Researches on the Fast Determination of Cell Type of Cartesian Grid

Xueliang Li

Central South University <https://orcid.org/0000-0002-5057-3937>

Lin Bi (✉ bzbaby1010@163.com)

China Aerodynamics Research and Development Center

Shuang Meng

Central South University

Hongkang Liu

Central South University

Tiantian Wang

Central South University

Xianxu Yuan

China Aerodynamics Research and Development Center

Research Article

Keywords: Cartesian grid, Grid generation, Painting algorithm, K-dimensional tree, Axis-aligned bounding box

Posted Date: July 27th, 2021

DOI: <https://doi.org/10.21203/rs.3.rs-725468/v1>

License: This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

RESEARCH

Researches on the fast determination of cell type of Cartesian grid

Xueliang Li^{1,3}, Lin Bi^{*}, Shuang Meng³, Hongkang Liu³, Tiantian Wang⁴ and Xianxu Yuan²

*Correspondence:

bzbaby1010@163.com

¹ Stat Key Laboratory of

Aerodynamics, China

Aerodynamics Research and

Development Center, Mianyang,

China

Full list of author information is available at the end of the article

Abstract

The relationship between the spatial cell and the object is unknown for the Cartesian grid using the immersed boundary method. For the researches about complex geometry or multi-body relative motion, grid generation is a very time-consuming work, and the consumption is mainly concentrated in the position determination of the Cartesian cells, which we called the cell type determination. In this study, based on the axis-aligned bounding box method and the ray casting method, we employed the dot product method and the painting algorithm to investigate the acceleration method for Cartesian grid generation. The octree structure is used to store the Cartesian cells, and the k-dimensional tree is used to store the object surface. These data management strategy can minimize the CPU's resource while have a small memory usage. The grid generation results show that the strategy we proposed has a high efficiency and well robustness, and the time consume can reduce more than 50% compare with the original method. When dealing with a enough complex problem, the time consume can even reaches several orders of magnitude difference compared with the original method.

Keywords: Cartesian grid; Grid generation; Painting algorithm; K-dimensional tree; Axis-aligned bounding box

1 Introduction

The traditional body-fit structured/unstructured meshes have great advantages in solving problems using complex geometries. However, the mesh generation for those meshes may cost a lot of manual time to get a enough good quality to ensure the stabilization of simulation [1, 2]. In particular for the relative motion simulations using multi-geometries, the meshes have to be deformed or regenerated at each time step [3–5].

Cartesian grid can neglect those problems due to the non-body-fitted characteristic. The hexahedral cells in domain intersect with solid walls, change the problem of generating a body-fitted mesh to a more general problem of computing and characterizing intersections between hexahedral cells and the geometry [6–9]. Thus, all difficulties associated with the flow field solving using Cartesian grid are restricted to the problem of flux reconstruction near solid walls. The immersed boundary technique [10, 11] is one of the most common approaches for this problem. To apply the immersed boundary technique, the position relationship between Cartesian cells and the objects should be determined into 4 types: non-intersecting outside cell, intersecting outside cell, non-intersecting inside cell, and intersecting inside cell.

The conventional cell type determination is based on the ray casting method [12, 13]. The intersection relationship can be determined by the relative position

between the eight vertices of cell and the objects. The inside/outside relationship can be determined by the relative position between the cell center and the objects. However, due to the special cases that the ray might tangent to the object surface [14], multiple rays or other auxiliary methods [15] should be used to improve the robustness. As the result, this method will usually facing the problem of a huge computational consumption [16]. Axis-aligned bounding box (AABB) method [17] is a robust and efficient method for intersection determinations between cells and objects, but it cannot be used to determine whether the cell is inside or outside the object.

On the basis of the two methods above, the focus of this paper is to find a more efficient and robust cell type determination method for Cartesian grid generation. By using the dot product method, the intersecting object surface obtained by AABB method is used for a quick inside/outside determination for intersecting cells. At the same time, the painting algorithm is adopted to mark the non-intersecting cells. The k-dimensional tree (KDT) is used to store the object surface for higher query efficiency.

