# Volume Rendering Method Applied to 3D Edge Impurity Emission in LHD to Produce Projection Image in Arbitrary Plane*)

Yuichi TAMURA, Masahiro KOBAYASHI[1], Taisuke KOBAYASHI[2], Wataru OMORI, Hiroaki NAKAMURA[1], Hiroaki OHTANI[1], Susumu FUJIWARA[3] and the LHD Experimental Group[1]

*Konan University, Kobe 658-8501, Japan*
[1]*National Institute for Fusion Science, Toki 509-5292, Japan*
[2]*SOKENDAI (The Graduate University for Advanced Studies), Toki 509-5292, Japan*
[3]*Kyoto Institute of Technology, Kyoto 606-8585, Japan*

(Received 27 December 2018 / Accepted 20 March 2019)

Understanding edge impurity transport is one of the important issues for fusion devices to control edge radiation distribution for detachment operation and impurity influx to the confinement region. In LHD, the edge magnetic field structure becomes complex stochastic magnetic field. In order to study relation between impurity transport and the magnetic field geometry, 3D edge impurity emission distributions are obtained by a multichannel spectrometer system and tomography scheme. However, it is difficult to understand the three-dimensional (3-D) structure. Therefore, we propose a visualization system that employs a volume rendering method. With the proposed system, which can be used on a PC or mobile device, the user can observe a 3D structure in an arbitrary plane. To realize this function, we propose a volume visualization system comprising preprocessing and real-time rendering stages. Therefore, the visualization framerate can exceed 30 frames per second on PCs and approximately six frames per second on mobile devices, although the user frequently changes the position and direction of the camera.

Keywords: visualization, volume rendering, ray marching, LHD, Large Helical Device, emission distribution, virtual reality

DOI: 10.1585/pfr.14.3406084

## 1. Introduction

Understanding of the edge impurity transport is one of the important issues for fusion devices in order to control edge radiation distribution for detachment operation and impurity influx to the confinement region [1]. In LHD, the edge magnetic field structure becomes complex stochastic magnetic field, where different connection length flux tubes co-exist. In order to study relation between impurity transport and the magnetic field geometry, 3D edge impurity emission distributions are obtained by a multichannel spectrometer system and tomography scheme [2, 3]. This tomography scheme is powerful to observe physical phenomena in LHD though output data is not structured and it is difficult to visualize 3D emission distribution image directly. Therefore, it is necessary to develop a visualization tool that can read and visualize 3D emission distribution automatically and allow the user to intuitively observe 3D structure.

In this paper, we propose a system that can regenerate an emission distribution image projected toward arbitrary direction in LHD using a volume rendering method. For example, we can visualize the emission distribution from a particular port of the LHD vessel or virtual port. The large-scale virtual reality system CompleXcope [4] and its applications have been developed in NIFS. However, it is difficult to use this system frequently and freely. Thus, we propose volume visualization method and system to observe the 3D structure of emission distribution from an arbitrary direction on various devices.

## 2. Method

Volume rendering methods are widely used to visualize volume data, and the most common technique is the ray casting method [5]. This method can render high quality images of volume data; however, it requires powerful computer resources and has difficulty rendering in real time. Computer performance and image processing methods have improved [6, 7], and it is now possible to render 3D objects in real time using a high-performance PC. However, rendering with mobile devices, such as iPads, Android tablets, and PC tablet, remains difficult. Thus, we propose a volume rendering method that uses a 3D texture, which realizes near-real-time rendering on mobile devices. Figure 1 shows the procedure of the proposed method.

*author's e-mail: tamura@konan-u.ac.jp*

*) This article is based on the presentation at the 27th International Toki Conference (ITC27) & the 13th Asia Pacific Plasma Theory Conference (APPTC2018).

