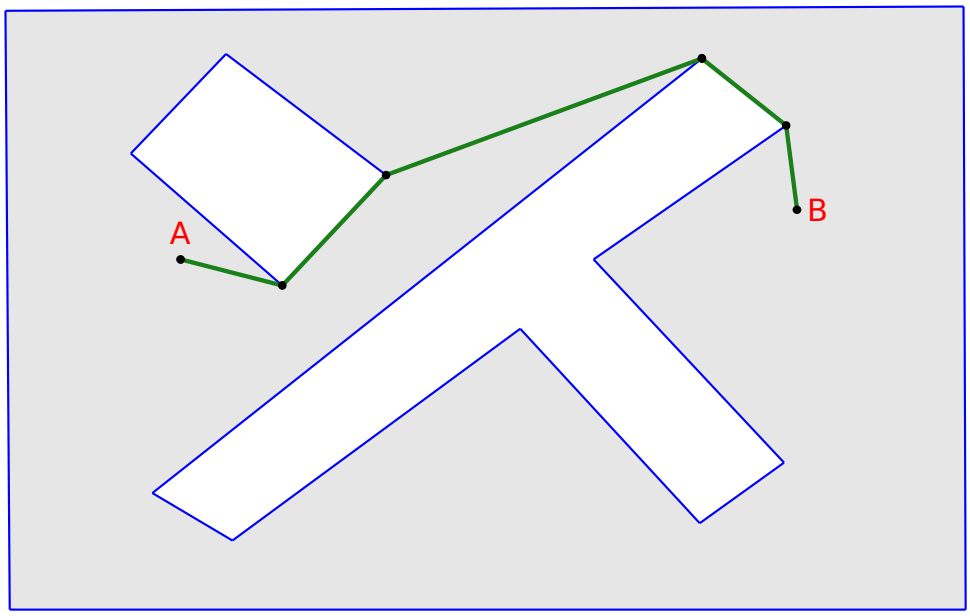
# Découpage en polygones convexes pour la recherche de chemin

But :

Concevoir un algorithme capable de tracer un chemin d'un point A à un point B.

Exemple :




**I. Présentation et démarche**

1. Position du problème
2. Intérêt de la convexité
3. Des algorithmes simples mais non adaptés
4. Les algorithmes retenus

**II. Détail de la triangulation par décomposition en polygones monotones**

1. Décomposition en polygones monotones
2. Décomposition d'un polygone monotone en « montagnes »
3. Triangulation d'une montagne

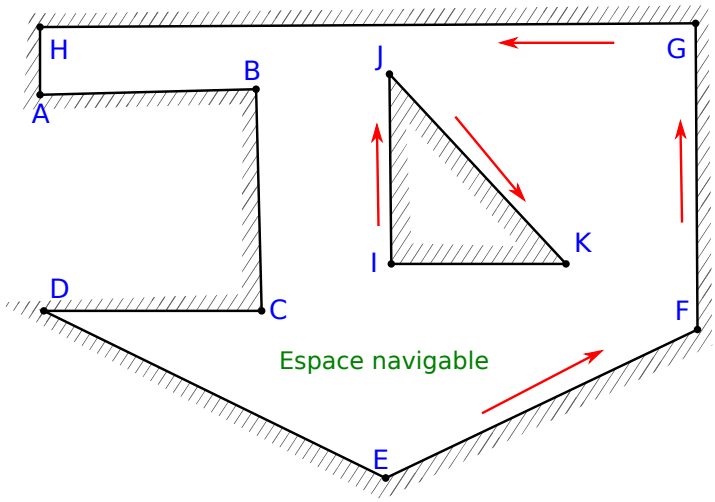
**III. Réalisation et résultats**

1. Le programme
2. Application : aller au lycée
3. Étude d'un invariant

## *I. Présentation et démarche*

### I.1. Position du problème

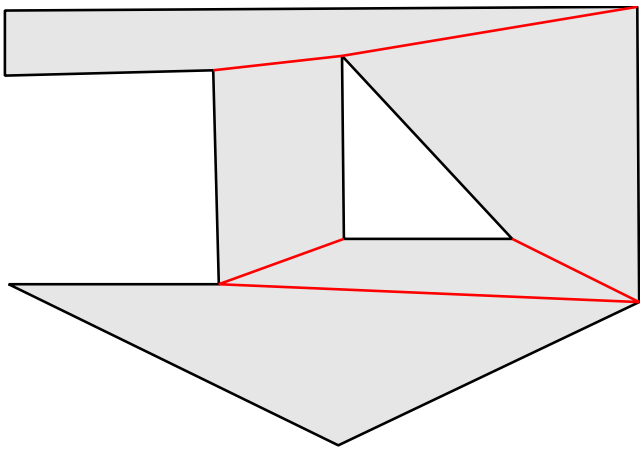
On se donne un polygone « à trous » :



