Literature Review

Justin Cossutti (CSSJUS002)

15 April 2014

Abstract

Visualising large 3D models and meshes is a long-standing problem in Computer Science and its applications. We have a look at two broad categories of approaches. First we cover progressive meshes, which rely on collapsing and expanding edges to iteratively refine a base model. We then take a look at solutions that use more common data structures such as octrees and bounding volume hierarchies to perform fast culling and level of detail rendering.

It was determined that while progressive meshes are a viable solution to the problem at hand, their complex nature will result in the project exceeding its time constraints and introduce dependencies between tasks. Such implications push the solution beyond the scope of this project, but is an area for further investigation. Thus, while octrees and bounding volume hierarchies are relatively simpler to implement, they have still proven to be an effective means of increasing rendering performance. Given this fact, such findings and implementations will provide a guide for our system designed later down the path.

Introduction

The Zamani Project is an initiative that aims to digitally document cultural heritage sites and architecture, specifically in Africa, for preservation, restoration and educational purposes. Such digital documentation is captured using laser scanners, which scan in surfaces of the sites and generate high-resolution point clouds. These point clouds undergo a number of processing stages to clean out unnecessary objects from the scene, remove noise and finally, convert the discrete points into a continuous surface for a 3D model – also known as meshing.

With the advancement of the technology used in laser scanners, the resolution of captured point clouds has increased greatly, with point clouds having grown from 50 million points per site to over 7 billion points per site (Rüther 2012). A popular example that is testament to the capabilities of modern scanners is the 3D model of Michelangelo's statue of David, consisting of 2 billion polygons (Levoy et al. 2000).

While this development aids the digital preservation of sites in greater detail, computers have a tough time processing and displaying such massive data sets. Due to current memory constraints, the meshing of large models often requires out-of-core processing techniques and the scanned site to be split into multiple smaller models for visualisation purposes.

While both problems have a negative impact on the task of digital preservation, we will focus on the latter – investigating techniques for visualising massive models.

Viewing the large, generated models is possible with current mesh processing tools such as MeshLab, however, not without severe lag during interaction with the model. Such interaction is important to be able to view models from multiple angles in real-time for use in virtual tours and fly-through videos, as well as the added convenience of being able to explore the full site without having to load multiple, smaller segments.

Previous work

Previous research on visualising large meshes and models can be roughly separated into two categories based on the complexity of the data structures they implement: those that make use of

relatively common data structures such as octrees, grids and bounding volume hierarchies and those that use more complex structures. The latter of which the majority of implementations are a form of progressive meshing.

The complexity of the implementation is important as it affects component dependencies during development. More complex implementations have a more cascading task dependency, resulting in a closely coupled project – which is not ideal.

Progressive meshes

Progressive meshing is a scheme first introduced by Hugues Hoppe (1996) that provides at its core, a loss-less level of detail (LoD) mesh representation via incremental levels of mesh simplification. It takes as input the original, high-resolution mesh and iteratively simplifies it using the edge collapse technique for a specified number of steps. During this simplification process, at each edge collapse it maintains a "history" of the operation performed and the resultant collapsed vertex. After each simplification, the simplified model has fewer "base vertices" each with a new history length of n+1.

After all simplification steps are complete, the result is the simplest "base model" which can be used for displaying a low polygon count representation of the original model. Maintaining a history of edge collapse operations at each step allows for the incremental, selective refinement of a particular segment of the model. The base model can be refined all the way back to the original, high resolution model smoothly and incrementally using a complementary edge expansion operator. The ability to refine only the subset of the model that is within the camera's view frustum is invaluable in terms of performance, only pushing a finer granularity mesh to the GPU of the segment which is visible.

Hu et al. (2009) later proposed such a system, offering view-dependant, parallel refinement of progressive meshes. They improved on the data structures such that the refinement operations could be performed on the GPU.

While progressive meshes appear to be a panacea to the task at hand, it has significant disadvantages. The proposed scheme is has a complex implementation that is not only would exceed this project's timeline, but would introduce many task dependencies between group members. The system would need to be broken into multiple components; one for simplifying and constructing the base model representation and another for the viewer that would incrementally refine and display the model.

Thus, progressive meshes, whilst likely a good candidate in terms of solving the problem at hand, is simply beyond the scope of this project, but is an area that can be investigated further in the future.

Volumetric data-structure implementations

With progressive meshing implementations being overly complex, we have a look at implementations based on octrees and bounding volume hierarchies.

One such implementation based on bounding volume hierarchies (BVH) is that proposed by Szymon Rusinkiewicz (2000) called QSplat. QSplat builds a BVH by recursively splitting a model segment along the longest axis and fitting the tightest bounding sphere around its child nodes. A parent node's splat colour and position is determined by the cumulative values of the child nodes. Leaf

node values are determined by the vertex's colour and are given a bounding sphere radius of 0. Only nodes that intersect with or are completely enclosed by the view frustum are considered for expansion into the child nodes. Distance and error metrics are used to determine whether a candidate node should be expanded for more detail.

