Année universitaire
2019-2020

DUT Informatique
Matthieu TRINQUART

Rapport de projet
de fin d'année
42 jours de projet

Département Informatique
Site d'Arles

Tuteur :
E. Rémy

Résumé du projet

Le projet consistait à diviser une scène OpenGL en plusieurs fenêtres sur plusieurs machines dans un réseau local pour permettre un chargement et des interactions plus rapidement. Il devait être possible de tourner et de zoomer autour de l'objet. La librairie SDL devait s'occuper du fenestrage. La gestion du multi processus dans un réseau devait être fait grâce à OpenMPI. Le rendu 3D est géré avec OpenGL et enfin les fichiers 3D (obj ou OFF) sont lus grâce à la librairie OpenMesh. Les interactions avec l'objet devaient être faites de manière synchronisée entre chaque fenêtre. L'application devait être développée en C++ et une documentation sous Doxygen devait être faite.

Mots clés décrivant le projet

Rendu 3D-réseau-serveur SSH-fenestrage-rendu distribué-fenêtre SDLOpenGL-OpenMesh-SDL-OpenMPI-multiprocessus

Remerciements

En premier lieu, je remercie mon professeur référent, M. Raffin qui m'a été d'une aide précieuse tout au long de la conception de mon projet en répondant à toutes mes interrogations.

J'aimerai aussi remercier mon tuteur de stage M. Rémi qui a répondu à toutes mes questions sur l'avenir de mon stage durant cette période particulière.

Je remercie également M. Grossi qui a bien voulu me prendre comme stagiaire même si malheureusement la situation particulière de cette année a rendu le déroulement de mon stage impossible.

Je remercie aussi l'équipe pédagogique de l'IUT informatique de Arles pour sa qualité d'enseignement.

Je remercie Mme Gaston qui s'est occupée de ma convention de stage et du dialogue entre l'entreprise ICUBE qui devait m'accueillir comme stagiaire.

Table des matières

I) Introduction

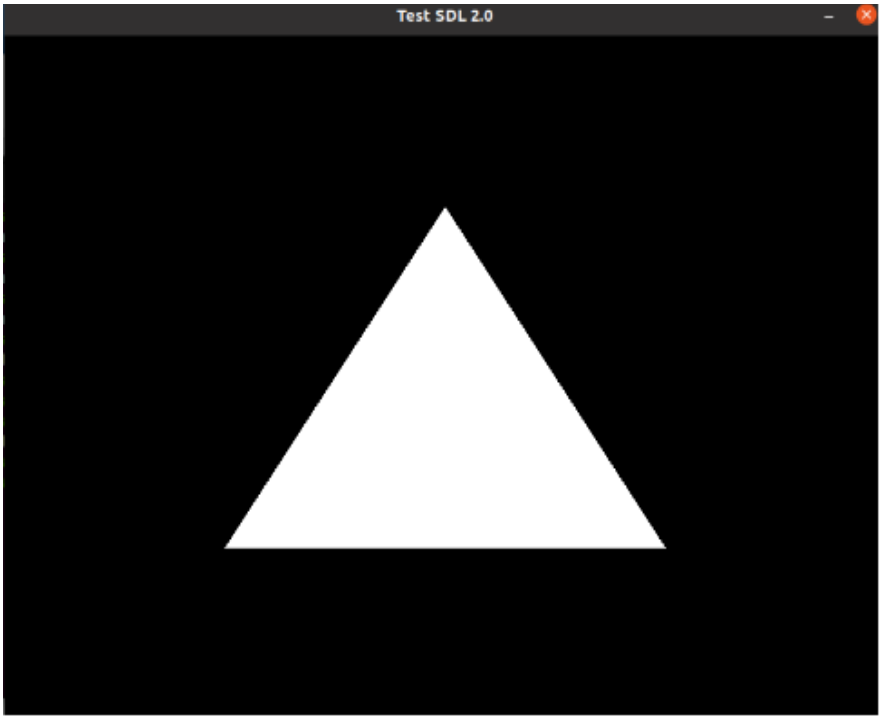
Durant ma deuxième année d'IUT, l'entreprise ICUBE m'a accepté comme stagiaire. Mon sujet de stage devait être de développer une API de contrôle des enceintes climatiques d'un laboratoire en python. L'API demandait une communication en TCP/IP dans un réseau local. Mais dû à la crise sanitaire, mon stage n'a pas pu avoir lieu. En effet, le stage ne pouvait pas s'effectuer en télétravail. L'IUT m'a donc fourni un projet de fin d'année en remplacement de mon stage. Ce projet de fin d'année d'IUT consistait à faire un rendu OpenGL distribué sur plusieurs machines dans un réseau local. La réalisation de ce projet a duré 42 jours. Il a débuté le 18 mai pour se terminer le 1 juillet 2020. Le projet demandait des compétences en gestion d'environnement 3D avec OpenGL, de la gestion de calculs en parallèle avec l'utilisation de OpenMPI et enfin du réseau avec la gestion de serveur SSH et de la création d'un cluster entre plusieurs machines. L'objectif de ce projet est de réussir à charger et tourner autour d'un objet 3D plus rapidement et de façon plus fluide que si le rendu soit fait de façon séquentielle. L'utilité d'un tel programme est de pouvoir charger et manipuler un objet 3D en utilisant la puissance graphique de plusieurs machines se situant dans un réseau local et donc pouvoir manipuler des objets 3D de grosses tailles de façon rapide et fluide. Ce sujet pourrait servir dans un contexte professionnel, si une entreprise recherche à charger ou à manipuler rapidement un objet 3D en utilisant toutes les capacités de calculs des machines dans son réseau local.

II)Présentation du travail accompli

1)déroulement du projet

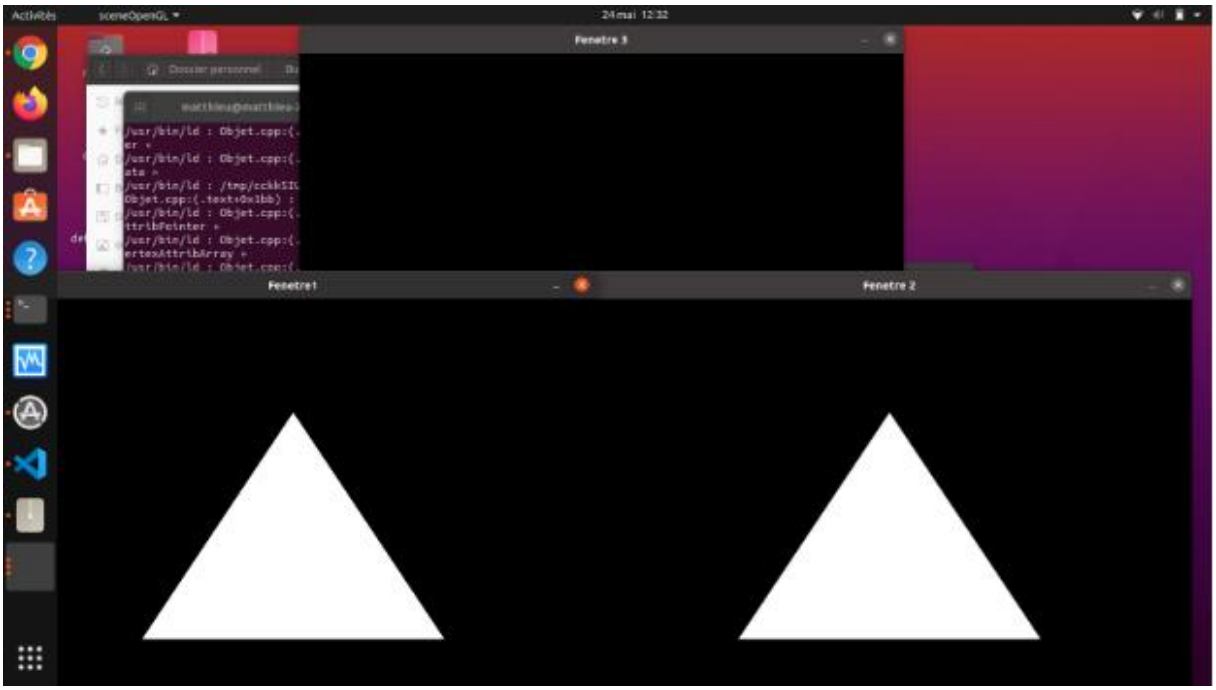
Le premier problème à résoudre était d'abord de réussir à lire un objet 3D avec openMesh. J'ai donc élaboré un premier programme qui me permet de lire un fichier contenant un objet 3D (format obj, OFF) et d'ensuite pouvoir afficher dans le bloc de commande tous les points qui constituaient l'objet 3D à lire. Cela me permettra d'utiliser ce programme plus tard pour lire et charger un fichier 3D.

