Eurographics Symposium on Parallel Graphics and Visualization (2013), pp. 1–2
F. Marton and K. Moreland (Editors)

Visualization of Large Cosmological Data on Hybrid Systems

M. Rivi[†1] and C. Gheller[2] and T. Dykes[3] and I. Cant[3] and M. Krokos[3] and K. Dolag[4]

[1]Department of Physics, University of Oxford, United Kingdom
[2]ETH-CSCS, Lugano, Switzerland
[3]School of Creative Technologies, University of Portsmouth, United Kingdom
[4]University Observatory Munich, Germany

Abstract

The abstract may be up to 3 inches (7.62 cm) long....

Categories and Subject Descriptors (according to ACM CCS): D.1.3 [Programming Techniques]: Concurrent Programming—Parallel programming I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation I.6.3 [Simulation and Modeling]: Applications—J.2 [Physical Sciences and Engineering]: Astronomy—

1. Introduction

2. Splotch: a visualisation tool for astrophysical data

Splotch is an open-source, pure C++ code for effectively visualizing large-scale, particle-based datasets. It was designed for visualising N-body and Smoothed Particles Hydrodynamics simulations where the data are particles represented by a Gaussian distribution with compact support. However it has also been efficiently used with data defined by regular or adaptive refined meshes by the development of readers able to convert them in the Splotch data type. Several readers are provided supporting a number of popular formats for astrophysics, e.g. HDF5, ...

The rendering algorithm is a derivate of what in general is called volume ray casting and uses an approximation of the radiative transfer equation which, together with the application of perspective, gives the produced images a very realistic appearance.

The main strengths of Splotch are high quality of images and the support for large data volumes through an optimised usage of HPC architectures. In fact a parallel version of the code combining the OpenMP and MPI programming paradigms allow to exploit multi-core and multi-node systems [JKR*10]. Moreover a CUDA version of the code is also available to accelerate the computation on GPUs [RGD*14].

Another important characteristic of this tool is its high level of customisation, through a parameter file, to suit specialised scientific dataset. To help the user in the fine tune of the parameters required by Splotch a Previewer tool is available (see Section 3).

2.1. Algorithm overview

The operational scenario [DRGI08] consists in two main stages after reading particles from single or multiple data files . In the first stage, that we call *rasterization*, a preprocessing step can perform ranging, normalization, and apply logarithms to particle attributes if required. Then particle coordinates and their *smoothing length* σ_p (i.e. the standard deviation of the gaussian distribution $\rho_p(x)$ describing the particle) are roto-translated, with reference to supplied camera and look-at positions, and projected according to a given field of view. Finally particles falling in the scene are identified and assigned with RGB colour values via a colour look-up table or palette (which can be provided as an external file).

The final stage is *rendering*, where rays originating from each pixel of the image are cast along lines of sight, and contributions of all encountered particles are accumulated. The contribution of particles is determined by solving the radiative transfer equation:

$$\frac{dI(x)}{dr} = (E_p - A_p I(x))\rho_p(x), \qquad (1)$$

where $I(x)$ is the colour intensity at position x, E_p and A_p

† Corresponding author

describe the strength of radiation emission and absorption for a given particle for the three RGB colour component.

2.2. MPI+CUDA implementation

The MPI implementation [JKR*10] simply distributes chunks of particles among different processors, each performing independently a serial computation and producing a partial rendering. The final image is composed by a collective reduction operation and saved only by the root processor.

By enabling the usage of GPUs, each MPI tasks executes the CUDA version of the code described in [RGD*14] to render its chunk of particles. Summarizing, each task transfers its data (or sub chunks of it a time if necessary) to the global GPU memory, then each particle is processed by the Rasterization kernel according to a one-thread-per-particle approach. Further this kernel classifies particles in three classes according to radius defined by

$$r(p) = A(p)\frac{\chi\sigma_p}{S_{box}}N_{pix} \qquad (2)$$

where $A(p)$ is the factor of geometric projection to screen coordinates, χ is a factor of the order of unity, σ_p is the smoothing length of the particle p, N_{pix} is the image size in pixels, and S_{box} represents the computational box size. Small particles with $r \leq 0.5$ pixels can be efficiently handled by the GPU by assigning one particle per thread. Particles with $0.5 < r \leq r_0$ (r_0 being an appropriate threshold assigned empirically) are processed by the GPU efficiently by exploiting a tiling based strategy: images are split into a number of tiles, each processed by a CUDA block treating only particles whose centre completely falls inside the tile. To be able to process all these medium particles, each block stores its tile in the shared memory with a boundary of r_0 pixels (creating an extended tile version) so that all particles assigned to individual tiles are completely contained. For each block particles are accessed in groups of elements, then rendered sequentially with a single parallel operation: each image pixel associated with a particle is processed by a different thread. When all particles of a block are rendered, the contribution of the tile and its boundary are added to the final image stored in the global memory, by taking particular care to the handling of overlapping regions. This prevents any concurrent memory access avoiding potential image artifacts. Finally, large particles ($r > r_0$) are asynchronously copied back to the CPU and processed concurrently with the GPU. Once all calculations are completed, two partial images (one processed by the GPU the other by the CPU) are composed to generate the final rendering for each task. Such operation is performed by the CPU, once the GPU image has been transferred in a single copy operation.

This basic MPI+CUDA implementation requires each task exploiting a different GPU. However the HyperQ feature (REF) provided by the Kepler architecture can be enabled allowing more tasks sharing the same device. In this case the chunks of particles to be transferred to the device by each task is resized according to the number of tasks per GPU.

3. Splotch Previewer

Motivation, features and implementation.

4. Visualization of large scale structures of the Universe

Movie description ....

- *The dataset*:
- *Visualization properties*: choice of the parameters describing colours, intensity (according to some scientific motivations?)
- *Geometrical setup and Animation*: Sampling of the data to be investigated by the Previewer in order to prepare camera path and parameter file. Sampling 1 particle every 1000 is enough to have a realistic representation of the data distribution (see Figure 1). Notice that for some specific cases requiring a more accurate sampling, Splotch provides a feature called "booster" (just few lines about the booster...). Previewer usage to generate the camera path plotted in Figure 2. Number of frames and scene file description... Usage of Splotch to generate final images. Figure 3 shows ....

Figure 1: *Dataset visualisation by the Previewer.*

Figure 2: *The geometrical setup of the camera path.*

Figure 3: *A visualisation of .....*

5. Performance Tests

Platform description....

Scalability comparison between MPI and MPI+CUDA.

6. Conclusions

References

[DRGI08] DOLAG K., REINECKE M., GHELLER C., IMBODEN S.: Splotch: visualising cosmological simulations. *New Journal of Physics 10*, 12 (2008), 125006. doi:10.1088/1367-2630/10/12/125006. 1

[JKR*10] JIN Z., KROKOS M., RIVI M., GHELLER C., DOLAG K., REINECKE M.: High-performance astrophysical visualisation using Splotch. *Procedia Computer Science 1*, 3 (2010), 1775–1784. (Proc. International Conference on Computational Science 2010). 1, 2

[RGD*14] RIVI M., GHELLER C., DYKES T., KROKOS M., DOLAG K.: GPU accelerated particle visualisation with Splotch. *Astronomy and Computing* (2014). 1, 2