

FingER: Strategies on the Design of a 3d Real Time Engine

Aldo Filippini¹, Gabriel Acosta, Javier Filippini¹, Eduardo Fernández¹, Tomás Laurenzo¹,

¹Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay

1. Introduction

It is well known [ASH06] that state changes in the rendering step of a 3d scene produce performance penalties since pipelined graphics cards have to render cached data on each change. This generates serialization and misuses of graphic pipelines. Performance gains arises when a scene is rendered in big chunks of data with minimal state changes. This suggests a sorting of geometric elements in sets that share graphic states. Graph description systems found in generic scene descriptors such as OpenSG and OpenSceneGraph does not allow the application of this kind of sorting to dynamic sets of geometric data. They sort the whole scene data scene or static parts of it. When applied with visual optimization algorithms, which generally outputs dynamic sets of data to the rendering pipeline, this sorting is partially effective.

2. Objective

The objective of this project was to implement a 3d engine, called FinER, to study the performance obtained when sorting state changes in the very last step of the rendering process. The idea was to combine this sorting with a visual optimization algorithm that outputs dynamic sets of geometric data according to the viewer's direction. For simplicity we choose a cell-portal graph and Luebke's [LTH95] algorithm for visualization, but the sorting is independent of it.

3. Algorithm

The optimization algorithm comprises two main steps: an offline process takes every entity of our scene graph and groups them according to state changes defined in their visual characteristics. The same is done for every cell in the cell portal graph of the whole scene. A second, real time step, sorts the geometric sets sent to the pipeline before rendering. The offline process groups the geometric data in a structure called GRD. Every node in this structure holds geometry with the same visual characteristics. The real time

process works in combination with the visualization algorithm: it takes the output of the Luebke's algorithm and uses the GRD structure to perform a final sort according to state changes. This final sort results in a set that minimizes state changes. Finally the sorted data is sent to the graphic pipeline to draw the actual frame.

4. Results

Because graphic drivers optimize the rendering process when multiple passes are performed using additive blending, we tested the algorithm forcing only one rendering step. For the tests we used three scenes of different complexities filled with objects with different shading characteristics. The architecture was designed to generate multiple state changes while rendering so it was a good scenario for testing the algorithm.

Screen Resolution	Apply algorithm	Number of meshes	Number of polygons	FPS	% Benefit
1024 x 768	Yes	101	10261	66	—
1024 x 768	No	101	10261	61	8%
1280 x 1024	Yes	101	10261	66	—
1280 x 1024	No	101	10261	61	8%
1024 x 768	Yes	360	32089	57	—
1024 x 768	No	360	32089	50	14%
1280 x 1024	Yes	360	32089	56	—
1280 x 1024	No	360	32089	49	14%
1024 x 768	Yes	913	89347	22	—
1024 x 768	No	913	89347	20	10%
1280 x 1024	Yes	913	89347	16	—
1280 x 1024	No	913	89347	14	14%

Figure 1: Results for three different scenarios and screen resolutions in an AthlonXP 2000+, Nvidia Geforce 6600 GT

We can conclude that the proposed algorithm generate benefits between 8% and 14%.

References

- [LTH95] LUEBKE D.P, GEORGES C.: Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. *Proceedings 1995 Symposium of Interactive 3D Graphics*, 1995.
- [ASH06] ASHIDA K.: Optimizing the Graphics Pipeline *NVIDIA Corporation*, 2006.