol{v}_s$ and ω_s further provide position and rotating angle of the solid object by time integration.

To derive an equation for $\boldsymbol{v}_s$, a control volume integration over Ω_s is taken on Equation (2). Note that the control volume integration of pressure gradient and viscous term over Ω_s would be equivalent to the surface integration of stress τ over Γ by divergence theorem, and it ends up as

$$\frac{d\boldsymbol{v}_s}{dt}\iiint_{\Omega_s}\rho_f\,dV=\iint_{\Gamma}\tau\cdot\boldsymbol{n}\,d\boldsymbol{A}+\iiint_{\Omega_s}\rho_f\boldsymbol{g}\,dV+\iiint_{\Omega_s}\rho_f\boldsymbol{f}\,dV. \tag{7}$$

With regard to the solid object, its equation of motion for translation takes the form as

$$\frac{d\boldsymbol{v}_s}{dt}\iiint_{\Omega_s}\rho_s\,dV=\iint_{\Gamma}\tau\cdot\boldsymbol{n}\,d\boldsymbol{A}+\iiint_{\Omega_s}\rho_s\boldsymbol{g}\,dV. \tag{8}$$

Since the surface drag term $\iint_{\Gamma}\tau\cdot\boldsymbol{n}\,d\boldsymbol{A}$ in Equations (7) and (8) is usually unknown a priori, taking a difference between Equations (7) and (8) would eliminate it and result in

$$\frac{d\boldsymbol{v}_s}{dt}\iiint_{\Omega_s}\rho_s\,dV=\iiint_{\Omega_s}(\rho_s-\rho_f)\boldsymbol{g}\,dV-\iiint_{\Omega_s}\rho_f\boldsymbol{f}\,dV+\frac{d\boldsymbol{v}_s}{dt}\iiint_{\Omega_s}\rho_f\,dV, \tag{9}$$

where gravity and buoyancy are denoted by the first term of the right hand side, and hydrodynamic drag is represented by the negative of virtual force in the second term. The last term denotes a fluid inertia term. We can further express Equation (9) as

$$m_s\frac{d\boldsymbol{v}_s}{dt}=(m_s-m_f)\boldsymbol{g}-\iiint_{\Omega}\eta\rho_f\boldsymbol{f}\,dV+m_f\frac{d\boldsymbol{v}_s}{dt}, \tag{10}$$

where m_s and m_f denote the mass of solid object and and its replacement by fluid, respectively, with expressions

$$m_s=\iiint_{\Omega_s}\rho_s\,dV=\iiint_{\Omega}\eta\rho_s\,dV \tag{11}$$

and

$$m_f=\iiint_{\Omega_s}\rho_f\,dV=\iiint_{\Omega}\eta\rho_f\,dV. \tag{12}$$

We can then discretize Equation (10) in time with $\boldsymbol{f}^{n+1/2}\approx\frac{3}{2}\boldsymbol{f}^n-\frac{1}{2}\boldsymbol{f}^{n-1}$, which is equivalent to employ 2nd-order-accurate Adams-Bashforth scheme for the virtual force term, and the equation becomes

$$m_s\frac{\boldsymbol{v}_s^{n+1}-\boldsymbol{v}_s^n}{\Delta t}=(m_s-m_f)\boldsymbol{g}-\left(\frac{3}{2}\iiint_{\Omega_s}\rho_f\boldsymbol{f}^n\,dV-\frac{1}{2}\iiint_{\Omega_s}\rho_f\boldsymbol{f}^{n-1}\,dV\right)+m_f\frac{\boldsymbol{v}_s^n-\boldsymbol{v}_s^{n-1}}{\Delta t}. \tag{13}$$

Note that the fluid inertia term is discretized by lagging with a time step on purpose. Without this lagging, the initial acceleration felt by solid at a quiescent environment would be g instead of the correct $\frac{m_s-m_f}{m_s}g$ on account of buoyancy. This lagging is comprehensible, since flow motion is driven by movement of solid first, as described by Equation (8), and then the response of fluid follows by Equation (7). Equation (10) can be rewritten in short as

$$\frac{d(m_s\boldsymbol{v}_s)}{dt}=\boldsymbol{f}_{gb}+\boldsymbol{f}_d+\frac{d(m_f\boldsymbol{v}_s)}{dt}, \tag{14}$$

where $\boldsymbol{f}_d$ and $\mathbf{f}_{gb}$ represent the hydrodynamic drag and gravity/buoyancy, respectively.

Similarly, to derive an equation for $\boldsymbol{\omega}_s$, we can first take a moment around the axis of rotation through center of mass on Equation (2). Following similar procedures as Equations (7)–(10), we can obtain

$$\frac{d(\boldsymbol{I}_s \cdot \boldsymbol{\omega}_s)}{dt} = \boldsymbol{T}_s + \frac{d(\boldsymbol{I}_f \cdot \boldsymbol{\omega}_s)}{dt}, \tag{15}$$

where $\boldsymbol{I}_s$ and $\boldsymbol{I}_f$ are the moment of inertia for the solid object and its fluid replacement, respectively. The torque $\boldsymbol{T}_s$ acting upon the solid is determined by

$$\boldsymbol{T}_s = -\iiint_{\Omega} \eta \rho_f \boldsymbol{r} \times \boldsymbol{f}\, dV. \tag{16}$$

Equation (16) can be further discretized similar to Equation (13). Once $\boldsymbol{v}_s$ and ω_s are found, $\boldsymbol{u}_s$ can then be determined by Equation (6). The trajectory of every solid cell can be further determined by

$$\frac{d\boldsymbol{X}_s}{dt} = \boldsymbol{v}_s, \tag{17}$$

and then $\Omega_s(t)$ can be determined, as well.

2.3. Collision Model

When we compute the free fall of multiple solid objects immersed in fluid, the collision among solid objects or between solid objects and wall are often inevitable. In computation, the interference (overlapping) among solid objects or between solid objects and wall during falling could happen, though in reality this is often prevented by a thin in-between lubrication fluid layer. Unless our mesh can resolve this thin lubrication layer, which would request a rather exhaustive resolution, collision model is necessary in practice to avoid possible interferences. Glowinski et al. [15] proposed a repulsive force model, in which an artificial short-range repulsive force was introduced to make sure the separation among solid objects and between solid object and wall. Here, we introduce a simple collision model by designing a repulsion force, which would be activated when body-body or body-wall collision is in progress.

For simplicity, we only consider solid objects as circular disks here. Assuming there are N_p disks immersed in the fluid and labeled as $1, 2, \cdots, N_p$, we denote the total repulsive force exerted on disk i by

$$\boldsymbol{F}_i^{co} := \boldsymbol{F}_i^p + \boldsymbol{F}_i^w := \sum_{j=1, j\neq i}^{N_p} \boldsymbol{F}_{ij}^p + \sum_{j=1}^{N_w} \boldsymbol{F}_{ij}^w, \tag{18}$$

where $\boldsymbol{F}_i^p$ is the repulsive force between disk i and the others, and $\boldsymbol{F}_i^w$ is the repulsive force between disk i and walls (labelled as $1, 2, \cdots, N_w$). $\boldsymbol{F}_{ij}^p$ is the disk-to-disk repulsive force that disk i bears from disk j. Similarly, $\boldsymbol{F}_{ij}^w$ is the disk-to-wall repulsive force that disk i bears from wall j.

In the present model, the repulsive force is activated whenever a disk is close enough to another disk or walls by the introduction of small positive parameters $\varepsilon_p, \varepsilon_w$ to control the size of the repulsive force and a small distance tolerance $\delta > 0$ for the activation and adjustment of the force. More specifically, we define the disk-to-disk repulsive force $\boldsymbol{F}_{ij}^p$, acting along center-to-center direction on disk i due to the collision with disk j, by [20]

$$\boldsymbol{F}_{ij}^p = \begin{cases} \boldsymbol{0}, & d_{ij} > R_i + R_j + \delta, \\ \dfrac{c_{ij}}{\varepsilon_p}\left(\dfrac{\boldsymbol{X}_c^{(i)} - \boldsymbol{X}_c^{(j)}}{d_{ij}}\right)\left(\dfrac{R_i + R_j + \delta - d_{ij}}{\delta}\right)^2, & d_{ij} \leq R_i + R_j + \delta, \end{cases} \tag{19}$$

where $\boldsymbol{X}_c^{(i)}$ and R_i are center coordinates and radius of disk i; d_{ij} is the center-to-center distance between disks i and j; c_{ij} is a scaling factor with a dimension of force. In this simulation, c_{ij} is chosen to be the buoyant force acting on disk i. Similarly, the repulsive force with walls $\boldsymbol{F}_{ij}^w$, acting normal to wall j and through center of disk i is given by [20]

$$\boldsymbol{F}_{ij}^{w} = \begin{cases} \mathbf{0}, & d'_{ij} > 2R_i + \delta, \\ \dfrac{c_{ij}}{\varepsilon_w}\left(\dfrac{\boldsymbol{X}_c^{(i)} - \boldsymbol{X}'_{ij}}{d_{ij}}\right)\left(\dfrac{2R_i + \delta - d'_{ij}}{\delta}\right)^2, & d'_{ij} \le 2R_i + \delta, \end{cases} \tag{20}$$

where $\boldsymbol{X}'_{ij}$ is the center coordinates of disk i's mirror image with respect to the wall j; d'_{ij} is the center-to-center distance between disk i and its mirror image against wall j.

