Available online at www.sciencedirect.com

ScienceDirect

Journal of Computational Physics 227 (2008) 6895–6921

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

An object-oriented and quadrilateral-mesh based solution adaptive algorithm for compressible multi-fluid flows

H.W. Zheng, C. Shu *, Y.T. Chew

Department of Mechanical Engineering, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260, Singapore

Received 26 September 2007; received in revised form 2 March 2008; accepted 31 March 2008
Available online 10 April 2008

Abstract

In this paper, an object-oriented and quadrilateral-mesh based solution adaptive algorithm for the simulation of compressible multi-fluid flows is presented. The HLLC scheme (Harten, Lax and van Leer approximate Riemann solver with the Contact wave restored) is extended to adaptively solve the compressible multi-fluid flows under complex geometry on unstructured mesh. It is also extended to the second-order of accuracy by using MUSCL extrapolation. The node, edge and cell are arranged in such an object-oriented manner that each of them inherits from a basic object. A home-made double link list is designed to manage these objects so that the inserting of new objects and removing of the existing objects (nodes, edges and cells) are independent of the number of objects and only of the complexity of $\boldsymbol{O(1)}$. In addition, the cells with different levels are further stored in different lists. This avoids the recursive calculation of solution of mother (non-leaf) cells. Thus, high efficiency is obtained due to these features. Besides, as compared to other cell-edge adaptive methods, the separation of nodes would reduce the memory requirement of redundant nodes, especially in the cases where the level number is large or the space dimension is three. Five two-dimensional examples are used to examine its performance. These examples include vortex evolution problem, interface only problem under structured mesh and unstructured mesh, bubble explosion under the water, bubble-shock interaction, and shock-interface interaction inside the cylindrical vessel. Numerical results indicate that there is no oscillation of pressure and velocity across the interface and it is feasible to apply it to solve compressible multi-fluid flows with large density ratio (1000) and strong shock wave (the pressure ratio is 10,000) interaction with the interface.

Keywords: Adaptive mesh refinement; Compressible multi-fluid; HLLC; Finite volume; Quadrilateral

1. Introduction

The dynamics of compressible multi-fluid flows have many practical applications in engineering, such as shock-bubble interactions [1,2] and underwater explosions [3]. Due to complexity of these problems such as steep material interface, high density and pressure gradients that occur in such flows, accurate predictions

* Corresponding author. Tel.: +65 6874 6476; fax: +65 6779 1459.
E-mail address: mpeshuc@nus.edu.sg (C. Shu).

0021-9991/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2008.03.037

of flow structure of compressible multi-fluid flows pose a major challenge. Thus, it is necessary to apply the adaptive mesh technique to improve the efficiency and accuracy.

In the past decades, intensive research and efforts have been devoted to the development of adaptive mesh refinement technique. As a result, a large number of adaptive algorithms have been proposed and can be categorized as three categories, h-refinement, p-refinement and r-refinement. The basic idea behind adaptive mesh refinement is to increase the density of cells by adding cells in regions where accurate solution is required. h-Refinement increases the mesh resolution by adding elements or vertices to the mesh. For the method of p-refinement, it improves solution by increasing the order of accuracy of the polynomial in each element (or cell). As compared to the previous two methods, r-refinement [4,5] modifies the mesh density through the use of node movement and keeps the number of vertices and elements unchanged. Among these three methods, h-refinement has been widely applied in many areas [6–8].

The popular h-refinement method is the Cartesian grid solver which may be implemented by the nested hierarchical data structure such as quad-tree in 2D [8,9] and octree in 3D [10]. A more general version is to use block adaptive Cartesian meshes [6,11,12] organized in a series of rectangular grid patches. The main drawback of these adaptive Cartesian meshes is their difficulty to be applied to complex geometry. Besides, all these tree-based Cartesian adaptive approaches encounter the difficulties of vectorization (on vector architectures) and tree traversal overhead due to the searching of neighboring cells. Another type of h-refinement method does not use the tree but an unstructured data structure where the connectivity is explicitly stored with the mesh [13–15]. This unstructured feature makes it be easily extended to complex geometry. Besides, as the neighboring references are explicitly stored, the calculation time required for computing quantities involving the neighbors will be lesser than that for the tree-based approach where the recursive tree traverses are required to search the neighboring connectivity.

Although there are many adaptive mesh methods, their extension and applications to compressible multi-fluid flows are still rarely conducted, especially for cases with a large density difference and strong shock wave interaction at the material interface. Recently, Nourgaliev et al. [3] presented the work by combining the ghost fluid method and the block structured adaptive Cartesian meshes. However, it is not easy to be applied to problems with complex geometry due to the use of structured meshes. Besides, as reported by the authors, the coarse-to-fine and fine-to-coarse inter-level transfer operators are very complicated and may violate the stability of the code. Moreover, the ghost fluid method employs the level set method to track the interface. The mass or momentum may not be conserved.

Hence, in order to deal with problems with complex geometry, we adopt the unstructured adaptive technique and the diffuse interface method [2,16] instead of the Cartesian structured mesh and the ghost fluid method. The well-designed data structure for the objects (node, edge and cell) and memory arrangement of the object lists contribute to the fact that the adding, deleting of a certain object is only of the order of $\boldsymbol{O}(\boldsymbol{1})$ as compared to $\boldsymbol{O}(\boldsymbol{n})$ of other cell-edge based methods [15,17]. Besides, due to the hierarchy storage of cell and the separated storage of leaf edges and non-leaf edges in different lists, the edge-based finite volume solver can be easily applied to the current adaptive mesh solver as compared to the cell-based finite volume solver [15,17]. Moreover, the separation of node will reduce the memory requirement of redundant nodes, especially in the cases where the level number is large or the space dimension is three. It will also make the conversion of conservative variables from cell-centroid to the cell-vertex and the extension of adaptive mesh method from 2D to 3D easier and more effective. Hence, from the viewpoint of implementation, our adaptive algorithm is quite simple and efficient.

For the simulation of compressible multi-fluid flows, a well-known difficulty [2,16,18] associated with Euler solvers in the conservative form is the possible appearance of spurious pressure oscillations at material interfaces. Thus, the quasi-conservative form is applied to conquer this problem. Besides, a suitable Godunov-type scheme for multi-fluid flows should resolve the contact wave properly in order to capture the interfaces. In this work, Harten, Lax and van Leer approximate Riemann solver with the contact wave restored, called the HLLC scheme [19–21] is adopted. This method is further extended to the second-order of accuracy by monotonic upwind schemes for conservation laws (MUSCL).

The paper is organized as follows. The finite volume algorithm for compressible multi-fluid flows under unstructured adaptive mesh is given in Section 2. The data structure and the implementation of the adaptive method are proposed in Section 2. The model is extensively validated for single fluid vortex evolution problem, and bubble explosion case. It is then applied to a bubble shock interaction problem under strong shock waves (the pressure ratio is 10,000). Accurate results are obtained at high resolution due to the solution adaptive technique.

2. Methodology

In this section, the governing equations for the compressible multi-fluid flows are described firstly. The HLLC (Harten, Lax and van Leer approximate Riemann solver with the contact wave restored) scheme is extended to solve these equations on the unstructured mesh. The scheme is further extended to the second-order accuracy by MUSCL gradient reconstruction technique and the second-order Runge–Kutta method.

2.1. Governing equations

For each fluid component of compressible multi-fluid flows, the governing equation is the Euler equation, which is written as

$$\partial_t \int_{\Omega} U \, \mathrm{d}V + \int_S \Phi \cdot \vec{n} \, \mathrm{d}S = 0, \tag{1}$$

where U is the state vector of conservative variables, and Φ is flux vector,

$$U = \begin{pmatrix} \rho \\ \rho\vec{u} \\ E \end{pmatrix}, \quad \Phi = \begin{pmatrix} \rho\vec{u} \\ \rho\vec{u} \otimes \vec{u} + p[I] \\ (E+p)\vec{u} \end{pmatrix}. \tag{2}$$

Here, ρ is the density, $\vec{u}$ is the velocity, E is the total energy, p is the pressure, and $[I]$ is the identity tensor.

Neglecting viscous, heat transfer and surface tension effects, the velocity and pressure should stay continuous across interfaces. That is,

$$\Delta\vec{u} = [\vec{u}]_-^+ = 0, \Delta p = [p]_-^+ = 0. \tag{3}$$

By using Eq. (3) and the analysis which is similar to the one-dimensional analysis by Abgrall et al. [16], we have the upwind discretization of Eq. (1) across interfaces in multi-dimensional space,

$$\begin{pmatrix} \delta\rho \\ \delta(\rho\vec{u}) \\ \delta(\rho e) \end{pmatrix} = \begin{pmatrix} -\upsilon u_n \Delta(\rho) \\ \vec{u}\delta(\rho) \\ -\upsilon u_n \Delta(\rho e) \end{pmatrix}, \tag{4}$$

where $\delta()$ denotes the time changes $()^{n+1} - ()^n$, $u_n = \vec{u} \cdot \vec{n}$ is the normal velocity at the material interface with the normal direction $\vec{n}$, and $\Delta()$ denotes the spatial changes. The first two equations are trivial. Thus, the third equation which is the discretization of the internal energy is the key to keep a non-oscillating pressure and velocity near interfaces.

If we further suppose that the equation of state for each fluid component is the Stiffened gas equation,

$$p = (\gamma(x,t) - 1)\rho e - \gamma(x,t)\pi(x,t), \tag{5}$$

we can easily prove that the equations below [2] satisfy Eq. (4),

$$\frac{\partial\Theta}{\partial t} + \vec{u} \cdot \vec{\nabla}\Theta = 0 \tag{6}$$

with the definition

$$\Theta = \begin{pmatrix} \beta \\ \theta \end{pmatrix}, \quad \beta = \frac{1}{\gamma - 1}, \quad \theta = \frac{\gamma\pi}{\gamma - 1}. \tag{7}$$

Here, γ and π are the property of the material, β is also used to indicate the interface because each fluid has a constant β. In conclusion, the governing equations for compressible multi-fluid flows are Eqs. (1) and (6).

2.2. Numerical methods

In this section, we will show numerical discretization of Eqs. (1) and (6) on the unstructured mesh. Eq. (1) can be easily discretized at each cell c by the finite volume method as

$$U_{\mathrm{c}}^{n+1} = U_{\mathrm{c}}^{n} - \frac{\Delta t}{A_{\mathrm{c}}}\mathrm{Res}_{\mathrm{c}} \tag{8}$$

with the residue defined as

$$\mathrm{Res}_{\mathrm{c}} = \sum_{\mathrm{f}} \Phi_{\mathrm{f}}(U^{\mathrm{L}}, U^{\mathrm{R}}, \vec{n}) \cdot \Delta l_{\mathrm{f}}. \tag{9}$$

Here, A_{c} is the area of the cell c, Φ_{f} is the numerical flux at the edge f of the cell c; the variables $U^{\mathrm{L}}, U^{\mathrm{R}}, \vec{n}$, and Δl_{f} denote the left state, right state, normal direction and the length of the edge f.

From Eq. (3), it is seen that a suitable numerical scheme should be the one that could well resolve the contact wave. Hence, the Harten, Lax and van Leer approximate Riemann solver with the contact wave restored (HLLC) scheme [19,20] is adopted in this paper. It is known that this scheme has a good resolution of shocks, contact waves and the feature of preservation of the positivity of density and internal energy. The good resolution of contact waves makes it very suitable for multi-fluid problems. Similar to one-dimensional HLLC descriptions [21], the numerical flux on multi-dimensional unstructured mesh can be expressed as

$$\Phi_{\mathrm{f}} = \Phi^{\mathrm{HLLC}}(\vec{U}^{\mathrm{L}}, \vec{U}^{\mathrm{R}}, \vec{n}) = \begin{cases} \Phi^{*\mathrm{L}} & 0 \leqslant s_m \\ \Phi^{*\mathrm{R}} & s_m \leqslant 0 \end{cases} \tag{10}$$

with

$$\Phi^{*\mathrm{L}} = \Phi(U^{\mathrm{L}}, \vec{n}) + s_{\mathrm{L}}(U^{*\mathrm{L}} - U^{\mathrm{L}}), \tag{11}$$

$$\Phi^{*\mathrm{R}} = \Phi(U^{\mathrm{R}}, \vec{n}) + s_{\mathrm{R}}(U^{*\mathrm{R}} - U^{\mathrm{R}}), \tag{12}$$

where s_{L} and s_{R} are two intermediate signal speeds of HLLC scheme,

$$s_{\mathrm{L}} = \min(u_n^{\mathrm{L}} - a^{\mathrm{L}}, \tilde{q} - \tilde{a}), \tag{13}$$

$$s_{\mathrm{R}} = \max(u_n^{\mathrm{R}} + a^{\mathrm{R}}, \tilde{q} + \tilde{a}). \tag{14}$$

Here, a^{L}, a^{R}, and $\tilde{a}$ are the left sound speed, right sound speed and the average sound speed and $\tilde{q}$ is the average velocity. The two intermediate states $U^{*\mathrm{L}}$ and $U^{*\mathrm{R}}$ in Eqs. (11), (12) can be written as this form in 2D,

$$U^{*k} = \frac{(s_k - u_n^k)}{s_k - s_m} \begin{pmatrix} \rho^k \\ \rho^k[u_1^k - (u_n^k - s_m)n_x] \\ \rho^k[u_2^k - (u_n^k - s_m)n_y] \\ E^k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \frac{p^{*k}s_m - p^k u_n^k}{(s_k - u_n^k)} \end{pmatrix}, \quad k = L, R \tag{15}$$

with the signal speed of the contact wave,

$$s_m = \frac{\rho^{\mathrm{R}} u_n^{\mathrm{R}}(s_{\mathrm{R}} - u_n^{\mathrm{R}}) - \rho^{\mathrm{L}} u_n^{\mathrm{L}}(s_{\mathrm{L}} - u_n^{\mathrm{L}}) + p^{\mathrm{L}} - p^{\mathrm{R}}}{\rho^{\mathrm{R}}(s_{\mathrm{R}} - u_n^{\mathrm{R}}) - \rho^{\mathrm{L}}(s_{\mathrm{L}} - u_n^{\mathrm{L}})}. \tag{16}$$

