PERGAMON

Computers & Graphics 27 (2003) 387–406

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Technical section

Direct volume rendering of unstructured grids

Hakan Berk, Cevdet Aykanat, Uğur Güdükbay*

Department of Computer Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey

Abstract

This paper investigates three categories of algorithms for direct volume rendering of unstructured grids, which are image-space, object-space, and hybrid methods. We propose three new algorithms. Cell Projection algorithm, which falls into object-space category, is capable of rendering non-convex meshes through a simple yet efficient sorting schema that exploits both image and object space coherencies. Existing hybrid methods use object-then-image traversal order that enforces the processing of each cell. Thus, these algorithms perform redundant operations and do not support early ray termination. We propose a hybrid method, called Span-Buffer Ray Casting (SBRC), that can support early ray termination discarding redundant operations by employing image-then-object traversal order. Another hybrid method, called Koyamada-SBRC (K-SBRC), is proposed with the motivation of refining image-space and hybrid methods to extract the best features of them. This method is developed by blending SBRC approach with Koyamada's algorithm, which is an efficient image-space algorithm. All proposed algorithms are capable of handling acyclic non-convex meshes and generating images of acceptable quality. SBRC and K-SBRC algorithms have the additional capabilities of rendering cyclic meshes and supporting early ray termination. The proposed algorithms and Koyamada's algorithm are implemented and experimented in a common framework for analyzing their relative performance.

Keywords: Unstructured grids; Direct volume rendering; Cyclic meshes; Early ray termination; Image-space coherency; Object-space coherency

1. Introduction

The vast amount of data produced by scientific and engineering simulations makes it very difficult for scientists to extract useful information from the data, and interpret it to reach a useful conclusion. Visualization of such numerical data as an image, which is named as *scientific visualization*, is an indispensable tool for researchers. *Volume rendering* is a very important branch of scientific visualization and makes it possible for scientists to visualize three-dimensional (3D) volumetric datasets.

Volumetric data used in volume rendering is in the form of a grid superimposed on a volume. The nodes of this grid contain the scalar values that represent the simulation results. Type of the grid also defines spatial characteristics of the volumetric dataset, which is important in the rendering process. Grids are classified into two categories: *structured* and *unstructured* [1–4]. Structured grids are topologically equivalent to the integer lattice, and as such, they can easily be represented by a 3D array. The mapping from the array elements to sample points and the connectivity relation between cells are implicit. On the other hand, the distribution of sample points do not follow a regular pattern in unstructured grids and there may be voids in the grid. Unstructured grids are also called *cell-oriented* grids because these grids are represented by a list of cells in which each cell contains pointers to the sample points in the cell. Due to the cell-oriented nature and the irregularity of unstructured grids, the connectivity information is provided explicitly. With recent advances in generating high-quality

*Corresponding author.

E-mail addresses: hakanb@microsoft.com (H. Berk), aykanat@cs.bilkent.edu.tr (C. Aykanat), gudukbay@cs.bilkent.edu.tr (U. Güdükbay).

0097-8493/03/$ - see front matter
doi:10.1016/S0097-8493(03)00034-7

adaptive meshes, unstructured grids are becoming increasingly popular in scientific and engineering simulations.

There are two major categories of volume rendering methods: *indirect* and *direct* methods. Indirect methods extract an intermediate geometric representation of the surfaces from the volume data and render those surfaces via conventional surface rendering methods. Direct methods render the data without generating an intermediate representation. Indirect methods are potentially faster and are more suitable for medical imaging and biological applications where the visualization of the surfaces in a volume makes sense. Since direct methods do not rely on surface extraction, they are more general and flexible. Direct methods are used when the inside of a material, such as a partially transparent fluid, should be visualized.

Direct volume rendering methods consist of two main phases: *resampling* and *composition*. Generally, these phases are handled in a highly interleaved manner. In resampling phase, new samples are interpolated by using the original sample points. The rendering method should locate the new sample point in the cell domain. This is because the vertices of the cell that contains the new sample being generated will be used in the interpolation. This problem is known as *point location* problem. In composition phase, generated samples are mapped to color and opacity values and these values are composited to determine the contribution of the data on a pixel. The composition operation is associative, but not commutative. Therefore, the color and opacity values should be composited in visibility order. The determination of the correct composition order is known as *view sort* problem. These two problems are easier to solve in structured grids, but the way that a volume rendering algorithm handles them is a crucial issue that strongly affects the performance of the rendering process for unstructured grids. The lack of implicit connectivity between cells and the irregularity of the distribution of the sample points in unstructured grids are the major factors that cause the difficulty.

Interactive visualization is very important since it enables scientists to change the simulation parameters so that the simulation is steered in the correct direction. The slowness of direct volume rendering of unstructured grids creates the lack of interactivity that prevents its wide use. One way to speed up the visualization process is to employ special graphics hardware. Developing parallel algorithms is another possibility. However, the need for a software solution for fast direct volume rendering of unstructured grids will always exist. The main concern of this work is to find efficient software solutions for direct volume rendering of unstructured grids without compromising the image quality.

1.1. Related work

Existing direct volume rendering algorithms for unstructured grids are classified into three categories; *image-space*, *object-space* and *hybrid* [5].

1.1.1. Image-space methods

In image-space methods, which are also called *ray-casting* methods, image-space is traversed to cast a ray for each pixel and each ray is followed, sampled and composited along the volume. For non-convex datasets, the rays may enter and exit the volume more than once. The parts of the ray that lie inside the volume, which in fact determine the contributions of data to the pixel color, are referred to here as *ray-segments*. Following a ray-segment inside the volume can be handled efficiently by exploiting the connectivity information since identifying the next cell reduces to determination of the exit face as the entry point of the ray to the next cell is the exit point of the ray from the current cell. Thus, solving point location and view sort problems reduces to generating ray segments and following them in the volume cell by cell, which are referred to here as ray-segment generation and next-cell operation, respectively. Ray-segment generation corresponds to the *first-cell* operation mentioned in the literature. Existing methods mainly differ in ray-segment generation and next-cell operation.

Garrity [6] resolves the ray-segment generation problem by geometrically sorting all external faces into a coarse 3D mesh. Only the faces in the mesh regions that are intersected by the ray are tested to generate the first ray-segment for the respective pixel. When the ray exits the volume through an external face, this procedure is repeated to generate the next ray-segment. For the next-cell operation, all the faces (except the entry face) of the current cell are intersected with the ray and the minimum of the intersections is chosen as the exit point.

Koyamada [7] projects and scan converts only the front external faces in sorted order according to their centroids for ray-segment generation. In his work, he states that better polygon sorting algorithms such as list-priority algorithms [8] can be used to generate high-quality images. For the next-cell operation, he proposes a ray-face intersection test that directly determines if the face is intersected by the ray. So his scheme tests two faces for each tetrahedral cell to determine the exit point on the average whereas Garrity's scheme [6] always tests three faces.

Bunyk et al. [9] present a simple and efficient ray casting algorithm that uses ideas from [6,10,11]. The algorithm essentially breaks the cells into their corresponding faces and visibility determination is performed after the faces have been transformed into screen space. The actual ray casting is performed independently for each pixel by performing a walk in the cell complex that

is stored as an ordered list of stabbing boundary faces computed for each pixel. Farias et al. [12,13] propose a time critical rendering system for unstructured grids. They use the ray-casting algorithm presented in [9] with some improvements and the algorithms for volume data simplification are also augmented to create hierarchical multiresolution representations. They trade accuracy for speed for achieving the goal of interactivity by placing a time budget in the algorithm.

1.1.2. Object-space methods

Object-space methods are also called *projection* methods. In these methods, the volume is traversed in object-space to perform a view-dependent depth sort on the cells. Then, the cells are projected onto the screen in sorted order to find their contributions on the image plane and composite them. Existing object-space methods differ either in sorting phase or in composition phase.

Max et al. [14] and Williams [15] present algorithms, which are linear in the number of faces, for the visibility ordering of acyclic convex meshes composed of convex polyhedra. Both Williams' algorithm, called Mesh Polyhedra Visibility Ordering (MPVO), and the algorithm proposed by Max et al. exploit the connectivity information to perform the visibility ordering efficiently. Williams also proposes heuristics for visibility ordering of non-convex meshes by filling the cavities introducing non-convexities with imaginary tetrahedral cells. Unfortunately, he reports that these heuristics are valid for only limited cases. Silva et al. [16] propose an extension of the MPVO algorithm, called XMPVO, to remove the assumption of MPVO algorithm that the mesh be convex and connected. In this way, the proposed XMPVO works for nonconvex meshes as well without resorting to heuristics. XMPVO algorithm employs the sweep paradigm to determine an ordering between pairs of boundary cells that can obstruct one another. Then, it uses the MPVO algorithm to exploit the ordering implied by adjacencies within the mesh. So, the directed acyclic graph (DAG) used in MPVO algorithm to store the cell ordering within the mesh is augmented by the partial ordering of the boundary cells. BSP-XMPVO algorithm proposed by Comba et al. [17] is an order of magnitude faster than XMPVO algorithm. This speed-up is obtained by moving the XMPVO view-dependent DAG augmentation into a view-independent preprocessing phase, based on constructing a Binary Space Partition tree on the set of boundary faces of the mesh.

Stein et al. [18] present an $O(n^2)$ method to sort n arbitrarily shaped convex polyhedra by generalizing *Painter's Algorithm* [19] for polygons to 3D elements. Yagel et al. [5,20] propose a fast approximation algorithm based on incremental slicing for visibility ordering. At each slice, which is a sweep plane *parallel* to the image plane, contributions of the polygons formed by the cells intersecting the slice are composited with the previously accumulated image from the preceding slices. In their work, they report an adaptive slicing scheme to increase image quality compromising the speed.

