Cylindrical Acceleration Structures for Large Hexahedral Volume Visualization

Junpeng Wang* Mai Elshehaly† Yong Cao‡

Virginia Tech

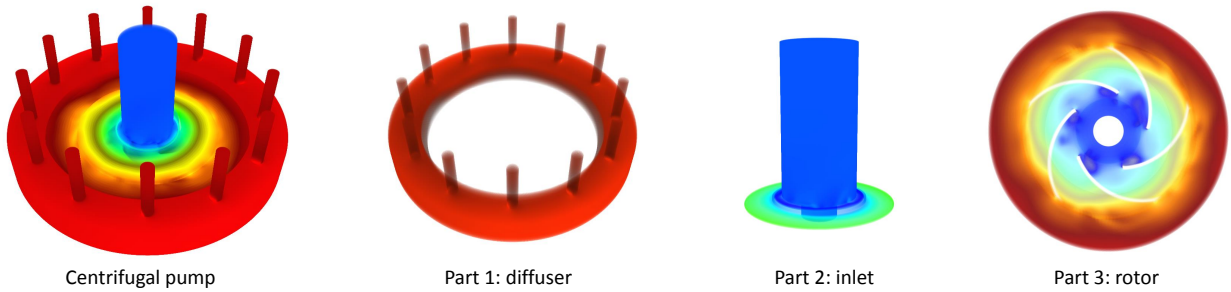

Figure 1: The experimental data set, i.e. a centrifugal pump; and its three separated parts: diffuser, inlet and rotor.

Abstract

Scientific simulations generate large irregular volumes with complex geometry. In volume rendering, the efficiency of the ray casting algorithms relies heavily on acceleration data structures to achieve accurate visual representations at interactive rates. In this paper, two acceleration structures are introduced to visualize a large irregular volume on the GPU: *(i) two-level cylindrical grid*, and *(ii) cylindrical bounding volume hierarchy (BVH)*. With thorough analysis and detailed experimental results, we demonstrate that the cylindrical data structures can effectively store the hexahedral volumes, reduce duplications, and increase ray casting performance.

1 Introduction

Scientific simulations tackle increasingly ambitious yet demanding problems that require modeling of complex 3D geometries. Challenges arise in creating accurate visual representations of the resulting structures while maintaining interactive rates. The efficiency of volume rendering techniques relies heavily on the careful design of acceleration data structures. However, the bulk of research in volumetric data visualization has primarily focused on datasets that are modeled as triangular meshes and/or rigid grid volumes. Such techniques fall short in addressing issues introduced by more complex primitive types, which are becoming more common in engineering simulations.

In this paper, we set the focus on complex models, organized as cells of irregular hexahedral volumes. We address the challenges they pose to traditional spatial indexing structures, and propose novel cylindrical structures that better serve the irregular nature of the cells, especially in applications where the data cells roughly follow a cylindrical contour. Namely, our contributions in this paper are: *(i) proposing two different cylindrical acceleration structures: two-level cylindrical grid and cylindrical BVH*; and *(ii) presenting a thorough comparison between traditional spatial indexing structures and their cylindrical counterparts*. To highlight the strengths of the proposed techniques for target application domains, we analyze results of rendering a large centrifugal pump data set (as shown in Figure 1), which is a cell-organized irregular hexahedral volume. The proposed techniques do not rely on any feature of the hexahedral cells, and thus can easily be applied to other irregular cell-organized data set, such as tetrahedral volumes. We conclude that the proposed cylindrical structures are especially effective in storing this type of irregular volumes with fewer duplications; and that they improve the irregular volume visualization performance on the GPU.

*e-mail: junpeng@vt.edu
†e-mail: maya70@vt.edu
‡e-mail: yongcao@vt.edu

2 Background and Related Work

In general, 3D volumes can be categorized into regular uniform grids or irregular volumes. The regular uniform grids are 3D volumes with samples evenly distributed in the space. Much research has been done to efficiently render these volumes [1] and GPUs have been used frequently to boost the rendering process [11]. Contrast to regular grid volume rendering, irregular volume visualization [14, 16, 19, 21] is challenging and no routine approach or hardware design is optimized for irregular volumes. The two most common forms of irregular volume data are: *(i) array-organized* and *(ii) cell-organized* data [4]. Cell-organized data sets usually use a tetrahedron or a hexahedron for a cell element. In this paper, we tackle the problem of irregular hexahedral volume visualization using a ray casting algorithm on the GPU.

A naïve ray traversal algorithm would test every primitive for intersection. In practice, to lower the time complexity, spatial indexing structures are used to accelerate the search and achieve sublinear performance. In general, these structures are either flat grids or hierarchical trees. Uniform grids were first introduced to the field of ray traversal in [3]. They were then improved in [8], and later optimized for GPU accelerations in [10, 18]. There are various derivations of the grid structure. For example, Kalojanov et al. [9] proposed the two-level grid to better serve unevenly distributed primitives. Guntury and Narayanan proposed the perspective grid [5] which attempts to increase ray coherence, and thus improve the ray tracing performance. Xie et al. [23] introduced the method of performing ray casting on spherical geodesic grids on

GPUs to deal with large scale geoscience data. This paper presents a two-level cylindrical grid. Different from the perspective grid [5] which transforms the structure to fit the ray space, we transform the grid to fit the geometry space. Grid-based structures are efficient, but they have the drawback of being less flexible in adapting to the data distribution. Hierarchical structures including BSP trees [7], kd-trees [2, 24], and BVHs [6, 12] provide more flexibility in space subdivision, while adapting cell size to primitive distribution. However, the hierarchical structure increases the cost of traversal for neighborhood lookup, which can slow down rendering performance. To address these issues, different splitting and termination criteria have been proposed [15]. The surface area heuristic (SAH) [13, 22] has been widely adopted to optimize both for efficient ray traversal.