2.4. Numerical Procedure

The numerical procedure used to compute fluid flow and trace the position and orientation of falling object at each time step can be summarized in the following steps:

1. Calculate $\boldsymbol{v}_s$ and ω_s via Equations (14) and (15) and then determine $\boldsymbol{u}_s$ for each solid cell via Equation (6).
2. Integrate $\boldsymbol{v}_s$ and ω_s to obtain center-of-mass position $\boldsymbol{X}_s$ and angle of rotation for solid object and then determine solid domain Ω_s.
3. Determine volume of solid function η through Ω_s.
4. Calculate $\boldsymbol{u}^{**}$ via projection method.
5. Advance to $\boldsymbol{u}^{n+1}$ and compute the virtual force by $\boldsymbol{u}_s$ via Equations (3) and (4).

3. Numerical Validations

This section presents benchmark testing of current method by sedimentation of a circular disk for 2D flows [15,20,21] and a sphere for 3D flows [16].

3.1. Sedimentation of a Circular Disk

The numerical simulation describing sedimentation of a circular disk released from rest inside a rectangular enclosure was conducted following the physical setting and geometric configuration of Glowinski et al. [15]. More specifically, the computational domain is $\Omega = (0\text{ cm}, 2\text{ cm}) \times (0\text{ cm}, 6\text{ cm})$ and the disk is located at $(1\text{ cm}, 4\text{ cm})$ initially. The densities of fluid and solid are $\rho_f = 1\text{ g/cm}^3$ and $\rho_s = 1.25\text{ g/cm}^3$, respectively. The diameter of the disk is $D = 0.25$ cm and the kinematic viscosity of the fluid is $\nu = 0.1\text{ cm}^2/\text{s}$. The gravity is $\boldsymbol{g} = -981\text{ cm/s}^2$. In current simulations, the computational domain is discretized by a uniform mesh with the mesh size $\Delta x = \Delta y = 0.025$ cm. To capture the trajectory and orientation of a free-fall solid object accurately, we deliberately choose a rather small time step in current study, which is much smaller than the time step requested by CFL constraint. For the current case, we chose $\Delta t = 10^{-4}$ s.

Simulated flow field featured by vorticity contours at various times is shown in Figure 2. Time traces of disk's vertical position and velocity were particularly calculated and shown in Figure 3. Figure 3a,b shows the grid-independence of current computations. Basically, results under different mesh sizes coincide with each other except the velocity near the hitting of floor in Figure 3b. Here, we further adopt the result with the finest mesh size in Figure 3a,b to compare with [15,20,21] and show it in Figure 3c,d. Our result generally bears a good agreement with references mentioned above, especially with Glowinski et al. [15]. This can be told from the immediate velocity near the hitting of floor in Figure 3d, where actually none of [15,20,21] is close to one another. From Figure 3d, the disk is first accelerated by gravity, and then the fluid drag increases as disk's velocity increases. When gravity is finally balanced by drag, the velocity of disk reaches its maximum terminal velocity, which is steady and not altered until hitting the floor. Note that the peak followed by oscillations in Figure 3d, happening near the hitting of floor in our computation, is because of the usage of collision model, described in Section 2.3. The peak and following oscillations imply the damping rebounds from the floor. The collision model is a necessary evil. Without it, falling solid object would generally penetrate the floor numerically.

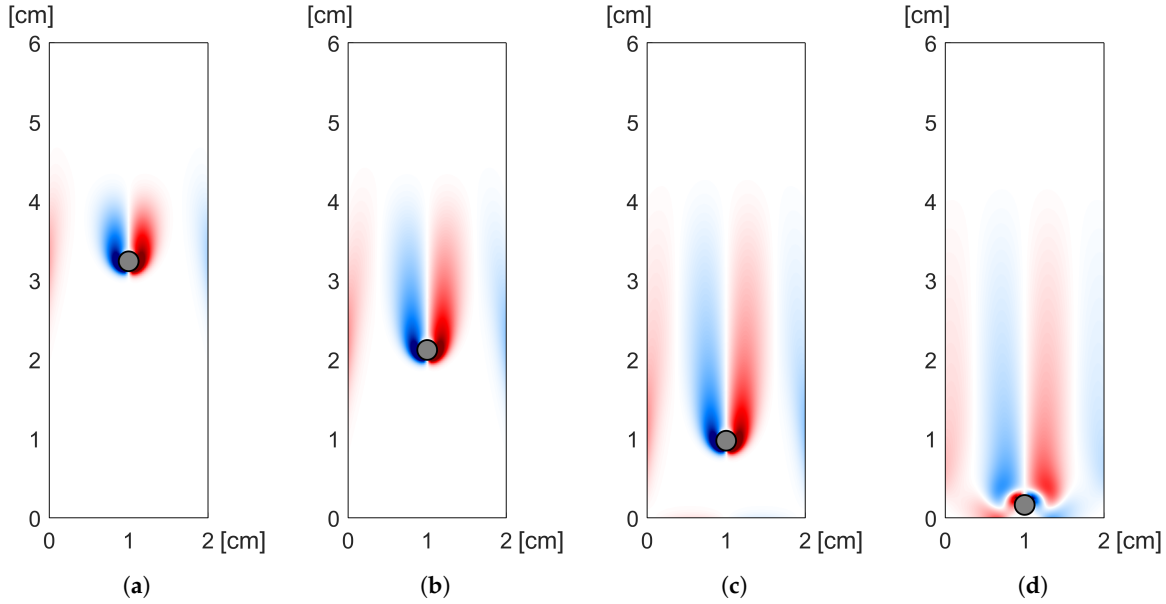


Figure 2. Flow field featured by vorticity contours at time (**a**) $t = 0.2$ s, (**b**) $t = 0.4$ s, (**c**) $t = 0.6$ s, and (**d**) $t = 0.8$ s.

