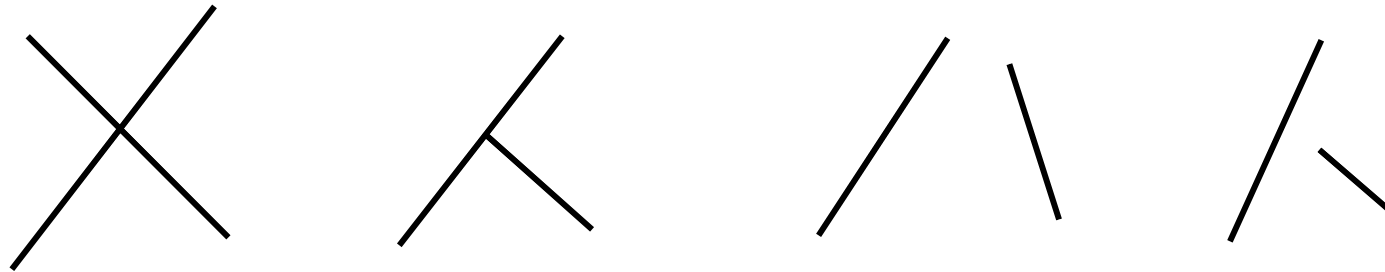
Algorithmes géométriques, Géométrie algorithmique

Frédéric Devernay

Types de base

- point : (x,y)
- segment : (point,point)
- polygone : array of point

Premier problème : intersection de deux segments



- Solution 1: calculer l'intersection des deux droites support, vérifier qu'elle est à l'intérieur des deux segments

Intersection de deux segments

```
int ccw(struct point p0,
    struct point p1,
    struct point p2)
{
    float dx1, dx2, dy1, dy2 ;
    dx1 = p1.x - p0.x ; dy1 = p1.y - p0.y ;
    dx2 = p2.x - p0.x ; dy2 = p2.y - p0.y ;
    if (dx1*dy2 > dy1*dx2) return +1 ; // pente du segment 1= dy1/dx1
    if (dx1*dy2 < dy1*dx2) return -1 ;
    // trois points colinéaires
    if ((dx1*dx2 < 0) || (dy1*dy2 <0)) return -1 ; // p0 entre p1 et p2
    if ((dx1*dx2+dy1*dy1) < (dx2*dx2+dy2*dy2)) return +1 ; // p0-p1-p2
    return 0 ;
}
```

- Fonction ccw (counter-clockwise) retourne 1 si les points sont dans le sens inverse des aiguilles d’une montre, -1 dans le sens inverse. Cas “limites” bien gérés

Intersection de deux segments

```
int intersect(struct line l1, struct line l2)
{
  return ((ccw(l1.p1, l1.p2, l2.p1)
          * ccw(l1.p1, l1.p2, l2.p2)) <= 0)
    && ((ccw(l2.p1, l2.p2, l1.p1)
          * ccw(l2.p1, l2.p2, l1.p2)) <= 0) ;
}
```

- fonction intersect : si les extrémités de chaque segment sont des deux côtés de l’autre segment, alors ils s’intersectent
- solution paraît compliquée, mais c’est la seule qui marche dans tous les cas (4 pts collinéaires par exemple)

Point à l'intérieur d'un polygone

- prendre un segment "test" partant du point allant « loin » dans la direction x
- compter le nombre d'intersections avec le polygone : pair = point à l'extérieur, impair = point à l'intérieur
- mais si le segment "test" passe exactement par des sommets, que faire ?
- solution : parcourir le polygone, et incrémenter un compteur à chaque fois qu'on passe d'un côté à l'autre du polygone. Ne rien faire si on tombe dessus

Point à l'intérieur d'un polygone

```
int inside(struct point t, struct point p[], int N)
{
   int i, count = 0, j = 0 ;
   struct line lt,lp ;
   p[0] = p[N] ; p[N+1] = p[1] ;
   lt.p1 = t ; lt.p2 = t ; lt.p2.x = FLT_BIG ;
   for (i=1 ; i<=N ; i++)
   {
      lp.p1 = p[i] ; lp.p2 = p[i];
      if ( !intersect(lp,lt))
      {
         lp.p2 = p[j] ; j = i ;
         if (intersect(lp,lt)) count++ ;
      }
   }
   return count & 1 ;
}
```

Enveloppe convexe : marche de Jarvis *(package wrapping)*

- partir d'un point dont on est sûr qu'il est sur l'enveloppe convexe : point d'ordonnée minimum
- trier les sergments reliant ce point à tous les autres par leur angle avec l'axe des abscisses. Celui qui a le plus petit angle donne le point suivant.
- l'algo termine quand on retombe sur le premier point
- Complexité : O(n.n) au pire, O(n.h) en réalité

Enveloppe convexe : Graham scan

- prendre le point de départ.
- trier tous les autres points par angle croissant avec ce point
- il en résulte un polygone non convexe
- parcourir ce polygone, et tant qu'on tourne à gauche, ajouter le point.
- Si on tourne à droite, éliminer le point précédent, jusqu'à ce qu'on tourne à gauche
- Complexité : tri O(nlogn) + au pire deux parcours de l'ensemble (pour ajouter les points + pour les éliminer) O(2n)

Enveloppe convexe : QuickHull

```
Quickhull( PointSet S, Point L, Point R )
1. if ( S = {L, R} )  // if the only points in the set are L
and R
2. return {L, R}
3. else
4. H = farthest( S, L, R )
5. S1 = points of S on or left of LH
6. S2 = points of S on or right of HR
7. return ( Quickhull(S1, L, H) ∪ Quickhull( S2, H, R ) )
where farthest( PointSet S, Point L, Point R) is a function
that determines which point yields the triangle of largest
area.
```

- on part de deux points extremes L et R, on sépare l'ensemble en deux selon LR, on lance Quickhull() sur chaque moitié
- Complexité moyenne O(nlogn), pire O(n2)

Enveloppe convexe : Diviser pour régner (Preparata et Hong)

- séparer en deux par une droite
- calculer l'enveloppe convexe de chaque moitié
- « fusionner » les deux enveloppes convexes en trouvant les deux segments à ajouter
- Complexité O(nlogn), mais implantation complexe

Conclusion

- Lesquels de ces algos “passent” en 3D ?
- Autres algos “classiques” :
- Diagrammes de Voronoï
- Triangulation de Delaunay
- Et des dizaines d’autres problèmes aux applications multiples (robotique, optimisation, synthèse d’images, jeux vidéos, simulateurs...)