RAPPORTS

Centre de Prestations d'Ingénierie Informatiques

Département Opérationnel Ouest

Mai 2017

jeux de données SIG

–

vérification et correction des géométries

Ressources, territoires, habitats et logement
Énergie et climat
Développement durable
Prévention des risques
Infrastructures, transports et mer

Présent pour l'avenir

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

Ministère de l'Écologie, de l'Énergie, du Développement durable et de la Mer

Ministère de l'Écologie, de l'Énergie,
du Développement durable et de la Mer
en charge des Technologies vertes et des Négociations sur le climat

www.developpement-durable.gouv.fr

Historique des versions du document

Version	Date	Auteur	Commentaire
1.0	12/05/2017	Alain FERRATON Michel ZEVORT	Première version
3.0	26/06/17	Alain FERRATON	Modifications suite aux premières relectures.

Affaire suivie par

Alain FERRATON – CP2I /DOO **Michel ZEVORT – CP2I - DOO**
Tél. :
Courriel :

Rédacteur

Alain FERRATON - CP2I Département Opérationnel de l'Ouest
Michel ZEVORT – CP2I département Opérationnel de l’Ouest

Relecteur

Jean-Claude Proteau – **SPSSI/PSI1**

Référence(s) internet

Référence(s) documentaires

SOMMAIRE

Avertissement

Ce document traite du thème de la détection et la correction des géométries. Il aborde aussi un peu les problèmes de topologie qui sont souvent associées dans les outils.

Il aborde essentiellement la problématique des polygones.

Bien que rappelant quelques éléments théoriques, il s'agit avant tout d'une synthèse empirique basée sur l'expérience des auteurs et de retours de service sur des cas concrets. Quelques exemples sont analysés en annexes.

Toute remarque visant à le corriger ou le compléter peut-être adressée au PNE Progiciels géomatiques:

assistance-nationale-progiciels-geomatiques@developpement-durable.gouv.fr

Remerciements

Nous ne citerons pas individuellement les personnes qui ont contribué, par leurs apports directs ou indirects, à l'élaboration de ce document, ils sont nombreux et nous aurions peur d'en oublier.

La liste interne au Ministère de la Transition Écologique et Solidaire 'labo-qgis' est d'une très grande richesse… Un grand merci global donc à tous les contributeurs.

Résumé :

Il existe plusieurs standards de validité des géométries pour un jeu de données dans un SIG. Le plus connu est GEOS, c'est celui que doit vérifier au minimum un jeu de données pour être utilisé sans problème sous QGIS ou PostGIS.

Au-delà de la validité au sens d'un standard, un jeu de données doit vérifier des spécifications propres. On distingue des spécifications géométriques (contraintes sur la géométrie de chaque objet), des spécifications topologiques (contraintes entre objets d'une même couche ou entre objets de couches différentes).

Le premier travail d'un administrateur de données (ADL) sur un jeu de données, est une **phase de diagnostic** (vérification). Il existe pour cela plusieurs méthodes disponibles. Nous recommandons pour vérifier la validité géométrique au sens GEOS sous QGIS ou PostGIS d'utiliser la méthode par requête SQL avec la fonction st_isvalidreason(). Openjump constitue une alternative intéressante qui donne les mêmes résultats. Cette phase d'analyse doit également être (re)exécutée après une conversion de format qui peut introduire des anomalies; Un exemple d'export de données initialement au format MapInfo est donné au chapitre 3.4. Un script d'analyse d'un patrimoine de données avec ogr2ogr est fourni en exemple. Dans la phase de diagnostic l'ADL peut également être amené à vérifier la conformité aux spécifications géométriques et topologiques. Plusieurs outils sous QGIS, ou Openjump sont présentés.

Une deuxième phase doit être, au minimum, de **rendre valide le jeu de données** au sens de GEOS. Nous recommandons d'utiliser sous QGIS ou PostGIS la méthode par requête SQL avec la fonction st_makevalid(). Il peut être intéressant d'en profiter pour supprimer les sommets en double avec la fonction st_simplify(). La partie de transformation des géométries de la requête SQL sera alors de la forme (exemple une correction de polygones sous PostGIS) :

```
st_multi(st_simplify(ST_Multi(ST_CollectionExtract(ST_ForceCollection(ST_MakeValid(geom)),3)),0))
```

La méthode avec st_buffer(geom, 0) est une alternative, si la méthode st_makevalid() échoue (gros jeu de données), mais elle ne doit être utilisée que si on a déjà corrigé les polygones en 'papillons'. Cette méthode peut-être automatisée avec le modeleur de traitement de QGIS, ou dans la console Osgeo4W et éventuellement exécutée sur un lot de données.