Max et al. [14] present an accurate but computationally intensive method to process polyhedral cells for composition. The method scan converts both front and back faces of each cell performing the interpolations in pixel basis. *Projective Tetrahedra* technique, proposed by Shirley and Tuchman [21], calculates the contribution of each cell with a set of partially transparent triangles. This polygon-oriented method is faster than the previous pixel-oriented approach as conventional graphics hardware can be exploited. Stein et al. [18] present extensions to Projective Tetrahedra algorithm for compositing colored elements with hardware assisted texture mapping. Wittenbrink [22] propose optimizations to Projective Tetrahedra algorithm using OpenGL triangle fans, customized quicksort, memory organizations for cache efficiency, display lists and tetrahedral culling.

Lucas [23] proposes a projection algorithm based on cell faces for irregular volume datasets. In this algorithm, after the faces of all the volume cells have been sorted using Painter's Algorithm, each face is scan converted.

Koyamada et al. [24] propose an algorithm that realizes volume rendering by accumulating parallel layers of partially transparent triangles perpendicular to viewing rays. Generation of these layers causes a major performance bottleneck. As a solution to this problem, Koyamada and Itoh [25] propose to generate these slicing surfaces from seed cells that are automatically determined according to the extremum points of the values of distances from a viewing point.

Cignoni et al. [26] also use projective methods for the interactive visualization of tetrahedral meshes by using multiple resolutions of the volume data. They built the multiresolution models for the volume data by using off-line data simplification techniques.

1.1.3. Hybrid methods

In the existing hybrid methods, the volume is traversed in object order such that the contributions of the cells to the final image are accumulated in image order, which is referred to here as *object-then-image* traversal order.

Challinger [27] employs a scanline z-buffer based algorithm to solve the point location and view sort problems. A y-bucket is used to sort faces with respect to their y coordinates in the projection coordinate system. An active face list is created for each scanline using the y-bucket list. The active faces are sorted into an x-bucket with respect to their x coordinates in increasing order. When processing pixels in the current scanline, an active face list is created for the current pixel using the

x-bucket. In this way, the number of faces to be tested for intersection is reduced considerably. Wilhelms et al. [10] propose a similar algorithm for hierarchical and parallelizable rendering of unstructured grids.

Giertsen [28] utilizes a 2D scan-plane buffer to store information within the plane perpendicular to a scan-line. In this approach, z-dimension is discretized due to the scan-plane buffer. The algorithm processes scanlines from top to bottom and determines the intersections of the cells with the respective scan-plane in an incremental manner using a list of active cells whose y-extents cover the current scanline. The volume elements intersecting the current scan-plane are sliced by finding the edge intersections of faces of volume elements with the current scan-plane. Then, each slice is divided into triangles and each triangle is further decomposed into line segments in the z direction. The composition is carried out by processing the line segments in front-to-back order and linearly interpolating them along the ray. The quality of the images and the performance of the algorithm is heavily dependent on the discretization level of the scan-plane buffer.

Silva et al. [29,30] extend Giertsen's method to avoid the discretization introduced by the scan-plane buffer, therefore allowing accurate rendering even for grids with large cell-size variation. The proposed algorithm, called Lazy Sweep Ray-Casting (LSRC), uses many optimizations to generate line segments along the ray efficiently. This is done by using a 2D ray-casting procedure based on a sweep in each scan plane. It avoids the explicit transformation of vertices and the sorting phase by maintaining only a subset of vertices during the 3D sweep. The LSRC algorithm can also handle cyclic meshes.

Westermann and Ertl [31] also use the sweep-planes in a two-pass rendering approach. In their algorithm, first the volume primitives are drawn in polygon mode to obtain their cross-sections in the VSBUFFER orthogonal to the viewing plane. Then, this buffer is traversed in front-to-back order and the volume integration is performed. In this way, the sorting complexity is reduced since it is done in 2D, similar to the methods presented in [28,29]. In addition, explicit connectivity information is not needed, allowing for the rendering of arbitrary scattered, convex polyhedra. In [32], they extend the idea of using graphics hardware by using the features of OpenGL, such as stencil buffer operations for clipping geometries, and using simple polygon drawing and frame buffer operations to speed-up the volume rendering.

1.2. Contributions

In this paper, three distinct categories, namely *image-space*, *object-space*, and *hybrid* methods, are investigated for fast direct volume rendering of unstructured grids. The main objective is to identify the relative superiority and inferiority of the algorithms in these categories. At least one algorithm from each category is implemented and experimented in a common framework. All the algorithms are capable of rendering acyclic non-convex meshes. Here, non-convexity does not only refer to concavity on the boundaries, but also covers disconnectedness and holes of the volume. Besides, the algorithms produce outputs at the same level of image quality. The following features are identified for a fair comparison of the algorithms:

1. early ray termination,
2. generality; cyclic meshes,
3. coherency utilization; *image-space coherency* and *object-space coherency*, and
4. redundancy; redundancies in cell processing and image-space coherency utilization.

Early ray termination is an optimization method used by many algorithms. The aim is to stop following the ray when opacity reaches a user defined threshold. *Generality* is defined as the capability of handling cyclic meshes. Image-space coherency relies on the observation that rays shot from nearby pixels are likely to pass through the same cells involving similar calculations. Image-space coherency can be exploited to speed up ray-face intersections. Object-space coherency uses the connectivity information available in the data. For example, when a ray enters a cell, it must exit through a back face of it. Hence, only the neighbor cells should be checked by using the connectivity information to determine the next cell.

Some algorithms perform redundant operations that slow down the rendering process. Two types of redundancies are identified. For the lighting model employed here (see Section 2), only the cells that are intersected by at least one ray contribute to the final image. The processing of cells that have no effect on the image is referred as the redundancy in cell processing. Image-space coherency is very important and has an important impact on the speed of the rendering algorithm. However, utilization of image-space coherency for cells with small projection areas may be more costly than employing a naive ray-casting approach. This type of redundancy is referred as redundancy in image-space coherency utilization.

Image-space approaches support early ray termination, generality, utilization of object-space coherency and both types of non-redundancies. Object-space methods support only the utilization of both object-space coherency and image-space coherency, failing to support other features. Hybrid approaches support generality, image-space coherency and object-space coherency utilization. Koyamada's algorithm, being one of the outstanding algorithms of image-space methods, is selected as the representative of image-space

approaches in this framework. One object-space algorithm, called Cell Projection (CP), and one hybrid algorithm, called Span-Buffer Ray-Casting (SBRC), are proposed and implemented. Another algorithm, namely Koyamada-SBRC (K-SBRC), stemmed from the idea of refining image-space and hybrid methods to extract the best features of each, is realized by blending Koyamada's and SBRC approaches.

The CP algorithm is similar to the other object-space methods exploiting image-space coherency. Therefore, it is faster than image-space methods. Unlike the object-space methods proposed in [5,20,21,33], CP handles the interpolations in face basis rather than cell basis thereby providing the capability to yield high-quality images as in [14]. However, CP scan converts each internal face only once whereas the scheme proposed by Max et al. [14] scan converts each internal face twice. Furthermore, CP is capable of rendering non-convex meshes through a simple yet efficient sorting schema exploiting both image-space and object-space coherencies. CP is also similar to the MPVO [15] and XMPVO [16] algorithms. However, CP avoids the construction of the DAG needed in these algorithms by generating the visibility order on-the-fly during the rendering phase with little overhead.

Despite the high performance of object-space methods, their shortcoming in handling cyclic meshes have constituted the major motivation towards hybrid methods. However, object-then-image traversal schema forces the existing hybrid methods to process all the volume data. Thus, they cannot support early ray termination. Furthermore, they suffer from both types of redundancies by the same reason. The SBRC algorithm is developed to overcome the deficiencies of existing hybrid methods by changing the traversal order to *image-then-object*. In this way, SBRC gains the ability to support early ray termination and avoids the redundancy in cell processing without compromising the full utilization of both image-space coherency and object-space coherency. Thus, it extends the set of features supported by hybrid methods to include early ray termination and non-redundancy in cell processing.

The SBRC algorithm does not support non-redundancy in image-space coherency utilization. The image-then-object traversal order used in SBRC can be exploited to process cells with small projection areas in a more cost-effective way. This could be done by employing a ray-casting schema that ignores image-space coherency, but still performing better. This idea motivated the development of K-SBRC algorithm by blending SBRC and Koyamada's algorithms. To determine the cell-processing schema to be employed, two schemes are proposed. These are Exact Area and Bounding Box Area schemes. Hence, K-SBRC supports the non-redundancy in image-space coherency utilization in addition to all the features supported by SBRC, thus covering all the desired features. Table 1 shows the supported features for each algorithm.

The rest of the paper is organized as follows. Data model, lighting model and sampling scheme employed in the algorithms are summarized in Section 2. Our implementation of Koyamada's algorithm is described in Section 3. The proposed CP, SBRC and K-SBRC algorithms are presented in Sections 4–6, respectively. Experimental results are presented in Section 7. Section 8 gives conclusions.

2. Preliminaries

The data model common to all algorithms presented in this work is tetrahedral cell model. In the tetrahedral model, faces are triangles and internal faces are shared exactly by two cells. An external face is a face that belongs to only one cell and is not shared by any other cell. Therefore, the set of external faces forms the boundary of the volume.

