JORDA Marco
DEMAREST Alexis
SAUGE Theo
CORNU Alexandre

MÉMOIRE DÉCOUVERTE DE LA RECHERCHE

M. M'HAND HIFI

SUJET 1 : ORDONNANCEMENT DE TÂCHES
SUJET 2 : PLACEMENT DE CERCLES DANS UNE ZONE (Ouvert & Fermé)
SUJET 3 : PLACEMENT DE SPHÈRES DANS UN CONTAINER (Ouvert & Fermé)

SUJET 1 : Réflexion au sujet de l'ordonnancement de tâches.

*

• En ce qui concerne l'ordonnancement des tâches, toutes les recherches effectuées se sont limitées à des problèmes avec des modèles à machine unique. C'est-à-dire qu'une seule tâche peut être effectuée à la fois.
Nous avons alors commencé par le plus simple l'ordonnancement sans date de livraison.
Dans cette idée, nous avons des tâches ayant un nom et une durée. Le projet est considéré terminé lorsque toutes les tâches ont été effectuées.
Premièrement, on voudrait essayer de faire en sorte que le projet se termine le plus tôt possible. Seulement le seul paramètre sur lequel nous pouvons influer est l'ordonnancement. Mais en raison de la commutativité de l'addition, il est impossible de faire varier la date de fin du projet autrement qu'en modifiant la durée des tâches. Donc l'ordonnancement des tâches n'influence pas la date de fin d'un projet.

*

Maintenant que l'on sait que la date de fin de projet est immobile. Nous pouvons nous demander si l'on peut modifier la durée moyenne de fin des tâches d'un projet.
Il est possible de modifier la durée moyenne, car lorsqu'une tâche est en cours, les autres doivent "attendre". Ainsi lorsque une tâche est en train de s'effectuer, la fin de toutes les autres tâches est reportée.
Dans cette optique, il convient de toujours commencer par les tâches les plus courtes afin de réduire le temps moyen d'exécution d'une tâche.

Exemple : On a 2 tâches A et B respectivement de longueur 30s et 120s.
Si nous faisons B puis A alors on a 120s d'exec pour B et 150s pour A (puisque A doit attendre que B finisse) donnant (120+150)/2=135s en moyenne
Si nous faisons A puis B alors on a A=30s et B=150s soit (150+30)/2=90s en moyenne
Référence : Partie 2 du cours de M'Hand HIFI en découverte de la recherche
Cet algorithme s'appelle le Shortest Processing Time First ou Shortest Job First, nous l'avons ainsi développé en Python.

Ajoutons maintenant un nouveau paramètre, le délai de livraison. Le délai de livraison est la date à laquelle une tâche doit être terminée. Si ce n'est pas le cas, la tâche accumule du retard jusqu'à sa complétion.
Nous nous demanderons alors comment minimiser le retard maximal des tâches du projet. Pour cela, nous avons utilisé l'algorithme de l'Earliest Due Date (EDD).
Cet algorithme dit que pour minimiser le plus grand retard des tâches d'un projet,

il faut effectuer les tâches ayant le délai de livraison le plus tôt en priorité. Cela peut paraître simpliste mais finalement très logique. En effet, si une tâche A doit à un délai de livraison plus tôt qu'une tâche B. Effectuer la tâche B fait attendre A donc aucune autre tâche ne pourra avoir un retard plus grand que A.
Nous avons aussi développé cet algorithme en Python.

*

Pour ce qui est de minimiser le retard moyen sur l'ensemble des tâches et minimiser la somme des retards. Nous avons considéré ces deux objectifs comme étant très proches et nous nous sommes concentrés sur la résolution du premier.
Une solution "simple" serait de simplement calculer toutes les possibilités d'ordonnancement pour chaque projet et de choisir celle dont le retard moyen est le plus bas. Bien que cette solution soit possible, elle reste tout de même particulièrement lourde à calculer. Pour un nombre de tâche N dans un projet, il faudrait calculer N! ordonnancements et choisir le meilleur.
Nous n'avons donc pas opté pour cette solution et ne l'avons pas développée.

