E3FI - 2022 - 2023 - Mathématiques 3D - TD #3

Détection de collision d'une *Sphere* en mouvement et d'un ensemble de *Box / RoundedBox* statiques

par David Bilemdjian

Pré-requis:

- TD#1 et TD#2 complétés, mais plus spécifiquement:
    - Méthode d'affichage d'une *Sphere* et d'une *RoundedBox*
    - Méthode de calcul d'intersection d'un *Segment* avec une *RoundedBox*

Objectifs du TD:

- Modélisation et développement du comportement dynamique de la *Sphere* soumise à un champ de gravitation
- Développement de la méthode de détection de collision entre une *Sphere* en mouvement rectiligne uniforme et une unique *Box* (ou *RoundedBox*) statique.
    - La position de la *Sphere* et sa direction de mouvement après rebond sont calculées.
- Développement de la méthode de détection de collision entre une *Sphere* en mouvement rectiligne uniforme et **un groupe** de *Box* (ou *RoundedBox*) statiques.
    - La position et la direction de mouvement de la *Sphere* après rebond sont calculées.
    - Cette méthode s'appuie bien évidemment sur la méthode précédente
- Développement de la méthode de détection de **collisions multiples** (le temps d'une frame) entre une *Sphere* en mouvement rectiligne uniforme et **un groupe** de *Box* (ou *RoundedBox*) statiques.
    - La position et la direction de mouvement de la *Sphere* après rebond(s) sont calculées.
    - Cette méthode s'appuie bien évidemment sur la méthode précédente

Modélisation et développement du comportement dynamique de la *Sphere* soumis à un champ de gravitation

Rappel de certains théorèmes de la dynamique newtonienne

Soit un solide indéformable.

Théorème du centre d'inertie: le mouvement de son centre d'inertie G est régi par l'équation

$$\sum \overrightarrow{F_{ext}} = m \times \overrightarrow{a_G}$$

Théorème du moment cinétique: ce théorème va nous permettre de prédire le comportement en rotation de la *Sphere*

$$\frac{\mathrm{d}\overrightarrow{L_G}}{\mathrm{d}t} = \sum \overrightarrow{M_G}\left(\overrightarrow{F_{ext}}\right)$$

- où $\overrightarrow{L_G}$ est le moment cinétique du solide en son centre de gravité
- et $\sum \overrightarrow{M_G}\left(\overrightarrow{F_{ext}}\right)$ la somme des moments des forces extérieures, calculés au centre de gravité

Formule du moment cinétique d'une sphère homogène en rotation selon un axe passant par son centre de gravité

- Pour une sphère de masse volumique homogène, son centre de gravité *G* est confondu avec son centre géométrique Ω
- et la magnitude de son moment cinétique vaut $\left\|\overrightarrow{L_G}\right\| = I \times \dot{\theta}^2$
  Avec
  - I, moment d'inertie d'une sphère homogène: $I = \frac{2}{5}mR^2$
  - $\dot{\theta}$ vitesse angulaire de rotation en *rad/s*
- La direction de l'axe de rotation de la *Sphere* est donnée par la direction de $\overrightarrow{L_G}$

Ainsi, si $\overrightarrow{L_G}$ est connu à un instant *t*, la direction du vecteur représente l'axe de rotation du solide, et sa norme est proportionnelle à la vitesse angulaire $\dot{\theta}$. Il est donc aisé de construire:

- un vecteur $\overrightarrow{Rot}$ de rotation de la *Sphere*, utile pour le calcul de la force de frottement au moment du contact entre la *Sphere* et l'obstacle (principe détaillé plus loin)
  - la direction de $\overrightarrow{Rot}$ est la direction de $\overrightarrow{L_G}$
  - la norme de $\overrightarrow{Rot}$ correspond à $\dot{\theta}$
- puis de manière évidente le quaternion de rotation instantanée qui assurera informatiquement la rotation effective de l'objet *Sphere*
- $\dot{\theta}$ devra être limité pour éviter un emballement du mouvement

Translation du centre de gravité de la *Sphere*

La *Sphere* de masse *m*, de rayon *r* et de centre Ω, est assimilée, du point de vue de la Dynamique (https://fr.wikipedia.org/wiki/Dynamique_(m%C3%A9canique)) , à un point matériel situé en son centre.

La seule force en présence est ici la force gravitationnelle $m\overrightarrow{g}$ due au champ de pesanteur terrestre.

L'équation différentielle qui régit le mouvement de la *Sphere* se réduit donc à $\overrightarrow{a} = \overrightarrow{g}$.

En intégrant l'équation du mouvement par la méthode d'Euler implicite (https://fr.wikipedia.org/wiki/M%C3%A9thode_d%27Euler), la vitesse et la position de la *Sphere* peuvent à chaque frame être calculées ainsi:

$$\overrightarrow{v}_{n+1} = \overrightarrow{v}_n + (t_{n+1} - t_n) \times \overrightarrow{g}$$

$$\overrightarrow{O\Omega}_{n+1} = \overrightarrow{O\Omega}_n + (t_{n+1} - t_n) \times \overrightarrow{v}_{n+1}$$

où

- $t_{n+1} - t_n$ correspond au temps écoulé entre 2 frames, *i.e.* informatiquement la variable *deltaTime*
- O est l'origine du référentiel dans lequel est étudié le mouvement, a priori le référentiel du Monde

Les arrondis de calcul, inhérents à tout algorithme, auront pour effet ici de ralentir la *Sphere* au cours du temps. Pour prévenir cela, il faut à chaque frame re-calculer la vitesse scalaire de la *Sphere* par application du principe de conservation de l'énergie. Voici comment.

