Projet Informatique – Sections Electricité et Microtechnique

Printemps 2021 : *Planet Donut* © R. Boulic & collaborators

Rendu1 (28 mars 23h59)

Objectif de ce document : *en plus de préciser ce qui doit être fait, ce document identifie des ACTIONS à considérer pour réaliser le rendu de manière rigoureuse. Ces ACTIONS sont équivalentes à celles indiquées pour le projet d'automne ; elles ne sont pas notées, elles servent à vous organiser. Vous pouvez adopter une approche différente du moment que vous respectez l'architecture minimale du projet (donnée Fig 6a). La différence principale avec le projet du semestre d'automne est qu'avec la compilation séparée il y a moins de contraintes concernant l'ordre d'exécution des actions, c'est pourquoi elles n'ont pas de numéro. Il y a aussi plus de liberté dans les choix de structuration des données ; vous avez une certaine marge de manœuvre si les options proposées ne vous conviennent pas. Sans surprise on retrouvera l'approche introduite avec la série théorique du Topic1 sur les méthodes de développement de projet. De ce point de vue le rendu1 est un travail de* scaffolding *pour effectuer le* test unitaire *du module* **geomod***.*

1. Buts du rendu1

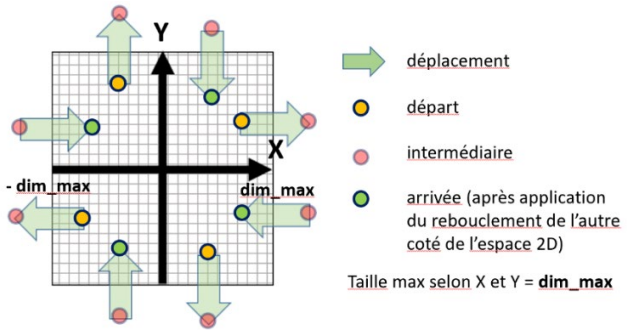
Le but du rendu1 est de valider individuellement les principales fonctions du module générique indépendant de bas-niveau **geomod** (section 7.3.1 de la donnée). Ce module présente la particularité de devoir gérer le rebouclement de l'espace selon x et selon y. Cela a des conséquences sur les concepts suivant : position, vecteur, norme, calcul d'intersection point-cercle ou cercle-cercle, et même dessin (ce dernier point sera traité au rendu3).

Une fois ces fonctions validées elles pourrons être utilisées pour le rendu2 pour vérifier s'il y a certaines intersections interdites au moment de la lecture d'un fichier et pour construire le voisinage des robots et pour le rendu3 pour la mise en œuvre de la simulation.

La section suivante précise ce qu'est un espace modulaire et les fonctions que doit fournir le module geomod.

2. Qu'est-ce qu'un espace modulaire et quelles en sont les conséquences ?

Un espace modulaire est un espace 2D qui se reboucle sur lui-même horizontalement (quand on sort du côté droit on rentre du côté gauche et vice versa) et verticalement (quand on sort du côté haut on rentre du côté bas et vice versa). Cette propriété de l'espace est illustrée dans la Fig2 de la donnée pour la Planète Donut.



propriété de rebouclement d'un espace modulaire 2D

2.1 Précision des calculs

On demande de travailler avec des nombres à virgule flottante en double précision (type **double**).
Une variable **epsilon_zero** sera déclarée dans l'espace de noms non-nommé de **geomod** afin d'être accessible par toutes les fonctions de ce module (uniquement). Sa valeur dépend de la valeur **max** du domaine (cf 2.2).

Une fonction **equal_zero** renvoie **vrai** si la valeur absolue du paramètre de type **double** est strictement inférieure à **epsilon_zero** (et renvoie **faux** sinon).