*

Nous avons donc essayé d'adapter la solution que nous avons vu pour l'ordonnancement sans délai à savoir le Shortest Job First. L'une des premières réflexions que nous avons eu est que nous n'avions aucun intérêt à finir une tâche en avance puisque un délai négatif revient à un délai égale à 0 (autrement on pourrait considérer ce problème comme étant identique à celui sans délai). Dans le cas idéal, chaque tâche se termine en même temps que son délai de livraison.
Nous avons alors essayé de mettre au point un algorithme suivant ce principe. Cet algorithme calcule pour chaque tâche le temps restant avant que la tâche soit en retard (autrement dit l'opposé du retard). Puis on soustrait la durée de la tâche par ce temps restant et si la valeur obtenue est plus proche de 0 que la valeur enregistrée précédemment, cette dernière est remplacée. On effectue alors la tâche enregistrée puis on réitère l'opération jusqu'à avoir terminé le projet.
Cet algorithme suit assez bien le principe énoncé juste avant. Mais sa grosse faiblesse est qu'il crée de gros artefacts. Certaines tâches se retrouvent alors avec des retards énormes alors que d'autres comptent de très faibles retards. Nous l'avons développé en Python.
Les résultats obtenus étaient satisfaisants mais souvent non optimaux. Nous pensons que les principes sur lesquels l'algorithme est écrit sont corrects mais malheureusement nous n'avons pas réussi à les adapter comme nous le souhaitions.

SUJET 2 : Placement de cercles dans une zone en 2D (Ouverte & fermée).
SUJET 3 : Placement de sphères dans une zone en 3D (Ouverte & fermée).

*

• Au niveau des placements de cercle, le problème restait tout de même assez complexe dans un point de vue global, le principe étant de base de les coller au maximum pour pouvoir prendre le moins de place et par conséquent en placer un maximum, il fallait considérer toute la surface du cercle avec des calculs pour vérifier les collisions.

Néanmoins, après deux séances de recherche, nous avons trouvé quelques indications concernant les algorithmes que nous pouvions utiliser et exploiter pour pouvoir les adapter à notre problème et avoir quelque chose de cohérent. Après donc avoir trouvé un algorithme qui nous permettait de comprendre comment la vérification des cercles allait plus ou moins se dérouler, nous avons convenu après une sorte de “réunion” que le placement allait se faire de manière aléatoire, de cette manière il nous restait en principe plus qu’à vérifier si le placement était bon et donc le placer.

*

De ce fait, nous avons pris au départ l’algorithme et l’avons adapté de façon à ce qu’il fonctionne de cette manière :

1. En premier lieu, nous créons la zone.
2. Ensuite, nous rentrons dans une boucle qui va tourner jusqu’à ce qu’il n’y ait plus de place.
3. Une fois dans la boucle, nous plaçons un cercle au hasard mais vérifions au préalable qu’il n’effectue aucune collision avec un autre cercle. Pour ce faire, nous vérifions que la distance entre les deux centres est inférieure à la somme des rayons des deux cercles (celui que nous voulons placer et celui que nous parcourons). Pour que le placement soit valide, nous vérifions qu’il n’y aucune collision avec **aucun** des cercles déjà placés dans la totalité du tableau et ne rater aucun cercle.
   (Il est important de noter que la vérification s’occupait en théorie aussi d’annuler le placement si le cercle touchait les bords du canevas ou de la zone prédéfinie)
4. Une fois le cercle placé, nous les faisons “grandir”, en effet, nous pensions que si nous faisions grandir chaque cercle déjà situé dans le tableau, nous allions optimiser au mieux le “remplissage” par les cercles mais, par logique, en en plaçant moins.
5. Nous vérifions donc si "l'agrandissement" du rayon de chaque cercle était possible et, le cas échéant, on passerait au prochain cercle..etc.