Most of schemes [2,16] used for numerical discretization of Eq. (6) are only suitable for structured mesh. For the case of unstructured mesh, an effective way for numerical discretization of Eq. (6) is to rewrite it to the following form [21,22]:

$$\frac{\partial \Theta}{\partial t} + \vec{\nabla} \cdot \Lambda - \Theta \vec{\nabla} \cdot \vec{u} = 0, \tag{17}$$

where the numerical flux is,

$$\Lambda = \Theta \vec{u}. \tag{18}$$

Eq. (17) can be discretized as

$$\Theta_c^{n+1} = \Theta_c^n - \frac{\Delta t}{A_c}\left(\sum_f \Lambda_f \cdot \Delta l_f - \Theta_c^n \sum_f q_f \cdot \Delta l_f\right), \tag{19}$$

where q_f is the velocity at the edge f, and the conservative part of flux Λ_f is evaluated in a similar way by,

$$\Lambda_f = \Lambda_f^{\mathrm{HLLC}}(\Theta^L, \Theta^R, \vec{n}) = \begin{cases} \Lambda_f^{*L} & 0 \leqslant s_m \\ \Lambda_f^{*R} & s_m \leqslant 0 \end{cases} \tag{20}$$

with

$$\Lambda_f^{*L} = \Lambda(\Theta^L, \vec{n}) + s_L(\Theta^{*L} - \Theta^L), \tag{21}$$

$$\Lambda_f^{*R} = \Lambda(\Theta^R, \vec{n}) + s_R(\Theta^{*R} - \Theta^R), \tag{22}$$

$$\Theta^{*L} = (s_L - u_n^L)/(s_L - s_m)\Theta^R, \tag{23}$$

$$\Theta^{*R} = (s_R - u_n^R)/(s_R - s_m)\Theta^R. \tag{24}$$

The scalar normal velocity at the edge f can be computed in a consistent HLLC way by,

$$q_f = \begin{cases} u_n^L + s_L[(s_L - u_n^L)/(s_L - s_m) - 1], & 0 \leqslant s_m \\ u_n^R + s_R[(s_R - u_n^R)/(s_R - s_m) - 1], & 0 > s_m \end{cases}. \tag{25}$$

It should be indicated that the above equations have no requirement of mesh structure. Thus, they can be well applied on the unstructured mesh.

2.3. Second-order approximation in space and time

The scheme described in the above section is only the first-order if the left and right state of an edge is simply set as the state at the left cell's center and the state at the right cell's center respectively. As we know, the first-order numerical schemes are highly dissipative and less accurate. Besides, higher order discretization in time may broaden the stable CFL range [15]. So, in this section, we will discuss the second-order discretization in both space and time.

To increase the solution accuracy in space, the MUSCL-type (Monotonised Upstream centered scheme for conservation laws) gradient reconstruction scheme [23] is employed. The left and right states are constructed from extrapolated values from cell centers to cell interfaces and then used to construct fluxes. However, gradients calculated may be inappropriate if they are based on the information across flow discontinuities, such as shock and contact waves. This may lead to catastrophic destabilization of the solution. To make the solution be monotonic, the slope limiters are enforced in the extrapolation,

$$W_{\partial\Omega}^L = W_L + \frac{1}{2}\psi(2\vec{\nabla}W_L \cdot d\vec{r}_L - \Delta W, \Delta W), \tag{26}$$

$$W_{\partial\Omega}^R = W_R - \frac{1}{2}\psi(\Delta W, 2\vec{\nabla}W_R \cdot d\vec{r}_R - \Delta W), \tag{27}$$

$$\Delta W = W_R - W_L, \tag{28}$$

where W is the vector of the primitive variables, $d\vec{r}$ is the distance vector between the cell center and the edge center, and $\psi(a, b)$ is a slope limiter. There are several choices for the slope limiter. In the present work, a minmod limiter is used since numerical experiments showed that the use of this limiter can produce more stable results. The minmod limiter is defined as

$$\psi(a,b) = \frac{\mathrm{sign}(a) + \mathrm{sign}(b)}{2}\min(|a|, |b|). \tag{29}$$

The minmod limiter captures non-oscillatory gradient information by taking the minimum modulus of the face-centered gradients and applying it to the cell centre. In other words, if any two gradients are of different signs, the cell-centered gradient is set to zero. Otherwise, the gradient with the minimum absolute value is used.

The gradient of a primitive variable η of cell c in Eq. (27) is calculated by the weighted least square method. That is,

$$\vec{\nabla}\eta = \begin{bmatrix} \sum_i \alpha_i^2 \Delta x_i \Delta x_i & \sum_i \alpha_i^2 \Delta x_i \Delta y_i \\ \sum_i \alpha_i^2 \Delta x_i \Delta y_i & \sum_i \alpha_i^2 \Delta y_i \Delta y_i \end{bmatrix}^{-1} \begin{pmatrix} \sum_i \alpha_i^2 \Delta x_i \Delta \eta_i \\ \sum_i \alpha_i^2 \Delta y_i \Delta \eta_i \end{pmatrix}, \tag{30}$$

where $\Delta\eta_i$ is the difference between the primitive variable w of the current cell c and that of the neighboring cell i, Δx_i, Δy_i are the differences of coordinates in x and y direction, andα_i is the correspondent weight coefficient,

$$\Delta x_i = x_i - x_c, \tag{31}$$

$$\Delta y_i = y_i - x_c, \tag{32}$$

$$\Delta \eta_i = w_i - w_c, \tag{33}$$

$$\alpha_i = \frac{1}{\text{sqrt}[(x_i - x_c)^2 + (y_i - y_c)^2]}. \tag{34}$$

To increase the accuracy in time, Eq. (8) can be replaced by using the second-order TVD Runge–Kutta scheme,

$$U_c^{(*)} = U_c^n - \frac{\Delta t}{A_c}\text{Res}(U_c^n), \tag{35}$$

$$U_c^{n+1} = \frac{1}{2}U_c^n + \frac{1}{2}U_c^{(*)} - \frac{\Delta t}{2A_c}\text{Res}(U_c^{(*)}). \tag{36}$$

3. Object-oriented adaptive method on unstructured mesh

The adaptive grid generator developed in this work belongs to the unstructured adaptive grid family which does not employ tree-like data structure. Instead, all of the connectivity information that is needed by each quadrilateral cell is explicitly stored. In the present work, the mesh refinement process is made by the object-oriented programming written in C++. As compared to procedural programming techniques such as FORTRAN, the object-oriented programming is a type of programming that encapsulates not only the data type of a data structure, but also the types of operations (functions). Besides, the data can be inherited and the operations can be redefined by the derived classes. These features help to design a good data structure which helps to reduce the memory and improve the efficiency. In the following, the memory arrangement, object-oriented cell-edge-node data structure, local refinement, local coarsening and the adaptive edge-based finite volume procedure are described.

3.1. Memory arrangement

As compared to the structured mesh generation, the adaptive mesh generation has no structured indices. That is, the indices of an object are usually stored in a one-dimensional container such as one-dimensional array. Thus, the memory arrangement is very important and critical for most of the adaptive techniques. One may choose the array with fixed size of memory by using FORTRAN language. For example, in the paper of Sun and Takayama [14], one part of the array with constant size is used to store leaf cells (which have no sub-cells) and the other part is used to store the father (non-leaf) cells (which have sub-cells). As a result, it limits the maximum number of cells, edges and the level of the refined mesh. What is more, the adding or removing of an object, and the allocating of the new object index needs to search from the beginning to the end of the array. This will reduce the efficiency of the calculation. In order to obtain a flexible memory arrangement, one may choose the ***vector*** or ***List*** containers of C++ language. However, these containers included in the standard template library (STL) of C++ language have a memory overhead that is associated with the excessive pointers employed by the containers themselves. Besides, the ***vector*** container can only add or delete an object from the rear or the head. In the standard ***List***, the new object is added to the list as the data part of the list item which also consists of two pointers that point to previous and next objects. Thus, the standard ***List*** container has a poor performance in deleting an object. For example, to delete an object, it needs to search from the first one to the last one in the list until the pointer to the object is found. That is, the complexity of deleting an object in ***List*** container is $\boldsymbol{O(n)}$ where n is the number of the items in the list.

Thanks to ***object-oriented*** feature of the C++ language, we designed a double linked list to improve the efficiency of the deleting procedure of ***List***. In this new list which is called ***TList*** herein, we replace the list item by a basic object data structure ***TObject***. That is, the new list does not store the object but a manger to set the previous and next pointer of an object. Thus, the basic member of the data structure ***TObject*** consists of the previous and next pointer to the other objects (list items). As a result, the adding or removing an object in the list is equivalent to set the previous pointer (or index) and the next pointer (or index) to be the last object of the list and the next object of the current object. The searching of the object is not needed. Hence, the complexity of the removing operator is independent of the number of items in the list and only the order of $\boldsymbol{O(1)}$ as compared to $\boldsymbol{O(n)}$ of the standard list. Besides, the list stores the pointer (indices) of the objects directly. Thus, any object can be directly accessed by the pointer.

3.2. Cell-edge-node object-oriented data structure

The cell-edge data structure is employed in most of the unstructured adaptive technologies [13–15,24]. That is, there is no independent data-structure of the node information. Thus, redundant node information [13,24] may be stored. For example, information of sixteen new nodes is stored if you refine a cell to four sub-cells in two-dimensional cases. In fact, the maximum numbers of nodes which are required to be added are five and may be only one in some cases. This is a serious issue when the level is very high, and is even worse in three-dimensional cases where 64 (8×8) new nodes are added for each refinement of a cell. Besides, the overlapping and redundant information of the node in cell-edge structure results in that the transfer of the values from cell-center to cell-node is difficult. Thus, it is necessary to use the independent variables to store the geometric parameters of the node which is denoted as ***TNode*** (as shown in Fig. 1) in our solver.

For the edge, one may decompose it into two half edges with opposite orientations as shown in computational geometry algorithms library (CGAL) [25]. This way would need a large amount of memory. In order to reduce the memory, we can only store one directional edge [17] and the two opposite edges of the cell have the same direction. Hence, the two neighboring cells can share the same directional edge. Like other unstructured method, the edge also stores the pointers to the two neighboring cells. Since we can easily obtain the neighboring cells by visiting the four edges, we can access the node easily. Thus, the current edge structure (***TEdge***) does not store the pointer (or indices) of nodes as done by Sun [17]. This will reduce the unnecessary storage of the pointer (or indices). Besides, it is clear that this structure of ***TEdge*** is suitable for the edge-based finite

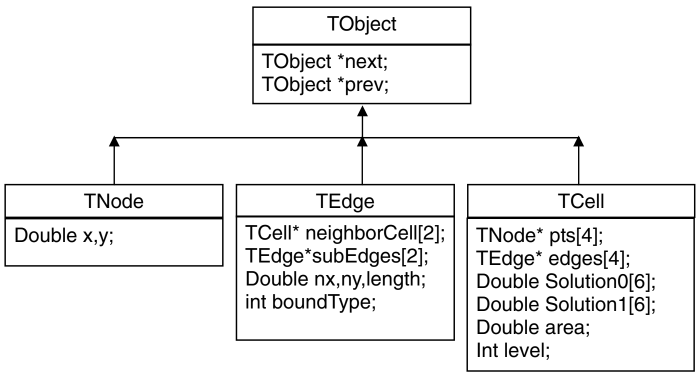


Fig. 1. Data structure of node, edge and cell.

volume solver. The pointer (or indices) to the two sub-edges is also included in edge data structure ***TEdge***. Thus, the adding or deleting node does not need to access the neighboring cells. In order to identify the boundary type, the information of boundary type is also stored.

Since the cell-centered finite volume method is used, all variables at the cell centroid are stored in the data structure ***TCell***. Like the method of Sun [14,17], the pointers (or indices) to the four edges and nodes are also stored by a fixed order. Thus, there is no need to store the pointers (or indices) of the four child cells as they can be accessed by using the sub-edges of the four edges. To extend the solver to the second-order of accuracy, the gradients of primitive variables are also calculated and stored. Besides, in order to implement the TVD Runge–Kutta time integration, the state variables of the previous time step are also stored. In summary, ***TCell*** consists of the state variables, previous state variables and the gradient of the primitive variables as shown in Fig. 2.

Finally, a data structure which is called ***TMesh*** is used to store all the objects of node, edge, and cell. Besides, the ***TList*** is used as a memory arrangement for all these objects. To improve the efficiency, the edges are further organized into two separated lists which are called *LeafEdgeList, and MotherEdgeList* respectively. For the methods such as quad-tree or octree based adaptive methods [9], and Cartesian adaptive methods [15,24], cells of the different levels are not stored in different lists. Thus, in order to satisfy the conservation law, the calculation of the solution of the mother cell must be recursively calculated by the summated results of the solution of each child cells. However, the recursive procedure cannot be vectorlizated. So, in order to avoid the recursive calculation, the cells in our methods are separately stored in different lists according to their levels. Thus, the solution can be obtained in a level by level manner so that the cells of high level are calculated after the cells of low level. This makes our algorithm very efficient and suitable for vectorization. Moreover, for those methods [17,15] where the information of leaf cells and non-leaf cells are stored in two different lists, the cells need to frequently move from one list to the other list during the process of refinement and coarsening. In contrast, the current method does not need to do so. This shows that our method also improves the efficiency especially when the finest resolution level is large.