3 Data Set

This paper presents a case study using a large irregular hexahedral volume, which was released in 2011 as the IEEE VIS contest data set [20]. The data set is the simulation of a centrifugal pump. We refer the interested reader to [17] for more information about the simulation. Several scalar and vector fields' values of the simulation are recorded in 80 frames and these frames share the same 3D geometry, which means although the scalar/vector values on each vertex are changing every frame, the topology and vertex locations are static in all 80 frames of the simulation. The volume consists of

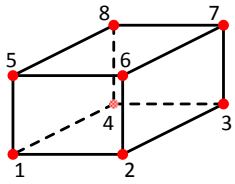

Figure 2: A hexahedron and its index order.

three parts (Figure 1): a *diffuser* domain (part 1), an *inlet* domain (part 2), and a *rotor* domain (part 3). Each part constitutes of a large number of hexahedral cells, the structure of which is shown in Figure 2. The number of vertices and hexahedra in each part is listed in Table 1.

Table 1: The number of vertices and hexahedra in each part.

	1. diffuser	2. inlet	3. rotor
Number of Vertices	3,551,244	892,086	2,247,800
Number of Hexahedra	3,444,324	856,492	2,147,855

We use the ray casting algorithm to visualize one scalar field of the volume. Rays are shot through the volume and samples are evenly taken along the ray. For each sample, it finds the hexahedron it is in. The contributions from the hexahedron's eight vertices to the sample are weighted according to their inverse distance. A transfer function then maps the sample's value to colors and the final pixel's color is derived by accumulating/blending color values of samples on the ray from front to back. The key problem during rendering is to efficiently locate which hexahedron a sample is in. This sampling problem is further complicated by the large number of hexahedra, significant variations in their sizes, and their uneven distribution in space, as shown in Figure 3. In some regions, like the blade area of part 3, hexahedra are very small and dense, whereas hexahedra in the boundary area are relatively large and sparse. In addition, the hexahedra are oriented to different directions.

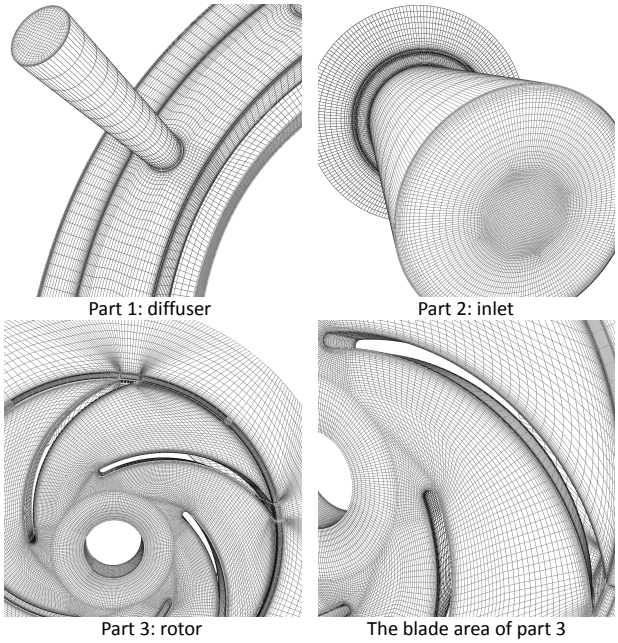

Figure 3: Primitives are unevenly distributed.

4 Two-Level Cylindrical Grid

Uniform grids [10] evenly subdivide the space and primitives are recorded into each grid cell. We observe that most hexahedra of the data set are distributed around the central line of the pump. In order to make the grid cell fit for the primitive distribution, we transform vertices into a cylindrical coordinate system and build a cylindrical grid on the data set. The three dimensions in the cylindrical coordinate system are *Radius* (R: distance to the longitudinal axis of the cylinder), *Angle* (θ: angle increases in counter-clockwise direction) and *Height* (H: the same as the Z dimension in the Cartesian coordinate system). The bounding volume of primitives in the new coordinate system is also changed (no longer a box), as shown in Figure 4.

