PERFORMANCE REPORT

The application profiled is a 3D raytracer which renders a scene composed of spheres. The raytracing algorithm implemented is based on Snell's Law and the Fresnel Equations.

After having implemented the algorithm iteratively, it was ported to an OpenCL kernel in order to exploit the parallel processing power of modern GPUs. Since each pixel's colour can be computed independently, there can be large benefits from calculating such colour for as many pixels as possible at the same time.

TEST MACHINE SPECIFICATIONS: *Packard Bell Easynote TM-86 – Intel Core i5 M430 – NVidia GeForce GT 320M*

TEST SETTINGS: *800x600 picture, 4 spheres, 3 lights*

RESULTS (R.T. ALGORITHM TIME FOR WHOLE PICTURE IN MS):

Anti-Aliasing Factor	*CPU - Iterative*	*GPU – 1 W.I. per Pixel*	*GPU – 1 W.I. per Img. Line*
1	417	224	214
2	1609	853	818
3	3610	1901	1825

TEST MACHINE SPECIFICATIONS: *Asus G75VW – Intel Core i7-3610QM Ivy Bridge – NVidia GeForce GTX 670M*

TEST SETTINGS: *800x600 picture, 4 spheres, 3 lights*

RESULTS (R.T. ALGORITHM TIME FOR WHOLE PICTURE IN MS):

Anti-Aliasing Factor	*CPU - Iterative*	*GPU – 1 W.I. per Pixel*	*GPU – 1 W.I. per Img. Line*
1	233	20	70
2	932	80	264
3	2093	180	598

Results Discussion

The results show the time spent by the raytracing algorithm to produce the entire buffer of pixels representing the image; it does not account the memory setup and the time spent in writing the image to file.

For each pixel, a ray is shot from the vantage point (the camera) to the pixel, and followed. Once the ray hits an object in the scene, colours are computed and the eventual refraction/reflection rays are computed too.

As it is possible to appreciate from the results, being able to compute more pixels at the same time enhances the overall performance and reduces the rendering time. The time taken scales up proportionally as the antialiasing factor increases due to the major number of rays shot compared to the situation without antialiasing (a factor of 1).

Surprisingly, on a machine with a low number of GPU compute units such as the GT 320M it can be beneficial to have less working items executing at the same time compared to having more working items, such as one for each pixel. This represents an interesting result as it shows that how the memory hierarchy and the GPU architecture are structured is a crucial factor in OpenCL applications: in fact, this result is explained by the fact that the GT 320M has a very limited amount of compute units and also a small amount of private memory is dedicated to each working item in the work groups, and therefore when too many work items are being utilised they find themselves in conflict for local and private memory resulting in a slower performance as the work items scheduler of the GPU has to coordinate the access to resources.

On the other hand, on a more modern and powerful GPU such as the GTX where resources are not as limited, there is a big improvement on performance between CPU and GPU, and the time taken to render one frame, with an antialiasing factor of one, is so low that the algorithm could be implemented on a real-time application. Although on the GTX the difference between the two different GPU executions is not extremely large, it proves the argumentations stated above.

In conclusion, it is possible to appreciate how the extremely high parallelisation derived from GPGPU implementations can improve the overall performance of an application on both dated and more recent machines, and more importantly how parallelisation in general can make a large difference in the performance of any application.

Execution:

Run the release exe from console, and use the following parameters (mandatory):

- N of alias, an integer from 1 to 4
- One of this set of modifiers:
    - –cpu: CPU mode
    - –ocl: GPU mode, with the following (one mandatory) modifiers:

- -d: one w.i. per pixel
- -l one w.i. per line