**Performance Analysis**

For the purposes of this submission, I did a simple performance analysis. I took my original code and modified it slightly so as to use global memory access wherever possible. Also, in the *raytraceRay* kernel, all shared memory variables had the \_\_shared\_\_ qualifier dropped, essentially turning them into automatic/register variables. Since there were at least 10 of them, I reckoned that, in conjunction with the register variables already being used in the original source, these might cause lesser number of threads to be scheduled in an SP. The modified code can be found in *raytraceKernel\_Slow.cu* and *intersections\_Slow.h*.

The scene was rendered on a system with a Quadro FX 5600 GPU (and a Xeon E5645 CPU) on Debug mode.

|  |  |  |  |
| --- | --- | --- | --- |
| Codebase | Round 1 (seconds) | Round 2 (seconds) | Round 3 (seconds) |
| raytraceKernel.cu & intersections.h | 80 | 79 | 79 |
| raytraceKernel\_Slow.cu & intersections\_Slow.h | 95 | 96 | 96 |

The results are evident. Caching global memory locally in each block through shared memory is definitely the way to go.

It must be said that in Release mode, though, the results are more neck-and-neck.

|  |  |  |  |
| --- | --- | --- | --- |
| Codebase | Round 1 (seconds) | Round 2 (seconds) | Round 3 (seconds) |
| raytraceKernel.cu & intersections.h | 12 | 13 | 12 |
| raytraceKernel\_Slow.cu & intersections\_Slow.h | 13 | 13 | 12 |