Low-density particle light source model [7] is employed for lighting calculations. This model assumes that the volume to be visualized consists of low-density

Table 1
Supported features of direct volume rendering methods for unstructured grids

Category	Algorithm	Cyclic meshes	Early ray term	Coherency utilization		Non-redundancy in		Image accuracy
						Cell proc.	ISC util.	
				ISC	OSC			
Image-space	Koyamada	√	√		√	√	√	√
Object-space	CP			√	√			√
I–O	SBRC	√	√	√	√	√		√
Hybrid I–O	K-SBRC	√	√	√	√	√	√	√
O–I	LSRC	√		√	√			√

ISC and OSC denote image-space and object-space coherencies, respectively. I–O and O–I denote image-then-object and object-then-image traversal orders, respectively.

particle light sources. All algorithms use the front-to-back composition schema to allow *early ray termination*.

Both *equi-distant* and *mid-point* sampling schemes are implemented in all algorithms. In mid-point sampling, a new sample is generated in the middle of the line segment formed by the entry and exit points of the ray intersecting the cell. Equi-distant sampling generates samples at fixed intervals of length Δt. Hence, more than one sample could be generated for some cells, but there could be cases in which no sample is generated for a cell. This problem may be solved by *adaptive sampling* [34], which chooses Δt small enough so that at least one sample will be generated in each cell. This scheme is not implemented because of its high computational cost.

In the conventional method, a scalar value at a sampling point inside a tetrahedral cell is computed by 3D inverse distance interpolation of the four vertices of the cell with respect to the sampling point. However, this is an expensive operation and it is repeated as many times as new samples are generated inside a cell. To speed up this process, *linear sampling* method exploits the fact that the change of the scalar in any direction is linear in a tetrahedral cell [7]. It estimates the scalar at a point along a line segment using 1D inverse distance interpolation of the entry and exit points of the tetrahedral cell. The scalars at the entry and exit points are estimated by using 2D inverse distance interpolation of the three vertices of the respective triangular faces. Linear sampling method is used in all algorithms.

Linear sampling method is faster than the conventional method for equi-distant sampling schema only when Δt is small. Linear sampling method may be expected to perform worse than the conventional method for mid-point sampling schema. However, in all algorithms presented, values needed for linear sampling method are calculated at a very low cost by utilizing the results of the computations performed during the intersection tests. Therefore, linear sampling method performs better than the conventional method even for mid-point sampling schema.

The major data structures common to all algorithms implemented in this work are described as follows. Tetrahedral cell data is stored in two arrays, namely a node array and a cell array. Node array keeps the scalar value and the x, y, and z coordinates for each node. Cell array stores data about the vertices and the neighbor cells of each cell. The other major data structure is the ray buffer structure. It is a 2D virtual array that holds a linked list of ray-segments for each pixel.

3. Koyamada's algorithm

Koyamada's algorithm is a ray-casting approach that makes use of the coherence in the image-space to generate rays and follows those rays in the object-space. The algorithm is given in pseudocode in Fig. 1.

The first step of Koyamada's algorithm is to generate the ray-segments. In his original algorithm, the front external faces are sorted with respect to z coordinates of their centroids in increasing order. In fact, this is an approximate order, which may be wrong in some cases [7]. To alleviate this problem, this step of the original algorithm is slightly modified. Instead of sorting the front external faces at the beginning, we scan convert them one by one. For each pixel covered by the projection area of an external face, we insert a list item into the respective ray list. Note that the same ray-segment generation scheme is adopted in all proposed algorithms.

After the rays are created, each ray is followed in the volume utilizing the connectivity information between cells. To trace a ray inside the volume, two things have to be known for each cell that is intersected by the ray:

- the entry face and the (z,s) values at the entry point to the cell and
- the exit face and the (z,s) values at the exit point from the cell.

Here, (z,s) pair stores the z-coordinate and the scalar value at the ray-face intersection point. Since the exit-point values from a cell can be used as the entry-point values to the next cell, the problem of tracing a ray-segment inside the volume reduces to the problem of determining the exit point from a cell, given the first entry-point values for each ray segment.

Koyamada's ray-face intersection method relies on the observation that if a ray intersects a face then the pixel that the ray is shot must be covered by the projection area of that face on the screen. So, he uses the projected area of a face to determine if the ray exits the cell from that face by using the normalized projection coordinates of the vertices of the face. This is done as follows. Consider a ray r shot from pixel (x_r, y_r) that intersects a tetrahedral cell $ABCD$ through point P of entry-face ABD. Let triangle ACD be the face of the cell that is subject to the ray-face intersection test. If the face is perpendicular to screen then the ray does not leave the cell through that face, so another face of the cell is tested. Otherwise, ray r intersects the plane determined by triangle ACD at a point Q, where $x_r = x_P = x_Q$ and $y_r = y_P = y_Q$. Then, vector $\overrightarrow{AQ}$ can be expressed as $\overrightarrow{AQ} = \alpha\overrightarrow{AC} + \beta\overrightarrow{AD}$, where the weighting values (α, β) are found by solving

$$\begin{bmatrix} x_C - x_A & x_D - x_A \\ y_C - y_A & y_D - y_A \end{bmatrix} \times \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} x_r - x_A \\ y_r - y_A \end{bmatrix}. \quad (1)$$

If α and β do not satisfy the conditions $\alpha \geqslant 0$, $\beta \geqslant 0$ and $\alpha + \beta \leqslant 1$, then Q is not inside ACD, so another face is

```
// Ray segment generation
for each front external face f do
   scan convert f generating ray-segments in
   sorted order according to z-coordinates

// Ray tracing and composition
for each scanline y do
   for each x in scanline y do {
      for each ray-segment r shot at pixel (x, y) do {
         read entry cell c and entry values (z_in, s_in) of r into volume
         while r is inside volume {
            find exit face of r from cell c
            calculate exit values (z_out, s_out)
            resample from (z_in, s_in) to (z_out, s_out)
            take composition onto pixel (x, y)
            (z_in, s_in) ← (z_out, s_out)
            determine next cell c
         }
      }
   }
```

Fig. 1. Koyamada's algorithm.

tested. Otherwise, Q is inside ACD, so no further tests need to be done.

As the exit face ACD is identified, the (z, s) values (z_Q, s_Q) at the exit point Q are calculated as:

$$(z_Q, s_Q) = (z_A + \alpha(z_C - z_A) + \beta(z_D - z_A), \\ \alpha s_C + \beta s_D + (1 - \alpha - \beta)s_A). \quad (2)$$

Note that the expression for s_Q is 2D inverse distance interpolation of the vertices of face ACD with respect to point Q. *View Reference Coordinate System* and *Normalized Projection Coordinate System* are taken to be the same so that the same weighting values (α, β) computed in Eq. (1) for the successful ray-face intersection test can be used in computing (z_Q, s_Q) values according to Eq. (2). Since the exit-point (z_Q, s_Q) values are computed and the entry-point (z_P, s_P) values are known, the sample(s) along the line segment PQ can be taken and composited using the linear sampling method discussed earlier.

4. Cell projection algorithm

The CP algorithm falls into the *projection* methods category of direct volume rendering methods. The algorithm runs by projecting the cells onto the screen one at a time. To project a cell, the cell before it on the ray path should be projected beforehand.

CP starts by scan converting the front external faces to generate ray segments in sorted order just like in Koyamada's algorithm. The rest of the algorithm consists of two phases; *initialization for visibility ordering* (see Fig. 2) and *rendering* (see Fig. 3). In the first phase, the information to be used in constructing the visibility order among the cells is gathered and in the second phase the cells are processed for sampling and composition while the visibility order is constructed gradually.

Unlike Koyamada's, SBRC and K-SBRC algorithms, which composite the image pixel by pixel, CP requires the explicit maintenance of a partial image-buffer since it is a pure object-space method. Image-buffer should maintain a color-opacity component for each pixel. It stores the composited RGB color values and the opacity value for a pixel. In order to reduce the memory overhead, we embed these components to the respective ray-lists for only active pixels.

In CP, a visibility order with respect to a view plane is found by using a sorting schema that minimizes both the additional memory requirement and the execution time. It uses the concept of *dependency* between the cells. Cell a is *dependent* of cell b, if cell b obstructs cell a in the

```
// Ray segment generation
for each front external face f do
   scan convert f generating ray-segments in
   sorted order according to z-coordinates

//Generate internal dependencies
for each cell c do
   for each face f of cell c do
      if f is an internal front face then
         increment indegree of cell c

//Generate external dependencies
for each scanline y do
   for each x in scanline y do
      for each ray-segment r shot at pixel (x, y) do
         if r is not first ray segment shot at (x, y) then {
            read entry cell c of r into volume
            increment indegree of cell c
         }

//Initialize cell queue
for each cell c do
   if indegree of cell c is 0 then
     enqueue cell c into cell queue
```

Fig. 2. Cell projection algorithm: initialization for visibility ordering.

visibility order. If a is dependent of b, then b must be processed before a. We define two kinds of dependencies: *internal* and *external*. An internal dependency occurs between each pair of neighbor cells sharing a face. An external dependency may only occur between a pair of external cells when the projection areas of their external front faces overlap. If the data is known to be a convex set, then the external dependency generation step need not to be performed. If an internal dependency exists between a pair of cells then this dependency will always exist, but its direction may change with varying viewing parameters. However, in the case of external dependencies, both the dependencies and their directions may change with changing viewing parameters. Each internal face that is not orthogonal to the image plane always induces an internal dependency whereas only external faces may be the source of external dependencies. These two types of dependencies are constructed during the initialization phase by calculating the indegree of each cell, which is the number of obstructions for the cell in the visibility order (see Fig. 2). Note that each internal dependency contributes by one to the indegree of a cell, whereas each external dependency contributes by an amount equal to the number of pixels shared between the projection areas of the external front faces of the external cell pair. Fig. 4 (a) illustrates a sample case for indegree assignments. Note that indegree value of 3 for cell F stems from an internal dependency to cell E and 2-pixel external dependency to cell A.