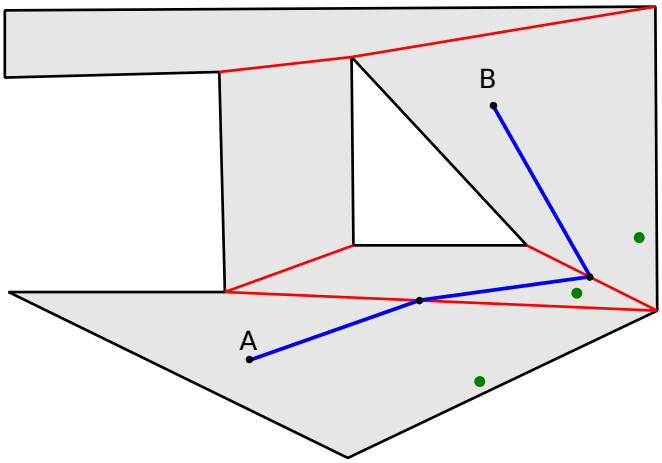

On cherche une méthode pour construire des chemins, sous forme de lignes brisées, reliant deux points de l'espace navigable.

### I.2. Intérêt de la convexité

Si l'on dispose d'un découpage de la zone navigable en parties convexes :

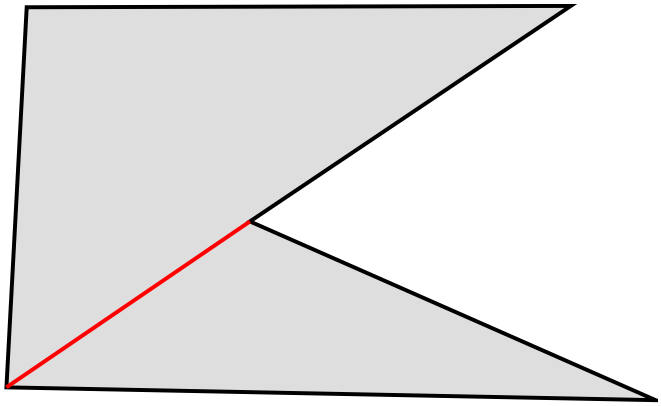


On peut construire un chemin très facilement :



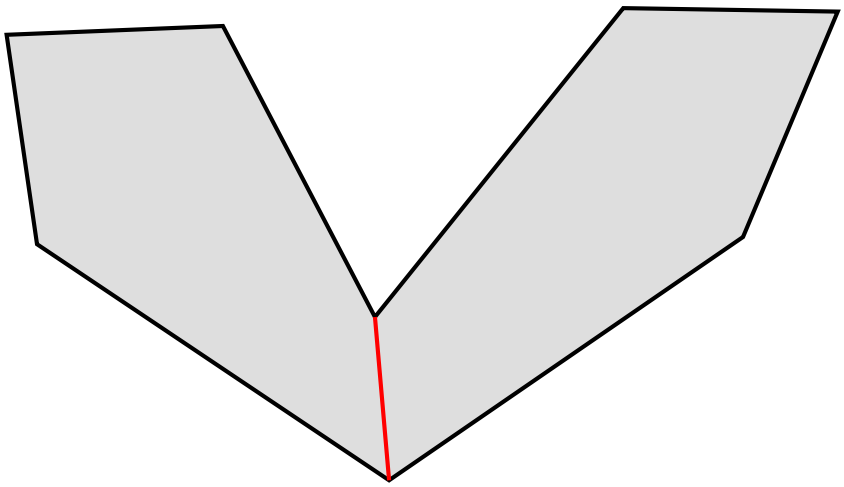
## I.3. Des algorithmes simples mais non adaptés

### *1.3.a. Enlever des oreilles*



Algorithme implémenté en C++. Complexité en $O(n^3)$.

### *1.3.b. Tracer des diagonales*



Algorithme implémenté en C++. Complexité en $O(n^2)$.

