TME1. Collisions simples et problème du cercle minimum.

BM Bui-Xuan

1 Collisions simples

Nous nous intéressons ici au problème de détecter si deux éléments géométriques s'intersectent proprement (dans un plan en 2D). L'objectif majeur de cette partie du TME est d'identifier les objets les plus facilement manipulables pour la détection de collision.

Exercice 1 – Collision entre cercles et bordures

Soit un rectangle `main_window` défini par les quatre extrémités suivantes.

- point $(0, 0)$ placé "en haut à gauche" du plan ;
- point $(\texttt{width}, 0)$ placé "en haut à droite" ;
- point $(\texttt{width}, \texttt{height})$ placé "en bas à droite" ;
- point $(0, \texttt{height})$ placé "en bas à gauche".

Soit un cercle centré au point `centre`, de rayon `radius`. On suppose que le point `centre` a pour coordonnées `x` et `y`, et qu'il est strictement contenu dans le rectangle `main_window`.

1. Déterminer une condition nécessaire et suffisante pour que le cercle intersecte la bordure "gauche" du rectangle, c.à.d. celle joignant le point $(0, 0)$ au point $(0, \texttt{height})$.
2. Compléter le code des quatre procédures `collisionLeftBorder`, `collisionRightBorder`, `collisionTopBorder`, `collisionBottomBorder` présentes dans le fichier canevas `RBB_collision_canevas.html`.
   N.B. *: pour débugger Javascript, utiliser une "Web console" ! ! !*
   Admirer votre beau travail en chargeant le fichier `html` sur votre navigateur ouèbe préféré.
3. Considérer maintenant deux cercles, initialement disjoints, et déterminer une condition nécessaire et suffisante pour que les deux cercles s'intersectent. Compléter le code de la procédure `collisionCircles` dans le fichier canevas.
   N.B. *: sur le rendu* `html`*, il est normal que la balle rouge (contrôlée par les touches "flèches" du clavier) ne rebondit pas sur la balle noir.*

Exercice 2 – Collision entre un cercle et un rectangle

1. Considérer maintenant un cercle et un rectangle, initialement disjoints, et déterminer les conditions pour que les deux objets s'intersectent. Compléter le code de la procédure `collisionCircleBox` dans le fichier canevas.
   N.B. *: le but du jeu est que la balle bleue (contrôlée par les touches clavier "qdzs") arrive à attraper le trésor contenu dans la boite bleue...*

Avec un peu de chance, nous aurions remarqué à ce stade du TME qu'il serait beaucoup plus simple pour le problème de détection de collisions si l'on pouvait toujours approximer nos objets géométriques par des cercles.

2 Problème du cercle minimum

Le problème du cercle minimum consiste à déterminer, à partir d'un ensemble de points dans le plan, un cercle de rayon minimum contenant tout point de l'ensemble. Naturellement, il a de nombreuses applications dans la géométrie algorithmique, dont la détection de collision. Il est aussi très étudié dans d'autres disciplines : problème du "bureau de poste" pour l'aménagement territorial ; problème du relais téléphonique pour de la télécommunication ; problème de la surveillance d'examen pour (certains) enseignants...

Exercice 3 – Quelques détails techniques...

1. Créer un projet Java `UnSuperNomDeProjet`, récupérer le fichier canevas `TME1_cpa20160120.jar`, et importer celui-ci dans `UnSuperNomDeProjet`.
   P.S. *pour supprimer les messages d'avertissement sous Eclipse : "cliquer droit" sur le projet pour choisir* `Build path`*, puis dans l'onglet "External jars" ajouter le fichier* `jars/TME1_supportGUI.jar`
2. Exécuter la classe `DiamRacer` du projet.
   N.B. *: sous Eclipse il est possible de "cliquer droit" sur* `DefaultTeam` *puis choisir* `RunAs` → `RunConfigurations`

Exercice 4 – Algorithme naïf

Il est toujours primordial de trouver une solution simple à tout problème algorithmique. Cette solution, souvent très lente mais presque toujours facile à coder, est communément appelée "algorithme naïf". Elle est de loin l'outil le plus pratique pour tester les futures codes optimisés.