Après cela il fallait que je commence à créer ma première fenêtre SDL avec un rendu OpenGL dedans.

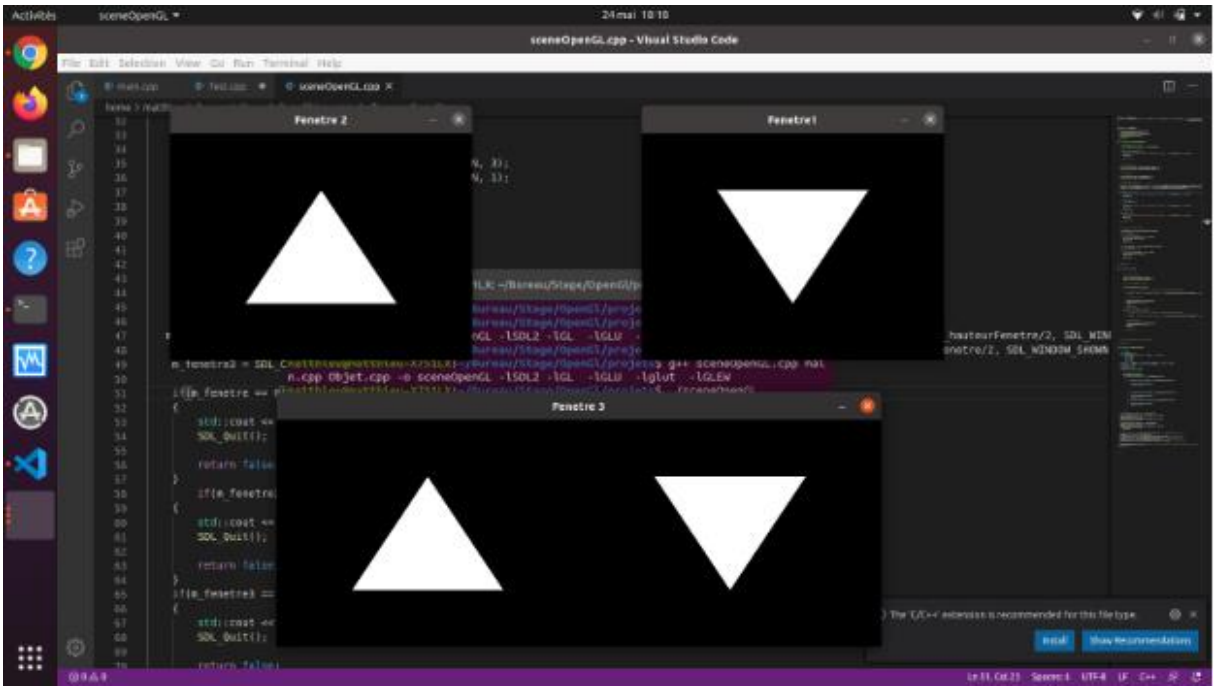


Mais à ce moment là du projet une première complication est apparue. Je n'arrivais pas à faire fonctionner les outils OpenMesh avec d'autres librairies comme OpenGL ou SDL .Afin de ne pas perdre de temps dans la continuité de mon projet j'ai utilisé ce rendu openGL en forme de triangle, ce qui symbolisera un objet 3D à charger et à distribuer sur plusieurs fenêtres.

J'ai dû ensuite créer plusieurs fenêtres SDL afin de pouvoir afficher une partie du contenu de l'objet 3D dans chacune des fenêtres et une dernière fenêtre qui doit assembler les bitmaps de chaque partie de l'objet pour pouvoir afficher l'objet 3D dans son ensemble.

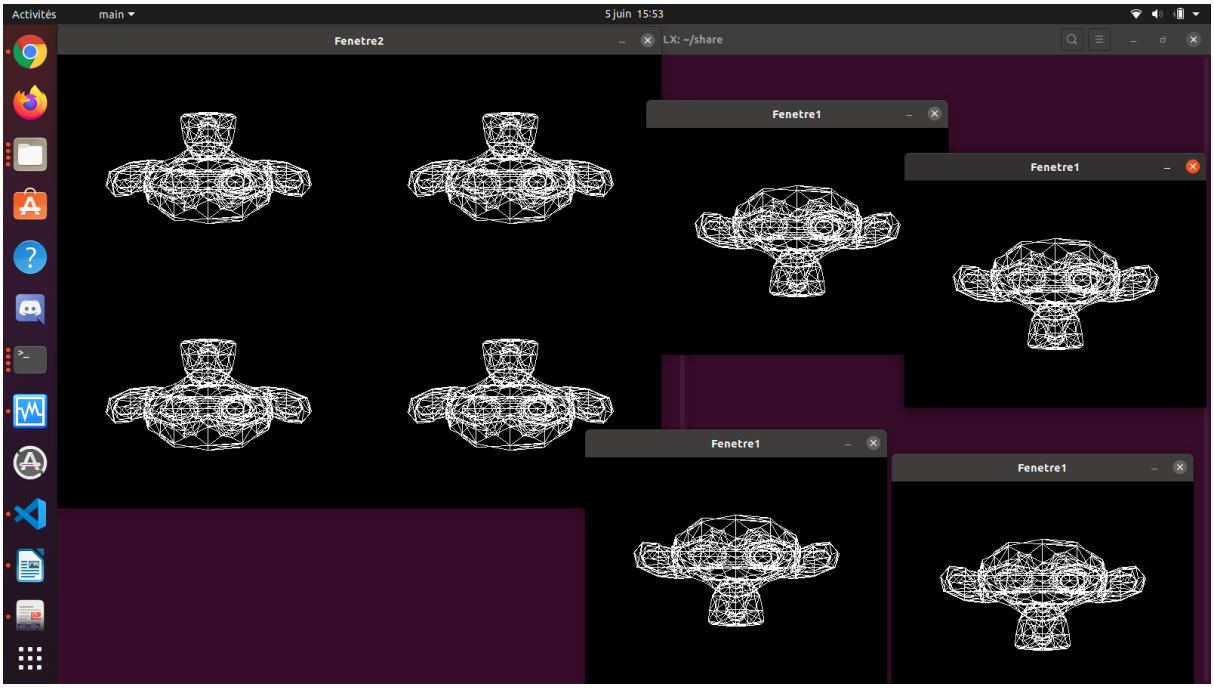


J'ai donc dû ensuite pouvoir lire la bitmap de chaque fenêtre et l'afficher dans la 3ème pour pouvoir avoir un rendu partagé. Le triangle de droite symboliserait la partie de droite d'un objet 3D et le triangle de gauche symboliserait la partie de gauche de l'objet 3D.