Une méthode dite de 'rastérisation / polygonisation' est présentée dans le cas de gros jeu de données obtenu par une méthode automatique de vectorisation d'un raster (cas des PPR par exemple). Il faut vérifier après correction que le jeu de données obtenu reste cohérent avec le jeu de données initial.

Aller plus loin et corriger les autres erreurs de géométries, ou les erreurs de topologie, nécessite de bien connaître les spécifications et d'avoir accès à une possibilité de vérification. Une telle phase doit être envisagée avec la plus grande prudence par un ADL. Dans un certains nombre de domaines, en particulier pour les zonages réglementaires comme les PLU ou PPRN les avis divergent ; a priori devrait être entrepris seulement des corrections automatiques qui ne changent pas les surfaces (ex : suppression d'arcs pendants ou de sommets en doubles), toute correction qui change les surfaces (ex : suppression de micro-zones) doit être validé par le producteur

métier. Plusieurs outils de correction sont présentées. Le plugin ‘vérifier les géométries’ de QGIS dispose de fonctionnalités intéressantes, mais est peu fiable dans la version examinée (QGIS 2.16). Openjump et GRASS proposent des outils qui sont à manier avec le recul nécessaire.

1 - POURQUOI VERIFIER OU CORRIGER ?

1.1 - Erreurs entraînant des dysfonctionnements

Chaque logiciel utilise une modélisation de la géométrie qui lui est propre ou qui respecte un standard. Le modèle le plus usité est celui de l'OGC décrit au chapitre 2.

Les logiciels sont plus ou moins tolérants aux 'erreurs' de géométrie. Certaines erreurs peuvent entraîner des résultats faux lors de l'exécution de fonction du logiciel (requêtes, algorithmes,...), il est alors parfois difficile de s'en rendre compte (ex : calcul de surface inexacte). D'autres erreurs peuvent entraîner des dysfonctionnements plus visibles; messages d'erreurs ou plantage du logiciel.

D'une façon générale, les requêtes et opérations spatiales sous QGIS (logiciel recommandé) ne sont possibles qu'avec des objets dont la géométrie est valide, au moins avec la méthode GEOS.

La *validité* (concept que nous définirons plus loin) des géométries est donc nécessaire pour réaliser des requêtes fiables. Il est fortement recommandé pour les Administrateurs de Données (ADL) de vérifier et corriger (ou faire corriger en cas de sous-traitance) le patrimoine mis à disposition des utilisateurs.

Différentes méthodes de correction de géométries sont présentés au chapitre 5. Elles ne peuvent pas toujours être automatisées et dans ce cas le travail de correction peut être très lourd.

Exemple de résultat faux:

ce polygone en papillon ;

qui est incorrect au sens de GEOS, a une surface de 0 m2 (car une des sous-surfaces est comptée en négatif).

Exemple de plantage suite à géométrie invalide :

Certaines fonctions comme st_union (ou dissolve) peuvent ne pas aboutir en cas de géométrie invalide (mouline sans fin ou message d'erreur du type 'TopologyException').

A noter, qu'à partir de QGIS 3.0 le comportement par défaut est un arrêt des algorithmes en cas de rencontre d'une géométrie invalide. Une option de réparation automatique (makevalid) sera probablement ajoutée.

1.2 - Conformité aux spécifications

La réalisation d'un lot de données obéit à un cahier des charges qui décrit, en particulier, les spécifications de saisie. On pourra sur ce point se reporter au module 6 – *spécification de saisie,* de la formation 'Concevoir et structurer une base de données géographique' (CSBDG).

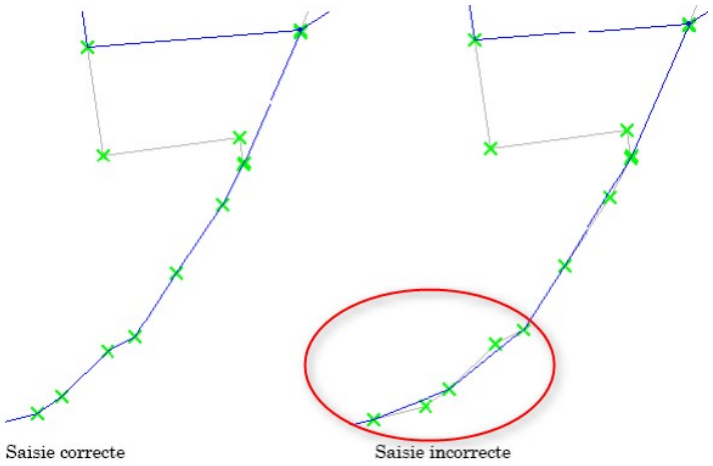
Les spécifications de saisie décrivent des règles à respecter pour la géométrie, ainsi que des règles topologiques (par exemple : ni lacune, ni recouvrement).

