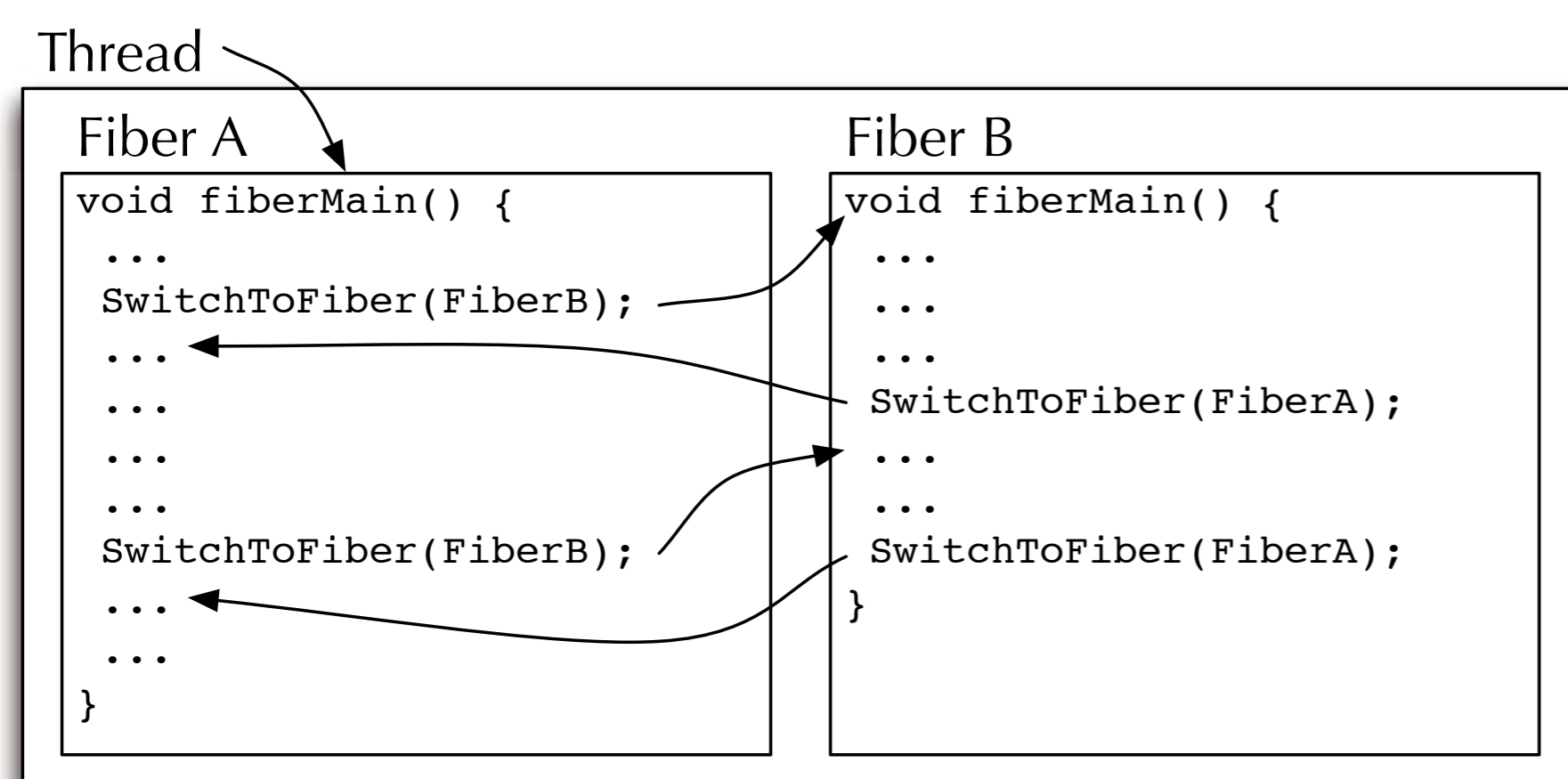


OFFLOADING RAY PROCESSING ONTO THE GPU USING COOPERATIVE WORKER THREADS

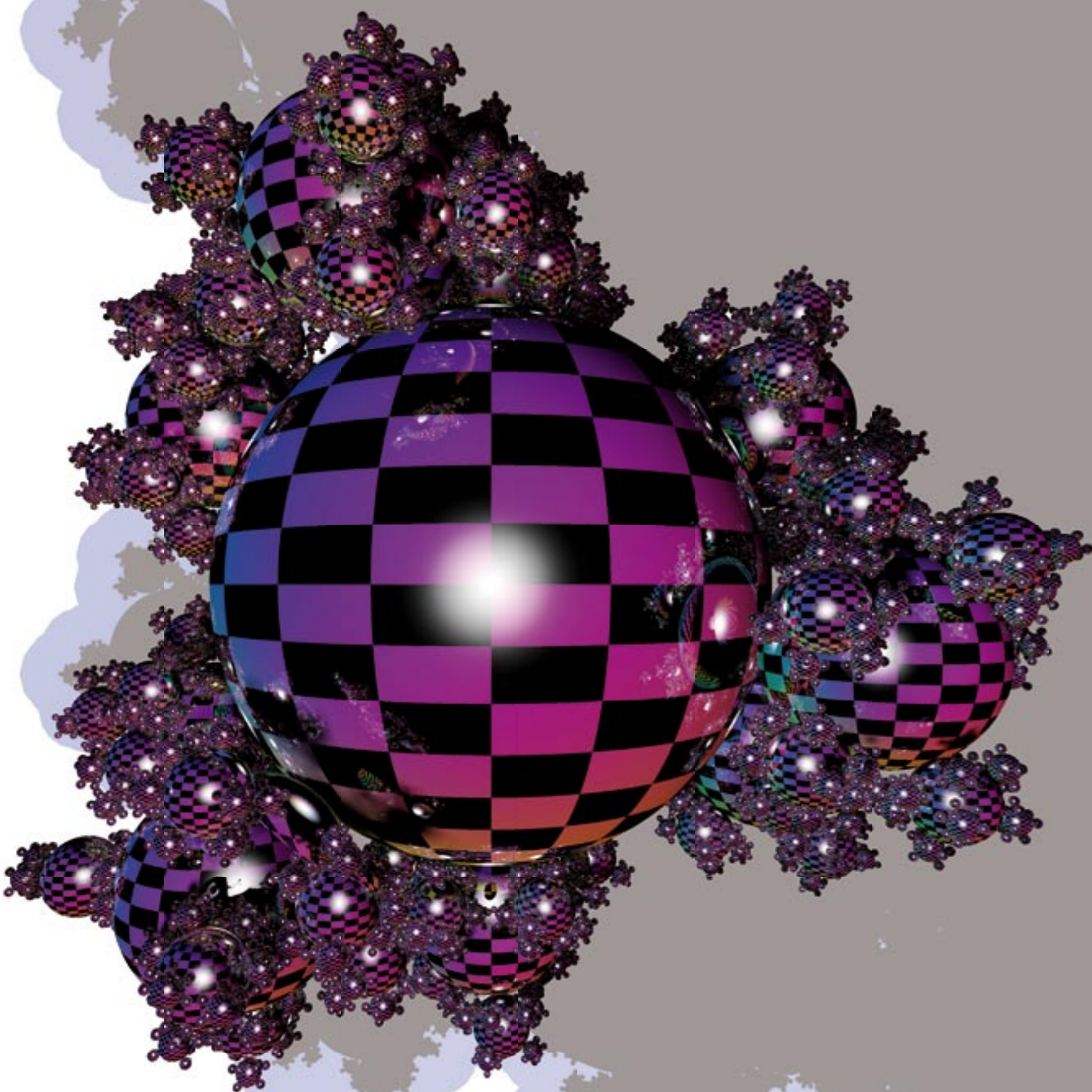
INTRODUCTION

- GPUs offer a great potential for speeding up floating-point intensive operations like ray processing.
- With PS 3.0 hardware it is possible to use spatial acceleration structures and optimized ray-object intersection tests in fragment programs.
- With the GPU as a co-processor for ray processing the CPU can spend more time on complex shading operations.

COOPERATIVE THREADS (AKA FIBERS, GREEN THREADS)



- Cooperative threads are managed and scheduled by the programmer and execute within one thread, where only one fiber can be active at a time.
- Fibers are extremely efficient and can be used in large numbers, because switching can be performed without going to kernel space.



