Triangulation de Delaunay (Méthode itérative)

 —

Méthode itérative ?

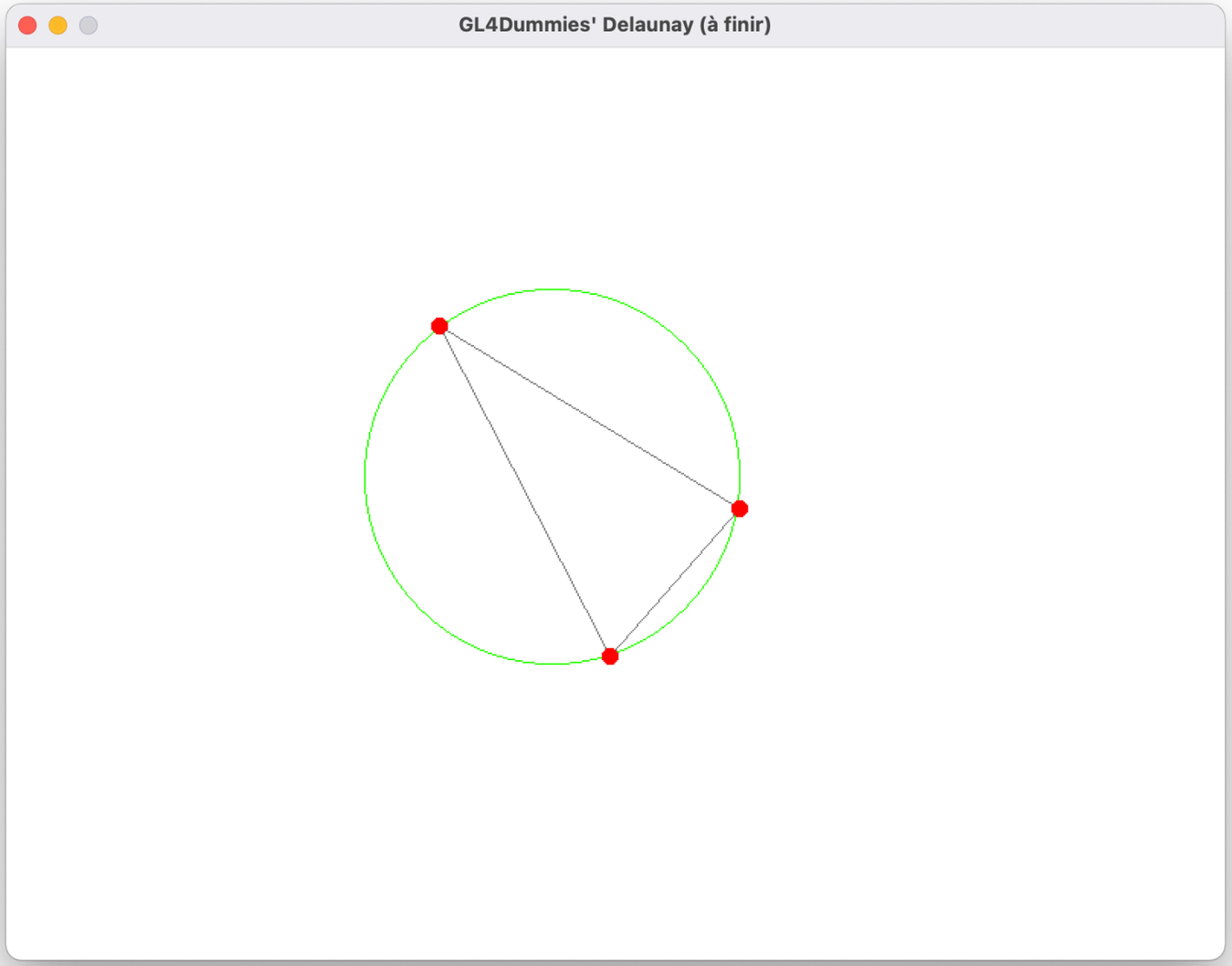
- Il s’agit de construire la triangulation point après point.
- Question : Comment faire car nous commençons par un, puis deux ?
    - Partir d’une triangulation initiale !

De quoi a-t-on besoin ?

- Identifier un triangle (par ses trois sommets)

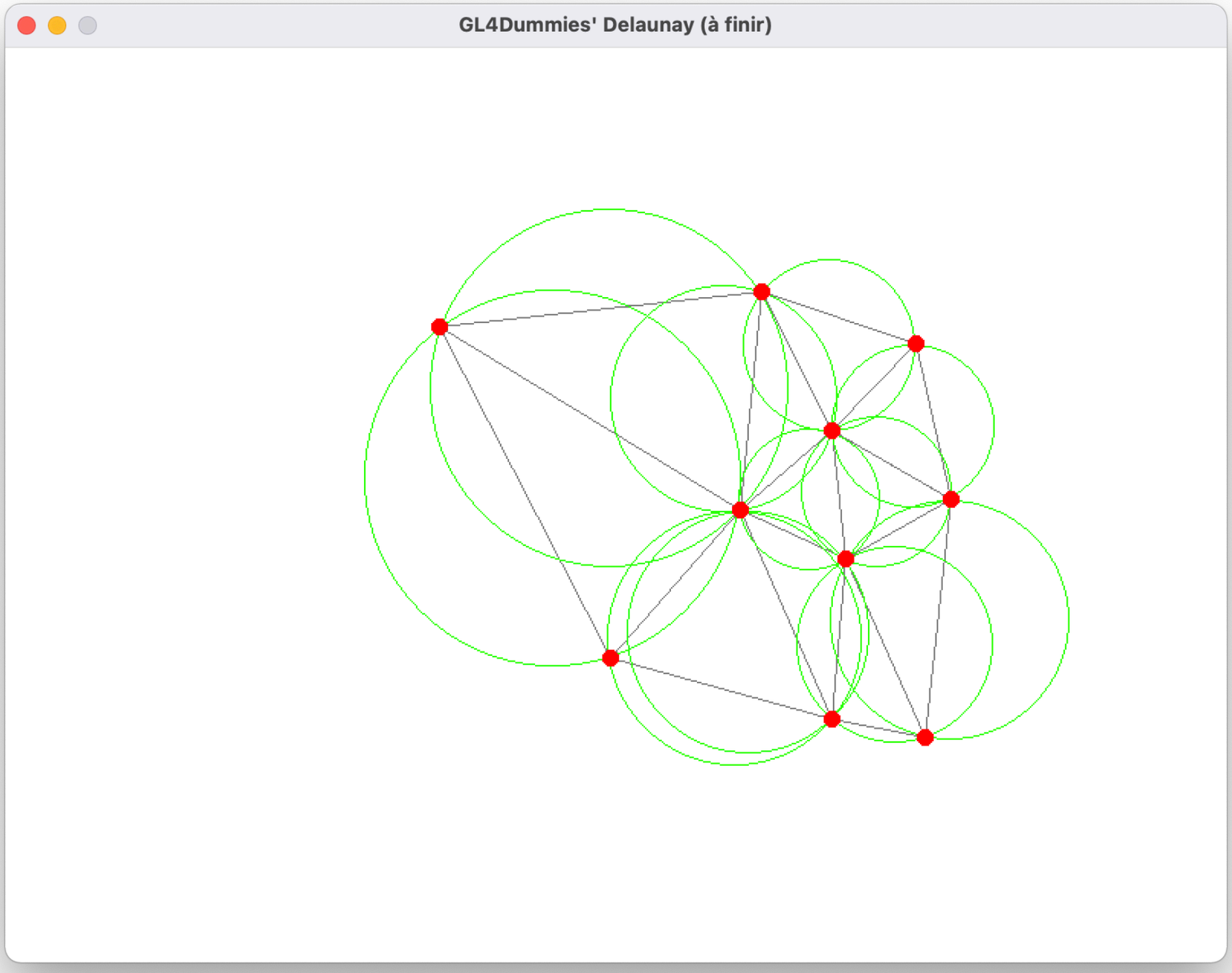
- Savoir calculer le cercle circonscrit au triangle

Un cercle circonscrit à un triangle ?

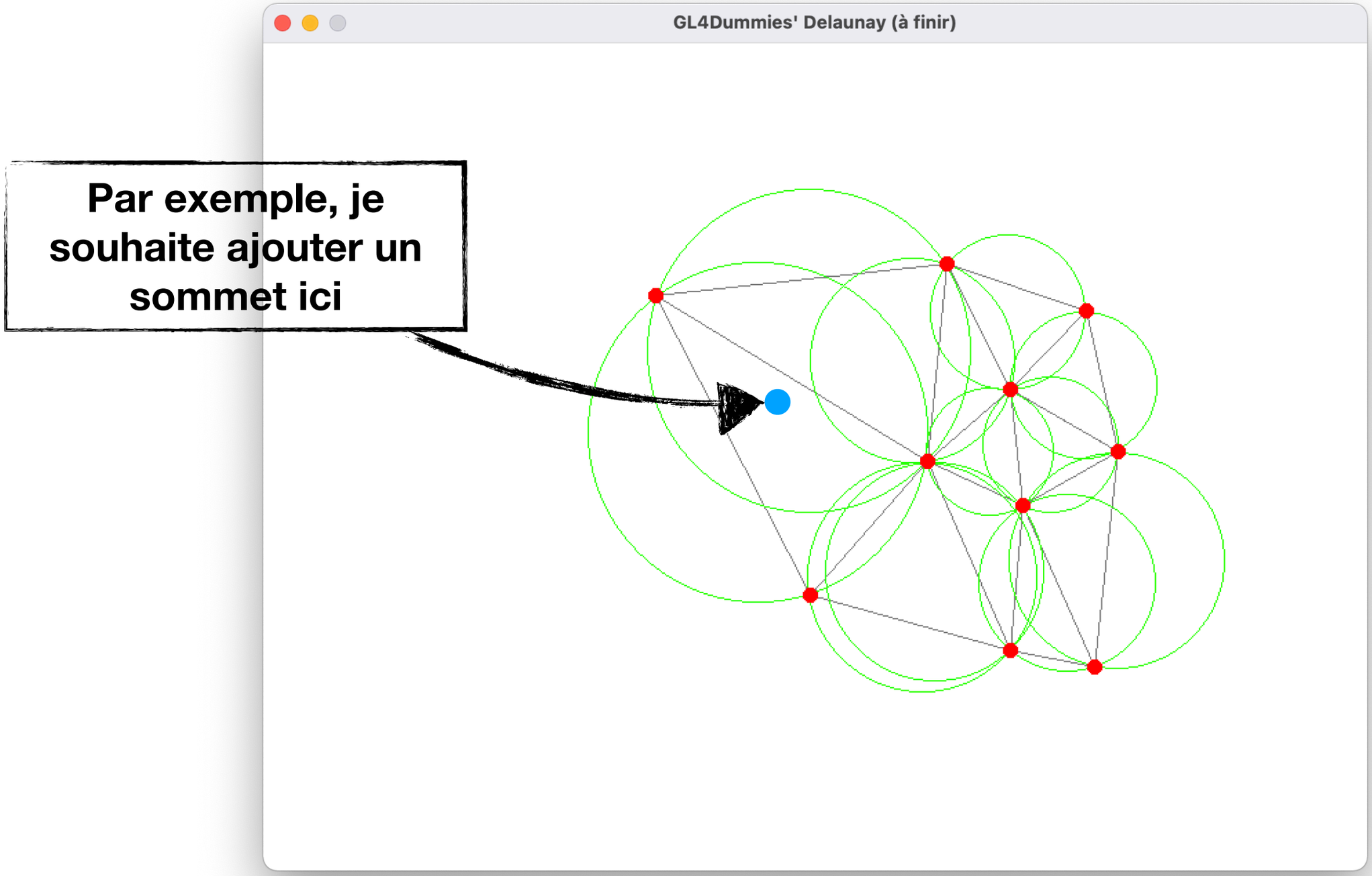


- Le cercle passant par les trois sommets du triangle
- Wiki : https://fr.wikipedia.org/wiki/Cercle_circonscrit_à_un_triangle