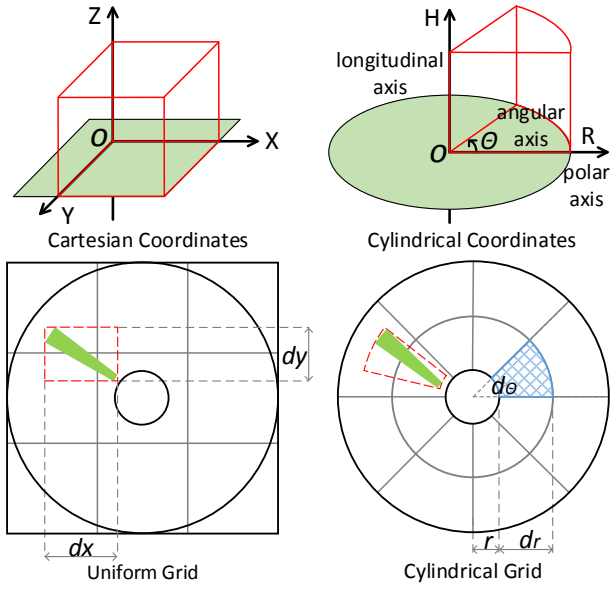

Figure 4: Uniform grid and cylindrical grid.

When constructing a uniform grid in the Cartesian coordinate system, the following equations are often used to derive the resolution (i.e. R_x, R_y and R_z) of the grid [10]:

$$R_x = d_x\sqrt[3]{\frac{\lambda N}{V}}, \quad R_y = d_y\sqrt[3]{\frac{\lambda N}{V}}, \quad R_z = d_z\sqrt[3]{\frac{\lambda N}{V}}. \qquad (1)$$

where N is the number of primitives in each cubic cell with edge lengths d_x, d_y and d_z; and V is the volume of the cube ($V = d_x d_y d_z$). In the cylindrical coordinate system, the volume of a grid cell (the blue region in Figure 4) can be calculated by $\frac{d_\theta(2r+d_r)}{2}d_r d_h$. So the following equations can be used to derive the resolution of a cylindrical grid:

$$R_r = d_r\sqrt[3]{\frac{\lambda N}{V}}, \quad R_\theta = \frac{d_\theta(2r+d_r)}{2}\sqrt[3]{\frac{\lambda N}{V}}, \quad R_h = d_h\sqrt[3]{\frac{\lambda N}{V}}. \qquad (2)$$

The two-level grid [9] has been proposed in the Cartesian coordinate system to address the "teapot in a stadium" problem. The first level of the grid is a coarse division of the space, while the resolution of the second level grid depends on the primitive distribution. To handle the unevenly distributed hexahedra, we derive the two-level cylindrical grid.

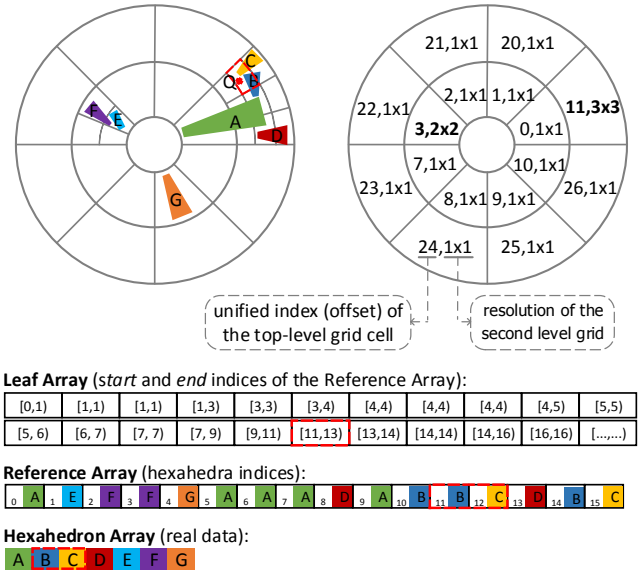

Figure 5: An example of the two-level cylindrical grid. The query point Q is in the grid cell with a unified index (offset) 16. Hexahedra recorded in this cell, i.e. hexahedron B and C, will be tested. Each element in the `Leaf Array` is a pair of indices representing the range of hexahedra indices in the corresponding grid cell. A cell is empty if the *start* and *end* indices are the same, such as [1, 1).

Figure 5 shows an example of using this structure to search hexahedra in a particular spatial region. The two-level cylindrical grid operates similarly to its Cartesian counterpart. Each top-level cell records the resolution of the corresponding second level and each grid cell, either from the first level or the second level, has a unified cell index. A query point Q first finds its top cell index by truncation using its cylindrical coordinate values and the top-level grid resolution. If the top-level cell is further subdivided into a low level grid, the algorithm performs a second truncation step for Q. Finally the cell offset can be found. For example, let us assume that Q truncates to cell offset 16, as outlined in red in Figure 5. Now searching the `Leaf Array` with this offset returns the *start* and *end* indices of the `Reference Array`, which are [11, 13). This indicates that the 11th and the 12th element (the 13th element is not included) of the `Reference Array` are the references of hexahedra in the 16th grid cell. Next, the program will test whether the query point Q falls in any of the hexahedra in the grid cell, i.e. hexahedron B and C in this example.

A major drawback with grids is the huge amount of primitive duplications, as shown in the `Reference Array` of Figure 5. These duplications are caused by two factors: *First*, when a hexahedron overlaps with two grid cells, its index has to be duplicated and recorded in both cells; *Second*, since the bounding volume of primitives is used as a proxy when constructing the structure, more false positive duplications are inevitable. In the cylindrical coordinate system, the grid uses bounding volumes that tightly enclose the primitives, thus reducing the number of duplications from the second part. There are several factors that decide the rendering performance, such as the transfer function, sampling distance and viewing direction. The best performance indicator is the number of hexahedra visited by all rays in a frame. Since the cylindrical grid has less duplications, the rendering program should visit less hexahedra during traversal, thus achieve better rendering performance. We will demonstrate the results supporting this solution in Section 7.

