Improvement of Particle-based Volume Rendering for Visualizing Irregular Volume Datasets

Naohisa Sakamoto[1], Takuma Kawamura[2], Koji Koyamada[1], and Kazunori Nozaki[3]

[1] Center for the Promotion of Excellence in Higher Education, Kyoto University [2] Graduate School of Engineering, Kyoto University, [1, 2] Kyoto 606-8501, JAPAN, [3] Cyber Media Center, Osaka University, Osaka 567-0047, Japan

e-mail address: Naohisa Sakamoto: naohisas@viz.media.kyoto-u.ac.jp, Takuma Kawamura: kawamura@viz.media.kyoto-u.ac.jp, Koji Koyamada: koyamada@mbox.kudpc.kyoto-u.ac.jp, Kazunori Nozaki: nozaki@cmc.osaka-u.ac.jp

Corresponding author: Naohisa Sakamoto,naohisas@mbox.kudpc.kyoto-u.ac.jp, TEL : +81-75-753-9365, FAX : +81-75-753-9365

Abstract We present a technique for previewing large-scale irregular volume datasets using an improved particle-based volume rendering (PBVR) technique. Volume rendering of irregular grid volume data is considerably more complicated than that of regular grid data, since the sampling and compositing processes, which must be done in visibility order, are not straightforward. In our original PBVR, rendering involves generating and projecting sub-pixel-size, opaque, and emissive particles without visibility ordering. To make it easier to preview large-scale irregular volume datasets, we improve our original PBVR technique in two respects. The first is that we exploit its scalability to develop a cell-bycell particle generation technique. The second is that we reduce the memory cost of the frame buffer using a pixel-by-pixel superimposing technique. To measure the effectiveness of our proposed method, we apply it to huge irregular volume datasets composed of 71 mega hexahedral cells or 1 giga tetrahedral cells.

Keywords volume rendering, large-scale irregular volume datasets, tetrahedral cells, hexahedral cells.

1 INTRODUCTION

Many volume rendering methods have been developed over the past two decades. In fact, the development of techniques for irregular volumes remains a challenging area in the visualization community. Irregular volume datasets consist mainly of scalar data defined on collections of irregularly ordered cells whose shapes are not necessarily orthogonal cubic. Data of this type can be found in the results of the finite element method (FEM) technique, which is widely used in computational mechanics. It is also becoming popular in computational fluid dynamics (CFD).

High performance computing calculates a huge FEM model, which cannot reside in a single computational node and is distributed to multiple computational nodes. A user must be able to do a fast preview in order to evaluate whether the computation was successful. For previewing a volume dataset, a volume rendering technique is suitable, since it lets the user visualize the whole volume space. However, splatting requires visibility sorting, in which all volume cells need to be processed in advance at each viewing point. Our particle-based volume rendering (PBVR) technique is suitable for previewing a huge irregular volume dataset, since it requires no visibility sorting. The technique represents a given volume dataset as a set of particles that are emissive and opaque, based on a particle model derived from Sabella's density emitter model [1]. Once a volume dataset is converted to a set of particles, the rendering process is efficient, regardless of the visibility order.

Scalability in the number of the volume cells is crucial to successful volume rendering of huge irregular volume datasets. We

were therefore obliged to modify our original PBVR so that it could process an irregular volume dataset cell-by-cell. In this paper, we describe an improved PBVR that make it easier to preview a huge irregular volume dataset. One improvement is that it generates particles cell-by-cell using a particle density estimation technique. The other improvement is that it relies not on the sub-pixel processing technique, which requires a large frame buffer, but rather on a pixel-superimposing technique.

The paper is organized as follows. First, we describe existing techniques for rendering irregular volumes. Second, we review our original PBVR technique. Third, we describe our improved PBVR technique for rendering huge irregular volumes. To show the effectiveness of this new rendering method, we use it to render huge volume data sets composed of one giga tetrahedral cells and 71 mega hexahedral cells.

2 RELATED WORK

Koyamada [2] implemented irregular volume rendering techniques using the image-order approach, that is, ray casting. The ray-casting approach generally takes a lot of CPU time, although it can produce a high-quality image. Garrity [3] proposed a similar technique. The most computationally intensive steps of the ray-casting rendering are ray-cell intersection and integration of the samples. In order to alleviate these two computational bottlenecks, Koyamada et al. [4] developed a new cell traversal method that reduces the number of intersection tests by using image coherence, and it interpolates data efficiently along a ray by taking advantage of the characteristics of tetrahedral cells.

After Shirley et al. [5] proposed a projected tetrahedra (PT) algorithm for generating a volume-rendered image using tetrahedral cells, the mainstream of irregular volume-rendering techniques has adopted the object-order approach. In their algorithm, each tetrahedral cell is projected onto the screen in visibility order, from back to front, in order to build up a semitransparent image. The work of Williams [6] sparked a considerable amount of research on visibility ordering techniques. Lucas [7] proposed a projection algorithm for irregular volume datasets based on cell faces. This algorithm sorts all the faces of the volume cells from back to front, then scan-converts each face in turn by linearly interpolating the vertex luminosity, opacity, and *z*-values across the face. This algorithm can handle complex polyhedra more easily than can other projection algorithms.

Koyamada et al. [4] proposed a volume-rendering algorithm that accumulates parallel layers of partially transparent triangles perpendicular to viewing rays. The layers form a set of concentric spherical slicing surfaces around a viewing point. In order to extract spherical slicing surfaces efficiently, Koyamada et al. [8] took advantage of the coherence between consecutive slicing surfaces.

Meredith and Ma [9] developed a technique for approximating irregular volumes as an octree structure and for rendering the octree using hardware-assisted splats. Although they found that when the octree nodes are traversed back-to-front, the difference between sorting and not sorting within octree nodes is hardly noticeable in the rendered image, their technique

requires the visibility sorting.. Wylie et al. [10] proposed hardware-accelerated methods to improve the performance of the PT algorithm, but even with current graphics hardware, no more than approximately 490K tetrahedra are possible (timings do not include sorting). Roettger et al. [11] proposed an algorithm without visibility ordering for irregular volume cells. Since their optical model considers only emission, it can be applied only to the visualization of gaseous phenomena, such as fire and ground fog. Csebfalvi proposed a sort-less volume-rendering technique [12][13] that can be categorized as X-ray volume rendering, because his optical model considers only absorption.