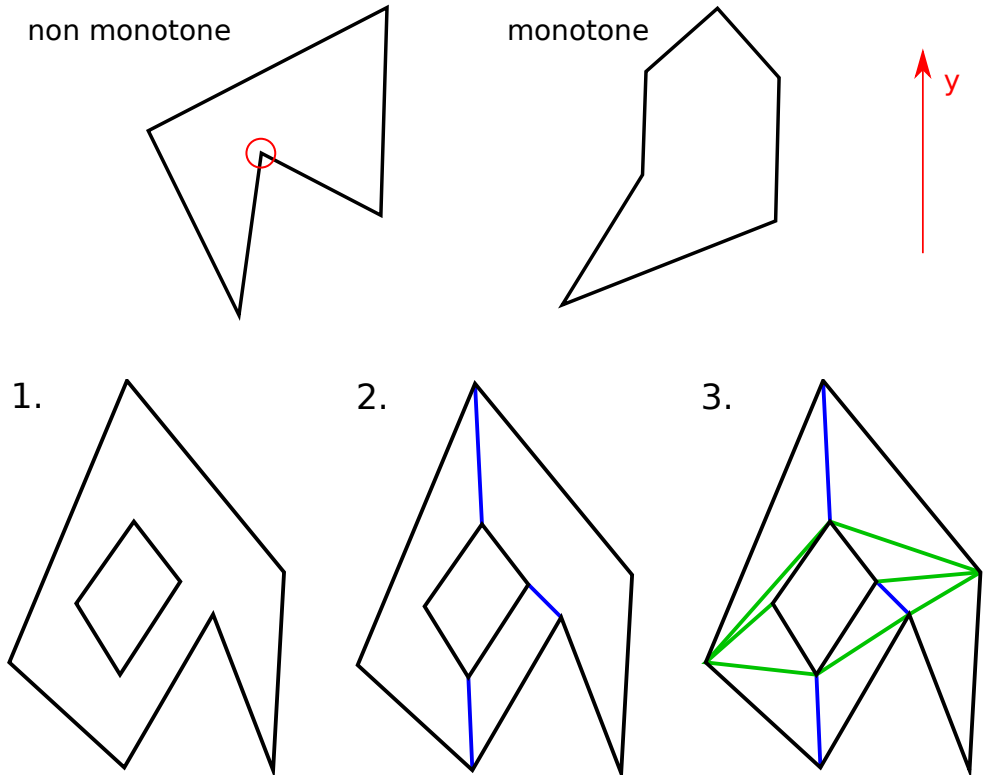
### *1.3.c. Les problèmes*

- Complexité trop mauvaise
- Non adapté au problème des polygones à trous

## I.4. Les algorithmes retenus

### *I.4.a. Décomposition en polygones monotones*

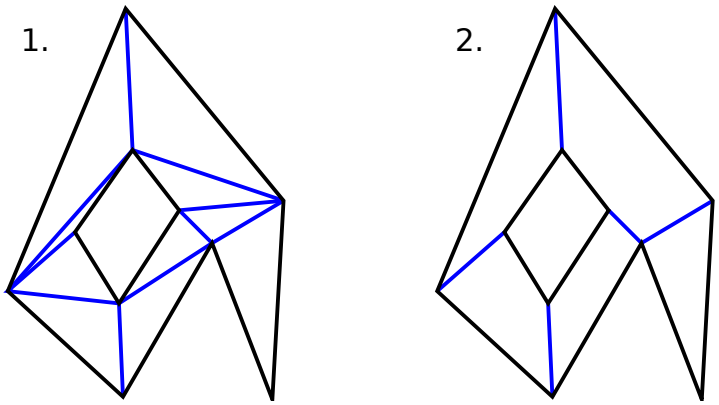
Après concertation avec Mme Mariette Yvinec (INRIA), je me suis orienté vers un algorithme à base de décomposition en polygones monotones.




### *I.4.b. Simplification convexe*

La triangulation est déjà un découpage convexe satisfaisant.

Mais afin d'améliorer le temps d'exécution et la qualité des résultats de l'algorithme de recherche de chemin, on peut effectuer d'abord une simplification du découpage.