5 Cylindrical BVH

Grid structures suffer from a lack of flexibility in dividing the space, which results in cells with highly diverse numbers of primitives. Dense regions of the volume require high grid resolutions, whereas sparse regions do not. Two-level grids, to some extent, mitigate the problem. However, they are still not flexible enough to effectively minimize the number of primitives in each cell, as primitives can still be unevenly distributed in the second level. In Section 7.1, we demonstrate that the maximum number of primitives in some grid cells is still large even with a very high grid resolution.

On the other hand, hierarchical acceleration structures provide more flexibility in subdividing the space and their node sizes adapt to primitive distribution. Consequently, these structures can effectively reduce the number of primitives in leaf nodes. However, their hierarchical depth incurs a heavier traversal cost during rendering. BVHs create tightly enclosing volumes for model primitives and organize them in a hierarchy. In BVHs, space overlap is possible, but duplication is avoided. During traversal, backtracking is needed to accommodate for space overlap. We derive the cylindrical BVH, as shown in Figure 6. In our implementation, we use the centroids of primitives' bounding volumes to cluster the primitives. The surface area heuristic is used to choose the split plane. Since the bounding volume of primitives is no longer a box, surface area calculation is different. For Cartesian BVH, the surface area is: $2(d_x d_y + d_x d_z + d_y d_z)$, whereas for cylindrical BVH, the equation changes to: $(2r+d_r)(d_r+d_h)d_\theta + 2d_r d_h$.

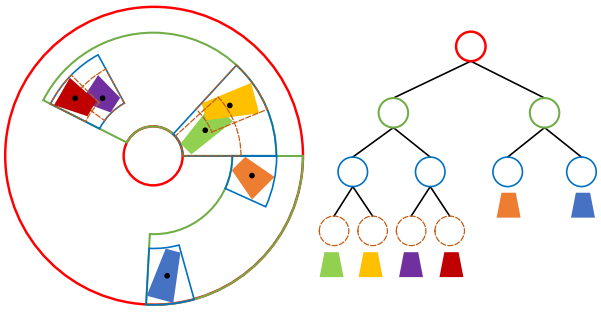

Figure 6: Cylindrical bounding volume hierarchy.

When traversing a BVH, visiting an internal node will decide which child to visit next; whereas visiting a leaf node will perform

intersection tests with primitives in the node. The cost of traversing is incurred by both hierarchical traversal steps, and the number of intersection tests with primitives in leaf nodes. We limit the number of primitives in leaf nodes to one to reduce the time spent on primitive intersection tests. Since the BVH does not generate duplications, the number of hexahedra is the same as the number of BVH leaf nodes. The cost of traversing BVHs can be quantified by the number of internal and leaf nodes visited in a frame. We will demonstrate these results of the cylindrical BVH in comparison with its Cartesian counterpart in Section 7.

6 Limitations

Some features of the cylindrical coordinate system pose challenges when constructing acceleration structures. In this section, we discuss two major limitations.

6.1 Minimum Radius

With the eight vertices of a hexahedron, one can easily construct the axis-aligned bounding box of a hexahedron in the Cartesian coordinate system. However the same is not true in the cylindrical coordinate system. To build the bounding volume for a hexahedron in the cylindrical system, one needs the values of r_{min}, r_{max}, θ_{min}, θ_{max}, h_{min} and h_{max}. The eight vertices of a hexahedron derive these values, except r_{min}. The minimum distance of a hexahedron to the longitudinal axis may be located on one edge or one face of the hexahedron, as shown in Figure 7 *A*. To find the value of r_{min}, we

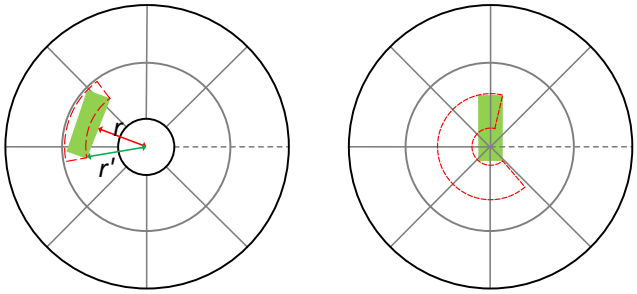

Figure 7: Incorrect minimum radius of cylindrical bounding volumes.

project the 12 edges of a hexahedron to the $R\theta$ plane (i.e. the XY plane in the Cartesian coordinate system). The longitudinal axis on this plane is a point. The shortest distance between this point and the 12 projected line segments is the required r_{min} when constructing the cylindrical bounding volume.

When primitives are on the longitudinal axis, r_{min} cannot be directly computed from the eight vertices of the hexahedron either. As shown in Figure 7 *B*, the algorithm constructs an incorrect bounding volume. This case can be detected by comparing the X and Y Cartesian coordinates of the eight vertices and these coordinates of the longitudinal axis. When building the cylindrical structures, we need to change the r_{min} of these primitives' bounding volumes to 0 and modify their θ range to $[0, 2\pi)$. This problem happens in part 2 (the *inlet* domain) of the volume, and we successfully managed this case with these modifications.