Callahan et al. [14] developed an integrated visibility ordering technique, called HAVS, in which the centroid of the cell faces are first sorted in order to make a rough visibility ordering, and pixel fragments generated from rasterized faces are used along with the *k*-buffer in order to increase accuracy. Although HAVS has achieved 1.3 fps for 1.4 mega tetrahedra, it generates some artifacts when the *k*-buffer is not large enough. It is difficult to determine the optimal value for *k*. The memory space needed to sort cell faces is about twice that required for tetrahedral cells.

In 2005, a particle-based parallel visualization method for regular volumes has been proposed by Liang et al. [15]. In this method, a frame rate of 9 fps and 1 fps have respectively achieved for 256^3 (16.8 mega particles) and 512^3 (134.2 mega particles) using a PC cluster system composed of 7 render nodes and a display node. On the other hand, Anderson et al. [16] proposed a point-based technique to render irregular volumes. They represent a tetrahedral cell as point primitives at the center, followed by rendering and compositing of the represented point primitives. Like HAVS, this technique requires sorting of all point primitives. Although this technique has achieved 5.3 fps for 1.4 mega tetrahedra and 0.3 fps for 6.3 mega tetrahedra, there may be artifacts in the rendered image when point primitives are rasterized as screen-aligned squares. These techniques require the visibility ordering.

Recently, Muigg et al. [17] proposed a hybrid ray-casting technique for irregular volumes. Their technique can process a 15 mega-tetrahedra volume in 22 seconds. As we pointed out [8], although this ray-casting approach may be promising especially for large volume datasets, it may miss contributions from important small-volume cells, which are smaller than a pixel. PBVR generates particles in a cell whose size is smaller than a pixel if a large opacity is specified. Since the size of the particle is dependent not on the cell size but on the pixel size, a small-than-a-pixel cell can contribute to a pixel value.

Roettger et al. [11] commented that with increases in the rendering speed of graphics accelerators, the memory bandwidth consumed by visibility sorting becomes the limiting factor. They speculated that a significant performance bump beyond the abovementioned limit of 1.5 mega tetrahedra per second is possible only with a significant leap in the processing capability of graphics accelerators or through the use of special-purpose hardware. We agree with them, and we have devised a stochastic approach to rendering irregular-grid volume data based on a particle model. In [18], we present the basic idea for this approach, called the PBVR technique, which does not require visibility sorting for irregular volume cells, and which takes into account

both emission and absorption.

3 ORIGINAL PBVR TECHNIQUE

3.1 Overview of Volume Rendering

In a volume ray-casting algorithm, a viewing ray is subdivided into n ray segments in which the particle luminosity and the density in the k^{th} ray segment can be regarded as constants, C_k and α_k . The brightness can be calculated as:

$$B_0 = \sum_{i=1}^{n} c_i \times (\alpha_i \prod_{j=1}^{i-1} (1-\alpha_j)) \cdot \quad (1)$$

In the density emitter model, the opacity is described as follows:

$$\alpha = 1 - e^{-\rho' \Delta t} . \quad (2)$$

Here, ρ' describes the density of particles in a unit length, and expresses the number of particles in a ray segment. Thus, ρ' represents the probability that the segment contains some particles, since this model assumes that the number of particles follows the Poisson distribution.

The computational complexity of Equation 1 is $O(n^2)$. If we consider an intermediate brightness value B_k in the interval $[t_k, t_n]$, we can obtain the following recurrence formula for the back-to-front accumulation:

$$B_{k-1} = c_k \alpha_k + (1-\alpha_k) B_k . \quad (3)$$

This shows that the computational complexity of the algorithm can be reduced to $O(n)$ if the sampling points are sorted in visibility order. Most volume-rendering techniques use back-to-front accumulation. Although visibility sorting has improved the performance of the brightness calculation, unfortunately it can become a bottleneck when a large irregular volume dataset is rendered.

3.2 Overview of the PBVR technique

To develop a volume-rendering technique that does not require visibility sorting, we represent a 3-D scalar field as a set of emissive and opaque particles such that no alpha blending is needed during the rendering calculation. Instead, only depth comparison is needed. This approach has advantages for distributed processing. The particle density is derived from a user-specified transfer function, which converts a scalar data value to an opacity data value. We assume that the density describes the probability that a particle is present at a given point.

Our original PBVR is composed of three phases: particle generation, particle projection, and sub-pixel processing. The first phase is to construct a probability function from the volume data and to generate particles according to the function. The second

phase is to project particles onto an image plane. The third phase is to divide a single pixel into multiple sub-pixels so that a particle is stored as precisely as possible, and so that averaging sub-pixel values yields a final brightness value [19].

Figure 1 (Color)

During the phase of particle generation, we used the Metropolis algorithm [20], which is an efficient Monte Carlo technique widely used in chemistry and physics. Since the algorithm is applied to all the volume data in the original PBVR technique, we have to keep all the data in main memory, which will become a bottleneck when we handle a huge volume dataset. The Metropolis algorithm uses the ratio of the energy of the current position of a particle to that of the next candidate position to generate particles. Although our original PBVR regarded the particle density described in Equation 2 as the energy of the position, it did not use the particle density as-is. This forced the user to specify a total number of particles, after which the particles were distributed according to the energy field. But this freedom in specifying the total number of particles makes it difficult to generate a unique image from a given transfer function. The more particles are generated, the less transparent the resulting image becomes. The larger the sub-pixel level that is specified, the more transparent the resulting image becomes (see Figure 1). A technique is needed for estimating a particle density field, which is integrated to yield the number of particles. Note that the particle density describes the number of particles in a unit length in Equation 2.

In sub-pixel processing, a pixel is divided into sub-domains (sub-pixels). Doing particle occlusion for each sub-pixel makes the pixel store the nearest particles, and affects the averaging procedure for determining the pixel value. When carrying out sub-pixel processing, it is necessary to store multiple particles for each of the pixels. Therefore, we allocated memory for storing these particles on the image plane. This allocation increases the memory cost relative to that in normal resolution by the square of the sub-pixel level.