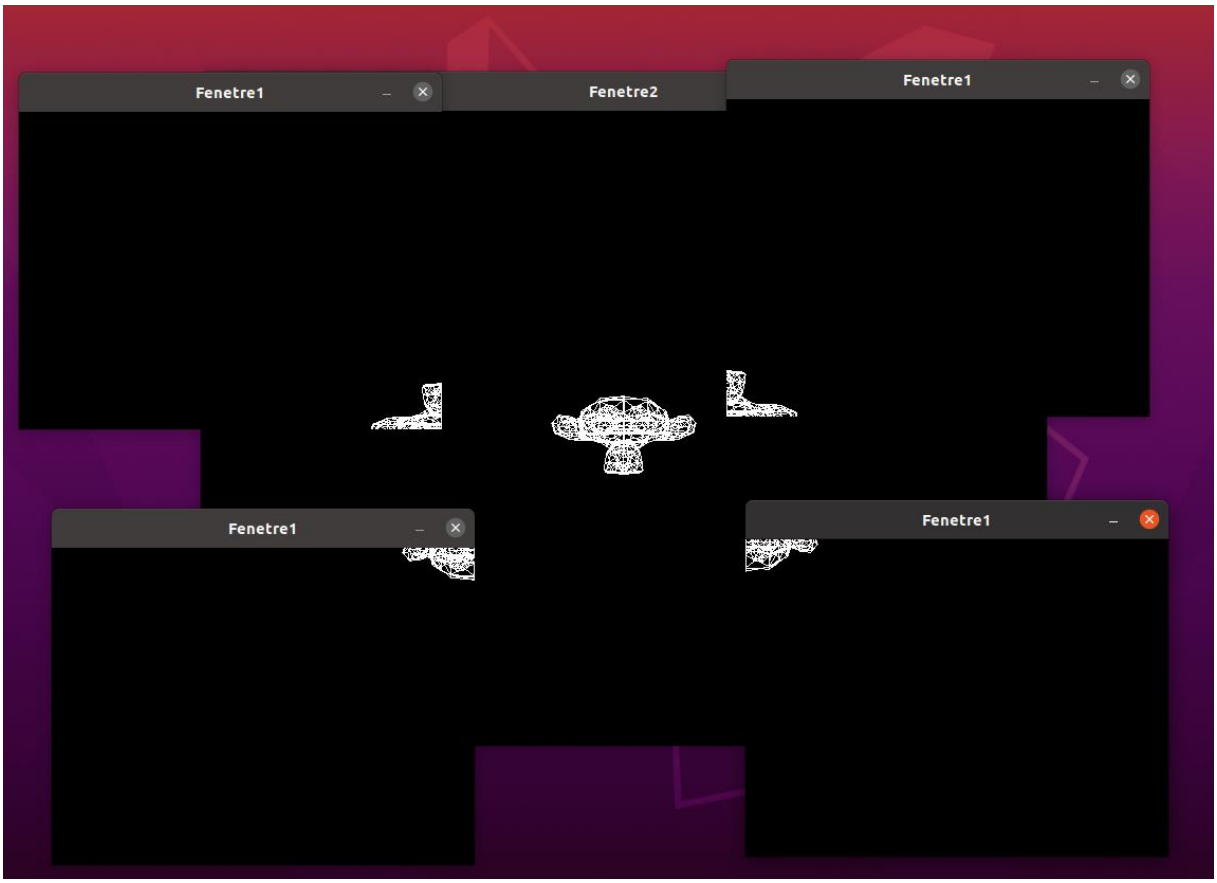


J'ai ensuite réussi à compiler openMesh avec les librairies SDL et openGL j'ai donc put lire ainsi qu'afficher les objets 3D fourni.A ce stade chaque fenetre affiche l'objet 3D dans son entiereter et envoie la bitmap dans la fenetre principal.Pour que le rendu soit partagé

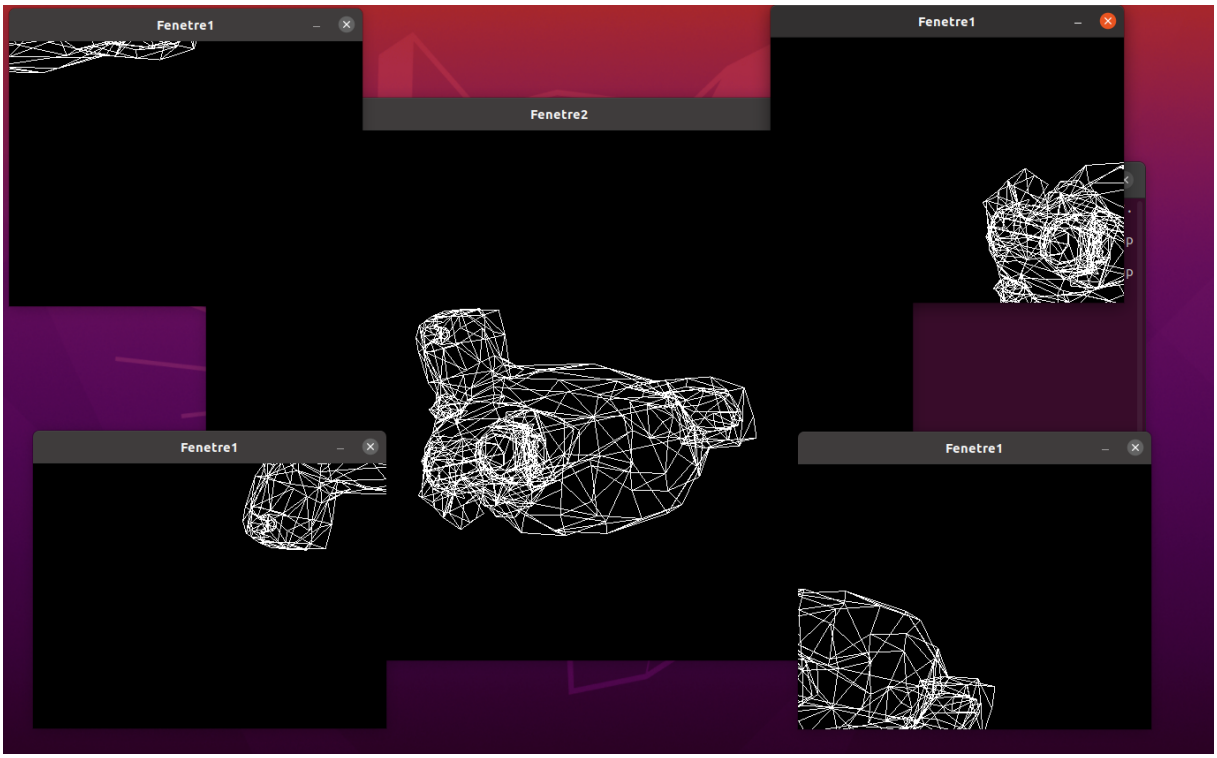
il faudrait que chaque fenetre affiche une parti du rendu 3D et que la fenetre principal assemble les 4 images.



J’ai donc fait sur chaque fenêtre une vue differente de l’objet 3D en deplaçant la caméra orthographique. Le rendu OpenGL doit donc charger qu’une partie de l’objet et la fenêtre principale assemble les 4 images et affiche donc l’objet 3D dans son entièreté.J’ai ensuite fait un rendu en VBO ce qui permet une meilleure optimisation du calcul du rendu de l’objet 3D.



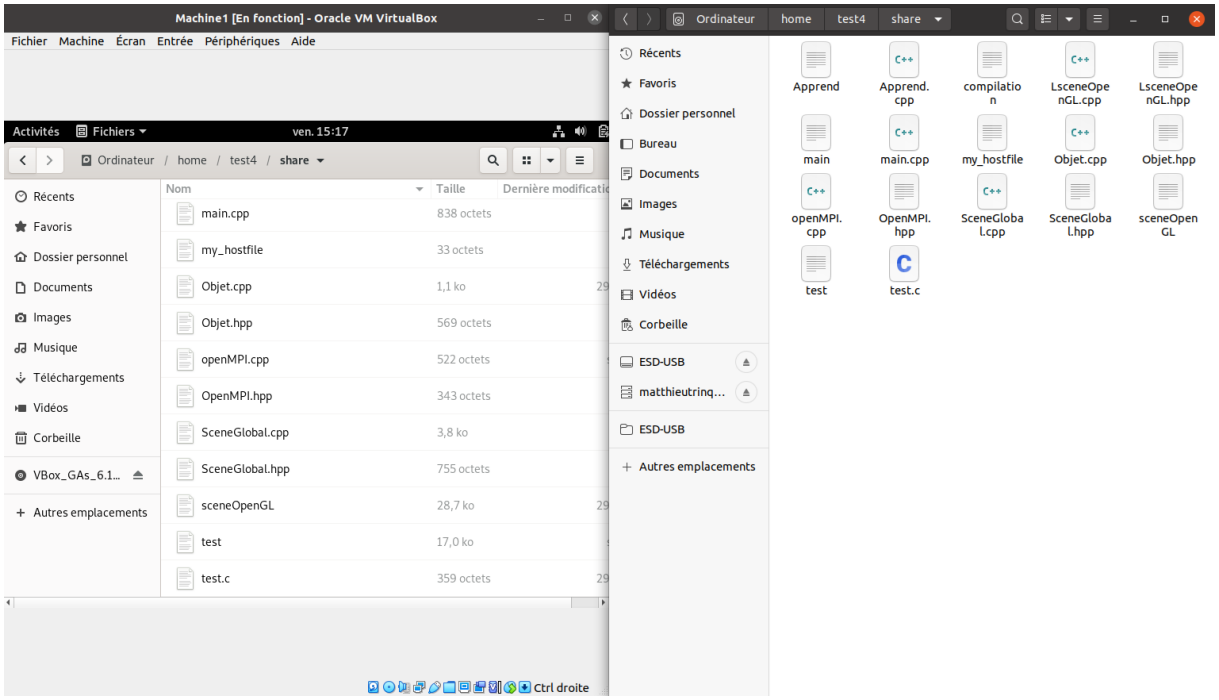
J’ai ensuite rendu mon programme en multi processus grâce à openMPI. J’ai fait en sorte que chaque fenêtre soit dans un processus différent ce qui rend le programme plus optimisé et me permettra ensuite de faire un rendu partagé sur plusieurs machines.



