

# Création des isolignes avec Fudaa-Prepro

## 1. Objectif

Pouvoir calculer et afficher des isolignes à partir des résultats issus des systèmes Reflux, Rubar ou Télémac.

## 2. Problèmes

1. Les maillages sont de types différents ( triangles avec 3 ou 6 noeuds et quadrilatères)
2. Les données peuvent être définies sur les noeuds ou sur les éléments
3. Il faut prendre en compte l'erreur numérique (représentation des réels)

## 3. Procédés utilisés

Pour répondre aux points 1) et 2), le problème est systématiquement ramené à un maillage triangulaire ( 3 noeuds) et avec des données définies sur les noeuds.

Pour l'erreur numérique, une marge est utilisée pour tester l'égalité de 2 réels.

Les 2 chapitres suivants décrivent comment les données initiales sont transformées pour obtenir finalement un maillage triangulaire et des données définies sur les noeuds.

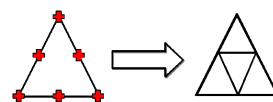
Le chapitre 6 présente le procédé utilisé pour calculer les isolignes sur le maillage triangulaire.

## 4. Cas simple: données définies sur les noeuds

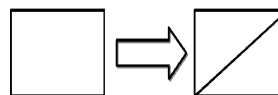
### Le maillage

Dans ce cas, **il s'agit de transformer les quadrilatères et les triangles T6 (6 noeuds dont 3 noeuds milieux) en triangle simple.**

Pour les éléments T6, nous les découpons en 4 triangles:



Pour les quadrilatères, 2 triangles sont créés



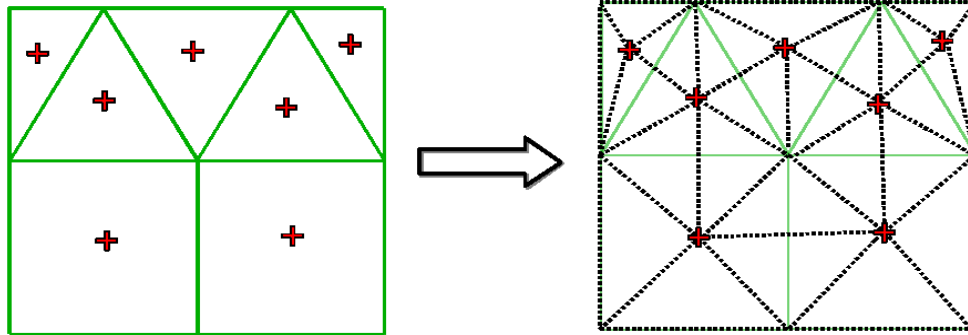
### Les données

Etant donné que nous ne modifions pas les noeuds, les données initiales ne sont pas modifiées.

## 5. Cas compliqué: données définies sur les éléments

### Le maillage

Ce cas nous impose de remailler le maillage initiale et d'ajouter des noeuds au milieu des éléments.

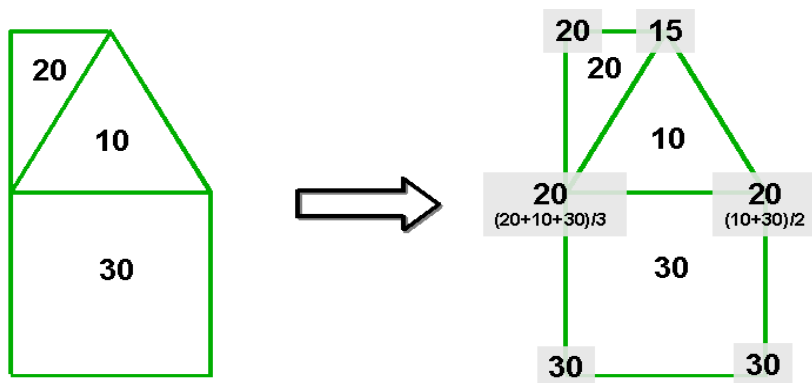


Le nouveau maillage connecte les noeuds milieux afin de construire des isolignes qui respectent au mieux l'organisation initiale.

