Report of the second submission

Architecture for the final submission

We mainly keep the structure of the figure 11b. The only difference is that we don't have a connection between graphic.c and utilitaire.c. We don't need to include the header graphic.h in utilitaire.c as the functions of graphic are general and complete enough to be used directly in our four modules: projecteur.c, reflecteur.c, absorbeur.c and photon.c. Concerning the communication between photons and the other modules to detect collisions, we think the best way is to do it over module.c, as we already do for the verification. Regarding the projectors, which produces new photons, we can use get- and set-functions. So the structure of the program stays clear, even if the performance may decrease a little bit. For the communication we use a simple vector structure between the elements-modeles and the module.c.

Data structure for the final submission

As requested, we now use dynamic allocation memory. We allocate tables for the projectors, reflectors and absorbers and use a linked list for the photons. In the first submission we just used tables of structure, with a defined length.

We consider that the table (of structure) was the most appropriate solution for the projectors, reflectors and absorbers for a few reasons: First, it's practical and easier to use. Moreover, it's simpler to directly access to one element of the table (which is interesting when we want to delete the selected item). Its main flaw is when we have to change its size. If we have to remove one element it's simple: we delete it, replace it by the last element of the table and, after eventually calling realloc (to free the memory), we're done. If we have to add an element, some issues can theoretically occur when the memory is full and it can take a "while", when the whole table has to move in the memory to get more space, but that's more of an internal problem, which doesn't really affect the way how we handle it. We also need to mention that when we draw manually our elements, we generally don't draw huge amounts of elements. Besides the fact that we can only delete and create them manually guarantees a low speed of modification, i.e. the size of our tables won't change radically, which is fortunate.

We consider that the linked list is more appropriate for the photons, because the number of photons can grow or drop extremely fast, which would be annoying to allocate them in tables. With linked lists, the creation and suppression of photons are managed automatically and goes very quickly. Furthermore, we can handle each photon completely separately. Finally, at each step and for each action concerning them, we'll have to run through all the linked list so the order in which they are ranged doesn't matter.

List of functions of modele called in main

1. **modele_lecture:** reads a file and inserts the data in the tables and linked list.
2. **modele_verification_rendu2:** makes all the verification
3. **modele_update:** for 3rd submission, will update the simulation
4. **modele_find_nearest:** for 3rd submission, will find and select the nearest element
5. **modele_delete_outside:** for 3rd submission, will delete all the photons that are not in the range of sight
6. **modele_delete_selected:** for 3rd submission, will delete the element that is selected
7. **modele_draw_element:** for 3rd submission, will draw the desired element where the user wants
8. **modele_statistic:** asks and inserts the number of each element in a table
9. **modele_save:** saves the current state of the simulation in a file with the desired name
10. **modele_delete:** deletes everything and frees all the memory
11. **modele_error_cl:** delegates the posting of a command-line error from main to the modele*
12. **modele_draw:** draws all the elements contained in the different modules

*we could have just called a printf directly in main for the error-message, but it's better to handle all errors in the same place, also in case we want to change the way how error-messages are shown (like e.g. \n\n instead of just one \n after the message)