J’ai ensuite utilisé les évènements des fenêtres SDL pour pouvoir detecter la pression des touches du clavier pour pouvoir manipuler l’objet 3D. Les touches Z et S permettent de tourner autour de l’axe X, les touches Q et D pour tourner autour de l’axe Y, les touches M et L pour tourner autour de l’axe Z et enfin les touches O et P pour pouvoir zoomer sur l’objet. J’ai dû pouvoir gérer la synchronisation de chaque rendu pour que l’objet 3D au final reste cohérent.

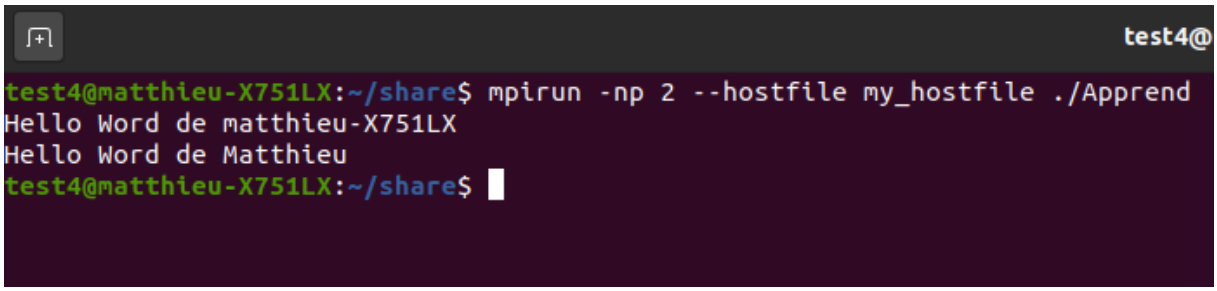
J’ai ensuite commencé à me consacrer à la gestion de programme OpenMPI dans un réseau local. Ne contenant pas plusieurs machines pour tester openMPI dans mon réseau, j’ai donc créé plusieurs machines virtuelles me permettant de simuler d’autres ordinateurs et donc de pouvoir créer des programmes OpenMPI dans un réseau local. Pour pouvoir faire fonctionner openMPI en réseau, j’ai du créer un cluster entre plusieurs machines. Pour cela j’ai dû créer un autre utilisateur sur mes machines et créer un serveur ssh sur ma machine ‘principale’ et connecter le serveur ssh à chacune de mes machines ‘zombies’. Il ne reste plus qu’à créer un dossier «share» qui sera un dossier partagé par toutes les machines en utilisant la commande «mount».

Voici donc mon cluster entre mes 2 machines créées.

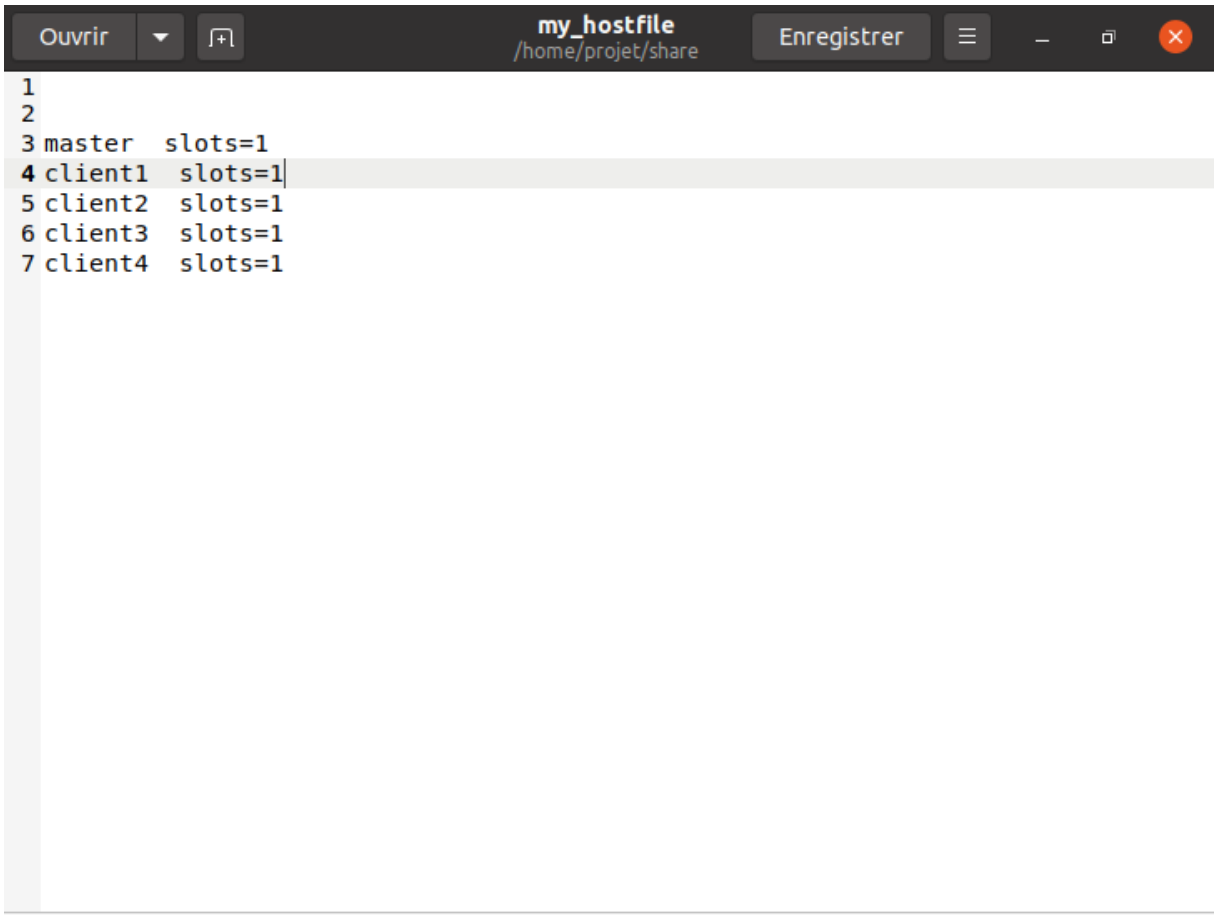


Il me faut aussi configurer un fichier hostfile qui me permettra de distribuer les processus openMPI sur les differentes machines. Par exemple si une machine est moins

performantes on ne peut lui donner qu'un seul processus ou si une machine est plus performante on peut lui donner 2 ou 3 processus en même temps.



Exemple de fichier hostfile :



Il me reste plus qu'à exécuter un programme openMPI dans le cluster. Mon premier programme affichera «Hello word» avec ensuite le nom de la machine où se situe le processus openMPI.