However, QSplat is designed to be a point render and is not well suited for architectural sites where engravings and cracks are important elements of the model that might otherwise be lost.

QSplat is not without its merits though. It illustrates a number of techniques to form the basis of a BVH-supported LoD renderer, such as: recursively splitting a coarse parent node into finer-grained child nodes to provide more detail of areas nearer to the viewer and the storage and traversal of the hierarchy in a breadth-first manner to increase sequential disk reads. It also suggests that the breadth-first storage allows a parent to only require a single pointer to its first child, rather than the naïve method of having a pointer to every child.

Similar system proposals to QSplat, but ones that are not a point-renderers, are those by Peter Lindstrom (2003) and Ali Lakhia (2004). Both proposals expand upon the idea of using a BVH for fast culling, but also include multiple improvements. Firstly, both Lindstrom and Lakhia's systems make use of a rectilinear grid octree that includes the enclosed triangles and bounding volume information. Both also make use of individual rendering and loading threads to avoid the interface from stalling while the system loads the next frame.

Lindstrom's system performs many of its sorting and processing of vertices externally using temporary files to store intermediate results. This allows it to simplify the model and construct the octree using less than 8MB of memory. It is similar to the progressive mesh solution by the fact that each level in the hierarchy does not store a finer-grained version of the mesh, but rather, it contains a list of coarse triangles that are rendered if the node is collapsed. The contents of each node up until the leaf node is rendered.

In contrast, the nodes in any level of the hierarchy in Lakhia's implementation have no dependency on its parent or child nodes. Each node in each level contains the full list of triangles for the region the node represents. The disadvantage of this approach is the significant increase in the duplication of triangles between each level; each full level of the hierarchy makes up the entire simplified model for that level. It also proposes the use of a priority queue during the node refinement stage to maintain a constant target frame rate if so desired.

	Progressive Mesh	**Lindstrom's incremental octree**	**Lakhia's level independent octree**
Node history requirement	Requires base transforms to be applied first.	Requires parent nodes to be traversed first.	Each node has its own complete geometry information.
Exploitation of hardware acceleration	Poor. Geometry changes regularly.	Contents of visible cells / nodes can be stored in VBOs.	Contents of visible cells / nodes can be stored in VBOs.
Implementation complexity	Complex.	Medium.	Medium.
Geometry duplication	Low. Only base model and history of edge operations recorded.	Low. Successive levels in the tree only store triangles required for finer detail.	High. Each level contains the complete corresponding simplified version of the model.

Conclusion

Both Lindstrom and Lakhia's system designs are likely to be viable solution to the problem of visualising large architectural models. While progressive meshes also solve the problem, their complex data structures push them beyond the scope of this project. The octree-based solutions both use the increasing levels in the tree to store more fine-grained geometry in smaller, but greater number nodes. It is worth investigating whether the use of a constant-size grid (content number of leaf nodes) for all levels of detail – and only making use of the inner nodes for testing intersection with the frustum – incurs any overhead for testing more intersections than required. If not, nodes that require collapsing can simple have their contents replaced with the corresponding node in the next coarser-grained level.

The common theme to take away from previous research is the use of a recursively finer grid-based implementation which allows for fast rejection polygon visibility. Also important is the ability to read detail hierarchies in a sequential, non-random manner off the disk to reduce the disk's seek time and ultimately reduce the stall time when a user interacts with the viewer.

References

Hoppe, H., (1996). Progressive meshes. *In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (SIGGRAPH '96). 99-108. Available at http://doi.acm.org/10.1145/237170.237216.

Hu, L., Sander, P., Hoppe, H., (2009). Parallel view-dependent refinement of progressive meshes. *In Proceedings of the 2009 symposium on Interactive 3D graphics and games* (I3D '09). 169-176. Available at http://doi.acm.org/10.1145/1507149.1507177.

Lakhia, A., (2004). Efficient Interactive Rendering of Detailed Models with Hierarchical Levels of Detail. *In Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium* (3DPVT '04). Available at http://dx.doi.org/10.1109/3DPVT.2004.47.

Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D. et al., (2000). The digital Michelangelo project: 3D scanning of large statues. *In Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '00). 131-144. Available at http://dx.doi.org/10.1145/344779.344849.

Lindstrom, P., (2003). Out-of-Core Construction and Visualization of Multiresolution Surfaces. To appear in Symposium on Interactive 3D Graphics (SIGGRAPH '03). Available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.8515.

Rusinkiewicz, S., Levoy, M. (2000). QSplat: a multiresolution point rendering system for large meshes. *In Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '00), 343-352. Available at http://dx.doi.org/10.1145/344779.344940.

Rüther, H., Held, C., Bhurtha, R., Schroeder, R., Wessels, S., (2012). From Point Cloud to Textured Model, the Zamani Laser Scanning Pipeline in Heritage Documentation. *South African Journal of Geomatics*, *1*(1), 44-59. Available at http://www.sajg.org.za/index.php/sajg/article/view/20.