Real-time ray casting of volumetric data

Žiga Lesar

University of Ljubljana,
Faculty of Computer and Information Science
E-mail: zl7962@student.uni-lj.si

Abstract

In this paper we present acceleration structures and techniques for real-time ray casting of large volumetric data sets, such as those obtained by CT or MRI scans. Ray casting is a well-known method for efficient volume rendering, as it produces quality images using simple geometric algorithms. Since a lot of computational time is used for determining surface intersections and data sampling, we employ different speedup methods, such as adaptive sampling and sparse casting. For achieving desired rendering quality we use the regula falsi method along with trilinear interpolation for zero-crossing refinement, Monte-Carlo ambient occlusion estimation and prefiltering with a gaussian kernel. Aforementioned methods and algorithms have been implemented with OpenCL, thus exploiting the highly parallel nature of ray casting, while trying to achieve interactive rendering rates. We also compare the results obtained with different visualization concepts - isosurface extraction, maximum intensity projection, and alpha compositing. We tested our methods on medical data sets, specifically angiograms.

1 Introduction

Volume rendering is a term for different approaches used to visualize volumetric data - 3D discretely sampled images usually obtained by medical imaging or numerical simulation. A number of different methods are used for the job, including splatting [10], texture-based rendering [1], conversion to triangular meshes (e.g. marching cubes [4]) etc. One of the oldest and most widely-used is ray casting, introduced in 1982 by Scott Roth [9]. All of these methods rely on more or less direct evaluation of the rendering integral. Some of them strive for visual quality of the rendered image (ray casting), while others aim for speed and performance (marching cubes, texture-based approaches). Although ray casting is a computationally intensive method, it can be easily parallelized for execution on the GPU, allowing us to render high-quality images relatively quickly. With the widespread use of medical imaging as support in making diagnoses came the need for fast and quality visualization of acquired data. For this purpose many acceleration structures have been developed over the years, most of them exploiting object-space and screen-space coherence, as well as neglecting irrelevant information.

2 Method

Volumetric data can be thought of as sampling of a continuous signal $f : \mathbb{R}^3 \to \mathbb{R}$. With a high enough sampling rate we are able to reconstruct the original signal exactly by using a 3D analogue of a sinc filter [6]. However, since the sinc function has infinite extent, exact reconstruction requires taking all the samples into consideration, which shows to be too expensive. In practice, we use simple filters to achieve a "good-enough" result. Box filter (nearest neighbour) and tent filter (trilinear interpolation) are used in our implementation, as it turns out they are a good trade-off between speed and resulting image quality. The data can optionally be prefiltered with a gaussian kernel to smooth out any unwanted features in the data. In the following text the term *plain sample* is used to denote sampling with nearest neighbour filtering. Interpolation sampling takes 8 plain samples and some computation time.

To project a volume onto the projection plane we need a camera object, which holds a position and orientation in 3D space. We use this information to cast rays into the scene, sampling the volumetric data uniformly along the way. To combine the sampled values into the final color we use different compositing schemes, briefly described below. Each one has unique properties, which can be taken advantage of to speed up the rendering process.

2.1 Isosurface extraction

First-hit compositing scheme, also known as isosurface extraction, is the process of finding the set of points $p \in \mathbb{R}^3$, for which $f(p) = \mu$. The value μ is called the isovalue and can be set at runtime by the user. Since the signal is assumed to be continuous and band-limited, the isosurface is a smooth closed set in $\mathbb{R}^3$.

While we are sampling the data along a ray in discrete steps, we check whether the sampled value exceeds the set threshold μ. When this condition is satisfied, we know that the isosurface lies between the last two sampling positions and we can stop the ray marching procedure. Then we can further refine the zero-crossing using conventional numerical methods. We have chosen regula falsi[1], since its assumption of linearity of the function

[1]http://en.wikipedia.org/wiki/False_position_method

makes it an ideal complement to our trilinear sampling choice. After 4 iterations of regula falsi, we shade the point on the isosurface using the Phong shading model [7]. To estimate the surface normal used in the shading procedure, we take 32 plain samples, which are used to estimate the gradient $\nabla f(x)$ in 8 points with a central difference formula, and are then linearly interpolated with the same factors as sample values.

