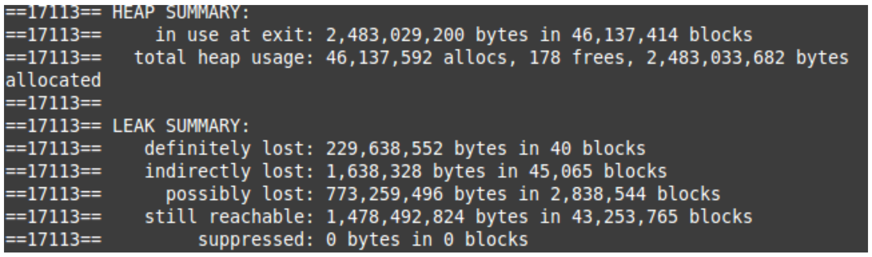
Optimization and Profiling

Gunnar Kollnitz

1 First analysis

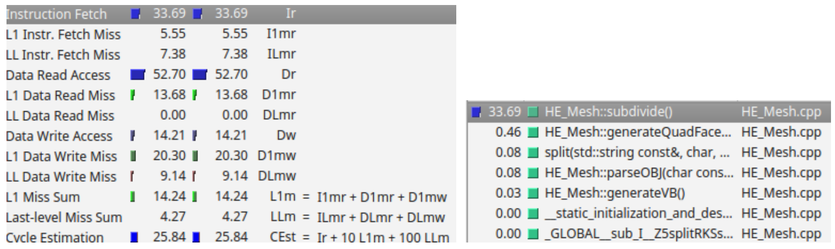
Running Valgrind, outputted no errors, though that was expected as it had already used it during the implementation of the Catmull-Clark Subdivision to debug and solve problems, and in the process cleared the code of what errors valgrind could find.



```
==17113== HEAP SUMMARY:
==17113==     in use at exit: 2,483,029,200 bytes in 46,137,414 blocks
==17113==   total heap usage: 46,137,592 allocs, 178 frees, 2,483,033,682 bytes
allocated
==17113==
==17113== LEAK SUMMARY:
==17113==    definitely lost: 229,638,552 bytes in 40 blocks
==17113==    indirectly lost: 1,638,328 bytes in 45,065 blocks
==17113==      possibly lost: 773,259,496 bytes in 2,838,544 blocks
==17113==    still reachable: 1,478,492,824 bytes in 43,253,765 blocks
==17113==         suppressed: 0 bytes in 0 blocks
```

Figure 1: Running Valgrind

During the first iteration of the suvdivision, it all ran in one function, subdivide().

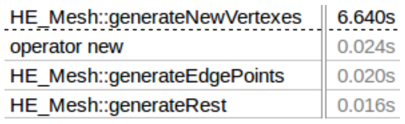


Instruction Fetch	33.69	33.69	Ir
L1 Instr. Fetch Miss	5.55	5.55	I1mr
LL Instr. Fetch Miss	7.38	7.38	ILmr
Data Read Access	52.70	52.70	Dr
L1 Data Read Miss	13.68	13.68	D1mr
LL Data Read Miss	0.00	0.00	DLmr
Data Write Access	14.21	14.21	Dw
L1 Data Write Miss	20.30	20.30	D1mw
LL Data Write Miss	9.14	9.14	DLmw
L1 Miss Sum	14.24	14.24	L1m = I1mr + D1mr + D1mw
Last-level Miss Sum	4.27	4.27	LLm = ILmr + DLmr + DLmw
Cycle Estimation	25.84	25.84	CEst = Ir + 10 L1m + 100 LLm

33.69	HE_Mesh::subdivide()	HE_Mesh.cpp
0.46	HE_Mesh::generateQuadFace...	HE_Mesh.cpp
0.08	split(std::string const&, char, ...	HE_Mesh.cpp
0.08	HE_Mesh::parseOBJ(char cons...	HE_Mesh.cpp
0.03	HE_Mesh::generateVB()	HE_Mesh.cpp
0.00	__static_initialization_and_des...	HE_Mesh.cpp
0.00	_GLOBAL__sub_I__Z5splitRKSs...	HE_Mesh.cpp

Figure 2: cachegrind report

So the subdivide function was split up into four parts, generateFaces(), generateEdgePoints, generateNewVertexes and generateRest. It was done to better be able to follow how well they did in comparison to eachother and to their future selves.