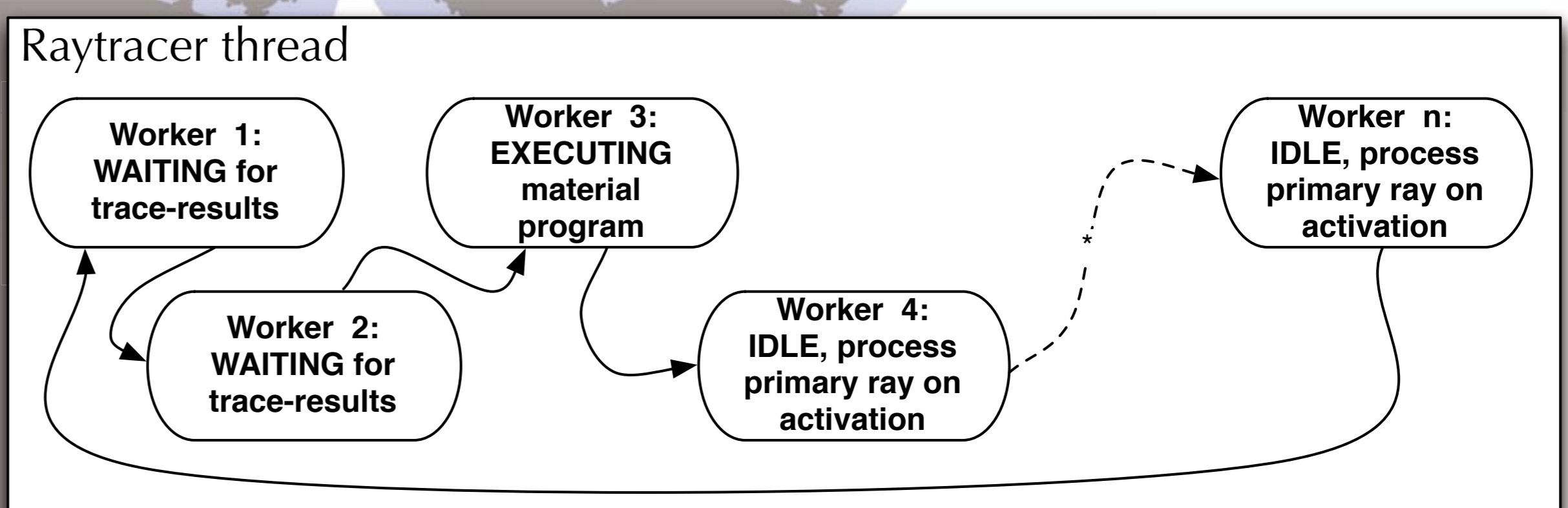
- Resolution: 4096x4096 pixels, 7381 spheres
- Phong lighting with one light and 128 shadow rays
- Reflections down to a recursive depth of 4
- 4096 workers, batch-size: 64x64 rays
- 880,5 million rays traced in 1005.35 secs
- 0.876 MRays/sec
- 78.168 secs spent waiting for gpu-data (7.78%)

PROBLEM STATEMENT

- The GPU is asynchronous to the CPU and optimized for throughput.
- Overhead for small batches is relatively large. In order to use the GPU effectively a large number of rays has to be submitted at once.
- Batching of rays is difficult to add to standard recursive raytracers, where only a single ray is processed at a given time and the involved trace-function is expected to return immediately.
- Furthermore reading back results may introduce a bubble in the command stream to the GPU which reduces performance.

METHOD

- Primary rays is processed in parallel by a set of cooperative threads.
- Upon calling a trace-function, the ray is inserted into the processing queue and the current thread is put on hold by passing control to the next "worker" in the set of threads.
- When the size of the ray queue exceeds a given threshold, a batch is issued to the GPU, which begins processing the contained rays asynchronously to the CPU and delivers the results in a texture.
- Once all cooperative threads are in waiting mode, the available results are retrieved from the GPU and the affected threads are woken up, where control flow resumes after the call to the trace-function.
- This way trace-calls still behave like in the traditional implementation, but multiple rays can be processed in parallel on the GPU.
- Material programs may therefore remain unmodified.



RESULTS

- The image was rendered on a computer with an AMD Athlon X2 3800+, 1 GB of RAM and an nVidia GeForce 7800 GTX; DirectX 9.0c.
- The scene was stored in a kd-tree with an alternating axis splitting scheme (x-y-z-x-...) and positioning of the splitting axis at the spatial mean. Cost-based spitting strategies have the potential for better performance.
- The CPU spent 7.78% of the rendering time waiting for results from the GPU. Retrieving results in a separate thread might increase performance by eliminating a blocking call to the Direct3D device in the main thread.
- The raytracer is a single-threaded application. Distributing work to perform raytracing on the second core will yield an additional speedup.

CONCLUSION

- Performing intersection tests and tracing of rays maps well to the SIMD-style nature of GPUs and allows the CPU to evaluate complex material programs.
- Research has to be done on accelerated retrieval of results from the GPU, usage of multi-core CPUs and the ideal configuration of the system (batch-size, number of workers, etc.) for a given image target resolution.

Stephan Reiter, stephan.reiter@students.jku.at