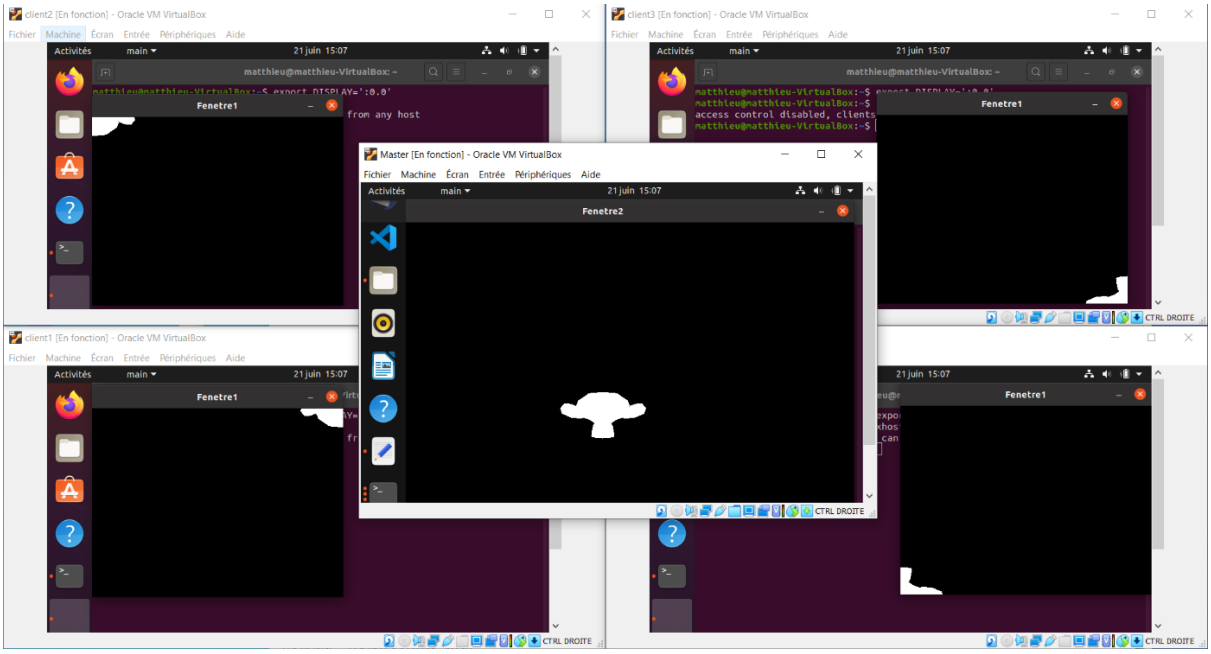
J'ai à ce moment là mis beaucoup de temps à réussir à faire fonctionner SDL dans mon cluster. En effet quand je faisais fonctionner une fenêtre SDL dans mon cluster, une erreur

d'initialisation des fenêtres SDL s'affiche sur le terminal. Donc avec l'aide de mon prof référent et des recherches sur internet, j'ai appris que pour pouvoir faire fonctionner l'affichage des fenêtres SDL dans le cluster, il faut mettre la valeur DISPLAY de chaque machine à la valeur 0 avec la commande export DISPLAY =' :0.0' et executer la commande xhost +.

Après cela, il ne reste plus qu'à rajouter dans la ligne de commande suivant lors de l'éxécution du programme :

-x DISPLAY=:0.0

Après l'exécution faite, le programme se déclenche sur chacune des machines précisées dans le fichier hostfiles.

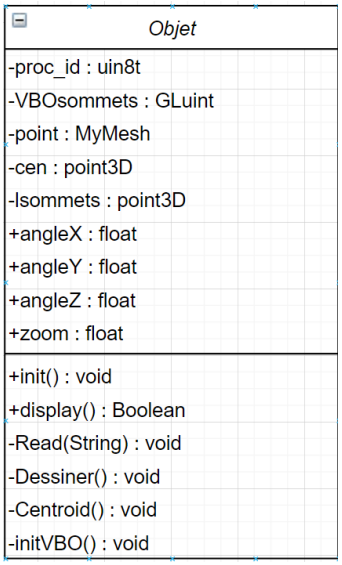


On voit bien que chacune de mes machines virtuelles charge une partie de l'objet 3D et que la machine du centre « la machine principale » affiche l'objet 3D dans son entièreté.

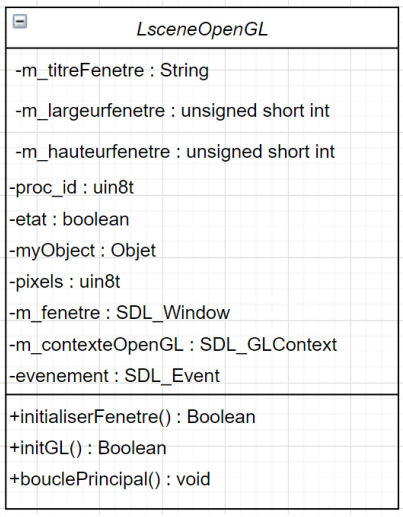
Et enfin j'ai créé une documentation avec l'outil doxygen en décrivant toutes mes classes, ma structure et chaque fonction de mon programme.

2)Explication du programme

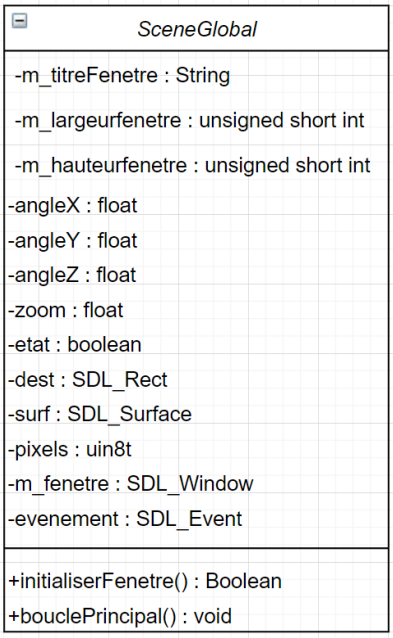
Mon programme contient une classe objet qui a pour but de gérer l'objet 3D. Il contient d'abord une fonction READ qui permet de lire un fichier contenant un objet 3D avec openMesh.



Après l'avoir lu, il fait le rendu grâce à la fonction Dessiner qui va se charger de faire le rendu de l'objet 3D et la fonction display qui va se charger du zoom ou de l'angle de rotation de l'objet ainsi que le positionnement de la camera orthographique. Ces fonctions utilisent les outils d'openGL. La fonction Centroid sert à calculer le centre de l'objet 3D pour pouvoir placer la caméra orthographique. Cette classe contient aussi une fonction init qui permet d'initialiser les outils OpenGL ainsi que d'initialiser le VBO grâce à la fonction initVBO. L'argument proc_id lui permettra de savoir dans quel processus d'openMPI se situe la fenêtre et à partir de conditions de décaler la caméra orthographique et d'afficher donc qu'une partie de l'objet 3D selon le processsus où se trouve la fenêtre.