4 THE IMPROVED PBVR TECHNIQUE

To solve the problems described in the previous section, we improve our original PBVR technique in two ways. One is that we exploit its scalability by developing a cell-by-cell particle generation technique that estimates the particle density in a given irregular volume. The pseudo-code for the algorithm is as follows:

Algorithm 1: `ImprovedPBVR_wo_PixelSuperimposing ()`

```
1:    // User-specified parameters.
2:    Volume data V;
```

```
3:    Sub-pixel level l;
4:    Transfer function TF;
5:
6:    for each frame do
7:      GetCameraData();
8:      Particle radius r = CalculateRadius(l);
9:
10:     // Cell-by-cell particle generation.
11:     m = CellByCellParticleGeneration(V,TF,r);
12:
13:     // Sub-pixel processing.
14:     for each pixel p_i do
15:       col_i = AverageSubPixelValues(p_i,m,l);
16:     end for
17:   end for
```

The other improvement is that we reduce the memory cost of the frame buffer using a pixel superimposing technique. The pseudo- code is as follows:

Algorithm 2: ImprovedPBVR_w_PixelSuperimposing ()

```
1:    // User-specified parameters.
2:    Volume data V;
3:    Sub-pixel level l;
4:    Transfer function TF;
5:
6:    for each frame do
7:      GetCameraData();
8:      // Calculate the particle radius as the
9:      // sub-pixel 1.
10:     Particle radius r = CalculateRadius(1);
11:
12:     // Pixel superimposing.
```

```
13:     col = PixelSuperimposing(V,TF,r,l);
14:
15:     // Sub-pixel processing as the sub-pixel
16:     // level 1.
17:     for each pixel p_i do
18:       col_i = AveragePixelValues(p_i,col_i);
19:     end for
20:   end for
```

4.1 Cell-by-cell Particle Generation

In this subsection, we describe a technique for generating particles cell-by-cell that does not require all the volume cells to be in main memory. To generate particles in a volume cell, we need to estimate the particle density of the volume data. In our improved particle modeling approach, in order to use a density that represents the number of particles in a unit volume, we replace ρ with $\pi r^2 \rho$, a projection area multiplied by the density of particles in a unit volume. Now the opacity can be expressed as:

$$\alpha = 1 - \exp(-\pi r^2 \rho \Delta t) . \quad (4)$$

Here, *r* represents the radius of a particle and delta *t* indicates ray segment length.

In our improved particle model, we consider three attributes of particles: shape, size, and density. The particle shape is assumed to be a sphere because its projection is a view-independent shape, i.e., a circle. The size of the sphere is determined by the radius, which in order to facilitate sub-pixel processing we assume to be the pixel side length divided by an even number. We describe the even number divided by double the sub-pixel level (*level*) as follows:

$$r = \frac{1}{2 \cdot level} . \quad (5)$$

The particle density can be estimated from the radius, an opacity value in the user-specified transfer function, and the ray-segment length used in ray-casting. From Equation 4, we have:

$$\rho = \frac{-\log(1-\alpha)}{\pi r^2 \Delta t} . \quad (6)$$

In order to use PBVR to generate an image equivalent to the volume ray-casting result, we should use the above relation to estimate the particle density function. From Equation 6, we see that the number of particles generated quadruples for each doubling of the sub-pixel level.

By assuming a hardcore-point process in the particle distribution, the density function ρ should have a maximum $\rho^{\max}$. Since the volume of the enclosing cube of the particle is $8r^3$,

$$\rho^{\max} = \frac{1}{8r^3}. \tag{7}$$

Thus, the opacity value α has a maximum $\alpha^{\max}$. If the opacity value α is between $\alpha^{\max}$ and 1.0, the relevant density function ρ becomes a constant value, $\rho^{\max}$. Here, $\alpha^{\max} = 1 - \exp(-\pi r^2 \rho^{\max} \Delta t)$. We can calculate the particle density distribution from the opacity distribution, which can be derived using the user-specified transfer function. Thus, the number of particles, *N*, in a volume cell is calculated as

$$N = \int_{Cell} \rho dV. \tag{8}$$

When the number of particles in Equation 8 is not an integer, the final number *n* is determined as follows:

$$n = \begin{cases} \lfloor N \rfloor + 1 & \text{if } R \le N - \lfloor N \rfloor \\ \lfloor N \rfloor & \text{otherwise} \end{cases}, \tag{9}$$

where *R* is a uniform random number [0,1). Particles can be generated cell-by-cell, at each tetrahedral or hexahedral cell. The locations of the particles can be calculated stochastically in the local coordinate system by using the metropolis method [20]. And then, the locations of the particles in the global coordinate system can be obtained by using a following equation;

$$\mathbf{X} = \sum_{i=1}^{p} N_i(\mathbf{x}) \mathbf{X}_i^{vertex}. \tag{10}$$

Here, X is the global coordinate corresponding to **x** in the local coordinate, and $\mathbf{X}_i^{vertex}$ are the coordinates of cell vertices. $N_i(\mathbf{x})$ are shape functions for each vertex and *p* represents the number of vertices.

If memory to store all the particles is not available in the main memory, we transfer particles to the GPU at each volume cell for rendering the particles, which we call a particle-streaming mode in Algorithm 3. The pseudo-code for the cell-by-cell particle generation technique is as follows:

Algorithm 3: CellByCellParticleGeneration (*V*, *TF*, *r*)

```
1:    for each volume cell C_i ∈ V do
2:      ρ_i = C_i.calculateDensity(TF, r);
3:      N_pc = C_i.getNumOfParticles(ρ_i);
4:      for each j=1,2,..., N_pc do
5:        M_ij = ParticleGeneration(C_i);
```

```
6:        if ParticleStreamingMode is true then
7:          m_ij = ParticleProjection(M_ij);
8:        end if
9:      end for
10:   end for
11:
12:   if ParticleStreamingMode is false then
13:     N_p = GetNumOfGeneratedParticles();
14:     for each i=1,2,...,N_p do
15:       m_i = ParticleProjection(M_i);
16:     end for
17:   end if
18:
19:   return m;
```

4.2 Pixel Superimposing

In this subsection, we show that in sub-pixel processing, the deviation of the final brightness value decreases with respect to the sub-pixel level. Sub-pixel processing can be implemented by superimposing pixel values.

To evaluate the average and the deviation of the pixel value with respect to the sub-pixel level, a simple experiment has been conducted. In the experiment, the brightness value is calculated at a single pixel by randomly distributing a set of particles in a square prism with side length 1.0 and height Δt. Here, we also assume that particle luminosity is constant: $c=1$. Given that α is an opacity value defined in Equation 4, the pixel value becomes 1 with a probability of α or 0 with a probability of $(1-\alpha)$. Thus, the average brightness value of a sub-pixel can be calculated as:

$$B_{Ave.} = 1 \cdot \alpha + 0 \cdot (1-\alpha) = \alpha \tag{11}$$

The variance can be calculated as:

$$B_{Var.} = (1-\alpha)^2 \alpha + (0-\alpha)^2 (1-\alpha) = (1-\alpha)\alpha \tag{12}$$

Thus, the deviation is the square root of the variance, that is $\sqrt{(1-\alpha)\alpha}$.

$$B_{Dev.} = \sqrt{(1-\alpha)\alpha} \tag{13}$$

The total brightness is calculated by averaging all the sub-pixel values contained in a single pixel.