La *Sphere* étant soumise uniquement à son poids, qui est une force conservative (https://fr.wikipedia.org/wiki/Force_conservative), on peut écrire:

$$E = E_c + E_p$$

$$E = \frac{1}{2}mv^2 + mgh$$

où *E* est l'énergie totale du système, E_c l'énergie cinétique et E_p l'énergie potentielle de pesanteur. Les énergies cinétique et potentielle fonctionnent ici un peu comme des vases communicants.

En prenant une altitude de référence y_0, il est possible à chaque frame de déterminer *h* puis v par les formules:

$$h = y - y_0$$

$$v = \sqrt{2\left(\frac{E - mgh}{m}\right)}$$

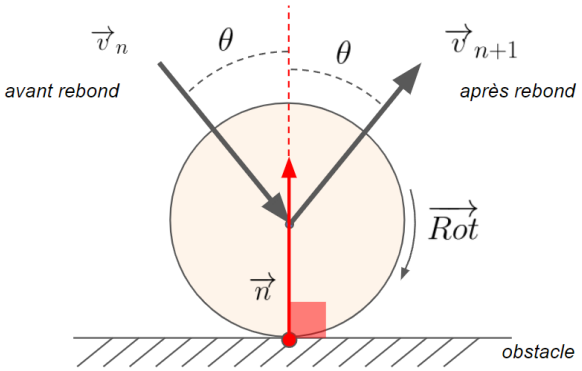
Nous considérons l'énergie totale du système *E* constante.
Cependant, si vous le souhaitez, vous pouvez, en sortie de chaque collision, appliquer à *E* un coefficient multiplicateur de perte d'énergie, inférieur à 1. Vous devez tout de même garder à l'esprit que les algorithmes de détection de collision proposés ici présupposent que la *Sphere* est toujours en mouvement: en deçà d'une certaine vitesse les algorithmes risquent de dysfonctionner.

Modélisation dynamique simplifiée du Rebond

Les obstacles sont considérés statiques.
Seule la *Sphere* est en mouvement:

- son mouvement de translation est considéré linéaire et uniforme le temps d'une frame, caractérisé par son vecteur vitesse $\overrightarrow{v}$
- son mouvement de rotation est également considéré uniforme le temps d'une frame (vitesse angulaire et axe de rotation constants), caractérisé par son vecteur de rotation $\overrightarrow{Rot}$

Voici les caractéristiques d'un **rebond idéal** lors d'un choc dit élastique; Nous nous intéressons ici uniquement à la *Sphere*, seul objet en mouvement:



- La *Sphere* ne change pas de forme et sa masse reste constante pendant le rebond
- Aucun frottement n'a lieu entre la *Sphere* et l'obstacle au moment du contact:
    - Aucune énergie n'est donc dissipée sous forme de vibrations et/ou de chaleur,
    - L'énergie cinétique de la *Sphere* étant conservée, sa vitesse scalaire l'est également
    - le moment cinétique de la *Sphere* est conservé, donc elle conserve également son vecteur de rotation $\overrightarrow{Rot}$
    - Les vecteurs vitesse avant et après rebond sont symétriques opposés par rapport à la normale $\overrightarrow{n}$ à la surface de collision

Si nous appliquons à la lettre tous ces principes du rebond idéal lors d'un choc élastique, nous aurons la satisfaction de voir la *Sphere* changer de direction de manière plutôt réaliste après rebond, mais nous serons assez déçus de ne pas constater de changement au niveau de sa rotation.

Ainsi, nous allons considérer que le choc n'est pas idéal et qu'il existe une force de frottement $\overrightarrow{F_f}$ au niveau du point de contact pendant toute la durée du contact (...le temps d'une frame).
Cette force de frottement doit théoriquement perturber les règles du rebond idéal listées ci-dessus, et toutes les grandeurs cinétiques s'en trouvent logiquement modifiées.
Nous n'allons pourtant retenir que la modification du moment cinétique de la sphère, ce qui aura pour conséquence la modification de son vecteur de rotation $\overrightarrow{Rot}$.

Pour simplifier le calcul de la force de frottement, nous considérerons que celle-ci est proportionnellement opposée à la composante tangentielle de la vitesse relative entre les deux objets (*Sphere* & obstacle) au niveau de leur point de contact.

- $\overrightarrow{F_f} = -k_f \times \overrightarrow{v_{t_{rel}}}$
- où k_f est un coefficient de friction positif qui varie théoriquement selon les caractéristiques géométriques et physique des objets en contact, leurs états de surface, leurs matériaux, mais également leurs vitesses de translation et de rotation. Il s'agit d'un coefficient qui se détermine donc expérimentalement et pour simplifier, nous le considérerons constant.