2.2 définition du maximum [-max, max] selon X et Y

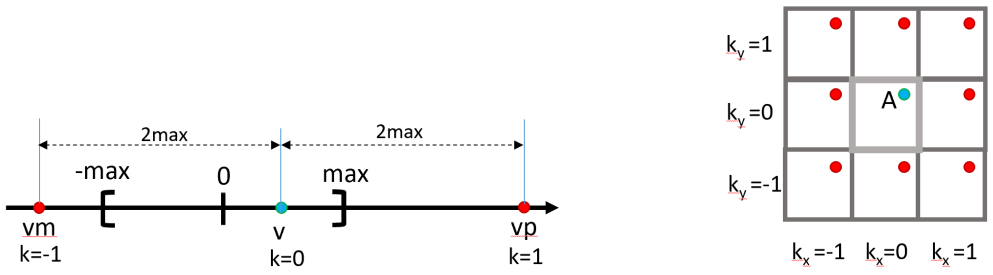
Le module **geomod** doit travailler dans un contexte général, c'est pourquoi la valeur maximum [-**max, max**] selon X et X est mémorisée dans une variable **max** de l'espace de nom non-nommé de **geomod** ; cela permet à toutes les fonctions de ce module d'y accéder. Nous posons que cette valeur est nulle par défaut. Il faut donc une fonction *setter* pour la définir avec *une valeur strictement positive* et une fonction *getter* pour obtenir sa valeur courante pour vérification.

La fonction *setter* de **max** met automatiquement à jour **epsilon_zero** avec la valeur : $\mathbf{10^{-10}.max}$ **.** Une fonction *getter* doit être fournie pour obtenir la valeur courante de **epsilon_zero** pour vérification.

2.3 Coordonnées équivalentes du fait du rebouclement de l'espace

Soit une coordonnée de valeur **v** comprise dans l'intervalle **[-max, max].** Du fait du rebouclement de l'espace cette coordonnée coïncide avec toutes les coordonnées de valeur **v +2kmax** avec **k** un entier relatif. Nous nous limiterons à **k** valant **-1** et **+1**, et nous noterons **vm** et **vp** les valeurs correspondantes.

Par défaut le module travaille avec des points 2D dont les 2 coordonnées sont normalisées, c'est-à-dire comprises dans l'intervalle **[-max, max].** Cependant il est nécessaire de construire des points équivalents pour certaines opérations comme la construction d'un vecteur et les tests d'intersection comme décrit plus loin.



a **b**

Fig1 : coordonnées équivalentes ; (a) dans un espace 1D, vm et vp sont les coordonnées équivalentes de v. (b) dans un espace 2D, les points rouges sont équivalents au point bleu A

2.4 Normalisation de coordonnée et de point 2D

Inversement il est nécessaire de fournir une fonction qui renvoie la valeur *normalisée*, c'est-à-dire *appartenant à l'intervalle [-max, max]* pour toute valeur **v'** fournie en paramètre.

Si v' appartient déjà à l'intervalle [-max, max], sa valeur reste inchangée.
Sinon le résultat est la valeur appartenant à]-max, max[en ajoutant ou soustrayant la quantité 2max.

Une telle fonction doit être disponible (surcharge) pour une coordonnée scalaire ainsi que pour un point 2D.

Remarque : on pose que les points ou centre de cercles fournis pour les fonctions suivantes ont des coordonnées normalisées afin d'éviter ce calcul supplémentaire. Si nécessaire, il convient d'utiliser la fonction de cette section avant d'appeler les fonctions des sections 2.5 à 2.9.

2.5 vecteur construit à partir de 2 points dans le plan 2D

La construction d'un vecteur AB à partir de deux points A et B est utile pour deux opérations :

- Connaître la plus courte distance séparant A de B
- Connaître la direction du plus court chemin de A vers B

Une illustration dans un espace à une dimension montre qu'il faut considérer les coordonnées équivalentes de B car l'une d'entre elles peut produire une distance plus courte, comme Bp dans la figure ci-dessous. Nous posons donc que le vecteur AB est finalement obtenu en calculant ABp car sa norme est plus petite que la distance de A à B.

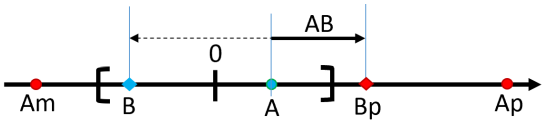


Fig 2 : prise en compte des coordonnées équivalentes pour le calcul d'un vecteur AB car on retient les coordonnées équivalentes de B qui produisent la plus petite norme.