Next, we analyze statistical information about total brightness theoretically. First, we define the brightness value of the i^{th} sub-pixel as B_i. Then, if the occurrence of particles at each sub-pixel is independent of other sub-pixels, the average and variance of the brightness value of a sub-pixel are calculated as:

$$\begin{aligned} B_{Ave.} &= B^i{}_{Ave.} &&= E(B^i) &&= \alpha \\ B_{Var.} &= B^i{}_{Var.} &&= Var(B^i) &&= (1-\alpha)\alpha \end{aligned} \tag{14}$$

The total brightness value is calculated by averaging brightness values in all the sub-pixels:

$$B^{total}(level) = \sum_{i=1}^{level^2} \frac{B^i}{level^2} \tag{15}$$

The variance of the total brightness can be calculated as follows:

$$\begin{aligned} B^{total}_{Var.}(level) &= Var\left(\sum_{i=1}^{level^2} \frac{B^i}{level^2} \right) \\ &= \frac{1}{level^4} Var\left(\sum_{i=1}^{level^2} B^i \right) \\ &= \frac{1}{level^4} \left\{ \sum_{i=1}^{level^2} Var\left(B^i\right) + 2 \sum_{i,j:i<j} Cov(B^i, B^j) \right\} \end{aligned}$$

Since the brightness values of sub-pixels are independent of each other, the covariance term $Cov(B^i, B^j)$ can be estimated as zero. This makes the variance decrease to the inverse of the number ($level^2$) of sub-pixels. Therefore, the deviation of the total brightness can be calculated as:

$$B^{total}_{Dev.}(level) = \frac{B_{Dev.}}{level} = \frac{\sqrt{(1-\alpha)\alpha}}{level}$$

When we fix the sub-pixel level at 1, and repeat the random distribution of particles $level^2$ times, the above formula for the deviation is valid for averaging pixel values over all the iterations if we can assume that the particle distribution is independent at each iteration. Thus, we can achieve the same effect as when we carry out sub-pixel processing by averaging, a total of $level^2$ times, a set of particles that are generated with sub-pixel level of 1. Figure 2 shows the result of error evaluation for the sub-pixel processing and the pixel superimposing in a single pixel. From this figure, we can confirm that the pixel superimposing is essentially equivalent to the sub-pixel processing. The pseudo code for the pixel superimposing is as follows:

Algorithm 4: PixelSuperimposing (*V*,*TF*,*r*,*l*)

```
1:    for each i=1,2,...,l^2 do
2:      // Cell-by-cell particle generation.
3:      m = CellByCellParticleGeneration(V,TF,r);
4:
5:      // Accumulate the pixel colors.
```

```
6:      for each pixel p_j do
7:        col_j += AverageSubPixelValues(p_j,m,1);
8:      end for
9:    end for
10:
11:   for each pixel p_i do
12:     col_i = AveragePixelValues(p_i,col_i);
13:   end for
14:
15:   return col;
```

Figure 2 (Color)

Figure 3 (Color)

5 EXPERIMENTAL RESULTS

We performed some experiments to verify the effectiveness of our proposed technique; specifically, we were interested in assessing its image quality and scalability. We used a PC with a 2.66 GHz Intel Core 2 Duo processor and 2GB of RAM for all experiments. The resolution of all rendered images was 512 x 512 pixels. The irregular volume datasets used for our experiment are listed in Table 1. Figure 3 shows the results of rendering for these datasets. The maximum number of tetrahedral cells was 2 mega.

Table 1. Statistics on test datasets.

Figure 4 (Color)

Figure 5 (Color)

5.1 Image Quality

To evaluate the image quality of our improved PBVR technique, we calculate the root mean square error in two images generated by the PBVR and volume ray-casting techniques (see Figure 4). The root mean square error E_{image} is defined as

follows:

$$E_{image} = \sum_{i}^{M} \frac{e_i}{M}, \quad e_i = \sqrt{\Delta R_i^2 + \Delta B_i^2 + \Delta G_i^2}, \tag{17}$$

where M is the number of pixels, and ΔR, ΔG, and ΔB represent difference values in the red, green, and blue components of the images, respectively. Figure 5 shows the results of the error evaluation as a function of rendering time. These figures show, that the image improved, both visually and numerically, by increasing the sub-pixel level.

Figure 6 shows that the pixel superimposing technique is equivalent to the sub-pixel processing technique. In this case, the number of superimpositions is equivalent to the number of sub-pixels in a single pixel. Figure 7 shows the results of error evaluation for the pixel superimposing and the sub-pixel processing. From this figure, we see similar image quality in the rendering results of both techniques, and the error is inversely proportional to the sub-pixel level.

Table 2 shows the performance results in terms of the number of generated particles, particle generation time (sampling time), and rendering speed; in all these trials, the sub-pixel level was maintained at 7 to keep the image quality as close as possible to that of the volume ray-tracing image. The rendering time for our technique is competitive with that of existing techniques. In our technique, performance depends on the number of tetrahedral cells and generated particles.

Figure 7 (Color)

Figure 8 (Color)

The latter depends on the density function, which is derived from the transfer function, and on the sub-pixel level (i.e., the particle radius). Since the image quality is improved by increasing the sub-pixel level, this level controls the level of detail (LOD) of the particle-based volume rendering (see Figure 8).

5.2 Scalability

For huge volume datasets that would not normally fit into available memory, it is preferable to use a streaming based technique for volume rendering, which breaks a large volume of data into smaller pieces in order to keep volume-rendering processing in physical memory. Our technique can adopt streaming naturally, since it does not require visibility sorting. In our system, only the information for a single volume cell needs to be stored in order to generate particles, which are then transferred to the next stages, namely projection and sub-pixel processing.

Table 2

Table3

Figure 9 (Color)

In order to demonstrate the scalability of our technique as a streaming-based method, we artificially constructed huge irregular volume datasets, based on the datasets listed in Table 1. We subdivided each dataset recursively into eight similar tetrahedral cells in order to obtain huge irregular volume datasets in which the maximum number of tetrahedral cells exceeded one giga, as shown in Table 3. The processing time is also shown in Table 3, and we confirmed that the processing time was proportional to the number of tetrahedral cells (see Figure 9.) In Table 3, numbers that are bracketed in the right side of the dataset names indicate the times needed to make that subdivision. Note that the processing time is composed of the recursive subdivision time and the sampling time, which cannot be measured independently since our technique generates particles with a tetrahedral cell subdivided, and a one giga tetrahedra volume dataset could not be stored simultaneously in the physical memory of our PC.