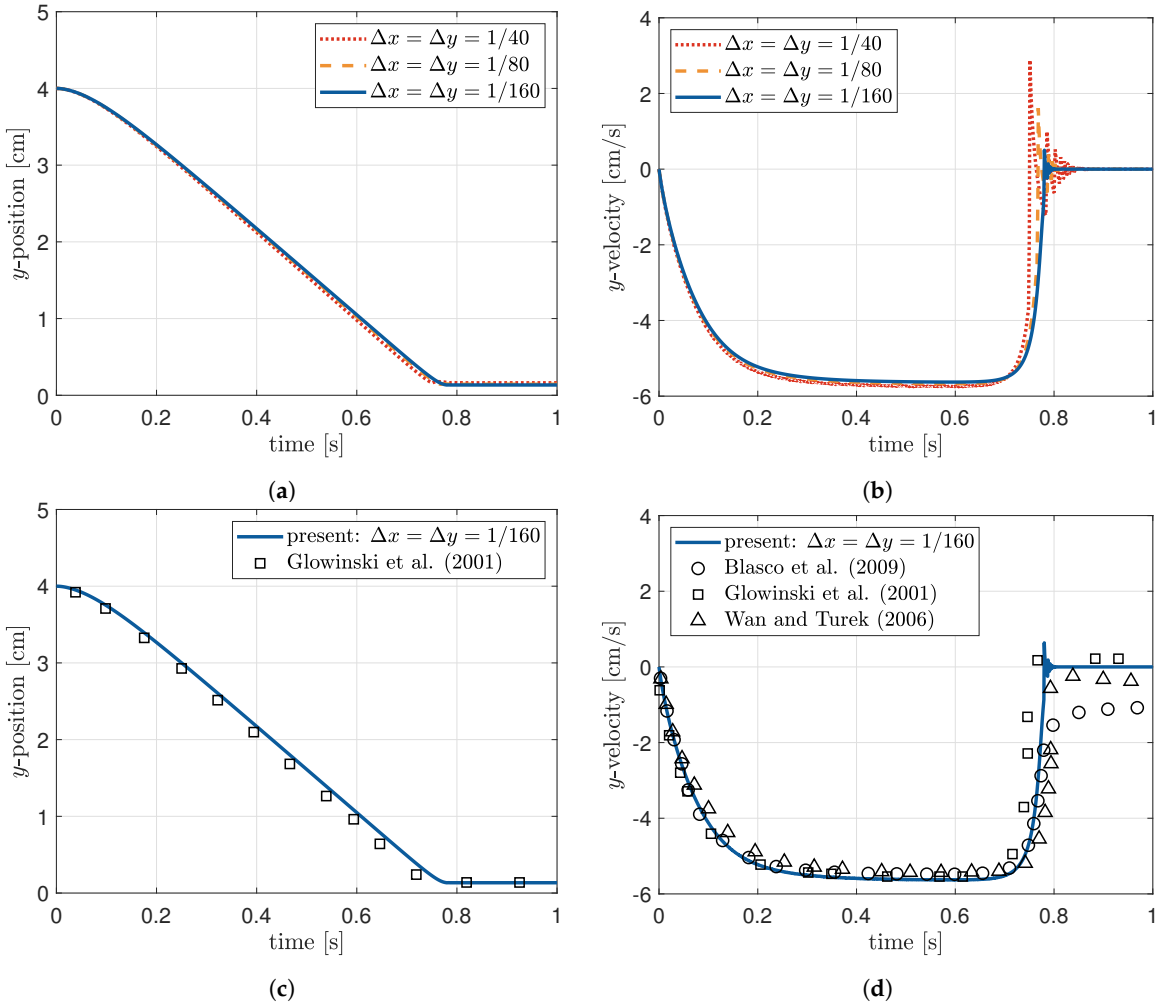
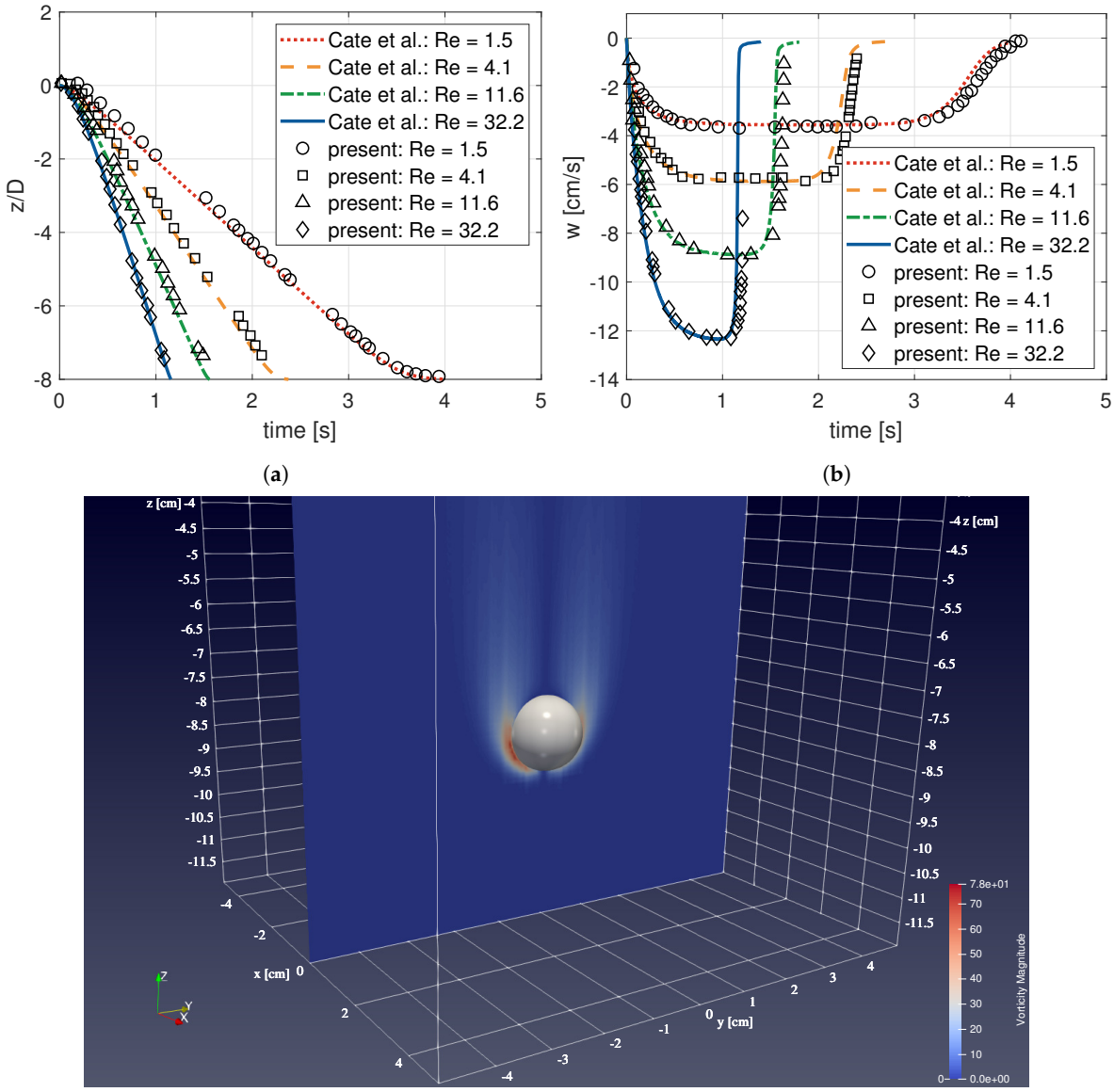


Figure 3. Time traces of (**a**) y-position and (**b**) y-velocity of circular disk showing grid independence. Time traces of (**c**) y-position and (**d**) y-velocity of circular disk compared with Reference [15,20,21].

3.2. Sedimentation of a 3D Sphere

Here, the numerical simulation of sedimentation of a 3D sphere inside a fluid tank was conducted with physical parameters and geometry settings same as the experiment reported by Cate et al. [16]. The computational domain is set to $\Omega = (-5\,\text{cm}, 5\,\text{cm}) \times (-5\,\text{cm}, 5\,\text{cm}) \times (-12\,\text{cm}, 4\,\text{cm})$ based on the rectangular fluid tank of size 10 cm $\times$ 10 cm $\times$ 16 cm mentioned in experiment. A sphere of diameter $D = 1.5$ cm is initially centered at $(0\,\text{cm}, 0\,\text{cm}, 0.75\,\text{cm})$ and released from rest with its density being $\rho_s = 1120\,\text{kg/m}^3$, slightly larger than the density of ambient fluid. Following the experiment, various fluid density, ρ_f, and dynamic viscosity, μ_f, were employed for simulations and are listed in Table 1 with Reynolds number defined as $Re = \rho_f \boldsymbol{U}_\infty D/\mu_f$, in which $\boldsymbol{U}_\infty$ is the terminal velocity. We conducted the numerical simulations in a uniform Cartesian grid with $\Delta x = \Delta y = \Delta z = 0.05$ cm, time step $\Delta t = 10^{-5}$ s, and the results are depicted in Figure 4, where the time traces of sphere's vertical position and velocity for each case are shown in Figure 4a,b, and a snap shot of vorticity magnitude contours for the falling sphere is shown in Figure 4c. We find that our numerical results are in excellent agreements with the experimental data in Reference [16].



(**a**) (**b**)

(**c**)

Figure 4. Time traces of sphere's (**a**) vertical position and (**b**) vertical velocity compared with [16]. (**c**) A snapshot of vorticity magnitude contours at $t = 0.9$ with $Re = 32.2$.

Table 1. Physical parameters used to simulate the sedimentation of a 3D sphere based on [16].

	ρ_f [kg/m^3]	μ_f [Ns/ m^2]	U_∞ [m/s]	Re
Case 1	970	3.73	3.8	1.5
Case 2	965	2.12	6.0	4.1
Case 3	962	1.13	11.5	11.6
Case 4	960	0.58	12.8	32.2

4. More Sedimentation Simulations by DFIB Method

Here, we further computed the sedimentation of (1) two tandem circular disks and (2) multiple circular disks to demonstrate current method.

4.1. Sedimentation of Two Tandem Circular Disks

The sedimentation of two tandem circular disks inside a rectangular enclosure is studied here with the densities of disk and fluid being $\rho_s = 1.5$ g/cm^3 and $\rho_f = 1$ g/cm^3, respectively. The kinematic viscosity of fluid is set as $\nu = 0.01$ cm^2/s. The rectangular enclosure is 2 cm wide and 6 cm long, and the diameter of the disks is $D = 0.25$ cm. These two tandem disks are located initially at the top of the rectangular enclosure with the distance between centers of disks being 0.5 cm, and so the gap between two disks is 0.25 cm. A uniform mesh of size $\Delta x = \Delta y = 1/128$ cm, and time step $\Delta t = 10^{-5}$ s are used in current simulation. The collision parameter are set as $\varepsilon_w = \varepsilon_p = 10^{-5}$, and the distance tolerance $\delta = 2\Delta x$.