In order to develop an edge-based solution adaptive solver, the edges are arranged in a different way from that of cells. That is, the edges are arranged in two different separated lists: a list which is used to store the leaf edges and a list which is used to store mother edges. Therefore, the edge-based solver can be applied only to the leaf edges. The numerical fluxes of the mother edges are not calculated. Besides, these numerical fluxes of the leaf edges are not stored and only used to update the residues of the left and right cells. As compared to the cell-based methods [14,17,15] which store the fluxes of the edges, the present method that store the residues at the cells will reduce the memory requirement since the number of edges is approximately two times of that of cells [14].

3.3. Local refinement

In order to determine which cell is to be refined, the refinement criterion is required. In this paper, it is chosen the same as that of Sun and Takayama [14]. That is, the error indicator is the maximum of the second-order Taylor truncation error over the first-order Taylor truncation error of density of each edge,

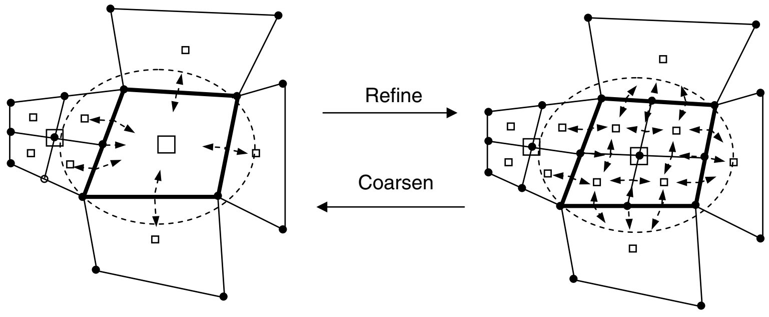


Fig. 2. Example of refinement and coarsening procedure.

$$\varepsilon_T = \max_f\left(\frac{|(\nabla_{l_n}\rho)_C - (\nabla_{l_n}\rho)_L|}{\alpha_f\rho_C/dl + |(\nabla_{l_n}\rho)_L|}, \frac{|(\nabla_{l_n}\rho)_C - (\nabla_{l_n}\rho)_R|}{\alpha_f\rho_C/dl + |(\nabla_{l_n}\rho)_R|}\right)_f, \tag{37}$$

where *dl* is the length of the edge and the gradients are

$$(\nabla_{l_n}\rho)_C = \frac{\rho_R - \rho_L}{|\vec{r}_R - \vec{r}_L|}, (\nabla_{l_n}\rho)_{L,R} = (\nabla\rho)_{L,R}\cdot\frac{\vec{r}_R - \vec{r}_L}{|\vec{r}_R - \vec{r}_L|}. \tag{38}$$

If the value of error detector in a certain cell is greater than the pre-defined refining threshold ε_R, then the cell is marked for refinement; otherwise, it is marked for coarsening. The default value of sensitivity α_f is 0.03. It has been shown that this error indicator can detect most of discontinuity such as shock wave and contact discontinuity [14].

In the refinement, each cell is divided into four quadrilateral sub-cells. It is clear that only four new cell objects are created. However, the adding of edges and nodes need to be carefully checked to make sure that only the necessary edges and nodes are created. To simplify the adaptation procedure, one may check whether the cell can be refined before the process of refinement. Like other adaptive Cartesian methods, there are only two constraints for this process. One is that the cell has no children (sub-cells). The other one is that the maximum difference of the levels between the current cell and all the neighboring cells cannot be greater than one. The procedure of refinement is listed as follows:

(1) The first step is to create the new nodes appropriately (Fig. 2). A new node is crated at the centroid of the cell firstly. If one edge has no sub-edges, a node is created. Otherwise, we just use the existing center node of edge. All the newly-generated nodes are added to the node list.
(2) The second step is to create the edges appropriately (Fig. 2). Similar to the creation of nodes, the two sub-edges of each edge are created if the edge has no sub-edges. The new sub-edges are then added to the leaf edge list. The original edges which have sub-edges must be re-assigned to the non-leaf edge list.
(3) Create four cells and add them to cell list with the corresponding level.
(4) Assign the neighboring cells of each edge.
(5) Assign the neighboring cells of each sub-edge if the edges have sub-edges.
(6) Set the boundary type of each new edge according to their mother edge.
(7) Initialize the solutions and the levels of all new cells.

3.4. Local coarsening

As a reverse process of the refinement, the coarsening process removes the unnecessary cells from the mesh. This helps to reduce the memory requirement and achieve maximum computational efficiency. Since the classes of TNode, TEdge and TCell are inherited from the ***TObject***, we can just call the removing function of ***TList*** to delete the object from the memory. Similar to the refinement procedure, we use the error indicator to determine which cell is to be deleted. Before the process of coarsening, we also need to check whether the marked cell can be coarsened. There are three constraints for this. The first one is that the cell must have children (sub-cells). The second one is that the maximum difference of the levels between the current cell and all the neighboring cells cannot be greater than one. The third one is that the value of error detector is smaller than the coarsening threshold. If all these three constraints are satisfied, then the cell is removed by calling the coarsening procedure. In this coarsening process, we also need to change the neighbor cell information of the cell. This is simply implemented by reassigning the two pointers (indices) of the neighbor cells of the four edges of the cell. In conclusion, the procedure of coarsening is listed as follows:

(1) The first step is to remove the nodes appropriately (Fig. 2). The node at the centroid of the mother cell is removed firstly. The center nodes of each edge are removed from the node list if the corresponding edge has no sub-edges.
(2) The second step is to remove the sub-edges appropriately. The two sub-edges of each edge are removed from the leaf edge list if the outer neighbor cell has no child cells. The edges of the cell that has no sub-edges are re-arranged from the non-leaf edge list to the leaf edge list.

(3) Delete four cells from the cell list of the corresponding level.
(4) Re-assign the neighbor cells of each edge.

3.5. Overall procedure

As stated above, all edges are arranged in a list of leaf edges and a list of non-leaf edges. Thus, the **edge-based** adaptive simulation of compressible multi-fluid flows can be easily implemented. The calculation of the numerical flux of leaf edge only requires the information at the left state and right state of the edge. The flow variables at the two states are calculated according to Hermit interpolation (functional value and its first-order derivatives are used) from the cell centers of both sides of the edge. So, there is no need to handle the handing nodes and no special treatment at the interface between the finer cell and the coarse cell. There is no difference between the calculation of the numerical flux at the coarse-fine grid interface and that at the fine-fine or coarse-coarse grid interfaces. The whole procedure for the **edge-based** adaptive simulation of compressible multi-fluid flows is summarized below:

```
(1) Initialize the mesh and solution;
(2) Perform the refinement or coarsening for all cells appropriately:
    for (each cell in all cell lists of different levels)
    {
    if (the error indicator of this cell is larger than the critical refinement threshold)
        Perform the local refinement of this cell;
    else if (the error indicator of this cell is smaller than the critical coarsening threshold)
        Perform the local coarsening of this cell;
    }
(3) Calculate the time step;
(4) Perform edge-based flux calculation for each leaf edge (excluding non-leaf edges):
    for (each leaf edge in the leaf edge list){
    Calculate the numerical flux and update the residues of the left and right cell;
    }
(5) Update the solutions of all cells:
    for (each cell list from the list of the largest level to the list of the smallest level) {
        for (each cell in the current cell list)
            Calculate and update the solution of the current cell;
        }
(6) If the termination condition is not satisfied, then goes to (2); otherwise, goes to (7)
(7) Output the results.
```

4. Results and discussion

In order to validate the performance of the present solution adaptive technique for compressible multi-fluid flows, test cases of two-dimensional vortex evolution, material interface problem, bubble explosion under the water and shock-interface interaction inside the cylindrical vessel are considered.

4.1. Two-dimensional vortex evolution problem

To investigate the order of accuracy of our solver, we first consider the single fluid vortex evolution problem by setting both of the parameters of two fluids as $\gamma = 1.4$, $\pi = 0$. This also shows that our solver has the capability to solve the single compressible flow problems. The problem is solved in the square domain $[0,10] \times [0,10]$ with periodic boundary conditions. The total time is two seconds. To our experience and knowledge, the periodic boundary condition is the hardest to be implemented for an adaptive solver.

The vortex is defined as the isentropic perturbation to the uniform flow. That is, the initial values of the primitive variables are set as [26],

$$\rho = T^{1/(\gamma-1)}, \quad u = 1 + \delta u, \quad v = 1 + \delta v, \quad p = \rho^{\gamma}, \quad T = 1 + \delta T \tag{39}$$

with the perturbation,

$$(\delta u, \delta v) = \frac{\varepsilon}{2\pi} \mathrm{e}^{0.5(1-r^2)}(-(y-5),(x-5)), \tag{40}$$

$$\delta T = -\frac{(\gamma-1)\varepsilon^2}{8\gamma\pi^2} e^{(1-r^2)}, \quad \varepsilon = 5, \tag{41}$$

where r is the distance,

$$r^2 = (x-5)^2 + (y-5)^2. \tag{42}$$

As stated by Sun and Takayama [27], a suitable error measure is critical to the calculation of order of accuracy for the adaptive method. The traditional norm calculation cannot be applied to the adaptive method due to the lack of including the effects of error localization. For the purpose of evaluating the convergence rate of present adaptive algorithm, the norm for level k is chosen as,

$$(L_\infty)^k = \|u - u^{\mathrm{ex}}\|_\infty = \max_{i \in N_k}\{|u_i - u_i^{\mathrm{ex}}|\}, \tag{43}$$

where N_k denotes the total number of cells of level k. That is, the error is measured at the same resolution level. Thus, if the finest resolution level is 2, there will be three L_∞ with respect to the three resolution levels correspondingly.

To study the convergence rate of the adaptive scheme, three background meshes are employed, i.e. 21×21, 41×41 and 81×81. Since the solution is smooth in the whole field, the limiter is not used in this case. The exact solution is a smooth vortex movement moving along the direction of the diagonal direction of the Cartesian mesh lines. Based on this analytical solution, the corresponding numerical results in terms of L_∞ norm of error are quantitatively shown in Table 1. It can be clearly observed that the solution is converging to the exact solution with approximately the second-order of accuracy in L_∞ norms for all the levels. This indicates the second-order convergence of our adaptive method. To further investigate the efficiency of current adaptive method, four cases with adaptive mesh and two cases with fixed mesh are performed on the PC with 1 GB RAM and 2.99 GHz CPU. The computational times and the numbers of objects are presented in Table 2. From this table, it is easily observed that, to achieve the same accurate solution in terms of finest L_∞ norm of error, the numbers of nodes, edges and cells required by the adaptive method are less than those of fixed mesh method. Besides, it can be seen that the time needed by present adaptive method is less than one-third (or even one-fifth) of the time needed by the fixed mesh. This shows that the adaptive algorithm is efficient and accurate.

4.2. Interface only problem

In this section, the free of oscillation feature and preservation of the sharp material interface of our adaptive algorithm is investigated. Initially, one fluid with a circular shape surrounded by another fluid is put at the position (0.25 m, 0.25 m) of the domain $[0,1] \times [0,1]\mathrm{m}^2$. The radius of the circular interface is $r_0 = 0.16$ m. The

Table 1
Numerical results of the unsteady vortex evolution problem

Background mesh spacing	L_∞ norm of error		
	Level 0	Level 1	Level 2
0.5	0.017	0.0072	0.0051
0.25	0.0046	0.0017	0.0011
0.0125	0.0011	0.00051	0.00033
Average convergence rate	1.97	1.91	1.97

Table 2
Efficiency comparison between adaptive grid and uniform grid

	Finest level	Background mesh	Final number of nodes	Final number of edges	Final number of cells	Finest L_∞ norm of error	CPU time (s)
Adaptive grid	1	41 × 41	2722	6028	2932	0.0046	2
		81 × 81	9940	21,920	10,804	0.00098	18
	2	23 × 23	2235	5506	2648	0.0043	2
		46 × 46	7540	18,516	9081	0.00091	16
Uniform grid		81 × 81	6561	12,960	6400	0.0046	6
		161 × 161	25,921	51,520	25,600	0.00092	90

periodic boundary condition is employed at all boundaries. The initial mesh is adaptively generated by a 6 × 6 background mesh (level 0) with the finest resolution level as 6. The initial values of the primitive variables for the two fluids are given by [2],

$$\rho_1 = 0.1\ \text{kg/m}^3, \quad u_1 = 1\ \text{m/s}, \quad p_1 = 1\ \text{Pa}, \quad \gamma_1 = 1.4, \quad \pi_1 = 0\ \text{Pa}, \quad \text{for } r < r_0, \tag{44}$$

$$\rho_2 = 1\ \text{kg/m}^3, \quad u_2 = 1\ \text{m/s}, \quad p_2 = 1\ \text{Pa}, \quad \gamma_2 = 1.6, \quad \pi_2 = 0\ \text{Pa}, \quad \text{for } r > r_0. \tag{45}$$

This initial condition indicates that there are no shock and other perturbation in the flow field. As a result, the inner fluid with circular shape should move with the constant velocity along the diagonal direction. Thus, it is also called material interface only problem. As reported by many researchers [2,16], it is a good test case to verify the solver and see whether there is no oscillation of pressure and velocity across the interface.