Shading is quite an expensive process, but it also adds a lot to depth perception of the rendering. Striving for visual quality this can further be extended with ambient occlusion, for which we have implemented a Monte-Carlo estimator. The estimator checks for near-distance occlusion in a number of different directions on the hemisphere surrounding the intersection point.

2.2 Maximum intensity projection

Maximum intensity projection (MIP) is a very fast compositing scheme, where we take the maximum value of the function on the viewing ray and project it on the screen using a color, which corresponds to the maximum value. Using interpolation for sampling does not contribute greatly to the final image, so plain samples are enough to get a good rendering. Optionally we can segment the data using simple thresholding, to hide irrelevant information. Compared to other compositing schemes MIP conveys the least amount of depth information, but due to its simplicity and ease of implementation it is still one of the most popular ways to visualize volumetric data.

2.3 Alpha compositing

This is a generalization of other compositing schemes, where we may use a so-called transfer function to map sample values to color and opacity. We have implemented the simplest form of alpha compositing with plain sampling and no shading. Front-to-back compositing was implemented, as described by [8]. A huge improvement in terms of performance can be achieved by early ray termination. The idea is to stop the sampling when the alpha reaches a certain threshold. We have chosen the value 0.95, as the last 5% makes almost no difference in the final image, though performance increases are not so subtle (in our tests up to 80%, depending on the volume and transfer function).

2.4 Exploiting screen-space coherence using sparse casting

Most of the time adjacent pixels hold similar colors or the color changes gradually. We can take advantage of this fact by casting rays for every k-th pixel on screen (in each direction) and interpolating the results to generate colors for intermediate pixels. In theory this means a k^2 speedup. A downside of this approach is that we may miss fine features that are not wider than k pixels when projected on screen. This may present a problem in some specific domains where fine precision of the rendering is of great importance, but users are generally not interested in pixel-perfect images.

Sparse casting does in fact produce somehow blurry images, a side effect caused by interpolation of colors in areas where otherwise sharp features should be preserved. We can overcome this problem by replacing interpolation with ray casting if we detect color gradients larger than some user-specified threshold.

3 Conclusion and future work

In this work we presented the possibilities for accelerating the ray casting procedure. GPU execution itself is a major improvement over CPU implementations, taking the parallel nature of ray casting into account. This comes at a price though, as GPUs are not designed for executing complex branching and control flow statements, meaning we have to avoid those as much as possible. Despite these limitations we can still make GPU ray casting efficient by exploiting coherence and neglecting irrelevant information, for example by early ray termination and sparse casting. While this affects rendering quality it effectively reduces the amount of required samples and operations needed to produce the image.

We are in fact only scratching the surface of what is possible. Future improvements in form of advanced hierarchical structures (e.g. octrees, [5]) could further minimize the number of iterations in the ray marching loop. We could also use the same hierarchy for adaptive sampling as described in [2]. Screen-space coherence could further be improved by beam optimization [3]. Employing a C-buffer [11] can enable us to benefit from temporal coherence to increase performance of ray casting with smooth camera animations.

References

[1] A. Van Gelder and K. Kim. Direct volume rendering with shading via three-dimensional textures. *Proceedings of the 1996 Symposium on Volume Visualization*, pages 23–30, 1996.

[2] Enrico Gobbetti, Fabio Marton, and José Antonio Iglesias Guitián. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *Visual Computer*, 24:797–806, 2008.

[3] Samuli Laine and Tero Karras. Efficient sparse voxel octrees. *IEEE Transactions on Visualization and Computer Graphics*, 17:1048–1059, 2011.

[4] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21:163–169, 1987.

[5] Donald Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19:85, 1982.

[6] Daniel P. Petersen and David Middleton. Sampling and reconstruction of wave-number-limited functions in N-dimensional euclidean spaces. *Information and Control*, 5:279–323, 1962.

[7] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18:311–317, 1975.

[8] Thomas Porter and Tom Duff. Compositing digital images. *ACM SIGGRAPH Computer Graphics*, 18:253–259, 1984.

[9] Scott D Roth. Ray casting for modeling solids. *Computer Graphics and Image Processing*, 18:109–144, 1982.

[10] Lee Alan Westover. *Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm*. PhD thesis, University of North Carolina, 1991.

[11] Ilmi Yoon, Joe Demeres, Taeyong Kim, and Ulrich Neumann. Accelerating Volume Visualization by Exploiting Temporal Coherence. 1997.