



**PROJET**

**FRACTALES**

**AUTEURS :**  
**PEZIER Pierre-Henri**  
**LECOMTE David**

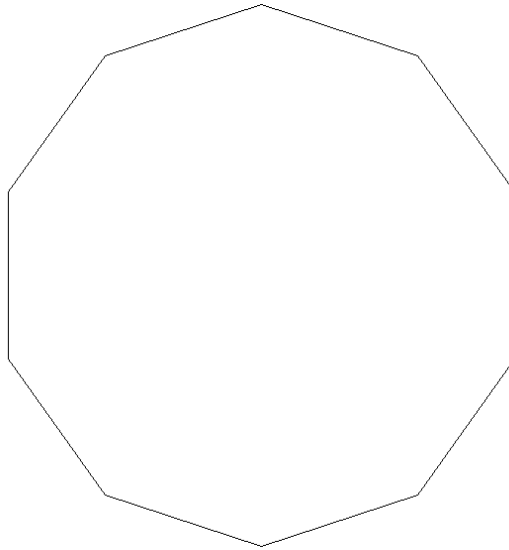
**FILIERE / ANNEE**  
**STI / 2007**

## I – Les différents contours :

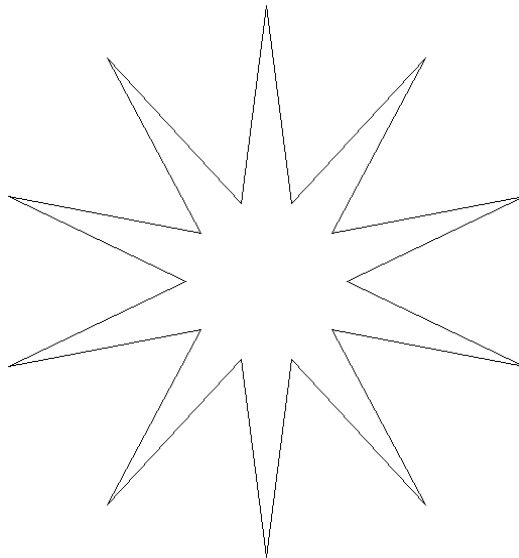
### 1) Les contours simples :

Notre programme nous permet de créer les contours simples suivants :

- polygones réguliers



- étoiles régulières



- cercles
- rectangles

Nous avons choisi de stocker les sommets qui constituent ces figures dans des tableaux (en choisissant le sens trigonométrique comme référence).

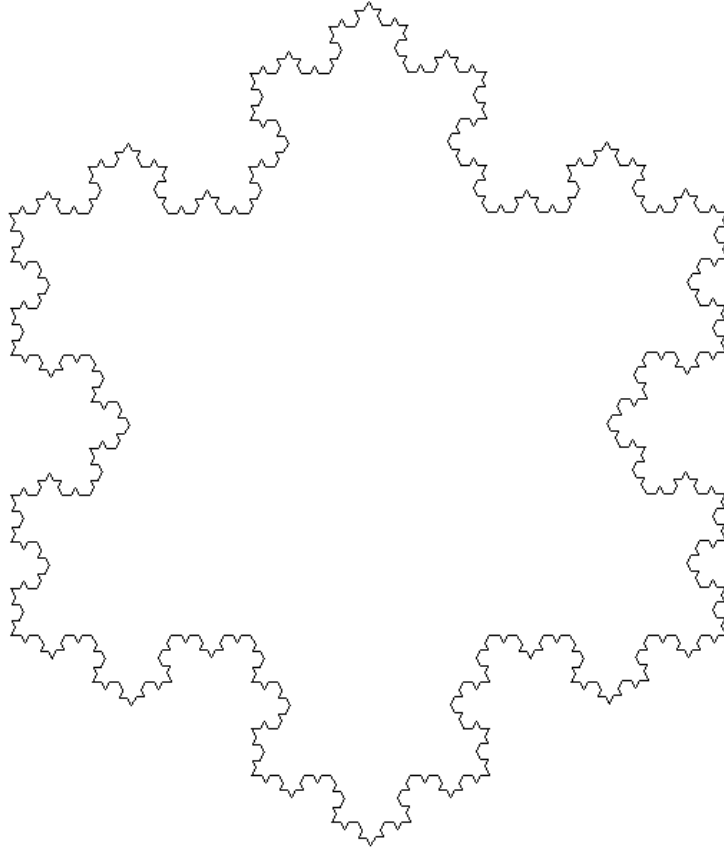
Nous avons modélisé les cercles par des polygones à 60 sommets.