The rendering phase begins by traversing the ray buffer to replace the first-ray segment of each active pixel with an active ray item. Active ray items represent the active ray-segments in the respective pixels during

```
// Rendering
while cell queue is not empty {
   dequeue a cell c from cell queue
   for each face f of cell c do
      if f is an internal back face then {
         nc ← cell that is neighbor to cell c through face f
         decrement indegree of nc
         if indegree of nc is 0 then
            enqueue nc into cell queue
         scan convert face f
         for each pixel p covered in projected area of face f do {
             read entry (z_in, s_in) values from active ray segment r at p
             calculate exit values (z_out, s_out)
             resample from (z_in, s_in) to (z_out, s_out)
             take composition onto pixel p
             (z_in, s_in) of active ray segment r ← (z_out, s_out)
         }
      }
      elseif face f is an external back face {
         scan convert face f
         for each pixel p covered in projected area of face f do {
             read entry (z_in, s_in) values from active ray segment r at p
             calculate exit values (z_out, s_out)
             resample from (z_in, s_in) to (z_out, s_out)
             take composition onto pixel p
             (z_in, s_in) of active ray segment r ← (z_out, s_out)
             if another ray segment exists for pixel p then {
                r ← next ray segment for pixel p
                read entry cell c of r into volume
                decrement indegree of cell c
                if indegree of cell c is 0 then
                   enqueue cell c into cell queue
             }
         }
      }
}
```

Fig. 3. Cell projection algorithm: rendering phase.

the course of the algorithm. The rendering phase continues by inserting the indices of the cells with indegree 0 into the cell queue. Fig. 4 shows a sample case from the execution of the algorithm. There are two tasks to be performed in processing of each cell; removing the respective dependencies induced by this cell and scan converting its back faces. The internal dependencies are removed by decreasing the indegree fields of the neighbor cells by 1, which are connected to this cell through its back internal faces. The process of removing external dependencies and scan conversion of a back face are performed in an interleaved manner for efficiency.

As a back face is scan converted, the resulting (z,s) values and the (z,s) values stored in the entry (z,s) component of the active ray item of the ray-list in each covered pixel are used as the exit-point and entry-point values of the rays from and into the cell, respectively, for sampling and composition operations. The color and opacity values obtained from the sampling are compos-

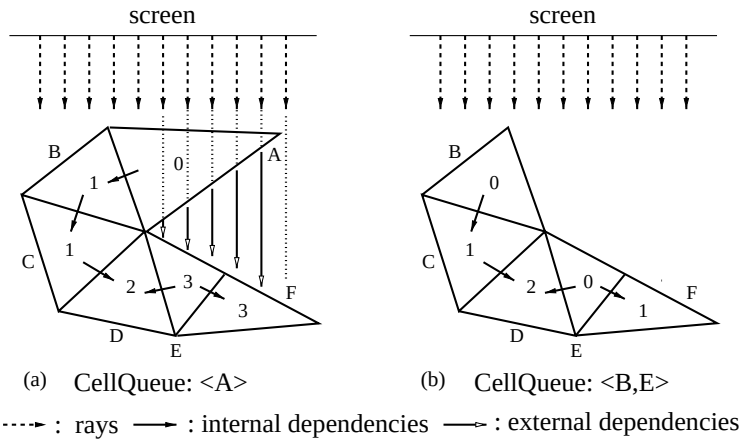

Fig. 4. A sample case from the rendering phase of the CP algorithm: (a) the cell queue is initialized with cell A, (b) after cell A is processed, its dependents B and E, are placed into the cell queue. Numbers inside cells show their current in degree values.

ited onto the color and opacity components of the respective active ray item, and its entry (z, s) component is replaced by the exit-point values obtained from the scan conversion. If the back face is an external face, then it means that the active ray-segment is leaving the volume. So if there is a ray-segment after the active ray item in the respective ray-list, then we decrement the indegree field of the cell that generated the ray-segment, thus effectively removing one of the pixel-basis external dependencies induced by the cell being processed.

In the case of an acyclic mesh, the cell queue will become empty after all cells are processed. In the case of a cyclic mesh, the cell queue will become empty before all cells are processed, thus showing the existence of cycle(s) in the volumetric dataset from the given viewing parameters.

Like all projection algorithms, CP exploits image-space coherency. Our internal-dependency generation scheme is similar to the sorting schemes proposed by Max et al. [14] and Williams [15] for convex meshes. It exploits the object-space coherency through connectivity information. Our external-dependency generation scheme enhances the algorithm to handle non-convex meshes at a very low cost. It efficiently exploits the ray-segments generated for the rendering phase, thus effectively utilizing image-space coherency. Our internal dependency generation scheme runs in linear time in the number of internal faces, and external dependency generation scheme runs in linear time in the sum of the projection areas of external front faces. Another nice feature of the algorithm is that it does not generate a directed dependency graph explicitly for topological sorting. Instead, it generates the visibility order on the fly during the rendering phase by using the indegree information of the cells gathered during the initialization phase. Unlike approximate object-space methods, CP handles the interpolations in face basis rather than cell basis thereby providing the capability to yield better quality images as in [14]. Besides, CP scan converts each face only once. Beyond these advantages, CP—being an object-space method—suffers from redundancies in cell processing and image-space coherency utilization, since it has to sort and scan convert all cells in the volume. Furthermore, as all other object-space methods, it cannot handle cyclic meshes and cannot support early ray termination.

5. Span-buffer ray-casting algorithm

Existing hybrid methods suffer from inability to support early ray termination and non-redundancy in cell processing. The Span-Buffer Ray-Casting (SBRC) algorithm is a hybrid method proposed to overcome these deficiencies by changing the computational traversal order from object-then-image to image-then-object without compromising full utilization of image and object space coherencies. Fig. 5 gives the pseudo-code for the algorithm. SBRC requires three additional data structures to maintain active cells, active edges and span buffers for the active cells.

The algorithm is inspired by the observation that a ray intersects a face if and only if the pixel that the ray is shot from is covered by the projected area of that face. SBRC follows the rays as in Koyamada's algorithm using the connectivity relation. When a cell is hit by a ray for the first time, its span for the current scanline is created and buffered. Each pixel of the span-buffer contains the (z, s) values of the exit point and the exit-face identifier for the respective ray. The current ray uses

```
//Ray segment generation
for each front external face f do
   scan convert f generating ray-segments in
   sorted order according to z-coordinates

//Ray tracing
for each scanline y do {
   for each pixel x in scanline y do
      for each ray-segment r shot at pixel (x, y) do {
         read entry cell c and entry values (z_in, s_in) of r into volume
         while r is inside volume {
            if cell c is not active then
               activate cell c
            if cell c doesn't have a span created for scanline y then
               create span of cell c for scanline y
            read exit values (z_out, s_out) from span
            resample from (z_in, s_in) to (z_out, s_out)
            take composition onto pixel (x, y)
            (z_in, s_in) ← (z_out, s_out)
            determine next cell c
         }
      }
}
```

Fig. 5. Span-buffer ray-casting algorithm.

the information in the first pixel of the span-buffer for sampling and enters the neighbor cell for which exit-point values will be used as the entry-point values. When a ray shot from the same scanline hits the cell, the information in the respective pixel of the span-buffer of the cell is directly used for sampling and next-cell operation. When the cell is hit by a ray shot from the next scanline for the first time, a new span is created and buffered for the cell. Fig. 6 shows a sample case of following a ray in SBRC.

In the rendering phase, we activate a cell when it is hit by a ray for the first time. The cell-activation process begins by allocating an entry in the active cell list for the cell. Then, we identify the edges necessary for the scan conversion of back faces of the cell. A tetrahedral cell has 6 edges and according to the view point at least 2 and at most 6 edges might be necessary for the activation process. The edges belonging to at least one back-face are called back-face edges. Then, we sort these back-face edges to find an order on the edges such that when they are cut by a virtual line parallel to a scanline, the intersection points will always appear sorted in increasing order of x coordinates. This sorting operation is performed only once when the cell is activated.

After the activation of the cell, its span-buffer for the current scanline is created. We need to identify the new states of the back-face edges of the cell for scan conversion along the current scanline. The scanlines are processed from top to bottom. We compare the y-extent of each edge, which is not currently in *done* state, with the current scanline. The states of the edges whose y-extents are above, intersecting and below the current scanline are set to *done*, *active* and *inactive*, respectively. If an edge passes from inactive to active state then it is searched in the active edge list through hashing. After this step, we have a sorted list of the intersection points of the current scanline with the active back-face edges of the cell. The current (x, z, s) values obtained for the successive intersection-point pairs in sorted order are used for scan conversion to compute the interpolated (z, s) values for the successive pixels covered by the cell along the current scanline.

