A Framework for Real-Time Volume Rendering

R.P. Billingsley, K.A. Hawick

Computer Science Department, University of Hull

Introduction

Volume rendering is the process of image synthesis from volumetric data, this data usually consists of a 3D grid of values (known as Voxels) commonly representing intensity. state, or vectors (Husselmann and Hawick, 2012). Volume Rendering techniques are typically used to render datasets such as medical scans, scientific simulations, and participating media such as smoke and fog.

There's a variety of techniques for Volume Rendering, these span from image order solutions such as ray-tracing, to object order approaches like Instancing, to hybrids of the two (Zhang et al., 2011) .

We are concentrating on the use of Volume Rendering techniques to produce interactive renderings of the output of Agent Based Models (ABMs) such as those seen in Figure 1 and 2.

These datasets are large scale, time varying, heterogeneous and opaque. This is a step away from typical volumetric dataset as they usually consist of partially transparent data that has to be sampled and blended. With opaque datasets the problem turns from one of sampling to one of sorting as much of the available rendering performance is wasted on non-visible fragments.

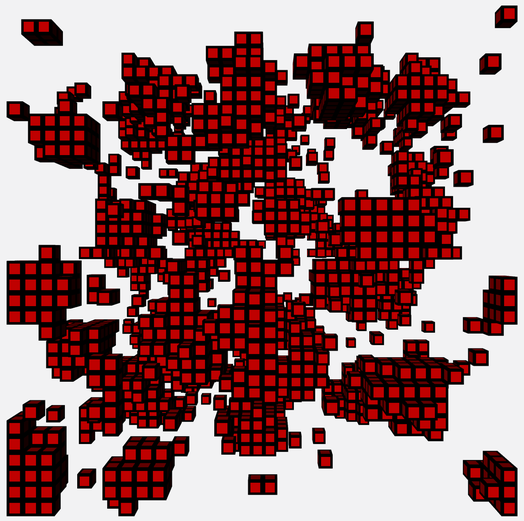

Figure 1: Rendering of a Kawasaki simulation dataset, where the agents group together, this brick is from the midway point in the data set.

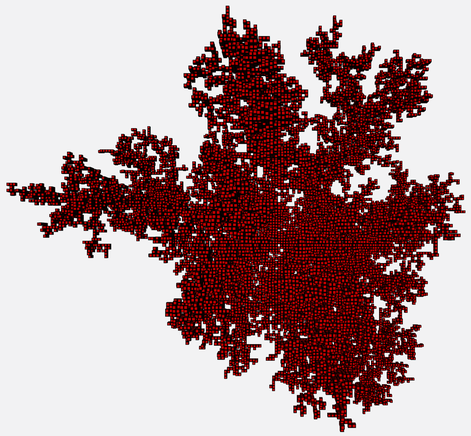

Figure 2: Rendering of a Diffusion Limited Aggregate growth simulation dataset (Hawick, 2010), where agents push out from the central point of the volume to produce a branching structure, this is the final state of the dataset.

Context

To develop and test real-time graphics research is a difficult task as it requires a large amount of boilerplate code. This code is for basic (but essential) implementations of mesh loading, resource management, shader management, and scene management and control. This is in addition to the use of graphics API's such as OpenGL, Vulkan, Metal, and Direct3D.

The idea of building a framework that allows quick implementation of graphics techniques is an attractive one, in the field of Volume Rendering these custom frameworks/applications are common (Crassin, 2011)(Kroes et al., 2012)(Noon, 2012), however these frameworks are all specialised to their technique. This framework is intended to be used primarily for Volume Rendering and as such this means it can be tailored to enable implementation of existing Volume Rendering techniques and relevant data structures.

This framework is also designed to be graphics API independent and cross platform, this is so that new graphics API's can easily be added to the framework to exploit previously unavailable API capabilities Foley (2015) for the context of Volume Rendering. API releases such as Microsoft's Direct3D 12, The Khronos Group's Vulkan and Apple's Metal, and this means that there are some new API capabilities (such as Asynchronous Compute (Boyd, 2015)) that can be used to improve Volume Rendering techniques.