We plot the interface position at $t = 0$ s and $t = 0.36$ s as shown in Fig. 3. As stated above, the circular interface moves with a constant velocity. Hence, there is an analytical solution for the interface position. From Fig. 3, it can be easily observed that the calculated position and shape of the interface at $t = 0.36$ s agrees well with the predicted one. To see it more clearly, the surface plot of the density is also shown in Fig. 4. This figure clearly shows that the density around the interface is so sharp that it is nearly the same as the initial one. We also perform the simulation with the first-order of accuracy and the results are shown in Fig. 5. It is clear that the profile is not as sharp as that of the second-order scheme. This indicates that the adaptive algorithm with

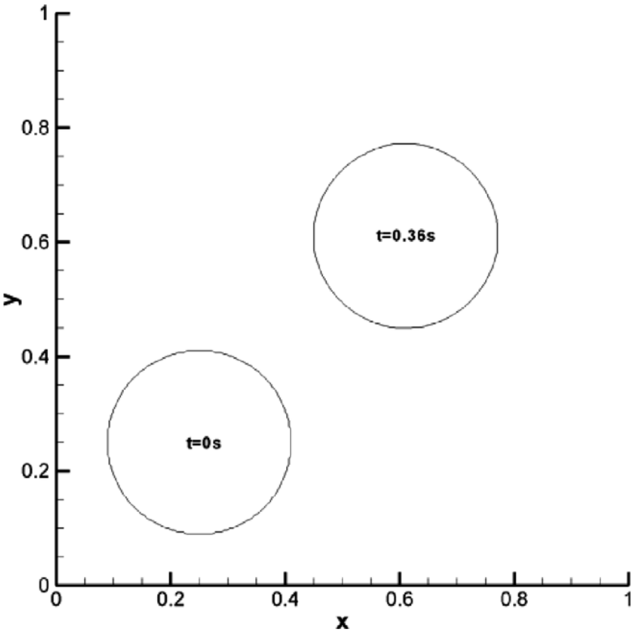


Fig. 3. Interface position of different time ($t = 0$ s and $t = 0.36$ s).

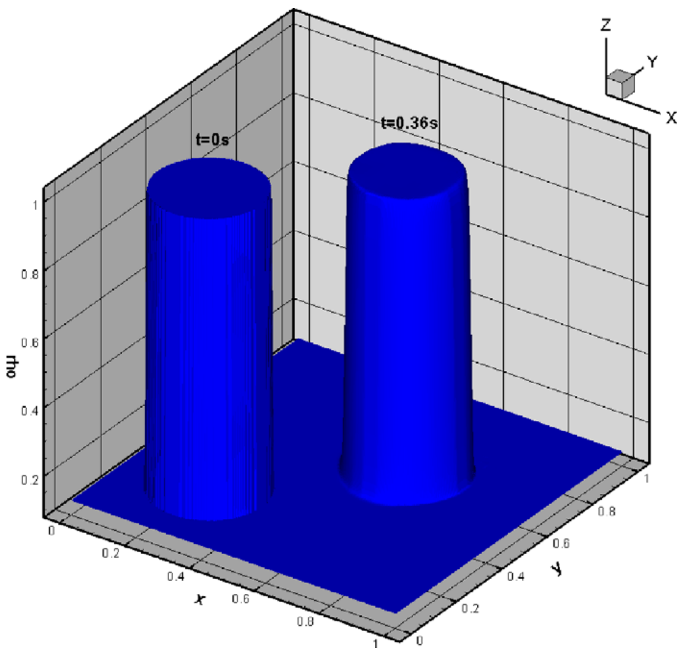


Fig. 4. Comparison of the surface density plot of interface only problem between time $t = 0.0$ s and $t = 0.36$ s by using second-order MUSCL scheme (max level is 6).

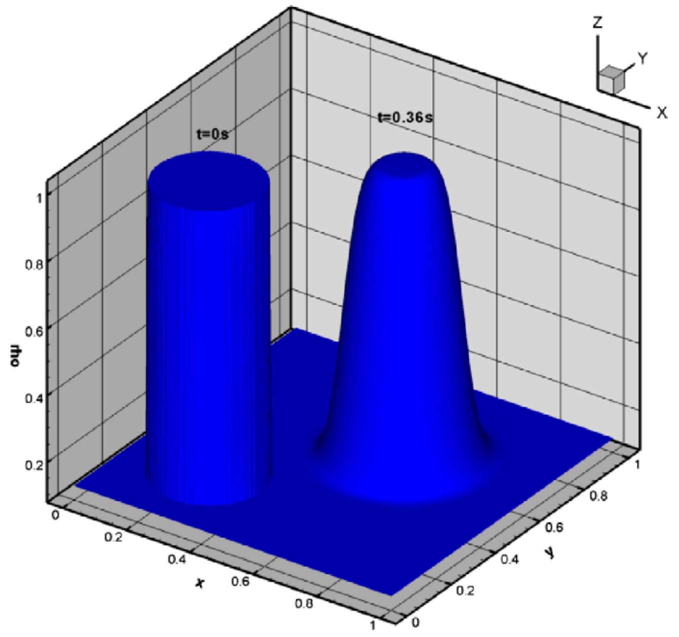


Fig. 5. Comparison of the surface density plot of interface only problem between time $t = 0.0$ s and $t = 0.36$ s by using first-order scheme (max level is 6).

the second-order of accuracy gives a better interface tracking. Besides, the computed pressure and velocity do not oscillate around the interface as shown in Figs. 6 and 7. This confirms the claim of oscillation-free of our adaptive method. The total nodes, leaf cells, and cells are 6747, 6316 and 8413, respectively, at $t = 0.36$ s. This

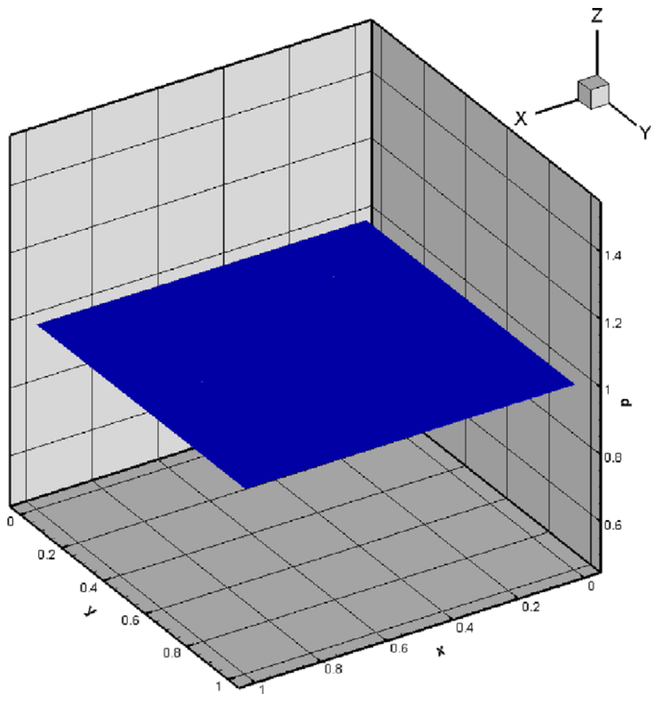


Fig. 6. Surface pressure plot of interface only problem at time $t = 0.36$ s (max level is 6).

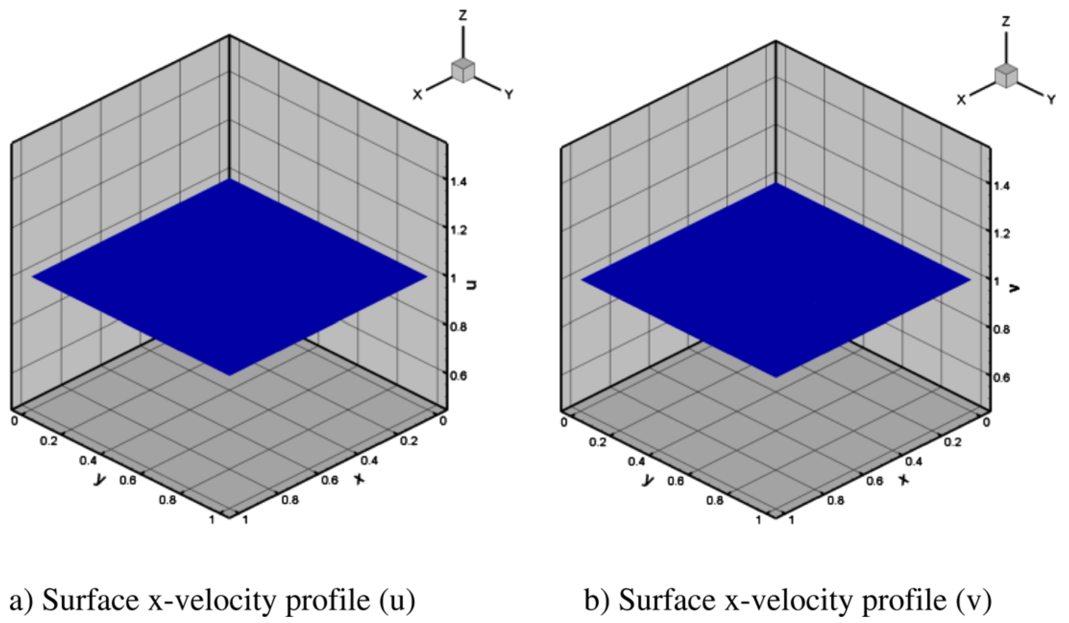


Fig. 7. Surface velocity plot of interface only problem at time $t = 0.36$ s (max level is 6).

cell number is smaller than that of structured mesh solver [2] with mesh size of 100×100. It is indicated that the surface plot of density by Shyue [2] is so diffused that it is similar to our result with the first order of accuracy (Fig. 5). This example does show that the adaptive algorithm is very accurate.

We can further test whether the oscillation-free is also valid on the unstructured mesh (with the finest resolution level 7) as shown in Fig. 8. The initial conditions are the same as those in Eqs. (44) and (45). In con-

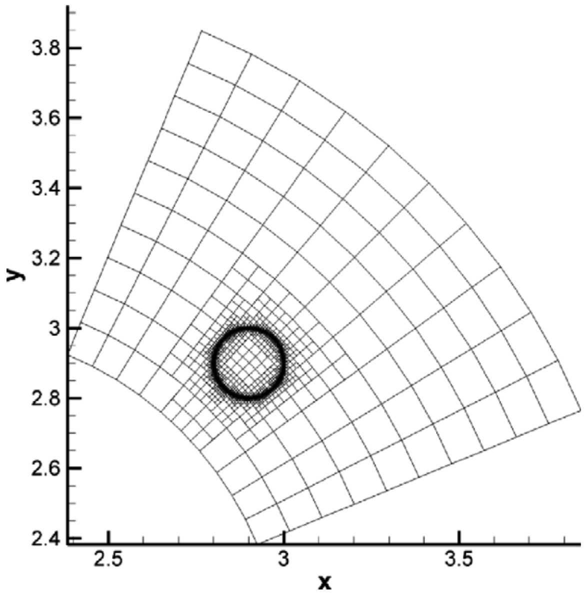


Fig. 8. Unstructured adaptive mesh for interface only problem.

trast, the extrapolation boundary conditions are set at all four boundaries. Similarly, we can draw the surface plot of the density at $t = 0$ s and at $t = 0.36$ s as shown in Fig. 9. The surface plot of pressure and velocity in Figs. 10 and 11 has no oscillation. These show that the algorithm is also valid on the unstructured mesh.

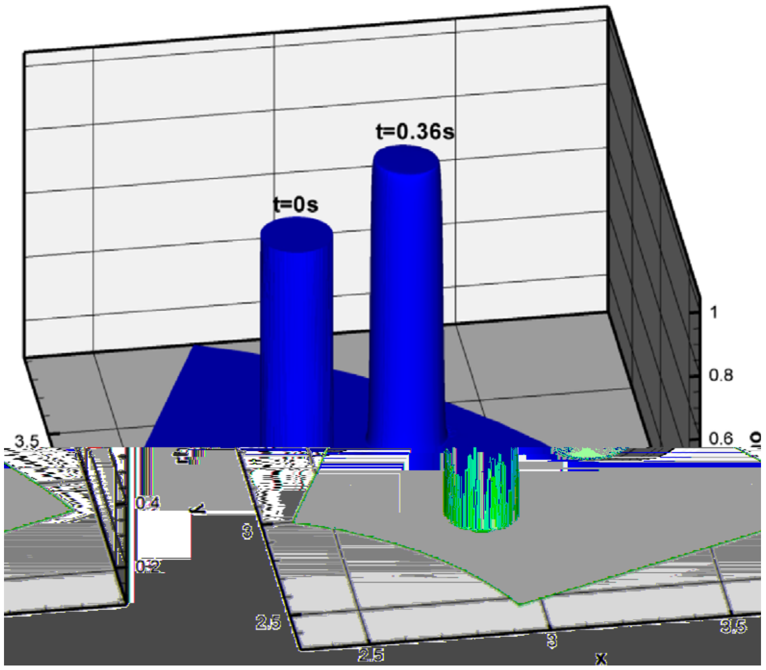


Fig. 9. Comparisons of the surface density plot of interface only problem between time $t = 0.0$ s and $t = 0.36$ s by using second-order MUSCL scheme (max level is 7).