Dans un espace à deux dimensions il faut considérer 3^2 cas de coordonnées équivalentes pour le point B comme illustré dans la figure 3 car chacune des 2 dimensions à 3 possibilités. Dans la figure 3 c'est le point $B_{(1,1)}$, c'est-à-dire de coordonnées équivalente obtenues pour k_x=1 et k_y=1, qui produit la plus petite distance à partir de A

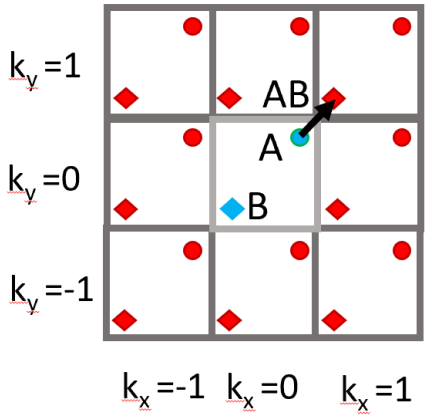


Fig 3 : prise en compte des points équivalents à B (losanges) pour le calcul d'un vecteur AB en 2D.

On demande donc au module **geomod** de fournir une fonction avec les propriétés suivantes :

- entrées non-modifiées :
  - Un point de départ (on n'exploite pas ses points équivalents)
  - Un point d'arrivée pour lequel on analyse les points équivalents
- entrée modifiée : le vecteur dont la norme est **strictement la plus petite**
- sortie : la norme du vecteur

Cette fonction peut être offerte avec des variantes (surcharge ou valeur par défaut).

<u>Traitement du cas particulier d'égalité entre plusieurs points équivalents</u> :

Selon la spécification ci-dessus, en cas d'égalité de normes, la fonction va retourner le vecteur qui est trouvé le premier. L'ordre d'examen des points équivalents est donc important. C'est pourquoi, pour garantir de trouver le même vecteur que le programme de démo, nous demandons d'examiner $\mathbf{k_x}$ de -1 à +1 en tant que variable de boucle externe et $\mathbf{k_y}$ de -1 à +1 en tant que variable de boucle interne.

Il est maintenant possible de construire d'autres fonctions à l'aide de la fonction définie ici (ou ses variantes).

2.6 Test d'égalité de 2 points

Pour tester l'égalité entre deux points, il faut proposer une fonction qui demande le calcul du vecteur reliant le premier point au second point fourni en paramètre avec la fonction définie en 2.5. Si la norme obtenue est strictement inférieure à **epsilon_zero** alors le test d'égalité renvoie vrai. Sinon il renvoie faux.

Une combinaison de la fonction définie en 2.5 et de la fonction **equal_zero** définie en 2.1 est aussi possible.

2.7 Test d'appartenance d'un point à un cercle

Il faut proposer une fonction qui demande le calcul du vecteur reliant le point au centre du cercle fourni en paramètre avec la fonction définie en 2.5.
Pour garantir que le point est bien à l'intérieur du cercle, on pose que **n** est la norme ainsi obtenue et **r** le rayon du cercle : si **n < r - epsilon_zero** alors le test d'égalité renvoie **vrai**. Sinon il renvoie **faux**.

2.8 Test d'intersection de deux cercles

A nouveau on utilise la fonction de calcul de vecteur en lui fournissant les centres des 2 cercles en paramètres.

Comme pour le test d'appartenance, on veut que le booléen **vrai** garantisse que les 2 cercles sont vraiment dans un état d'intersection (le cas tangent renvoie faux). Pour cela on pose que **n** est la norme du vecteur reliant les centres et **r1** et **r2** les rayons des deux cercles : si **n < r1 + r2 - epsilon_zero** alors le test d'égalité renvoie **vrai**. Sinon il renvoie **faux**.

2.9 Autres fonctions qu'il faudra mettre au point à partir du rendu3

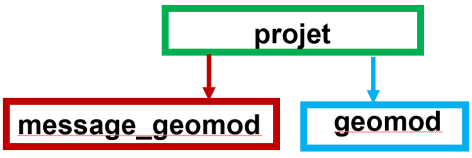
Pour le rendu3 il faudra utiliser le module **graphic** pour demander le dessin de points, lignes, cercles. Le dessin d'une ligne reliant deux points demande plus d'attention car il faut dessiner la ligne de norme minimum (ce qui est facile à obtenir avec le calcul de vecteur) MAIS il faut le faire deux fois : une fois depuis le point de départ de la ligne et une fois depuis le point d'arrivée. Le dessin d'un cercle demandera enfin de tester si les cercles équivalents débordent sur l'espace visible de la planète (comme par exemple dans la Fig 1 de la donnée sur le côté droit).