L'obstacle étant statique, la vitesse relative entre les deux objets n'est rien d'autre que la vitesse du point de la *Sphere* en contact avec l'obstacle.

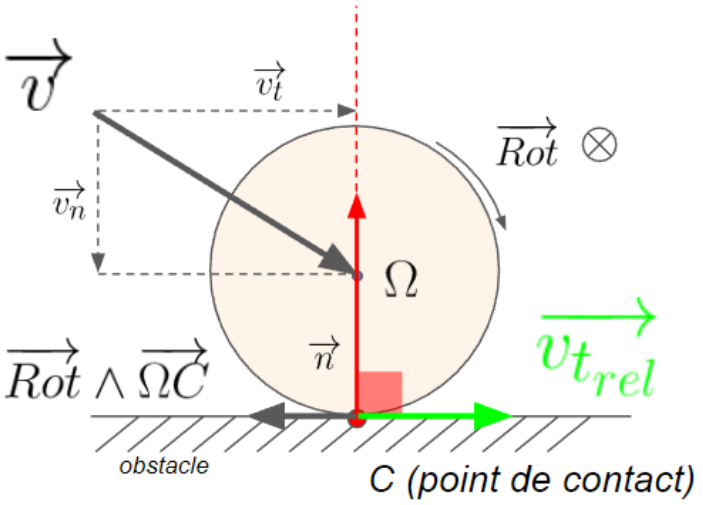
Cette vitesse se compose de deux parties (voir schéma ci-desous):

1. la vitesse de translation de la *Sphere*: $\overrightarrow{v} = \overrightarrow{v_n} + \overrightarrow{v_t}$
    - où $\overrightarrow{v_n}$ est la composante de $\overrightarrow{v}$ normale à la surface de contact, donc colinéaire à $\overrightarrow{n}$
    - et $\overrightarrow{v_t}$ la composante tangentielle de $\overrightarrow{v}$
    - il s'en suit que $\overrightarrow{v} = (\overrightarrow{v} \cdot \overrightarrow{n})\,\overrightarrow{n} + \overrightarrow{v_t}$, et $\overrightarrow{v_t} = \overrightarrow{v} - (\overrightarrow{v} \cdot \overrightarrow{n})\,\overrightarrow{n}$
2. la vitesse tangentielle (au point de contact) due à la rotation de la *Sphere*: $\overrightarrow{Rot} \wedge \overrightarrow{\Omega C}$ (l'opérateur "V à l'envers" est l'opérateur "produit vectoriel")

Ainsi la composante tangentielle de cette vitesse relative est:

- $\overrightarrow{v_{t_{rel}}} = \overrightarrow{v_t} + \overrightarrow{Rot} \wedge \overrightarrow{\Omega C}$
- soit

$$\overrightarrow{v_{t_{rel}}} = \overrightarrow{v} - (\overrightarrow{v} \cdot \overrightarrow{n})\,\overrightarrow{n} + \overrightarrow{Rot} \wedge \overrightarrow{\Omega C}$$



(attention: pour un souci de clarté le schéma ci-dessus propose une configuration particulière en 2D, où $\overrightarrow{Rot}$ est orthogonal au plan de la feuille. Notre *Sphere* tourne réellement en 3D, donc $\overrightarrow{Rot}$ pourra adopter n'importe quelle direction, et les vecteurs $\overrightarrow{v}$, $\overrightarrow{n}$ et $\overrightarrow{v_{t_{rel}}}$ ne seront pas forcément dans un même plan)

En intégrant par la méthode d'Euler l'équation différentielle proposée par le théorème du moment cinétique, nous pouvons calculer à chaque frame le moment cinétique ainsi:

$$\overrightarrow{L_{Gn+1}} = \overrightarrow{L_{Gn}} + (t_{n+1} - t_n) \times \left(\overrightarrow{\Omega C} \wedge \overrightarrow{F_f}\right)$$

$$\overrightarrow{L_{Gn+1}} = \overrightarrow{L_{Gn}} - k_f\,(t_{n+1} - t_n) \times \left[\overrightarrow{\Omega C} \wedge \left(\overrightarrow{v} - (\overrightarrow{v} \cdot \overrightarrow{n})\,\overrightarrow{n} + \overrightarrow{Rot} \wedge \overrightarrow{\Omega C}\right)\right]$$

Le moment cinétique initial de la *Sphere* peut tout à fait être choisi nul:

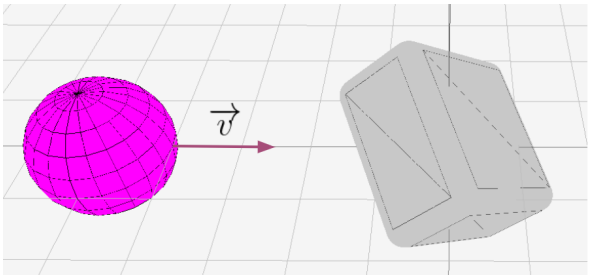
$$\overrightarrow{L_{G0}} = \overrightarrow{0}$$

k_f sera choisi de telle manière à obtenir un comportement en rotation visuellement plaisant