This paper is organized as follows. First, represent the purpose of this article and discuss the relevant research status, Sect.1. Second, introduce the original Cartesian grid generation strategy, Sect.2. The basic cell type determination methods, including the AABB method and ray casting method are described in Sect.3. Then, analyze the efficiency of original methods and propose the improvement grid generation strategy, Sect.4. The accelerating method developed in this paper including DP method, painting algorithm, and KDT data structure are introduced in Sect.5. Finally, we compared the accelerating performance between new methods and original methods, Sect.6.1. Robustness and efficiency of the new method is examined by different complex models in Sect.6.2. Also the comparison of grid generation efficiency with other literatures is performed in Sect.6.2. The conclusions and future work are presented in Sect.7.

2 Grid generation strategy

The data structure for Cartesian grid has two typical types: unstructured [18, 19] and octree [20–22]. The octree structure has minimum memory requirement as well as a straightforward grid hierarchy [23]. In this paper, the octree data structure is used for the Cartesian grid storage. The general strategy of the process of Cartesian grid generation with octree is given as follow.

- a.* Generate the computational domain specified by user, see Fig. 1(a).
- b.* Generate the background Cartesian grid, divide the computational domain by equal space, see Fig. 1(b).
- c.* Determine intersection by the AABB method for all Cartesian cells, mark the intersecting cells and continuously refine them, until the intersecting cells reaches the maximum layer level, see Fig. 1(c-d).
- d.* Uses the ray casting method to traverse and determine the inside/outside relationship for all cells, mark the cell as non-intersecting outside cell, intersecting outside cell, non-intersecting inside cell, and intersecting inside cell, see Fig. 1(e-f).

- 65 e. Blank the inside cells, release the memory, see Fig. 1(g).
- 66 f. Perform grid smoothing on all non-intersecting outside cells, make sure the level
 67 differences for all adjacent cells are not more than 1, and eliminate the hole cell,
 68 see Fig. 1(h).

Figure 1 The general strategy for Cartesian grid generation. (a) Computational domain generation; (b) Back-ground grid generation; (c-d) Intersecting determination and refinement; (e-f) Inside/outside determination; (g) Blank the inside cells; (h) Grid smoothing.

69 3 Cell type determination

70 3.1 Intersecting determination

71 Since the Cartesian cell has a axis-aligned bounding box (AABB) form, a quick
 72 intersecting determination method can be employed based on the separating axis
 73 theorem [17]. This theorem needs to test 13 axes in the implementation process.
 74 The first three axes are the normal axes of the AABB (e_x , e_y , e_z in Fig. 2), which
 75 are used to determine whether the triangle overlaps with the AABB. The fourth
 76 axis is the normal direction of the triangle (n in Fig. 2) used to determine whether
 77 the AABB has intersected with the plane of the triangle. The last nine axes are the
 78 cross product of e_x , e_y , e_z with the sides of the triangle (l_1 , l_2 , l_3 in Fig. 2). If all
 79 the tests for the 13 axes above have passed, it indicates that there is no separation
 80 axis, the AABB does not intersected with the triangle. Otherwise, if any separation
 81 axis presents, the test ends and the cell is intersected with the triangle.

82 However, this method can only used to identify whether the cell is or is not
 83 intersected with the object. Other method is needed for the determination of the
 84 inside/outside relationship of cells.

85 3.2 Inside/outside determination

86 To determine the inside/outside relationship of the Cartesian cells, the ray cast-
 87 ing method [24] and the winding number approach [25] are the usually methods
 88 in most Cartesian grid generation programs. While both approaches conceptually
 89 straightforward, they are considerably different computationally. Computation of
 90 the winding number involves a lot of floating-point computations which are prone
 91 to rounding error. Thus, the ray casting method is used in this paper.

92 The basic principle of the ray casting method is shown in Fig. 3(a). A ray D in any
 93 selected direction is emitted from the cell center O . To determine the inside/outside
 94 relationship, only a simply count for the number of intersections of D with objects
 95 surface is needed. If point O lies outside the object, the number of intersections
 96 is even. If point O lies inside, the number of intersections results odd. However,
 97 this method have a special case as shown in Fig. 3(b). While ray D is just passing
 98 the inflection point of the object, the count of intersections will lead to a opposite
 99 situation. In order to avoid this problem, at least two different rays are needed in
 100 application. If the results of two rays are consistent, the results credible. Otherwise,
 make a third determination is needed.