## Les données

Les données initiales sont définies sur les éléments. Ce seront les données utilisées pour les nouveaux noeuds ajoutés aux centres des mailles.

Nous aurons également besoin des valeurs sur les anciens noeuds du maillage: ils seront déterminés en calculant la moyenne sur les éléments adjacents.



## 6. Déterminer les isolignes à partir d'un maillage triangulaire

Une marge d'erreur est autorisée ( erreur numérique), elle sera nommée  $\epsilon$ .

Note: cet algorithme est basique et il pourra être amélioré en terme de performance.

### Etape 1: détecter les éléments plats

L'algorithme parcourt tous les éléments et marque les éléments plats ( à  $\epsilon$  près). Ces éléments seront ignorés par la suite.

### Etape 2: parcourt des éléments à la recherche des isolignes

L'algorithme parcourt tous les éléments restants du maillage. Dès qu'un élément est analysé, il est marqué et ne sera pas retraité.

Pour chaque élément, les tests suivants sont effectués:

1. Les 3 noeuds du triangles sont testés. Si la valeur à un des noeuds est égale ( à  $\epsilon$  près) à la valeur cherchée, il est enregistré.
2. Sinon, on teste les 3 arêtes. Si une arête convient, elle est enregistrée.

Dès qu'un élément intersecte la valeur de l'isoligne, l'algorithme parcourt les éléments voisins

de ce dernier pour construire l'isoligne correspondante:

1. l'intersection a été trouvée sur un noeud: on recherche la suite (ou les suites) de l'isoligne sur tous les éléments adjacents au noeud
2. l'intersection a été trouvée sur une arête: on recherche l'élément adjacent à l'arête et on le teste.

Note: Si l'intersection est trouvée sur un noeud, il se peut que plusieurs branches soient ainsi trouvées. L'algorithme parcourera alors ces différentes branches.

### Etape 3: finalisation et simplification des isolignes

Au cours du parcours, l'algorithme simplifie les isolignes: les points alignés sont concaténés et si une isoligne « simple » (sans branche) s'avère être fermée, elle est considérée comme résultat définitif et est enregistrée.

Pour les autres lignes trouvées, plusieurs algorithmes sont utilisés pour reconstruire des isolignes fermées et des isolignes plus « grandes »:

1. le premier algorithme est un « Polygonizer » qui permet de reconstruire des lignes fermées.
2. Le deuxième est un « LineMerger » qui permet de construire des lignes à partir de segments restants.

Note: dans le cadre d'un jeu de données définies sur les éléments, l'algorithme simplifie également les isolignes si une alternance « noeud ajouté au milieu de l'élément » – « noeud initial » est trouvée. Cela est effectué dans le but d'éviter les dents de scie dans le résultat.

## 7. Codes sources de Fudaa

Voici les références: les algorithmes de fudaa se trouvent dans le package [dodico/ef/operation](#). Les algorithmes « Polygonizer » et « LineMerger » sont issues du projet JTS (projet utilisé par Geotools, OpenJump, ...).

Voici le détail des classes de Fudaa:

- [EfIsoActivity](#) est la classe utilisée pour calculer les isolignes
- L'interface [EfIsoResultInterface](#) et la classe [EfIsoResultDefault](#) permettent de stocker les résultats.
- [EfIsoRestructuredGridActivity](#) permet de remailler le maillage initiale dans le cas où les données sont définies sur les éléments.
- Les tests unitaires sont dans la classe [TestJSearchIso](#).

Pour l'instant, ces classes sont utilisées dans l'application Fudaa-Prepro et plus particulièrement dans le [post-traitement](#). C'est l'action [TrIsoLineAction](#) qui permet de lancer l'assistant [TrIsoLineWizard](#).

L'utilisateur peut choisir les variables, les pas de temps et les valeurs voulues. A partir de cela, Fudaa-Prepro calcule les isolignes et ajoute des calques de données géographiques au projet en cours.

## 8. Améliorations possibles

1. Optimisation de l'algorithme (non prévu)
2. Constuire des surfaces exportables pour les SIG du commerce (en cours: juin 2007)