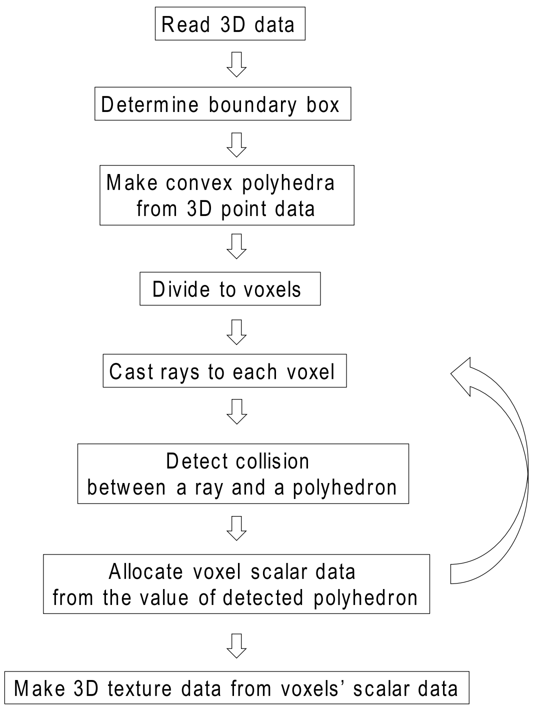
Read 3D data
Determine boundary box
Make convex polyhedra from 3D point data
Divide to voxels
Cast rays to each voxel
Detect collision between a ray and a polyhedron
Allocate voxel scalar data from the value of detected polyhedron
Make 3D texture data from voxels' scalar data

Fig. 1 Processing flowchart of the proposed visualization system.

Table 1 An example of emission distribution data file.

| x | y | z | |
|---|---|---|---|
| −1.8836 | 3.9077 | 0.1973 | Vertex data |
| −1.8229 | 3.9121 | 0.20196 | |
| −1.8794 | 3.9144 | 0.19558 | |
| −1.8187 | 3.9185 | 0.19989 | |
| −1.9069 | 3.9365 | 0.2784 | |
| −1.8492 | 3.938 | 0.2863 | |
| −1.9121 | 3.9281 | 0.28055 | |
| −1.8544 | 3.93 | 0.28889 | |
| 1.328 | 0 | 0 | Scalar data |
| ⋮ | ⋮ | ⋮ | |

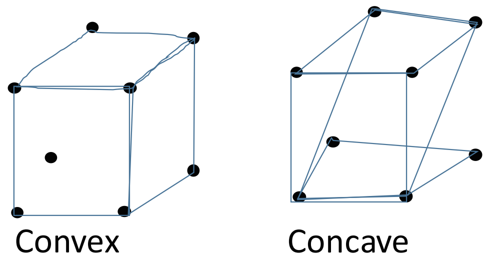

Fig. 2 Example polyhedron. Left: concave polyhedron; right: convex polyhedron.

The proposed system comprises preprocessing and real-time rendering stages. Notably, it is difficult to create 3D images from large volumetric data and simultaneously render them in real time. The proposed method creates a 3D texture meaning volumetric texture data in the preprocessing stage, stores the data in a local storage, and renders a 3D volume image in the real-time rendering stage.

## 2.1 Preprocessing stage

First, input data are read from the storage device, and then emission distribution [2] data and their position data are read. An example of this data is shown in Table 1. This data file consists of positions of vertices and scalar data (emission distribution data) in volumes composed of these vertices. The emission distribution data comprises 8 or 6 vertices and scalar data.

From the input data, a boundary box is determined from the position range of the position data of vertices.

The next step is to create convex polyhedra from the original data. Generally, 3D data comprises polyhedra and their vertices data. Notably, surface data may be provided to 3D model data. If surface data are not provided, various polygonal patterns can be considered. Figure 2 shows an example, where both images comprise vertices and six surfaces. In this case, the right object is incorrect; however, this incorrect object can be generated if the surface data are not provided. In this system, a polyhedron, which comprises eight vertices and has largest volume, is automatically determined. Therefore, correct polyhedron can be defined if the input data is not aligned.

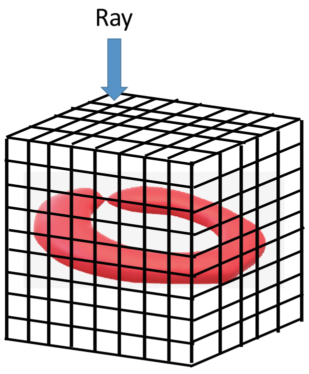

Fig. 3 Casting ray onto each polyhedron.

The main step in the preprocessing stage is to cast rays onto each polyhedron, detect collisions, and assign scalar data for rendering the image. To achieve high-speed calculation using a graphic processing device, it is necessary to create a volume comprising a cube. Conversely, polyhedra from the input data are not cuboids; therefore, the boundary box is divided into cuboids (hereinafter, this is called voxel). Herein, to avoid data loss, the division size of the boundary box must be sufficiently small relative to the polyhedron size of the original input data.