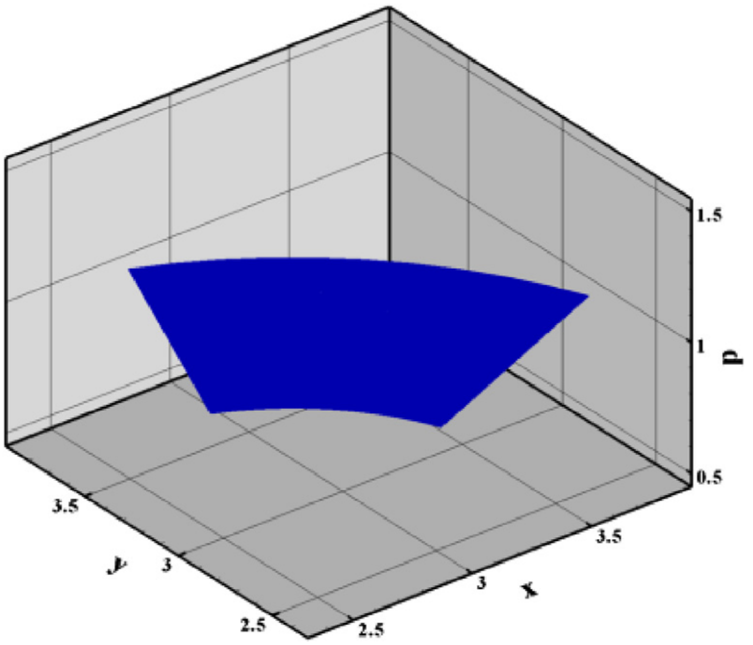


Fig. 10. Surface pressure plot of interface only problem at time $t = 0.36$ s (max level is 7).

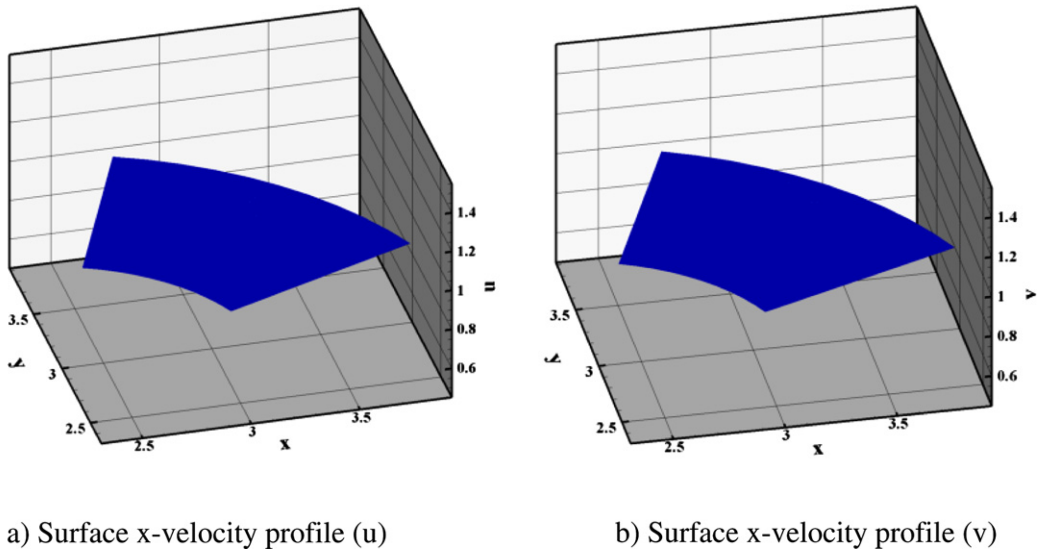


Fig. 11. Surface velocity plot of interface only problem at time $t = 0.36$ s (max level is 7).

4.3. Bubble explosion under the water

In this section, the efficiency and memory reduction of our method are investigated. Initially, one circular fluid (gas) with the radius $r_0 = 0.2$ m surrounded by another fluid is put at the center of the domain $[0,1] \times [0,1]\text{m}^2$ [2]. The periodical boundary conditions are employed at the top and bottom boundary, while the extrapolation boundary conditions are employed at the left and right boundary. The two fluids are initially at rest and there is a jump on the density, the pressure and the material parameters (γ, π) across the interface,

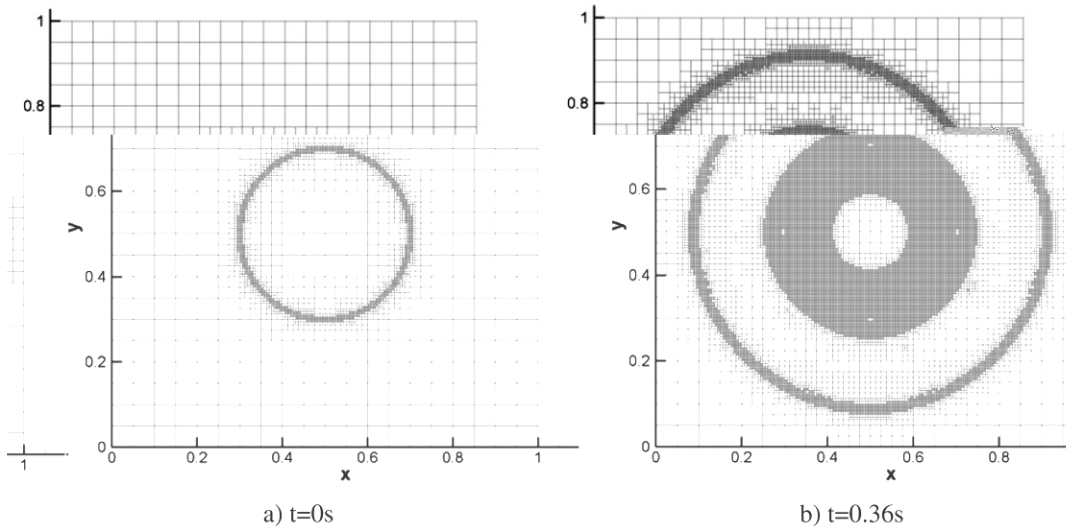


Fig. 12. Density contour for bubble explosion under water.

$$\rho_1 = 1.241\ \text{kg/m}^3,\quad u_1 = 0\ \text{m/s},\quad v_1 = 0\ \text{m/s},\quad p_1 = 2.753\ \text{Pa},\quad \gamma_1 = 1.4,\quad \pi_1 = 0\ \text{Pa},\quad \text{for } r < r_0, \tag{46}$$

$$\rho_2 = 0.991\ \text{kg/m}^3,\quad u_2 = 0\ \text{m/s},\quad v_1 = 0\ \text{m/s},\quad p_2 = 3.059\times 10^{-4}\ \text{Pa}\quad \gamma_2 = 5.5,$$
$$\pi_2 = 1.505\ \text{Pa}\quad \text{for } r > r_0. \tag{47}$$

To evaluate our method, numerical computation is performed on the adaptive mesh generated by a uniform background mesh (level is 0) of 20 × 20 with the finest resolution level as 4. The solution of this problem is composed by an outgoing shock wave traveling in the air, an incoming rarefaction wave traveling in the water and the material interface (contact discontinuity) lying in between these waves (Figs. 13–15). From the observation of the mesh distributions (Fig. 12), it is clear that the region with the shock wave and rarefaction wave is well refined and the region with smooth solution is coarsened. That is, our adaptive algorithm can appropriately generate the suitable mesh to reflect the important features of the flow. Thus, it is reasonable to expect

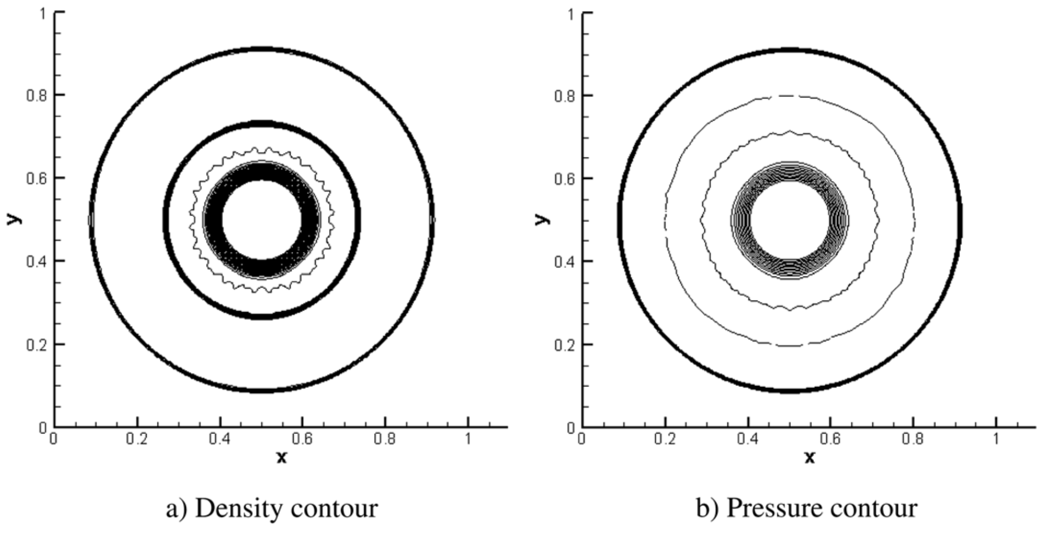


Fig. 13. Density and pressure contour for bubble explosion under water.

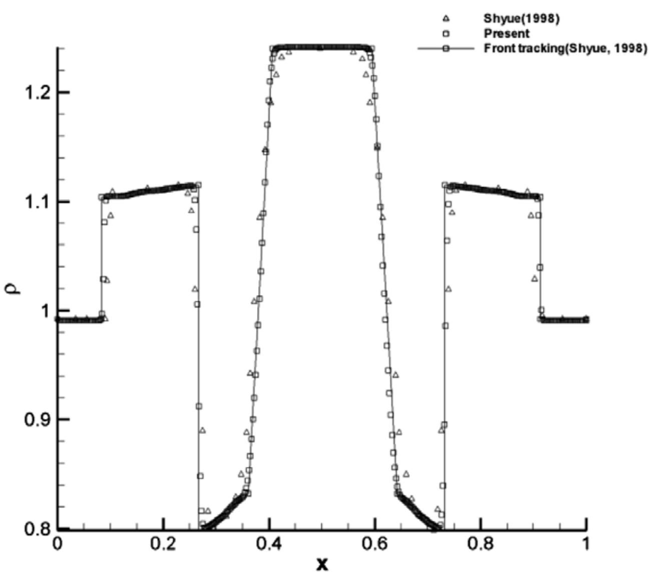


Fig. 14. Density profile in vertical centerline for the bubble explosion under water.

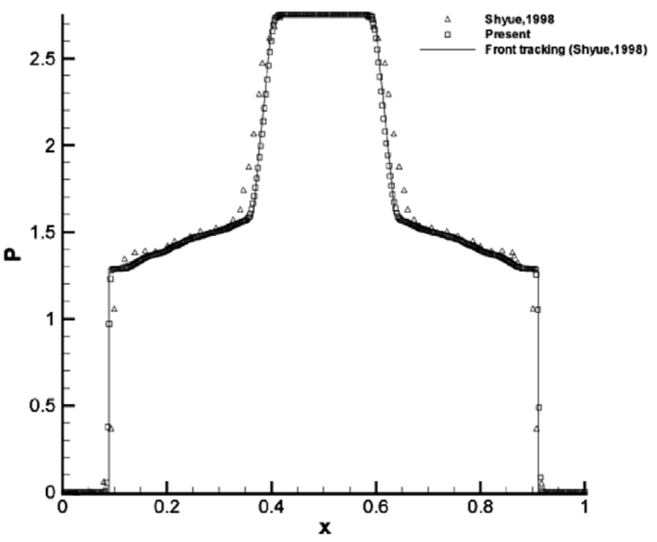


Fig. 15. Pressure profile in vertical centerline for the bubble explosion under water.

a favorable return of improved computational efficiency. From the density and pressure profiles along the center line (Figs. 14 and 15), good agreement was found between the present results and those of [2]. Note that the results of [2] in Figs. 14 and 15 are obtained from the curves of [2] by using the software of Marisoft Digitizer. As compared with high resolution wave propagation results on the fixed grid [2] and the front tracking algorithm [2], our adaptive algorithm requires much less total number of nodes to achieve an equivalently accurate solution.

