I-005

VISUALIZATION OF LARGE RLE-ENCODED VOXEL VOLUMES

スヴェン　フォストマン†　　大谷 淳†
Sven Forstmann　　Jun Ohya

Abstract

We present a method for visualizing large voxel volumes based on optimized ray-casting. Other than conventional methods casting a ray for each pixel on the screen, our method only casts one ray per column and then traverses the voxel volume in a front to back manner. This can be done efficiently as our data is encoded by run-length-encoding (RLE), reducing the overall cost for the traversal. To exploit frame-to-frame coherency and to make the visualized scene rotation invariant, we are storing the rendered result temporarily in a cube-map. As the cube-map might have any orientation when finally rendered as cube around the view-point, 6 DOF are achieved.

1． Introduction

Volume rendering is an area with many applications and it has been studied well in the past history. The algorithms that have been utilized to achieve the visualization of volume data are various, and range from simple ray-casting over slice based rendering to more complex acceleration structures like octrees, Kd-trees or other space-skipping techniques to improve the visualization speed. A comprehensive review of all significant techniques can be found in [1]. The intention of our research is, to optimize ray-casting based rendering especially for voxel volumes. This is a special case of volume rendering, as empty-space skipping can be treated more efficiently. The main purpose of our algorithm is, to utilize newest graphics hardware for performing ray-casting directly on the GPU using CUDA [4], rather than using inflexible Shaders, which have been used so far for general-purpose computations.

2. Algorithm design

Our intended algorithm is based on the partially documented method of Ken Silverman [3]. The algorithm basically does ray-casting of run-length-encoded (RLE) volume data in a front-to-back manner. The main advantage over conventional volume rendering approaches is that the RLE-encoding drastically accelerates the ray-traversal for average complex scenes. For performing an accurate volume traversal, a gradient based traversal algorithm similar to [2] is used. To further improve the speed, mip-maps of the volume data are generated which speed-up the rendering of distant objects. Our desire is to improve and modify the existing algorithm in several ways, to make it feasible for up-to-date graphics hardware. We therefore have to take care of data alignments, caching, parallel computing issues and do further plan to take advantage of frame-to-frame coherencies.

† Waseda University Tokyo, GITS

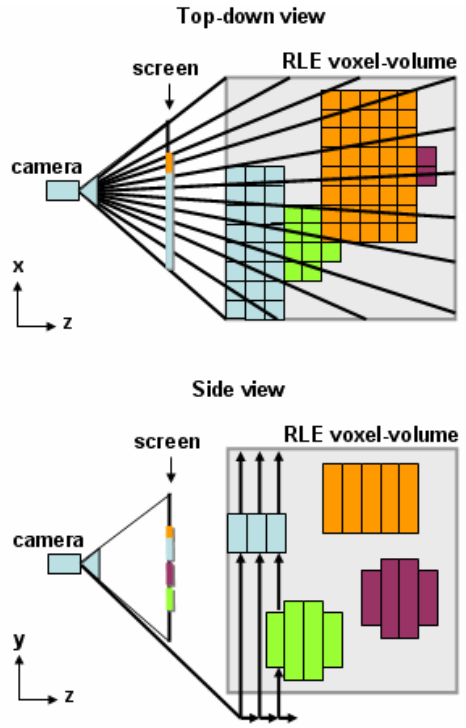

Fig.1. Volume traversal: The volume is RLE compressed in y-direction. The traversal order is performed from front to back in z-direction, as indicated by the black arrows.

3. The basic volume traversal

The algorithm processes the screen successively column by column from left to right in x-direction and in a near-to-far manner in z per column. For each column, one ray needs to be casted in xz-direction, while for each xz-position, all vertical voxel-sticks (the RLE-encoded elements) have to be visited. To allow an early ray termination, pixels on the screen that have been drawn are marked. If one column is marked completely, the traversal can be stopped as no more pixels will be drawn.

The advantage of the proposed method compared to conventional volume rendering is, that we do not have a complexity of $O(n^3)$.The complexity is $O(n*n*c)$, where n is the side length of the volume cube, and c the maximum complexity in y-direction. In worst case situations, c might be equal to n, but for average 3D-scenes (especially landscapes), c is a number that is much smaller than n.

4. 6 degrees of freedom

The method explained above works well if the camera is oriented orthogonal to the xz-layer as shown in Fig.2, resulting in 4 degrees of freedom (DOF). However, our idea of visualization is, to allow a free camera orientation with 6 degrees of freedom.