101 Fig. 3 only gives a illustration in 2-dimensions. In the situation of 3-dimensions, the
 102 determinations between ray and triangular elements of the object are needed. Moller
 103 [26] presented a fast, minimum storage algorithm for the determination of ray-
 104 triangle intersections. In the Moller's algorithm only three vertices of the triangle
 105 are stored and two times of cross product, four times of dot product are computed.
 106 In this paper the Moller's algorithm is used to saving the memory and speed up the
 107 cay casting progress.
 108

109 4 Efficiency analysis and process improvement

110 Combine with the AABB method and the ray casting method in Sect.3, we can
 111 get an accurately program to determine the cell type of Cartesian cells. However,
 112 these methods are not efficiency enough. For N_{cells} Cartesian cells and N_{tri} tri-
 113 angle elements, the number of determinations for two methods above are of order
 114 $O(N_{cells} \cdot N_{tri})$. Which means the computational time will increasing sharply with
 115 the increasing of the count of triangle elements and Cartesian cells. Especially the

ray casting method, at least twice determinations for each cell is needed. And each determination needs a traversal for all triangle elements to get the total number of intersections.

To saving the consumption of computational resources, we make the improvement in two aspects: reduce the times of determinations, and reduce the time complexity of the algorithm. We divide the inside/outside determination process into two parts to avoid the use of ray casting method: the determination for the intersecting cells by a simply dot product, and a quick painting process for the non-intersecting cells. Also a efficient data structure is used to order the elements of object surface.

By using those measures, the Cartesian grid generation strategy changes. Based on the original method (Fig. 1), the divergence started from step (c) in Sect. 2:

c. Intersecting determination and refinement. If a cell is intersected with a triangle surface, the dot product method is used for the inside/outside determination. Then refine this intersecting cell. The process above will keep repeating until reaches the maximum layer level. See Fig. 4(a-b).

d. Find a outside cell as the painting source by the ray casting method, then paint all non-intersecting outside cells. See Fig. 4(c-d).

e. Blank the inside cells, perform the grid smoothing, same with the step (e-f) in Sect. 2.

Figure 4 The improved Cartesian grid generation process, start from Fig. 1(b). (a-b) Intersection determination, inside/outside determination and refinement for the intersecting cells; (c-d) Find a outside painting source, paint all non-intersecting outside cells.

5 Accelerating method

5.1 Dot product method

The dot product method is used to determine the position of cell center under the situation that any intersecting triangle is known. The position is determined by the dot product result obtained from two vectors. One vector $\mathbf{L}$ is defined from the intersecting triangle to the cell center, and the other vector $\mathbf{n}$ is defined as the outer normal of the intersecting triangle (as shown in Fig. 5). The determination process is as follow:

The outer normal of the intersecting triangle is given by:

$$\mathbf{n} = (\mathbf{V2} - \mathbf{V1}) \times (\mathbf{V3} - \mathbf{V2}) \quad (1)$$

Here $V1, V2, V3$ are three vertices of the triangle. And the vector from triangle to the cell center can be defined as:

$$L = P - O \quad (2)$$

Here O is any point both inside the cell and the triangle.

Finally, the dot product result can get as follow:

$$d = \mathbf{n} \cdot \mathbf{L} \quad (3)$$

According to the definition of dot product ($\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta$), we know that if $d > 0$, the cell center is outside the object. If $d = 0$, the cell center is on the object surface. If $d < 0$, the cell center is inside the object.

In this method, the intersecting triangle returned by the AABB method are used to avoid the traversal operation of the ray casting method. The total computational requirement are only 9 multiplications and 9 subtractions for one cell. The complexity of this method is $O(1)$ for each interesting cell. Fig. 6 shows the grid generation result for a level 2 Cartesian grid tagged by the dot product method.

Figure 5 Illustration of the dot product method.

Figure 6 A level 2 Cartesian grid tagged by the dot product method.