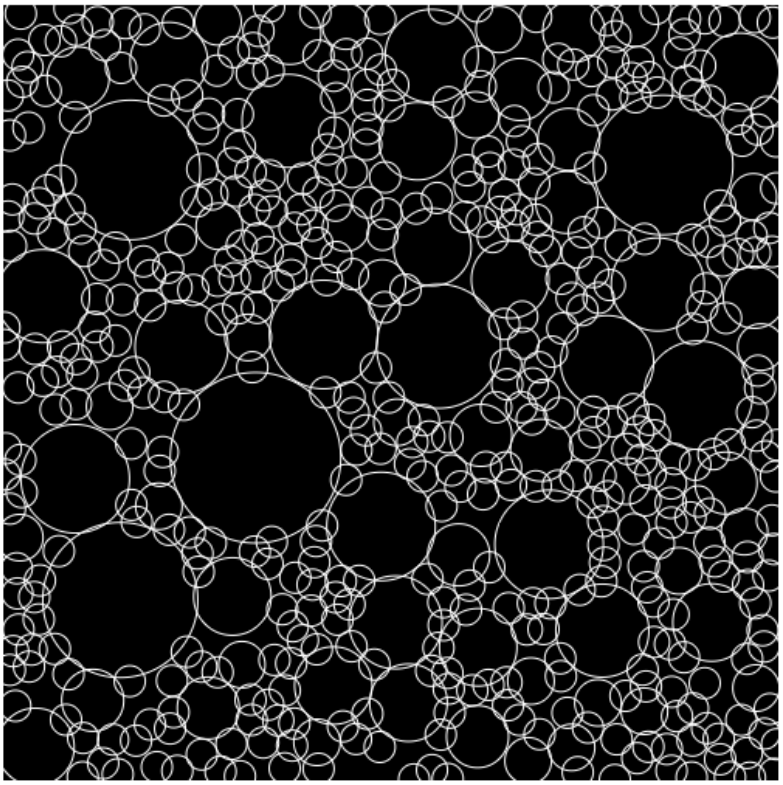
6. Une fois un cercle supplémentaire placé et les autres ayant “grandi” (ou non si collision il y avait), nous recommençons la boucle.

Cette boucle avait bien entendu une limite, et malgré le fait qu’au départ nous voulions faire une idée de pourcentage qui sur le concret s’est avéré plus compliqué que prévu, nous avons donc opté pour une simple vérification avec un nombre de tentatives maximum, c'est-à-dire que si le programme faisait 5 000 tentatives (5 000 tentatives de placement au hasard) d’affilée sans succès, nous considérions que le rectangle était “rempli”.

Le problème que nous avons trouvé ici, ou plutôt la “non-optimisation” est qu’on aurait tout de même pu continuer à faire grandir les autres cercles et faire cette même vérification dessus, afin de vérifier que les cercles déjà placés ne pouvaient pas non plus grandir.

Néanmoins, nous n’avons pas abouti à quelque chose de vraiment propre :

