TP - La structure «polygone»

Licence 1 - Maths Info

Algorithmique et programmation

---

1. Reprendre la structure `point` ainsi que les procédures de lecture et d'affichage (textuel et graphique) du précédent TP.
2. Déclarer le type tableau de points `typeTabPoint` pouvant contenir $TMAX = 50$ points.
3. Déclarer le type `polygone` contenant un tableau de points et un entier correspondant au nombre de points effectivement utilisés.
4. Écrire une procédure de saisie d'un polygone. On s'assurera que 2 points consécutifs sont distincts.
5. Écrire une procédure d'affichage graphique d'un polygone.
6. Écrire une procédure de test `void testPolygone()` qui demande à l'utilisateur de saisir un polygone et qui affiche le résultat dans la fenêtre graphique.
7. Écrire une fonction `airePoly` qui calcule l'aire exacte d'un polygone (qui peut être convexe ou non). On utilisera pour ceci la formule suivante :

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (x_{i+1} + x_i)(y_{i+1} - y_i)$$

   où les (x_i, y_i) sont les sommets du polygone et $x_n = x_0$ et $y_n = y_0$.
8. Écrire une fonction `aGauche` qui renvoie `vrai` si le point `p` est à gauche de la droite (p_1, p_2). On utilisera la propriété suivante : `p` est à gauche de la droite (p_1, p_2) si et seulement si $det(\overrightarrow{p_1p_2}, \overrightarrow{p_1p}) > 0$.
9. On s'intéresse maintenant à déterminer si un point se trouve à l'intérieur d'un polygone quelconque.
   (a) Pour rappel, on appelle demi-droite horizontale une droite partant d'un point `p` et se prolongeant uniquement dans une seule direction (droite ou gauche). Écrire une fonction renvoyant `vrai` lorsque la demi-droite issue de `p` (vers la droite) coupe le segment $[p_1, p_2]$.
   (b) Écrire une fonction `estDansPolygone` qui renvoie `vrai` lorsque le point `p` se trouve à l'intérieur du polygone `poly`.
10. Estimation de l'aire d'un polygone quelconque à l'aide d'un algorithme de Monte-Carlo. Il s'agit d'une approche basée sur les nombres aléatoires pour estimer un résultat. L'objectif ici est de générer un certain nombre de points aléatoires, puis de déterminer la proportion de points se trouvant dans le polygone. De cette proportion nous pourrons estimer l'aire du polygone.

Remarque : On utilisera la fonction suivante, qui renvoie une valeur aléatoire comprise entre `a` et `b` :

```
double aleat(double a, double b){
 return a+(b-a)*rand()/static_cast<double>(RAND_MAX);
}
```

(a) Écrire une fonction `aireApprochee` qui prend en paramètre un polygone, un point en bas à gauche et un point en haut à droite qui définissent une «fenêtre», ainsi qu'un nombre de points à générer aléatoirement. Cette fonction, en s'appuyant sur la fonction `estDansPolygone`, déterminera la proportion des points se trouvant dans le polygone. Elle renverra l'aire approchée du polygone.

(b) Écrire une procédure qui détermine la boite englobante d'un polygone.

(c) Écrire une variante d'`aireApprochee` qui calculera automatiquement la fenêtre adaptée au polygone.

(d) Écrire une fonction qui calcule et retourne le nombre de points qu'il faut générer aléatoirement pour s'approcher à ϵ près de l'aire exacte du polygone. Le polygone et la valeur ϵ seront donnés en paramètre.

Pour aller plus loin :

Bonus : Approximation de la valeur de π en calculant le périmètre d'un polygone régulier. Le principe est le suivant :

1. Créer un polygone carré.
2. Écrire une procédure `polygoneSuivant(polygone& pol,double r)` qui, à partir d'un polygone, coupe chaque segment en 2 (double le nombre de points du polygone) et déplace les nouveaux points créés de façon à obtenir un polygone régulier. Tous les points sont donc à une distance `r` du centre du polygone.
3. Utiliser une fonction `perimetre(const polygone& pol)` pour déterminer π. Notons que `polygoneSuivant` devra être appelé autant de fois que nécessaire pour affiner la précision de π.