View Dependent Sequential Point Trees

Wen-Cheng Wang1 (王文成), Feng Wei1,3 (魏 峰), and En-Hua Wu1,2 (吴恩华)

1 State Key Lab of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080, P.R. China
2 Department of Computer and Information Science, Faculty of Science and Technology, University of Macau
Macau Special Administration Region, P.R. China
3 Graduate University of Chinese Academy of Sciences, Beijing 100039, P.R. China
E-mail: {whw, weif}@ios.ac.cn; ehwu@umac.mo; enhua@ucas.cn
Revised November 18, 2005.

Abstract Sequential point trees provide the state-of-the-art technique for rendering point models, by re-arranging hierarchical points sequentially according to geometric errors running on GPU for fast rendering. This paper presents a view dependent method to augment sequential point trees by embedding the hierarchical tree structures in the sequential list of hierarchical points. By the method, two kinds of indices are constructed to facilitate the points rendering in an order mostly from near to far and from coarse to fine. As a result, invisible points can be culled view-dependently in high efficiency for hardware acceleration, and at the same time, the advantage of sequential point trees could be still fully taken. Therefore, the new method can run much faster than the conventional sequential point trees, and the acceleration can be highly promoted particularly when the objects possess complex occlusion relationship and viewed closely because invisible points would be in a high percentage of the points at finer levels.

Keywords view dependence, visibility culling, GPU, point-based rendering, real time rendering

1 Introduction

Points have become powerful primitives for modeling and rendering large-scale complex 3D objects[1–4]. By the scheme, 3D objects are represented in hierarchical point trees with topological information ignored completely and the method is rendered by handling points at suitable level of details. To produce high quality images, various methods have been presented to approximate points in efficient splats and render splats in efficient measures[5–11]. For fast rendering, various methods have been proposed to quickly find the points to save time on processing useless points, and use graphic processing units (GPUs) to obtain hardware acceleration. Here, the useless points include those over coarse or over fine, as well as those invisible at suitable levels of details for rendering.

By a depth traversal of hierarchical point trees, the point set for rendering can be conveniently found[4,12–15], but such a traversal prevents the points from using GPU efficiently. To take advantage of GPU, various methods have been proposed[5,7,16–21]. Among those, the best one[20,21] is the scheme re-organizing the hierarchical points sequentially in a list according to their geometric errors in modeling objects. In the scheme, over coarse and over fine points could be efficiently culled and the points selected can be sent to GPU in a stream. However, these methods lack efficient measures to cull invisible points, and as a result, though they could be used efficiently for rendering distant objects, they are not well for close objects because large amount of finer points from close objects demand intensive computation on visibility processing.

As we know, in any view of a large-scale complex object, only a small percentage of its primitives are visible[22,23]. So efficiently culling invisible points from a view can speed up rendering. In this paper, we present a view dependent method to organizion the hierarchical points in a sequential list with the hierarchical tree structures efficiently embedded in the list. As a result, invisible points can be culled in high efficiency to greatly reduce the points sent to GPU, and the points can be still sent to GPU in a stream to obtain great hardware acceleration.

2 Related Work

Up to now, point-based modeling and rendering techniques have been proposed in large quantities. It is beyond the scope of the paper to discuss them all. Here, we mainly discuss the methods related to our work on the topic of selecting suitable points and using GPU for acceleration, especially the methods on re-arranging points sequentially.

In general, a 3D object is modeled in hierarchical points for level-of-detail rendering, with the finest points simplified in coarser points gradually to the coarsest point representing the whole object. These points are always organized in a tree according to the hierarchical relations between them.

By top-down searching the hierarchical point tree, the points at suitable level of details are selected to render. Once a point is selected to render, the points of its subtree will not be processed any longer. Apparently, if...
a point is invisible and its subtree is invisible too, the subtree will be also culled from rendering. By such a hierarchical traversal, suitable points can be found conveniently without treating the over-fine points and invisible points\cite{2,12-15}. However, selecting the suitable points by traversing up or down in the hierarchical tree cannot guarantee to process points sequentially, which makes it difficult to use GPU’s programmability.