3. Test unitaire de geomod

Normalement pour effectuer le test unitaire d'un module vous écrivez un programme de test (*scaffolding*) qui effectue des appels de toutes les fonctions offertes par le module et ce programme compare le résultat renvoyé par chaque appel au résultat attendu (que vous avez calculé indépendamment par un autre moyen).

Dans le cadre du rendu1 nous devons adapter cette approche pour nous permettre de vérifier automatiquement que votre code est correct (il fait ce qu'il est supposé faire).

L'architecture à utiliser est celle de la Fig6a de la donnée rappelée ci-dessous. Le module de test s'appelle **projet.** La fonction **main** de son implémentation **projet.cc** analyse les arguments fournis sur la ligne de commande pour effectuer **un** appel d'une fonction ou de plusieurs fonctions de votre module **geomod**. Le résultat est affiché à l'aide de fonctions prédéfinies que nous mettons à disposition dans le module **message_geomod** ; il ne doit pas être modifié. Si nécessaire la fonction main() fait appel à d'autres fonctions en vertu du principe d'abstraction.



L'exécutable du rendu1 doit être obtenu par la commande **make** sur la VM du cours et s'appeler **projet** .
Voici la syntaxe d'un appel de votre exécutable :

```
./projet i  a b c d e …
```

La valeur indiquée par l'argument **`i`** désigne le scénario du test à effectuer. Le reste des arguments sert à fournir les paramètres du test comme détaillé dans la section suivante.

3.1 Scénarios de test avec utilisation du module message_geomod

Le nombre d'arguments dépend du scénario. Ceux-ci sont rassemblés dans la table suivante :

`i`	**ordre et nature des arguments suivants (tous les arguments sont de type double)**	**Fonctions à appeler/tester**	**Utiliser cette fonction de message_geomod pour l'affichage :**
1	**init_max** cet argument est fourni en premier pour tous les tests suivants ; il est complété avec d'autres arguments.	Section 2.1 et 2.2 : d'abord la fonction setter de **max** avec **init_max** ; puis récupérer **max** et **epsilon_zero** avec leur fonctions getter	**print_max** affiche dans l'ordre : **init_max**, **max** et **epsilon_zero**
2	**init_max init_x init_y** (coordonnées d'un point)	Initialiser **max** avec **init_max** puis appeler : Section 2.4 : Normalisation du point	**print_point** affiche les coordonnées initiale et les coordonnées normalisées
3	**init_max Init_x1 init_y1 init_x2 init_y2** (coordonnées normalisées de 2 points)	Initialiser **max** avec **init_max** puis appeler : Section 2.5 : construction d'un vecteur	**print_vect** affiche la norme et les coordonnées x y du vecteur
4	**init_max Init_x1 init_y1 init_x2 init_y2** (coordonnées normalisées de 2 points)	Initialiser **max** avec **init_max** puis appeler : Section 2.6 : test d'égalité de 2 points	**print_equal_point** affiche le booléen et les coordonnées des 2 points.
5	**init_max init_x init_y init_xc init_yc init_r** (coordonnées d'un point suivi par le centre du cercle et son rayon)	Initialiser **max** avec **init_max** puis appeler : Section 2.7 : appartenance du point au cercle	**print_include_point** affiche le booléen et les coordonnées des 2 points et du rayon
6	**init_max init_xc1 init_yc1 init_r1 init_xc2 init_yc2 init_r2** (centre et rayon de 2 cercles)	Initialiser **max** avec **init_max** puis appeler : Section 2.8 : intersection de 2 cercle	**print_intersect** affiche le booléen et les centres et rayon des 2 cercles

L'interface **message_geomod.h** fournit aussi une fonction **bad_argc()** qu'il faut appeler si le nombre d'argument est incorrect.

