Primary points

Present a simple but powerful solution to GPU Stalls

Avoid latency when obtaining result of occlusion.

Discard is done at geometry stage discarding vertices.

Avantages

No hardware extensions needed

Occlusion Result available in shaders

Objective

Show that in certain scenarios, a image space occlusion culling can be done completely at GPU without any hardware extensions, preventing GPU stalls.

#### Abstract

#### Per object, draw call granularity, non stalling.

#### Accessible by shaders

Culled Geometry is discarded at geometry culling stage

#### No GPU-CPU sync needed

#### Introduction

#### Nombrar walktrhough de silva

#### Hay hardware occlusion culling, pero que require hardware especial. Queries agregan latencia. Y Predicate rendering ( o conditional) no permite acceder a los resultados por los shaders.

* Granular: Uses Summed Area Tables to identify the ids of ítems that are not culled completely.
* We propose a implementation of gpu occlusion culling based on HIerarchial Z-Buffer

#### Related work

#### Hierarchical Occlusion Culling Zhang

#### Nombrar Predicate Rendering

#### Hardware occlusion culling

#### Granular visibility (ver buen resumen de esto en patch based)

#### Patch based, goes at primitive level

#### Asyncronous GPU Occlusion Culling

#### overview

* + - Nombrar las dos faces:
    - Decir que no se necesita conocer resultado del occlusion test
    - No vuelve a cpu

We divide the async occlusion culling method in three phases. The first phase is the Hierarchical Z Buffer (or Occlusion Map) construction in which the most representative occluders in the scene are rendered to a texture which will contain the coverage and depth information. This texture will be queried later in the second phase of the process, which is the visibility test, in where the actual potentially visible set of occludees is obtained. Based on the screen space bounded rectangle of the occludees, we perform the overlap and depth test to determine if the occludee is potentially visible or if it is completely occluded.

#### Hierarchical Occlusion/Z Map Construction

* + - Render HiZ map based on occlude simplification.
    - Fast rendering, render states.
    - Texture format.
    - Build mipmap chain
    - Determine limits of chain

#### Visibility Test

* + - Perform overlap test and Depth test
    - Hierarchical problem
    - Mip map determination
    - Results go to a texture

#### Objects Discard

* + - Once we have occlusion results
    - Send all potentially visible objects to GPU. It will discard the pipeline by doing geometry culling.

#### Implementation

Directx 9

C#

Geometry culling done by setting z to -1

#### Results

What gpu was used.

Time to build mipmap chain

Time to discard vertexes

Occlusion effectiveness

#### Conclusions

#### Future work

#### Webgl

Compute Shader

Level of detail obtained in shaders

Speheres instead of AABB

#### Aknowledgements

#### References

Daniel Es journal

Daniel Rákos - Rastergrid

Nick Darnell

Granular visibility engelhardt