HE_Mesh::generateNewVertexes	6.640s
operator new	0.024s
HE_Mesh::generateEdgePoints	0.020s
HE_Mesh::generateRest	0.016s

Figure 3: Time used by each function during 7 subdivides

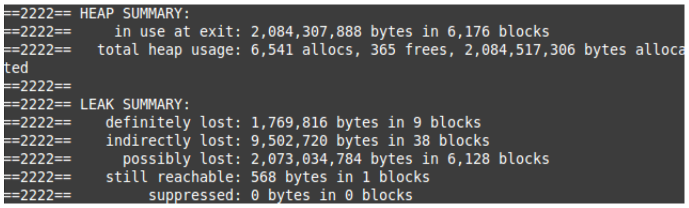
2 Optimization

If subdivision was to go past 7 subdivides at this point, the generateNewVertexes function needed to be improved, as that took up most of the time. Before optimizing it, it had O(n*n), and afterwards O(n) which is what was set as the goal. It was now able to reach 10 subdivides in just under 3.5 seconds, but that was not enough. It was already using arrays instead of vectors for its lists, so the next step was to implement a memorypool.

After the implementation of the memory pool the count was 3.18 seconds, which was a significant improvement, but still not quite enough. What was now focused upon was the how pointers were accessed and saved, and that had a surprisingly big impact on the time it took. Now the subdivision function was running at 10 subdivides at 2.918 seconds.

3 Final Analysis

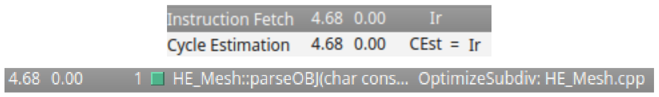
When running Valgrind now, the number of allocs as well as the leaks have been greatly reduced.



```
==2222== HEAP SUMMARY:
==2222==     in use at exit: 2,084,307,888 bytes in 6,176 blocks
==2222==   total heap usage: 6,541 allocs, 365 frees, 2,084,517,306 bytes alloca
ted
==2222==
==2222== LEAK SUMMARY:
==2222==    definitely lost: 1,769,816 bytes in 9 blocks
==2222==    indirectly lost: 9,502,720 bytes in 38 blocks
==2222==      possibly lost: 2,073,034,784 bytes in 6,128 blocks
==2222==    still reachable: 568 bytes in 1 blocks
==2222==         suppressed: 0 bytes in 0 blocks
```

Figure 4: Running Valgrind after optimization

After the optimization, the only fault found in the cachegrind report, is that of the .obj file parser, with the subdivision in itself being in the clear.

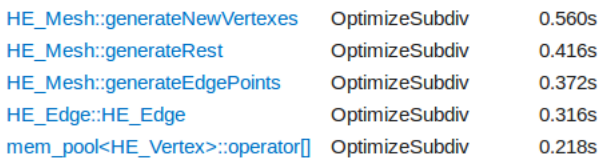


Instruction Fetch	4.68	0.00	Ir
Cycle Estimation	4.68	0.00	CEst = Ir

```
4.68  0.00        1 ■ HE_Mesh::parseOBJ(char cons...  OptimizeSubdiv: HE_Mesh.cpp
```

Figure 5: cachegrind report after optimization

When running VTune again, it's apparent that the functions now have a more similar time in comparison to before, where generateNewVertexes() was way ahead of the rest. And now, the memorypool makes an appearence.



HE_Mesh::generateNewVertexes	OptimizeSubdiv	0.560s
HE_Mesh::generateRest	OptimizeSubdiv	0.416s
HE_Mesh::generateEdgePoints	OptimizeSubdiv	0.372s
HE_Edge::HE_Edge	OptimizeSubdiv	0.316s
mem_pool<HE_Vertex>::operator[]	OptimizeSubdiv	0.218s

Figure 6: Time used by each function during 10 subdivisions