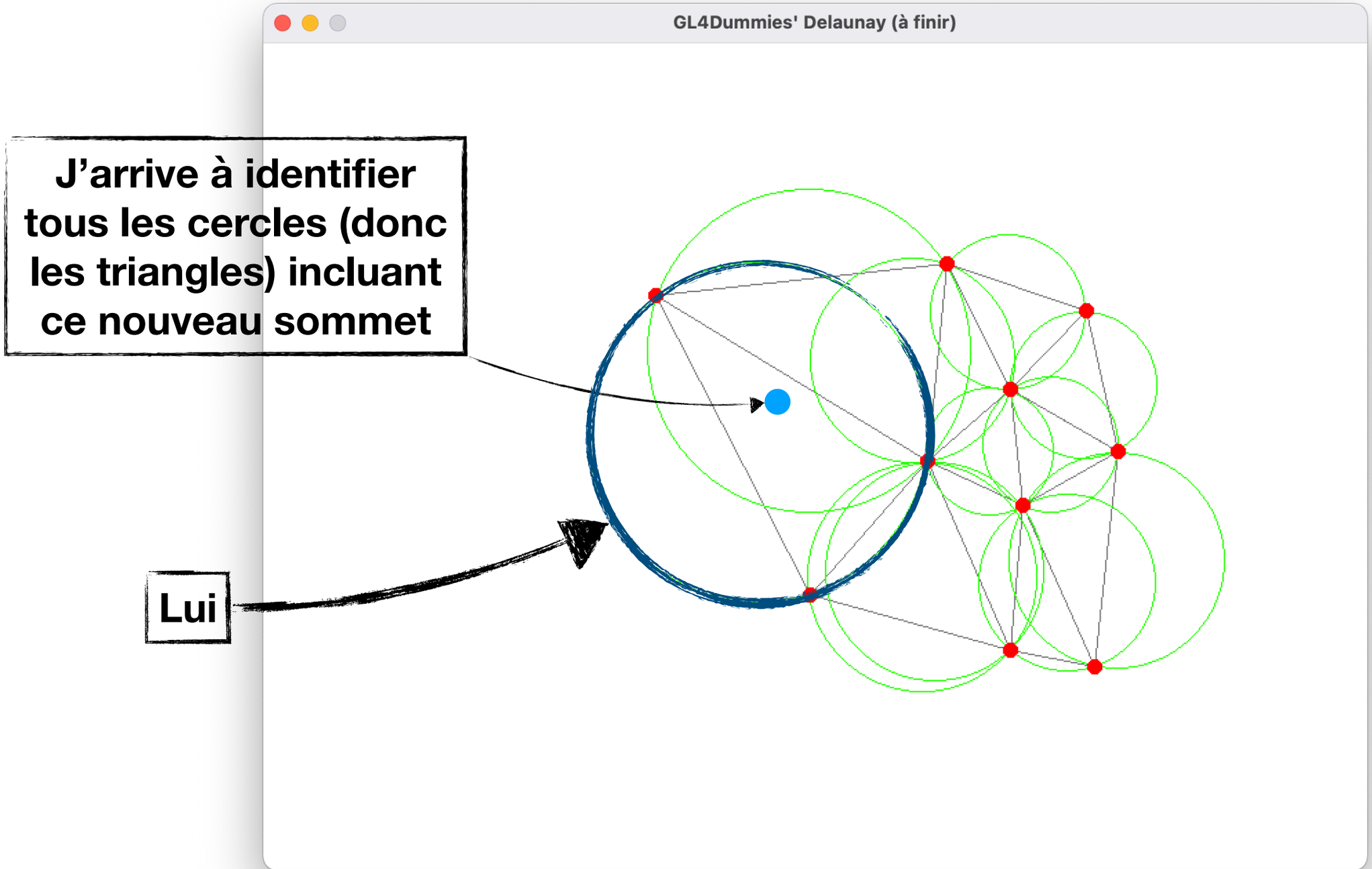
A quoi ça sert ? (1/13)



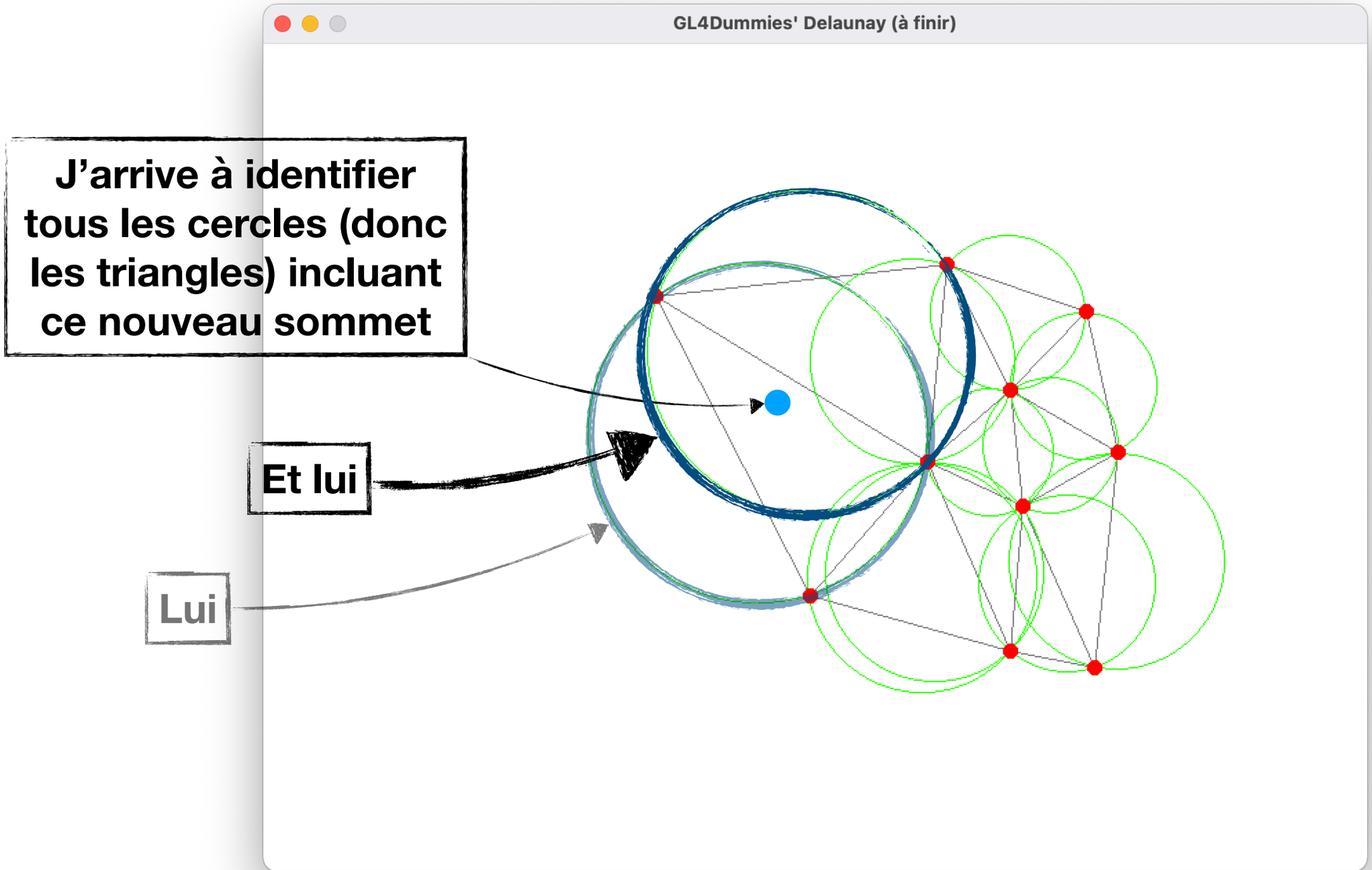
A quoi ça sert ? (2/13)



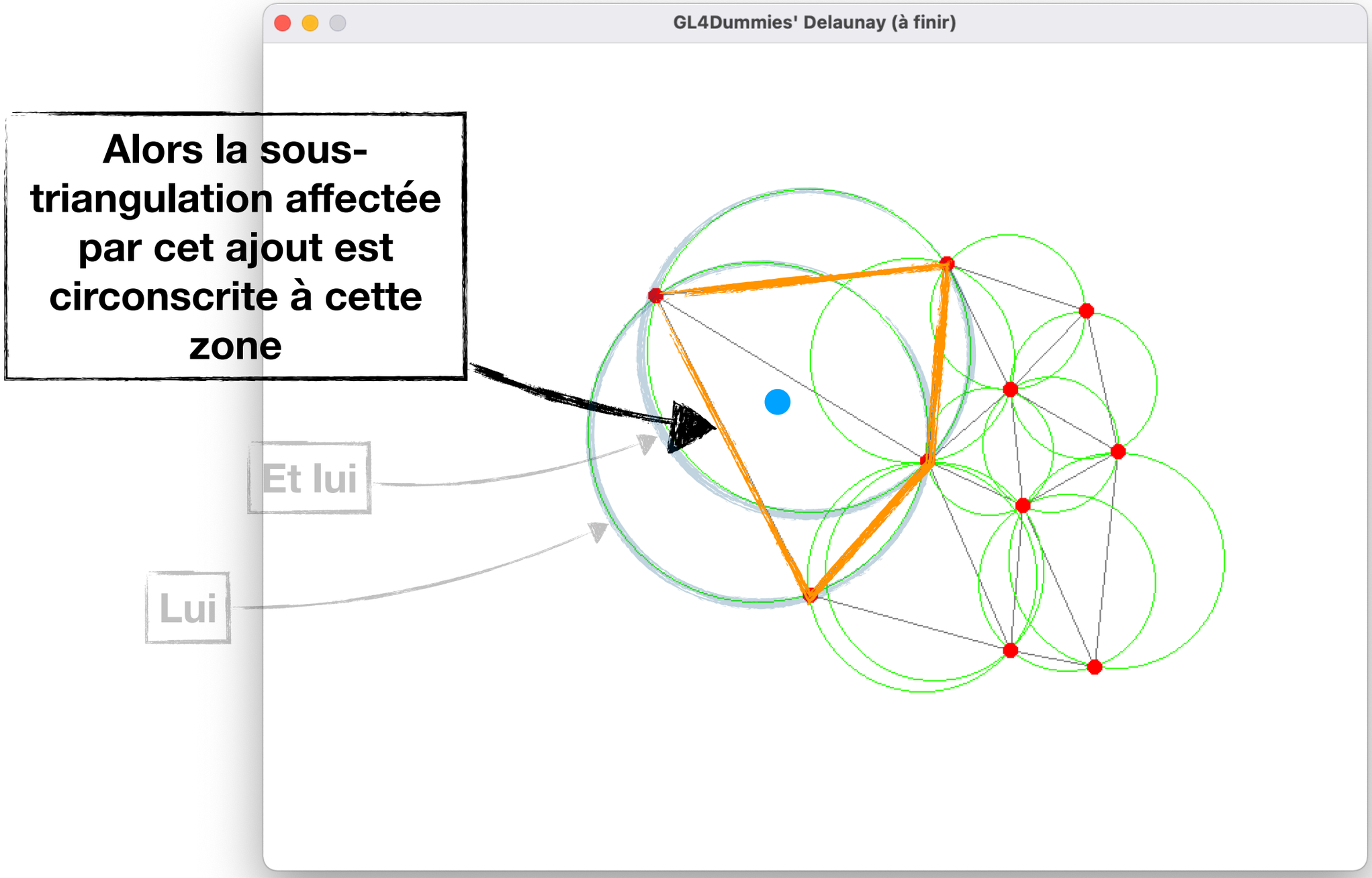
A quoi ça sert ? (3/13)