5.3 Application to oral airflow visualization

Our technique has been applied to visualize the result of oral airflow simulation. The dental fricative voice has been analyzed with a large-scale CFD simulation because the sound was thought to be generated from turbulence around the front teeth. In the simulation model, the oral cavity shape for the dental fricative was obtained using a Cone Beam CT (CBCT) scanner that can record 512 slices, each measuring 512 x 512 pixels, within 18 seconds. From a volume dataset that comprises those image slices, the oral cavity was extracted with two threshold CT values, and 71.45 mega hexahedral cells were constructed for the large eddy CFD simulation. The resulting irregular volume dataset was composed of 16 datasets that resulted from the distributed CFD computation. Surface-based visualization was carried out, since currently available volume rendering software cannot handle multiple, large hexahedral volume datasets. We applied our improved streaming-based PBVR technique to render these 16 datasets. In Figure 10 (a), we can easily see that there are multiple areas with high pressure values, which relate to the sound source. We can also confirm a more precise distribution of the velocity magnitude of the flow field in the oral cavity from Figure 10 (b).

Figure 10 (Color)

6 DISCUSSION

The performance of the improved PBVR is influenced by the number of particles. The number of particles changes according to the visualization parameter. If a transfer function or the pixel size in the world coordinates changes, the density function will need to be changed, and the number of particles may exceed the system capacity. To solve this problem, we consider two potential techniques. First, we could adopt a streaming approach to render irregular volumes. In this approach, not all particles are stored in the main memory, but rather, the particles are generated at each volume cell and projected onto the image plane (similarly to the PT technique). Our improved PBVR technique can be seen as a kind of sort-free splatting technique, in which a splat is composed of opaque and emissive particles. Second, we could develop a technique that reshapes a transfer function such that its features are preserved and the number of particles decreases as a function of some defined criteria. A possible feature can be extreme points in the transfer function.

In our improved PBVR technique, the density function is constant within a tetrahedral or hexahedral cell, and the density value is represented by the cell in the centroid. For example, when the density function changes a lot in the tetrahedral cell, or when the aspect ratio is high, the cell boundary may be prominent. This is the case when a tetrahedral cell becomes large with respect to the screen size, which is frequently the case, especially in an immersive display environment. To solve this problem, we need to consider the density distribution within the cell. In other words, a time-consuming metropolis sampling is required at each volume cell. Therefore, we need to develop a lightweight sampling technique. Additionally, in case of the use of cell-by-cell particle generation, some rendering artifacts may be generated on the boundary surfaces of the cell, which are shared between adjacent cells. Although we could not confirm the artifacts in our experiments, we will develop a technique for removing the artifacts.

7 CONCLUSIONS

In this paper, we propose a technique that is useful for previewing huge irregular volume datasets with minimal pre-processing. For efficient previewing, we have improved our original PBVR technique. The new technique introduces an approach for rendering irregular volumes by regressing to a particle model in which a set of particles is stochastically generated from a user-specified transfer function.

To facilitate previewing when a limited amount of memory is available for storing entire volume cells, we have developed a cell-by-cell particle generation technique by estimating a particle density field from a given transfer function. The only restriction is that there should be sufficient memory space to store at least a single volume cell. Each cell may be processed to generate particles to be projected in an arbitrary order. Simulations using huge irregular volumes are usually run in parallel on clusters of high-bandwidth supercomputers or PCs. The resulting datasets can be so massive that they require parallel

computing resources of similar magnitude in order to visualize them effectively. A streaming-based approach is one promising solution for coping with such a requirement, and our technique is a natural fit to this approach.

In our improved PVBR, there are fluctuations in the resulting pixel values due to the stochastic approach. Recently, we showed analytically that the fluctuations can be decreased by increasing the sub-pixel level, which controls the LOD of the volume rendering [19]. On the other hand, this increase requires the frame buffer space to increase in proportion to the square of the sub-pixel level. To reduce memory costs for storing all the sub-pixel values, we have developed a pixel superimposing technique. When we repeat the random distribution of particles $level^2$ times with the original image resolution, i.e., the sub-pixel level is 1, the pixel superimposing technique can achieve the same effect as that of sub-pixel processing as long as we can assume that the particle distribution is independent at each repetition.

REFERENCES

[1] P. Sabella, A Rendering Algorithm for Visualizing 3D Scalar Fields, ACM SIGGRAPH Computer Graphics, Vol. 22, No. 4: pp. 51-58, 1988.

[2] K. Koyamada, Volume Visualization for the Unstructured Grid Data, Proc. of SPIE; Vol.1259, pp.14-25, 1990

[3] M. P. Garrity, Raytracing irregular volume data, Proc. of the 1990 workshop on Volume visualization; pp. 35-40, 1990

[4] K. Koyamada, S. Uno, A. Doi and T. Miyazawa, Fast Volume Rendering by Polygonal Approximation, Journal of Information Processing; Vol. 15, No. 4, pp. 535-544, 1991.

[5] P. Shirley, and A. Tuchman, A Polygonal Approximation to Direct Scalar Volume Rendering, Proc. of San Diego Workshop on Volume Visualization; pp. 63-70, 1990

[6] P. Williams, Visibility-ordering of Meshed Polyhedra, ACM Transactions on Graphics; Vol. 11, No. 2, pp. 103-126, 1992.

[7] B. Lucas, G. Abram, N. Collins, D. Epstein, D. Gresh, and K. McAuliffe, An Architecture for a Scientific Visualization System, Proc. of IEEE Visualization; pp. 107-113, 1992

[8] K. Koyamada, and T. Itoh, Fast generation of spherical slicing surfaces for irregular volume rendering, Visual Computer; Vol. 11, No. 3, pp. 167-175, 1995.

[9] J. Meredith, and Kwan-Liu Ma, Multiresolution View-Dependent Splat-based Volume Rendering of Large Irregular Data. Proc. of IEEE 2001 Symp. On Parallel and Large-Data Visualization and Graphics; pp. 93-99, 2001

[10] B. Wylie, K. Moreland, L. A. Fisk, and P. Crossno, Tetrahedral Projection using Vertex Shaders, Proc. of IEEE Symposium on Volume Visualization '02, ACM Press; pp. 7-12, 2002.