Various methods have been proposed to use GPU for acceleration. In \cite{18}, the points are sorted from back to front by CPU and then rendered by GPU, but it may process a lot of useless points. The methods in \cite{16, 17, 19} generate random point samples on the fly and sort them in a sequence to render by GPU, but they are less suited for smooth, connected surfaces. In \cite{5, 7, 11}, the algorithms proceed in two or three passes to leverage the computational power and programmability of GPU for increasing the performance of EWA splatting, but they do not care much for selecting suitable points for acceleration.

Till now, sequential point trees\cite{20} are regarded as the most efficient scheme for using GPU to point based rendering for acceleration. By the errors of points in modeling objects, the points are re-arranged sequentia-
lly in a list from coarse to fine. Thus, the points at suitable levels for rendering can be quickly found in a segment of the list because they are always in similar errors, and so they can be sent to GPU in a stream to render in high efficiency. The geometric error consists of two components: the perpendicular error, $e_p$, for measuring variance and the tangential error, $e_t$, for measuring approximation. By expressing a point in a disk with the same center, normal and diameter, the $e_p$ of the point is computed as the minimum distance between two planes parallel to its disk that encloses all the disks of its child points, and the $e_t$ of the point is computed with its diameter minus the width of the tightest slab that clamps its projected child disks to the smallest. As the viewing distance decides whether a point is suitable for rendering by its geometric error, the point’s geometric error can be transferred to be represented in a pair of distances, the minimum distance, $r_{\text{min}}$, and the maximum distance, $r_{\text{max}}$, from the viewpoint to the point, to facilitate selection of suitable points. In implementation, all the hierarchical points are sequentially re-arranged by their related $r_{\text{max}}$ values, and in rendering, a pair of $r_{\text{min}}$ and $r_{\text{max}}$ will be computed for a view to choose suitable points. Although this method can efficiently find the points suitable for rendering by their errors to take advantage of GPU, visibility computation has to be made on the points individually in the GPU. Because large portion of points suitable for rendering by their errors might be invisible, visibility computation on these invisible points may waste much time, especially when the invisible points take up a high percentage of the selected points, e.g., if an object is viewed closely.

The sequential point trees have been extended to handle out-of-core objects\cite{21}, EXTreme Splatting (short Xsplat). By the extension, the points are also re-
arranged sequentially from coarse to fine by their related errors, and paginated into a list of blocks. To support efficient visibility computation, it arranges the points in similar errors with the nearby ones in the space close each other in the list to facilitate searching the occluded points early, and produces new points for representing blocks respectively, called block points, to cull invisible points in groups. In rendering, the block points, se-
quently arranged by their errors, are first checked in the CPU to find visible blocks, and then the points of visible blocks are sent to the GPU for rendering. As a result, much time can be saved on loading and process-
ing the points of invisible blocks. Our new method sounds similar to this method. But they are different in many fundamental aspects as compared in the following points. In particular, our new method can execute visibility determination more efficiently.

- Our method arranges points sequentially by the hierarchical tree levels, but the Xsplat method by the errors. If the tree is balanced, they may produce a similar list of points. Otherwise, they produce different lists of points because the finer points at higher levels may be placed at posterior positions by the Xsplat method, but placed at anterior positions by our method. As a result, in processing the sequentially re-arranged points from an unbalanced tree, our method can handle some finer points earlier to cull the points occluded by the finer points, but the Xsplat method may render some blocks at higher levels that will be occluded by finer points since the finer points are rendered behind.

- The block points can help culling points in groups. However, there is no parent-child relation between block points. Thus, every block has to be checked individually to know whether it is visible, even when it is inside an invisible block that has been previously checked. As our method embeds the parent-child relations between points in the list, if a point is checked to be invisible and its subtree is judged to be invisible also, its subtree will be implicitly culled to reduce visibility computation.

- Our new method can be also used to process out-of-core objects as it processes points in segments, without the need to load all points into the main memory at one time.

3 Sequential Layout

Without loss of generality, we will apply an octree to model objects in describing the new method, as done in Qsplat\cite{3} and the sequential point trees\cite{20}.

In constructing an octree for an object, a set of uniformly distributed points are sampled on objects, and are inserted into the octree as the leaf nodes. Then, position and normal values of the children are averaged to obtain the corresponding values for the inner nodes iteratively until the root node. By the octree constructed, the new method re-arranges the hierarchical points se-
quently in the following steps:

- According to the hierarchical levels, the points are