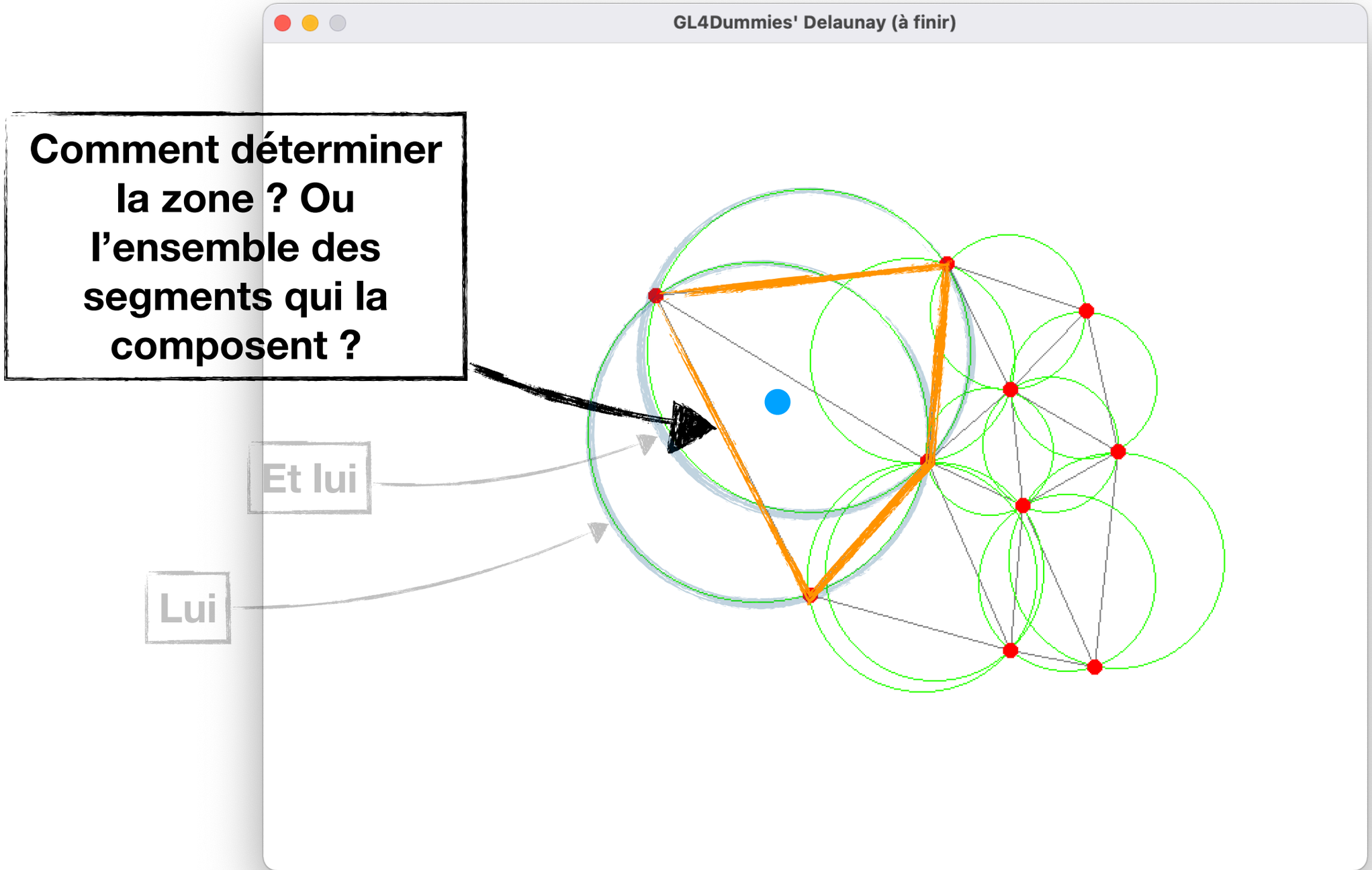
A quoi ça sert ? (4/13)



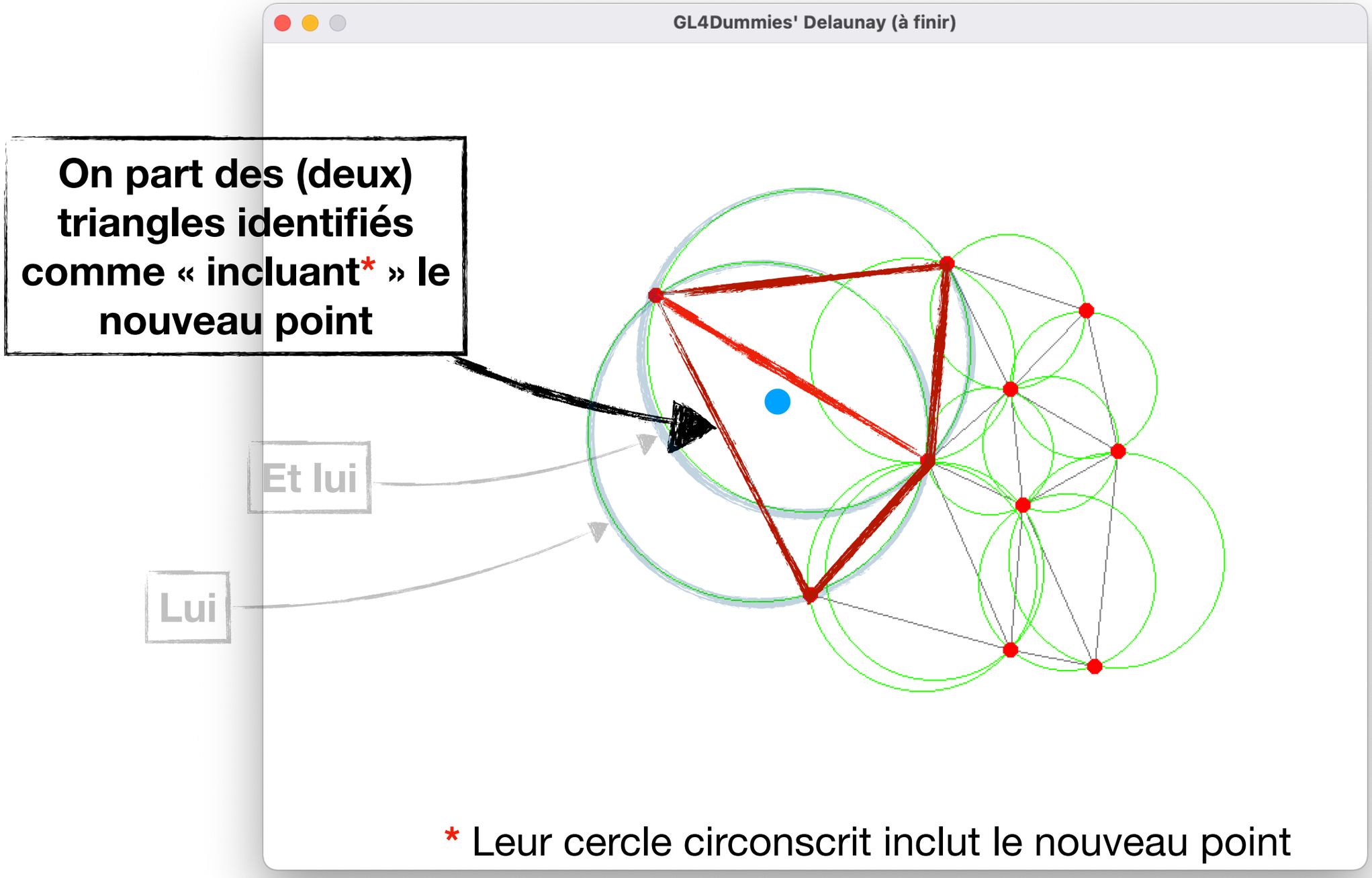
A quoi ça sert ? (5/13)



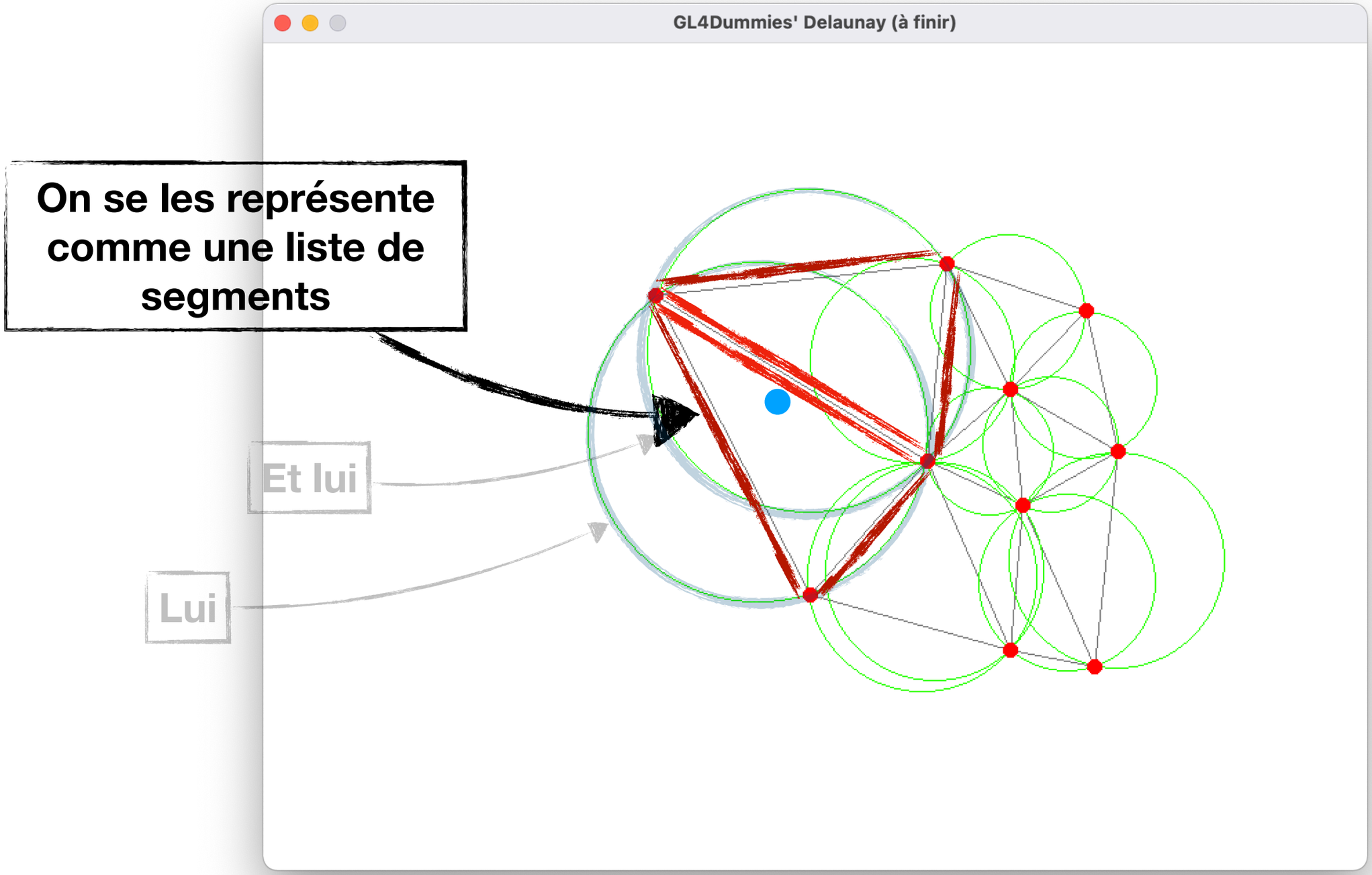
A quoi ça sert ? (6/13)