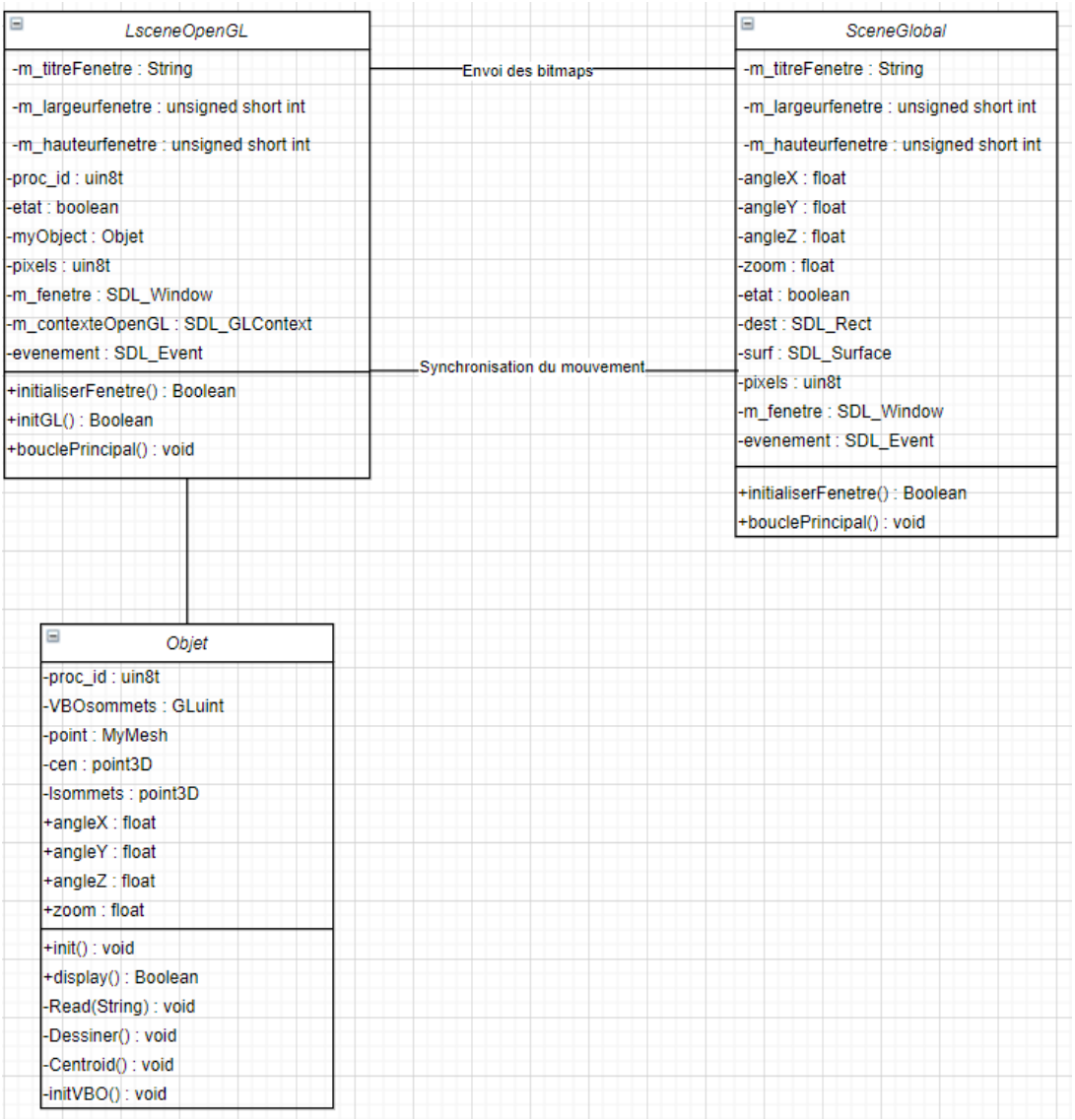


J'ai aussi créé une classe LsceneopenGL qui me permet d'afficher les petites fenêtres qui contiendront chacune une partie de l'objet 3D. La fonction bouclePrincipal se charge de garder afficher la fenêtre SDL. Elle s'occupe aussi d'enregistrer la bitmap de la fenêtre avec la fonction glReadPixel lorsqu'une modification se fait sur l'objet 3D et l'envoie à la fenêtre principale grâce à la fonction MPI_Send d'OpenMPI. La fonction glReadPixel lit la fenêtre à l'envers ce qui fait que l'affichage final de l'objet est à l'envers. J'ai donc fait une rotation de 180° à l'objet 3D pour que l'image final soit à l'endroit. Cette fonction reçoit les instructions de rotation et de zoom de la fenêtre principale grâce à la fonction MPI_Recv et les donne à la classe objet qui se charge de redessiner l'objet avec les modifications apportées. La classe LsceneOpenGL contient aussi une fonction initialiséFenetre qui se charge d'initialiser la fenêtre SDL et une fonction initGL qui initialise l'argument myObject avec tous les outils d'OpenGL.



La classe sceneglobal me permet d'afficher la grande fenêtre qui contiendra l'objet 3D dans sa globalité. Cette classe contient une fonction boucleprincipal qui se charge d'afficher en continue la fenêtre SDL et de recevoir les images envoyées par la classe LsceneOpenGL et de les assembler ce qui donnera l'objet 3D dans son entièreté. Cette fonction s'occupe aussi de gérer les évènements de pression de touche de clavier pour pouvoir les envoyer à la classe LsceneOpenGL grâce à la fonction d'OpenMPI MPI_send. Cette classe contient aussi une fonction initialiserFenetre qui permet d'initilialiser les outils SDL.

Une dernière classe qui contiendra toutes les fonctions d'OpenMPI. J'ai aussi créé une structure point3D qui me permet de stocker plus facilement les points en 3 dimensions avec les paramètres x,y,z.



Pour OpenMPI, j'ai décidé de créer 5 porcessus : 1 qui me permet d'afficher la grande fenêtre avec l'objet 3D dans sa globalité (donc contenir la classe sceneglobal) elle sera donc la machine maître et 4 autres qui vont afficher chaque partie de l'objet 3D (donc contenir la classe LsceneopenGL) qui seront donc les machines zombies. La fonction main va contenir la classe OpenMPI pour pouvoir faire du multi processus.

Pour compiler j'utilise la ligne de compilation suivante :

sudo mpiCC *.cpp -o /home/projet/share/main -lGLEW -lGL -lGLU -lSDL2 -lOpenMeshCore

sudo: permet d'executer la commande en mode ROOT pour pouvoir le placer dans le cluster

mpiCC: permet de compiler un programme C++ avec openMPI

*.cpp : permet de selectionner tous les fichiers cpp du dossier pour pouvoir les compiler

/home/projet/share/main : chemin où se situera le programme principal (dans le cluster)

-lGLEW -lGL -lGLU : permet de compiler avec openGL

-lSDL2 : permet de compiler avec SDL2

-lOpenMeshCore : permet de compiler avec OpenMesh

Puis après avoir compiler, il faut se situer dans le dossier où se situe le programme compiler et se connecter à l'utilisateur commun entre chaque machine et exécuter la commande :

mpirun -x DISPLAY=:0.0 -np 5 -hostfiles my_hostfile ./main

mpirun : exécuter un programme openMPI

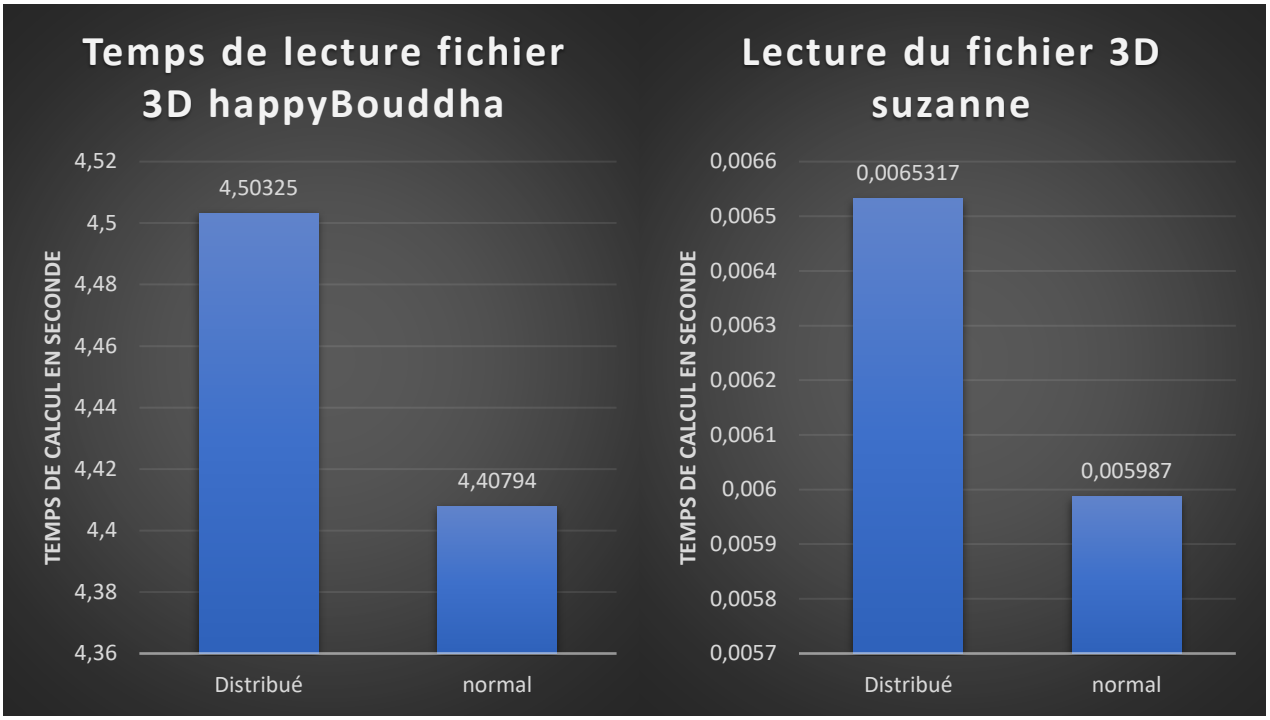
-x DISPLAY=:0.0 : permet de mettre la valeur DISPLAY à 0.