Les géostandards de la COVADIS décrivent de telles règles.

Exemple : Géostandard des PLU

Règles de saisie de la classe d'objets <ZoneUrba> :

- Cohérence par rapport au référentiel de saisie : La numérisation doit se faire en partage de géométrie entre le référentiel cadastral et le PLU/POS numérisé. Les limites du zonage du PLU/POS doivent correspondre parfaitement avec celles du parcellaire cadastral lorsqu'elles sont identiques. Cette précision doit permettre d'effectuer un calcul d'intersection des surfaces pour déterminer dans quelle zone se trouve une parcelle.



- Cohérence topologique entre objets de la classe <ZoneUrba> : Il s'agit *d'une partition totale du territoire* : pas de trou, pas de recouvrement, pas de lacune... Les polygones de cette classe doivent par conséquent respecter la topologie d'un graphe planaire à savoir:
  - Le contour d'un objet <ZoneUrba> est un polygone obligatoirement fermé ou plusieurs polygones obligatoirement fermés
  - Les superpositions ou les lacunes entre deux objets de la classe <ZoneUrba> sont proscrites (les objets voisins sont saisis en partage de géométrie)
  - Les polygones ne présentent pas d'auto-intersection
  - Les polygones ne présentent pas d'arcs pendants
  - Les polygones formant des îlots évident le polygone englobant
- <...>

Il est important de vérifier que le livrable respecte bien les spécifications de saisie. C’est la phase de réception. Les vérifications peuvent parfois dépendre de l’outil utilisé, en conséquence, il peut être utile d’indiquer au prestataire la méthodologie qui sera utilisée pour la réception du lot de données.

2 - THEORIE et DEFINITIONS

2.1 - Définitions

Il existe malheureusement plusieurs définitions, mais on définit le plus souvent un objet valide par ce qu'il ne doit pas être … Autrement dit :

Objet Valide = objet non invalide ou sans erreurs de géométrie

On distingue les erreurs de géométrie et les erreurs de topologie.

Les erreurs de géométrie :

Il faut encore distinguer les erreurs de *géométrie invalide* pour un modèle donné et les erreurs par rapport aux *spécifications* complémentaires de saisie.

Les géométries invalides entraînent des erreurs voir des blocages dans les calculs. Par exemple, comme nous l'avons vu, une auto-intersection génère une surface positive et une autre négative, la surface totale du polygone sera finalement minorée. La validité est surtout fondamentale pour les polygones qui définissent des surfaces (polygones) et requièrent une bonne structuration.

Certaines des règles de validation des polygones semblent évidentes, et d'autres semblent arbitraires (et le sont vraiment) :

- Les contours des polygones doivent être fermés.
- Les contours qui définissent des trous doivent être inclus dans la zone définie par le contour extérieur.
- Les contours ne doivent pas s'intersecter (ils ne doivent ni se croiser ni se toucher).
- Les contours ne doivent pas toucher les autres contours, sauf en un point unique.

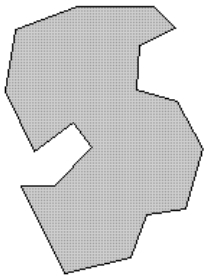
Les deux dernières règles font partie de la catégorie arbitraire.

Les erreurs de topologie :

La topologie d'une table gère les relations entre les objets (connectivité d'un réseau routier, lacunes ou recouvrements entre objets …) au sein d'une même table. On pourra consulter cette introduction à la topologie sous QGIS.
Les erreurs de topologies peuvent générer des résultats erronés lorsque la topologie ne correspond pas à la réalité. Cependant elles ne bloquent généralement pas les calculs.