[11] S. Roettger, and T. Ertl, Cell Projection of Convex Polyhedra, Proc. of Eurographics workshop on Volume Graphics 2003; pp. 103-107, 2003

[12] B. Csebfalvi, and L. Szirmay-Kalos, Monte Carlo Volume Rendering, Proc. of IEEE Visualization 2003; pp. 449-456, 2003

[13] B. Csebfalvi, Interactive Transfer Function Control for Monte Carlo Volume Rendering, Proc. of IEEE Symposium on Volume Visualization and Graphics 2004; pp. 33-38, 2004

[14] S. P. Callahan, M. Ikits, J. L. D. Comba, and C. T. Silva, Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering, IEEE Transactions on Visualization and Computer Graphics; Vol. 11, No. 3, pp. 285-295, 2005.

[15] K. Liang, P. Monger, and H. Couchman, Interactive Parallel Visualization of Large Particle datasets, Parallel Computing, Vol.31, No.2, pp.243-260, 2005.

[16] E. W. Anderson, S. P. Callahan, C. E. Scheidegger, J. Schreiner, and C. T. Silva, Hardware-Assisted Point-Based Volume Rendering of Tetrahedral Meshes, Proc. of Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI), pp. 163-172, 2007

[17] P. Muigg, M. Hadwiger, H. Doleisch, H. Hauser, Scalable Hybrid Unstructured and Structured Grid Raycasting, IEEE Transactions on Visualization and Computer Graphics; Vol. 13, No. 6, pp. 1592-1599, 2007.

[18] N. Sakamoto, J. Nonaka, K. Koyamada, and S. Tanaka, Particle-based Volume Rendering, Proc. of Asia-Pacific Symposium on Visualization 2007; pp.129-132, 2007

[19] K.Koyamada, N.Sakamoto, and S.Tanaka, A Particle Modeling for Rendering Irregular Volumes, Proc. of Int. Conf. on Computer Modeling and Simulation; pp. 372-377, 2008

[20] N. Metroplis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E.Teller, Equation of State Calculations by Fast Computing Machines, Journal of Chemical Physics, Vo. 21, No.6, pp.1087-1092, 1953.

Figure1

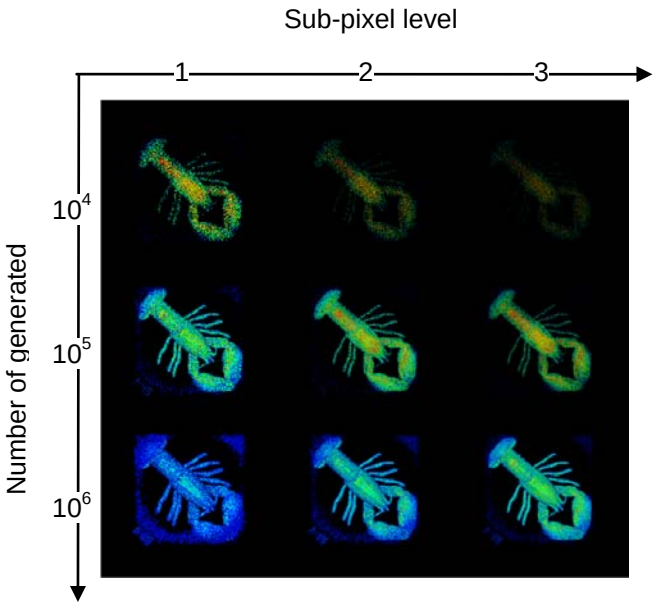


Figure 1. Rendering images by using original PBVR.

Figure 2

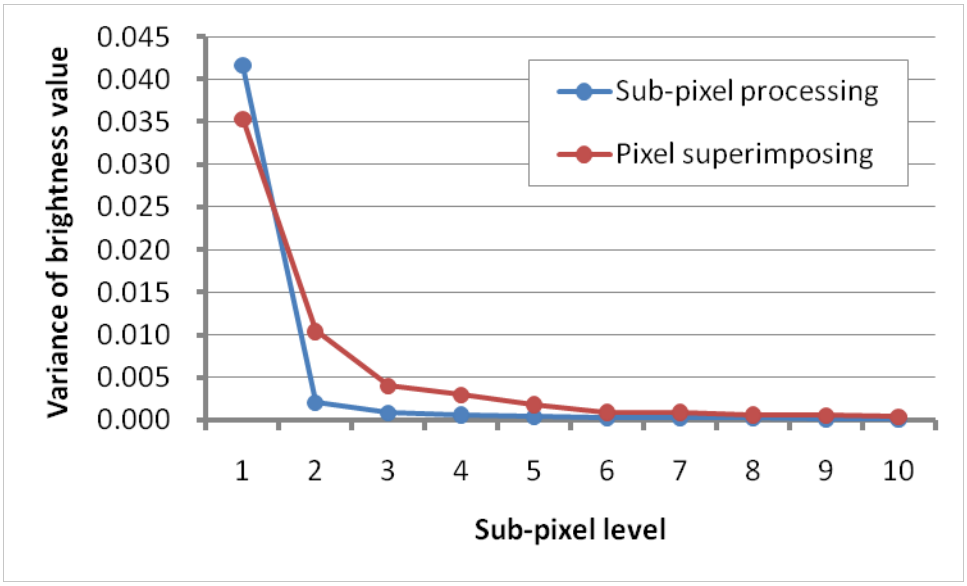


Figure 2. Accuracy comparison of the sub-pixel processing and the pixel superimposing in a single pixel.