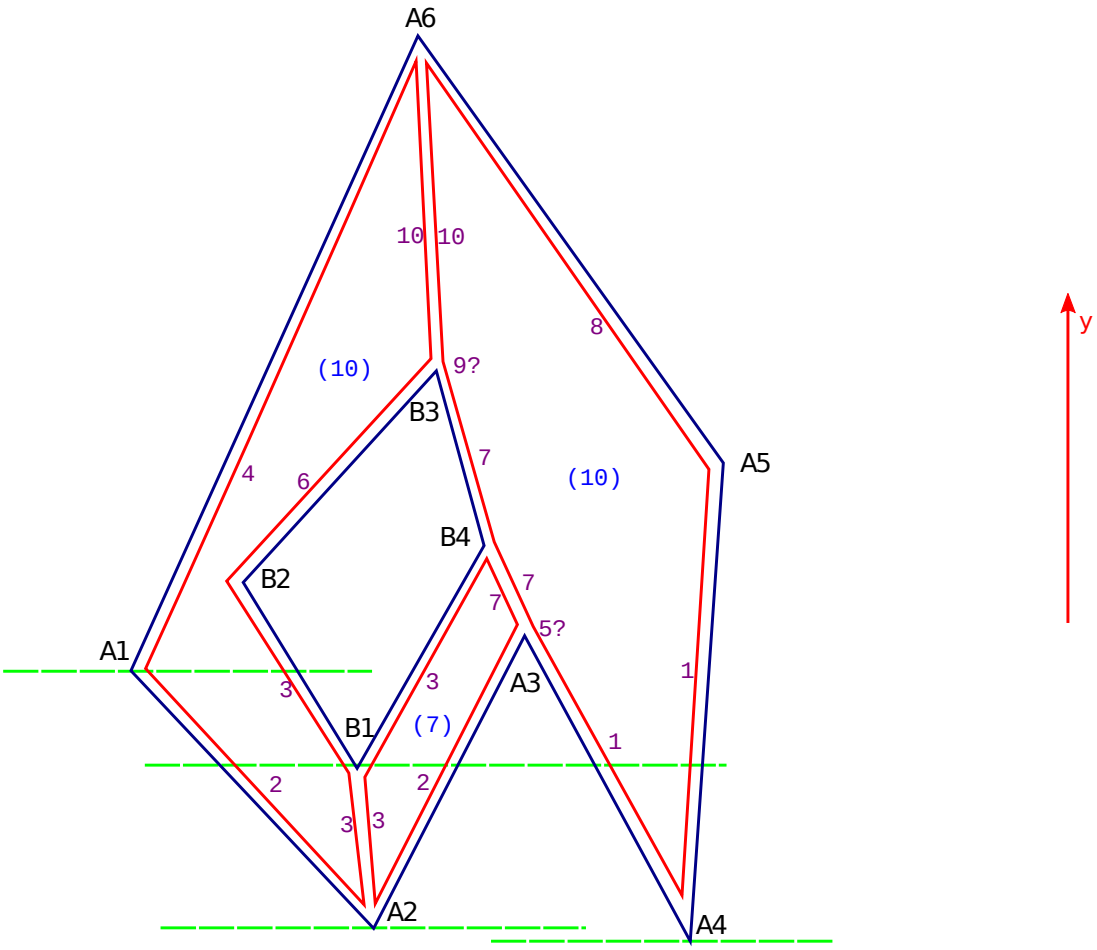



### *I.4.c. Recherche de chemin*

Pour la recherche de chemin, on utilise une adaptation de l'algorithme de parcours de graphe de Dijkstra.

## *II. Détail de la triangulation par décomposition en polygones monotones*

### II.1. La décomposition en polygones monotones

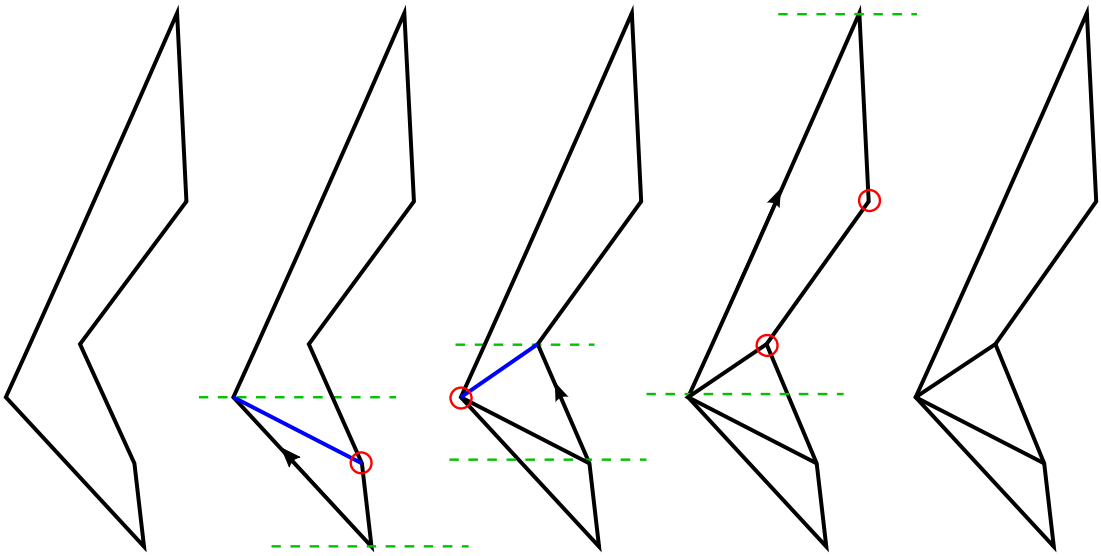



On construit les chaînes monotones en effectuant un balayage des points :