The next step is to cast rays from outside the boundary box. Herein, all polyhedra are present in the boundary box. Consider a ray cast through one of the sides of the bounding box passing through the center of the voxels. Figure 3 shows an example of this process.

Table 2 Algorithm to apply scalar data from input data to constructed voxel.

| **T(x, y, z)** | |
|---|---|
| 1: | object_number = DetectCollision(ray(x, y, z)) |
| 2: | scalar_value = ExtractScalarValue(object_number) |
| 3: | T(x, y, z) = SetValue(x, y, z, scalar_value) |
| 4: | T(x, y, z) = (T(x, y, z) - scalar_min)(scalar_max - scalar_min) |
| 5: | **repeat** (while x < l, y < m, z < n) |

Table 3 Algorithm to apply colormap to voxels.

| **T(x, y, z)_r, g, b** | |
|---|---|
| 1: | if(T(x, y, z) < 0.5) |
| 1-1: | T(x, y, z)_r = 0, T(x, y, z)_g = 2 * T(x, y, z), T(x, y, z)_b = -2 * T(x, y, z) + 1 |
| 2: | else |
| 2-2: | T(x, y, z)_r = 2 * T(x, y, z) - 1, T(x, y, z)_g = -2 * T(x, y, z) + 2, T(x, y, z)_b = 0 |
| 3: | **repeat** (while x < l, y < m, z < n) |

When a collision between a ray and polyhedron occurs, the scalar value of a polyhedron is assigned to the voxel at the collided position. Additionally, the collision calculation between a ray and a polyhedron continues until a ray leaves the boundary box. If the boundary box comprises $l \times m \times n$ voxels, $l \times m$ rays are cast from outside the boundary box, and the collision test is performed $l \times m \times n$ times. In this study, the division size $(l, m, n)$ is manually determined and native voxelization approach is used.

Finally, 3D texture data are created from the scalar data of the voxels and stored in an array $T(x, y, z)$ of size $l \times m \times n$. Additionally, all scalar data of the voxels are normalized between 0 and 1. The algorithm for this procedure is shown in Table 2.

### 2.2 Real-time visualization stage

At this stage, the system reads the 3D texture data from the storage device and applies a colormap to visualize the 3D data using the algorithm shown in Table 3. This color map is consistent with the color map of the visualized experimental data on the left side of Figs. 5 - 7. The 3D texture data created in the preprocessing stage are stored in ASCII data format. Notably, the preprocessing requires significant computer power and time to create 3D texture data; thus, a high-performance PC is necessary. If the preprocessing and visualization computers are different, the data must be sent to the visualization device. This visualization application is available for PC, Android devices, and iOS devices. Real-time visualization methods on mobile device have been proposed [8]. In this study, we use 3D-textured slicing approach for visualization.

## 3. Results

### 3.1 Software environment

The proposed system is developed using the Unity 3D engine[1], which was originally created for game development. However, the Unity 3D engine has many useful functions, including functions that facilitate scientific visualization. This engine supports various devices and platforms; therefore, the proposed system can be implemented on various devices, such as PCs and tablet devices.

[1]http://unity3d.com/

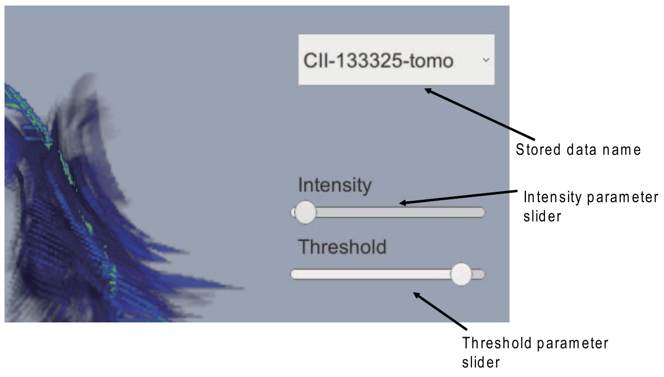

Fig. 4 Interface to change visualization parameters.