Figure 3

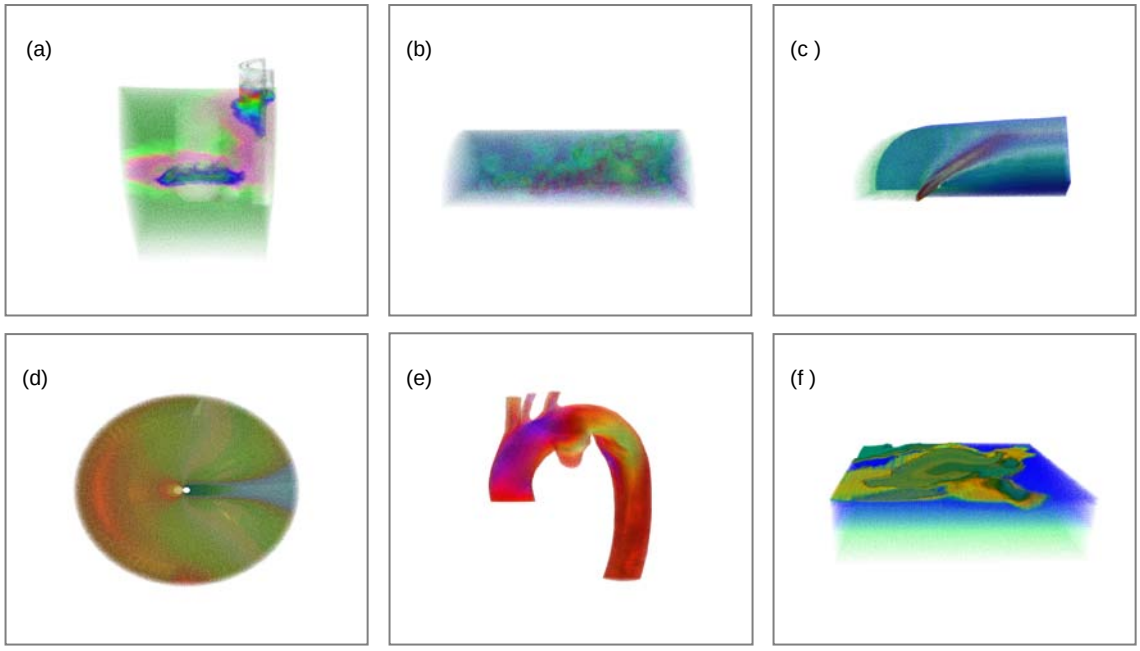


Figure 3. Results of rendering for (a) Spx, (b) Fighter, (c) Blunt, (d) Post, (e) Aorta and (f) Sf2.

Figure 4

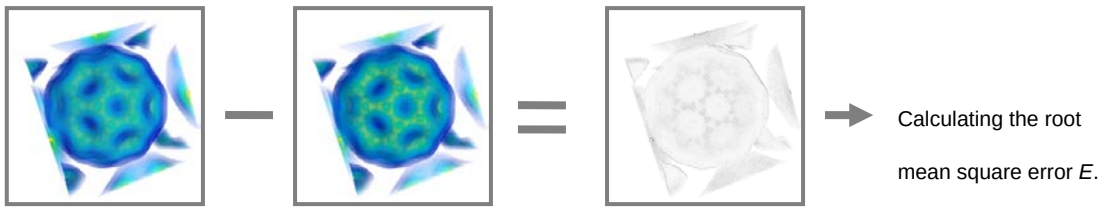


Figure 4. Evaluation of the image quality.

Figure 5

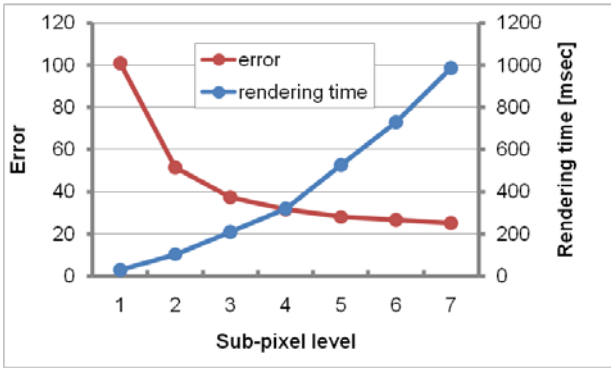


Figure 5. Error distribution of the image quality and rendering time.

Figure 6

l = 3 *l* = 5 *l* = 7

(a) Sub-pixel processing (*l*: sub-pixel level).

n = 9 *n* = 25 *n* = 49

(b) Pixel superimposing (*n*: number of superimposing times).

Figure 6. Comparison of the sub-pixel processing and the pixel superimposing.

Figure 7

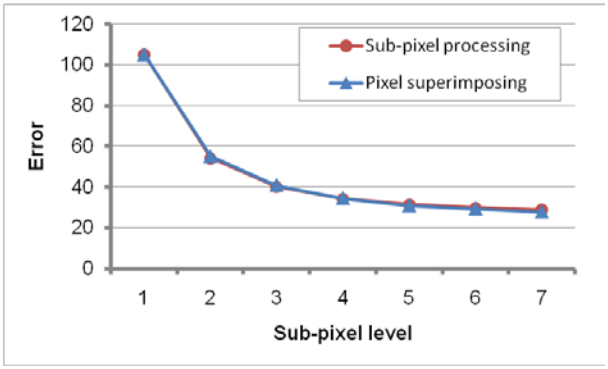
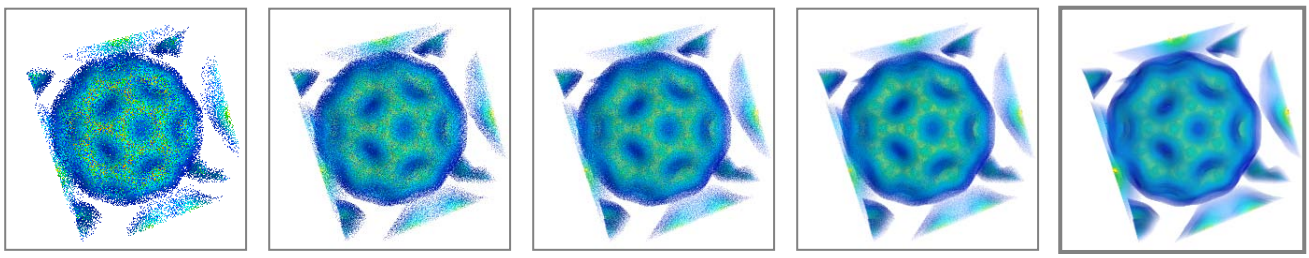


Figure 7. Error distribution of the pixel superimposing.