Développement de la méthode de détection de collision entre une *Sphere* en mouvement rectiligne uniforme et une **unique** *Box* (ou *RoundedBox*) statique

Nous considérons ici:

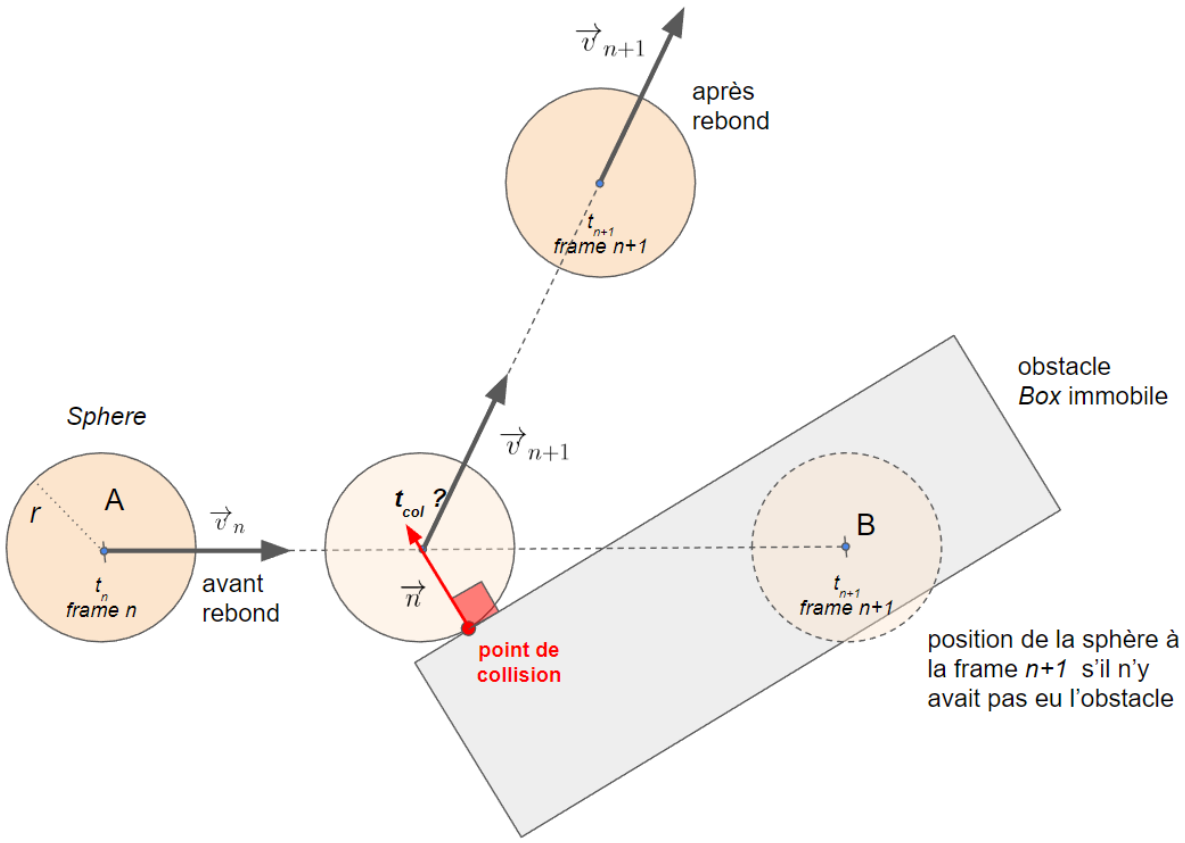
- une *Sphere* de rayon *r* et de masse *m* , de vitesse constante $\vec{v}$ durant le temps de la frame
- une *Box* (possiblement une *RoundedBox*) orientée et immobile dans l'espace



A chaque frame du jeu, il faut déterminer si, au cours du lapse de temps qui va s'écouler jusqu'à la frame suivante, la *Sphere* collisionne avec la *Box / RoundedBox*. En cas de collision, il faut alors calculer les éléments cinématiques de la *Sphere* après rebond, à savoir:

- sa nouvelle position,
- son nouveau vecteur vitesse,
- son nouveau vecteur de rotation

Pour des raisons de simplification, la situation de collision est ici illustrée en 2D. Les principes sous-jacents restent identiques en 3D:



Explications:

- d'une frame *n* à une frame *n+1* une *Sphere* doit théoriquement parcourir le segment *[AB]*

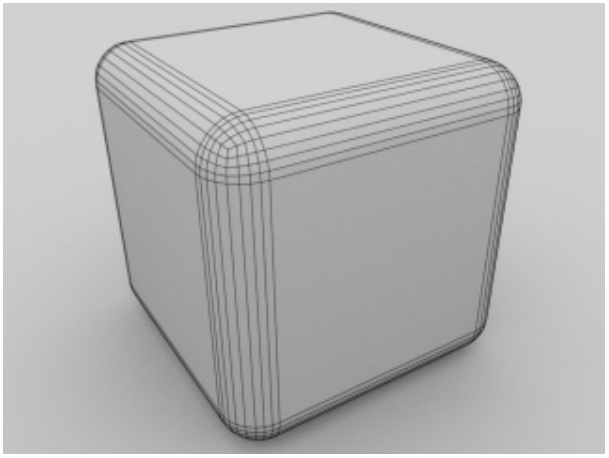
  avec $\overrightarrow{AB} = (t_{n+1} - t_n) \times \overrightarrow{v_n}$