4.4. Bubble-shock interaction

In this section, the bubble shock interaction with a large density ratio (1000) and pressure ratio (10,000) between the water and gas bubble is investigated. Initially, a left going planar shock wave with Mach number of 1.422 traveling in the water and a circular bubble gas with radius of 0.2 m are put at the left side (0.7 m, 0 m) of the shock wave [1,28]. The domain is $[0,1.2] \times [-0.5, 0.5]\mathrm{m}^2$ and the shock wave is located at 0.95 m. The pre-shock and post-shock fluids are initially at rest and the parameters such as density, velocity, pressure and the material parameters (γ, π) are listed as follows:

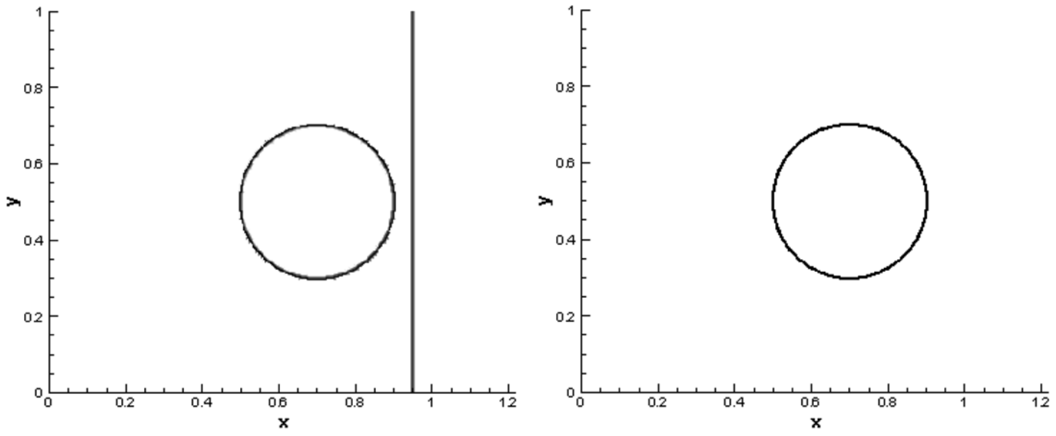
$$\rho_1 = 1000\ \mathrm{kg/m^3}, \quad u_1 = 0\ \mathrm{m/s}, \quad v_1 = 0\ \mathrm{m/s}, \quad p_1 = 1 \times 10^5\ \mathrm{Pa}, \quad \gamma_1 = 4.4, \quad \pi_1 = 6 \times 10^8\ \mathrm{Pa}, \tag{48}$$

$$\rho_3 = 1230\ \mathrm{kg/m^3}, \quad u_3 = -432.69\ \mathrm{m/s}, \quad v_3 = 0\ \mathrm{m/s}, \quad p_3 = 10^9\ \mathrm{Pa}, \quad \gamma_3 = 4.4, \quad \pi_3 = 6 \times 10^8\ \mathrm{Pa}. \tag{49}$$

The primitive variables for the bubble are set as,

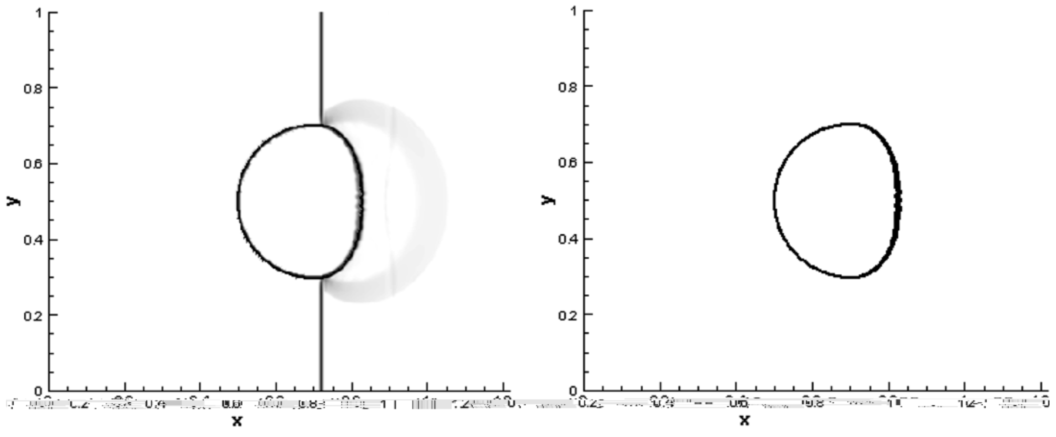
$$\rho_2 = 1.2\ \mathrm{kg/m^3}, \quad u_2 = 0\ \mathrm{m/s}, \quad v_2 = 0\ \mathrm{m/s}, \quad p_2 = 10^5\ \mathrm{Pa}, \quad \gamma_2 = 1.4, \quad \pi_2 = 0.0\ \mathrm{Pa}. \tag{50}$$

The reflective boundary conditions are employed at the top and bottom boundary, while the extrapolation boundary conditions are imposed at the left and right boundary. As stated by Shyue [28], this is a challenging problem due to the large pressure jump across the shock wave and the large ratio of the acoustic impedances of the liquid to gas.



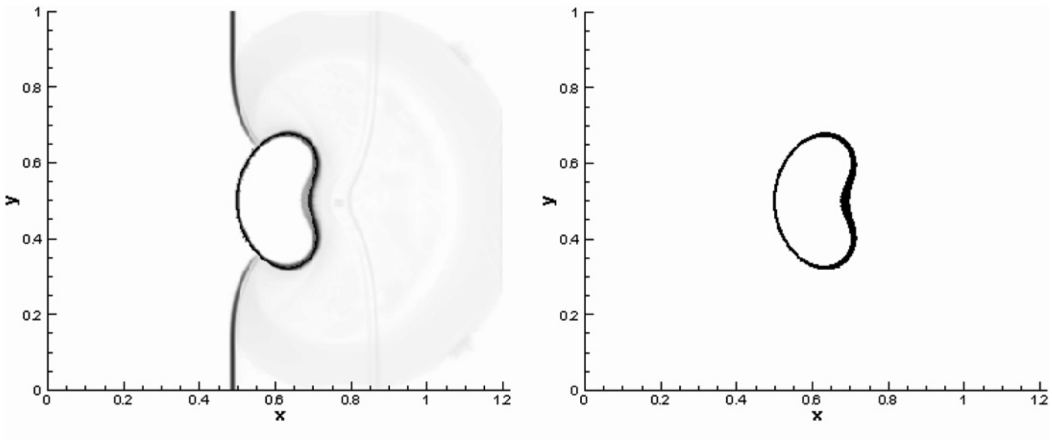
a) Density contour b) Beta contour

Fig. 16. Density and beta at $t = 0$ s.



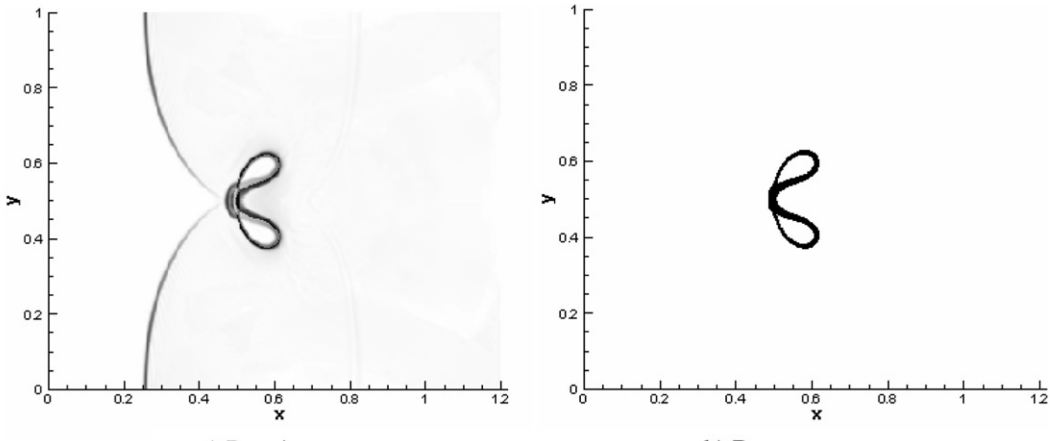
b) Beta contour a) Density contour

Fig. 17. Density and beta at $t = 0.0001$ s.



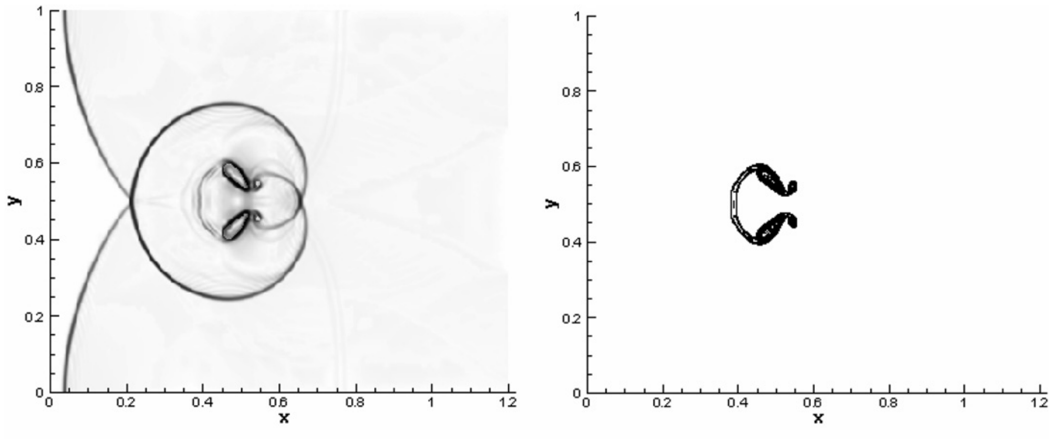
a) Density contour b) Beta contour

Fig. 18. Density and beta at $t = 0.0002$ s.



a) Density contour b) Beta contour

Fig. 19. Density and beta at $t = 0.0003$ s.



a) Density contour b) Beta contour

Fig. 20. Density and beta at $t = 0.0004$ s.

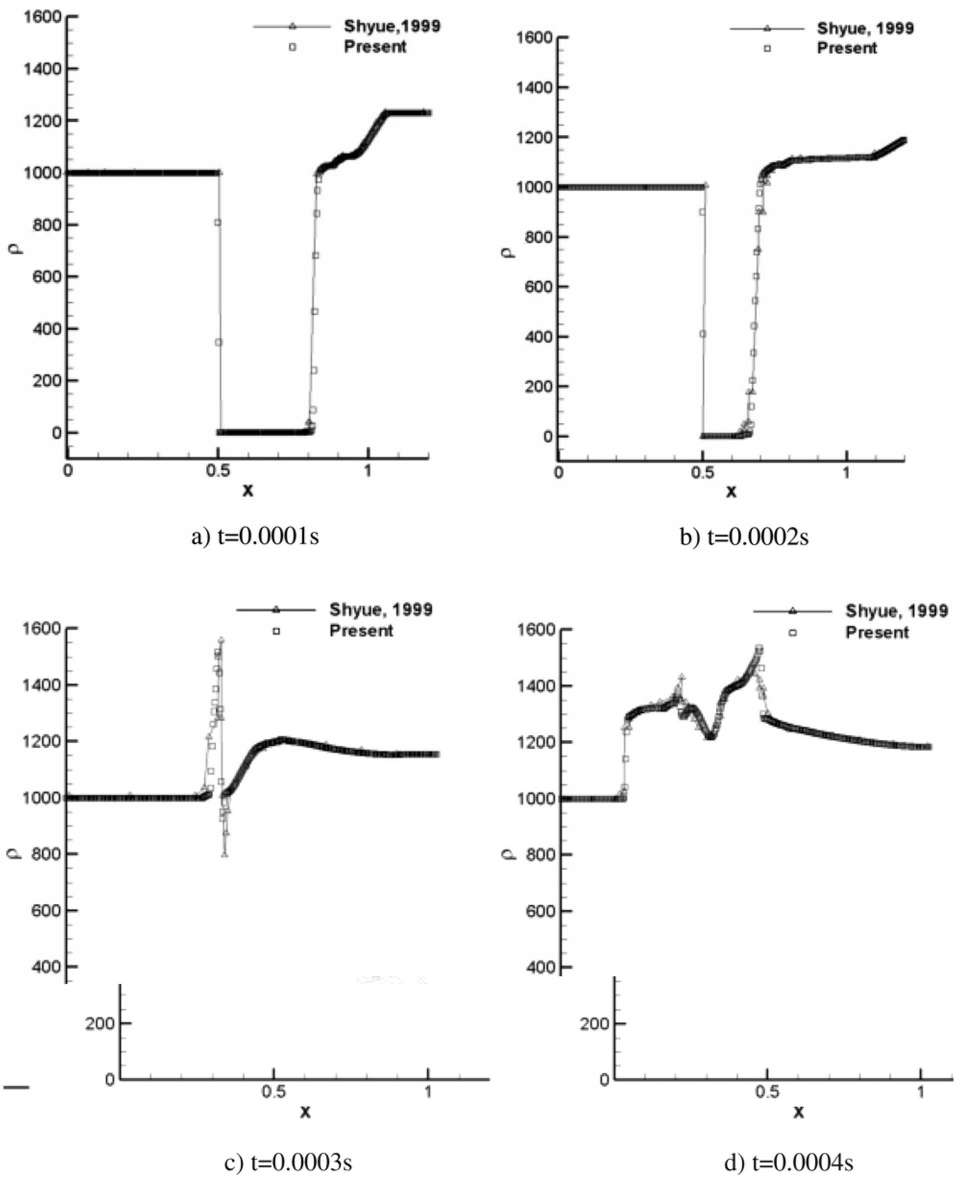
To see a clear flow structure, we plot the schlieren figure of density defined by

$$s_c = \exp\left(-k_c \frac{|\nabla\rho|_c}{\max_{cj}|\nabla\rho|_{cj}}\right) \tag{51}$$

with

$$k_c = \begin{cases} 6, & \text{if } \beta_c < 1 \\ 1, & \text{otherwise.} \end{cases} \tag{52}$$

The simulation is performed on the adaptive mesh generated by a uniform background mesh (level is 0) of 6×5 with the finest resolution level of 6. The results of density (displayed by schlieren figure) and the contour of β at different time from 0 s to 0.0004 s are plotted in Figs. 16–20. As can be seen in Fig. 16, the shock wave