1. A4 (CUSP_A_IN) A3A4 A5A4
2. A2 (CUSP_A_IN) A1A2 A3A2 A3A4 A5A4
3. B1 (CUSP_A_OUT) A1A2 <B2B1A2 B4B1A2 A3A2 A3A4 A5A4
4. A1 (NORMAL_N) >A6A1A2 B2B1A2 B4B1A2 A3A2 A3A4 A5A4
5. A3 (CUSP_B_OUT) A6A1A2 B2B1A2 B4B1A2 *<A3A2 *>A3A4 A5A4
6. B2 (NORMAL_P) A6A1A2 <B3B2B1A2 B4B1A2 *A3A2 *A3A4 A5A4
7. B4 (NORMAL_N) A6A1A2 B3B2B1A2 >B3B4B1A2 *A3A2 *A3A4 A5A4
   >out(B4B1A2+B4A3A2) B4B1A2A3
   A6A1A2 B3B2B1A2 >B3B4A3A4 A5A4
8. A5 (NORMAL_P) A6A1A2 B3B2B1A2 B3B4A3A4 <A6A5A4
9. B3 (CUSP_B_OUT) A6A1A2 *<B3B2B1A2 *>B3B4A3A4 A6A5A4
10. A6 (CUSP_B_IN) >A6A1A2 *B3B2B1A2 *B3B4A3A4 <A6A5A4
    >out(A6A1A2+A6B3B2B1A2) A6A1A2B1B2B3
    >A6B3B4A3A4 <A6A5A4
    #out(A6B3B4A3A4+A6A5A4) A6B3B4A3A4A5*

## II.2. Décomposition d'un polygone monotone en « montagnes »

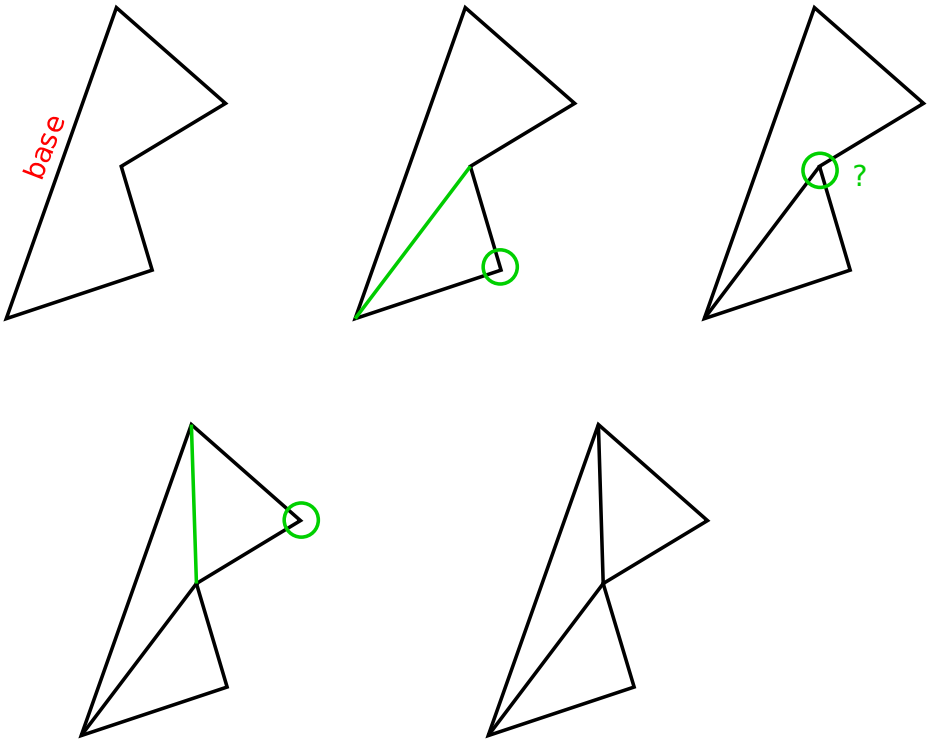
Une montagne est constituée d'une base (à droite ou à gauche), définie par les deux sommets extrémaux sur l'axe vertical, et d'un ensemble de sommets situés de l'autre côté formant une « crête ».



Algorithme implémenté en C++. Complexité linéaire.

## II.3. Triangulation d'une montagne

Les sommets de la montagne, s'ils sont localement convexes, sont des oreilles que l'on peut enlever. On peut les parcourir en un temps linéaire.




Algorithme implémenté en C++. Complexité linéaire.