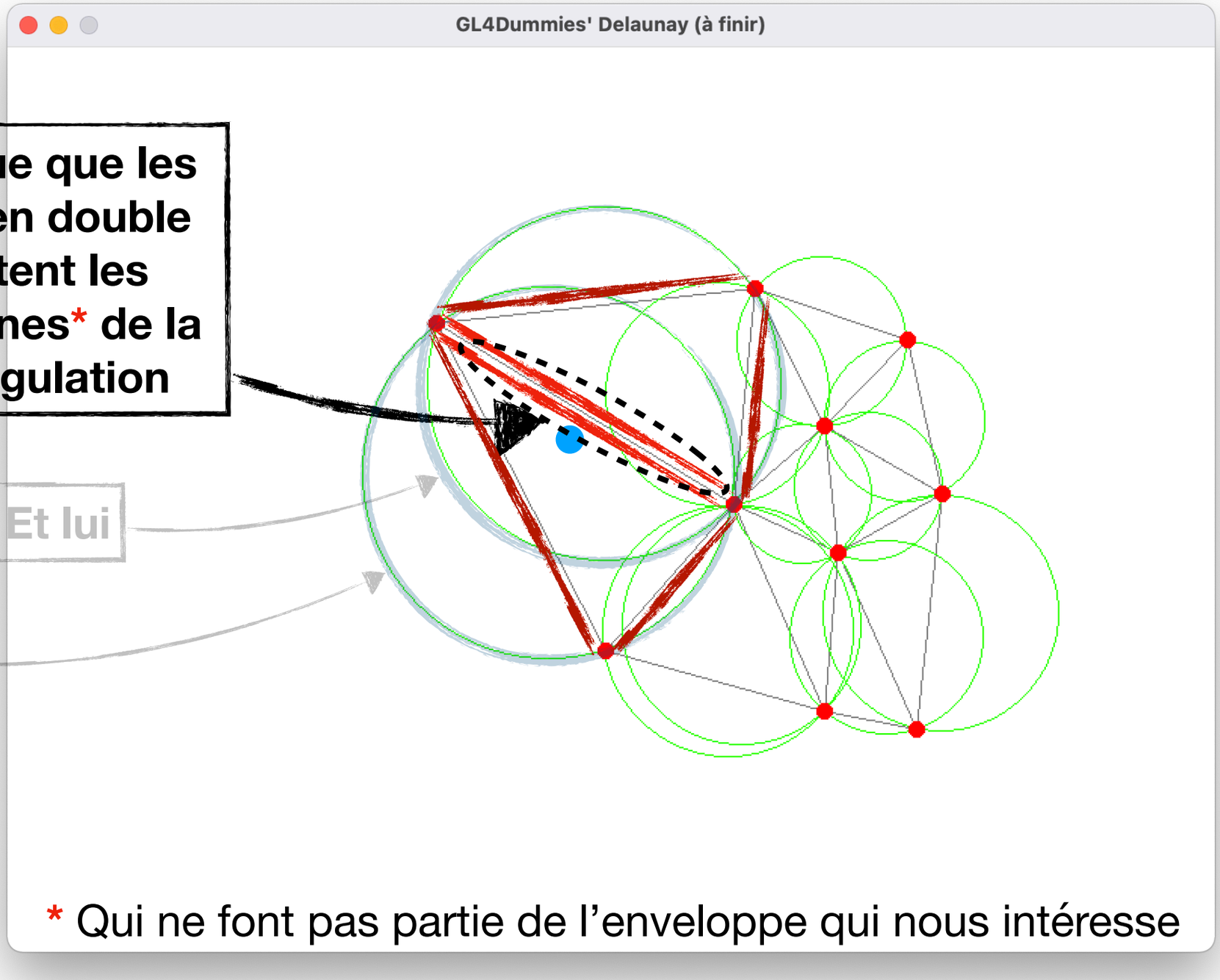
A quoi ça sert ? (7/13)



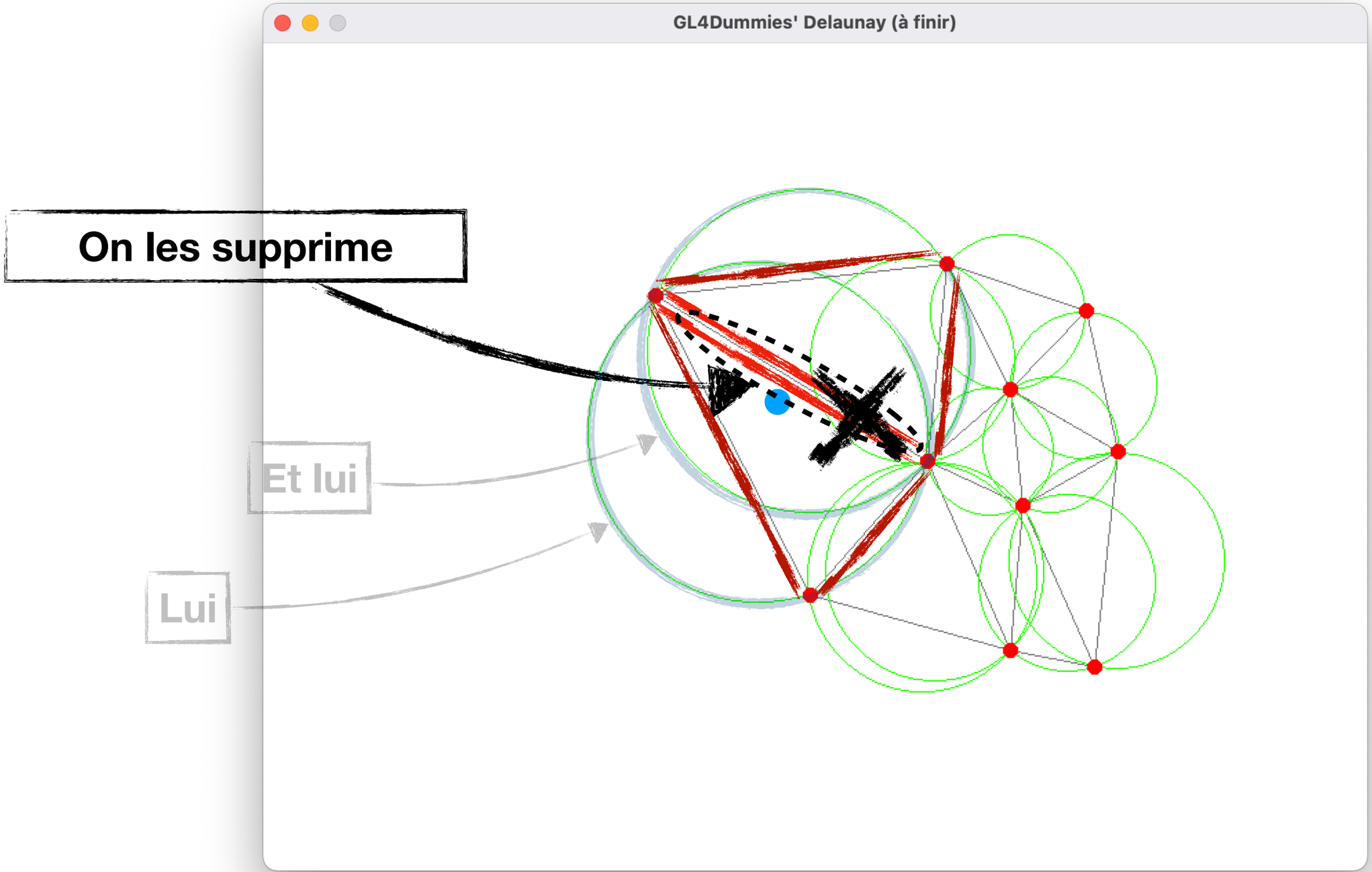
A quoi ça sert ? (8/13)



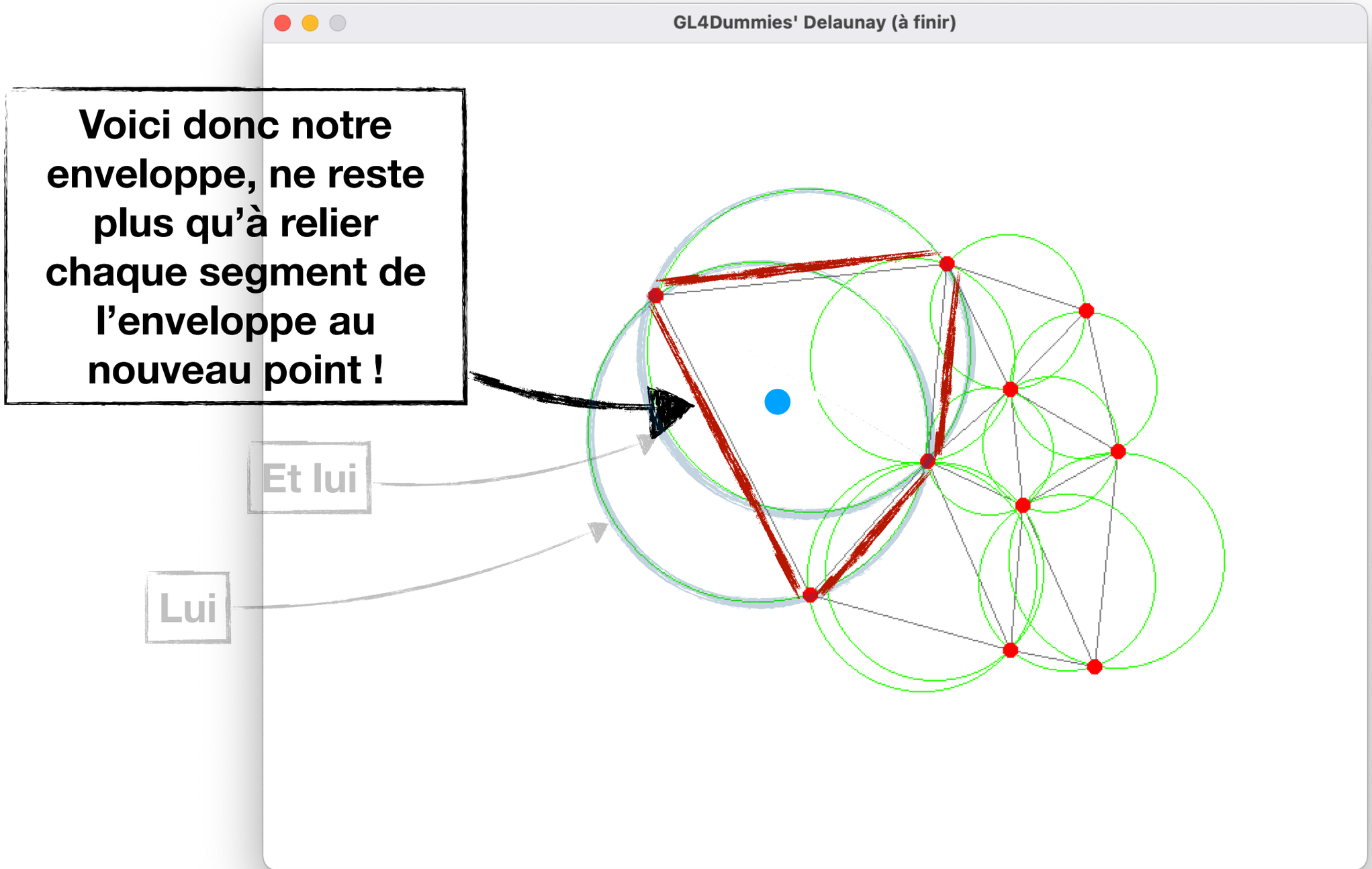
A quoi ça sert ? (9/13)