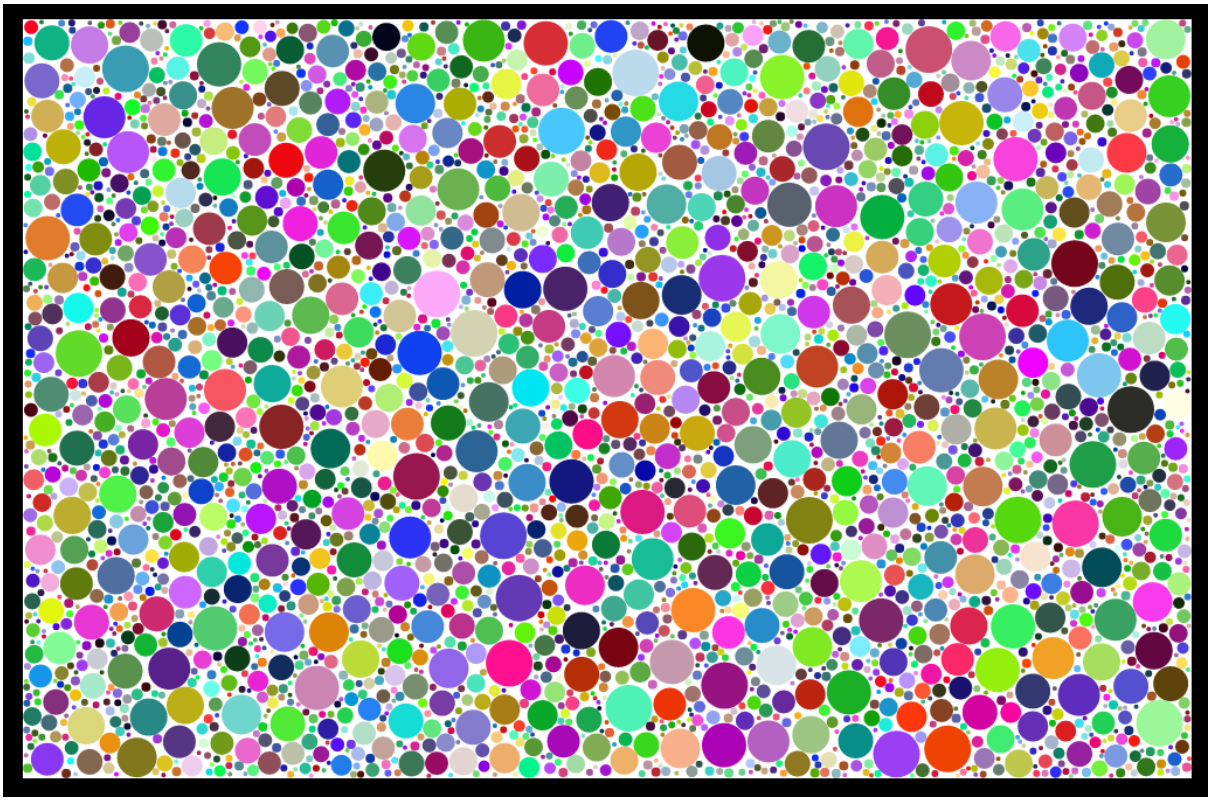
Notre algorithme était fonctionnel mais mettait en avant quelques erreurs de vérification néanmoins.



*

Ayant néanmoins une base sur laquelle nous pouvions travailler, nous sommes partis sur une variante que nous considérions à l'origine comme inutile, celle de prédéfinir la taille des cercles. Ainsi, nous avons pris la structure brute de l’algorithme pour la recalquer avec une taille de cercle prédéfinie et un placement jusqu’à atteindre la même limite de 5 000 tentatives.
Il se basait toujours sur un placement au hasard dans une zone avec une taille (en l’occurrence un rayon) au hasard entre un minimum et un maximum.

Visiblement, cet algorithme fonctionnait mieux. Afin d'avoir une meilleure visibilité des différents cercles (et un peu de mosaïque pour le plaisir visuel), nous avons également joué avec un remplissage **des** cercles avec une couleur au hasard.
Avec ce même algorithme mais en effectuant quelques adaptations pour répondre au principe du placement direct et non de "l'agrandissement" différent, nous obtenions quelque chose de ce style là :



On remarque qu'il y a tout de mêmes quelques creux où des petits cercles peuvent se placer, mais la coïncidence de placement au hasard est tellement infime (et grandit d'ailleurs de manière exponentielle plus la zone est grande) que les 5 000 tentatives sont bien souvent atteintes avant de trouver le placement libre.