As the span-buffer for the first scanline intersecting the cell is created, we can read the values from the

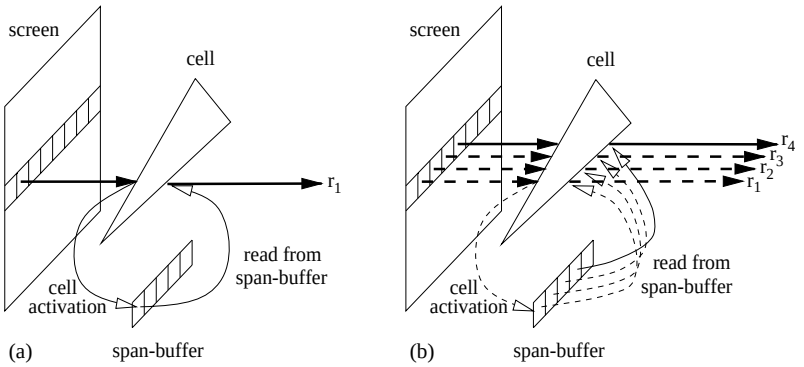

Fig. 6. A sample case of following a ray in SBRC algorithm. (a) Ray r_1 hits the cell, activates it, creates the span-buffer, reads the exit-point values from the span-buffer and continues in the volume. (b) The succeeding rays r_2, r_3, r_4 on the same scanline hit the cell and directly read the exit-point values from the span-buffer.

created span-buffer to determine the exit face and the (z, s) values at the exit point of the ray from the cell. The exit face will be used to find the next cell hit by the ray and the (z, s) values at the exit point will be used as the entry-point values at the next cell. Then, the ray is followed as in Koyamada's algorithm. When another ray hits the same cell later, we check whether the ray belongs to the same scanline. If this is the case, we know that the current span-buffer of the cell is valid and the information in the respective pixel of the span can be directly used for sampling and next-cell operation. If they are not equal then we know that the information in the span-buffer is not valid for this scanline and we should create the new span-buffer of the cell for this scanline. Active edge list is traversed after processing each scanline to delete the edges that will never be accessed again. The edges that are not deleted are rasterized for the next scanline by updating their coordinates.

Supporting early ray termination in SBRC needs special attention in span-buffer and cell-entry deletions. The active life of the span-buffer of a cell ends when the ray shot from its rightmost pixel location hits the cell. The active life of a cell ends when it is hit by the ray shot from the rightmost bottom pixel covered by the projected area of the cell. However, we can never be sure that these rays will travel enough in the volume to hit the cell due to early ray termination. Hence, a scanline-based deletion scheme is adopted in the dynamic data structures. For span-buffer deletions, the span buffer list is initialized through a simple pointer operation to overwrite the previous span-buffers with the new span-buffers to be created for the next scanline. For cell-entry deletions, we use a y-bucket structure, which has a size equal to the height of the screen. After the activation of each cell, the pointer to the respective entry in the active cell list is inserted into the y-bucket list according to its bottom y-coordinate. After processing a scanline, the pointers in the respective y-bucket list are used to delete the cell entries in the active cell list.

SBRC may be viewed as being the hybrid of Koyamada's [7], Challinger's [27], and LSRC [30] algorithms. It tries to exploit the best features of these algorithms. Koyamada's algorithm suffers from the lack of exploiting the image-space coherency whereas SBRC utilizes both image and object space coherencies. SBRC avoids the extensive sorting operations in Challinger's algorithm by making use of the connectivity information. Both SBRC and LSRC algorithms make maximum use of both image and object space coherencies. However, the sorting operations performed during the 3D and 2D sweeps may degrade the performance of the LSRC algorithm. Furthermore, object-then-image traversal schemes adopted in Challinger's and LSRC algorithms compel these algorithms to process all cells, thus preventing them to support early ray termination and making them suffer from redundant computations especially in low resolutions as the portion of the cells contributing to the image is relatively small. The image-then-object space traversal scheme adopted in SBRC overcomes all these deficiencies since a cell is activated only if it is hit by a ray.

6. Koyamada-SBRC algorithm

In the SBRC algorithm, for cells whose projection areas occupy a small number of pixels, the benefit of using image-space coherency by scan conversion might be suppressed by the overhead of the cell activation process. So Koyamada's algorithm can outperform SBRC when there is a large number of cells with small projection areas. The similarity between Koyamada's and SBRC algorithms allows the emerging of a new hybrid method called *Koyamada-SBRC* (K-SBRC) algorithm that blends these two approaches. The main

```
// Ray segment generation
for each front external face f do
   scan convert f generating ray-segments in
   sorted order according to z-coordinates

//Ray tracing
for each scanline y do {
   for each pixel x in scanline y do
      for each ray-segment r shot at pixel (x, y) do {
         read entry cell c and entry values (z_in, s_in) of r into volume
         while r is inside volume {
            if rendering schema is not selected for cell c then
               select rendering schema for cell c
            if rendering schema for cell c is Koyamada then
               use Koyamada's algorithm for ray-segment r and cell c
            else
               use SBRC algorithm for ray-segment r and cell c
         }
      }
}
```

Fig. 7. Koyamada-SBRC algorithm.

idea is to use Koyamada's algorithm to render a cell when the cost of using SBRC to render the cell is more costly. This approach will also reduce the space complexity of SBRC by not activating each cell hit by a ray. The pseudocode for the K-SBRC algorithm is given in Fig. 7.

In order to decide which algorithm should be employed to process a cell, we should develop a *bias*. If we can estimate the amount of time to be spent to render a cell by each of the two algorithms with a small error then the algorithm with the smaller rendering time should be used to render the cell. The execution times of both algorithms can be dissected onto four components; time $T_{NT} = \alpha_{NT}N$ spent for transforming nodes from World Coordinate System to Normalized Projection Coordinate System, time $T_R = \alpha_R R$ spent for generating ray-segments by scan converting front external faces, time $T_S = \alpha_S S$ spent for sampling and composition operations, and time T_I spent for computing the ray-face intersections. In Koyamada's algorithm, T_I is equal to the time spent for ray-face intersection tests, i.e., $T_I = \alpha_{IT} I_T$. In SBRC, T_I can be further dissected into three components; time $T_{CA} = \alpha_{CA} C$ spent for cell activation, time $T_{SC} = \alpha_{SC} H$ spent to initialize the scan-conversion process needed for span-buffer creation, and time $T_{I'} = \alpha_{I'} I$ spent for span-buffer creation and incremental ray-face intersection using the span-buffers. Note that T_{SC} involves inserting (edge activation) and retrieving edges to and from the active edge list. So the expressions for the rendering times of a dataset by Koyamada's and SBRC algorithms are:

$$T_{Koy} = \alpha_{NT}N + \alpha_R R + \alpha_S S + \alpha_{IT} I_T, \tag{3}$$

$$T_{SBRC} = \alpha_{NT}N + \alpha_R R + \alpha_S S + \alpha_{CA} C + \alpha_{SC} H + \alpha_{I'} I, \tag{4}$$

respectively. In both equations, N is the number of nodes in the data, R is the number of ray-segments generated, S is the number of samples taken, and α_O is the unit cost of the respective operation "O". I_T in Eq. (3) denotes the number of ray-face intersection tests performed by Koyamada's algorithm, whereas I in Eq. (4) denotes the number of ray-face intersections. In Eq. (4), C and H denote the number of cells activated and the number of spans created, respectively, by SBRC.

In bias computation, we can ignore T_{NT}, T_R and T_S times because the same routines are employed in both algorithms. Consider a cell c with a projection area of a^c in terms of pixels, and height (number of spans) h^c. Then, the expected execution cost of each algorithm for

cell c can be written as

$$(a)\ t^c_{Koy} = \alpha_{IT} i^c_T \approx 2\alpha_{IT} a^c$$
$$(b)\ t^c_{SBRC} = \alpha_{CA} + \alpha_{SC} h^c + \alpha_{I'} a^c \quad (5)$$

where i^c_T denotes the expected number of intersection tests to be performed on cell c by Koyamada's algorithm. As also mentioned earlier, Koyamada's algorithm is expected to perform 2 intersection tests per intersection on the average (i.e., $I_T \approx 2I$). Therefore, i^c_T is approximated by $2a^c$ for cell c in Eq. (5). So, if $2\alpha_{IT} a^c \leqslant \alpha_{CA} + \alpha_{SC} h^c + \alpha_{I'} a^c$ then we should use Koyamada's algorithm, otherwise we should use SBRC to process the cell.

As now the bias is defined, the problem is to find a^c and h^c in a fast way. Two schemes, namely *Exact Area* and *Bounding Box Area*, are developed to estimate a^c and h^c. Exact Area scheme tries to estimate a very close approximation to a^c. To do this, it calculates the areas of all faces of the cell in Normalized Projection Coordinate System and sums them up. The value obtained should be divided into two because originally it is twice as the original coverage area, as both front and back faces are used. Bounding Box Area scheme uses half of the area of the bounding box of the projection area of a cell as an approximation to a^c. As a tetrahedral cell either forms a triangle or a four sided polygon when projected onto the screen, it is very easy to calculate the area of the bounding box of the projection.

Both schemes compute h^c exactly by finding the height of the projection of the cell in Normalized Projection Coordinate System. Exact Area scheme will estimate a^c more accurately, but it is computationally more expensive than Bounding Box Area scheme. Bounding Box Area scheme is expected to overestimate a^c, which in turn will result in favoring SBRC in cases where the difference between projection area and the bounding box area of a cell is large.

The K-SBRC algorithm introduces an overhead of taking a decision on the algorithm to be employed for each cell that is intersected by at least one ray. So, if the bias chooses Koyamada's algorithm for all cells then this will result in a larger execution time than the execution time of Koyamada's algorithm. On the other hand, if the bias chooses SBRC for all cells then this will result in a larger execution time than the execution time of the original SBRC algorithm.