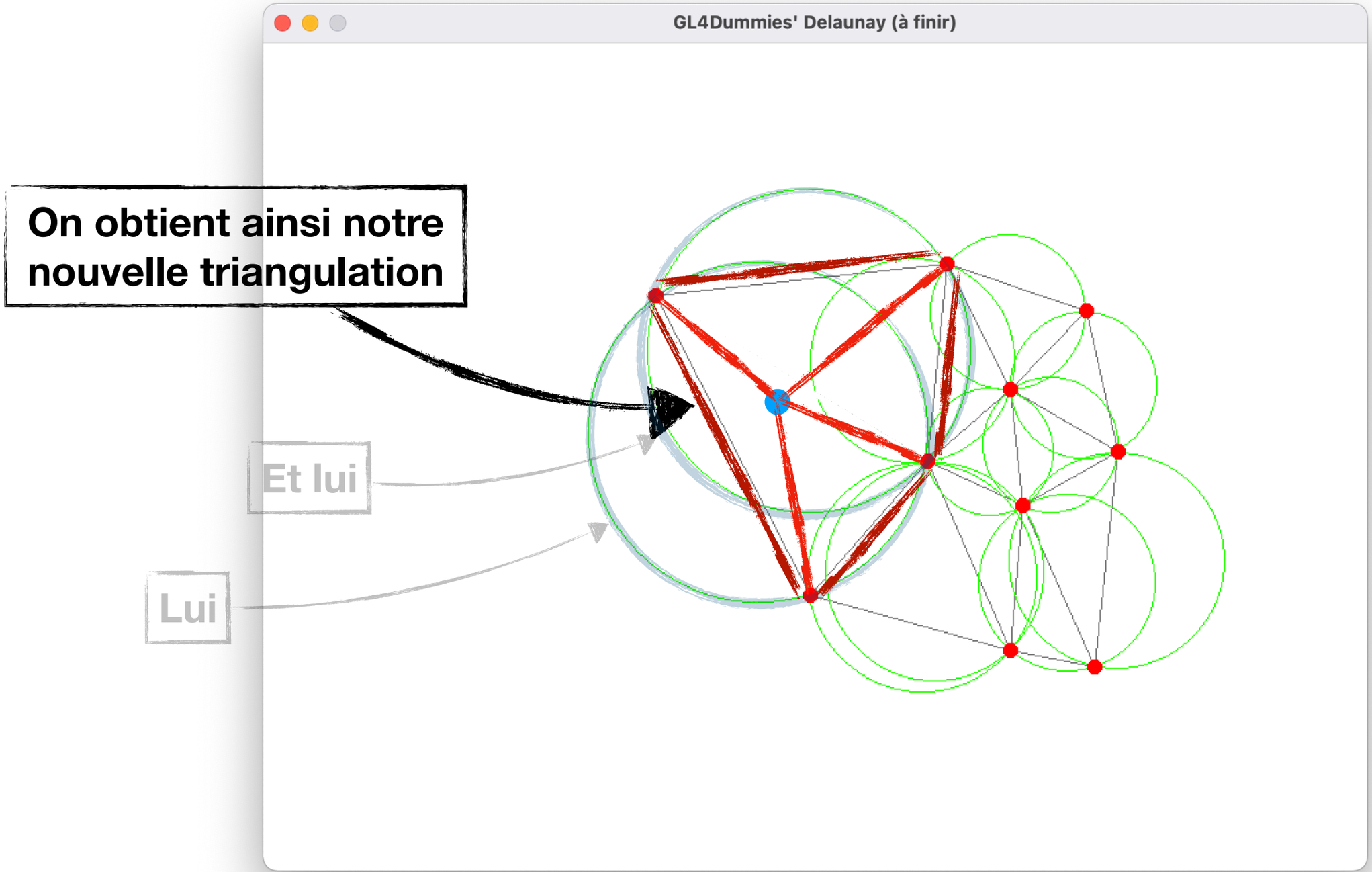
A quoi ça sert ? (10/13)



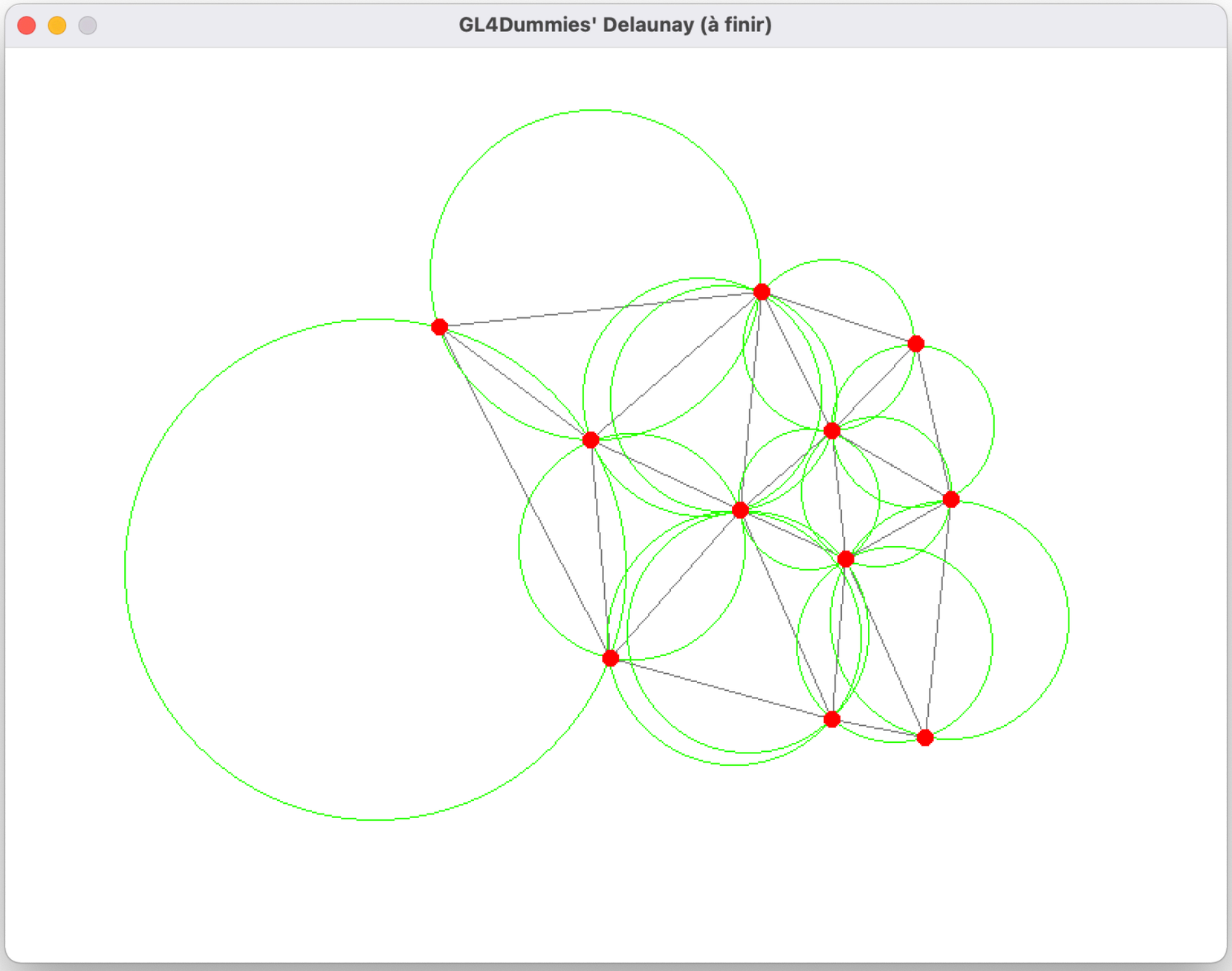
A quoi ça sert ? (11/13)



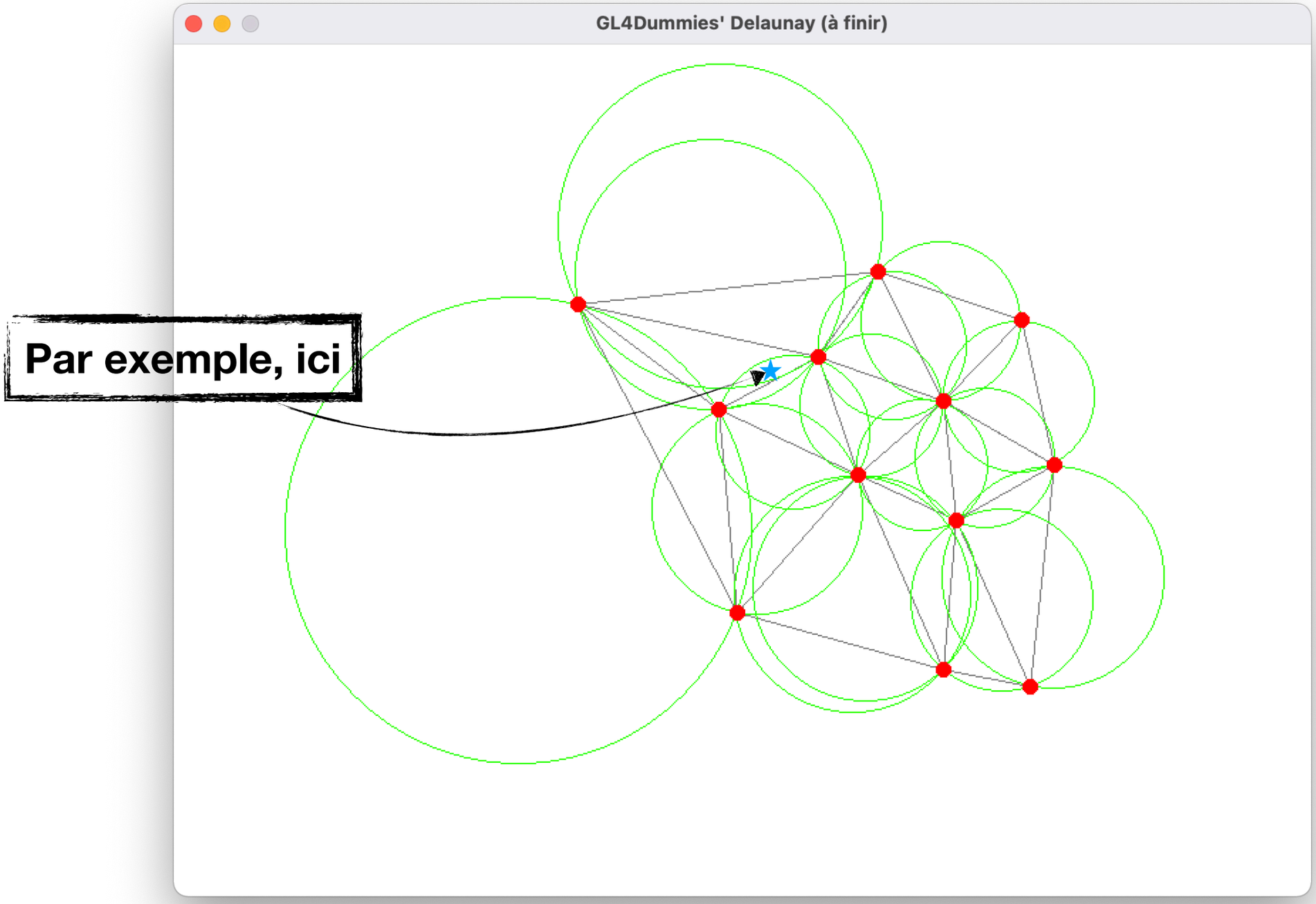
A quoi ça sert ? (12/13)