Figure 8



level = 1 level = 2 level =3 level = 7 Volume ray-casting image

Figure 8. LOD rendering of bucky ball (1.25M tetrahedral cells) by changing the sub-pixel level: 1 at 31.7 fps (A), 2 at 9.4 fps (B), 3 at 4.7 fps (C), and 7 at 1.0 fps (D). The right image is generated by using volume ray-casting. An animation is available at http://www.viz.media.kyoto-u.ac.jp/PBVR/

Figure 9

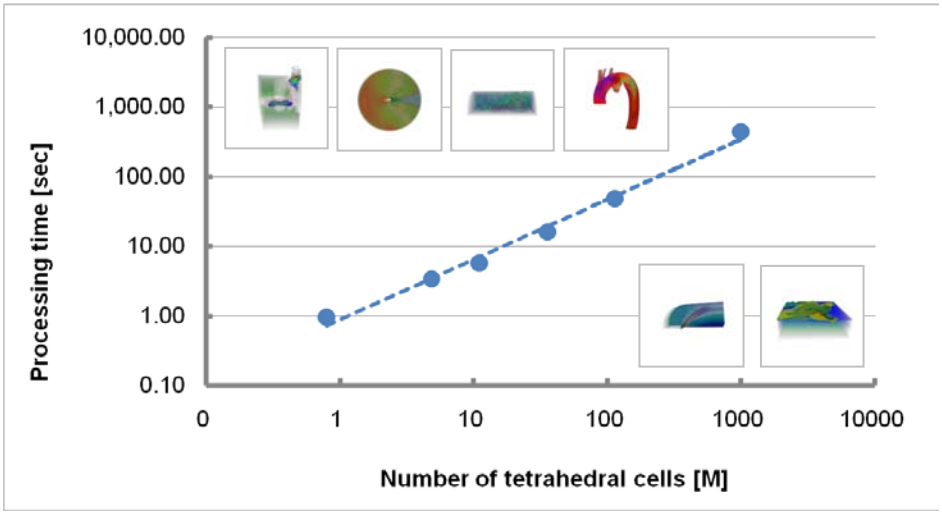
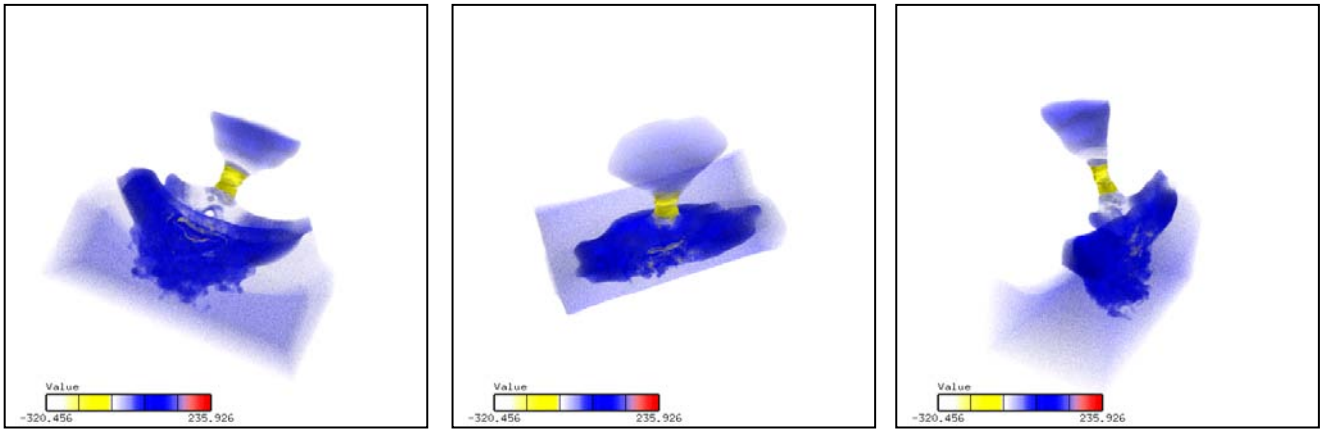
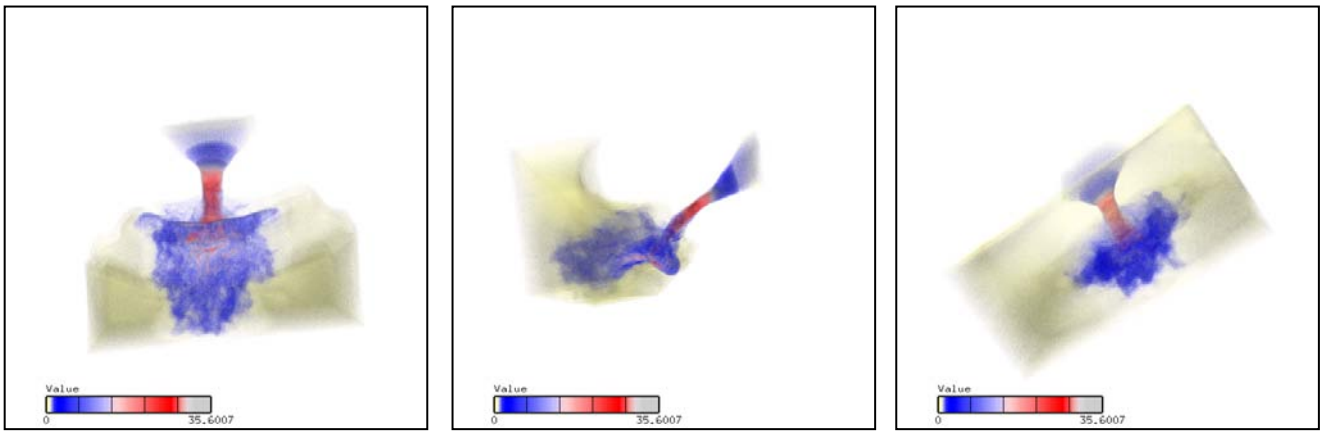


Figure 9. Processing time for large irregular volume datasets.

Figure 10



(a) Pressure



(b) Velocity magnitude

Figure 10. Results of the oral airflow visualization by applying our improved PBVR.

Tables

Table 1. Statistics on test datasets.

Data set	Num. of tetrahedra	Num. of vertices
Spx	12,936	2,896
Fighter	70,125	13,832
Blunt	222,414	40,948
Post	616,050	108,300
Aorta	1,386,882	248,992
Sf2	2,067,739	378,747

Table 2. Sampling time and rendering time [msec]. (sub-pixel level: 7)

Data set	Num. of particles	Sampling	Rendering
Spx	1.8M	751	374.5
Fighter	2.4M	926	452
Blunt	3.3M	1244	467.5
Post	6.7M	2276	753
Aorta	4.0M	1684	433.5
Sf2	9.0M	3898	741.5

Table 3. The table shows the processing time which is composed of the recursive subdivision time and the sampling time, and it is proportional to the number of tetrahedral cells. In this table, numbers that are bracketed in the right side of the dataset names indicate the times needed to make that subdivision.

Data set	Num. of tetrahedra	Processing time [sec]
Spx (2)	0.8M	0.53
Fighter (3)	35.9M	7.29
Blunt (3)	113.9M	21.70
Post (1)	4.9M	1.74
Aorta (1)	11.1M	2.59
Sf2 (3)	1.0G	196.25