7. Experimental results

7.1. Datasets and environment

Table 2 displays the properties of the datasets used in the experimentations. These four datasets were obtained from NASA-Ames Research Center. All datasets were originally *curvilinear* consisting of hexahedral cells, so we converted them into unstructured standard tetrahedral data format by breaking each hexahedral cell into tetrahedral cells. Fig. 8 shows the rendered images of the datasets. Scalar values in the input datasets are shifted and scaled to fit [0,255] range. A user-specified piecewise-linear transfer function, which is generated automatically based on histogram equalization, specifies the mapping from this range to the set of opacity and RGB values as described in [9,35].

Experimentations were done on a single processor of the shared memory parallel machine, *Sun Ultra Enterprise 4000* computer equipped with 512 Mbytes of memory and 8 UltraSparc II (250 MHz) processors each with a 256 Kbyte level 2 cache.

The relative performances of the presented algorithms are evaluated by the visualization of each dataset using four different views for three image resolutions. Table 3 displays the visualization statistics of each dataset to guide the comparison of memory usage and the performance analysis of the algorithms.

7.2. Memory requirements

Table 4 illustrates the memory requirements of the algorithms for each dataset and three image resolutions as the average of four views. Koyamada's algorithm introduces no additional memory overhead to the existing data structures. The memory overheads introduced by SBRC and K-SBRC algorithms are due to the data structures necessary for keeping the information about the active edges, active cells, and the span-buffers created. The sources of the memory overhead in CP are

Table 2
List of datasets used for testing

Name	Dimensions	N	C
Blunt Fin (BF)	$40 \times 32 \times 32$	40,960	187,395
Combustion Chamber (CC)	$57 \times 33 \times 25$	47,025	215,040
Oxygen Post (OP)	$38 \times 76 \times 38$	109,744	513,375
Delta Wing (DW)	$56 \times 54 \times 70$	211,680	1,005,675

Dimensions are the original NASA Plot3D sizes. N and C denote the number of nodes and tetrahedral cells, respectively.

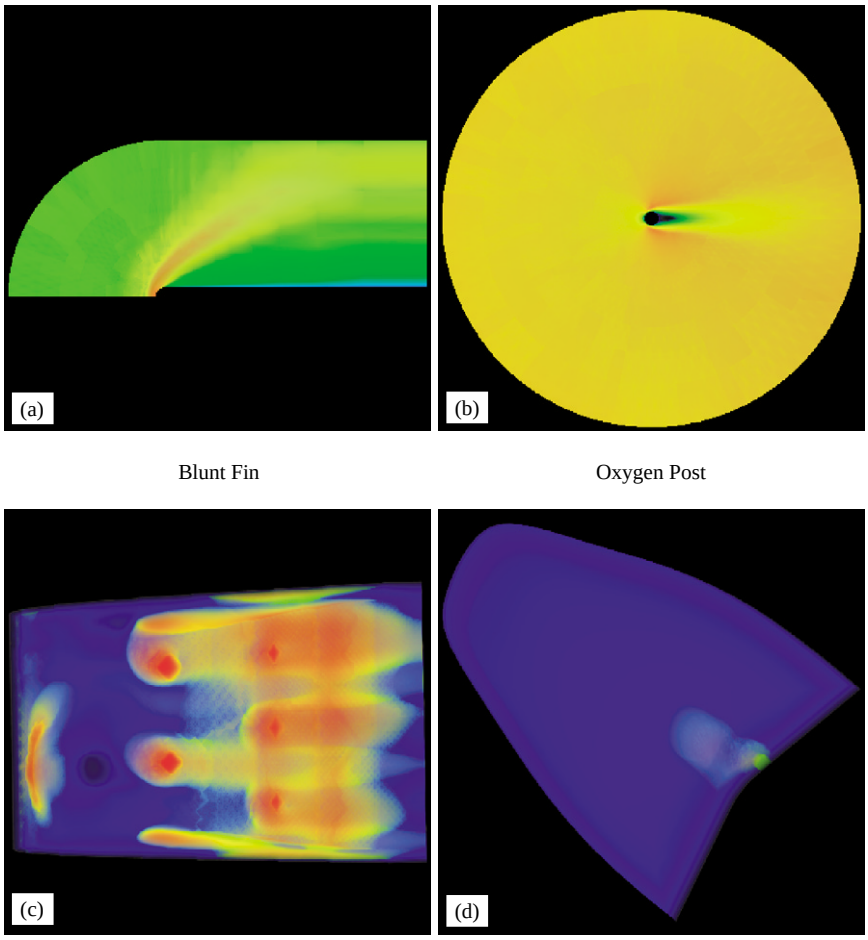

Fig. 8. Rendered images of the datasets. (a) Blunt Fin, (b) Oxygen Post, (c) Combustion Chamber, (d) Delta Wing.

indegree field added for each cell, color-opacity component needed for each active pixel, and the cell queue structure. The major overhead for CP comes from the color-opacity component and this fact can be clearly observed from the higher rate of increase in the memory overhead with increasing resolution relative to the other algorithms.

As shown in Table 4, the percent memory overhead of SBRC with respect to the core data size is always below 30%, and it decreases with increasing dataset size at a fixed image resolution. As expected, K-SBRC behaves even slightly better than SBRC due to less number of cell activations. CP algorithm introduces considerable amount of memory overheads of 60% and 67% for smaller datasets Blunt Fin and Combustion Chamber, respectively, at highest resolution (900×900). However, percent memory overheads drastically reduce below 23% for larger datasets Oxygen Post and Delta Wing. The ray buffer structure occupies more space than the core data for the smaller datasets at highest resolution, but for the larger datasets the percentage of the ray buffer structure to the core data reduces below 50%.

7.3. Performance analysis and comparisons

Table 5 displays the dissection of execution times of the algorithms for V_1. T_R in all algorithms and T_V in CP were accurately measured as they constitute distinct phases. However, T_S and T_I cannot be measured directly because of their highly interleaved manner of execution. To determine T_S and T_I, each program was run twice for each visualization instance, one with sampling and the other without sampling. The measured time of the latter run directly gives T_I, and the measured time difference between the former and latter runs yields T_S. However, dissection of T_I into T_{CA}, T_{SC}, and $T_{I'}$ in

Table 3
Visualization statistics of the datasets for different views

Dataset	Image res.	View											
		$V_1 = (0,0,0)$			$V_2 = (0,90,0)$			$V_3 = (90,0,0)$			$V_4 = (45,45,45)$		
		P	R	I	P	R	I	P	R	I	P	R	I
	300×300	28.4	28.4	2645	57.7	57.7	5584	22.0	22.0	1687	47.9	48.3	3924
BF	600×600	114.1	114.1	10615	231.5	231.5	22408	88.3	88.3	6771	192.4	193.7	15751
	900×900	257.1	257.1	23907	521.5	521.5	50476	198.8	198.8	15251	433.5	436.3	35481
	300×300	54.7	54.7	3655	60.9	66.4	6165	37.0	38.8	3499	52.1	52.4	3412
CC	600×600	219.2	219.2	14671	244.4	266.2	24746	148.5	155.6	14044	208.9	210.3	13696
	900×900	493.8	493.8	33047	550.5	599.7	55741	334.6	350.6	31633	470.5	473.7	30850
	300×300	67.4	67.4	7481	13.7	14.2	1187	13.7	20.8	1188	43.5	47.1	4039
OP	600×600	270.4	270.4	30010	55.2	57.0	4766	55.2	83.7	4768	174.7	189.2	16214
	900×900	609.0	609.0	67599	124.2	128.3	10734	124.2	188.4	10740	393.6	426.3	36524
	300×300	54.9	60.5	7508	46.5	46.7	3312	56.9	56.9	3204	58.5	59.3	5209
DW	600×600	220.1	243.0	30146	186.7	187.3	13294	228.3	228.3	12861	234.5	238.1	20908
	900×900	495.7	547.2	67866	420.7	422.0	29944	514.2	514.2	28970	528.3	536.3	47097

The 3-tuples for each view define the Euler angles of rotation around x, y and z axes, respectively. P, R and I denote the numbers of active pixels (10^3), ray-segments (10^3) and ray-face intersections (10^3), respectively. If $R > P$ then the respective visualization instance is non-convex. Since mid-point sampling schema is used, the number of intersections is equal to the number of samplings ($I = S$).

Table 4
Memory consumptions of the algorithms in MBytes (averages of 4 views)

Dataset	Image resolution	Core	Ray buffer	Memory overhead				
				Koy.	SBRC	K-SBRC		CP
						EA	BBA	
	300×300		1.0	0.0	1.5	1.2	1.2	0.8
BF	600×600	7.2	4.1	0.0	1.8	1.6	1.6	2.1
	900×900		9.2	0.0	2.1	1.9	1.8	4.3
	300×300		1.3	0.0	1.6	1.5	1.3	1.0
CC	600×600	8.2	5.1	0.0	1.7	1.7	1.7	2.7
	900×900		11.4	0.0	1.9	1.9	1.9	5.5
	300×300		1.0	0.0	3.1	2.6	2.4	1.4
OP	600×600	19.6	4.0	0.0	3.5	2.9	2.8	2.6
	900×900		9.0	0.0	3.8	3.3	3.1	4.5
	300×300		1.3	0.0	6.3	4.8	4.7	2.6
DW	600×600	38.3	5.3	0.0	7.0	5.6	5.6	4.4
	900×900		11.8	0.0	7.6	6.4	6.3	7.4

"Core" column denotes the memory occupied by the dataset itself (i.e., node and cell arrays). "Ray-Buffer" column represents the memory usage for the ray buffer structure containing ray-segment lists. The following five columns illustrate the additional memory overhead introduced by the respective algorithms.