Features

The framework is designed to be platform and API independent, current working builds exist for Windows and OS X, with graphics API support varying by platform and currently include OpenGL 4+ and Direct3D 11, with runtime compilation of GLSL and HLSL currently supported.

The framework can read volume files from a variety of different sources, including Hyperbrick/Sparsebrick (.hbrk/.sbrk) files, and MagicaVoxel VOX files (.vox) and SLAB6 VOX files (.slab.vox/.vox), with more sources to be added as required.

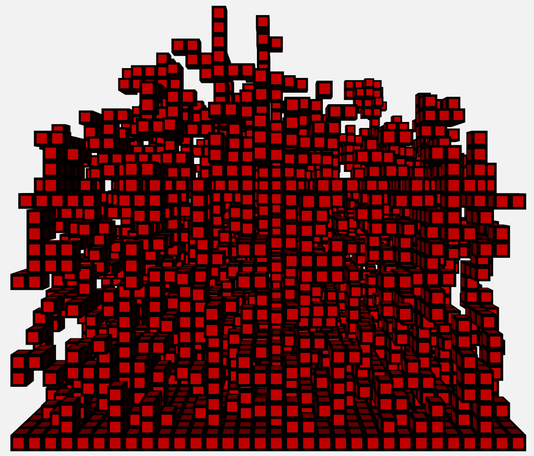

Figure 3: Rendering of a VPE dataset, where the agents start from the base layer of the volume and grow upwards, this block is from the end of the dataset.

Design

The framework has been designed to be API independent, this has been achieved by abstracting API calls into their own classes, with data being passed to the API class to set up the graphics card ready for rendering, and later on, data passed in to render the volumes.

To add support for a new graphics API into the framework, a class implementing the relevant API calls for initialisation of resources on the graphics card, and the setup and dispatch of draw calls for rendering the volume needs to be written, along with a class implementing Shader loading/compilation. The user can then choose the desired API at runtime (depending on platform).

Future Work

A framework is being produced to enhance and accelerate research into Volume Rendering, it is currently being used to render Agent Based Model datasets for observation and analysis. The framework is capable of producing renderings using a variety of techniques and graphics API's, with further support for additional techniques, data structures, and API's to be added.

References

Boyd, C. (2015). Direct3D12. In *An overview of next-generation graphics API's SIGGRAPH 2015*, New York, NY, USA. ACM.

Crassin, C. (2011). *GigaVoxels: A Voxel-Based Rendering Pipeline For Efficient Exploration Of Large And Detailed Scenes*. Ph. D. thesis, Universite de Grenoble.

Foley, T. (2015). Next-Generation Graphics API's: Similarities and Differences. In *An overview of next-generation graphics API's SIGGRAPH 2015*, New York, NY, USA. ACM.

Hawick, K. A. (2010). 3D Visualisation of Simulation Model Voxel Hyperbricks and the Cubes Program 3D Visualisation of Simulation Model Voxel Hyperbricks and the Cubes Program. Technical report, Massey University.

Husselmann, A. V. and K. A. Hawick (2012). 3D Vector-Field Data Processing and Visualisation on Graphical Processing Units. *Proc. Int. Conf. Signal and Image Processing (SIP 2012)*, 92–98.

Kroes, T., F. H. Post, and C. P. Botha (2012). Exposure render: An interactive photo-realistic volume rendering framework. *PLoS ONE 7*(7), e38586.

Noon, C. J. (2012). *A Volume Rendering Engine for Desktops , Laptops , Mobile Devices and Immersive Virtual Reality Systems using GPU-Based Volume Raycasting*. Ph. D. thesis, Iowa State University.

Zhang, Q., R. Eagleson, and T. M. Peters (2011). Volume visualization: A technical overview with a focus on medical applications. *Journal of Digital Imaging 24*(4), 640–664.