- or, le long de ce parcours, la *Sphere* collisionne avec une *Box* statique
- le point de "premier" contact entre la *Sphere* et la *Box* est forcément détecté à un temps t_{col} compris entre t_n et t_{n+1}
- le vecteur vitesse de la *Sphere* est recalculé dans la direction du rebond:
  - nous considérons un rebond idéal, à savoir que $\overrightarrow{v}_n$ **et** $\overrightarrow{v}_{n+1}$ **sont symétriques opposés par rapport à la normale** $\overrightarrow{n}$ **de la surface de collision**
  - attention à ne pas confondre *n*, l'indice de frame, avec $\overrightarrow{n}$ le vecteur normal à la surface
- La *Sphere* repart donc dans une nouvelle direction pour parcourir le reliquat de la distance qu'elle aurait dû parcourir, s'il n'y avait pas eu d'obstacle, de sa position de collision au point B.
- Remarque: le spectateur ne voit pas le moment précis de la collision, les étapes décrites ci-dessus ne sont que des étapes mathématiques pour déterminer la nouvelle position après rebond.

Comment calculer la position de la *Sphere* au moment de la collision ?
Le problème géométrique ne semble pas simple, … mais peut être simplifié.

Plutôt que de chercher un premier point de collision entre une *Sphere* et une *Box* lorsque la *Sphere* parcourt un chemin rectiligne, nous allons réduire le problème à la recherche de l'intersection entre un *Segment* (représentant le chemin parcouru par la *Sphere*) et une *RoundedBox*, de rayon *r* identique au rayon de la *Sphere*. Cette *RoundedBox* s'appelle la "Somme de Minkowski de la *Box* avec la *Sphere*".
https://fr.wikipedia.org/wiki/Somme_de_Minkowski

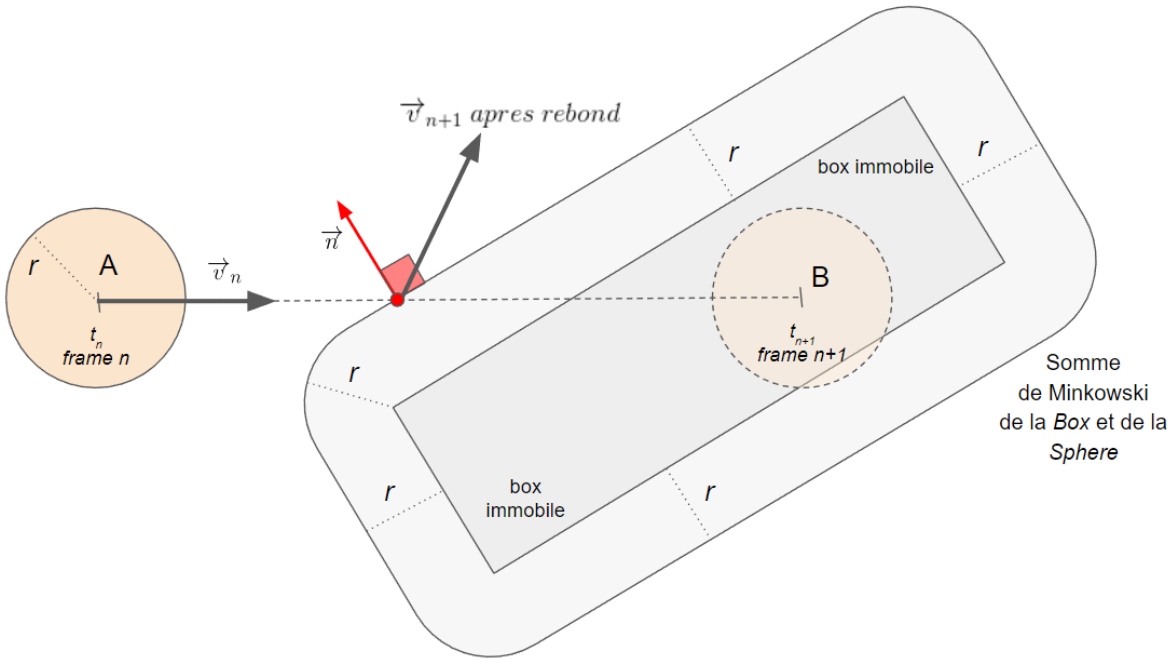
Pour vous aider à comprendre ce qu'est une Somme de Minkowski, imaginez le lieu des points défini par le centre d'une *Sphere* qui roulerait sur la *Box*, sans jamais perdre contact avec elle:



On peut voir sur l'illustration ci-dessus que cette somme de Minkowski peut être décomposée en plusieurs primitives 3D:

- des faces planes rectangulaires: les régions de Voronoï des faces de la box initiale ("les faces de Voronoï")
- des portions de cylindre: les régions de Voronoï des arêtes de la box initiale
- des portions de sphère: les régions de Voronoï des vertices de la box initiale
- Pour chaque arête, si on réunit les régions de Voronoï de l'arête proprement dite et des deux vertices la délimitant, on obtient une région de Voronoï en forme de capsule ("les capsules de Voronoï").