5.2 Painting algorithm

The principle of painting algorithm is shown as Fig. 7. The initial background grid is divided into two categories after the intersecting determination: non-intersecting cells (white and green cells in Fig. 7) and interesting cells (golden cells in Fig. 7). Firstly, the painting algorithm needs to find a non-intersecting outside cell as the painting

157 source. Here the ray casting method is used for querying the non-intersecting cells.
 158 Once a outside cell occurs, stop the query. Then the painting process started from
 159 this cell. Examine the neighbors of this cell, if the neighbor cell is not intersected,
 160 mark the neighbor cell as an outside cell, then check the neighbor cell's neighbors. If
 161 the neighbor cell is interested, return. By recurring the steps above, all outside cells
 162 in the computational domain will be found out and marked. By choosing a most
 163 possible cell to start the painting source querying in the implementation process of
 164 the painting algorithm, the determination of ray casting method can be reduced to
 165 nearly once in most cases.

Algorithm 1 Painting algorithm

```

1. for a cell of painting source  $cell_i$  {
2.   if ( $cell_i \% position == non-interesting$ ) then {
       $cell_i \% position = outside$ 
3.   for each neighbor of  $cell_i$  {
       $cell_i \Rightarrow cell_i \% neighbor$ 
      go to step 1. (recur the painting algorithm)
    }
  }
}
```

Figure 7 Illustration of the painting algorithm.

166 5.3 Data structure of object surface

167 According to the algorithms complexity analysis in Sect.4, the number of object
 168 surface elements will greatly affect the efficiency of mesh generation. A sufficiently
 169 detailed model generally has a large number of surface elements. The scattered
 170 points (see Fig. 8(a)) are arranged in a disorderly way, which has a low efficiency for
 171 directly application. The alternating digital tree (ADT) structure [27] realizes the
 172 orderly storage of all elements by storing the scattered points in an ordered binary-
 173 tree according to the spatial relationship. K-dimensional tree (KDT) structure [28]
 174 is an improved form of ADT, which solves the problem of unbalanced binary-tree
 175 of ADT. By using the KDT structure, the query efficiency of the surface elements
 176 can be stabilized at the optimal solution [29]. In this paper, a quick KDT building
 177 method [30, 31] is used to divide the triangle elements of the object according to the
 178 median point in each dimension (Fig. 8(b)). Finally, a balanced binary-tree with
 179 the depth of $\log_2 N_{tri} + 1$ is obtained for the object with N_{tri} triangle elements
 180 (Fig. 8(c)).

181 In the intersecting determination process, the AABB method do determinations
 182 for each KDT node. Once intersected, the intersecting process terminates. If does
 183 not intersected, only one branch of the KDT will entered according to the division
 184 position of this KDT node. By applying the KDT structure, even in the worst
 185 situation, only $\log_2 N_{tri} + 1$ times of AABB determinations are needed to find out the
 186 intersecting triangle. Also for the ray casting method, disjoint branch of the KDT
 187 can be quickly eliminated by the bounding box [32] generated in the KDT division
 188 process. In general, after applying the KDT structure, the algorithm complexity is
 189 simplified from $O(N_{cells} \cdot N_{tri})$ to $O(N_{cells} \cdot \log N_{tri})$.

6 Applications

6.1 Accelerating Performance

192 In order to investigate the grid generation efficiency of the accelerating method
 193 used in this paper, we test the time consuming on generating Cartesian grids using
 194 different methods. As shown in Fig. 9, the test object is a sphere with the diameter
 195 of 100. The computational domain is set to a $300 \times 300 \times 300$ cube, the scale of
 196 initial grid is $15 \times 15 \times 15$. All scales above are dimensionless. In order to ensure
 197 the consistency of the test results, all computations are performed as a single-core
 198 test on the laptop with *Inter(R) Core(TM) i7-10875H CPU 2.3GHz*. The program
 199 is compiled by *ifortran -O3* option and runs under the Windows system. For each
 200 case, 10 times of tests are carried out, and the averaged running time of the program
 201 are exhibited.

202 Fig. 10 gives the grid generation efficiency compared by different methods.
 203 AABB+KDT is the original method, AABB+KDT+DP is the original method
 204 optimized by the dot product method, and the AABB+KDT+DP+PA method is
 205 the finial refinement method proposed in this paper. Due to the inconsistency of
 206 the number of Cartesian cells under different cases, the ordinate in Fig. 10 is set to
 207 the total grid generation time T_{total} divided by the number of cells N_{cells} , and the
 208 abscissa is set in logarithmic.