The system implements an interactive interface that changes parameters as shown in Fig. 4. The "intensity" parameter can change the weight parameter for visualization. The value of each pixel is determined using Eq. (1).

$$I(u, v) = \sum_{k=1}^{n} wT(x, y, z), \tag{1}$$

where $I(u, v)$ is the calculated value of the pixel color at $(u, v)$ on a projection plane, n is the number through which a ray penetrates voxels from the eye position to outside the boundary box, and k is the $k - th$ voxel to penetrate. $T(x, y, z)$ is a normalized scalar value on a projection plane of the rendering computed using the algorithm in Tables 2 and 3. If color map is applied, the output image will be blue if the "intensity" parameter is too small.

The "threshold" parameter is the cut-off value. If the normalized scalar value of a voxel is less than the threshold value, its value is ignored in Eq. (1). The threshold param-

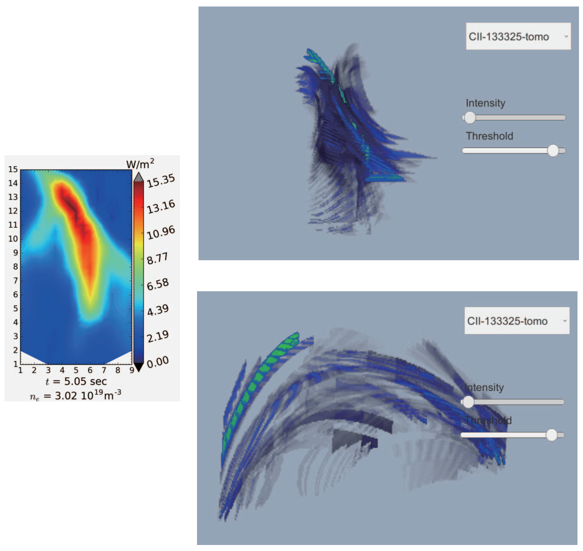

Fig. 5 Left: experimental result of carbon $C^{1+}$ emission (CII) distribution. Right: Front and Side views of the visualization result (intensity parameter = 20; threshold = 0.0 (all data are represented)).

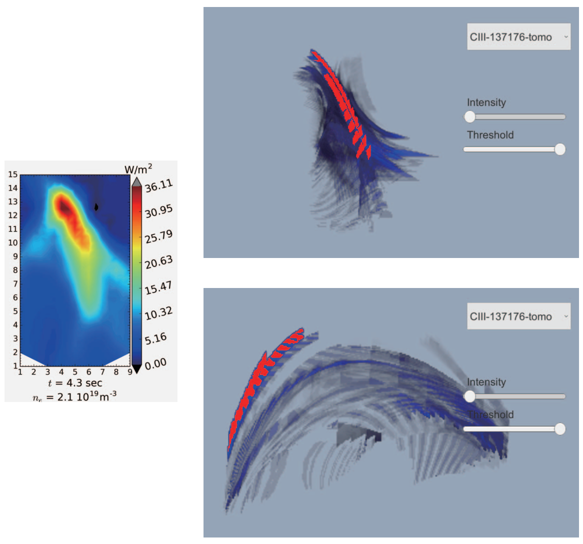

Fig. 6 Left: experimental result of carbon $C^{2+}$ emission (CIII) distribution. Right: Front and Side views of visualization result (intensity parameter = 20 threshold = 0.0 (all data are represented)).

eter is used if the user wants to exclude values that are less than the given threshold value in the visualization.

The proposed system supports touch interfaces; thus, the user can interact with the display and change the position, angles, and parameters intuitively. Therefore, the visualization results can be observed on an arbitrary plane.

## 3.2 Examples of visualization results

Figures 5 - 7 show visualization examples, where the images on the left are the experimental results, and images on the right are the visualization results obtained by