SBRC cannot be computed through measurement. Instead, we have estimated the unit costs α_{CA}, α_{SC} and $\alpha_{I'}$ for these operations statistically using the Least-Squares Approximation method, and used these unit costs to determine them approximately as $T_{CA} = \alpha_{CA}C$, $T_{SC} = \alpha_{SC}H$, and $T_{I'} = \alpha_{I'}I$. The average error in estimating T_I using these costs as $T_I = \alpha_{CA}C + \alpha_{SC}H + \alpha_{I'}I$ is measured to be below 6%. Dissection of T_I is not given for K-SBRC because of the increased number of parameters.

Table 5
Execution-time dissection of the algorithms for view V_1

Algorithm	Exec. time	Image resolution											
		300×300				600×600				900×900			
		Dataset											
	(s)	BF	CC	OP	DW	BF	CC	OP	DW	BF	CC	OP	DW
Koyamada	T_R	0.2	0.4	0.5	0.9	0.4	0.8	1.0	1.4	0.7	1.4	1.6	2.0
	T_S	1.2	1.7	4.0	3.3	4.9	7.1	15.1	13.4	10.6	15.6	32.9	29.9
	T_I	9.1	14.0	26.6	25.4	36.6	55.6	105.9	100.7	82.8	125.1	238.4	225.5
	T_T	10.6	16.2	31.3	30.0	42.0	63.6	122.2	115.9	94.1	142.2	273.1	257.8
SBRC	T_R	0.2	0.5	0.6	1.0	0.5	1.0	1.2	1.6	1.0	1.9	2.3	2.7
	T_S	1.3	1.7	3.5	3.6	5.0	6.9	14.2	14.3	11.3	15.6	32.0	32.1
	T_{CA}	1.6	3.5	3.6	8.1	2.2	3.5	4.7	11.9	2.4	3.5	5.3	13.5
	T_I T_{SC}	2.0	6.1	6.1	10.3	5.3	14.2	14.6	26.7	7.0	21.5	24.4	46.0
	$T_{I'}$	1.5	2.2	4.5	4.5	5.3	9.5	18.0	18.1	14.3	21.5	40.6	40.7
	T_T	6.6	14.1	18.6	27.9	18.4	35.4	52.9	73.0	36.2	64.2	104.9	135.5
K-SBRC (EA)	T_R	0.2	0.4	0.5	1.0	0.4	0.8	1.0	1.4	0.7	1.4	1.7	2.1
	T_S	1.4	2.0	3.7	4.5	5.6	7.8	14.9	14.3	11.9	16.1	34.3	32.1
	T_I	5.0	12.8	13.9	19.5	14.2	29.3	38.7	55.4	26.0	50.5	73.8	104.2
	T_T	6.6	15.3	18.3	25.4	20.3	38.0	54.8	71.6	38.7	68.1	109.9	138.8
K-SBRC (BBA)	T_R	0.2	0.4	0.5	0.9	0.4	0.8	1.0	1.4	0.7	1.4	1.7	2.4
	T_S	1.4	1.9	3.6	3.6	5.4	8.0	15.1	14.5	11.1	15.6	33.3	28.7
	T_I	5.0	13.4	13.6	18.8	14.1	28.7	40.0	54.2	26.9	50.4	78.7	113.6
	T_T	6.6	15.7	17.9	23.8	20.0	37.6	56.3	70.4	38.8	67.5	113.9	145.0
CP	T_R	0.2	0.4	0.5	0.9	0.4	0.8	0.9	1.4	0.7	1.4	1.6	2.0
	T_V	0.6	0.9	1.7	3.1	0.9	1.4	2.2	3.6	1.3	2.1	3.1	4.5
	T_S	1.3	1.8	3.5	3.3	5.1	6.7	14.0	13.1	11.6	14.6	32.0	30.8
	T_I	4.6	8.7	13.4	25.5	10.5	18.9	30.3	46.5	19.4	34.1	55.7	75.6
	T_T	6.6	11.4	18.9	32.3	16.6	27.1	46.8	63.6	32.3	50.9	91.0	111.2

T_R, T_S, T_I, and T_T denote ray-segment generation, sampling/composition, ray-face intersection, and total rendering times, respectively, In SBRC, T_{CA}, T_{SC}, and $T_{I'}$ denote cell-activation, span-buffer initialization, and span-buffer creation and incremental ray-face intersection times, respectively. In CP, T_V denotes preprocessing time for visibility ordering.

As shown in Table 5, T_R and T_S components are almost the same in all algorithms for a fixed visualization instance, thereby verifying the accuracy of our method for dissecting T_T into T_R, T_S, and T_I (and T_V in CP). Although T_R increases with increasing image resolution for a fixed dataset, it always remains negligible in total rendering time T_T.

In Koyamada's algorithm, T_I increases almost linearly with I as expected. This can be explained by the fact that it does not exploit image-space coherency.

In SBRC, cell activation time T_{CA} is directly proportional to number of cells touched during rendering, instead of total number of cells. Therefore, in Table 5, T_{CA} gently increases with increasing resolution for Blunt Fin, Oxygen Post and Delta Wing datasets. However, it remains constant for Combustion Chamber, because all cells are sufficiently large so that all of them are hit by a ray even at the lowest resolution. It is also observed that the rate of increase in $T_{I'}$ with increasing resolution for a fixed dataset is much more pronounced than that of T_{SC}. This is due to the fact that number of spans created is related to the height dimension of the resolution whereas number of intersections is associated to both width and height dimensions. Table 5 also shows that T_{CA} becomes negligible with increasing resolution, and T_{SC} and $T_{I'}$ become the dominating components in T_I. K-SBRC shows similar characteristics as SBRC for different datasets and resolutions.

T_V component of CP is directly affected by two factors, namely, number of faces (and cells) and number of ray-segments generated. The former factor is independent of both viewing parameters and resolution

Table 6
Execution times normalized with respect to those of Koyamada's algorithm (averages of 4 views)

Dataset	Image resolution	Algorithm				
		Koy.	SBRC	K-SBRC		CP
				EA	BBA	
BF	300×300	1.00	0.67	0.65	0.64	0.73
	600×600	1.00	0.48	0.51	0.49	0.46
	900×900	1.00	0.42	0.44	0.44	0.39
CC	300×300	1.00	0.93	1.00	0.96	0.78
	600×600	1.00	0.59	0.64	0.63	0.50
	900×900	1.00	0.49	0.52	0.52	0.43
OP	300×300	1.00	1.03	0.83	0.79	1.67
	600×600	1.00	0.70	0.64	0.62	0.77
	900×900	1.00	0.57	0.55	0.56	0.55
DW	300×300	1.00	1.05	0.89	0.86	1.85*
	600×600	1.00	0.70	0.67	0.66	0.83*
	900×900	1.00	0.57	0.57	0.62	0.59*
Averages						
Avg. of 300×300		1.00	0.92	0.84	0.81	1.26
Avg. of 600×600		1.00	0.62	0.61	0.60	0.64
Avg. of 900×900		1.00	0.51	0.52	0.53	0.49
Overall averages		1.00	0.68	0.66	0.65	0.80

Values with * for CP are averages of views V_1 and V_4 because of the cyclic meshes obtained in other views V_2 and V_3, which cannot be handled by CP.

whereas the latter one depends on both. Therefore, the time spent for generating internal dependencies is constant for a fixed dataset. As shown in Table 5, percent T_V/T_T ratio reduces from 9% at 300×300 resolution to 4% at 900×900 resolution, on the average. Hence, T_V can be considered to be negligible especially at higher resolutions.

Table 6 shows the average relative run-time performances for four views. For each visualization instance, the execution times of the proposed algorithms are normalized with respect to that of Koyamada's algorithm. Then, each value in Table 6 is computed by averaging the normalized execution times of the visualizations of the same dataset with a fixed resolution from four different views. Relative performances of all proposed algorithms with respect to Koyamada's algorithm increase with increasing resolution for each dataset, thereby stressing that the benefit of using image-space coherency is more at high resolutions due to the increase in the projection areas of the cells. This rate of performance increase is much more pronounced in CP for all datasets except Combustion Chamber in which SBRC and K-SBRC achieve slightly larger rate of performance increase, because the number of cells processed by SBRC and K-SBRC does not increase with increasing resolution in Combustion Chamber that has large cells.

As shown in Table 6, for all datasets except Combustion Chamber, K-SBRC performs better than SBRC at low resolutions, but its relative performance decreases with increasing resolution. The reason is that the projection areas of cells increase with increasing resolution, resulting most of the cells to be processed by SBRC. Consequently, it performs worse than SBRC due to the bias-computation overhead. By nature, K-SBRC is expected to give the best payoff in situations where the variation in cell projection areas is large. Table 6 shows that K-SBRC performs considerably better than SBRC at both 300×300 and 600×600 resolutions of Oxygen Post and Delta Wing datasets that have large variation in cell projection areas. On the contrary, K-SBRC performs worse than SBRC for Combustion Chamber dataset that has almost equally sized large cells.

In SBRC, Bounding Box Area scheme always performs better than Exact Area scheme at low resolutions, and this performance difference decreases with increasing resolution so that Exact Area scheme begins to perform better than Bounding Box Area scheme at high resolutions. At low resolutions, the errors introduced by Bounding Box Area scheme do not exaggerate the cell

projection areas enough causing the bias to select SBRC approach erroneously. As resolution increases, these errors cause the bias to select SBRC for cells which should be processed by Koyamada's approach indeed, thus resulting in worse performance.