Diagramme et Région de Voronoï:
(https://fr.wikipedia.org/wiki/Diagramme_de_Vorono%C3%AF
Par extension une région de Voronoï est ici interprétée comme étant le lieu des points de la box arrondie les plus proches d'une face, d'une arête ou d'une vertice.
La situation de collision devient donc une situation d'intersection entre le segment *[AB]* et la *RoundedBox*, comme illustré ci-dessous:



Attention: le point d'intersection trouvé correspond à la position du centre de la *Sphere* au moment du "premier" contact avec la *Box*.

Quelques conseils pour optimiser la recherche de l'intersection entre le segment *[AB]* et la *RoundedBox*:

- Déterminez en premier lieu le point d'intersection de la droite *(AB)* avec l'*OBB* (oriented bounding box) de la *RoundedBox*. La seule différence entre la *RoundedBox* et son *OBB* est que cette dernière n'a plus les coins arrondis.
- Examinez la position de ce point d'intersection par rapport au point A et repérez si ce point appartient ou pas à une "face de Voronoï"
- Déduisez-en si votre point est le point d'intersection recherché. Si oui, bingo, sinon poursuivez votre recherche en cherchant le point d'intersection entre le segment *[AB]* et **1** ou **3** "capsules de Voronoï" judicieusement choisies.

La méthode à développer pourrait avoir le prototype suivant:

bool GetSphereNewPositionAndVelocityIfCollidingWithRoundedBox(
Sphere sphere,
RoundedBox rndBox,
Vector3 velocity,
float deltaTime,
float& colT,
Vector3& colSpherePos,
Vector3& colNormal,
Vector3& newPosition,
Vector3& newVelocity)

- *sphere*: la sphère géométrique à l'instant t_n
- *rndBox*: la *RoundedBox* obstacle statique
- *velocity*: les coordonnées du vecteur vitesse de la *sphere*
- *deltaTime*: la durée entre deux frames
- *colT*: la valeur du paramètre d'interpolation entrant dans le modèle vectoriel du segment $[AB]$, au moment de la collision. Rappel*:* le segment $[AB]$ est modélisé par l'équation vectorielle de paramètre *t* réel suivante
  $$\overrightarrow{AM} = t \times \overrightarrow{AB},\ t \in [0,1]$$ (M étant un point quelconque de $[AB]$)
- *colSpherePos*: les coordonnées du centre de la sphère au moment de la collision
- *colNormal*: les coordonnées du vecteur normal à l'obstacle au niveau du point de collision
- *newPosition*: les coordonnées de la nouvelle position de la *Sphere* après rebond
- *newVelocity*: les coordonnées du nouveau vecteur vitesse de la *Sphere* après rebond

La méthode retourne *true* si une collision a été détectée. Les paramètres *colT, colSpherePos, colNormal, newPosition* et *newVelocity* ne sont renseignés par la méthode que si la collision est effective. Ils ne doivent donc être utilisés en retour de méthode que si celle-ci a retourné *true*.

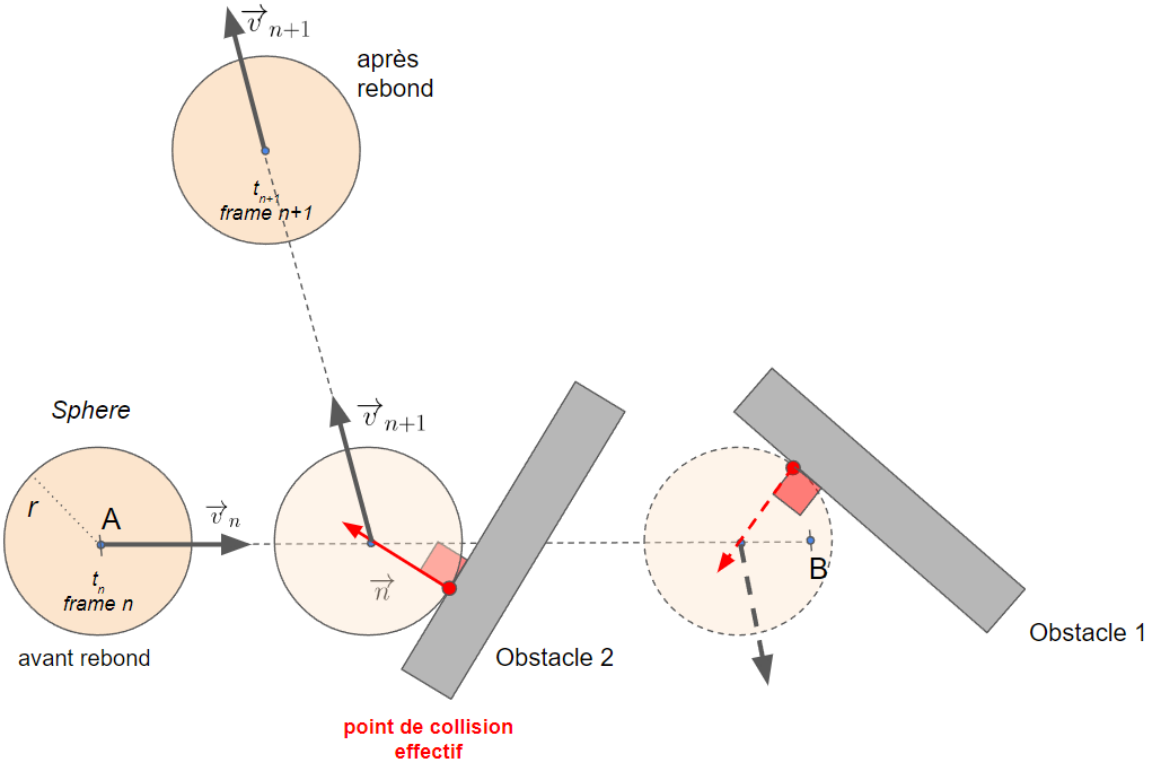
La méthode ne modifie ni la position de la *Sphere*, ni son vecteur vitesse. Ceux-ci doivent être modifiés en sortie de méthode.