## *III. Réalisation et résultats*

### III.1. Le programme

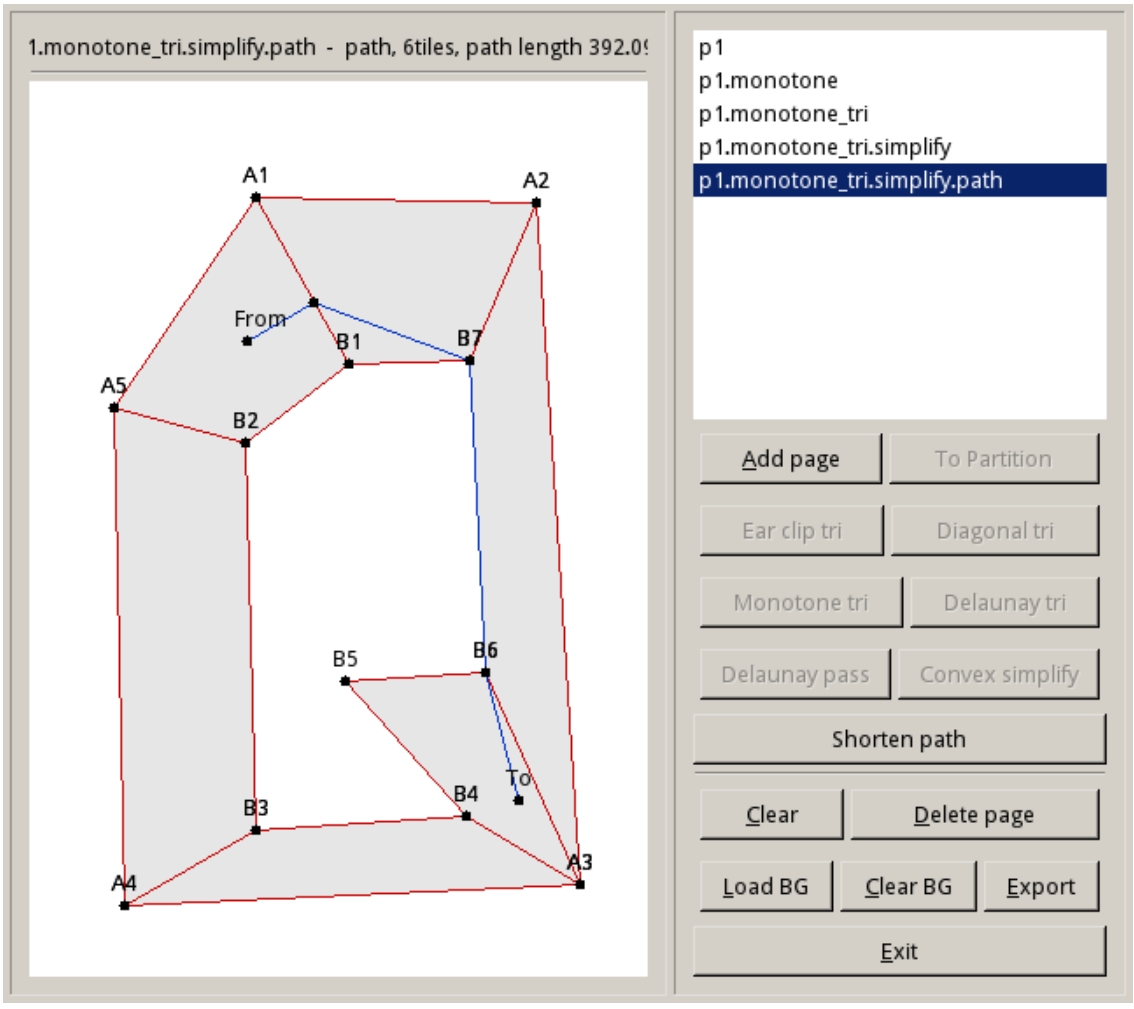
Statistiques :

- Programme écrit en C++
- Totalisant environ 3100 lignes de code
- Dispose d'une interface graphique

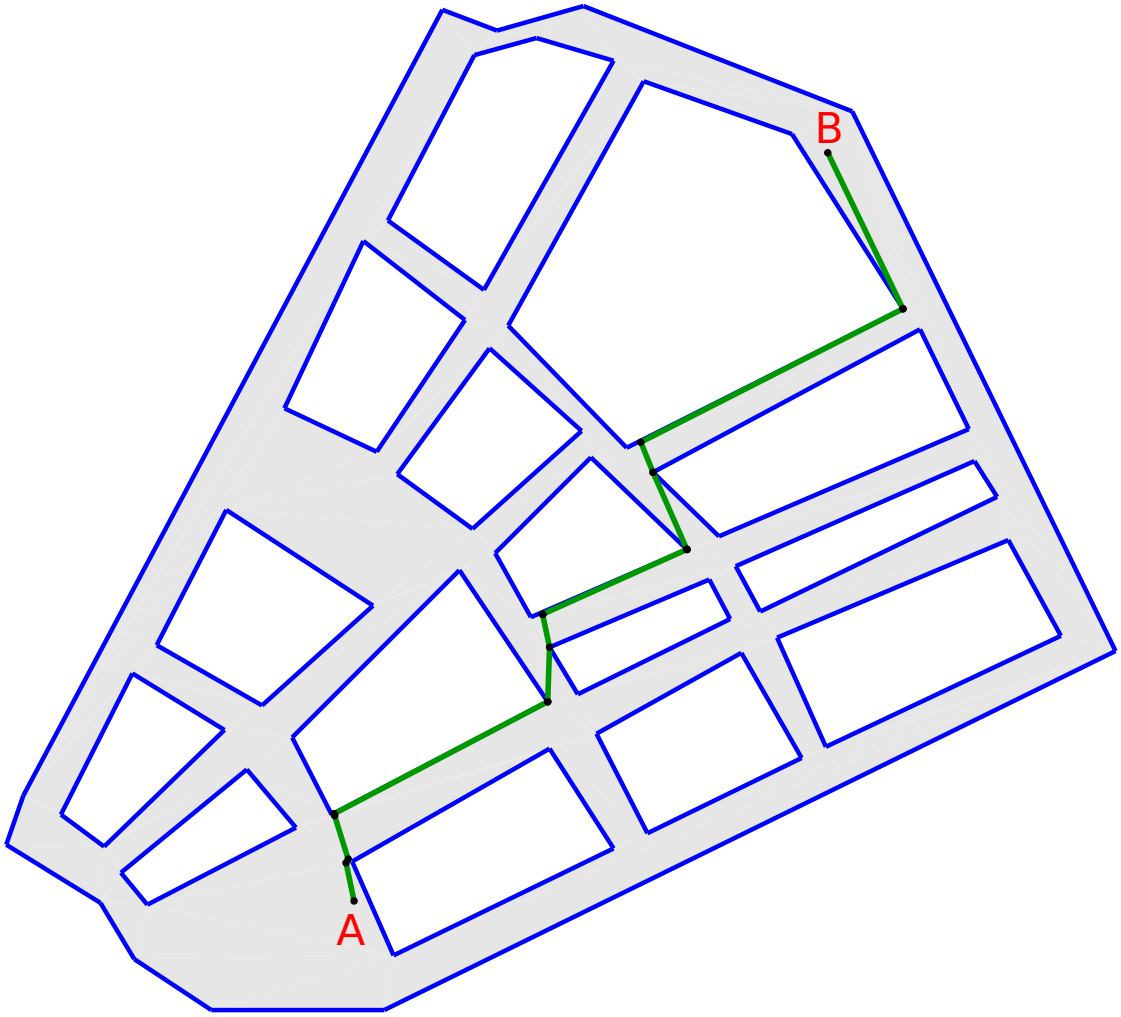
Les algorithmes que j'ai écrit :