-np 5 : permet de mettre le nombre de processus que openMPI va créer (dans notre cas il en faudra 5 : 1 pour l'affichage complet de l'objet 3D et 4 autres pour afficher 4 parties différentes de l'objet 3D)

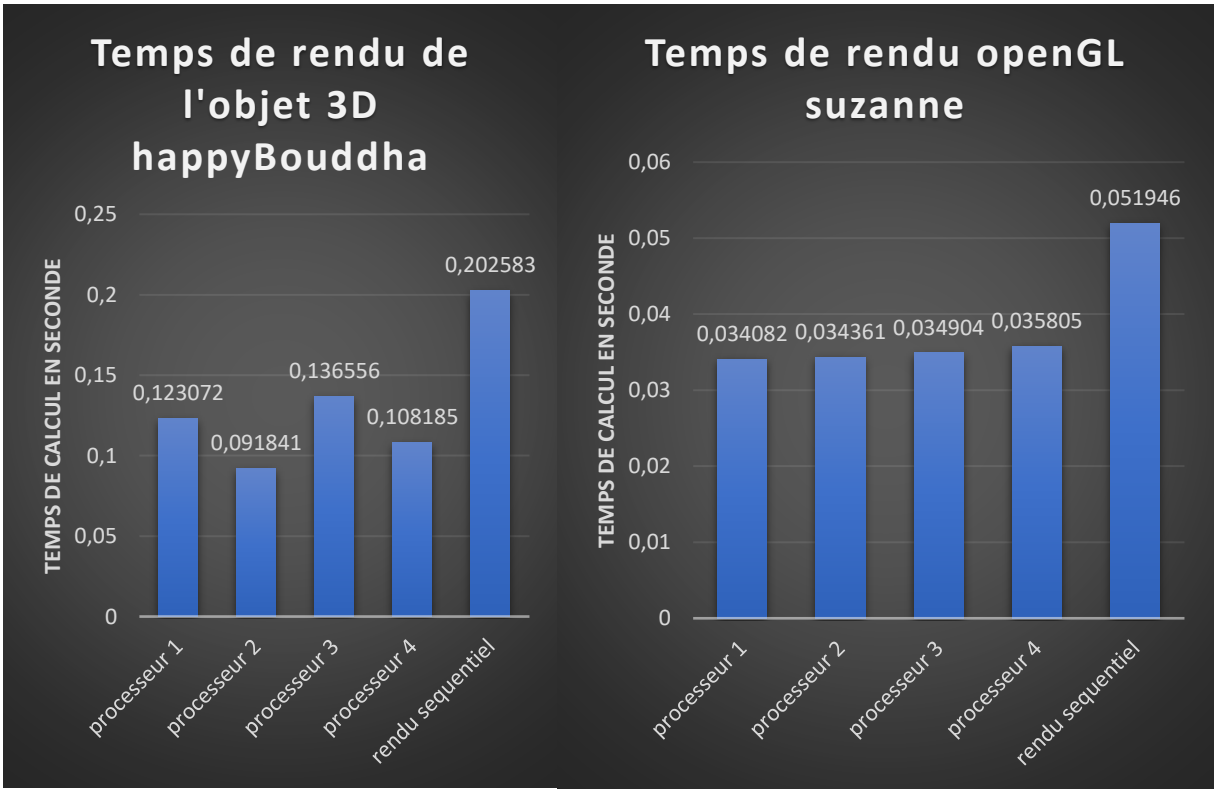
-hostfiles my_hostfile : permet de donner le fichier hostfile à la compilation

./main : permet de donner le nom du programme à exécuter.

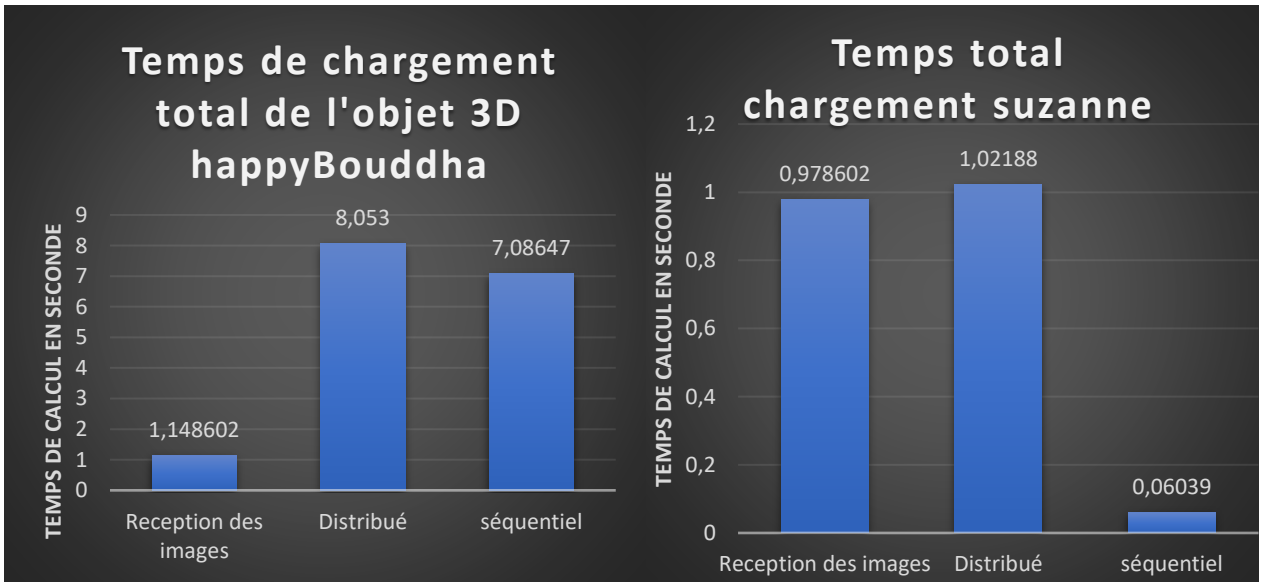
3)Mesure des temps et explication



On peut remarquer que la lecture du fichier obj est généralement plus long quand la répartition de l'objet est partagée. Ce décalage peut être dû au fait que le fichier se trouvant dans le cluster retarde la lecture mais cette différence étant très légère (2% pour happyBouddha et 8% pour suzanne) il est possible que ça soit dû à une erreur d'approximation ou au hasard.



Le rendu OpenGL est très avantageux pour le rendu distribué. Le gain de temps va de 0.07s jusqu'à 0.11s pour happyBouddha. Pour l'objet suzanne le gain de temps est d'environ 0.017s sur chaque processus. On peut nettement imaginer que plus l'objet 3D est gros à charger plus le gain de temps entre le rendu distribué et le rendu séquentielle sera grand. Certaines variations de temps sont possible entre chaque processus (cette variation est bien visible sur les temps de happyBouddha) elle est dû au fait que la répartition du nombre de polygones entre chaque processus n'est pas égale ce qui fait que certains rendus se créent plus rapidement que d'autres. Si dans une entreprise des machines sont plus performantes que d'autres il serait judicieux que les rendus les plus lourds se fassent dans les machines les plus performantes et inversement les rendus les plus légers se fassent dans les machines les moins performantes. Cela permettra un gain de temps dans le rendu final.

