Increasing Scene Complexity - Distributed Vectorized View Culling

Andrew Routledge
Electronic Arts Black Box
andyr@ea.com

1. Introduction

Interactive Entertainment requires increasingly complex scenes at high performance. The apex of the Rendering pipeline is the Culling System which filters entities based on visibility and optimizes the load on the GPU. Culling Systems have standardized around spatial trees to produce a visible set of entities which are inefficient on current console hardware.

2. Outline

2.1 The Problem.

In order to increase visual complexity and improve performance, the culling system needed to be rebuilt. Expensive fragmented memory reads needed to be reduced and culling needed to be a concurrent process.

2.3 Solution

The solution was to move away from spatial trees with fragmented memory accesses and move to a flattened, localized, high density data representation and using console Vector units, linearly process the data stream and evaluate entity visibility. This technique significantly reduces memory access times, allows for pre-fetching of data into memory unit caches and using the Vector unit instructions to accept/reject entities 4 entities at a time.

With linear high density data, portions of the data can be sliced up and batched into concurrent job tasks yielding higher even higher performance.

3. Implementation

3.1 Views

Typical AAA titles have 10-15 unique views to produce a variety of advanced rendering effects. Each view requires culling and for each additional view the overall cost of the culling increases which in turn decreases scene complexity.

A typical view setup may include 1xPlayer View, 3xOrthographic Cascade Shadow, 6x Environment Map reflection views, and dependent on genre additional views could include a Rearview Mirror, multiple Parabolic Reflection and Spotlight Shadow views.

3.2 Vectorized Job

In the pipeline preprocess a spatial subsection of entities, typically encapsulating 200m², is collected and flat packed into a high density linear data stream containing series of entity reference id, transform and bounding box information.

The culling system, determines all visible spatial subsections, generates a series of job tasks and slices the data streams into sliced batches that are passed as inputs into a vectorized culling task for visibility testing. The culling job outputs a list of visible entities that is subsequently passed onto the Rendering Engine for processing.

3.3 Culling Methods

For maximum performance, dependent on view projection the culling method can be tailored to the dataset typical in a given view. Typically for a PlayerView a perspective Frustum Culling is performed, however for Orthographic and Paraboloid projections it is possible to optimize the culling method. For Orthographic a viewport clip method can be applied and for a Paraboloid View a radial distance and single hemisphere plane method is best.

In addition to projection optimal methods, occlusion planes and pixel size rejection methods were implemented. Pixel size rejection removes all entities that when projected cover less pixel space than a given threshold. Occlusion planes are 3 dimensional finite planes that are placed in a scene to occlude entities behind them.

4. Conclusion

This solution improved our visual complexity whilst significantly improving the performance of the Culling System on target consoles. This was important addition to the product as we could increase our scene complexity and add additional advanced rendering features whilst reducing the impact on CPU performance.