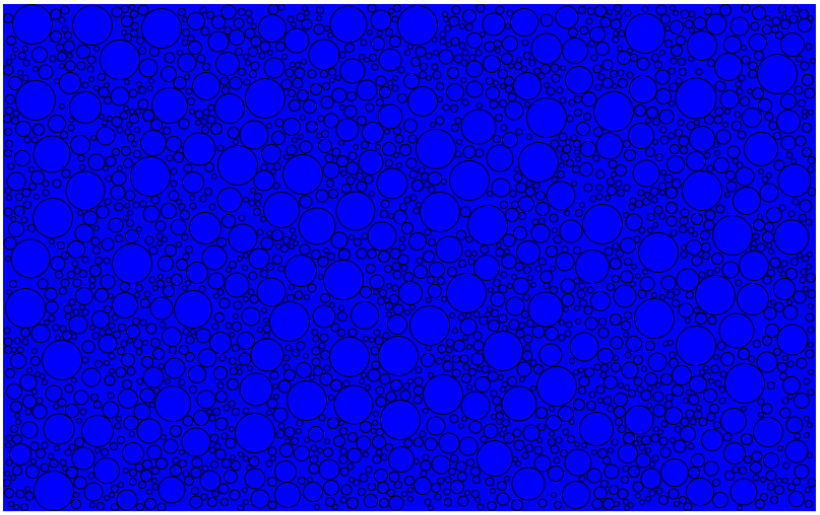
*

Après avoir obtenu cette version qui nous plaisait tout de même, nous avons voulu en faire quelques statistiques pour analyser les placements de cercle rapidement et voir les performances. On avait ainsi un programme qui nous donnait combien de cercles étaient dans notre zone, avait leurs rayons respectifs ainsi que le temps que prenait le programme à remplir la zone.(en l'occurrence, les, puisque le programme "nettoyait" le canevas et recommençait jusqu'à atteindre le nombre d'occurrences voulus)

Nous avions ainsi, pour 2 tentatives, quelque chose qui était tout de même indicatif :

```
In this attempt : 3408 circles were drawn.                    fillcircles.js:56
▶ Map(19) {2 => 866, 3 => 828, 4 => 443, 5 => 299, 6 => 161, …}   fillcircles.js:113
In this attempt : 3521 circles were drawn.                    fillcircles.js:56
▶ Map(19) {2 => 895, 3 => 903, 4 => 504, 5 => 258, 6 => 179, …}   fillcircles.js:113
3464.5                                                        fillcircles.js:129
▶ (2) [3408, 3521]                                            fillcircles.js:130
▶ (2) [63663, 59492]                                          fillcircles.js:131
>
```

Toujours pour essayer d'exploiter un maximum, nous avons fait une autre variante pour quelque chose de plus clair et de final qui nous donnait ceci :