6.2 One Closed Dimension

In the Cartesian coordinate system, three dimensions are unbounded, i.e. coordinate values range from negative infinity to positive infinity. However, in the cylindrical coordinate system, one dimension, i.e. θ, is closed and periodical. One may find a hexahedron covers θ ranging from 0.1 to $2\pi - 0.1$. Primitives on the polar axis (Figure 8 *A* shows a 2D example, the axis will be a plane in 3D case) have to be handled carefully.

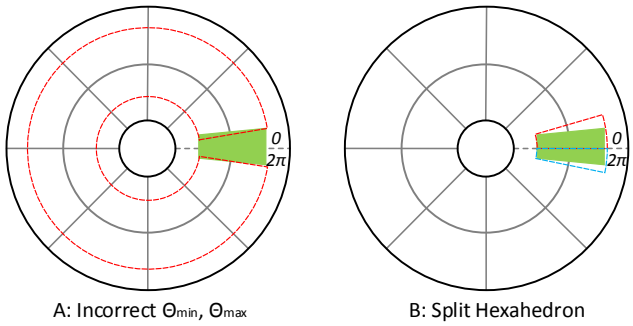

Figure 8: One closed dimension in the cylindrical coordinate system.

The solution for this case is to duplicate the hexahedra. For the experimental data set, primitives on the polar axis can be detected when $\theta_{max} - \theta_{min} > \pi$. For two-level cylindrical grid, these primitives will be put into cells on both sides of the polar axis. When working with cylindrical BVH, we create two separated bounding volumes for each of these primitives. In other words, the primitives on the polar axis will be split by the axis (Figure 8 *B*). As a result, some leaf nodes of the BVH will have different bounding volumes, but the same primitive index.

7 Results and Analysis

The results in this section are collected on an NVIDIA GeForce GTX TITAN GPU with 6 GB device memory. The irregular volumes are rendered with the ray casting algorithm at a resolution of 768×768. The 80 frames of the volume data share the same geometry structure. So all acceleration structures discussed in this paper can be built offline. Two important factors for evaluation are the memory used for these structures and the rendering performance.

7.1 Two-Level Cylindrical Grid

Similar to [9], we set the λ value for the top-level of the two-level cylindrical grid to $\frac{1}{16}$ ($\lambda_1 = \frac{1}{16}$). Table 2 shows the results when the λ value for the second level is 4 ($\lambda_2 = 4$). With detailed comparisons to the two-level uniform grid in the Cartesian coordinate system, we draw the following conclusions:

1. Two-level grids are not flexible enough to reduce the number of primitives in a grid cell, as there are still cells containing more than a hundred hexahedra in Table 2. The maximum number of hexahedra in cylindrical grid cells is smaller than that in Cartesian grid cells (the row labeled *"Hexahedra Per Cell"*).
2. The equations used to derive the resolution of cylindrical grids have similar performance as the equations for uniform grids, as the numbers of grid cells in both structures are similar when using the same λ_1 and λ_2 values (the row labeled *"Number of Grid Cells"*).
3. Two-level cylindrical grid effectively reduces the amount of primitives duplications and uses 69%∼82% memory space of its Cartesian counterpart (the row labeled *"Number of Hexahedra in Total"* and *"Memory Usage"*).
4. The number of hexahedra visited by all rays in a frame is less when using the two-level cylindrical grid and the rendering program achieves 9%∼36% speedups (the row labeled *"Hexahedra Visited in Total"* and *"Frame Per Second"*).

The effect of different λ_2 values has been demonstrated in Figure 9. A larger λ_2 value leads to a finer grid resolution, as well as a larger amount of duplications. The finer resolution of the grid structures helps to improve the rendering performance. As shown

Table 2: The two-level cylindrical grid performs better than the two-level Cartesian grid in terms of memory usage and rendering performance ($\lambda_1 = \frac{1}{16}$, $\lambda_2 = 4$). Ratio=row(Cylindrical)/row(Cartesian).

		1: diffuser	2: inlet	3: rotor
Top Grid Resolution	*Cartesian*	88×88×28	36×36×42	113×113×11
	Cylindrical	17×323×38	19×61×45	54×218×12
Hexahedra Per Cell	*Cartesian*	0∼154	0∼108	0∼165
	Cylindrical	0∼115	0∼60	0∼126
Number of Grid Cells	*Cartesian*	28,248,118	6,467,778	23,858,462
	Cylindrical	28,190,022	6,938,619	22,535,140
	Ratio	1.00	1.07	0.94
Number of Hexahedra in Total	*Cartesian*	234,495,582	41,774,805	229,291,796
	Cylindrical	143,348,566	30,853,798	147,278,914
	Ratio	0.61	0.74	0.64
Memory Usage (in MB)	*Cartesian*	1139.2	214.3	1083.7
	Cylindrical	782.6	175.3	753.0
	Ratio	0.69	0.82	0.69
Hexahedra Visited in Total	*Cartesian*	16,624,898	21,469,156	15,108,410
	Cylindrical	9,743,971	12,620,823	13,512,711
	Ratio	0.59	0.59	0.89
Frame Per Second	*Cartesian*	12.80	14.86	18.61
	Cylindrical	17.39	17.96	20.24
	Ratio	1.36	1.21	1.09

in the figure, both the memory usage and the rendering performance increase as the λ_2 value gets larger. The two-level cylindrical grid always takes less memory space and achieves better rendering performance than its Cartesian counterpart.