1. Trouver un algorithme naïf résolvant le problème du cercle minimum.
2. Compléter le code de la procédure `calculCercleMin` dans le fichier canevas.
3. Tester votre code "visuellement" avec `DiamRacer`.
   N.B. *: il est bon, ici, d'admirer la puissance de calcul de la nature : nos yeux ont, de loin, des bibliothèques de calcul bien plus optimisées que n'importe quel "driver graphique" de dernier cri.*

Exercice 5 – Algorithme Ritter

Tout problème algorithmique possède (au moins) un *trade-off* : temps de calcul vs. qualité du résultat. L'algorithme de Ritter est un algorithme d'*approximation* du cercle minimum : on dégrade la qualité du résultat pour un temps de calcul plus rapide. Ceci dit, l'algorithme Ritter a à la fois le mérite de calculer très vite, tout en ne pas trop dégradant la qualité du résultat retourné. De plus, son principe est très simple :

— 1. Prendre un point `dummy` quelconque appartenant à l'ensemble de points de départ.
— 2. Parcourir l'ensemble de points pour trouver un point `P` de distance maximum au point `dummy`.
— 3. Re-parcourir l'ensemble de points pour trouver un point `Q` de distance maximum au point `P`.
— 4. Considérer le point `C`, le centre du segment `PQ`.
— 5. Considérer le cercle `CERCLE` centré en `C`, de rayon $|\texttt{CP}|$: il passe par `P` et `Q`.
— 6. Retirer les points `P` et `Q` de l'ensemble des points.
— 7. Tant qu'il reste des points dans l'ensemble, considérer un point `S` quelconque.
— 8. Si `S` est couvert par `CERCLE`, retirer `S` de l'ensemble des points ; répéter l'étape 7.
— 9. Sinon, tracer au brouillon la droite passant par `S` et `C`. Celle-ci coupe le périmètre du cercle courant en deux points : soit `T` le point le plus éloigné de `S`.
— 10. Déterminer les coordonnées du point $\texttt{C}'$, le centre du segment `ST`.
— 11. Remplacer `CERCLE` par le cercle centré en $\texttt{C}'$, de rayon $|\texttt{C}'\texttt{T}|$: il passe par `S` et `T`.
— 12. Répéter l'étape 7. jusqu'à ce qu'il ne reste plus de points dans la liste.

1. Coder l'algorithme Ritter dans la procédure `calculCercleMin` dans le fichier canevas. Tester "visuellement" que le résultat renvoyé couvre bien tous les points de l'ensemble de départ.
2. Montrer formellement que l'algorithme Ritter retourne bien un cercle couvrant tous les points de l'ensemble de départ.
3. Estimer la complexité en temps de l'algorithme Ritter.
4. (A faire à la maison parce qu'il serait probable que le projet principal de cette année demande quelque chose de similaire...) Faire tourner l'algorithme Ritter sur un nombre conséquent d'instances tirées au hasard, 1664 par exemple, et calculer le taux d'approximation par rapport à la solution optimale, que l'on peut calculer avec l'algorithme naïf par exemple.

3 Problème du diamètre d'un ensemble de points

Les trois premières étapes de l'algorithme Ritter servent à trouver un premier cercle-candidat pour devenir la solution finale. Les autres étapes réalisent ce que l'on appelle la technique du "*glouton*". Il s'agit d'améliorer le cercle-candidat courant de manière incrémentale, et par des opérations d'optimisation locale, jusqu'à ce qu'il devient une solution. Cette technique est l'une des plus largement utilisées pour réduire la complexité en temps des algorithmes.

Les trois premières étapes de l'algorithme Ritter consistent, en réalité, à approximer ce que l'on appelle le diamètre de l'ensemble de points de départ. Le diamètre d'un ensemble de points est défini comme la distance maximum entre deux points appartenant à l'ensemble. Il serait intéressant de remplacer ces trois étapes par le calcul du diamètre exacte de l'ensemble de points de départ. Par ailleurs, le calcul du diamètre est un calcul de base en géométrie algorithmique. Beaucoup d'algorithmes assument ce calcul comme primitive.

