Interactive and offline rendering in Blender Cycles using MPI and Intel® Xeon Phi™ Offload

Milan Jaros, Lubomir Riha
VSB - Technical University of Ostrava
IT4Innovations - National Supercomputing Center
Ostrava, Czech Republic
Email: milan.jaros@vsb.cz

***Abstract*—In this paper, we describe interactive and offline rendering performed on multiple nodes of an HPC (High Performance Computing) cluster. Compute nodes can be equipped with Intel Xeon processors and also Intel Xeon Phi coprocessors. Rendering is utilized in Blender (open source 3D creation suite). We have modified the kernel of the Blender Cycles rendering engine and then extended its capabilities to support the HPC cluster. We call it CyclesPhi. The CyclesPhi supports hybrid MPI/OpenMP/Offload concepts. The original Blender Cycles engine has limited network rendering capabilities, which cannot be used on supercomputers. Our paper presents a basic algorithm for image rendering, decomposition tasks for application of parallel strategies, and basic collective communication routine using MPI methods. All of the presented techniques improve strong scalability of a cluster in both the offline and also interactive rendering.**

I. Introduction

One of the areas where massive exploitation of HPC occurs is computer graphics, particularly synthesis of image or rendering of virtual scenes. Rendering has many practical areas of application, e.g. in mechanical engineering, industrial design, or entertainment industry.

Rendering can be sped up by effectively employing more CPU cores of an HPC cluster or by using graphic accelerators. Beside, it is possible to also use coprocessors like Intel Xeon Phi to speed up the rendering.

The Intel Xeon Phi coprocessor contains 61 cores and in cooperation with CPU it can reach, in many cases, multiple accelerations. The peak performance of the top-of-the line Xeon Phi is over 1.1 TFLOP (10^{12} floating point operations per second) in double precision and over 2.2 TFLOPS in single precision. Xeon Phi can be programmed using both shared memory models such as OpenMP or OpenCL (provides compatibility with codes developed for GPU) and distributed memory models such as MPI. We have proved that the rendering is slower using OpenCL than OpenMP in [5].

The implementation presented in this paper has been developed and tested on the Salomon supercomputer located at IT4Innovations National Supercomputing Center in the Czech Republic. Salomon is equipped with 432 computing nodes, each with two Intel Xeon Phi 7120P coprocessors.

II. Background

There are several renderers which offer interactive rendering. But not many of them also offer utilization on a cluster.

One of the renderers, which are capable of both is Chromium [1]. It is a system for interactive rendering on the cluster of graphics workstations. Unfortunately, the Chromium project is no longer updated or maintained.

The Corona renderer offers fully-featured interactive rendering that runs solely on the CPU [2]. It does not support the cluster solution.

In the case of GPU-accelerated renderers, OctaneRenderer can be mentioned [3]. It offers fully interactive rendering. The Latest version of the OctaneRenderer provides integration to the beta version of the cloud renderer to acquire higher processing power.

We should not omit NVIDIA Iray Server/Client [4]. NVIDIA Iray Server is a software solution that provides distributed Iray rendering across networked machines. It uses common installation to deliver traditional offline batch rendering and interactive rendering to all NVIDIA Irays plugged in.

III. Used methods

Blender has a very nice rendering engine called Cycles, which is based on the path-tracing algorithm (uses Quasi-Monte Carlo and Sobol's sequence). It supports both the interactive and offline rendering. The interactive rendering offers possibilities to change materials, position of a light source, object geometry, etc. There is no need to manually restart the renderer as opposed to the offline renderer, which is therefore used mainly for rendering in backgrounds.

In the poster, we can see the example how path tracing works (see section Algorithm for image rendering). To avoid noise, we need a high value of samples, which is very expansive to calculate.

The first step to efficient parallelization is to decompose the task to smaller tiles and distribute them to allocated nodes. It is a big difference between the interactive and offline rendering in the task decomposition. In the case of interactive rendering, we need the maximum speed of one sample to calculate. When we move with a camera (walk through the scene) the new position of the camera is sent to all clients. To achieve maximum speed of the interactive rendering, we divide the original tasks to smaller tasks by the number of MPI ranks. On the other side, when we use the offline rendering, we think about load balancing. We divide the main task into rows, push

them to a stack and wait for the client's request (see section Task Decomposition in the poster).

We separate the main calculation of path tracing to external application called Blender Client (the workflow we can see in section Collective communication routine of the poster). We run two applications, Blender and Blender Client, on the cluster using mpirun command. That means the root (rank = 0) is original Blender and clients have a rank bigger than zero. We created a new MPI device and renamed the original GPU Compute to Acc Compute in the Blenders environment (see section Blenders environment).

IV. Conclusion

In section Benchmark, we improve the scalability of modified path tracing algorithm using MPI libraries. For example, when we use 64 nodes, we can almost achieve 60 frames per second by the interactive rendering. On the other hand, we can use more than one node by the offline rendering for a very difficult scene. For example, when we rendered the movie of Cosmos Laundromat, one frame of one difficult scene took almost 20 hours by the regular Blenders renderer.

The main idea of the interactive rendering is to create a web application, which will be similar to Iray Server/Client. In our case, we are using an open source variant of the Blender Cycles rendering engine. The Server is represented by CyclesPhi (Blender and Blender client), which runs on the supercomputer. In the web interface, we will see the current rendered image. We can then move the camera, and we will see the scene from a different point of view.

In the future, we plan to use explicit vectorization to speed up calculation of path tracing algorithm.

Acknowledgment

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science - LQ1602" and from the Large Infrastructures for Research, Experimental Development and Innovations project "IT4Innovations National Supercomputing Center - LM2015070" and by Intel Corporation from the project "Xeon Phi coprocessors: Test Cases on Image Rendering Curriculum - Grant Number: 23469721".

References

[1] http://chromium.sourceforge.net
[2] https://corona-renderer.com/features/interactive-rendering/
[3] https://home.otoy.com/render/octane-render/
[4] http://www.nvidia.com/object/iray-server.html
[5] Jaros Milan, et al.: Acceleration of Blender Cycles Path-Tracing Engine Using Intel Many Integrated Core Architecture, CISIM 2015, Warsaw, Poland, p. 86-97, September 2015.
[6] Frederik Steinmetz, Gottfriend Hofmann: The Cycles Encyclopedia
[7] https://wiki.blender.org/index.php
[8] Source available at: git@code.it4i.cz:blender/cyclesphi.git