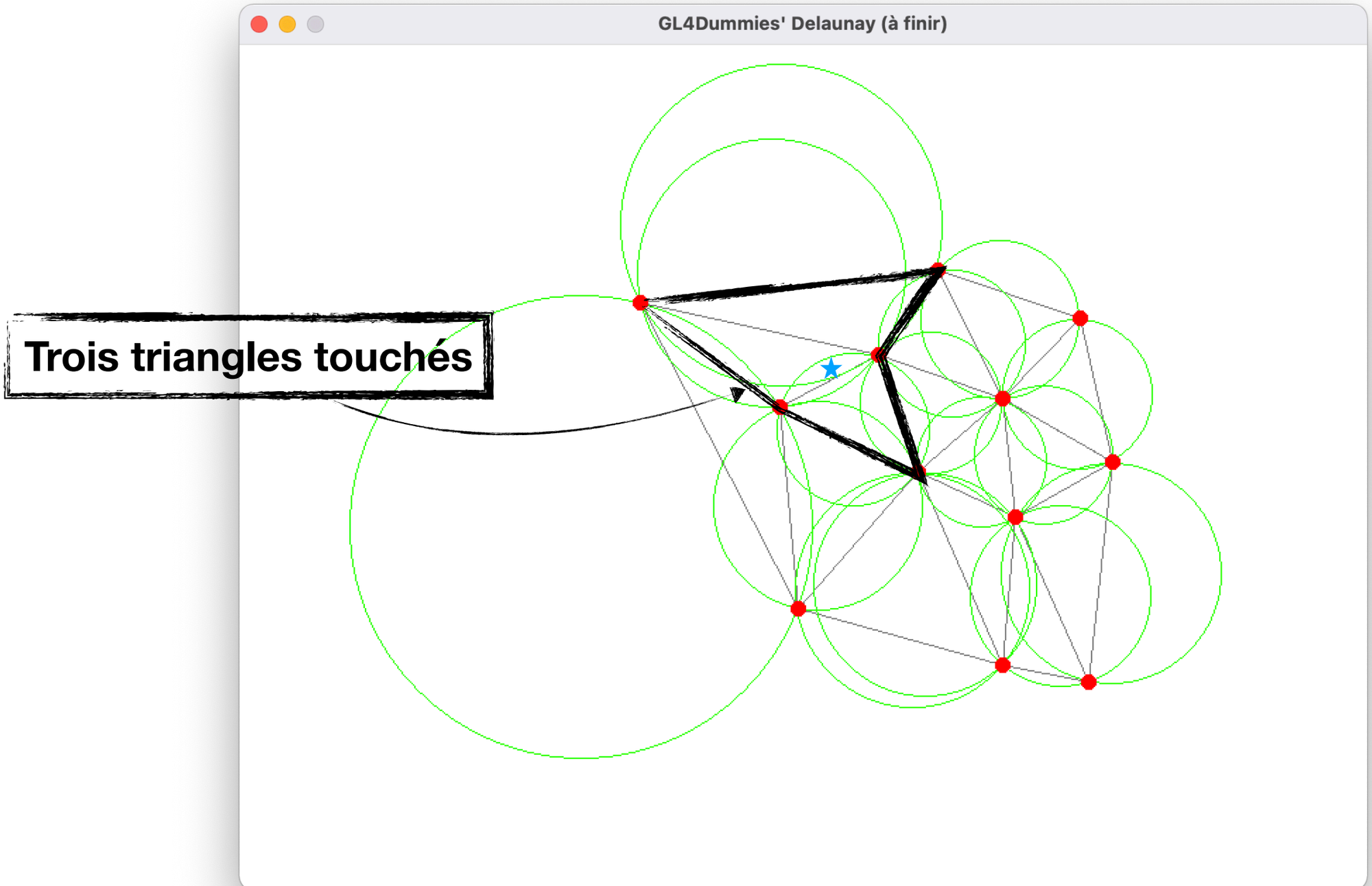
Voici le résultat ! (13/13)



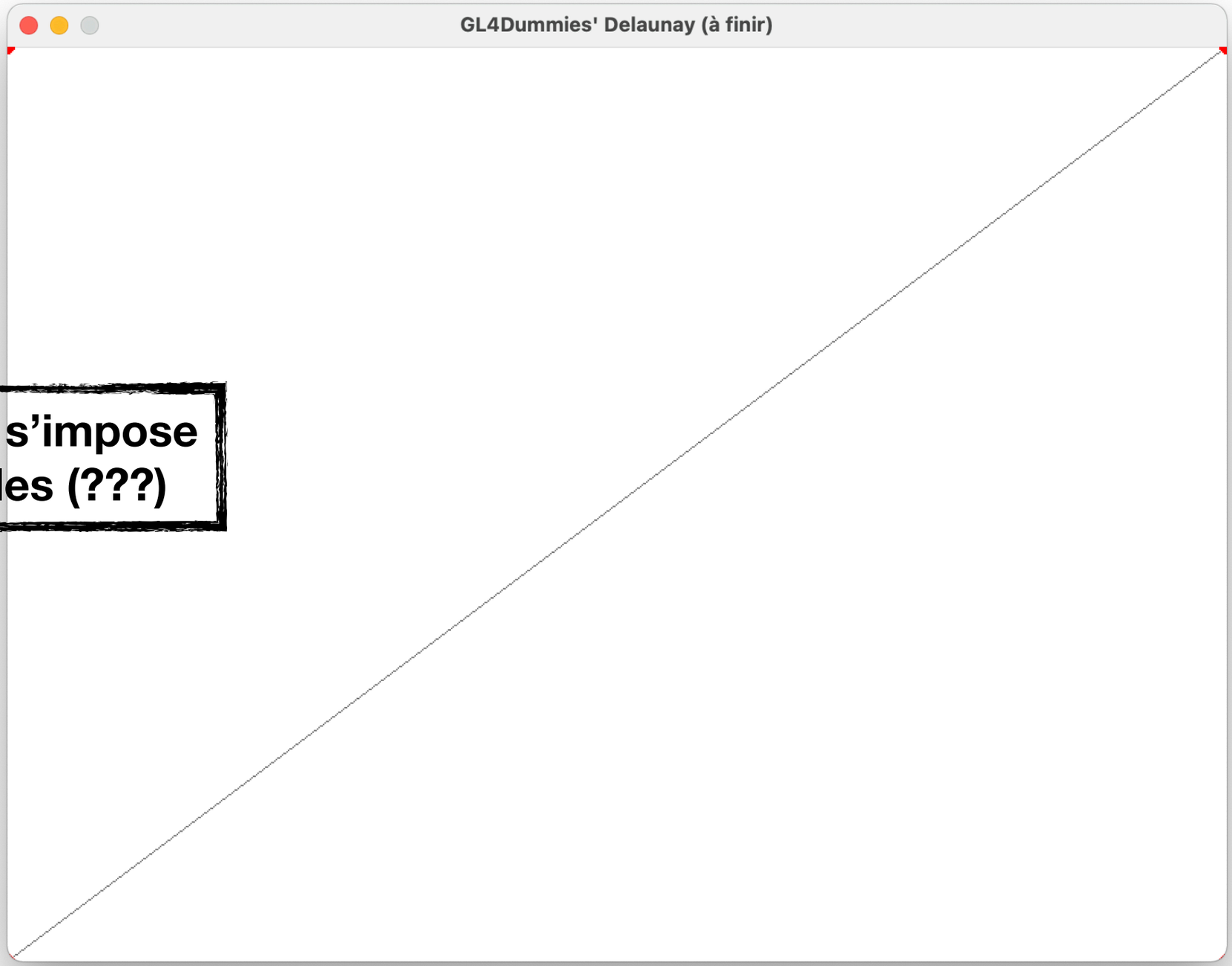
Plus de triangles impactés



■ ■ ■

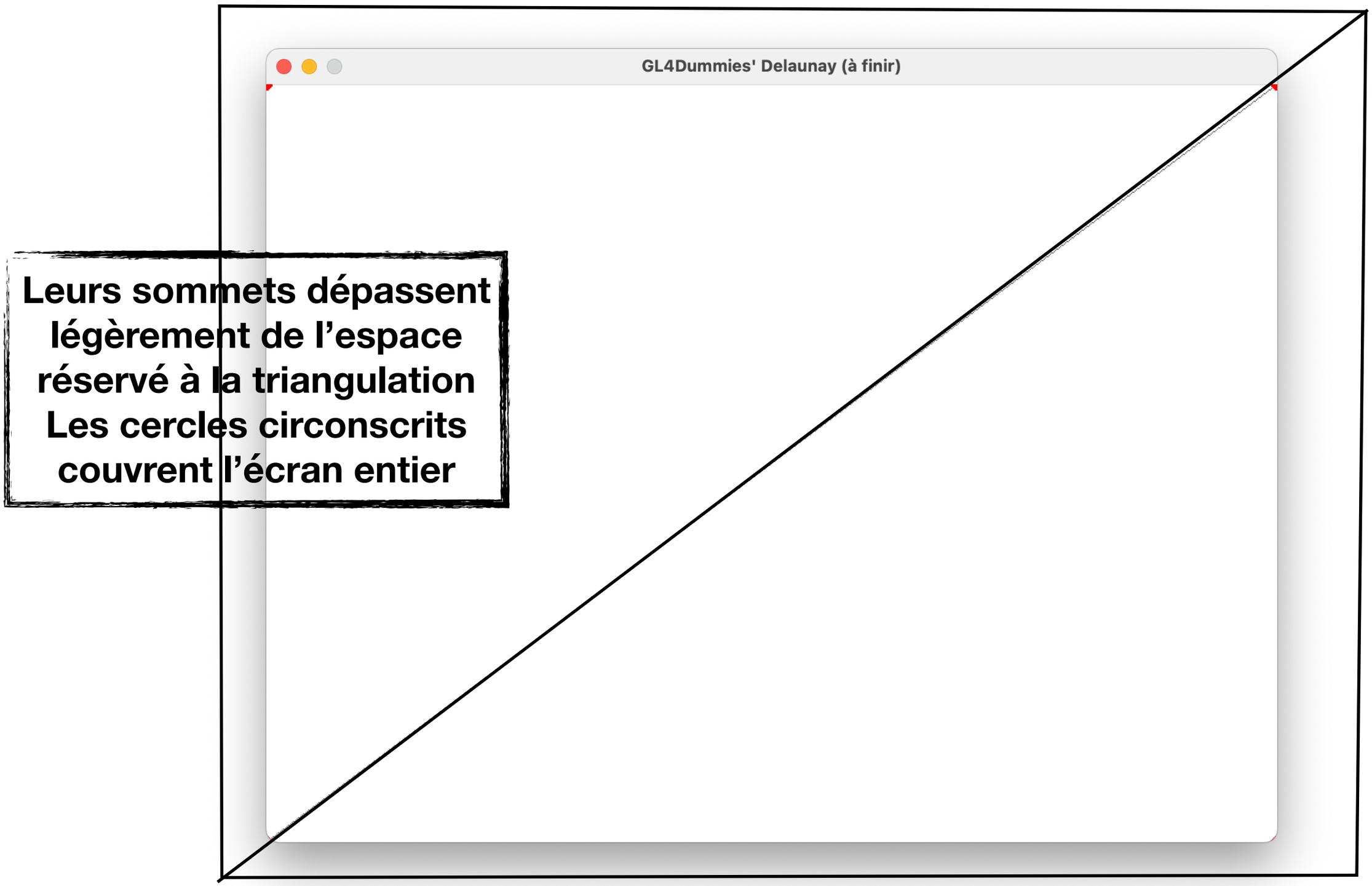

GL4Dummies' Delaunay (à finir)
Trois triangles touchés

OK, mais comment commencer ? (1/6)