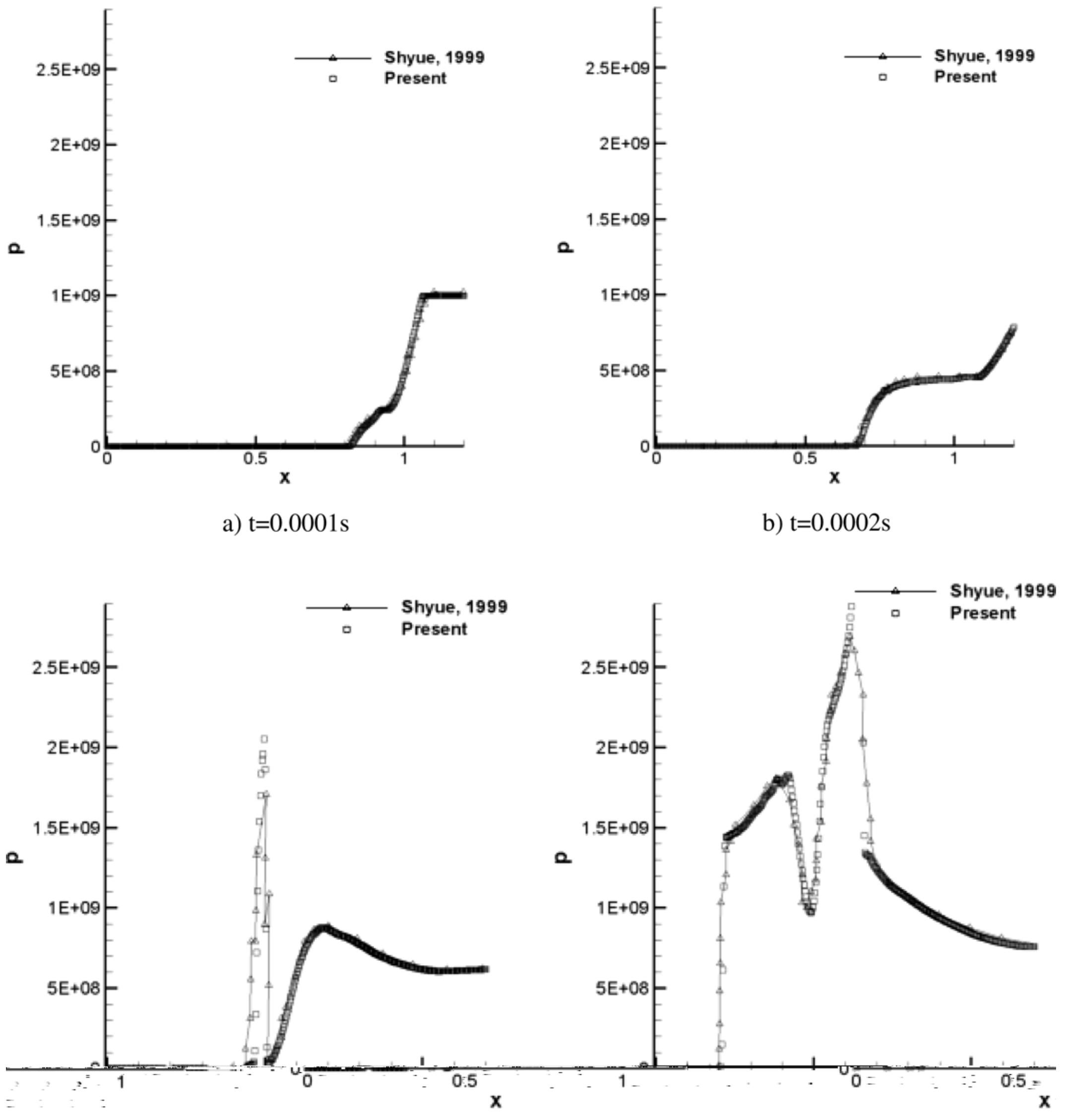


a) t=0.0001s b) t=0.0002s

c) t=0.0003s d) t=0.0004s

Fig. 21. Density profile along centerline at $y = 0$.

first propagates in the water. After the shock wave hits the bubble, a circular wave is generated and reflected from the interface (Fig. 17). Then the left-going shock wave continues to propagate through the bubble and the reflected circular wave moves outward as shown in Fig. 18. When the right moving circular wave hits the upper and lower boundary, the second reflected wave is generated (Fig. 19). The second reflected wave will interact with other waves to form a complex flow. Two small vortices are found to attach the bubble as shown in Fig. 20. The density and pressure profiles are plotted in Figs. 21 and 22. Also included in these two figures are the results of Shyue [28], which are obtained from the curves of [28] by using the software of Marisoft Digitizer. It is clear that our results agree well with those of Shyue [28]. This shows that our algorithm works well on complex flow field where shock waves, rarefaction wave and material interface are interacted with each other. From Fig. 23, it can be easily observed that the adaptive mesh can reflect the pattern of density contours as shown in Figs. 17 and 18. This implies that the current method can effectively capture the discontinuous flow structure such as shock wave and material interface.



a) t=0.0001s

b) t=0.0002s

0004s

c) t=0.0003s

d) t=0.

Fig. 22. Pressure profile along centerline at $y = 0$.

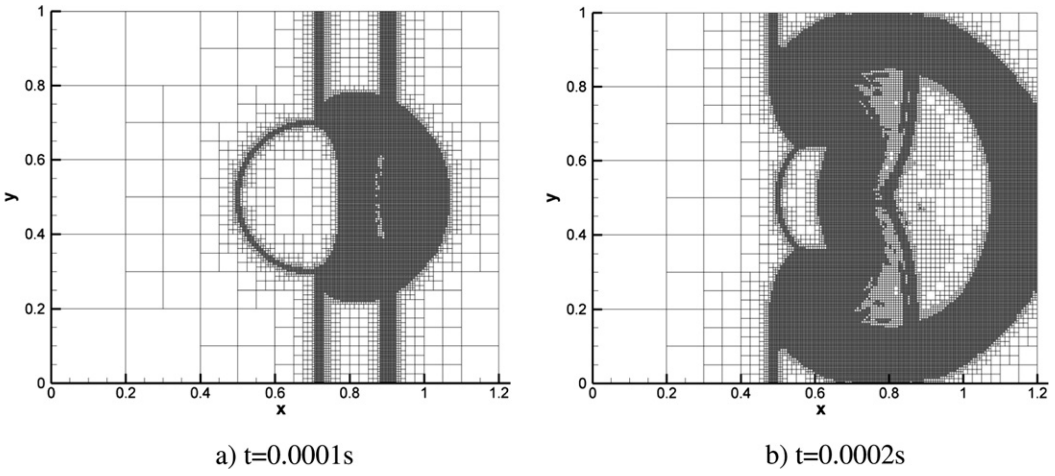


Fig. 23. Adaptive mesh for bubble-shock interaction problem.

4.5. Shock-interface interaction inside the cylindrical vessel

The final test case is the shock-interface interaction in the cylindrical vessel. The radius and center of the vessel are respectively 0.8 m and (0 m, 0 m). Initially, two ideal gases, the air and the helium, are separated by a planer interface. The vessel is then impulsively driven normal to the interface causing a curved shock to form along the compressive portion of the boundary and a rarefaction to form along the opposing portion of the boundary. The material interface is located at $x = 0$ initially, and the air is put at the left side. The parameters of air are listed as follows:

$$\rho_1 = 1\ \text{kg/m}^3,\quad u_1 = -1\ \text{m/s},\quad v_1 = 0\ \text{m/s},\quad p_1 = 1\ \text{Pa},\quad \gamma_1 = 1.4,\quad \pi_1 = 0\ \text{Pa}. \tag{53}$$

The helium is put at the right side with parameters as

$$\rho_2 = 0.138\ \text{kg/m}^3,\quad u_2 = -1\ \text{m/s},\quad v_2 = 0\ \text{m/s},\quad p_2 = 1\ \text{Pa},\quad \gamma_2 = 1.67,\quad \pi_2 = 0\ \text{Pa}. \tag{54}$$

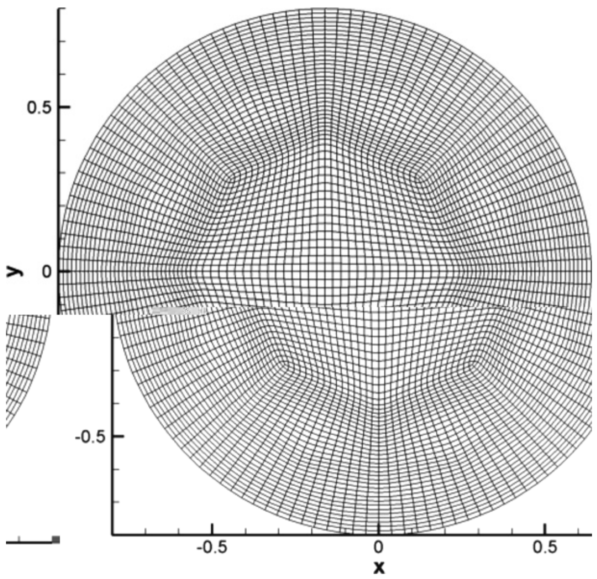


Fig. 24. The back ground mesh for cylindrical vessel.

The background mesh for the simulation is shown in Fig. 24. The reflective boundary conditions are employed at the boundary. Figs. 25–28 display the adaptive mesh and density contours at different times. The solution of this problem is composed by a right-going semi-circular shock wave traveling in the air, a left-going rarefaction wave traveling in the helium and the material interface (contact discontinuity) lying in between these waves. As shown in Fig. 25, the shock wave is initially formed on the left and propagates to the right towards the material interface and the rarefaction wave moves towards the left when time is 0.25 s. It can be easily observed that the adaptive mesh is refined locally around these waves and interface. After that, the shock wave continues to move to right and forms a diverging shock which is bow shape and is clearly captured by the adaptive mesh as shown in Fig. 26a when time is 0.5 s. It also drives the interface to move towards right as shown in Fig. 26. The right moving shock wave will reflect back by the fixed vessel. This reflected wave will

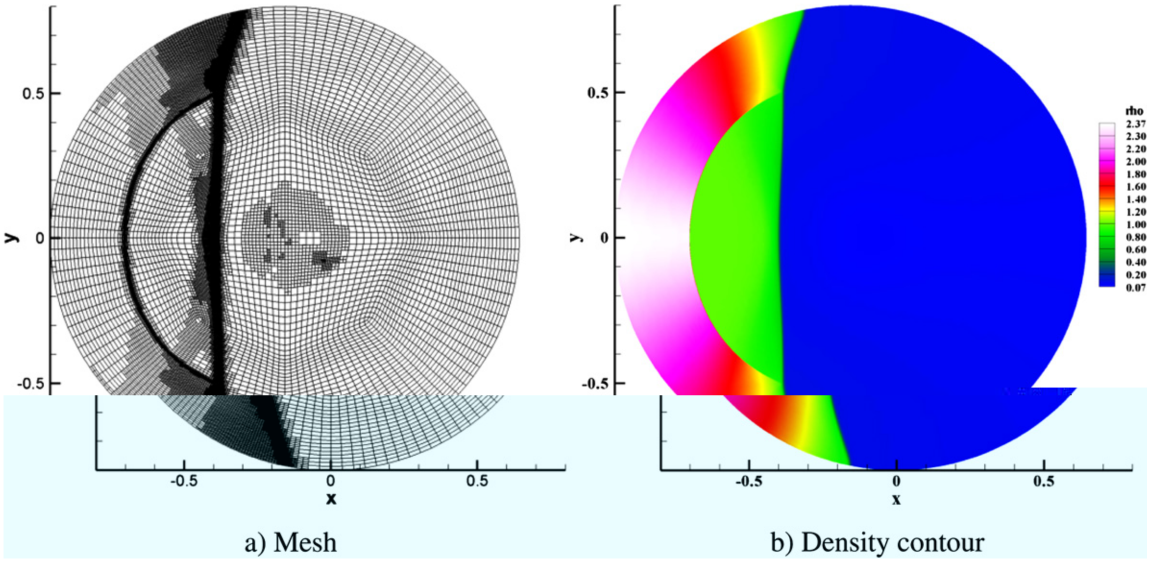


Fig. 25. Mesh and density contour at time $t = 0.25$ s (max level is 5).

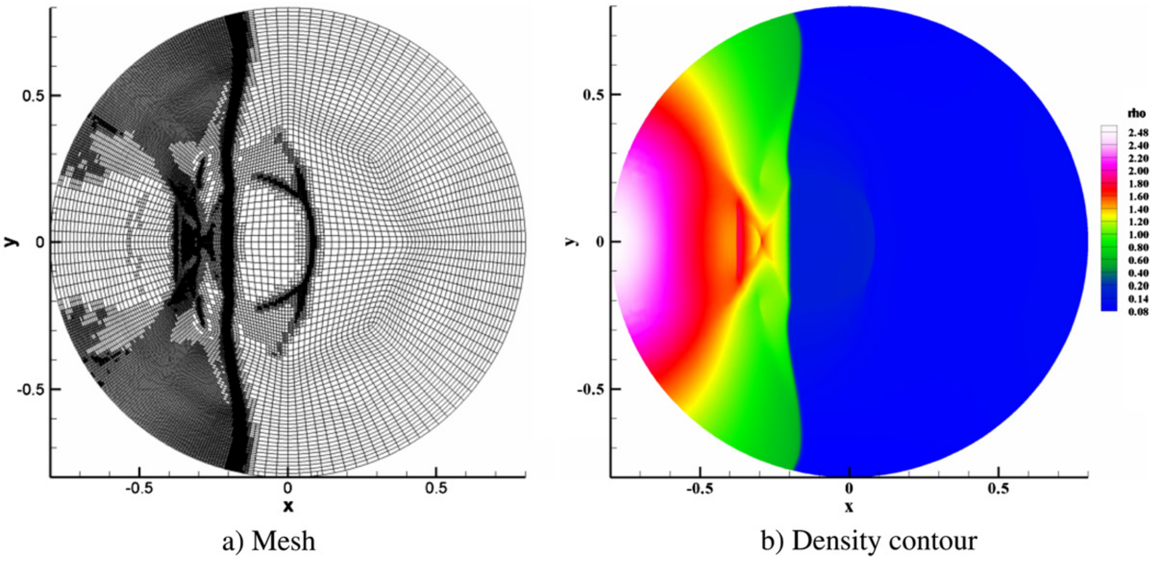


Fig. 26. Mesh and density contour at time $t = 0.5$ s (max level is 5).