209 Fig. 10(a) gives out the grid generation efficiency under different refinement levels
 210 for a sphere consisted by 1016 triangles. With the increasing of refinement levels,
 211 N_{cells} increases exponentially, while the T_{total}/N_{cells} of the three methods are all

212 increasing approximately exponential ratio (linearly in the logarithmic coordinates).
 213 Among them, the AABB+KDT+DP+PA method have a 50% efficiency increasing
 214 compared with the original AABB+KDT method. Fig. 10(b) gives out the grid
 215 generation efficiency for different numbers of triangles (N_{tri}) of the sphere, and the
 216 refinement level is set as 5. With the increasing of the surface elements of object, the
 217 original method exhibits as an approximately exponential growth in the logarithmic
 218 coordinate. While the AABB+KDT+DP+PA method grows inconspicuous. This
 219 means the new method applied in this paper can achieve at least 50% optimization
 220 when applied a simple model. And for a complex model, much higher optimization
 221 effect will realized.

6.2 Examination

223 In order to examine the robustness and the stability of the grid generation method
 224 developed in this paper, different geometrics including a simple sphere, a medium-
 225 complex bogie, and a complex pantographs are used for the test, as shown in Fig. 11.
 226 The bogie and the pantograph are constructed by many slits and component with

large slenderness ratio. All other settings and the environment are consistent with Sect.6.1.

Fig. 12 shows the Cartesian grid generated by the new method applied in this paper. It proved that the new method can be applied in complex geometrics with good robustness and fast grid generation speed. Table 1 shows the result of examination. All three cases are used a 7-level geometric adaptive refinement. The T_1 is the intersecting determination time and the T_2 is the inside/outside determination time in table 1. It can be seen that the T_1 increases slightly with the increasing of N_{cells} , while the T_2 almost unchanged. The total efficiency will decreases slightly with the increasing of N_{tri} . Compared with the original method, the speed-up ratio ($T_{AABB+KDT}/T_{AABB+KDT+DP+PA}$) increases significantly with the increase of N_{tri} . Also the grid generation time in table 1 has compared with various literatures [19, 33, 34]. Although the CPU used in this paper is more advanced, it still has great advantages compared with other literatures. Here the $T_{total}=43.6$ sec. of the automobile in [33] is the cube generation time 6.21 sec. multiplied by the parallel efficiency 7.02.

Table 1 Performance test

Geometry	N_{tri}	N_{cells} ($\times 10^6$)	T_1 (sec.)	T_2 (sec.)	T_{total} (sec.)	T_{total}/N_{cells} (sec.)	speed-up ratio
Sphere	1,016	35.0	15.8	3.8	60.0	1.71×10^{-6}	1.78
Bogie	82,744	26.0	15.5	3.4	50.7	1.94×10^{-6}	4.77
Pantograph	2,254,812	26.5	22.3	3.6	57.6	2.17×10^{-6}	25.41
Launcher [19]	113,772	1.64	/	/	150.0	9.15×10^{-5}	/
Automobile [33] (cube)	744,404	0.037	/	/	43.6	1.17×10^{-3}	/
Sphere [34]	9,096	0.17	/	/	24.9	1.47×10^{-4}	/

7 Conclusions and future work

In the present paper, the dot product method, the painting algorithm, and the k-dimensional tree is employed to accelerate the process of Cartesian grid generation. The optimized program uses the octree to store Cartesian cells and the k-dimensional tree to store surface elements of objects. Through the test, the algorithm developed in this paper can greatly reduce the time consuming of grid generation under different situations. Finally, the robust and efficient grid generation results are obtained through three geometric configurations of different complexity. It is proved that the proposed methods in this paper can suitable for the complex models in engineering.

Ongoing and future efforts will concentrate on several topics. For the present research, the focus of optimization is on the cell type determination. However, the neighbor query of cells takes a lot of time during the grid smoothing process. The following work will consider to use some new data structure instead the octree to store Cartesian cells to optimize the neighbor founding process and the memory consumption. A further subject of great interest is the adaptability of dirty geometry. Here a central question is to find a robust method to handling the multi-body intersecting and slit surface problems.

