159.735 Assignment 4

Modelling Gravitational Lensing on a GPU

Gravitational (micro)lensing occurs when a foreground star (the lens) passes along the line-of-sight of a background star (the source). A consequence of Einstein's theory of general relativity is that light rays do not always travel in straight lines. A light ray will follow the natural "curvature" of space-time if it passes by an object that has mass. The result of this is that the foreground star acts as a lens on the background star. The lens star will produce two arc shaped images of the source star. More complicated image patterns are produced if there are several lens stars. In this exercise, you will see what they look like.

In gravitational lensing, the mapping between the source and the lens plane is given by the lens equation. If the two dimensional source, ζ and image positions, z, are expressed in complex form, the lens equation for N_{L} lenses with positions $z_{m,i}$ and mass fractions ε_i is

$$\zeta = z - \sum_{i}^{N_{\mathrm{L}}} \frac{\varepsilon_i}{\bar{z} - \bar{z}_{m,i}}$$

In principle, given the position of the source star, one can solve this equation to derive the image positions on the source plane. In practice, this is very difficult to do, as it requires one to solve a high order complex polynomial. A conceptually simple way is to use the technique of "inverse ray tracing". the lens plane is divided into pixels and a ray is shot from each pixel position. The corresponding position on the source plane is then calculated directly from the lens equation. If the ray shot from a particular pixel happens to land on the source star, then that pixel forms part of the gravitationally lensed image of the star. This works for simple lens configurations and very complex patterns resulting from many lenses.

The Assignment

On the Stream site, download a copy of `lensing.tar` and unpack this.

This package includes `lenses.cpp` which provides an implementation of the lens equation and a startup program `lens_demo.cpp` You need to complete the section of the code that generates the lens image. Try the sequential version for different lens configurations. You can use `ds9` to view the images.

Note that stars are not generally of uniform brightness across their discs. They appear darker towards the edge. You can include this ”limb darkening” by weighting the lens image pixel according to where the ray lands within the source star disc.

$$f = 1 - \lambda(1 - \mu)$$

where

$$\mu = \sqrt{1 - (r/r_{\text{star}})^2}$$

and r is the distance from the center of the star where the ray lands, r_{star} is the radius of the star, and λ is the limb darkening coefficient (typically $\lambda = 0.5$).

Now write a CUDA implementation that runs on the GPU. You may use the GPU cards installed in the lab workstations, or any other card on which you have access.

Submission

Please submit your C or C++ source code together with a report where you should address at least the following.

- State the GPU card that use implement your assignment. Give the number of processing units on that card and any other information that is relevant.
- Describe how you implemented your solution to the inverse ray tracing problem on the GPUs. Marks will be awarded for thoughtful answers that demonstrate your understanding of the GPU architecture.
- Try running your program for different problem sizes, ie try generating lens images of different resolutions and for different sizes of the lensing system. Comment on the performances you see.

Due date: TBD

This assignment is worth 20% of your final grade