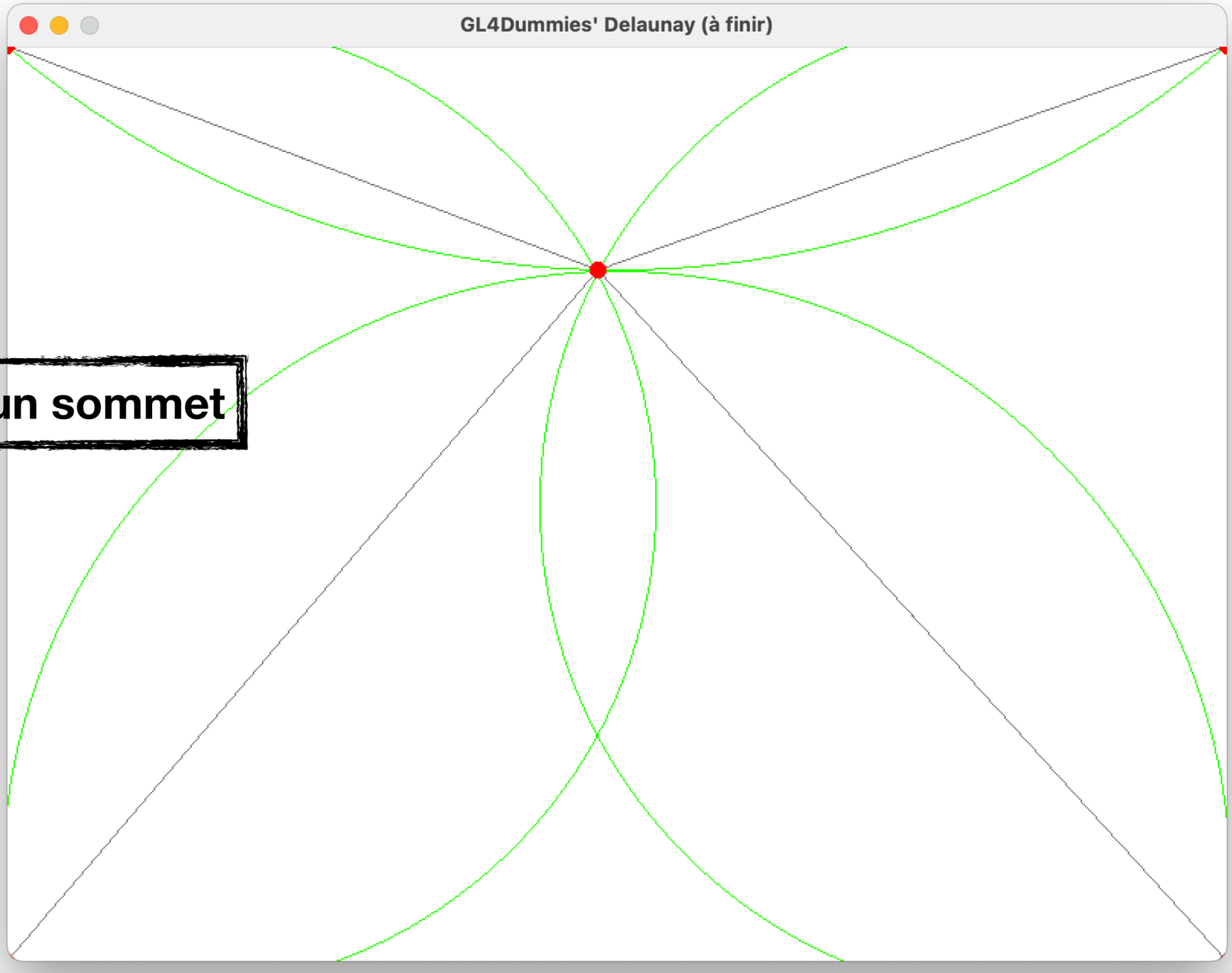


Au début, on s’impose deux triangles (???)

OK, mais comment commencer ? (2/6)

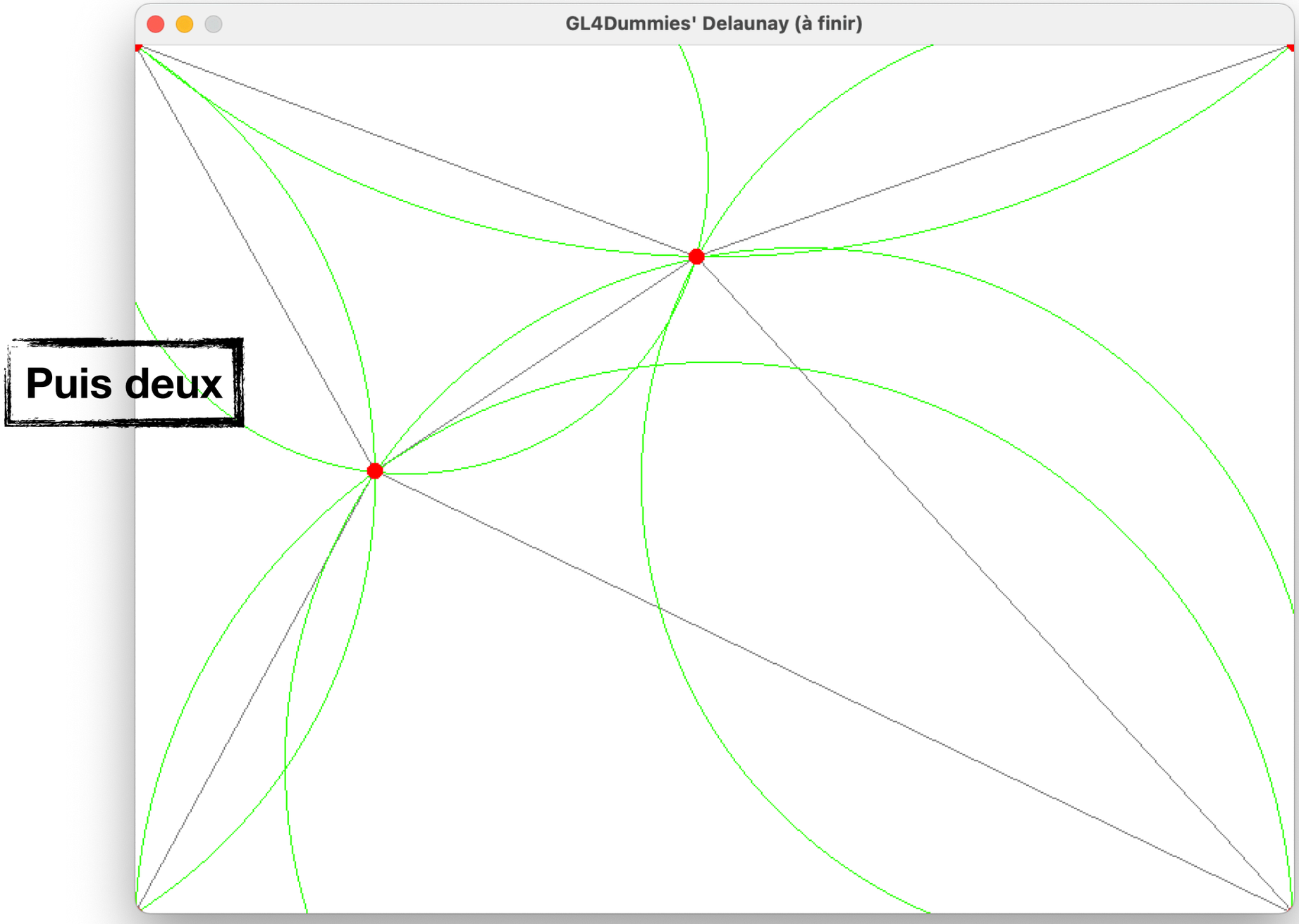


OK, mais comment commencer ? (3/6)

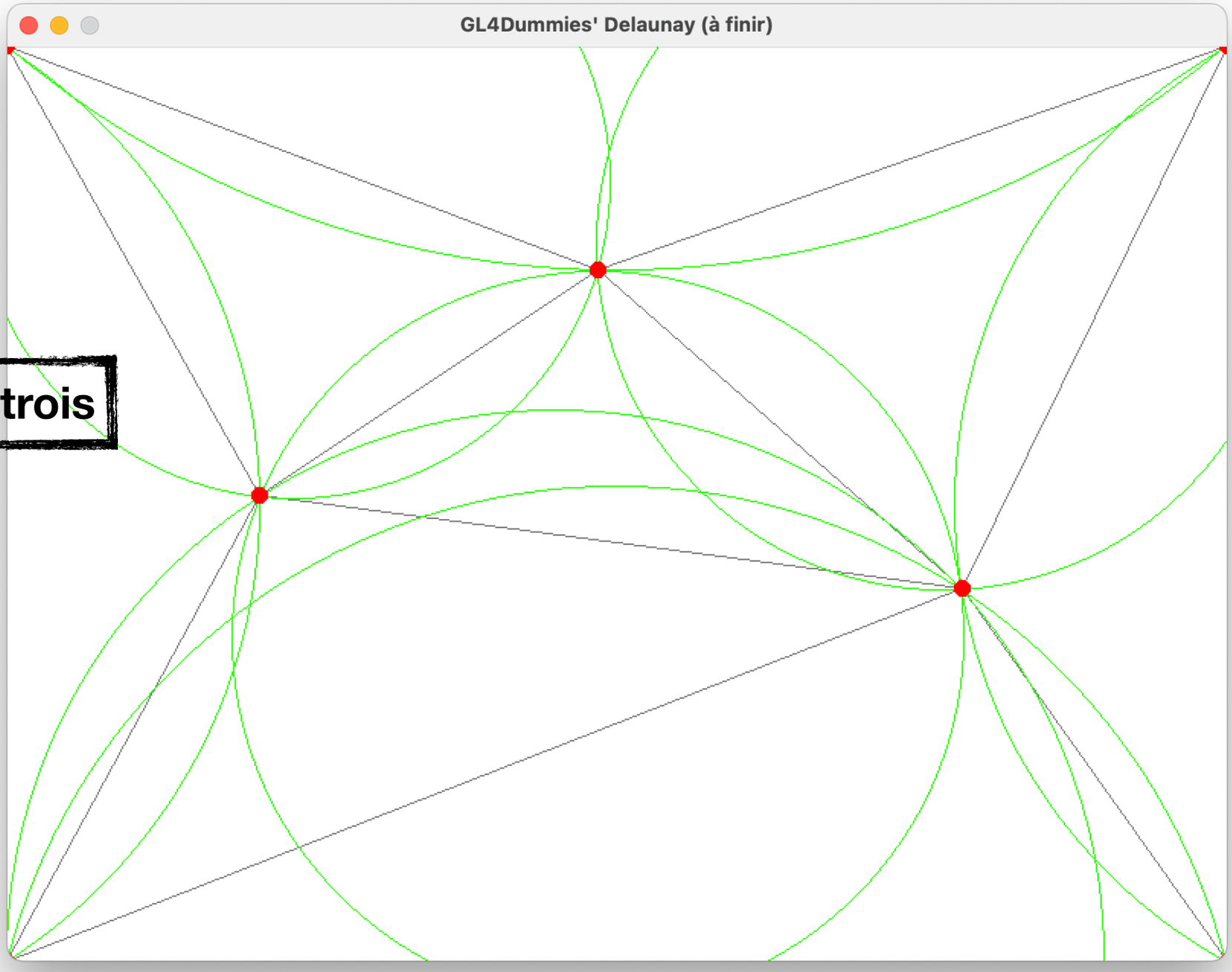


On ajoute un sommet

OK, mais comment commencer ? (4/6)