Circles with diameter 5px: 441

Circles with diameter 6px: 332

Circles with diameter 7px: 260

Circles with diameter 8px: 176

Circles with diameter 9px: 118

Circles with diameter 10px: 104

Circles with diameter 11px: 84

Circles with diameter 12px: 45

Circles with diameter 13px: 51

Circles with diameter 14px: 42

Circles with diameter 15px: 42

Circles with diameter 16px: 20

Circles with diameter 17px: 18

Circles with diameter 18px: 28

Circles with diameter 19px: 11

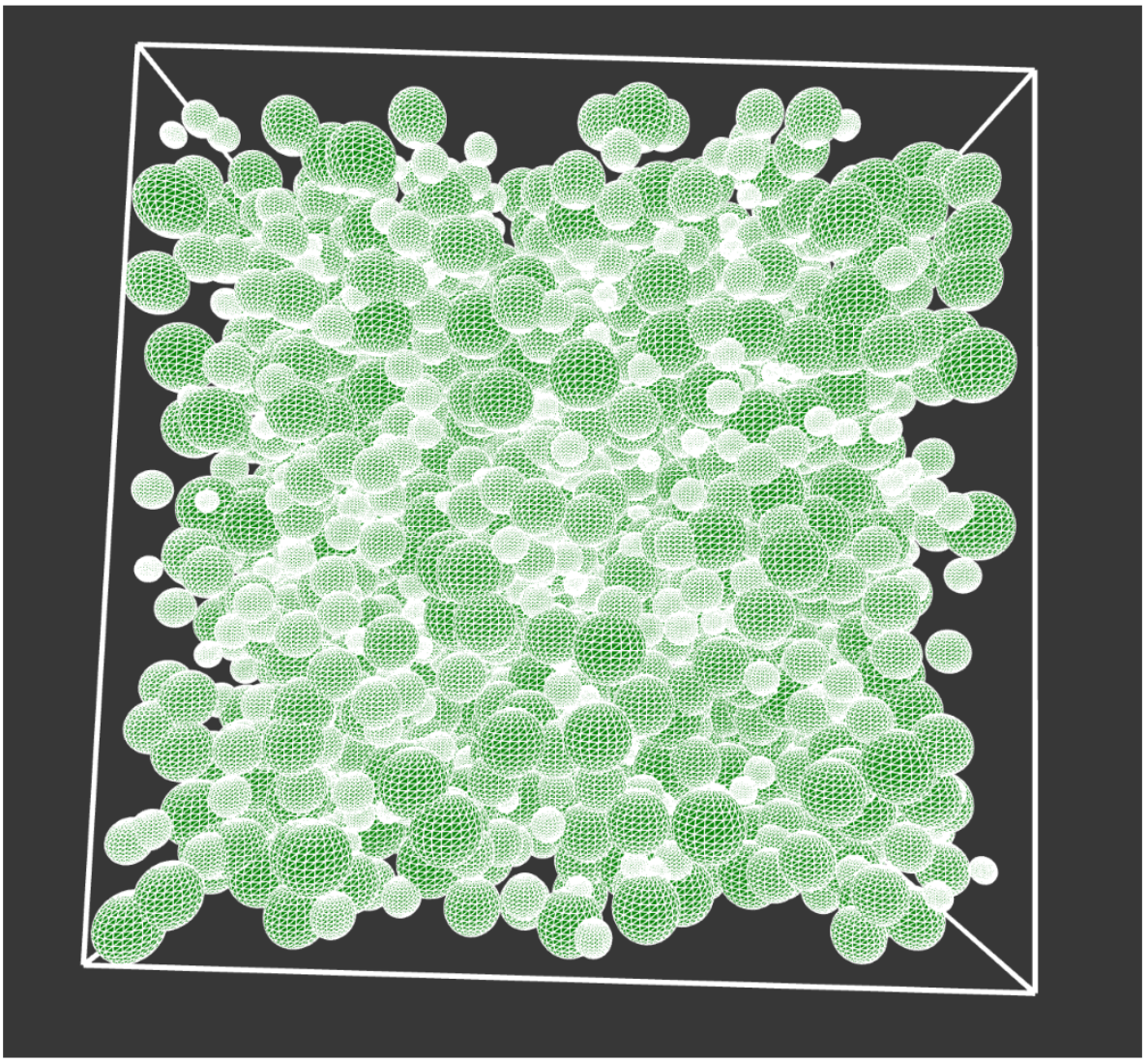
Circles with diameter 20px: 16

Circles with diameter 21px: 11

Circles with diameter 22px: 12

*

Finalement, après avoir obtenu en 2D quelque chose qui nous semblait correct, nous voulions passer sur la 3D car, ayant déjà l'algorithme fonctionnel pour la 2D, en théorie nous devions simplement recalquer les calculs de distance en 3D ainsi que les placements pour avoir quelque chose de fonctionnel.
Néanmoins, cela a été plus difficile que prévu et nous n'avons pas pu obtenir quelque chose de bien correct, l'adaptation s'avérant plus difficile que prévue.
En effet, notre résultat final (avant donc la soutenance) était donc celui-ci :