Figure 5 displays movements of two tandem disks with vorticity contours used for flow visualization. From the outset, these two disks fall in tandem. Eventually, the upper disk falls faster than the lower one due to the wake of the lower disk, since the low pressure in lower disk's wake accelerates the upper one. The hydrodynamic force affects the lower disk as the upper one approaches, and causes the lower disk to drift a little sideways. Ultimately, the upper disk catches up with the lower one, and the two disks collide with each other in what is referred to as the kissing stage. Afterwards, the two disks separate, and then tumble and fall in a side-by-side way. The formerly lower disk is then pushed sideways and upwards by the tumbling motion, and swaps place with the formerly upper disk, which now becomes the lower one. If the rectangular enclosure is long enough, these events would be repeated. The modes of motion delineated above are classified as drafting, kissing and tumbling, which was first observed experimentally by Fortes et al. [22] and simulated in two dimensions by Hu et al. [23], Glowinski et al. [15], and Choi [24]. These well-known phenomena are also predicted by the current DFIB method as shown in Figure 5.

4.2. Sedimentation of Multiple Circular Disks

In order to demonstrate the ability of handling complicated collisions among multiple solid objects by current DFIB method, we also computed the sedimentation of multiple circular disks here. A set of 10×10 identical circular disks are initially placed at the top of a rectangular cavity filled with incompressible Newtonian fluid, released from rest, fall by gravity, and settle at bottom of cavity at last. The parameters and geometry configuration are all the same as the previous example in Section 4.1, except the diameter of disk reduced to be $D = 0.15$ cm. The complicated sedimentation process of multiple disks is successfully simulated here, and the result is shown in Figure 6, from which we can observe those drafting, kissing, and tumbling phenomena mentioned above keep happening among disks during sedimentation.

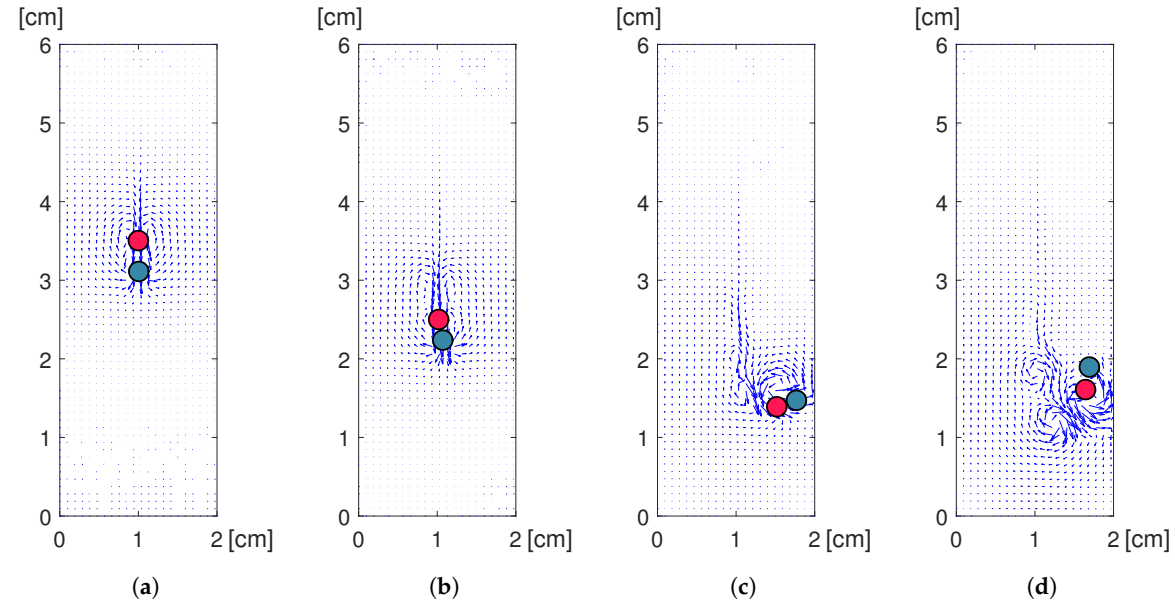


Figure 5. The positions of two tandem circular disks and the associated flow velocity field at time (**a**) $t = 0.15$ s, (**b**) $t = 0.20$ s, (**c**) $t = 0.25$ s, and (**d**) $t = 0.3$ s.

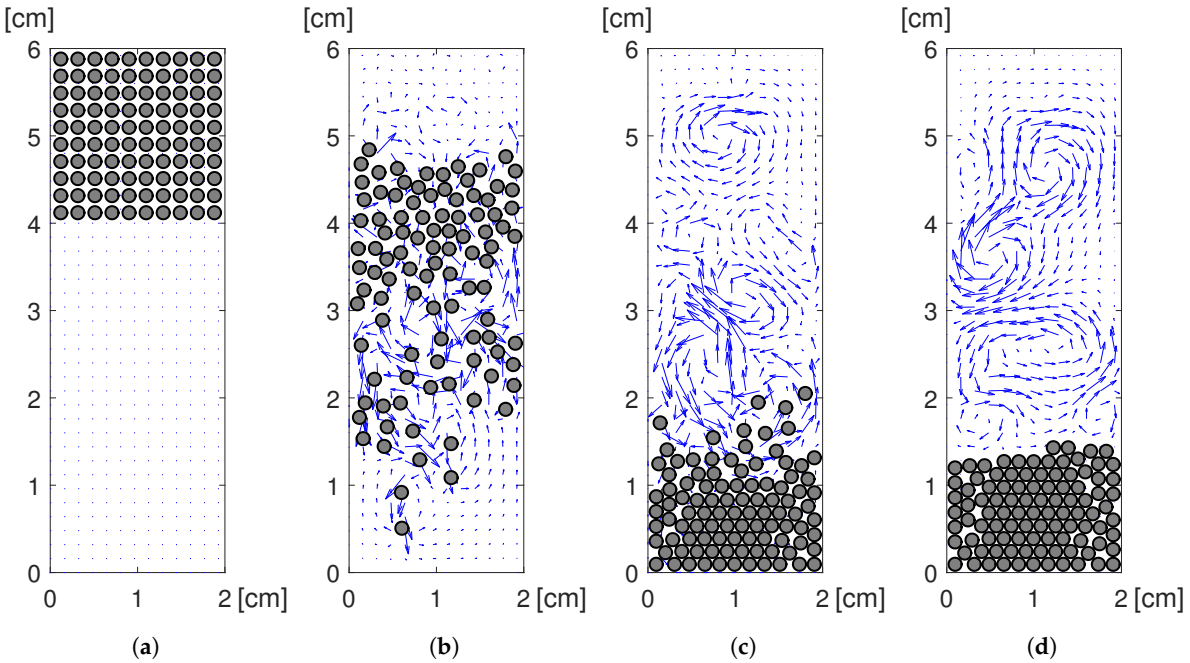


Figure 6. The positions of 10×10 circular disks during sedimentation and the associated flow velocity fields at time (**a**) $t = 0$ s, (**b**) $t = 0.6$ s, (**c**) $t = 1.5$ s, and (**d**) $t = 2$ s.

5. Sedimentation of Solid Objects without Rotational Symmetry

5.1. Sedimentation of a Regular Triangle

In this example, we consider the sedimentation of a equilateral triangle with edge length $\sqrt{3}D/2$, where D is the diameter of the circumscribed circle of triangle, in an incompressible viscous fluid. D is 1 cm here. The regular triangle is initially placed at rest with the center-of-mass position at $(15\text{ cm}, 28\text{ cm})$ in a 30 cm $\times$ 30 cm rectangular domain, with an initial angle of attack (AOA), $\alpha_{att} = 5°$. Note that this small AOA would not affect the asymptotic falling pattern of triangle, but is meant to break the symmetry in flow field in order to accelerate the development of its falling pattern.

Otherwise, a regular triangle, placed with symmetry initially, would just steadily fall all the way down, since the vertical span is not long enough for disturbances to kick in and break flow symmetry. All the physical parameters are set to be the same as the previous example in Section 4.1. A uniform mesh of size $\Delta x = \Delta y = 1/64$ cm, and time step $\Delta t = 10^{-4}$ s are employed in current simulation.

The numerical results are displayed in Figure 7. Figure 7a shows triangle's trajectory and orientation, and Figure 7b–d shows the snap shots of flow field at certain times during the falling. From that, we can tell the falling pattern is fluttering at the beginning, followed by tumbling at some tipping point. A MATLAB code for current sedimentation simulation of a regular triangle is attached at Appendix A for readers further interested in this problem.