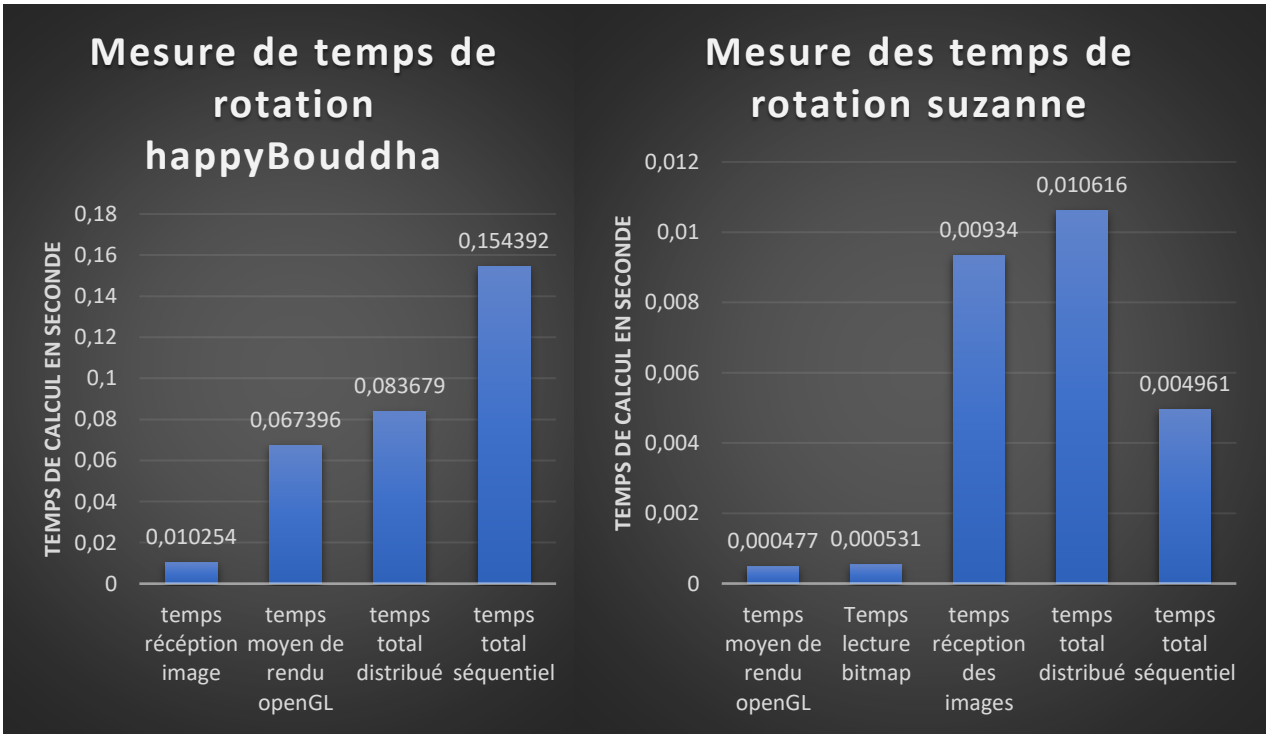


La répartition du rendu 3D n'est pas avantageux pour le chargement des objets 3D. Le problème est que le temps d'envoi et de réception des images prend beaucoup de temps. Pour le chargement de l'objet suzanne l'envoi et la réception des bitmaps prennent quasiment la totalité du temps total du chargement de l'objet. Cependant pour des plus gros objets le gain de temps gagné dans le rendu de l'objet pourrait combler le temps perdu dans l'envoi et la réception des bitmaps. Des optimisations du programme sont possibles pour rendre le chargement des objets 3D plus rapide.

Mon programme envoie directement toute la bitmap RGBA de la fenêtre SDL. La fenêtre SDL faisant 400 * 300 le programme doit donc envoyer dans le réseau une image de taille 400*300*4 octets ce qui donne 480Ko pour chaque fenêtre SDL ce qui donne au total 1,92M à envoyer dans le réseau. La taille de l'image peut être largement diminuée si on le convertit directement en PNG dans la mémoire des machines ce qui ferait un gain de temps non négligeable.

Dans le programme que j'ai fait chaque processus de chaque machine fait une lecture du fichier 3D il serait possible qu'un seul processus lise le fichier contenant l'objet 3D et qu'il le redistribue ensuite à chaque processus ce qui éviterait de lire 4 fois le fichier.

Chaque machine garde dans sa mémoire l'objet 3D. Il serait mieux de partager une mémoire entre chaque machine pour éviter de garder 4 objets en mémoire. Cela permettrait de libérer de la mémoire et ferait un gain de temps dans l'initialisation du VBO.



Pour la manipulation de l'objet (tourner autour de l'objet 3D ou de zoomer dessus) la version distribuée sur l'objet HappyBoudha est largement plus avantageuse que la version séquentielle. Le temps de réception des images étant plus court, cela permet de diminuer drastiquement le temps total et permet donc un gain de temps d'environ 0.7s en moyenne entre la version distribuée et la version séquentielle. Mais malheureusement pour les objets plus petits comme suzanne le gain de temps fait dans le rendu OpenGL ne permet de compenser le temps

perdu dans l'envoi des bitmaps ce qui fait que la version distribuée n'est toujours pas avantageuse pour les petits objet 3D. Comme dit plus haut nous pouvons optimiser l'envoi des bitmaps en les convertissant en PNG dans la mémoire des machines.

III)Conclusion

Durant tout le déroulement du projet les principales difficultés ont été de pouvoir compiler OpenMesh avec d'autres librairies et aussi de réussir à initialiser SDL dans un cluster. Mais j'ai réussi à surmonter ces difficultés et toutes les fonctionnalités demandées dans mon intitulé ont été faites. Des optimisations sont possibles comme la conversion des bitmaps en PNG ou le rendu fait dans une mémoire partagée par chaque machine. Mais ce projet ne remplacera jamais l'expérience professionnelle que m'aurait apporté un stage en entreprise. Les relations humaines sont différentes ou encore le fonctionnement sur l'avancement du projet. Mais ce projet m'a permis l'apprentissage de nouveaux outils informatiques comme la gestion des fenêtres par SDL que je n'avais jamais utilisée auparavant. J'ai aussi appris le fonctionnement de OpenMPI pour le multiprocessus qui est donc une nouvelle expérience apportée par le projet. Cela m'a appris aussi l'outil OpenMesh. Ce projet m'a aussi permis d'améliorer mes connaissances en serveur SSH ou en réseau grâce à la réalisation de mon cluster. Cela m'a aussi permis d'améliorer mes compétences en rendu 3D en OpenGL, je me sens maintenant plus à l'aise avec l'outil OpenGL qu'auparavant.

VI)Lexique :

SDL : SDL est une bibliothèque de développement multiplateforme qui permet un accès de bas niveau au matériel audio, souris, clavier, et d'afficher des fenêtres.

OpenGL : OpenGL est une bibliothèque multiplateforme qui permet des rendus 2D et 3D.

OpenMesh : OpenMesh est une bibliothèque permettant de lire un fichier 3D et d'y restituer les points.

OpenMPI : OpenMPI est une bibliothèque MPI qui permet de faire du multiprocessus sur une machine ou plusieurs machines dans un réseau local.

Cluster : le cluster est un répertoire commun partagé par plusieurs machines dans un réseau local.

SSH : un réseau SSH (Secure Shell) est un protocole de réseau cryptographique qui permet de faire dialoguer des machines client-serveur de façon sécurisé.

Multiprocessus : le multiprocessus est de le faire d'exécuter un programme de façon parallèle et ainsi exécuter plusieurs calculs en même temps.

Programme Séquentiel : un programme séquentiel est l'inverse du multiprocessus, c'est un programme exécuté dans un ordre précis.

Doxygen : outils permettant de faire des documentations d'un code sous une page web.

TCP/IP : système qui règle la connexion des systèmes informatiques sur internet

C++ : langage informatique couramment utilisé.

Bitmap : format d'image fait sous une forme de matrice.

Caméra orthographique : une représentation orthographique est une représentation ne mettant pas de perspective dans l'image (différent de la vision humaine).

VBO : le VBO (vertex buffer Object) est le fait de mettre un objet 3D directement dans la mémoire de la carte graphique sans passer par la mémoire de l'ordinateur.

PNG : format d'image compressé sans perte.

V)Bibliographie :

https://www.supinfo.com/articles/single/922-parallelisation-linux-mettre-place-cluster-mpi

https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c/17117-installation-de-la-sdl

https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c/17546-creation-dune-fenetre-et-de-surfaces

https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c/17796-afficher-des-images

https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMFHS-099/Une-introduction-a-la-programmation-parallele-avec-Open-MPI-et-OpenMP

https://stackoverflow.com/questions/33187159/error-xdg-runtime-dir-not-set-in-the-environment-gtk-warning-cannot-open-d

https://zestedesavoir.com/forums/sujet/10438/sdl2-erreur-dinitialisation-no-available-device/

https://www.graphics.rwth-aachen.de/software/openmesh/

https://openclassrooms.com/fr/courses/966823-developpez-vos-applications-3d-avec-opengl-3-3/963847-la-camera

https://openclassrooms.com/fr/courses/966823-developpez-vos-applications-3d-avec-opengl-3-3/964437-les-vertex-buffer-objects