Remarque: la somme de Minkowski d'une *RoundedBox* de rayon *R* et d'une *Sphere* de rayon *r* est une *RoundedBox* de rayon *R+r* . Ainsi il est préférable et aisé d'utiliser d'emblée des obstacles de type *RoundedBox* plutôt que des *Box*.

Développement de la méthode de détection de collision entre une *Sphere* en mouvement rectiligne uniforme et **un groupe** de *Box* (ou *RoundedBox*) statiques

Si plusieurs obstacles existent dans la scène, il faut calculer les informations de collision éventuelle de la *Sphere* avec les différents obstacles, pour ne retenir que la position de collision de la *Sphere* la plus proche de sa position d'origine (le point A).

La figure ci-dessous illustre une situation où la *Sphere* rencontre deux obstacles le long de son chemin $[AB]$. Seul la collision avec l'Obstacle 2, plus proche de A, est retenue.



La méthode à développer pourrait avoir le prototype suivant:

bool GetSphereNewPositionAndVelocityIfCollidingWithRoundedBoxes(
Sphere sphere,
const std::vector<RoundedBox>& rndBoxes,
Vector3 velocity,
float deltaTime,
float& colT,
Vector3& colSpherePos,
Vector3& colNormal,
Vector3& newPosition,
Vector3& newVelocity)

- *sphere*: la sphère géométrique à l'instant t_n
- *rndBoxes*: les *RoundedBox* , obstacles statiques

- *velocity*: les coordonnées du vecteur vitesse de la *sphere*
- *deltaTime*: la durée entre deux frames
- *colT*: la valeur du paramètre d'interpolation entrant dans le modèle vectoriel du segment $[AB]$, au moment de la collision. Rappel*:* le segment $[AB]$ est modélisé par l'équation vectorielle de paramètre *t* réel suivante

  $$\overrightarrow{AM} = t \times \overrightarrow{AB},\ t \in [0,1]$$ (M étant un point quelconque de $[AB]$)
- *colSpherePos*: les coordonnées du centre de la sphère au moment de la collision
- *colNormal*: les coordonnées du vecteur normal à l'obstacle au niveau du point de collision
- *newPosition*: les coordonnées de la nouvelle position de la *Sphere* après rebond
- *newVelocity*: les coordonnées du nouveau vecteur vitesse de la *Sphere* après rebond

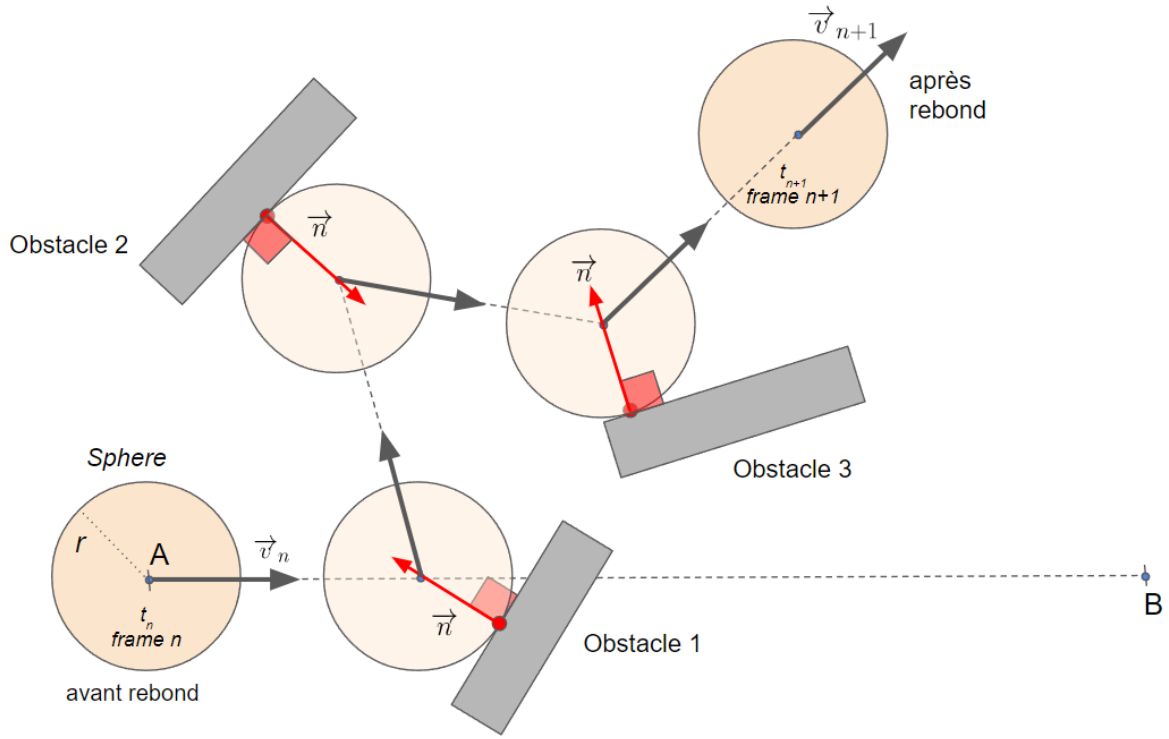
La méthode retourne *true* si une collision a été détectée.
Les paramètres *colT, colSpherePos, colNormal, newPosition* et *newVelocity* ne sont renseignés par la méthode que si la collision est effective. Ils ne doivent donc être utilisés en retour que si la méthode a retourné *true*.
La méthode ne modifie ni la position de la *Sphere*, ni son vecteur vitesse. Ceux-ci doivent être modifiés en sortie de méthode.