As shown in Table 6, K-SBRC based on Bounding Box Area scheme (K-SBRC$_{BBA}$) achieves the best run-time performance at 300×300 resolution in all datasets except Combustion Chamber in which CP achieves the best. At 600×600 resolution, CP achieves the best performance for Blunt Fin and Combustion Chamber datasets, but K-SBRC$_{BBA}$ performs still better than CP in Oxygen Post and Delta Wing datasets by taking the advantage of large variation in cell projection areas. At 900×900 resolution, CP outperforms SBRC and K-SBRC in Blunt Fin and Combustion Chamber datasets, however, it cannot beat K-SBRC based on Exact Area scheme (K-SBRC$_{EA}$) in Oxygen Post dataset, and it still performs worse than both SBRC and K-SBRC$_{EA}$ in Delta Wing dataset. When we consider the averages over datasets for different resolutions given at the bottom of Table 6, we see that K-SBRC$_{BBA}$ achieves the best performance both at 300×300 and 600×600 resolutions, and all the proposed algorithms perform nearly the same at 900×900 resolution. Finally, K-SBRC turns out to yield the best performance among the proposed algorithms on the overall average.

8. Conclusion

Three distinct categories, namely image-space, object-space, and hybrid methods, were investigated for fast direct volume rendering of unstructured grids. One of the main objectives was to identify the relative superiority and inferiority of the algorithms in terms of the supported features of these categories both theoretically and experimentally. Various algorithmic features were identified for a relative performance analysis.

Three new and fast algorithms were proposed. The Cell Projection (CP) algorithm, that falls into object-space category, scan converts each face only once, and is capable of rendering non-convex meshes through a simple yet efficient sorting schema that exploits both image and object space coherencies. Existing hybrid methods use object-then-image traversal order which enforces the processing of each cell. Thus, these algorithms perform redundant operations and lack supporting early ray termination. The Span-Buffer Ray-Casting (SBRC) algorithm is a hybrid method proposed to overcome all these deficiencies by changing the computational traversal order from object-then-image to image-then-object without compromising full utilization of image and object space coherencies. The Koyamada-SBRC (K-SBRC) algorithm relies on the observation that the benefit of using image-space coherency for cells with small projection area might be suppressed by the overhead of scan conversion process. It extracts the best features of hybrid and image-space methods by blending SBRC approach with Koyamada's algorithm, which is an efficient image-space algorithm. All proposed algorithms are capable of handling acyclic non-convex meshes and generating images of high accuracy. SBRC and K-SBRC algorithms have the additional capabilities of rendering cyclic meshes and supporting early ray termination.

The proposed algorithms and Koyamada's algorithm were implemented and experimented in a common framework for relative performance analysis. Through experimental results we have concluded the following. Image-space methods are slow in general, but their relative performance is better at low resolutions. Object-space methods are fast and especially at high resolutions their relative performance is very good, but they may perform unexpectedly bad at low resolutions, especially for large datasets. Hybrid methods constitute the most promising category, because they may perform as fast as object-space methods at high resolutions, much faster at low and medium resolutions, and are much more flexible, in general. In hybrid approaches, the proposed image-then-object traversal schema performs better than the currently employed object-then-image schema. K-SBRC, which supports all identified features, appeared to be the fastest algorithm in our experimentations. This verifies the impact of these features on the performance.

As a possible future work, some optimizations could be done to reduce the space complexity of the proposed algorithms by totally avoiding ray lists without compromising the run-time efficiency.

References

[1] Challinger J. Scalable parallel direct volume rendering for nonrectilinear computational grids. Ph.D. thesis, University of California, 1993.

[2] Speray D, Kennon S. Volume probes: interactive data exploration on arbitrary grids. In: Volume visualization (VVS '90), 1990. p. 5–12.

[3] Yagel R. Volume viewing: state of the art survey. In: Visualization '93, Tutorial #9, Course Notes: Volume Visualization Algorithms and Applications, 1993. p. 82–102.

[4] Williams PL. Interactive direct volume rendering of curvilinear and unstructured data. Ph.D. thesis, University of Illinois, Urbana, Champaign, 1992.

[5] Yagel R, Reed D, Law A, Shih P-W, Shareef N. Hardware assisted volume rendering of unstructured grids by incremental slicing. In: IEEE-ACM Volume Visualization Symposium, November 1996. p. 55–62.

[6] Garrity MP. Raytracing irregular volume data. Computer Graphics 1990;24(5):35–40.

[7] Koyamada K. Fast traversal of irregular volumes. In: Kunii TL, editor. Visual computing—integrating computer

graphics with computer vision, 1992. Berlin: Springer, p. 295–312.
[8] Rogers DF. Procedural elements for computer graphics, 2nd ed. New York: McGraw-Hill, 1998.
[9] Bunyk P, Kaufman A, Silva C. Simple, fast, and robust ray casting of irregular grids. In: Proceedings of the Dagstuhl '97—Scientific Visualization Conference, 1997. p. 30–6.
[10] Wilhelms J, Gelder AV, Tarantino P, Gibbs J. Hierarchical and parallelizable direct volume rendering for irregular and multiple grids. In: Proceedings of the IEEE Visualization '96, 1996. p. 57–64.
[11] Hong L, Kaufman AE. Fast projection-based ray-casting algorithm for rendering curvilinear volumes. IEEE Transactions on Visualization and Computer Graphics 1999;5(4):322–32.
[12] Farias R, Mitchell JSB, Silva CT, Wylie B. Time-critical rendering of irregular grids. In: Proceedings of SIBGRAPI 2000, 2000. p. 243–50.
[13] Farias R, Mitchell JSB, Silva CT. Zsweep: An efficient and exact projection algorithm for unstructured volume rendering. In: Proceedings of ACM/IEEE Volume Visualization and Graphics Symposium, 2000. p. 91–9.
[14] Max N, Hanrahan P, Crawfis R. Area and volume coherence for efficient visualization of 3D scalar functions. In: Computer graphics, San Diego Workshop on Volume Visualization, November 1990. p. 27–33.
[15] Williams PL. Visibility ordering meshed polyhedra. ACM Transactions on Graphics 1992;11(2):103–26.
[16] Silva C, Mitchell J, Williams PL. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In: IEEE-ACM Volume Visualization Symposium, 1998. p. 87–94.
[17] Comba J, Klosowski JT, Max N, Mitchel JSB, Silva CT, Williams PL. Fast polyhedral cell sorting for interactive rendering of unstructured grids. Computer Graphics Forum 1999;18(3):367–76.
[18] Stein C, Becker B, Max N. Sorting and hardware assisted rendering for volume visualization. In: Symposium on Volume Visualization, October 1994. p. 83–90.
[19] Newell M, Newell R, Sancha T. A new approach to shaded picture problem. In: Proceedings of the 1972 ACM National Conference, 1972. p. 443–50.
[20] Yagel R. Volume rendering polyhedral grids by incremental slicing. Technical Report OSU-CISRCS-10/93-TR35, Ohio State University, Department of Computer and Information Science, 1993.
[21] Shirley P, Tuchman A. A polygonal approximation to direct scalar volume rendering. Computer Graphics 1990;24(5):63–70.
[22] Wittenbrink C. CellFast: Interactive unstructured volume rendering. In: IEEE Conference on Visualization (Vis '99), Late Breaking Hot Topics, 1999.
[23] Lucas B. A scientific visualization renderer. In: Proceedings of IEEE Visualization '92. Silver Spring, MD: IEEE Computer Society Press, October 1992. p. 227–34.
[24] Koyamada K, Uno S, Doi A, Miyazawa T. Fast volume rendering by polygonal approximation. Journal of Information Processing 1992;15(4):535–44.
[25] Koyamada K, Itoh T. Fast generation of spherical slicing surfaces for irregular volume rendering. The Visual Computer 1995;11(3):167–75.
[26] Cignoni P, Montani C, Puppo E, Scopigno R. Multiresolution representation and visualization of volume data. IEEE Transactions on Visualization and Computer Graphics 1997;3(4):352–69.
[27] Challinger J. Scalable parallel volume raycasting for nonrectilinear computational grids. In: Proceedings of the 1993 Parallel Rendering Symposium. Silver Spring, MD: IEEE Computer Society Press, 1993. p. 81–8.
[28] Giertsen C. Volume visualization of sparse irregular meshes. IEEE Computer Graphics and Applications 1992;12:40–8.
[29] Silva C, Mitchell J, Kaufman A. Fast rendering of irregular grids. In: IEEE-ACM Volume Visualization Symposium, November 1996. p. 15–22.
[30] Silva CT, Mitchell JSB. The lazy sweep ray casting algorithm for rendering irregular grids. IEEE Transactions on Visualization and Computer Graphics 1997;3(2): 142–57.
[31] Westermann R, Ertl T. The VSBUFFER: Visibility ordering of unstructured volume primitives by polygon drawing. In: Proceedings of IEEE Visualization '97, 1997. p. 35–42.
[32] Westermann R, Ertl T. Efficiently using graphics hardware in volume rendering applications. In: ACM Computer Graphics (Proceedings of SIGGRAPH '98), 1998. p. 169–77.
[33] Williams PL. Interactive splatting of rectilinear volumes. In: Proceedings of Visualization'92, Boston, MA, October. 1992.
[34] Ma K. Parallel volume ray-casting for unstructured-grid data on distributed-memory multicomputers. In: Proceedings of 1995 Parallel Rendering Symposium, October 1995. p. 23–30.
[35] Kindlmann G, Durkin J. Semi-automatic generation of transfer functions for direct volume rendering. In: Proceedings of 1998 Symposium on Volume Visualization, 1998. p. 79–86.