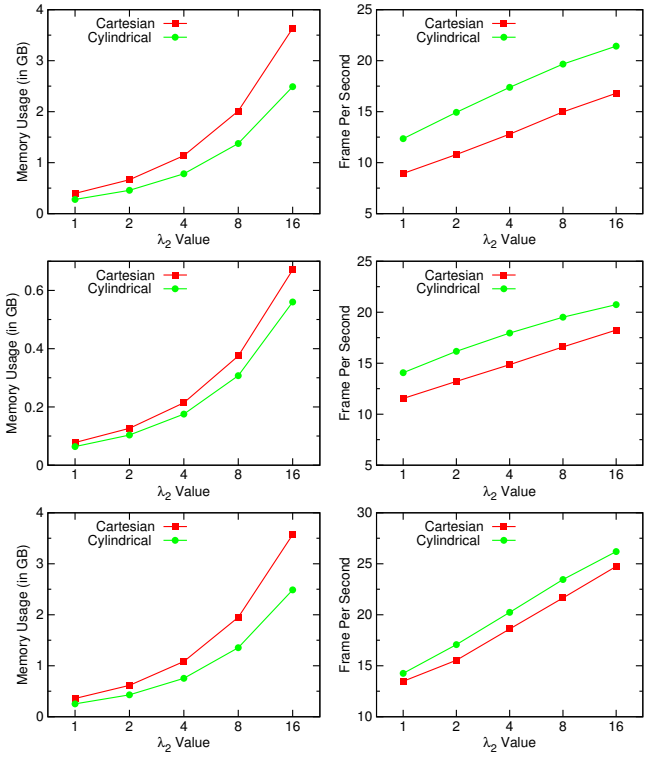

Figure 9: Effects of different λ_2 values. Row 1 to row 3 are the results of: part 1 diffuser; part 2 inlet; and part 3 rotor.

7.2 Cylindrical BVH

As shown in Table 3, the cylindrical and Cartesian BVH have similar tree levels in all three parts of the volume. Since the Cartesian BVH does not generate any duplication and the termination criteria is each node contains one hexahedron, the number of leaf nodes is the same as the number of hexahedra in total. The cylindrical BVH has a few duplications due to the limitation we discussed in Section 6 (Figure 8). As a result, the number of BVH nodes and the the number of hexahedra are a little larger than those of the Cartesian BVH. Since the number of duplications is small, there is no noticeable difference in the memory usage.

Table 3: Results of the Cartesian and cylindrical BVH. Ratio=row(Cylindrical)/row(Cartesian).

		1: diffuser	2: inlet	3: rotor
Tree Levels	*Cartesian*	31	27	32
	Cylindrical	31	27	31
Number of BVH Nodes	*Cartesian*	6,888,647	1,712,983	4,295,709
	Cylindrical	6,907,577	1,725,155	4,312,835
	Ratio	1.00	1.01	1.00
Number of Hexahedra in Total	*Cartesian*	3,444,324	856,492	2,147,855
	Cylindrical	3,453,789	862,578	2,156,418
	Ratio	1.00	1.01	1.00
Memory Usage (in MB)	*Cartesian*	309.5	76.9	193.0
	Cylindrical	310.5	77.5	193.7
	Ratio	1.00	1.00	1.00
Surface Area	*Hexahedra*	42.66	9.78	19.25
	Cartesian	87.13	14.21	41.07
	Cylindrical	49.10	10.72	33.01
	Ratio	0.56	0.75	0.80
Internal Nodes Visited	*Cartesian*	229,709,037	215,910,139	99,384,906
	Cylindrical	125,911,725	99,607,091	87,071,167
	Ratio	0.55	0.46	0.88
Leaf Nodes Visited	*Cartesian*	5,702,405	5,830,747	5,981,982
	Cylindrical	3,638,955	3,967,473	5,148,690
	Ratio	0.64	0.68	0.86
Frame Per Second	*Cartesian*	7.55	9.50	12.16
	Cylindrical	11.37	16.53	13.92
	Ratio	1.51	1.74	1.14

Although the cylindrical BVH generates a few duplications, its rendering performance is better than the performance of the Cartesian BVH. The row of *"Surface Area"* in Table 3 shows the total surface area of: *(i) all hexahedra*; *(ii) all Cartesian bounding boxes*; and *(iii) all cylindrical bounding volumes*. When calculating the surface area of cylindrical bounding volumes, we have taken the duplications (due to the limitation shown in Figure 8) into consideration. Although the cylindrical BVH has more hexahedra, the surface area of all cylindrical volumes is less than (56%∼80% of) the total surface area from all Cartesian boxes. The results indicate that the cylindrical bounding volumes more tightly enclose the hexahedra. Therefore, a smaller number of internal and leaf nodes are visited. For example, in part 1 (the *diffuser* domain), numerous samples in the central region of the volume are quickly eliminated with the condition $r < r_{min}$ and the rendering performance of the cylindrical BVH is 1.51 times faster than the Cartesian BVH.