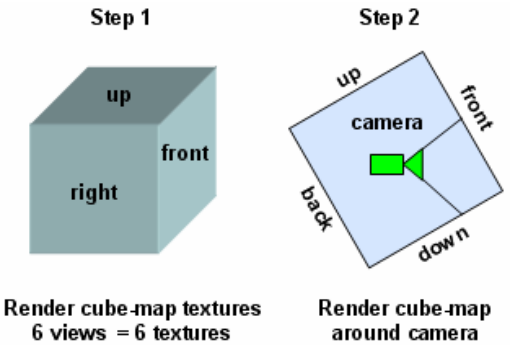

Fig.2. 6 DOF by using a cube-map: The scene is rendered in six directions in a first step, where each direction is equal to one texture on the cube-map. In the second step, the cube-map is rendered around the camera.

The method described in [3] therefore utilizes a two-step approach. In the first step, the scene is rendered in columns that are aligned to the RLE-encoded volume data to a temporary buffer. In a second step, the temporary buffer is used as a texture and mapped onto the screen.

We propose a different way of handling 6 DOF by storing the rendered results achieved in section 2 temporarily in a cube-map. For the final visualization, we have to place the cube-map as a standard cube around the camera as demonstrated in Fig.2. This allows a rotation invariant caching of the rendered scene.

To speed up the process and avoid rendering all cube-map textures each time completely, it is possible to mark visible areas in a pre-processing step.

5. Shading and Materials

In order to shade the rendered geometry, we need to have normal vectors stored for each of the rendered volume elements. This has to be handled a little bit special manner in our case, as the volume is run-length encoded. In particular, elements like a simple cube for example (see Fig.3) have to be split into multiple parts for accurate normal vectors.

To provide even more surface details, we can apply texture mapping by storing an additional material index in the RLE data structure. The texture coordinates can then be computed based on the voxel-position in 3D-space and the normal-vector orientation, while the final color is computed using tri-planar mapping, as described in [5].

One RLE element finally consists of 8 bytes, which are divided into the following parts:

Start coordinate (2 bytes), run-length (2 bytes), surface normal (3 bytes) and material index (1 byte).

6. GPU Optimizations

As we are dealing with graphics hardware, the scene is stored in textures, as in Fig.4, rather than conventional arrays. The scene is therefore subdivided into quadratic regions where the amount of texture layers for each region depends on the y-complexity. As recent graphic cards have up to 128 and more cores, it is possible to parallelize the rendering task by assigning one column of the screen to each core. Frame-to-frame

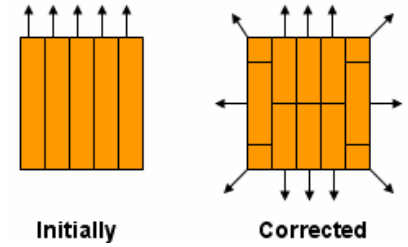

Fig.3. Normal vectors: As our RLE encoded volume only stores one normal per element, we are required to split the object for achieving accurate surface normals.

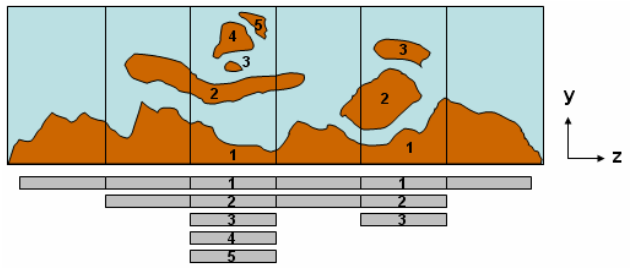

Fig.4. Memory management: The scene is stored in textures, where each texture can store one y-layer. The textures are indicated as grey blocks below the scene

coherency can further be employed, by caching the visible RLE elements of the view-frustum in a texture, avoiding a complete traversal for each xz-position.

6. Conclusions and Future Work

We have presented a volume rendering method for large voxel volumes and proposed several ways to improve existing methods for nowadays graphics hardware. Future work will include the implementation of the algorithm using the CUDA programming language and testing the method on various examples.

Acknowledgements

We would like to thank Samuel Moll of the Ludwig-Maximilian University Munich for his support in the preparation phase of this paper.

References

1). T.Todd Elvins, "*A survey of algorithms for volume visualization*", Computer Graphics, vol.26-3, pp.194--201, 1992

2). J. Amanatides, A. Woo. "*A Fast Voxel Traversal Algorithm for Ray Tracing*", Eurographics Conference Proceedings 1987, pp. 003-010

3). Ken Silverman: Voxlap *http://advsys.net/ken/voxlap.htm* , visited 5/2007

4). NVidia Corp, Compute Unified Device Architecture (CUDA) http://developer.nvidia.com/object/cuda.html

5). Ryan Geiss, Michael Thompson: "*NVIDIA Demo Team Secrets – Cascades*", Talk at GDC2007