Cette méthode doit s'appuyer sur la méthode du chapitre précédent *GetSphereNewPositionAndVelocityIfCollidingWithRoundedBox.*

Optimisation: détecter la collision de la *Sphere* avec tous les obstacles de la scène peut s'avérer coûteux pour le processeur. Le rangement des obstacles dans une structure de type octree (https://fr.wikipedia.org/wiki/Octree) aide à optimiser la recherche de la collision effective, mais l'utilisation de cette technique dépasse les objectifs de ce module.

Développement de la méthode de détection de **collisions multiples** (le temps d'une frame) entre une *Sphere* en mouvement rectiligne uniforme et **un groupe** de *Box* (ou *RoundedBox*) statiques

Au cours d'une frame, il se peut que la *Sphere* subisse mathématiquement plusieurs collisions successives avec les obstacles de la scène. Sur l'illustration ci-dessous la *Sphere* collisionne avec 3 obstacles durant la frame.



La longueur de la ligne brisée "suivie" par le centre de la *Sphere* doit être égale à la longueur du segment $[AB]$, (Rappel: B aurait été la position de la *Sphere* au temps t_{n+1} en l'absence d'obstacles)

La méthode à développer pourrait avoir le prototype suivant:

```
bool GetSphereNewPositionAndVelocityIfMultiCollidingWithRoundedBoxes(
        Sphere sphere,
        const std::vector<RoundedBox>& rndBoxes,
        Vector3 velocity,
        float rotInertia,
        Vector3 angularMomentum,
        float deltaTime,
        int nMaxSuccessiveCollisions,
        Vector3& newPosition,
        Vector3& newVelocity,
        Vector3 newAngularMomentum)
```

- *sphere*: la sphère géométrique à l'instant t_n

- *rndBoxes*: les *RoundedBox ,* obstacles statiques
- *velocity*: les coordonnées du vecteur vitesse de la *sphere*
- *rotInertia:* “rotational inertia”, i.e. le moment d’inertie de la *sphere*, $I = \frac{2}{5}mR^2$
- *angularMomentum :* les coordonnées du vecteur moment cinétique de la *sphere*
- *deltaTime*: la durée entre deux frames
- *nMaxSuccessiveCollisions*: le nombre maximum de collisions successives “autorisées” durant une frame. N’est véritablement utile que dans la situation où la *Sphere* adopterait une très grande vitesse (relativement à la taille des obstacles). Cette valeur permet ainsi de limiter le volume de calculs à effectuer, et donc de prévenir les lags éventuels.
- *newPosition*: les coordonnées de la nouvelle position de la *Sphere* après rebond(s)
- *newVelocity*: les coordonnées du nouveau vecteur vitesse de la *Sphere* après rebond(s)
- *newAngularMomentum*: les coordonnées du nouveau vecteur moment cinétique de la *sphere*

Cette méthode doit, à chaque collision détectée, calculer la nouvelle position de la sphère après rebond, mais également l’ensemble de ses composantes dynamiques, à savoir son nouveau vecteur vitesse et son nouveau vecteur moment cinétique.

La méthode retourne *true* si au moins un rebond a été détecté.
Les paramètres *newPosition, newVelocity* et *newAngularMomentum* ne sont renseignés par la méthode que si la collision est effective. Ils ne doivent donc être utilisés en retour que si la méthode a retourné *true*.
La méthode ne modifie ni la position de la *Sphere*, ni ses composantes dynamiques (vecteurs vitesse & moment cinétique). Celles-ci doivent être modifiées en sortie de méthode.

Cette méthode doit s’appuyer sur la méthode du chapitre précédent *GetSphereNewPositionAndVelocityIfCollidingWithRoundedBoxes.*