OPTIMIZATION

The assessment of this project was made by using the Visual Studio profiler.

The first time the project is compiled and ran, we can see that it takes too long to run. In 5 minutes, it only computed 7 steps. So, as a first assessment it is not runnable. By looking at the CPU usage of the functions we have realized that the one that produces the bottleneck is the function “particle\_at()”. It is as in the function it makes a for loop to access all the elements in a list, whose memory is not contiguous. Furthermore, all the project is done by using doubles when floats do the same job and waste less memory and resources. Besides, the project properties are not set so that the project is optimized and runs faster while being more efficient.

Captura de pantalla de computadora

Descripción generada automáticamenteCaptura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

* Change nº 1

As a first change, we switch the particles from being a list to being a vector. We did that as it was the main reason why the project was taking so long to run. Of course it was also due to the particles\_at() function, which every time that it was called was doing a for loop until the particle was found. If you take this and the fact that it is a list and the objects inside it are not contiguous in memory into account, you have the reason of the low runnability of the project.

Summarizing, the first change that was made was changing the particle pool from a list to a vector and completely discarding the particles\_at() function by dereferencing directly the memory inside the vector.

The change was quite notorious as the project was running better than before and also, just by looking at the profiler the change was visible. As you can see below, the kernel usage went from 43.9% to just 14.3% just by making this change. This was because of the memory accessing that the list was making each time the function particle\_at() was being called. Now Captura de pantalla de computadora

Descripción generada automáticamentethe functions that waste more CPU are Captura de pantalla de computadora

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteCaptura de pantalla de computadora

Descripción generada automáticamenteCaptura de pantalla de computadora

Descripción generada automáticamentefunctions that are written in the project code.

The change in fps is quite notorious, in both cases. On top we have the case were the particles are at max count and below we have an average particle count.

* Change nº 2

As the second change we realized that no project properties were set so that the whole project was optimized.

The properties that we changed are in the C/C++ and Linker fields:

In the C/C++ field this are the changes that we made:

* **Optimization**: Disabled -> Maximum Optimization (Favor Speed)
* **Inline Function Expansion**: Disabled -> Any Suitable. With this option we can speed up the execution by eliminating function call overhead and is useful for functions that are frequently called.
* **Enable Intrinsic Functions**: No -> Yes.
* **Favor Size or Speed**: Neither -> Favor fast code. The same as with the previous option, we specify that we care more about execution speed than code size.
* **Whole Program Optimization**: No -> Yes
* **Struct Member Alignment**: 2 bytes -> 4 bytes. This way the alignment adjusts better to the variable type we use like int or floats (one of the next changes is going to be to change all the doubles to floats).
* **Enable Parallel Code Generation**: No -> Yes
* **Enable Enhanced Instruction Set**: Not Set -> Streaming SIMD Extensions (SSE). This is helpful since one of the other optimization changes is going to be based on SIMD’s SSE
* **Floating Point Model**: Precise -> Fast. Here we care more about floats being fast and not precise. The simulation uses a lot of float arithmetic, and this option saves a lot of CPU usage.
* **Enable Floating Point Exceptions**: Yes -> No. This option must be set in order to use the previous one.
* **Debug Information Format**: Program Database for Edit and Continue -> None. For the optimization options to be set, this must be disabled.

In the Linker field this are the changes that we made:

* **Link Time Code Generation**: Default -> Use Link Time Code Generation. This option must be set in order to use the Whole Program Optimization one.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Here we can see how the fps have gone up, after the changes were made.

* Change nº3

For the third change we realized that the entire project was using doubles. It is not completely wrong to use them but for this simulation floats are better. They waste less space in memory and doing computation with them is much cheaper. This project does not need that much precision as to need the use of doubles, floats are more than enough.

Apart from that, in the properties we change that the struct member alignment was changed from 2 bytes to 4 bytes which is exactly the size of a float. So, memory wise it is a much better idea to change from double to floats.