3.2 Méthode de travail

Plusieurs approches sont possibles ; dans tous les cas il est important de compiler fréquemment, c'est-à-dire au fur et à mesure de l'écriture de chaque module (recompiler après l'écriture de quelques lignes).

3.2.1 ACTION : test du module projet / de la ligne de commande

Pour le module **projet**, la fonction **main()** peut très rapidement être testée pour savoir si le bon nombre d'argument est fourni pour chaque valeur de i. Si oui, effectuer la conversion des chaines de caractères en valeurs numériques de type double (cours sem1 Topic11 slide 8 et série12) et affichez -les. Si non appelez la fonction **bad_argc()** de **message_geomod** et quittez.

3.2.2 ACTION : test du Makefile

A partir du dessin de l'architecture du rendu1 (page précédente) en déduire les dépendances et écrire le fichier Makefile. Testez-le avec des modules contenant le minimum pour être compilable et exécutable, c'est-à-dire les includes et une fonction main vide ou simplement avec affichage d'un message.

3.2.3 ACTION : définition de l'interface de geomod et développement séparé

A partir des indications de ce rendu décidez les noms des fonctions et leur prototypes et mettez les fonctions exportées dans geomod.h. Une fois cela fait une personne peut developper main (action 3.2.1) et l'autre peut écrire et compiler geomod. La personne qui développe main() peut appeler des stubs de geomod pour faire ses tests avec message_geomod.

3.2.4 ACTION : intégration

Un fois que le module projet fonctionne avec le traitement des arguments et que le module geomod compile, on peut passer au test unitaire en intégrant les appels de fonctions de geomod dans main(). C'est votre responsabilité de trouver un nombre suffisant d'exemples pertinents pour s'assurer qu'on n'oublie pas de cas particuliers.

Autograder non garanti: nous évaluons actuellement s'il est possible d'adapter l'autograder du premier semestre au contexte du rendu1. S'il est fourni il sera utilisé pour noter l'exécution du rendu1.

Dans tous les cas, un autograder ne peut faire qu'un nombre limité de tests et ne peut pas garantir l'absence de bugs pour d'autres tests. Donc, commencez à *organiser votre propre batterie de tests* indépendamment de cette possibilité.

4. Forme du rendu1

Documentation : l'entête de vos fichiers source doit indiquer les noms des membres du groupe.

Rendu : pour chaque rendu **UN SEUL membre d'un groupe** (noté **SCIPER1** ci-dessous) doit télécharger un fichier **zip** sur moodle (pas d'email). Le non-respect de cette consigne sera pénalisé de plusieurs points. Le nom de ce fichier **zip** a la forme :

SCIPER1_ SCIPER2.zip

Compléter le fichier fourni **mysciper.txt** en remplaçant 111111 par le numéro SCIPER de la personne qui télécharge le fichier archive et 222222 par le numéro SCIPER du second membre du groupe.

Le fichier archive du rendu1 doit contenir (**aucun répertoire**) :

- Fichier texte édité **mysciper.txt**
- Votre fichier **Makefile** produisant un executable **projet**
- votre code source (.cc et .h)

*On doit pouvoir produire l'exécutable **projet** à partir du Makefile après décompression du contenu du fichier zip.*

Auto-vérification : Après avoir téléversé le fichier zip de votre rendu sur moodle (upload), téléchargez-le (download), décompressez-le et assurez-vous que la commande make produit bien l'exécutable et que celui-ci fonctionne correctement.

Exécution sur la VM: votre projet sera évalué sur la VM du second semestre (disponible depuis le 6 mars).

Backup : Vous êtes responsable de faire votre copie de sauvegarde du projet. Il y a un backup automatique seulement sur votre compte myNAS. Sur la VM, vous devez activer vous-même le backup (icone « engrenage » en haut à droite, choisir system settings, choisir backup et activer cette fonction en précisant les paramètres). Une alternative est de s'envoyer la dernière version du code source par email après chaque session de travail.

Gestion du code au sein d'un groupe :

- vous pouvez envisager d'utiliser **gdrive.epfl.ch** pour définir un répertoire partagé par les 2 membres du groupe et pas plus. Il n'y a pas d'éditeur de code en mode partagé.

5. Date du rendu1

La date du rendu1 est le 28 mars à 23h59