- Décomposition de polygones simples (oreilles, diagonales)
- Décomposition monotone, triangulation de polygones monotones
- Simplification convexe
- Recherche de chemin (adaptation de Dijkstra)
- Algorithme itératif d'optimisation du chemin

Capture d'écran :



### III.2. Application : aller au lycée




Tracé des polygones effectué d'après une vue aérienne prise sur Google Maps.

Dans cette exemple, mon programme traite :

- 16 polygones (15 trous), 73 points
- La décomposition en polygones monotones nécessite 15 polygones
- 101 triangles
- La simplification donne entre 43 et 48 tuiles convexes

Conclusion : mon programme fonctionne sans problème sur des données de grosse taille !

## III.3. Étude d'un invariant

Tableau de résultats sur de nombreux exemples :

| Ex | 1 | 2 | A | 9 | 6 | 7 | 4 | 5 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **NbPoly** | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 9 | 16 |
| **NbPts** | 12 | 8 | 9 | 9 | 10 | 12 | 20 | 36 | 73 |
| **PtRéflexe** | 4 | 4 | 4 | 4 | 5 | 6 | 10 | 32 | 64 |
| **Trous** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 8 | 15 |
| **PolysMono** | 3 | 2 | 2 | 3 | 3 | 2 | 4 | 6 | 15 |
| **Triangles** | 10 | 8 | 9 | 9 | 10 | 12 | 20 | 50 | 101 |
| **Tuiles** | 5 – 6 | 4 | 4 | 5 | 5 | 6 | 10 – 12 | 24 – 26 | 43 – 48 |

On remarque l'invariant suivant :

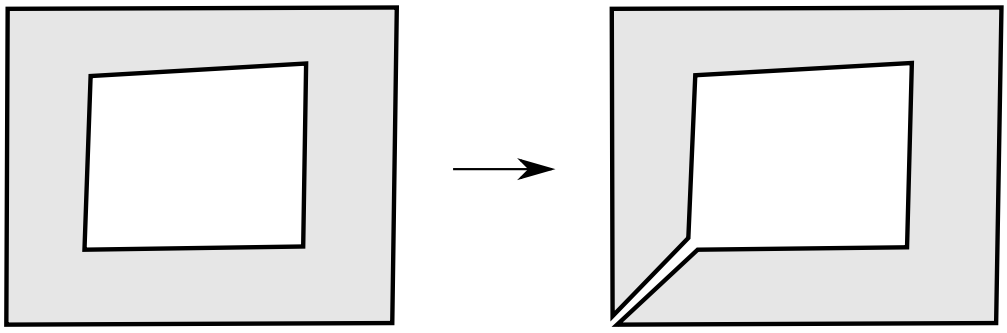
$$nbTriangles = nbPoints + 2 * nbTrous - 2$$

**Propriété.** Pour un polygone simple on a l'invariant :

$$nbTriangles = nbPoints - 2$$

(démonstration par récurrence forte sur le nombre de points, comme une triangulation)