7.3 Sampling Distance

The ray casting algorithm used in this paper does not traverse volumes cell by cell. Instead, it evenly takes samples on cast rays and accumulates samples' contributions from front to back to derive the

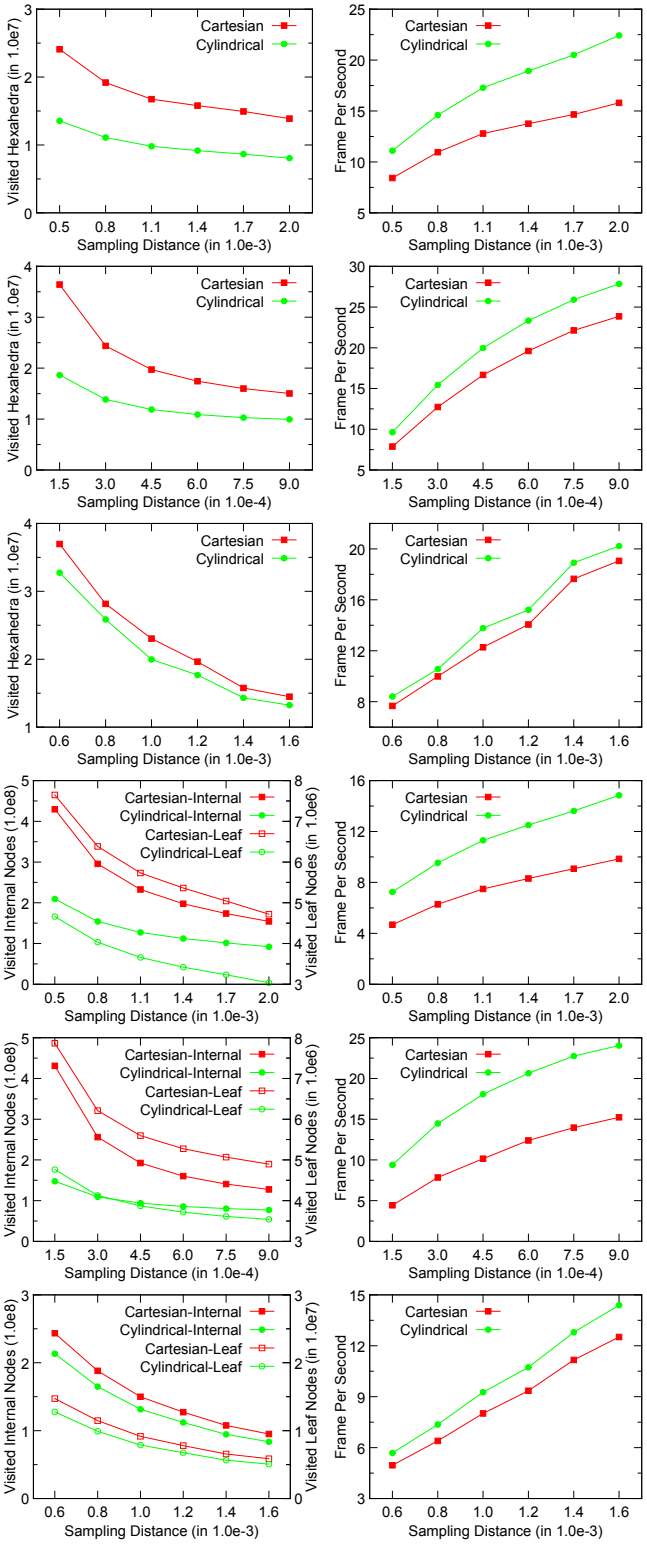

Figure 10: Effects of different sampling distances. Row 1 to row 3 are the results of: part 1 diffuser; part 2 inlet; and part 3 rotor using two-level Cartesian/cylindrical grids. Row 4 to row 6 are the results of part 1; part 2; and part 3 using Cartesian/cylindrical BVHs. The cylindrical structures always visit fewer hexahedra/nodes and achieve higher frame rates than their Cartesian counterparts.

final pixels' color. Rays will be terminated when the corresponding pixels are saturated. The distance between samples taken along rays has a crucial impact on the rendering performance, as it decides which grid cell or BVH node a ray will visit. The sampling distances used for part 1, part 2 and part 3 are $1.12e{-}3$, $3.8e{-}4$ and $1.5e{-}3$ when collecting the results in Table 2 and Table 3. To maintain the rendered image quality (under a fixed transfer function), the sampling distance should be bounded within a certain range. Figure 10 gives more performance sets of the cylindrical structures and their Cartesian counterparts when rendering with varying sampling distances. In general, when the sampling distance increases, the number of visited hexahedra (in two-level grids) and the number of visited internal/leaf nodes (in BVHs) decrease. The rendering performance improves with these numbers' declines.

The top three rows (row 1 to row 3 are results of part 1, part 2 and part 3 when rendering with the two-level grids) of Figure 10 indicate that the two-level cylindrical grid always visits fewer hexahedra in a frame. Therefore, it is always faster than the two-level Cartesian grid. Compared to the Cartesian BVH, the bottom three rows (performance of part 1, part 2 and part 3 when using BVHs) of Figure 10 denote that the cylindrical BVH always traverses less internal and leaf nodes and it always performs better regardless of the sampling distance. The results shown in this figure are consistent with the results we have demonstrated in Table 2 and Table 3.