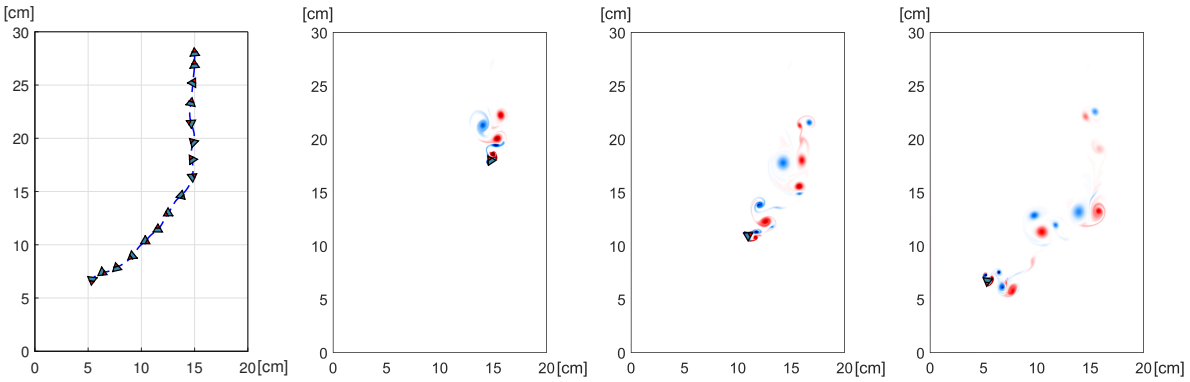


Figure 7. (**a**) Falling trajectory and orientation of a regular triangle. Flow field featured by vorticity contours at time (**b**) $t = 1.2$ s, (**c**) $t = 2.1$ s, and (**d**) $t = 3$ s.

5.2. A Freely Falling Ellipse

It is a common fact that not all objects fall straight down by gravity. For example, a leaf, feather and paper card all fall in a seemingly unpredictable manner. From time to time, a leaf or feather may reverse its falling direction and momentarily rise against the gravity as it flutters or tumbles through air. This rich dynamical behavior has been investigated in many experimental and modeling works, e.g., Reference [1,3,5,25,26]. Numerical simulations regarding a freely falling rectangular or elliptical plate, to mimic a falling leaf or feather, have been studied before in Reference [3,5,25,26]. Taking 2D ellipse as an example, its dynamics can be characterized by the following three dimensionless parameters [3,25]:

- Reynolds number, $Re^* = \boldsymbol{u}_t^* a/\nu$, with $\boldsymbol{u}_t^* = \sqrt{2b\boldsymbol{g}(\rho_s/\rho_f - 1)}$,
- the dimensionless moment of inertia, $I^* = \left(\rho_s b(a^2 + b^2)\right)/(2\rho_f a^3)$,
- the aspect ratio of the ellipse, $e = a/b$, where a and b are the width and thickness of the ellipse, respectively.

Re^*, I^*, and e can be recast to I, Re, ρ_s/ρ_f, and e, where $I = \left(b(a^2 + b^2)\right)/(2a^3)$, $Re = \boldsymbol{u}_t b/\nu$, with $\boldsymbol{u}_t = \sqrt{b\boldsymbol{g}}$. The latter set of parameters is more physically intuitive. Note that the dimensionless moment of inertia without density ratio, I, is actually equivalent to $(e^2 + 1)/(2e^3)$, which means I solely depends on e.

Here, we use the current DFIB method to study the falling patterns of a 2D ellipse. The ellipse is initially placed at rest with its center of mass positioned at $(5\,\text{cm}, 28\,\text{cm})$ in a 30 cm $\times$ 30 cm rectangular domain, with an initial AOA, $\alpha_{att} = 45°$. A uniform mesh of size $\Delta x = \Delta y = 1/100$ cm, and time step $\Delta t = 10^{-5}$ s were employed in current simulation. We fixed the density of fluid to be $\rho_f = 1.0\,\text{g/cm}^3$ and changed ρ_s to vary the density ratio between solid and fluid. We also fixed $b = 0.162$ cm and varied a to generate various aspect ratios e. Through exhaustive surveys, three combinations of dimensionless parameters as shown in Table 2 were employed in simulations to demonstrate three typical falling patterns: fluttering, chaotic falling, and tumbling [3,25]. The trajectory

of ellipse together with its orientation for these three falling patterns was calculated and is shown in Figure 8. The associated vortex shedding and wake structure of these three falling patterns are shown respectively in Figures 9–11.

Table 2. Physical parameters employed for three falling patterns of ellipse.

	Re^*	I^*	e	ρ_s/ρ_f	ν [cm^2/s]	Re
fluttering	1000	0.08	8	1.2603	0.0118	173.2413
chaotic falling	1000	0.16	8	2.5206	0.0285	71.6779
tumbling	1000	0.25	8	3.9385	0.0396	51.5626

In Table 2, e is equally set to 8. This is because we found a falling ellipse is more probable to exhibit all three typical falling patterns at a larger e. In addition, the equal setting of e implies an equal setting of I, as well. We further discovered the falling pattern is more sensitive to density ratio, ρ_s/ρ_f, and less sensitive to viscosity ν. By increasing the density ratio gradually, we explored fluttering, chaotic falling, and tumbling, respectively. Generally larger density ratio implies larger driving force and torque, which tends to rotate the ellipse more severely during falling. This can be seen from maximum AOA hardly reaching 90° in fluttering as shown in Figure 8a, but its counterparts in chaotic falling and tumbling can pass 90° during falling as shown in Figure 8b,c.

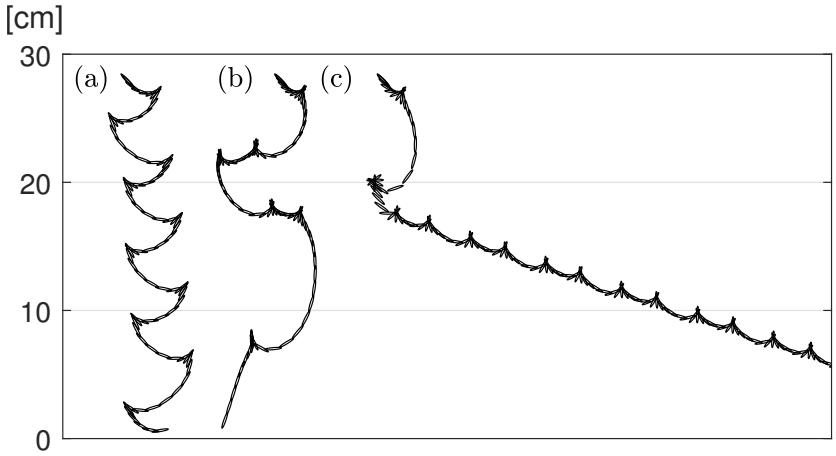


Figure 8. Falling trajectory and orientation of ellipse. (**a**) Fluttering, (**b**) chaotic falling, (**c**) tumbling.

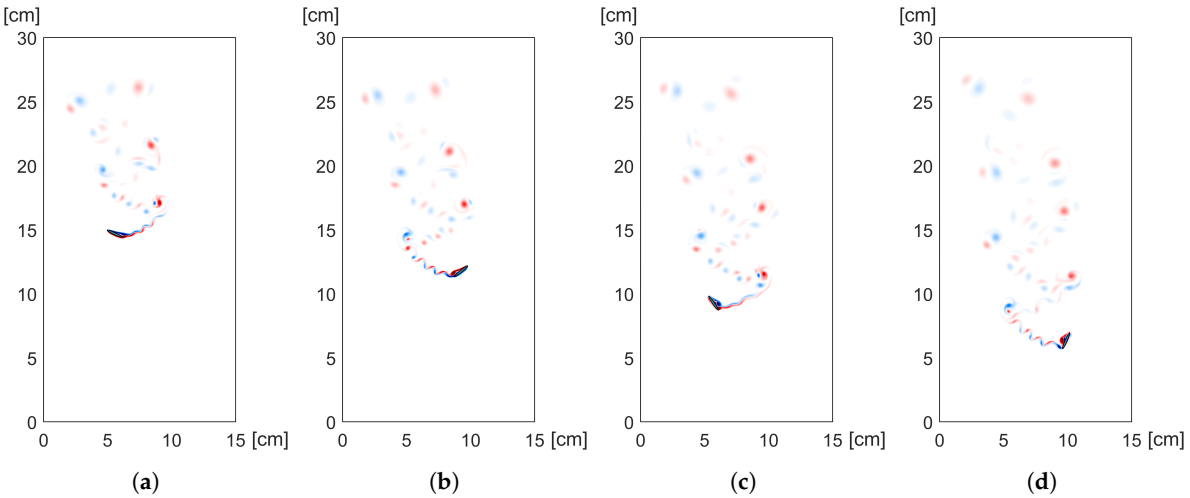


Figure 9. Wake structure in vorticity contours at time (**a**) $t = 2.5$ s (**b**) $t = 3$ s, (**c**) $t = 3.5$ s, and (**d**) $t = 4$ s. This corresponds to the fluttering mode in Figure 8a.