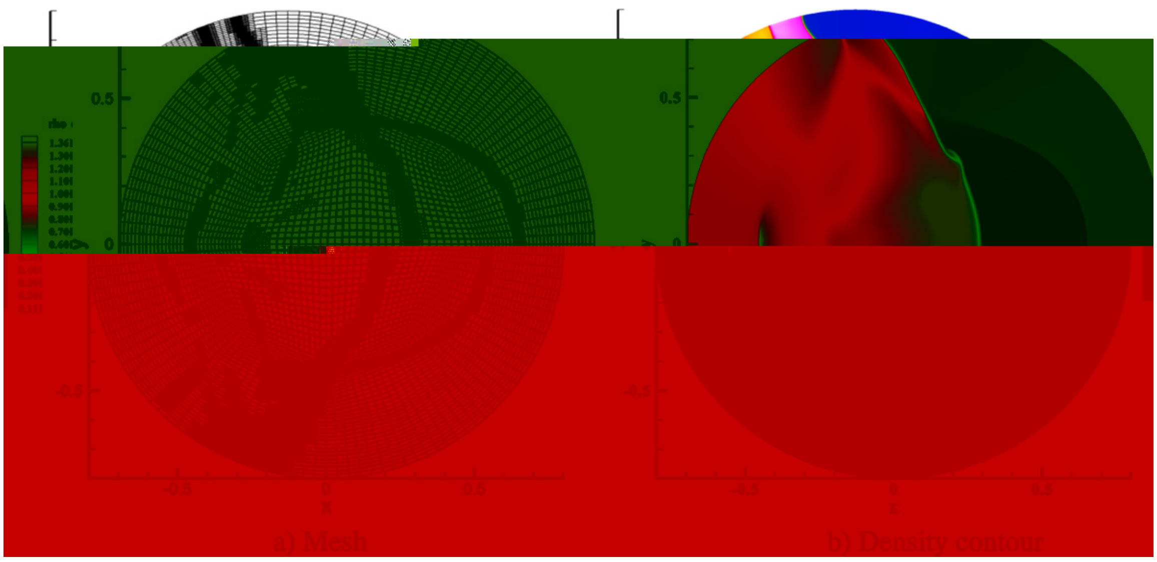


Fig. 27. Mesh and density contour at time $t = 0.75$ s (max level is 5).

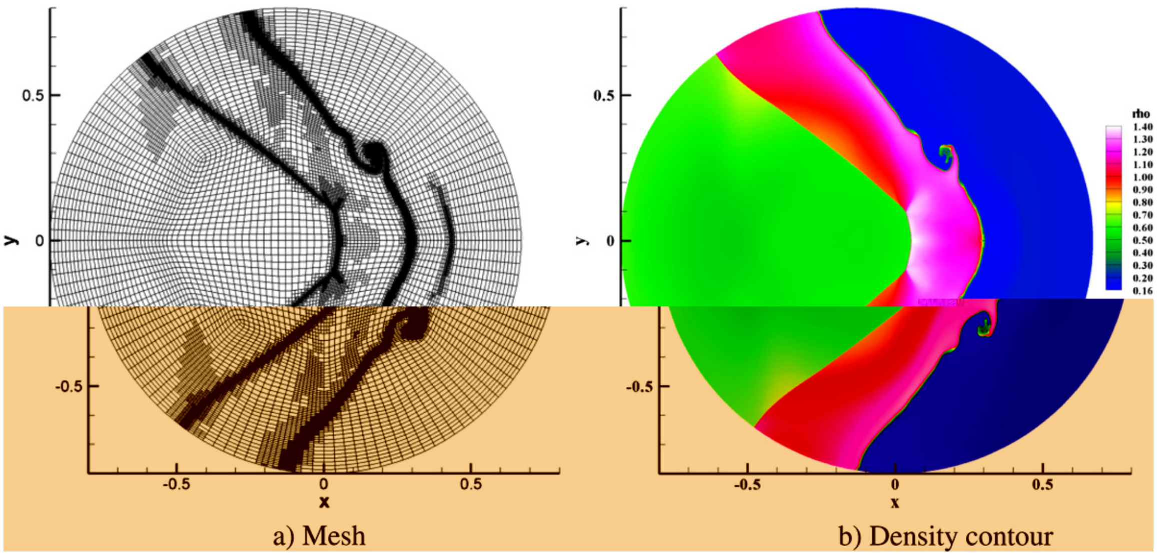


Fig. 28. Mesh and density contour at time $t = 1$ s (max level is 5).

then travel in the opposite direction (left going), and collide with the interface and move to the air again. The mesh and density contours at time of 0.75 s are shown in Fig. 27. As the time goes on, the interface starts to roll up and form the Richtmyer–Meshkov (RM) instability. At the same time, the left-going reflected shock wave will be focused at the left-hand side of the interface. These phenomena can be clearly seen in Fig. 28. As RM instability develops, tiny structures will be appeared on the interface. As shown in Fig. 28, these tiny structures are well captured by our adaptive mesh.

Note that this problem has been previously solved by Banks et al. [29] using the adaptive Cartesian grid together with overlapping mesh. It was found that our density contours as shown in Figs. 25–28 well reveal the flow pattern as shown by the Schlieren images of Banks et al. [29]. Fig. 29[1] further compares the colored

[1] For interpretation of color in Fig 29, the reader is referred to the web version of this article.

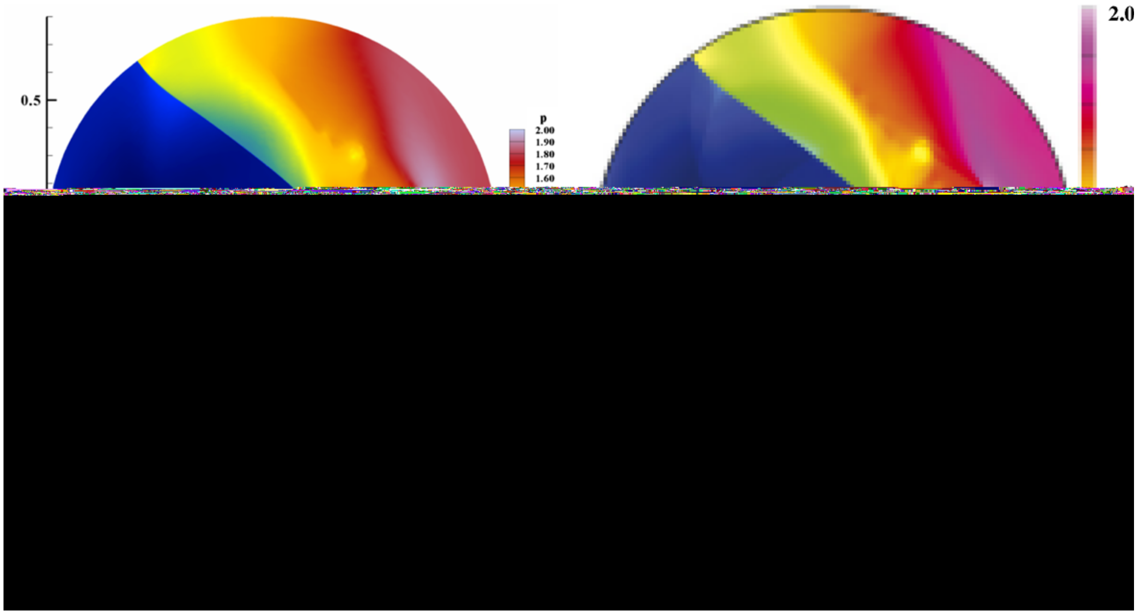


Fig. 29. Comparison of pressure contour at time $t = 1$ s.

pressure contours at time of 1 s between the present results and those of Banks et al. [29]. Although the color mapping (the mapping between numerical results and the RGB color) may be different between the two works, the present results agree very well with those of Banks et al. [29].

5. Conclusions

An object-oriented and quadrilateral-mesh based solution adaptive method for two-dimensional compressible multi-fluid flows is proposed in this paper. The approach is able to automatically and efficiently adjust the local mesh density to reflect the transient behavior of the solution. Due to the well-designed cell-edge-node data structure and double link list, the inserting and removing of the existing objects (nodes, edges and cells) are independent of the number of objects and only of the complexity of $\boldsymbol{O(1)}$. The memory requirement is greatly reduced since the unused objects are freed from the memory. Besides, the HLLC scheme is extended to solving compressible multi-fluid flows on unstructured mesh. To examine the performance of present adaptive algorithm, five examples have been carried out. They are vortex evolution, interface only, bubble explosion under water, bubble shock interaction and shock-interface interaction in the cylindrical vessel cases. Numerical results show that the second-order convergence has been achieved by the present adaptive method. They also indicate that there is no oscillation of pressure and velocity across the interface and it is feasible in solving compressible multi-fluid flows with large density ratio (1000) and strong shock wave (the pressure ratio is 10,000) interaction with the interface.

References

[1] G. Allaire, S. Clerc, S. Kokh, A five-equation model for the simulation of interfaces between compressible fluids, J. Comput. Phys. 181 (2002) 577–616.

[2] K.M. Shyue, An efficient shock-capturing algorithm for compressible multi-component problems, J. Comput. Phys. 142 (1998) 208–242.

[3] R.R. Nourgaliev, T.N. Dinh, T.G. Theofanous, Adaptive characteristics-based matching for compressible multifluid dynamics, J. Comput. Phys. 213 (2006) 500–529.

[4] W.M. Cao, W.Z. Huang, R.D. Russell, An r-adaptive finite element method based upon moving mesh PDEs, J. Comput. Phys. 149 (1999) 221–244.

[5] R. Li, T. Tang, P. Zhang, Moving mesh methods in multiple dimensions based on harmonic maps, J. Comput. Phys. 170 (2001) 562–588.
[6] M.J. Berger, J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, J. Comput. Phys. 53 (1984) 484–512.
[7] R.D. Hornung, J.A. Trangenstein, Adaptive mesh refinement and multilevel iteration for flow in porous media, J. Comput. Phys. 136 (1997) 522–545.
[8] S. Popinet, Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries, J. Comput. Phys. 190 (2003) 572–600.
[9] Q. Liang, A.G.L. Borthwick, G. Stelling, Simulation of dam and dyke-break hydrodynamics on dynamically adaptive quadtree grids, Int. J. Num. Meth. Fluids 46 (2004) 127–162.
[10] E.F. Charlton, K.G. Powell, An octree solution to conservation-laws over arbitrary regions (OSCAR), AIAA Paper, 1997, pp. 97–0198.
[11] M.J. Berger, R. LeVeque, Adaptive mesh refinement for two-dimensional hyperbolic systems and the AMRCLAW software, SIAM J. Numer. Anal. 35 (1998) 2298–2316.
[12] M.J. Berger, P. Colella, Local adaptive mesh refinement for shock hydrodynamics, J. Comput. Phys. 82 (1998) 64–84.
[13] G.H. Schmidt, F.J. Jacobs, Adaptive local grid refinement and multi-grid in numerical reservoir simulation, J. Comput. Phys. 77 (1988) 140–165.
[14] M. Sun, K. Takayama, Conservative smoothing on an adaptive quadrilateral grid, J. Comput. Phys. 150 (1999) 143–180.
[15] W.A. Keats, F.S. Lien, Two-dimensional anisotropic cartesian mesh adaptation for the compressible Euler equations, Int. J. Num. Meth. Fluids 46 (2004) 1099–1125.
[16] R. Abgrall, S. Karni, Computations of compressible multifluids, J. Comput. Phys. 169 (2001) 594–623.
[17] M. Sun, Numerical and experimental studies of shock wave interaction with bodies, Ph.D. Thesis, Tohoku University, 1998.
[18] R. Abgrall, B. Nkonga, R. Saurel, Efficient numerical approximation of compressible multi-material flow for unstructured meshes, Comp. Fluids 32 (4) (2003) 571–605.
[19] E.F. Toro, M. Spruce, W. Speares, Restoration of the contact surface in the HLL Riemann solver, Shock Waves 4 (1994) 25–34.
[20] P. Batten, N. Clarke, C. Lambert, D.M. Causon, On the choice of wavespeeds for the HLLC Riemann solver, SIAM J. Sci. Comput. 18 (6) (1997) 1553–1570.
[21] E. Johnsen, T. Colonius, Implementation of WENO schemes for compressible multicomponent flow problems, J. Comput. Phys. 219 (2006) 715–732.
[22] J. Massoni, R. Saurel, B. Nkonga, R. Abgrall, Propositions de méthodes et modèles eulériens pour les problèmes á interfaces entre fluides compressibles en présence de transfert de chaleur, Int. J. Heat Mass Transfer 45 (6) (2002) 1287–1307.
[23] B. Van Lear, Towards the ultimate conservative difference scheme V.A Second order sequel to Godunov's method, J. Comput. Phys. 32 (1979) 101–136.
[24] F.E. Ham, F.S. Lien, A.B. Strong, A cartesian grid method with transient anisotropic adaptation, J. Comput. Phys. 179 (2002) 469–494.
[25] Stefan Schirra, Remco Veltkamp, Mariette Yvinec (Eds.), CGAL Reference Manual 2, 2000, pp. 129–179.
[26] Z.J. Wang, L. Zhang, Y. Liu, Spectral (finite) volume method for conservation laws on unstructured grids IV: extension to two-dimensional Euler equations, J. Comput. Phys. 194 (2) (2004) 716–741.
[27] M. Sun, K. Takayama, Error localization in solution-adaptive grid methods, J. Comput. Phys. 190 (2003) 346–350.
[28] K.M. Shyue, A fluid-mixture type algorithm for compressible multicomponent flow with van der Waals equation of state, J. Comput. Phys. 156 (1999) 43–88.
[29] J.W. Banks, D.W. Schwendeman, A.K. Kapila, W.D. Henshaw, A high-resolution Godunov method for compressible multi-material flow on overlapping grids, J. Comput. Phys. 223 (2007) 262–297.