Exercice 6 – Algorithme naïf

1. Trouver un algorithme naïf résolvant le problème du diamètre d'un ensemble de points.
2. Compléter le code de la procédure `calculDiametre` dans le fichier canevas.
3. Remplacer les trois premières étapes de l'algorithme Ritter par le calcul du diamètre.
   ***N.B.** : Constater une nette dégradation du temps de calcul.*

Exercice 7 – Introduction au précalcul

1. Proposer des solutions pour améliorer le calcul du diamètre. Pour ce calcul de diamètre en particulier, on s'intéressera uniquement à des solutions exactes, pas des approximations.
   ***N.B.** : On peut penser à la même technique que Ritter : peut on enlever des points inutiles ?*

Exercice 8 – Un premier précalcul : Filtrage Akl-Toussaint

Soient A, B, C, D quatre points de $\mathcal{S}$ tels que : A est d'abscisse minimum ; B est d'ordonnée minimum ; C est d'abscisse maximum ; D est d'ordonnée maximum. Le filtrage Akl-Toussaint consiste à enlever tout point appartenant au quadrilatère défini par A ,B, C, et D. Ce filtrage demande l'implantation d'un test d'appartenance d'un point à un triangle.

1. (Une bien mauvaise tentative) Un point X appartient au triangle ABC si et seulement si la somme des aires des trois triangles ABX, BCX, et ACX est égale à l'aire de ABC.
   On rappelle la formule de Héron pour calculer l'aire d'un triangle : $\text{Aire} = \sqrt{s(s-a)(s-b)(s-c)}$ avec a, b, c les longueurs des côté du triangle et $s = \frac{a+b+c}{2}$ le demi périmètre du triangle.
   Implanter le filtrage Akl-Toussaint utilisant la formule de Héron. Pourquoi cette méthode ne "marche" pas dans le plupart des cas ?
2. (Produit vectoriel) Soient $\overrightarrow{u}$ et $\overrightarrow{v}$ deux vecteurs de coordonnées $(u.x, u.y, u.z = 0)$ et $(v.x, v.y, v.z = 0)$. Le produit vectoriel $\overrightarrow{u} \wedge \overrightarrow{v}$ est défini par $(0, 0, u.x * v.y - u.y * v.x)$. Un point X appartient au triangle ABC si et seulement si :

— Le signe de la troisième coordonnée de $\overrightarrow{AB} \wedge \overrightarrow{AX}$ est le même que celui de $\overrightarrow{AB} \wedge \overrightarrow{AC}$, et

— Le signe de la troisième coordonnée de $\overrightarrow{BC} \wedge \overrightarrow{BX}$ est le même que celui de $\overrightarrow{BC} \wedge \overrightarrow{BA}$, et

— Le signe de la troisième coordonnée de $\overrightarrow{AC} \wedge \overrightarrow{AX}$ est le même que celui de $\overrightarrow{AC} \wedge \overrightarrow{AB}$.

On rappelle que les coordonnées du vecteur $\overrightarrow{AB}$ est données par $(B.x - A.x, B.y - A.y)$. Implanter le filtrage Akl-Toussaint utilisant le produit vectoriel.

3. (Coordonnées barycentriques) En nous épargnant d'un peu (beaucoup) de discours, un point X est à l'intérieur d'un triangle ABC si et seulement si les coordonnées barycentriques de X par rapport à ABC sont toutes comprises entre 0 et 1, voir
   `http://en.wikipedia.org/wiki/Barycentric_coordinate_system_(mathematics)`
   Ces coordonnées sont définies par :
   $l1 = ((B.y - C.y) * (X.x - C.x) + (C.x - B.x) * (X.y - C.y))/((B.y - C.y) * (A.x - C.x) + (C.x - B.x) * (A.y - C.y))$
   $l2 = ((C.y - A.y) * (X.x - C.x) + (A.x - C.x) * (X.y - C.y))/((B.y - C.y) * (A.x - C.x) + (C.x - B.x) * (A.y - C.y))$
   $l3 = 1 - l1 - l2$
   Implanter le filtrage Akl-Toussaint utilisant les coordonnées barycentriques.

Exercice 9 – Un excellent précalcul pour le diamètre : algorithme QuickHull

Le principe est le suivant :

— Considérer les quatre points A, B, C, D définis dans le filtrage Akl-Toussaint.

— Pour chaque côté du quadrilatère, disons AB, trouver le point X le plus éloigné de AB "du côté extérieur" du quadrilatère : on peut utiliser le produit vectoriel.

— Enlever tout point appartenant au triangle ABX.

— Ajouter AX et BX aux côtés à traiter, remplacer le terme quadrilatère par pentagone et répéter...

1. Implanter l'algorithme QuickHull et analyser sa complexité.