LIÈGE université
Sciences Appliquées

University of Liège

Parallel Programming (INFO9012-1)

A parallel ray tracer

Authors :
Louis Hogge s192814
Juliette Waltregny s194379

Professor : P. Fontaine
Year : 2021-2022

1 Prior informations

All experimentations were made on ms8xx machines and these have 4 real cores. One thread was used for displaying images and dealing with user interactions as well as for managing the computation of images. Three threads were used for the proper computation (number of real cores in ms8xx machine minus one). All critical sections (access to or read/write on shared memory) were protected thanks to mutex.

The given sequential version has a score of around 6 fps and the phase 1 version has a score of around 11 fps.

2 Phase 2

To implement phase 2, the *main* file had to be modified :

First, two queues were created. A basic one for the angles and a priority one for the images. We simply chose to create a counter that increment each time an image is added to the priority queue and take the counter value as priority.

Then, we created a manager thread. This one updated angles (user request or automatic movement), sent angles to the queue, got images from the priority queue and displayed them. In order to rotate the logo at a frequency of one rotation per minute we used this formula : ”$angle\_logo+ = angle\_logo >= 359.? - 359. : 6./fps;$”.

After that, we made a thread pool to compute images. It got angles from the queue, computed images thanks to the *render* function and sent images associated with their priorities to the priority queue.

Later, we had to limit the size of the basic queue and priority queue so as not to drown out user requests. If we did not, too many angles and images were put int the queue before those of the user.

Finally, we added two boolean variables, one for the manager thread and one for the thread pool in order to be able to stop their execution loop and properly quit the program.

The phase 2 version has a score of around 17 fps.

3 Phase 3

To implement phase 3, new variables had to be added to the code of the phase 2 :

First, the variables *centerRedBall* and *radiusMirrorBall*, both of float types, needed to be added to be able to respectively manage the position of the center of the red ball and the size of the radius of the mirror ball. To make the red ball oscillate every 5 seconds, the *centerRedBall* variable was incremented/decremented by $\frac{2}{5}$ and divided by the number of frames per second. The *radiusMirrorBall* was incremented/decremented by $\frac{2}{3}$, and also divided by the number of fps, in order to make the mirror ball oscillate every 3 seconds. All these manipulations were made in the *animate* section, in the same mutex as the one locking the computation of the rotation of the Uliège logo.

Then, To know if the variables needed to be incremented or decremented, two boolean values, *isGoingUp* and *isIncreasing* were introduced. *isGoingUp* informed if the red ball was going up. If it was the case, then *centerRedBall* was incremented, otherwise it was decremented. The same reasoning was applied to the mirror sphere, with *isIncreasing* checking if the sphere was growing and incrementing it.

Finally, we needed to add the two variables in the *render* function and assign the values of the center/radius to them.

The phase 3 version has a score of around 16 fps.