Introduction

This projects implements frustum culling, stop and watch culling and CHC with KD-Tree culling. Also, I've added a temporal filter so occluded objects will remind occluded for a time. Also I've tested different temporal, visible pixels filters and different KD-Tree policies.

Although CHC accepts multiples acceleration structures, I chose KD-tree because is a popular and common acceleration structure and though it would be good for practice.

As a side note, this project has been developed along the SRRGE lab which creates a scene from a tile map file, so there are some shared files.

How To Run

This has been developed in a ubuntu 20.04 version, although it should be compatible in other OS. For this, you will need: freetype and cmake. Optionally eigen3 and tinyply, if you have problem with this libraries just remove "CalcLOD" class (CalcLOD.cpp, CalcLOD.h, and the include in main.cpp) and the CMake dependencies.

The *run.sh* bash script is a simple shortcut for easy execution. If this is the first time, run *run.sh init*, then you can use *run.sh build* or *run.sh run* to rebuild or execute it.

The script also accepts extra flags parameters, as *big_musuem*, *big_grid*, etc... and to choose the culling policy just add the flag $-c$ and a number from 0 to 5 corresponding to the *Scene.h CullingMethod* the default value is 3 corresponding to CHC plus frustum culling. There is a *install* parameter for the tinyply library but it's experimental.

An examples: $./run.sh\ build\ small\_grid\ -c\ 5$

Some Troubles

KD-tree are good, but for tiled scenes can cause a lot of problems, because each wall and ground is a independent mesh, they give a lot of noise and problems, so I had to ignore them and try to group them. Also the vertical axis has no information and I've just ignore it. And finally because the KDTree decide the occlusion in groups so although does less queries we have more false positive.

Next the bounding box of the walls are perfect equally so the occlusion query has a holistic behavior.

The pixel threshold is a good idea but by per se it's incomplete, remote objects are small whilst they are visible the threshold could decide not render it. And the same in the CHC group query, for this I decide to multiply the threshold by the number of elements.

Finally the temporal filter, occluding the objects without evaluating more than one frame, in the stop and watch algorithm it's quite visible even if it's 3-4 frames. A good solution for this is only apply to complex "secundary objects" and keep visible walls and floor all the time.

Result

Finally the results where evaluated in a Lenovo Ideapad 530S-14IKB[1], note the graphical controller is a Intel HD Graphics, in windows mode but full HD. Using the *small_museum* map during 30 seconds. The path consist in walking to the bunny, occluding the rest of them, walk back to the other room, use the wall to occlude one side, and finally a full view from above.

Because we are not taking advantage of the scene structure, the stop and watch gives better results. This can be easily seen in the "FPS vs Triangles" plots, where the stop and watch load is above the 30fps and the CHC is around 40 to 20 having a average of 44 fps and 35 fps respectively although both approach give more stability than just frustum culling.

Conclusion

- If you want a optimal consider first the application structure over popularity or experience.
- Magic numbers are a fast solution in implementation but for situations like the fps or number of pixels in a filtering process, a dynamic heuristic should be better. E.g.: distance, number of objects etc...
- If you don't know if your algorithm is improving use a naive approach to compare.
- As a extra conclusion, the triangles vs FPS gives a more visual plot than the classic FPS over time plot, at least for geometry bottleneck.

[1] `https://www.pccomponentes.com/lenovo-ideapad-530s-14ikb-intel-core-i5-8250u-8gb-256-ssd-14`

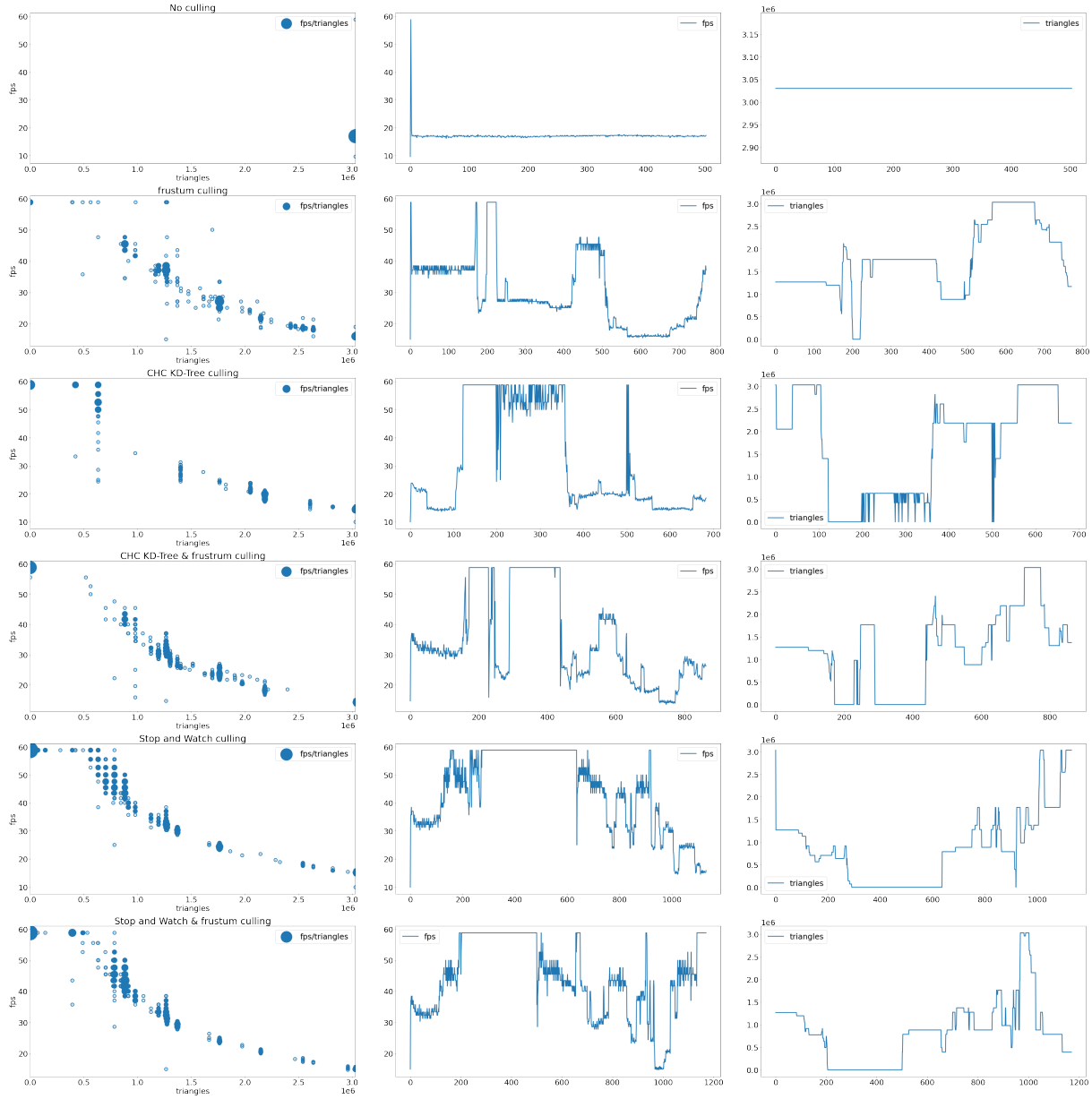

Figura 1: Different culling policies from top to bottom.
From left to right: FPS vs Triangles, the size of the dot represent the number of dots in that point; fps over time and triangles over time