Le fait de stocker les points permet par la suite de colorer les figures ou de déterminer si un point se situe dans un contour ou à l'extérieur.

## 2) Les contours de fractals :

Nous avons la possibilité de tracer le contour du triangle de Koch dont tous les points de construction sont contenus dans un tableau, ce qui nous permet par la suite de colorer le flocon ou de le pavé avec des triangles.

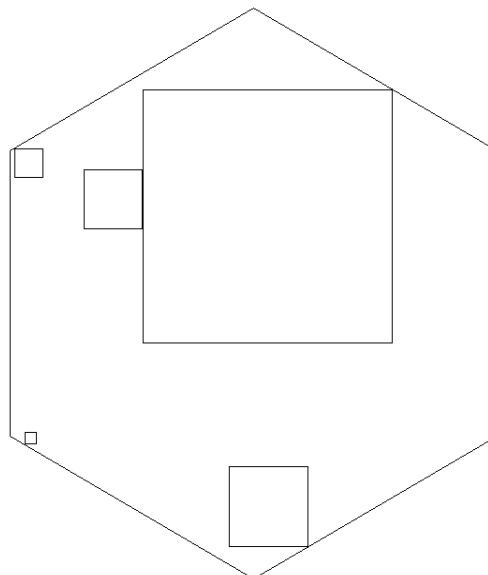
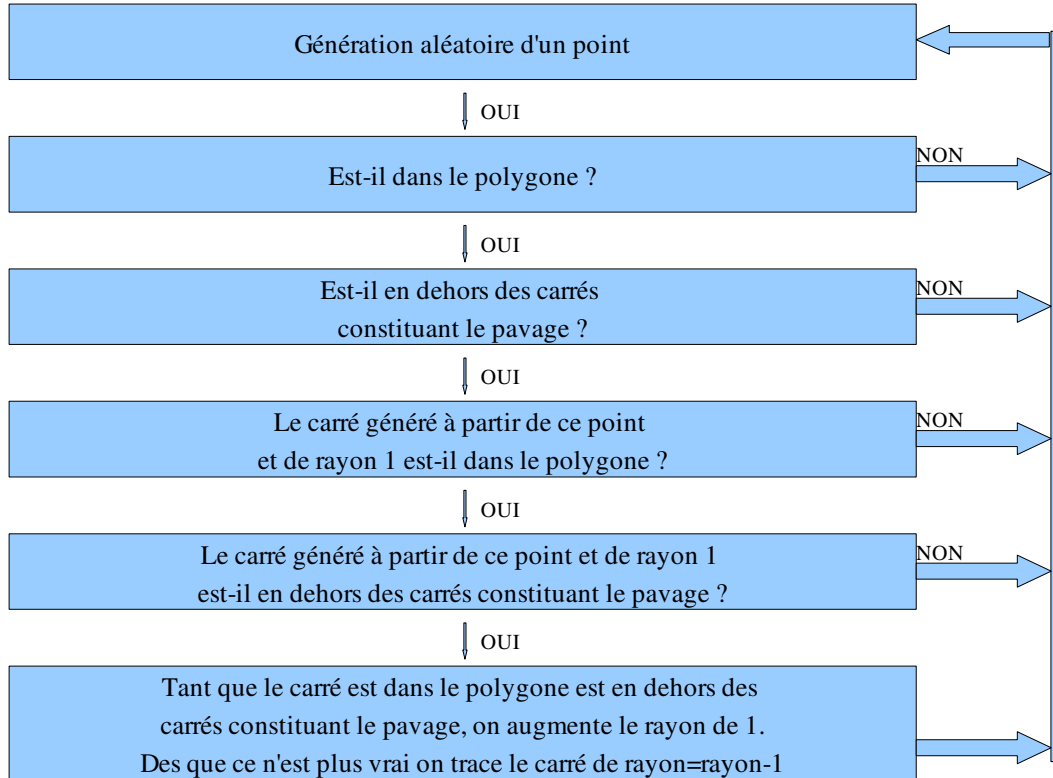
Le flocon se construit de la façon suivante : on donne un segment au programme qui va déterminer le troisième point pour former un triangle équilatéral, puis il va appliquer la transformation de Koch pour chaque segment dans le sens horaire et réitère cette opération sur les segments créés jusqu'à obtenir l'ordre souhaité.