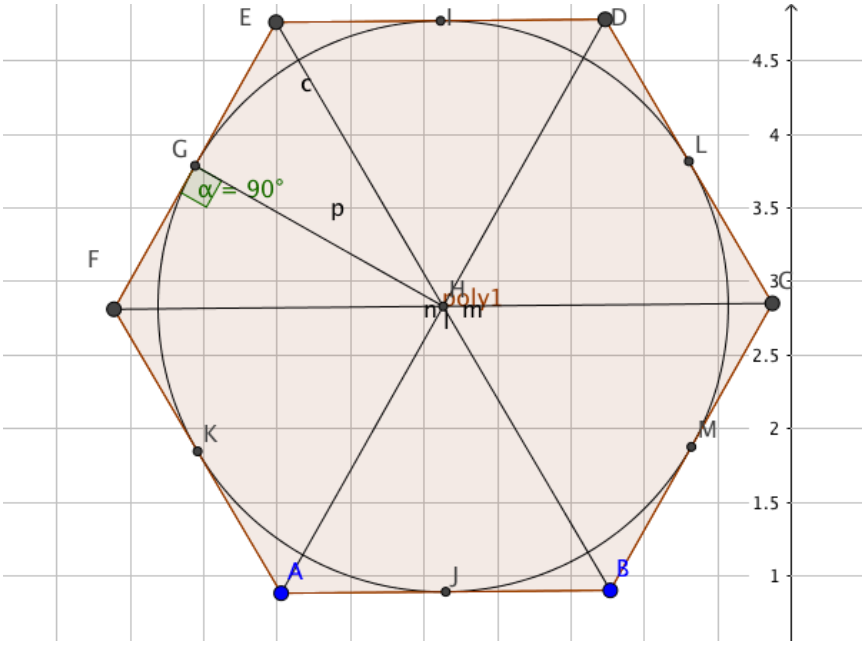


On remarque que l'algorithme fonctionne (comme vous pourrez le voir sur le code) bien sur le placement donné de nombre de sphères, mais la vérification est mauvaise pour certaines sphères.

*

Malheureusement à cause du temps dans les séances qui passait très vite, nous souhaitions passer directement sur le placement "volontaire" de cercles (au contraire du placement au hasard).

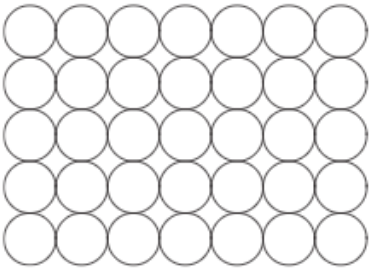
Et malgré le fait qu'à la fin de la dernière séance nous n'ayions pas pu obtenir quelque chose de vraiment exploitable, nous possédions quelques idées qui se basaient, par exemple, sur le fait de considérer un cercle comme un hexagone de cette manière là :



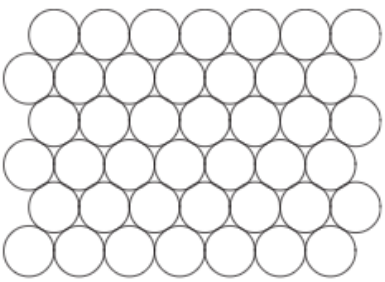
Avec cette considération, les collisions n'allaient pas se faire vu que si les hexagones se collent, les cercles à l'intérieur se collent mais ne se superposent pas.

*

En deuxième point, nous devions penser au placement de cercles, allait-il s'effectuer de cette manière là ?