Abbreviations

AABB: Axis-aligned bounding box; ADT: Alternating digital tree; DP: Dot product; KDT: K-dimensional tree; PA: Painting algorithm.

Acknowledgements

The authors would like to thank Dr. Peng Ji, Ms. Zhe Wang, Ms. Fang Hu, and Mr. Snowdog Lee for their helpful work. The authors are thankful to the reviewers for their valuable comments to improve the quality of the manuscript.

Authors' contributions

The research output comes from joint effort. All authors read and approved the final manuscript.

Funding

This work is supported by the National Natural Science Foundation of China (Grant number 51905547 & 52078199) and the National Numerical Wind-Tunnel Project (NNW2019ZT5-A11).

Availability of data and materials

The datasets used during the current study are available from the corresponding author on reasonable request.

Competing interests

The authors declare that they have no competing interests.

Author details

¹ Stat Key Laboratory of Aerodynamics, China Aerodynamics Research and Development Center, Mianyang, China.

² Computational Aerodynamics Institute, China Aerodynamics Research and Development Center, Mianyang, China.

³ Key Laboratory of Traffic Safety on Track, Ministry of Education, School of Traffic & Transportation Engineering, Central South University, Changsha, China. ⁴ Joint International Research Laboratory of Key Technologies for Rail Traffic Safety, Changsha, China.

References

1. Park S, Jeong B, Lee JG, Shin H (2013) Hybrid grid generation for viscous flow analysis. *International Journal for Numerical Methods in Fluids* 71(7):891–909.
2. Work D, Yanagita Y, Giblette T, Katz AJ, Hunsaker DF (2017) Validation of flux correction on three-dimensional strand grids with an overset cartesian grid. In: 55th AIAA Aerospace Sciences Meeting.
3. Zhang LP, Chang XH, Duan XP, Wang ZY, Zhang HX (2009) A block lu-sgs implicit unsteady incompressible flow solver on hybrid dynamic grids for 2d external bio-fluid simulations. *Computers & Fluids* 38(2):290–308.
4. Zhang X, Lai KL (2012) Simulation of transonic aeroservoelasticity using cartesian-grid based flow solver. In: 42nd AIAA Fluid Dynamics Conference and Exhibit, p 3266
5. Zhang L, Zhao Z, Ma R, Chang X, Li Y, Wang N, Li H (2020) Validation of numerical virtual flight system with wind-tunnel virtual flight testing. *AIAA Journal* 58(4):1566–1579.
6. Melton J, Enomoto F, Berger M (1993) 3d automatic cartesian grid generation for euler flows. In: 11th Computational Fluid Dynamics Conference.
7. Zeewu DD, Powell KG (1993) An adaptively refined cartesian mesh solver for the euler equations. *Journal of Computational Physics* 104:56–68