## II – Les différents pavages :

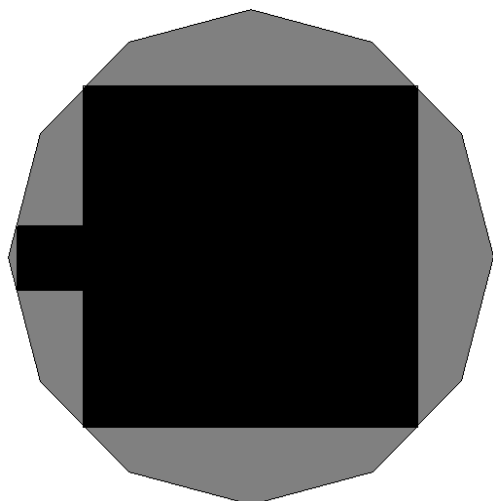
### 1) Pavage aléatoire :

Voici l'algorithme d'un pavage simple à réaliser mais non optimal. Il a l'inconvénient de devenir de plus en plus lent à mesure que l'on remplit la figure.



Cependant, notre algorithme ne pave pas entièrement le polygone pour une raison qui nous échappe.

## 2) Pavage optimal pour fractal polygonal :



Au départ, on prend pour référence le centre de la figure supposé connu. Un programme permet de tester si un point se trouve dans la figure à paver privée du pavage déjà effectué(en noir sur la figure). En parcourant chaque point de la figure non pavée (en gris), nous testons la taille maximale du carré que l'on peut faire en ces points, le carré le plus grand est mis en mémoire.

Lorsque la surface grise est entièrement parcourue, nous savons quel est le carré le plus grand ainsi que sa position. En itérant ceci, nous obtenons un pavage optimal pouvant s'adapter à tous les polynômes.

Cette méthode, très coûteuse en ressources au début va peu à peu être plus efficace lorsque l'aire de la figure grise diminue. Les figures convexes étant beaucoup moins coûteuses en CPU. Cet algorithme peut être utilisé avec n'importe quel polygone régulier.

Le codage de cet algorithme nous à posé des problèmes pour la seconde itération, où le programme nous donne des résultats aberrants.

Il se trouve dans le dossier

## 3) Autres problèmes rencontrés :

Nous avons essayé de programmer une interface graphique(dans le dossier interface) codée avec GTK. Présentée sous forme d'un assistant, plus convivial. Des problèmes de compilation ont été rencontrés sous ubuntu.

Les tutoriaux nous ont été d'une aide précieuse pour maîtriser la SDL ou le GTK, cependant, ils sont souvent incomplets ou imprécis.

### **III – Les différents options :**

La majorité des options ont été seulement utilisées avec la fonction Koch

#### 1) Le zoom :

Le zoom, utilisé pour le flocon de koch, zoom autour de la position (0, 0). Il s'utilise en utilisant la touche + ou -. Pour le programmer, les coordonnées du triangle de koch d'origine ont été multipliées ou divisées par 1,25 à chaque usage de la touche + ou -. Une erreur s'affiche lorsque le zoom est trop important.

#### 2) Le déplacement :

Il consiste simplement à translater tous les points de la figure; il s'utilise avec les fleches directionnelles. Il peut être utile afin d'empêcher le déplacement avec le zoom.

#### 3) Le remplissage des polygones :

La détection des points dans le polygone permet de le remplir celui ci avec des couleurs.

Freinés par des difficultés d'ordre de programmation davantage que l'algorithmique en tant que tel. Ce projet enrichissant nous à également permis de porter un regard critique sur notre façon de programmer et d'optimiser nos algorithmes