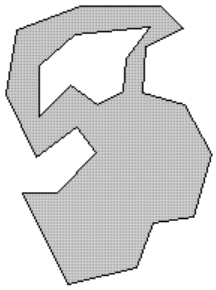
Les différents type de polygones:

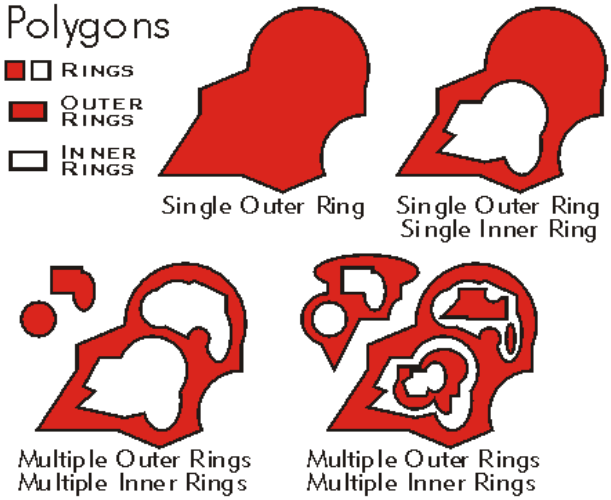
Un polygone est formé par une polyligne fermée qui délimite l'extérieur et l'intérieur. L'intérieur est la surface du polygone.



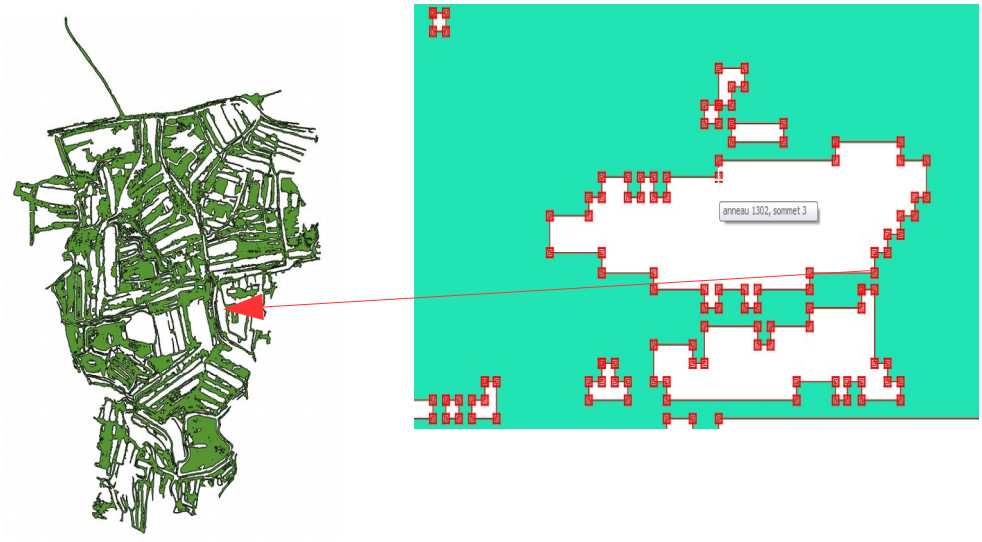
Polygone « simple » : Il s'agit d'un polygone sans trou avec une seule partie définie par une polyligne fermée.
On peut y inclure les multi polygones sans trou qui sont l'assemblage de plusieurs polygones « simples ».

Avec trous : les polygones peuvent avoir un ou plusieurs trous. La limite / frontière entre le trou et l'intérieur est constitué d'une polyligne appelée anneau intérieur (Interior Ring ou Inner Ring) . La frontière extérieure est l'anneau extérieur (Exterior Ring ou Outer Ring) .





Un polygone peut avoir avec plusieurs trous (plus de mille ci-dessous)



2.2 - Les normes et standards

En la matière, plusieurs normes existent. La plus utilisée est la norme OGC, mais Esri utilise aussi sa propre référence de validité. Nous verrons que QGIS propose également une méthode 'QGIS' qui est présente pour des raisons historiques.

2.2.1 - OGC/GEOS

JTS et GEOS sont des bibliothèques utilisables par les logiciels SIG dont le cahier des charges est édicté par l'OGC « **99-049_OpenGIS_Simple_Features_Specification_For_SQL_Rev_1.1-1.pdf ».**
GEOS est un sous-ensemble de la JTS (JavaTopology Suite). La JTS est écrite en Java, GEOS en C/C++ pour être utilisable par les outils tels que Postgis, Qgis,..