- 298 8. Aftosmis MJ (1997) Solution adaptive cartesian grid methods for aerodynamic flows with complex geometries.
299 In: 28th Computational Fluid Dynamics
- 300 9. Nakahashi K (2013) Aeronautical cfd in the age of petaflops-scale computing: From unstructured to cartesian
301 meshes. *European Journal of Mechanics - B/Fluids* 40:75–86,
- 302 10. Chaudhuri A, Hadjadj A, Chinnayya A (2011) On the use of immersed boundary methods for shock/obstacle
303 interactions. *Journal of Computational Physics* 230(5):1731–1748,
- 304 11. Yousefzadeh M, Battiato I (2019) High order ghost-cell immersed boundary method for generalized boundary
305 conditions. *International Journal of Heat and Mass Transfer* 137:585–598,
- 306 12. Sang W, Li F (2003) Omni-tree and adaptive cartesian hybrid grid method in steady and unsteady flows. In:
307 21st AIAA Applied Aerodynamics Conference, p 4078
- 308 13. Kim LS, Nakahashi K, Xu ZZ, Xiao H, Lyu SK (2015) Three-dimensional building-cube method for inviscid
309 compressible flow computations. *International Journal of Precision Engineering and Manufacturing*
310 16(13):2673–2681,
- 311 14. O'Rourke J (1998) *Computational Geometry in C*. Cambridge University Press
- 312 15. Chen H, Bi L, Hua R, Zhou Q, Tang Z, Yuan X (2021) An efficient cartesian mesh generation method based on
313 fully threaded tree data structure. *Acta Aeronautica et Astronautica Sinica* 42(12):125,170,
- 314 16. Meakin RL (2001) Object x-rays for cutting holes in composite overset structured meshes. In: 15th AIAA
315 Computational Fluid Dynamics Conference, p 2537
- 316 17. Melton JE, Berger MJ, Aftosmis MJ, Wong MD (1995) 3d applications of a cartesian grid euler method. In:
317 33rd Aerospace Sciences Meeting and Exhibit
- 318 18. Delanaye M, Aftosmis MJ, Berger MJ, Liu Y, Pulliam TH (1999) Automatic hybrid-cartesian grid generation
319 for high-reynolds number flows around complex geometries. In: 37th AIAA Aerospace Sciences Meeting and
320 Exhibit, vol 99
- 321 19. Capizzano F (2018) Automatic generation of locally refined cartesian meshes: Data management and
322 algorithms. *International Journal for Numerical Methods in Engineering* 113(5):789–813,
- 323 20. Coirier WJ, Powell KG (1996) Solution-adaptive cartesian cell approach for viscous and inviscid flows. *AIAA*
324 *Journal* 34(5):938–945,
- 325 21. Kamatsuchi T (2007) Turbulent flow simulation around complex geometries with cartesian grid method. In:
326 45th AIAA Aerospace Sciences Meeting and Exhibit, p 1459
- 327 22. Péron S, Benoit C (2011) Automatic off-body overset adaptive cartesian mesh method based on an octree
328 approach. In: 20th AIAA Computational Fluid Dynamics Conference, p 3050
- 329 23. Ham FE, Lien FS, Strong AB (2002) A cartesian grid method with transient anisotropic adaptation. *Journal of*
330 *Computational Physics* 179(2):469–494,
- 331 24. Ma T, Li P, Ma T (2020) A three-dimensional cartesian mesh generation algorithm based on the gpu parallel
332 ray casting method. *Applied Sciences* 10(1):58
- 333 25. Foley J, Dam VA, Feiner S, Hughes J (2013) *Computer graphics: principles and practices*, 3rd edn.
334 Addison-Wesley Professional
- 335 26. Möller T, Trumbore B (1997) Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*
336 2(1):21–28,
- 337 27. Bonet J, Peraire J (1991) An alternating digital tree (adt) algorithm for 3d geometric searching and
338 intersection problems. *International Journal for Numerical Methods in Engineering* 31(1):1–17
- 339 28. Bentley JL (1975) Multidimensional binary search trees used for associative searching. *Communications of the*
340 *ACM* 18(9):509–517,
- 341 29. Shevtsov M, Soupikov A, Kapustin A (2007) Highly parallel fast kd-tree construction for interactive ray tracing
342 of dynamic scenes. *Computer Graphics Forum* 26(3):395–404,
- 343 30. Popov S, Gunther J, Seidel HP, Slusallek P (2006) Experiences with streaming construction of sah kd-trees. In:
344 2006 IEEE Symposium on Interactive Ray Tracing, IEEE, pp 89–94
- 345 31. Wald I, Havran V (2006) On building fast kd-trees for ray tracing, and on doing that in $O(n \log n)$. In: 2006
346 IEEE Symposium on Interactive Ray Tracing, IEEE, pp 61–69
- 347 32. Meakin RL (2001) Automatic off-body grid generation for domains of arbitrary size. In: 15th AIAA
348 Computational Fluid Dynamics Conference, p 2536
- 349 33. Ishida T, Takahashi S, Nakahashi K (2008) Fast cartesian mesh generation for building-cube method using
350 multi-core pc. In: 46th AIAA Aerospace Sciences Meeting and Exhibit
- 351 34. Hu O (2013) Development of cartesian grid method for complex compressible flows and its applications. Thesis,
352 College of Aerospace Engineering, Nanjing University of Aeronautics and Astronautics