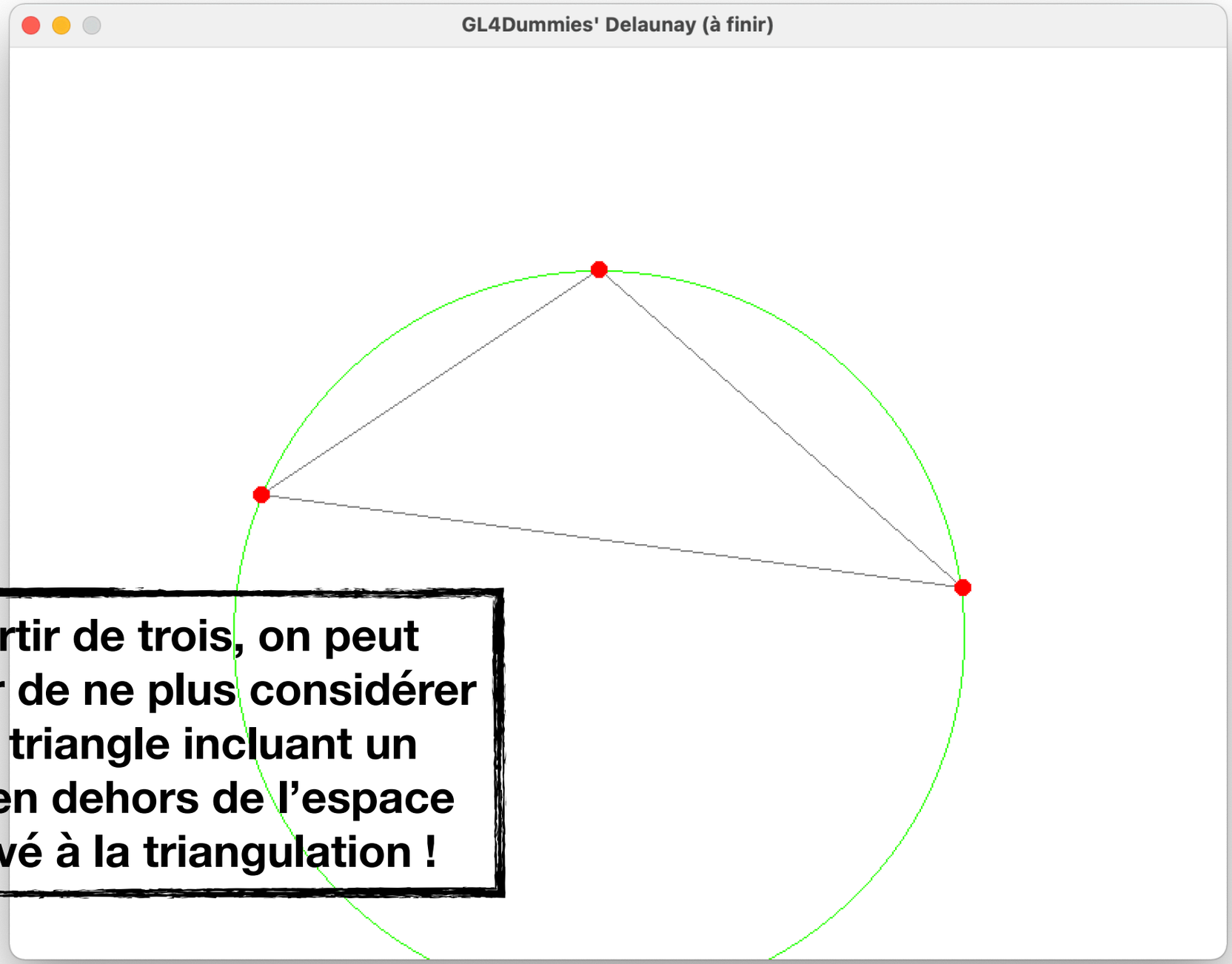


OK, mais comment commencer ? (5/6)



Puis trois

OK, mais comment commencer ? (6/6)



À partir de trois, on peut décider de ne plus considérer tous triangle incluant un point en dehors de l’espace réservé à la triangulation !

Élaborons l’algorithme

- À l’initialisation :
  - Avoir une liste de triangles **TRL** où on a déjà inséré deux triangles « englobant l’écran » (voir page 21) ; chaque triangle est automatiquement associé à son cercle circonscrit (pré-calcul)
  - Avoir une liste de segments **SL** vide
  - Attention : ces listes doivent garantir l’unicité des éléments (surtout sur les segments)
  - Avoir un tableau de **sommets** qui composent la future triangulation ou une interface qui permet de les ajouter un par un
- À chaque « nouveau » **sommet P** à ajouter :
  - Pour tout **T** dans **TRL**
    - Si **P** est dans le cercle circonscrit de **T** :
      - Pour chacun des trois segments de **T,** noté **S** :
        - Si S est déjà dans SL, supprimer l’occurrence déjà dans **SL**
        - Sinon : insérer **S** à **SL**
  - Pour tous les segments **S** dans **SL** :
    - Composer le triangle **T** à partir des deux sommets de **S** et du sommet **P**
    - Calculer le cercle circonscrit de **T**
    - Insérer **T** dans **TRL**
  - Vider la liste **SL**

Aide (1/3)
Quelques structures

```c
typedef struct point2di point2di_t;
typedef struct point2df point2df_t;
typedef struct segment segment_t;
typedef struct triangle triangle_t;
struct point2di {
  int x, y;
};
struct point2df {
  float x, y;
};
struct segment {
  point2di_t p[2]; /* les deux points du segment */
};
struct triangle {
  point2di_t p[3]; /* les trois points du triangle */
  point2df_t ccc; /* centre du cercle circonscrit */
  float ccr; /* rayon du cercle circonscrit */
};
```

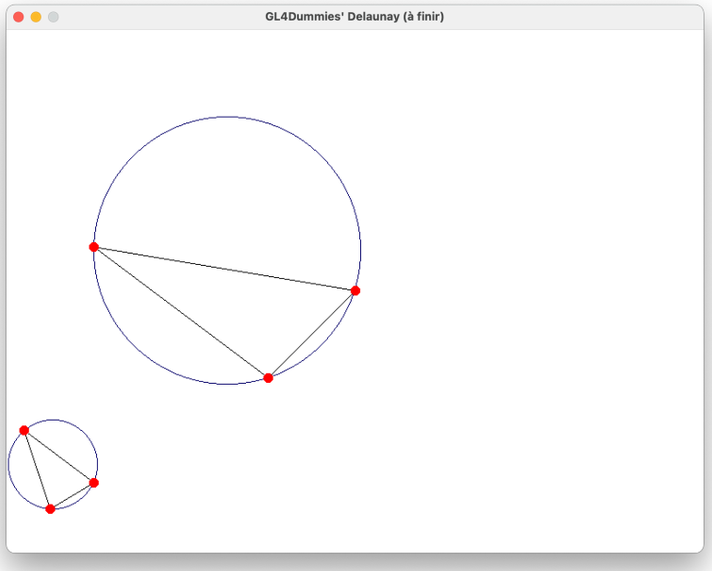
Aide (2/3)
Quelques fonctions

```c
/* déterminant d'une matrice 3x3 */
static inline float mat3_det(float * mat) {
  /* en ajoutant les copies des deux premières lignes */
  return ( (mat[0] * mat[4] * mat[8] + mat[3] * mat[7] * mat[2] + mat[6] * mat[1] * mat[5]) -
           (mat[3] * mat[1] * mat[8] + mat[0] * mat[7] * mat[5] + mat[6] * mat[4] * mat[2]) );
  /* ou en ajoutant les copies des deux premières colonnes */
  /* return ( (mat[0] * mat[4] * mat[8] + mat[1] * mat[5] * mat[6] + mat[2] * mat[3] * mat[7]) - */
  /*       (mat[6] * mat[4] * mat[2] + mat[7] * mat[5] * mat[0] + mat[8] * mat[3] * mat[1]) ); */
}

/* carré de la distance euclidienne d'un point à coordonnées entières */
static inline int eucl_dist2_i(point2di_t * p) {
  return p->x * p->x + p->y * p->y;
}

/* la distance euclidienne d'un point à coordonnées entières */
static inline float eucl_dist_i(point2di_t * p) {
  return sqrtf((float)eucl_dist2_i(p));
}

/* carré de la distance euclidienne d'un point à coordonnées flottantes */
static inline float eucl_dist2_f(point2df_t * p) {
  return p->x * p->x + p->y * p->y;
}

/* la distance euclidienne d'un point à coordonnées flottantes */
static inline float eucl_dist_f(point2df_t * p) {
  return sqrtf(eucl_dist2_f(p));
}

/* calcule le cercle circonscrit pour un triangle */
static inline void compute_cc_4_triangle(triangle_t * t) {
  float m0[] = {
    t->p[0].x, t->p[0].y, 1.0f,
    t->p[1].x, t->p[1].y, 1.0f,
    t->p[2].x, t->p[2].y, 1.0f
  };
  float de0 = eucl_dist2_i(&t->p[0]);
  float de1 = eucl_dist2_i(&t->p[1]);
  float de2 = eucl_dist2_i(&t->p[2]);
  float m1[] = {
    de0, t->p[0].y, 1.0f,
    de1, t->p[1].y, 1.0f,
    de2, t->p[2].y, 1.0f
  };
  float m2[] = {
    de0, t->p[0].x, 1.0f,
    de1, t->p[1].x, 1.0f,
    de2, t->p[2].x, 1.0f
  };
  float delta = 2.0f * mat3_det(m0);
  point2df_t pr;
  assert(delta);
  t->ccc.x = mat3_det(m1) / delta;
  t->ccc.y = -mat3_det(m2) / delta;
  pr.x = t->ccc.x - t->p[0].x;
  pr.y = t->ccc.y - t->p[0].y;
  t->ccr = eucl_dist_f(&pr);
}
```

Aide (3/3)
Deux codes pour commencer

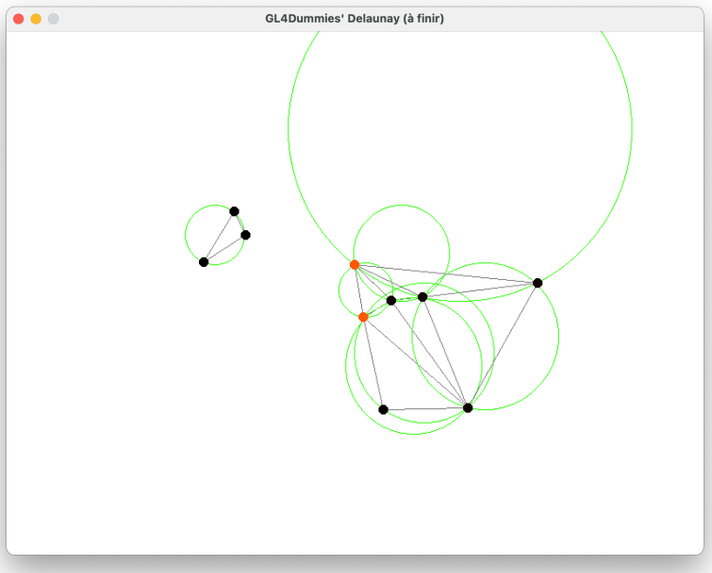
Version 0.1 : permet de montrer que les calculs de cercles circonscrits effectués

https://expreg.org/amsi/C/APG2223S1/code/sc_00_03_delaunay_it-0.1.tgz



Version 0.2 : permet de créer des triangles à la main (et même d'utiliser des sommets existants

https://expreg.org/amsi/C/APG2223S1/code/sc_00_03_delaunay_it-0.2.tgz



DM Final

- En partant d'un des deux codes donnés sur la précédente page, finaliser l'implémentation de l'algorithme itératif de la triangulation de Delaunay.

- Bonus (seconde archive pouvant être soumise) : utiliser une triangulation générée pour la texturer (exercice en fin de support de rastérisation) ; produire un mouvement non uniforme sur les sommets pour montrer la déformation liée au *mapping* de texture.

- Date limite de soumission : 26/12/22 à 10h (moodle)