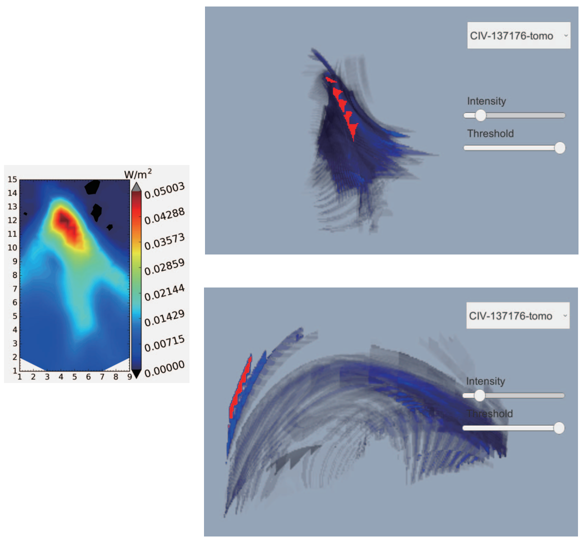

Fig. 7 Left: experimental result of carbon $C^{3+}$ emission (CIV) distribution. Right: Front and Side views of visualization result (intensity parameter = 20; threshold = 0.0 (all data are represented)).

Table 4 3D texture generation time comparison based on 3D texture size.

| number of pixels (pixel) | processing time (second) |
|---|---|
| $100 \times 100 \times 100$ | 3.8 |
| $200 \times 200 \times 200$ | 28.7 |
| $400 \times 400 \times 400$ | 224.3 |

the proposed system. Here the resolution of the 3D texture is $400 \times 400 \times 400$ pixels. The original data comprise approximately 25,000 polyhedra.

Since the colormap of 3D data is consistent with the experimental result, the user can compare physical phenomena intuitively.

## 3.3 Speed estimation

We measured the time required to create a 3D texture, read the 3D texture data, and start the 3D view. We also measure the rendering frame rate. Table 4 shows the measurement results for 3D texture generation time with various volume sizes.

This test was executed with an Intel Core i7-4770 (3.4 GHz, 8 MB cash memories), an NVIDIA GeForce GTX 1070 GPU (graphics clock is 1506 MHz and processor clock is 1683 MHz.), 32 GB main memories and a solid state disk for data storage, on Windows 10. The memory usage of the largest case ($400 \times 400 \times 400$) is approximately 492 MB. The average time required to read the 3D texture data was approximately 6.8 s. The average rate to render the 3D data on the PC was greater than 30 frames per second (fps), and that with an Android device (Snapdragon Kryo 280 CPU) was approximately about 5 - 10 fps.

In this study, the division size is manually determined. If the edge length of the voxel is shorter than the minimum edge of the input polyhedron, at least one voxel is included in each polyhedron. This means all input data is completely visualized without loss. For this data, if division size is ($400 \times 400 \times 400$), 99.1% edge of the input polyhedron is longer than the edge of voxel. If the size is ($200 \times 200 \times 200$), it is 95.6%. Also, if the size is ($100 \times 100 \times 100$), it is 85.3%.

## 4. Conclusion

In this paper, we proposed a system to visualize unstructured emission distribution data in two stages. In the first stage, the unstructured data is transformed into 3D texture data, and in the second stage, the transformed 3D texture data is read out from the storage and visualized in real time. The advantage of the proposed system is that it can be used on handheld devices, such as smartphones. It is difficult to render 3D volume data using a volume rendering method; however, the proposed system is a two-stage visualization system that involves preprocessing and real-time rendering stages.

With this system, the visualization results of emission distribution can be confirmed on PCs, tablet devices (e.g., iPad), and CompleXcope. The visualization results of the proposed system can be used to further develop diagnostic systems in LHD. In future works, it is necessary to determine the division size of voxels automatically.

## Acknowledgment

This study was funded partly by a Grant-in-Aid for Scientific Research KAKENHI (15K06650, 17K06280, 16H04622) and MEXT, Japan.

[1] M. Kobayashi *et al.*, Nucl. Fusion **55**, 104021 (2015).
[2] M. Kobayashi *et al.*, Rev. Sci. Instrum. **88**, 033501 (2017).
[3] T. Kobayashi *et al.*, Rev. Sci. Instrum. **89**, 123502 (2018).
[4] A. Kageyama *et al.*, Proc. 16th International Conference on the Numerical Simulation of Plasmas (1998).
[5] M. Levoy, IEEE Comput. Graph. Appl. **8**, 03 (1988).
[6] J. Kruger *et al.*, Proc. 14th IEEE Visualization (2003).
[7] P. Lacroute *et al.*, Proc. SIGGRAPH '94 (1994).
[8] J.M. Noguera *et al.*, IEEE Trans. Vis. Comput. Graphics (2016).