[1]VYATKIN S.I., [2]ROMANYUK A.N., [2]ROMANYUK O.V., [2]Kokushkin V.M
[1]Institute of Automation and Electrometry SB RAS
[2]Vinnytsia National Technical University

# OPTIMIZED VOLUME RENDERING USING OCTREE ON A GPU

Volumetric data is a three-dimensional array of cubic elements (voxels) representing units of 3D space. Volume visualization is the process of converting volume data, after pre-processing, into a two-dimensional image, which can be shown on a computer screen, representing an object or phenomenon in a realistic way, visually transmitting them the internal structure [1]. A review of existing methods for accelerating algorithms for tracking and discarding rays shows that most of them are based on one or more principles: coherence in voxel space; coherence in pixel space; coherence of rays; coherence of frames; spatial jumps (space-leaping) [2].

In this paper, we propose a method for speeding up calculations using the octal tree structure.

When performing volume rendering, usually only a small amount is used the percentage of all voxels actually contributes to the final the image, the rest remain invisible. The goal is to reduce the selection of voxels in 3D areas that contain the same or similar values. First, a low-frequency sample of volume points is considered when the beam passes through. To do this, take a large step between the selection points, if between two neighboring selection points show a large difference in values, and an additional point is selected between them. This the basic idea is generalized in order to reduce the sample size in areas where opacity makes a small contribution or areas where the volume is uniform. Every octree node corresponds to a cuboid part of the voxel volume. A cuboid is divided into eight parts, corresponding to the child nodes. Octree is kept in main memory. It describes the area of the visible data.

For trilinear interpolation, the cell is defined as a cube whose eight corners are adjacent to the voxel values. For each position inside the cell, the intensity value is defined as a trilinear interpolation of angular values. Hence, the cell it can be completely empty only if it has eight angular values they are completely transparent after application the transfer function. Each node of the octal tree has a variable describing the ratio of visible data to the total amount of data in the cube. At the last level of the tree, each node represents one cell that is either completely filled in or empty. Every higher octree level nodes ratio is calculated by averaging the ratios of its children. This calculation only needs to be performed when the transfer function has changed. In process traversing a node, its children have to be sorted in a front to back order. While a node is to be drawn, the cuboid box corresponding to this node is sliced, and the slices are rasterized and blended into the previously drawn slices.

A software model was developed and implemented to ensure the most effective interactive work with the GPU [3, 4]. The model is based on presenting the entire work to the user in a hierarchical tree, where individual elements represent the current work parameters and possible settings.

With this approach, the continuous process of getting an image on the screen looks like a dynamically managed data interpretation pipeline, with the ability to enable/disable and configure individual nodes. All settings that you can change are grouped into separate elements, which allows you to work with them (including writing and saving) as with ready-made sets. Each key stage of visualization corresponds to an element in the tree that is presented to the user. To ensure that the tree is represented and interacts effectively with it, the class representing each element inherits basic functions from a single class. This class provides:

- visual presentation elements (for example, that the active elements are shown in bold and full-color icons), change notification status or properties of an item all related items and their Windows, shortcut, and popup menus, etc.;

- the installation and removal activity element and, after the appropriate check constraints, the corresponding changes in the rendering process;
- basic properties (name, file path, whether the item can be copied, deleted, etc.), templates to ensure that these properties are changed, loaded, and saved;
- All work with descendants in the tree (for example, limiting the maximum number of active descendants and checking its execution, etc.);

To ensure uniform interaction between the element and its representation to the user, this class, in turn, is inherited from the MFC - class CDocument, which makes available all the functionality of the "document-view" architecture.

The most important element is an element that represents a data file in some format. This element has internal structures that describe values at volume points, and can describe multiple fields at each point, such as density and temperature. All these fields are visually represented in the tree as descendants of the "data file" element and can be active (i.e. rendered) just double-click on the corresponding element. In this way, various fields can be drawn together or separately, including from different files.to do this, just drag and drop (or copy and paste) the desired field element from one data file element to another and make it active. All processes and necessary transformations to get a new volume from different fields, such as data re-scaling, are implemented so that the user's participation is minimal. The values at the points of each field can be converted to color and transparency values via a separate table (CLUT - Color Lookup Table), which is represented to the user by the fact that each element of the "field" type has descendants of the "CLUT" type, of which only one can be active, i.e. enabled.

The translation table field values to color and transparency, is presented to the user separately for each component of RGBA (Red Green Blue Alpha) plane, where one axis represents the possible value of the field at point and the other with the corresponding value of the component.

Testing was performed on a computer with an Intel Core 2 CPU E8400 3.0 GHz, and a GeForce 8800 GTX graphics accelerator. Performance was tested on $512^3$-size scenes, and the optimized version worked much faster. Visualization of the non-optimized version took an average of 9 frames per second, while the optimized version took 80 frames per second.

**Conclusion**

In this paper, we present a method for accelerating volumetric rendering based on the GPU. Using the octal tree structure, it becomes possible skipping data that is not visible after applying the transfer function. Therefore, if the rasterization is eliminated bottleneck.

Optimal ratio the parameters of the octal tree are determined by the rasterization phase and the compromise between the number of rasterization operations and the data throughput. Because octree depends on the transfer function, it must be recalculate when the transfer function changes. The results show that parameters can be optimized for various graphics cards.

**References**

1. Barthold Lichtenbelt, Randy Crane, Shaz Naqvi, “Introduction to Volume Rendering” (Hewlett-Packard Professional Books), Hewlett-Packard Company 1998.
2. Sobeirajski L., D. Cohen, A.Kaufman, R.Yagel, and D.Acker, "A Fast Display Method for Volumetric Data", The Visual Computer, 10(2):116-124, 1993.
3. Романюк О. Н., Довгалюк Р. Ю., Олійник С. В. Класифікація графічних відео-адаптерів // Наукові праці Донецького національного технічного університету. Сер. : Інформатика, кібернетика та обчислювальна техніка. - 2011. - Вип. 14. - С. 211-215.
4. Романюк О. Н., Дудник О. О.,. Костюкова Н. С Реалізація альтернативного конвеєра рендерингу на GPU з використанням об-числювальних шейдерів // Наукові праці Донецького національного технічного університету. Серія : «Інформатика, кібернетика та обчислювальна техніка». – 2017. – № 2. – С. 103-109.