Summary of *Far Voxels: …*

Victor Shu

Some 3D rendering use cases like 3D scanning or CAD require rendering huge models interactively, while these models often overload the computational and memory capacity of the state-of-the-art hardware. To solve the problem, the paper proposed an efficient approach for rendering huge arbitrary surface models, which integrates visibility culling, off-core data management, and level-of-detail framework. When pre-processing, a coarse volume hierarchy is generated by binary space partitioning the input data, and the inner nodes of such hierarchy are discretized into voxels. These voxels contain approximations of the appearance of the model when viewed from far away. When rendering, the structure is refined and rendered in front-to-back order.

There are several related works listed in the paper. There are four types of methods that are related to the paper used by prior researches. The out-of-core view-dependent simplification method constructs a graph of possible refinement operations at the point, vertex, or triangle level. Researches that used this method mostly are optimized for small numbers of high-complexity, densely-tessellated objects. Visibility culling algorithms include from-point and from-region algorithms, where from-region algorithms compute a potentially visible set (PVS) for a fixed subdivision of the scene and are applied offline in a pre-processing phase, and from-point algorithms are applied online for each viewpoint. Apart from that, many hybrid algorithms have been proposed. Finally, ray tracing techniques can be used instead of rasterization. Ray queries can be answered in logarithmic time with spatial indexing, and it can access only the visible part of the scene.

The multiresolution framework in the paper hierarchically partition the model with an axis-aligned BSP tree, whose leaf nodes are rendered using original triangles, and inner nodes are approximated with voxels. The first phase of constructing such a structure is to partition the scene with the bounding box of the scene is the root node, and leaf nodes contain a predefined number of triangles. This tree is constructed out-of-core, and the final structure is stored on disk in binary form. After that, the LOD structure for the scene is generated with the BSP tree. First, the final multiresolution structure layout is generated by constructing a coarse hierarchical structure on top of the BSP tree, and then the final representation of the nodes is generated and stored on disk. Generating rendering data is the main task.

First, shading information samples are acquired by casting rays against the BSP from many possible viewing positions. If the hit is inside the node's bounding volume, the voxel index is computed and stored in a list containing the surface reflectance, surface normal, and the ray direction, while the hit outside the bounding volume is discarded. This method is very generalized for many kinds of models, but if not enough rays are cast, the pre-processing cost is high, and there are going to be aliasing problems. After that, the samples are compressed by fitting them to parameterized shader models and choosing a shader that provides the best approximation. There are three types of shaders in the implementation of the paper: K1a/K1b, a flat shader for planar surfaces, parameterized by a plane normal, a front material and a back material, with the normal calculated with different methods, and K2, a smooth shader parameterized by 6 reflectance and 6 normals for complex voxels.

At rendering time, a refinement algorithm executes first to decide the maximum required project voxel size on the screen and do the visibility culling. Then, the nodes are rendered with a splatting method

that draws a point primitive per voxel and a triangle-strip per leaf node. The GPU cached version of a node is used when the node is rendered to minimize bus traffic. This method is sufficient for the target applications of the paper but limits the ability to process high-resolution textures and transparency.

The paper evaluates the proposed method with three models: St. Matthew, a very dense high-resolution laser scanned model; the Richtmyer-Meshkov Isosurface, a very convoluted mesh with holes, a vast depth complexity; and Boeing 777, a very complex CAD model, composed of many loosely connected parts. There are two parts in the evaluation: the pre-processing part and the adaptive rendering part. The pre-processing part was executed on 16 PCs. Each PC has two CPUs, 1GB memory, and 70GB hard disk. The processing time of the proposed method range from about 1K input triangles per second for 1 CPU to 20K input triangles per second for 16 CPUs, while other similar approaches based on geometric simplification ranging from 3K triangles per second for 1 CPU to 30K triangles per second on 16 CPUs. This result indicates that the proposed method is slower, but it scales better on multiple CPUs. The rendering performance is evaluated with a Linux PC with 1 Intel Xeon CPU, 1GB memory, two 70GB hard disks, and an NVIDIA GeForce 6800 GT card. In addition to the three models rendering with 640x480 resolution, two other tests rendering all of the three models with 640x480 and 2x1024x768 resolution are presented. The system can maintain interactivity in all 5 test cases when producing detailed images. The rendering rate of the proposed system is around 45M primitives per second, where one primitive can be a triangle or a voxel. According to the test, a single PC is able to produce two 1024x768 images at 20FPS on average and 3FPS for the worst case.

To summarize, the proposed system is efficient for out-of-core construction and view-dependent rendering of huge models on consumer-level platforms. The pros of the system include the performance and the ability to be applied to general models. The future work includes compression of the output data to reduce disk usage and latency, implementation of multiresolution sampling methods, exploration of texture-based volume rendering methods, and so on.