8 Conclusion

This paper tackles the problem of visualizing large irregular hexahedral volumes on the GPU by adapting existing spatial indexing structures to the cylindrical coordinate system. Specifically, our discussion focuses on two-level cylindrical grid and cylindrical BVH, as they can handle unevenly distributed primitives in space. Two-level cylindrical grid outperforms the traditional two-level grid in effectively storing primitives and reducing duplications. The performance increases, because the number of visited primitives is reduced. Compared to the traditional BVH, the cylindrical BVH uses bounding volumes that tightly enclose primitives. Therefore, fewer internal and leaf nodes of the BVH are visited and higher rendering frame rates are achieved.

For different data sets, customized acceleration structures should be considered to achieve the best performance. Although the Cartesian coordinate system is used extensively, it may not be the best choice for all data sets with varying features. We hope that the results we were able to achieve would inspire more research and better customizations of acceleration structures to meet the increasingly complex requirements of the data.

9 Acknowledgments

The data is courtesy of the Institute of Applied Mechanics, Clausthal University, Germany (Dipl. Wirtsch.-Ing. Andreas Lucius).

References

[1] K. Engel, M. Hadwiger, J. M. Kniss, A. E. Lefohn, C. R. Salama, and D. Weiskopf. Real-time volume graphics. In *Proceedings of the conference on SIGGRAPH 2004 course notes - GRAPH '04*, pages 29–es, New York, New York, USA, 2004. ACM Press.

[2] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[3] A. Fujimoto, T. Tanaka, and K. Iwata. Arts: Accelerated ray-tracing system. *Computer Graphics and Applications, IEEE*, 6(4):16–26, 1986.

[4] M. P. Garrity. Raytracing irregular volume data. *ACM SIGGRAPH Computer Graphics*, 24(5):35–40, 1990.

[5] S. Guntury and P. Narayanan. Raytracing dynamic scenes on the gpu using grids. *Visualization and Computer Graphics, IEEE Transactions on*, 18(1):5–16, 2012.

[6] H. J. Haverkort. Introduction to bounding volume hierarchies. *Part of the PhD thesis, Utrecht University*, 2004.

[7] V. Havran and J. Zara. Evaluation of bsp properties for ray-tracing. In *Proceedings of Spring Conference On Computer Graphics*, pages 155–162, 1997.

[8] P.-K. Hsiung and R. Thibadeau. Accelerating arts. *The Visual Computer*, 8(3):181–190, 1992.

[9] J. Kalojanov, M. Billeter, and P. Slusallek. Two-level grids for ray tracing on gpus. In *Computer Graphics Forum*, volume 30, pages 307–314. Wiley Online Library, 2011.

[10] J. Kalojanov and P. Slusallek. A parallel algorithm for construction of uniform grids. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 23–28, NY, USA, 2009. ACM.

[11] J. Kruger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 38. IEEE Computer Society, 2003.

[12] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast bvh construction on gpus. In *Computer Graphics Forum*, volume 28, pages 375–384. Wiley Online Library, 2009.

[13] J. D. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6(3):153–166, 1990.

[14] G. Marmitt and P. Slusallek. Fast ray traversal of tetrahedral and hexahedral meshes for direct volume rendering. In *Proceedings of the Eighth Joint Eurographics/IEEE VGTC conference on Visualization*, pages 235–242. Eurographics Association, 2006.

[15] D. M. Mount. Ann programming manual. Technical report, Technical report, Dept. of Computer Science, U. of Maryland, 1998.

[16] P. Muigg, M. Hadwiger, H. Doleisch, and E. Gröller. Interactive volume visualization of general polyhedral grids. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2115–2124, 2011.

[17] M. Otto, A. Kuhn, W. Engelke, and H. Theisel. 2011 ieee visualization contest winner: Visualizing unsteady vortical behavior of a centrifugal pump. *Computer Graphics and Applications, IEEE*, 32(5):12–19, 2012.

[18] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 703–712. ACM, 2002.

[19] C. T. Silva, J. L. D. Comba, S. P. Callahan, and F. F. Bernardon. A survey of gpu-based volume rendering of unstructured grids. *Revista de informática teórica e aplicada. Porto Alegre, RS. Vol. 12, n. 2 (out. 2005), p. 9-29*, 2005.

[20] VIS. http://sciviscontest.ieeevis.org/2011/. *IEEE Visualization Contest 2011*, 2011.

[21] I. Wald, H. Friedrich, A. Knoll, and C. D. Hansen. Interactive isosurface ray tracing of time-varying tetrahedral volumes. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1727–1734, 2007.

[22] I. Wald and V. Havran. On building fast kd-trees for ray tracing, and on doing that in o(n log n). In *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 61–69, Sept 2006.

[23] J. Xie, H. Yu, and K.-L. Ma. Interactive ray casting of geodesic grids. In *Computer Graphics Forum*, volume 32, pages 481–490. Wiley Online Library, 2013.

[24] K. Zhou, Q. Hou, R. Wang, and B. Guo. Real-time kd-tree construction on graphics hardware. In *ACM Transactions on Graphics (TOG)*, volume 27, page 126. ACM, 2008.