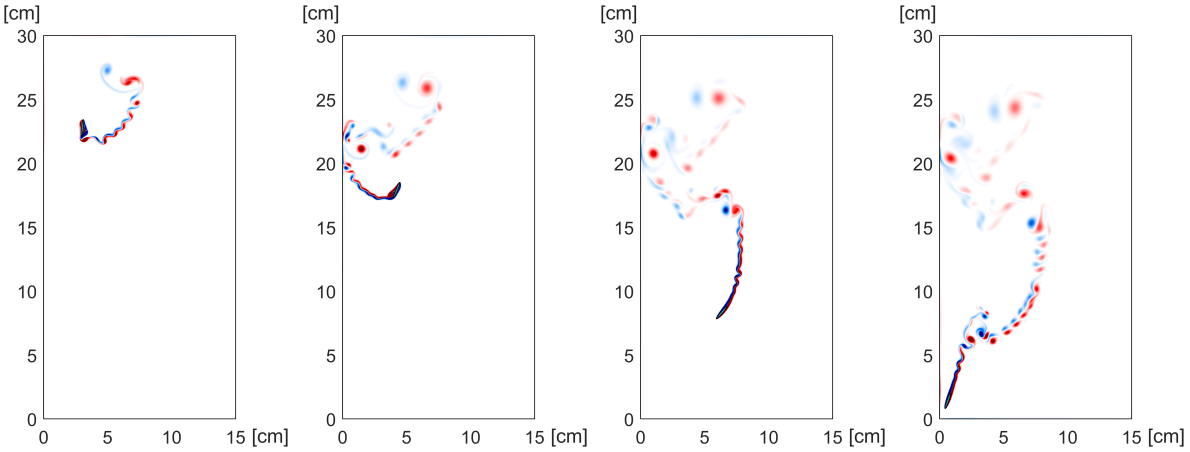


Figure 10. Wake structure in vorticity contours at time (**a**) $t = 0.4$ s, (**b**) $t = 0.8$ s, (**c**) $t = 1.2$ s, and (**d**) $t = 1.5$ s. This corresponds to the chaotic falling mode in Figure 8b.

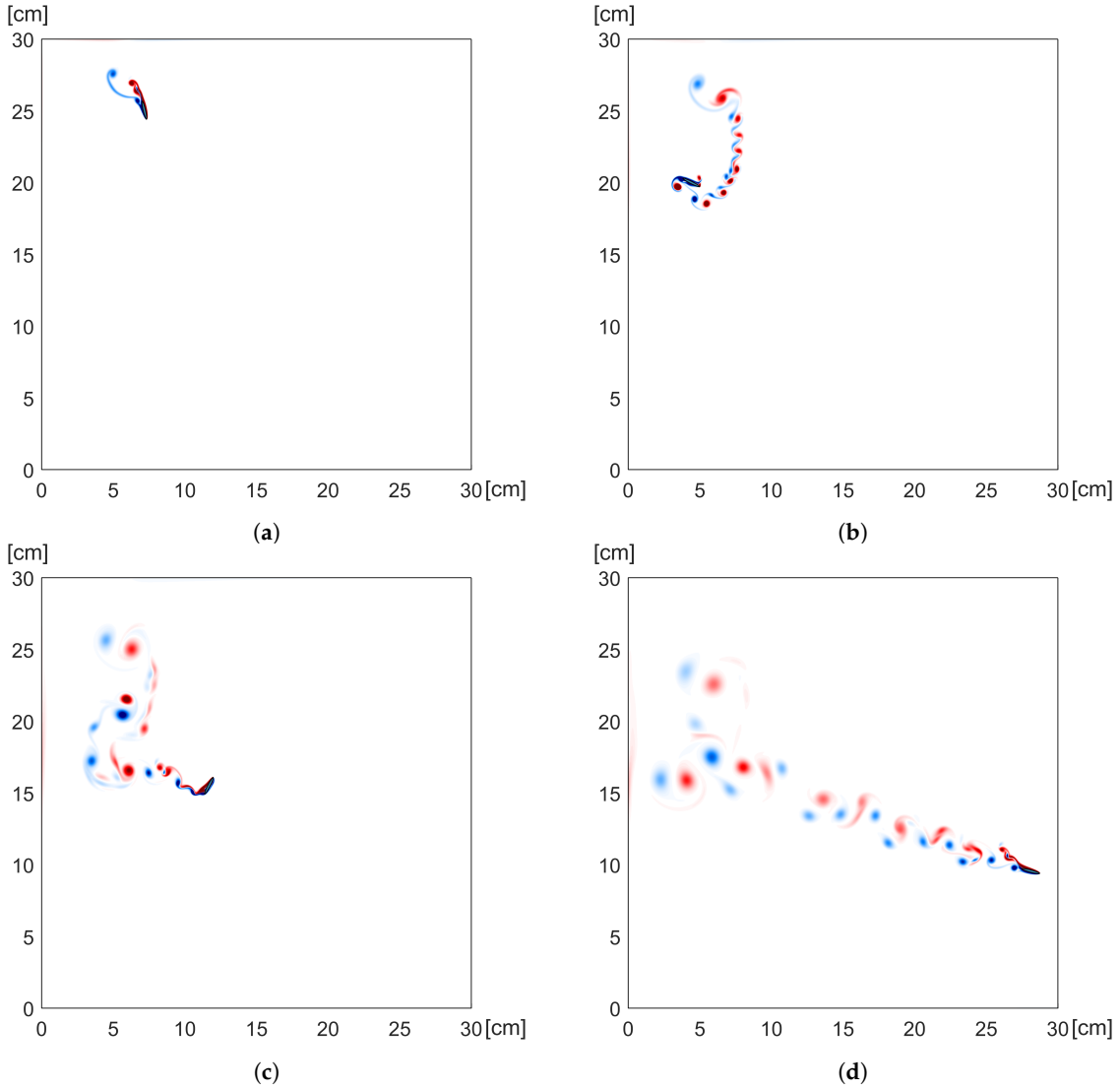


Figure 11. Wake structure in vorticity contours at time (**a**) $t = 0.2$ s, (**b**) $t = 0.4$ s, (**c**) $t = 0.8$ s, and (**d**) $t = 1.6$ s. This corresponds to the tumbling mode in Figure 8c.

6. Summary and Conclusions

The current DFIB approach successfully predicted the interaction between falling solid objects and fluid through simulation cases demonstrated above, with some of them validated with the literature. The method is efficient to calculate the resultant force and torque exerted on the solid object and determine its displacement and orientation at each time step. Among all sedimentation cases studied here, rotation becomes important for the sedimentation of a regular triangle and an ellipse due to the breaking of rotational symmetry. The sedimentation of a regular triangle, which is rarely discussed in the literature, shows a falling pattern of fluttering at the beginning, followed by tumbling starting at some tipping point. The sedimentation of an ellipse has the richest dynamics here. With various combinations of physical parameters, our simulation exhibits three typical falling patterns: fluttering, tumbling, and chaotic falling. Animations of sedimentation simulations in current paper can be found at https://www.youtube.com/channel/UC9Qz6kOB-WVFnwijBHZRfVA/.

Author Contributions: T.-L.H. and M.-J.C. chiefly contributed to numerical method development and paper writing; C.-S.Y. and D.Z.N. mainly contributed to numerical simulations. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Science and Technology of Taiwan with grant numbers: MOST 107-2115-M-035-007-MY2 (Cheng-Shu You), MOST 107-2221-E-011-075-MY3 (Ming-Jyh Chern), and MOST 108-2115-M-035-002-MY2 (Tzyy-Leng Horng).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

A MATLAB code based on present DFIB method for the case of sedimentation of a regular triangle discussed in Section 5.1 is provided as follows.

Main program:

```
% Initiallize the parameters of the simulation
nu = 0.01; % cm^2/s
% Create the mesh
lx = 30; ly = 30; n = 64;
Th = RectangleMesh([0, lx, 0, ly], lx*n, ly*n);
% Create the triangle object
r = 0.5; % radius of the circumscribed circle
edge_length = sqrt(3)*r;
angle_of_attack = 5*pi/180;
center_of_mass = [15, 28];
rho_s = 1.5;
tri = TriangleObject(Th, center_of_mass, edge_length, r, angle_of_attack, rho_s);
% Time setting
dt = 1e-4;
nt = 50000;
% Boundary condition
UN = 0; VN = 0;
US = 0; VS = 0;
UE = 0; VW = 0;
UW = 0; VE = 0;
% Initialize
[u, v, p] = initialize(Th, UW, UE, US, UN, VW, VE, VS, VN);
u_old = u; v_old = v;
% DFIB
curtime = 0;
```

```
for iter = 1:nt
    curtime = curtime + dt;
    fprintf('current time: %f\n', curtime)
    % Move the object to next time level
    tri.move(dt);
    % Preprocessing
    dpdx = diff(p, 1, 1)/Th.hx; dpdy = diff(p, 1, 2)/Th.hy;
    [H1, H2] = H(u, v, nu, Th.hx, Th.hy);
    [H1_old, H2_old] = H(u_old, v_old, nu, Th.hx, Th.hy);
    % Solve the intermediate velocity
    u_old = u; v_old = v;
    u(2:end-1, 2:end-1) = u(2:end-1, 2:end-1) + dt*(-dpdx + 1.5*H1 - 0.5*H1_old);
    v(2:end-1, 2:end-1) = v(2:end-1, 2:end-1) + dt*(-dpdy + 1.5*H2 - 0.5*H2_old);
    u = updateBC_u(u, UW, UE, US, UN); v = updateBC_v(v, VW, VE, VS, VN);
    % Solve the Poisson system
    S = diff(u(:, 2:end-1), 1, 1)/Th.hx + diff(v(2:end-1, :), 1, 2)/Th.hy;
    phi = FFT_Poisson_solver(S/dt, Th.hx, Th.hy);
    % Projection step
    u(2:end-1, 2:end-1) = u(2:end-1, 2:end-1) - dt*diff(phi, 1, 1)/Th.hx;
    v(2:end-1, 2:end-1) = v(2:end-1, 2:end-1) - dt*diff(phi, 1, 2)/Th.hy;
    % update pressure
    p = p + phi;
    % update virtual force
    tri.computeVirtualForce(u, v, dt);
    % update velocity
    u(:, 2:end-1) = tri.eta1 .* tri.us + (1-tri.eta1).*u(:, 2:end-1);
    v(2:end-1, :) = tri.eta2 .* tri.vs + (1-tri.eta2).*v(2:end-1, :);
end
```