L'invariant pour les polygones à trous se déduit en effectuant la transformation suivante pour supprimer les trous et se ramener au cas d'un polygone simple :



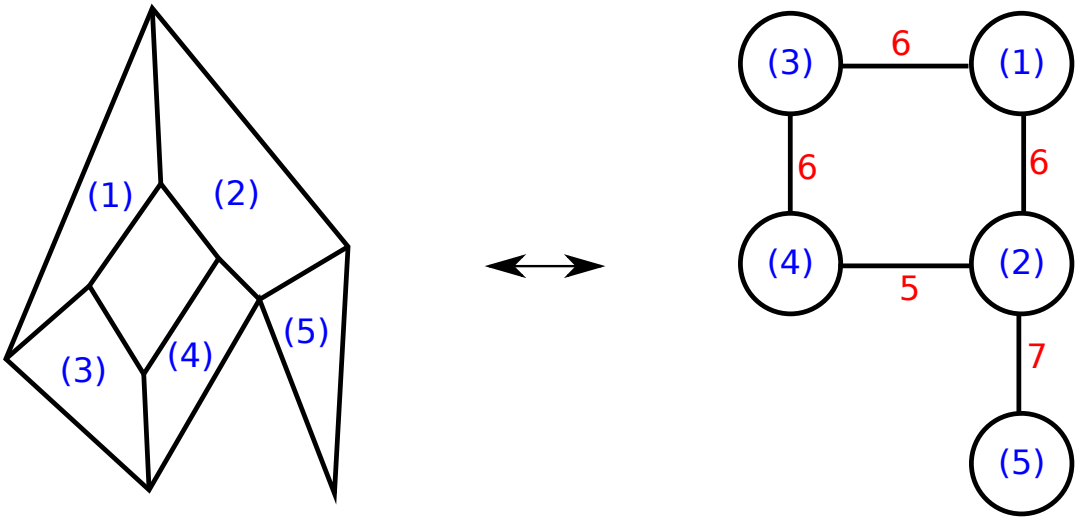
2 polygones
1 trou
8 points

1 polygone simple
(0 trous)
10 points

nombre de triangles : 8 dans les deux cas

*Recherche de chemin : l'algorithme de Dijkstra*

Graphe associé à un découpage convexe :



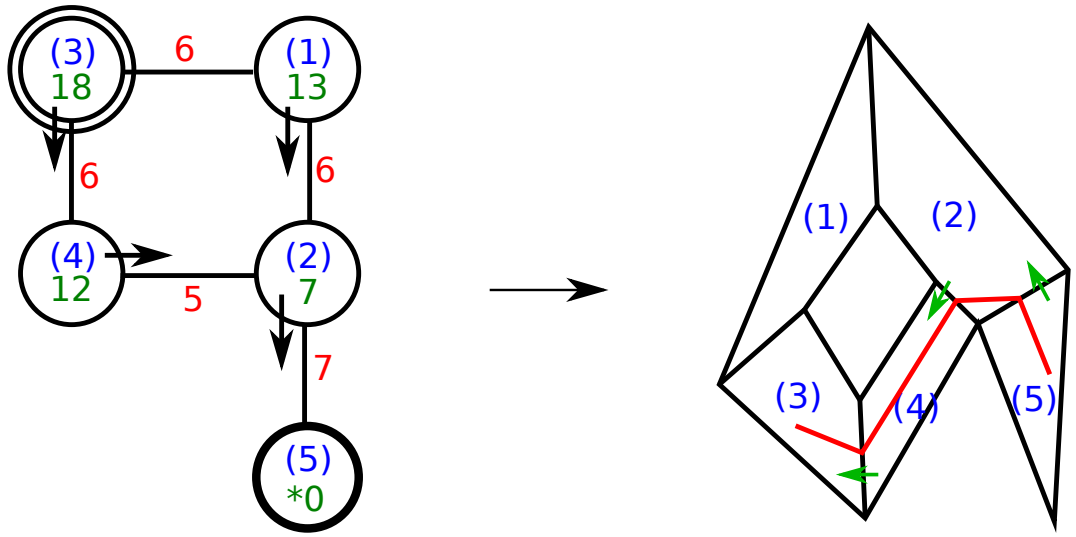

Il s'agit de déterminer par quelles tuiles passer pour relier deux points.

Pour cela on parcours le graphe en associant à chaque nœud accessible un poids (la distance à parcourir pour y parvenir), ainsi que le nœud « parent » qui a permis d'y accéder (et qui a donc un poids strictement inférieur).

Une fois que le nœud d'arrivée est marqué accessible, on remonte les parents successifs jusqu'à arriver au nœud de départ, ce qui donne la liste des tuiles voulue.

Il suffit ensuite de tracer le chemin correspondant sur le dessin.

Exemple ici du nœud 5 au  nœud 3 :




(Sur l'exemple : poids arbitraire pour les arrêtes du graphe.)