FRR - Occlusion Queries

By J. Hugo Ballesteros

How to run

To run this program you need to install the viewer as per the NewViewer_52d9d92 README file.

Once installed, run `./GLarenaPL`. Press the spacebar and select open plugin option and navigate to the "occlusionQ" plugin. Select the "occlusionQ.pro" file and the plugin should load.

Implementation

This application is built for Linux in C++ using the provided viewer. The main body of the algorithm is in the `drawScene()` of the "occlusionQ.cpp" file. Depending on the selected optimisation, the rendering of objects is done differently. If Occlusion Queries is selected, we will perform a non-screen render of the bounding box of the objects and check if they were visible. If they were, we render the corresponding object normally, if not, we discard the rendering of said object. If View Frustum Culling is selected, then we use the representation of a bounding sphere, as it speeds calculations, to check if it is outside of our frustum. If the sphere is completely outside of our view frustum, then we discard that object and render it otherwise. When no mode is selected we perform a normal rendering.

How to configure

How to switch between optimisation algorithms

This plugin supports 3 modes: Normal Rendering, Occlusion Query Rendering (OQ) and View Frustum Culling Rendering (VFC). To use or stop using OQ press the `o` key. To use or stop using VFC press the `v` key. On screen there is information on which mode is currently active.

How to change the number of objects being rendered

In the file "occlusionQ.cpp" in the line 23 you will find a `#define copies` statement. Change the value in front of `copies` to an N of your choice. The number of rendered objects will be N*N.

Results

For the purposes of testing, the FPS of the application where monitored using the "default.obj" object provided in the viewer. As it is a fairly simple object this ensures that the bottleneck of the application is the number of objects and not any of the shaders.

Experiments

N = 20

When rendering 400 copies of the object in normal mode we have a steady 60 fps. This does not allow us to check if OQ or VFC are a significant improvement.

N = 50

When rendering 2,500 copies of the object in normal rendering we have around 35 fps. VFC is very angle (and view) dependent, as expected to be. As we have a large number of objects inside our frustum, the algorithm poses no significant improvement to the original, but if we start rotating the camera around the fps rise up to around 40/45. OQ however, have a much better performance. When using OQ we have an FPS average of around 50, and as we remove more objects from view we can achieve the 60 fps cap imposed by vertical sync.

N = 100

When rendering 100,000 copies of the object in normal rendering we have around 10 fps. VFC here raises the fps to around 40 very easily, and rotating it around we can achieve the 60 fps cap. This is due to the `zfar` of the view frustum culling a lot objects of the scene, greatly reducing the number of render calls and therefore raising the fps. OQ in this case falls short, due to the number of objects in the scene, rendering them twice, although one is much less costly, takes an amount of time that does not result in a very significant improvement.

Conclusions

We can draw a number of conclusions from this implementation of OQ and VFC: The first one is that OQ are simpler to perform, as the hardware is already prepared for their implementation and the OpenGL API has methods that facilitate their implementation and use. VFC however requires the creation of new classes and computation of the viewing frustum, which may not be trivial. Regarding performance, we can conclude that OQ are the better option when we have a moderate to large number of elements in a scene with a camera that keeps changing. When the number of objects starts being too large, as seen by the last experiment, the number of rendering calls is expensive and therefore bottlenecks the performance. When the number of objects is beyond large, then we conclude that we should use other techiques such as VFC. However, VFC specifically is very dependent on the frustum, and if we have a very large frustum we may end up not culling most objects, therefore imposing a decrease in performance as we are requiring more computations than a normal rendering.