Subroutines:

```
function [u, v, p] = initialize(Th, UW, UE, US, UN, VW, VE, VS, VN)
u = zeros(Th.nx+1, Th.ny+2); v = zeros(Th.nx+2, Th.ny+1); p = zeros(Th.nx, Th.ny);
u = updateBC_u(u, UW, UE, US, UN); v = updateBC_v(v, VW, VE, VS, VN);
end

function [H1, H2] = H(u, v, nu, hx, hy)
[uconv, vconv] = convection_div(u, v, hx, hy);
ulap = laplacian(u, hx, hy); vlap = laplacian(v, hx, hy);
H1 = nu*ulap - uconv; H2 = nu*vlap - vconv;
end

function [Nu, Nv] = convection_div(u, v, hx, hy)
uce = (u(1:end-1,2:end-1) + u(2:end,2:end-1))/2;
uco = (u(:,1:end-1) + u(:,2:end))/2;
vco = (v(1:end-1,:) + v(2:end,:))/2;
vce = (v(2:end-1,1:end-1) + v(2:end-1,2:end))/2;
uuce = uce.*uce; uvco = uco.*vco; vvce = vce.*vce;
Nu = (uuce(2:end,:) - uuce(1:end-1,:))/hx ...
   + (uvco(2:end-1,2:end) - uvco(2:end-1,1:end-1))/hy;
Nv = (vvce(:,2:end) - vvce(:,1:end-1))/hy ...
```

```
   + (uvco(2:end,2:end-1) - uvco(1:end-1,2:end-1))/hx;
end

function lap = laplacian(f, hx, hy)
[m, n] = size(f);
i = 2:m-1; j = 2:n-1;
lap = (f(i+1, j) + f(i-1, j) - 2*f(i,j))/(hx*hx) ...
    + (f(i, j+1) + f(i, j-1) - 2*f(i,j))/(hy*hy);
end

function u = updateBC_u(u, UW, UE, US, UN)
u(1, :) = UW; u(end, :) = UE;
u(:, 1) = (8/3)*US - 2*u(:, 2) + (1/3)*u(:, 3);
u(:, end) = (8/3)*UN - 2*u(:, end-1) + (1/3)*u(:, end-2);
end

function v = updateBC_v(v, VW, VE, VS, VN)
v(:, 1) = VS; v(:, end) = VN;
v(1, :) = (8/3)*VW - 2*v(2, :) + (1/3)*v(3, :);
v(end, :) = (8/3)*VE - 2*v(end-1, :) + (1/3)*v(end-2, :);
end

function uh = FFT_Poisson_solver(fh, hx, hy)
[M, N] = size(fh);
demon = -(2*sin(pi*(0:M-1)'*ones(1, N)./(2*M))./hx).^2 ...
        -(2*sin(pi*ones(M, 1)*(0:N-1)./(2*N))./hy).^2;
fhat = dct(dct(fh)')';
uhat = fhat./demon; uhat(1,1) = 0;
uh = idct(idct(uhat)')';
end

classdef RectangleMesh < handle
% RectangleMesh Mesh of the 2D rectangle (x0, y0) x (x1, y1)
properties (SetAccess='private')
    domain;
    x, y, xc, yc;
    Xu, Yu, Xv, Yv, Xc, Yc;
    nx, ny, hx, hy;
end

methods
    function obj = RectangleMesh(domain, nx, ny)
        obj.domain = domain;
        obj.x = linspace(domain(1), domain(2), nx + 1);
        obj.y = linspace(domain(3), domain(4), ny + 1);
        obj.xc = avg(obj.x); obj.yc = avg(obj.y);
        obj.nx = nx; obj.ny = ny;
        obj.hx = (domain(2) - domain(1)) / nx;
        obj.hy = (domain(4) - domain(3)) / ny;
        [obj.Xu, obj.Yu] = ndgrid(obj.x, obj.yc);
```

```
        [obj.Xv, obj.Yv] = ndgrid(obj.xc, obj.y);
        [obj.Xc, obj.Yc] = ndgrid(obj.xc, obj.yc);
    end
end
end
```

Class of regular triangle:

```
classdef TriangleObject < handle
% Class of Regular triangle
properties
    Th, center_of_mass, edge_length, r, angle_of_attack, density;
    vx, vy, vel, vel_old;
    omega, omega_old; theta, theta_old, us, vs;
    eta0, eta1, eta2, eta0_old, eta1_old, eta2_old;
    forceX, forceY, forceX_old, forceY_old;
    gravity = [0, -981];
end

methods
    function obj = TriangleObject(Th, center_of_mass, edge_length, r, ...
                                  angle_of_attack, density)
        obj.Th = Th;
        obj.center_of_mass = center_of_mass;
        obj.edge_length = edge_length;
        obj.r = r;
        obj.angle_of_attack = angle_of_attack;
        obj.density = density;
        obj.initialize();
        obj.updateEta();
    end

    function move(obj, dt)
        % Compute translational motion
        obj.translational_motion(dt);
        % Compute rotation motion
        obj.rotational_motion(dt)
        % Compute center_of_mass at next time level
        obj.center_of_mass = obj.center_of_mass + 0.5*dt*(obj.vel + obj.vel_old);
        % Compute theta at next time level
        obj.theta = obj.theta + 0.5*dt*(obj.omega + obj.omega_old);
        % update velocity
        r1 = obj.Th.Xv - obj.center_of_mass(1);
        r2 = obj.Th.Yu - obj.center_of_mass(2);
        obj.us = obj.vel(1) - obj.omega*r2; obj.vs = obj.vel(2) + obj.omega*r1;
        % Update body position
        obj.update_position();
        % Update eta
        obj.updateEta();
    end
```

```
    function computeVirtualForce(obj, u, v, dt)
        obj.forceX_old = obj.forceX; obj.forceY_old = obj.forceY;
        obj.forceX = obj.eta1.*(obj.us - u(:, 2:end-1))/dt;
        obj.forceY = obj.eta2.*(obj.vs - v(2:end-1, :))/dt;
    end
end

methods(Access='private')
    function initialize(obj)
        obj.forceX = zeros(obj.Th.nx+1, obj.Th.ny);
        obj.forceY = zeros(obj.Th.nx, obj.Th.ny+1);
        obj.forceX_old = obj.forceX; obj.forceY_old = obj.forceY;
        vx0 = [-obj.r*cos(pi/6), obj.r*cos(pi/6), 0];
        vy0 = [-obj.r*sin(pi/6), -obj.r*sin(pi/6), obj.r];
        obj.vx = obj.center_of_mass(1) ...
               + (vx0.*cos(obj.angle_of_attack) - vy0.*sin(obj.angle_of_attack));
        obj.vy = obj.center_of_mass(2) ...
               + (vx0.*sin(obj.angle_of_attack) + vy0.*cos(obj.angle_of_attack));
        obj.vel = [0, 0]; obj.vel_old = [0, 0];
        obj.omega = 0; obj.omega_old = 0; % rad/s
        obj.theta = 0; obj.theta_old = 0;
        obj.us = 0; obj.vs = 0;
    end

    function translational_motion(obj, dt)
        hx = obj.Th.hx; hy = obj.Th.hy;
        HFX = trapz(trapz(obj.forceX,1)*hx, 2)*hy;
        HFY = trapz(trapz(obj.forceY,1)*hx, 2)*hy;
        HFX_old = trapz(trapz(obj.forceX_old, 1)*hx, 2)*hy;
        HFY_old = trapz(trapz(obj.forceY_old, 1)*hx, 2)*hy;
        area = (sqrt(3)*obj.edge_length*obj.edge_length)/4;
        Ms = obj.density*area; Mf = 1.0*area;
        vel_1_ = obj.vel(1) + dt*((1-Mf/Ms)*obj.gravity(1) ...
               - (1.5*HFX - 0.5*HFX_old)/Ms ...
               + (Mf/Ms)*(obj.vel(1) - obj.vel_old(1))/dt);
        vel_2_ = obj.vel(2) + dt*((1-Mf/Ms)*obj.gravity(2) ...
               - (1.5*HFY - 0.5*HFY_old)/Ms ...
               + (Mf/Ms)*(obj.vel(2) - obj.vel_old(2))/dt);
        obj.vel_old = obj.vel; obj.vel = [vel_1_, vel_2_];
    end

    function rotational_motion(obj, dt)
        hx = obj.Th.hx; hy = obj.Th.hy;
        r1 = obj.Th.Xc - obj.center_of_mass(1);
        r2 = obj.Th.Yc - obj.center_of_mass(2);
        torque = getTorque(avg(obj.forceX), avg(obj.forceY')', r1, r2);
        torque_old = getTorque(avg(obj.forceX_old), avg(obj.forceY_old')', r1, r2);
        HT = H(1, obj.eta0.*torque, hx, hy);
        HT_old = H(1, obj.eta0.*torque_old, hx, hy);
```

```
        If = sqrt(3)*obj.edge_length/48;
        Is = obj.density*If;
        omega_ = obj.omega + dt*(-1.5*HT + 0.5*HT_old)/Is ...
               + (If/Is)*(obj.omega - obj.omega_old);
        obj.omega_old = obj.omega; obj.omega = omega_;
    end

    function update_position(obj)
        aoa = obj.angle_of_attack;
        vx0 = [-obj.r*cos(pi/6), obj.r*cos(pi/6), 0];
        vy0 = [-obj.r*sin(pi/6), -obj.r*sin(pi/6), obj.r];
        obj.vx = obj.center_of_mass(1) ...
               + (vx0.*cos(aoa+obj.theta) - vy0.*sin(aoa+obj.theta));
        obj.vy = obj.center_of_mass(2) ...
               + (vx0.*sin(aoa+obj.theta) + vy0.*cos(aoa+obj.theta));
    end

    function updateEta(obj)
        obj.eta1 = inpolygon(obj.Th.Xu, obj.Th.Yu, obj.vx, obj.vy);
        obj.eta2 = inpolygon(obj.Th.Xv, obj.Th.Yv, obj.vx, obj.vy);
        obj.eta0 = inpolygon(obj.Th.Xc, obj.Th.Yc, obj.vx, obj.vy);
    end

end
end
% help funtion in this class
function output = getTorque(Fx, Fy, rx, ry)
output = rx.*Fy - ry.*Fx;
end

function output = H(rho, F, hx, hy)
output = sum(sum(rho.*F))*hx*hy;
end
```

