Lab session 1

Threads

This session introduces the usage of threads in C++. Threads are a useful tool to exploit the parallelization features of our processors which can have an important role in several aspects within the architecture of a videogame. A couple of examples are subsystems (which can be parallelized performing their job independently from each other, just to export their results at the end of each iteration and share their computations with the rest of the system) and tasks (independent pieces of very specific code that fulfill some goal in an isolated way). A good example of a task can be the preload of game resources at boot time.

Focusing on the latter goal, in this session, we will implement a task manager that will allow us loading textures in the background (yes, without blocking the application) while a loading screen with a dynamic progress bar shows.

ModuleTaskManager

This is the TODO list:

- ***init()***
  - It must create a new thread in each position of the threads array (we will have *MAX_THREADS* worker threads). Each thread will execute the method threadMain().
- ***threadMain()***
  - It contains the body of each worker thread. Its job is entering into an infinite loop that will sequentially perform these steps:
    i. Retrieve a task from the queue *scheduledTasks*.
    ii. Execute the task.
    iii. Insert the task into the queue *finishedTasks*.
  - Be careful at points i) and iii), as they are accesses to shared objects (*scheduledTasks* and *finishedTasks*). Take also into account the proper use of the condition variable to wait and block until there are new tasks to execute.
- ***update()***
  - Will dispatch all the finishedTasks to their owners and clear the finishedTasks queue. Protect the shared access to finishedTasks.
- ***scheduleTask()***
  - It inserts the passed task into the *scheduledTasks* queue. Mind the shared access to *scheduledTasks* and notify some thread there are tasks pending.
- ***cleanUp()***
  - Notify all threads they must finish and join them all.

Note that, except *threadMain*, the previous methods *init*, *update*, *scheduleTask*, and *cleanUp*, are methods executed from the main thread (as you are used to). However, *threadMain* is executed repeatedly in a few separated threads. For that reason it is extremely important that the crítical sections of code that access shared resources by different threads are properly protected with mutex objects.

Deadline:

The delivery of this exercise is due by the first week of October during the lab session.