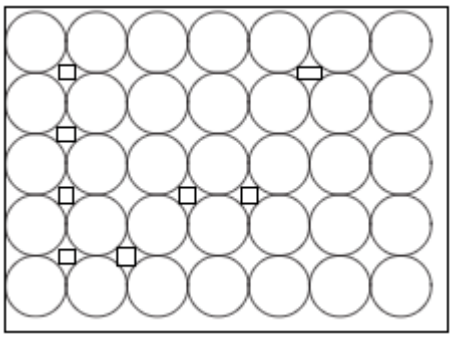


Ou bien de cette manière là ?



Dans les comparaisons, nous avions conclu que même si les cercles dans le deuxième cas semblaient mieux placés, le premier cas nous offrirait de nouvelles possibilités.
En effet, après quelques analyses on s'est rendu compte que le premier cas nous permettait de faire un algorithme récursif se basant sur les "zones vides".
On a pensé à développer un algorithme qui allait donc vérifier toutes les zones blanches et les agrandir jusqu'à faire une collision, et ensuite remplir chaque zone blanche.
De manière évidente, le deuxième cas ne nous le permettait pas.

On avait donc prévu quelque chose qui était semblable à ce schéma :



L'algorithme allait donc sur le principe tâcher
de trouver toutes les zones une fois aucun cercle "posable" dans le grand puis rentrer dans la zone et recommencer la boucle **jusqu'à ce que l'algo ne trouve aucune zone pouvant contenir un cercle de taille minimale**.

Malheureusement nous n'avons pas pu aboutir à développer cette méthode, qui aurait pu être un bon sujet de réflexion et d'implémentation pour notre problématique de placement de cercles **et** de sphères.

Le but étant "d'agglutiner" le plus de cercles dans un petit endroit, nous pensions donc que celle-ci était une bonne manière de remplir une zone en utilisant un espace minimal puisqu'avec le deuxième cas, nous pensions que les petits cercles allaient prendre plus de place en se mettant sur les bords, une fois que les grands étaient déjà placés. A contrario, dans le premier cas, nous pensions que les limites de la zone occupée étaient déjà prédéfinies après le placement des plus grands cercles et que les plus petits allaient se placer entre les plus grands jusqu'à ce qu'il n'y ait plus de cercles à placer.

Le problème d'optimisation étant là plus complexe qu'avec le placement au hasard, nous n'avons pas pu obtenir un programme fonctionnel pour développer plus à ce sujet là.

Par manque de temps donc, nous n'avons pas non plus pu obtenir un programme fonctionnel pour le placement de sphères dans une zone 3D ouverte.

*

CONCLUSION :

• En conclusion, nous avons réussi à obtenir quelques bons résultats au niveau de l'ordonnancement de tâches et la gestion de cercles et de sphères, mais nous n'avons malheureusement, dans aucun domaine, réussi à aboutir à l'objectif final.

Cet "échec" au niveau de l'objectif final étant un choix, nous avons en effet préféré développer et réfléchir à chaque sujet pour pouvoir avoir une sorte de polyvalence et pouvoir développer un peu plus sur chaque sujet au lieu d'en perfectionner un seul.

Cependant, avec plus de temps et bien entendu une meilleure organisation puisque celle ci était notre première dans le domaine de la recherche, nous aurions pu aboutir à quelque chose de bien complet pour au moins 2 sujets sur les 3 proposés, ce qui aurait pu largement améliorer notre satisfaction pour nos rendus.

SOURCES PRINCIPALES :

https://www.youtube.com/watch?v=QHEQuoIKgNE

https://www.youtube.com/watch?v=XATr_jdh-44

https://www.youtube.com/watch?v=MvZAjkD8-wk

https://mathworld.wolfram.com/CirclePacking.html

https://en.wikipedia.org/wiki/Circle_packing

https://php.developpez.com/cours/circle-packing/

https://www.youtube.com/watch?v=iJh8dvEt03E&t=2s

https://www.youtube.com/watch?v=p4Iz0XJY-Qk&t=855s

https://www.cristal.univ-lille.fr/~decomite/papers/circlePacking.pdf

https://www.r-graph-gallery.com/circle-packing.html