References

1. Belmonte, A.; Eisenberg, H.; Moses, E. From flutter to tumble: inertial drag and froude similarity in falling paper. *Phys. Rev. Lett.* **1998**, *81*, 345–348. [CrossRef]
2. Mahadevan, L.; Ryu, W.S.; Samuel, A.D. Tumbling cards. *Phys. Fluids* **1999**, *11*, 1–3. [CrossRef]
3. Pesavento, U.; Wang, Z.J. Falling paper: Navier-Stokes solutions, model of fluid forces, and center of mass elevation. *Phys. Rev. Lett.* **2004**, *93*, 1–4. [CrossRef] [PubMed]
4. Andersen, A.; Pesavento, U.; Jane Wang, Z. Unsteady aerodynamics of fluttering and tumbling plates. *J. Fluid Mech.* **2005**, *541*, 65–90. [CrossRef]
5. Jin, C.; Xu, K. Numerical study of the unsteady aerodynamics of freely falling plates. *Commun. Comput. Phys.* **2008**, *3*, 834–851.
6. Chern, M.-J.; Noor, D.Z.; Liao, C.-B.; Horng, T.-L. Direct-forcing immersed boundary method for mixed heat transfer. *Commun. Comput. Phys.* **2015**, *18*, 1072–1094. [CrossRef]
7. Horng, T.-L.; Hsieh, P.-W.; Yang, S.-Y.; You, C.-S. A simple direct-forcing immersed boundary projection method with prediction-correction for fluid-solid interaction problems. *Comput. Fluids* **2018**, *176*, 135–152. [CrossRef]

8. Noor, D.Z.; Chern, M.J.; Horng, T.L. An immersed boundary method to solve fluid-solid interaction problems. *Comput. Mech.* **2009**, *44*, 447–453. [CrossRef]
9. Fernandes, A.C.; Gomes, H.C.; Campello, E.M.; Müller, A.S.; Pimenta, P.M. A coupled FEM–DEM method for the modeling of fluids laden with particles. *Comput. Part. Mech.* **2020**. [CrossRef]
10. Hamid, A.; Arshad, A.B.; Mehdi, S.; Qasim, M.D.; Ullah, A.; Molina, J.J.; Yamamoto, R. A numerical study of sedimentation of rod like particles using smooth profile method. *Int. J. Multiph. Flow* **2020**, *127*, 103263. [CrossRef]
11. Kwon, J.; Monaghan, J.J. Sedimentation in homogeneous and inhomogeneous fluids using SPH. *Int. J. Multiph. Flow* **2015**, *72*, 155–164. [CrossRef]
12. Riazi, A.; Türker, U. The drag coefficient and settling velocity of natural sediment particles. *Comput. Part. Mech.* **2019**, *6*, 427–437. [CrossRef]
13. Walayat, K.; Talat, N.; Jabeen, S.; Usman, K.; Liu, M. Sedimentation of general shaped particles using a multigrid fictitious boundary method. *Phys. Fluids* **2020**, *32*, 063301. [CrossRef]
14. Walayat, K.; Zhang, Z.; Usman, K.; Chang, J.; Liu, M. Dynamics of elliptic particle sedimentation with thermal convection. *Phys. Fluids* **2018**, *30*, 103301. [CrossRef]
15. Glowinski, R.; Pan, T.W.; Hesla, T.I.; Joseph, D.D.; Périaux, J. A fictitious domain approach to the direct numerical simulation of incompressible viscous flow past moving rigid bodies: Application to particulate flow. *J. Comput. Phys.* **2001**, *169*, 363–426. [CrossRef]
16. Cate, A.T.; Nieuwstad, C.H.; Derksen, J.J.; Van den Akker, H.E.A. Particle imaging velocimetry experiments and lattice-Boltzmann simulations on a single sphere settling under gravity. *Phys. Fluids* **2002**, *14*, 4012–4025. [CrossRef]
17. Aulisa, E.; Bnà, S.; Bornia, G. A monolithic ALE Newton–Krylov solver with Multigrid-Richardson–Schwarz preconditioning for incompressible fluid-structure interaction. *Comput. Fluids* **2018**, *174*, 213–228. [CrossRef]
18. Saksono, P.H.; Dettmer, W.G.; Perić, D. An adaptive remeshing strategy for flows with moving boundaries and fluid–structure interaction. *Int. J. Numer. Methods Eng.* **2007**, *71*, 1009–1050. [CrossRef]
19. Sahin, M.; Mohseni, K. An arbitrary Lagrangian-Eulerian formulation for the numerical simulation of flow patterns generated by the hydromedusa Aequorea victoria. *J. Comput. Phys.* **2009**, *228*, 4588–4605. [CrossRef]
20. Blasco, J.; Calzada, M.C.; Marín, M. A Fictitious Domain, parallel numerical method for rigid particulate flows. *J. Comput. Phys.* **2009**, *228*, 7569–7613. [CrossRef]
21. Wan, D.; Turek, S. Direct numerical simulation of particulate flow via multigrid FEM techniques and the fictitious boundary method. *Int. J. Numer. Methods Fluids* **2006**, *51*, 531–566. [CrossRef]
22. Fortes, A.F.; Joseph, D.D.; Lundgren, T.S. Nonlinear mechanics of fluidization of beds of spherical particles. *J. Fluid Mech.* **1987**, *177*, 467–483. [CrossRef]
23. Hu, H.H.; Joseph, D.D.; Crochet, M.J. Direct simulation of fluid particle motions. *Theor. Comput. Fluid Dyn.* **1992**, *3*, 285–306. [CrossRef]
24. Choi, H.G. Splitting method for the combined formulation of the fluid-particle problem. *Comput. Methods Appl. Mech. Eng.* **2000**, *190*, 1367–1378. [CrossRef]
25. Gazzola, M.; Chatelain, P.; van Rees, W.M.; Koumoutsakos, P. Simulations of single and multiple swimmers with non-divergence free deforming geometries. *J. Comput. Phys.* **2011**, *230*, 7093–7114. [CrossRef]
26. Kolomenskiy, D.; Schneider, K. A Fourier spectral method for the Navier Stokes equations with volume penalization for moving solid obstacles. *J. Comput. Phys.* **2009**, *228*, 5687–5709. [CrossRef]