D'après le trac OSGEO une géométrie valide doit vérifier les conditions :

- Les anneaux des polygones ne doivent pas se toucher (un polygone de type « papillon intérieur» devrait être réécrit comme un "polygone avec un trou qui touche en un seul point", un polygone en 8 doit être réécrit en MultiPolygone et si une des parties est très petite, elle peut être supprimée).
- Les anneaux ne peuvent pas avoir de surface nulle, les polygones non plus
- Les anneaux doivent être correctement imbriqués et ne se toucher qu'en un seul point. (Les polygones avec des anneaux qui se touchent le long d'un segment doivent avoir la couronne intérieure et le couloir de largeur zéro supprimés)
- Les nœuds ne doivent pas être dupliqués (jusqu'à une tolérance)
- Il ne doit pas y avoir de pointes externes ou internes.
- Les parties des multiPolygones ne doivent pas se toucher.
- Les anneaux ne doivent pas se croiser.

PostGIS par exemple est conforme au standard OGC OpenGIS.

Les entités géométriques des bases PostGIS doivent ainsi être à la fois **simples** et **valides**. Il existe deux fonctions sous PostGIS ; *St_IsSimple()* permet de vérifier si une géométrie est simple, *St_IsValid()* permet de vérifier si une géométrie est valide.

Par exemple, calculer la surface d'un polygone comportant un trou à l'extérieur ou construire un polygone à partir d'une limite non simple n'a pas de sens.

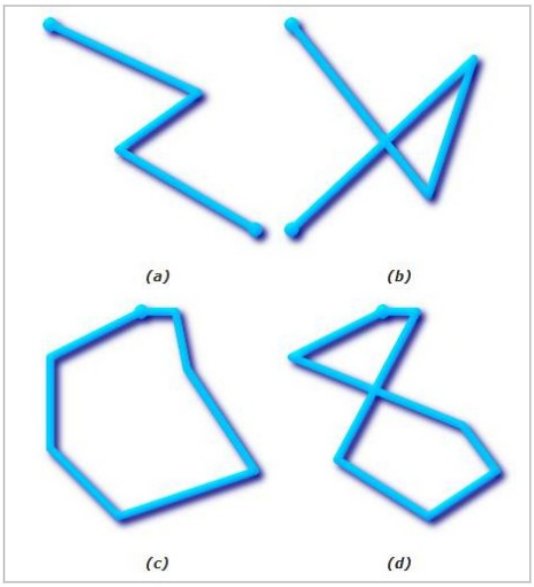
Selon les spécifications de l'OGC, une géométrie **simple** est une géométrie qui ne comporte pas de points géométriques anormaux, comme des auto-intersections ou des auto-tangences, ce qui concerne essentiellement les points, les multi-points, les polylignes et les multi-polylignes.

La notion de géométrie **valide** concerne principalement les polygones et les multi-polygones et le standard définit les caractéristiques d'un polygone valide.

Un point est par nature simple, ayant une dimension égale à 0.

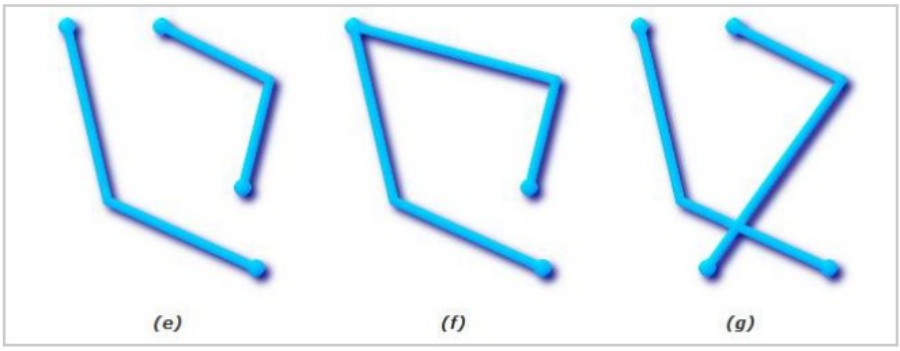
Un objet multi-points est simple si tous les points le composant sont distincts.

Une polyligne est simple si elle ne se recroise pas (les extrémités peuvent être confondues, auquel cas c'est un anneau et la polyligne est fermée).



Les polylignes (a) et (c) sont simples, mais pas les polylignes (b) et (d)

Une multi-polyligne est simple si toutes les polylignes la composant sont elles-mêmes simples et si les intersections existant entre 2 polylignes se situent à une extrémité de ces éléments :



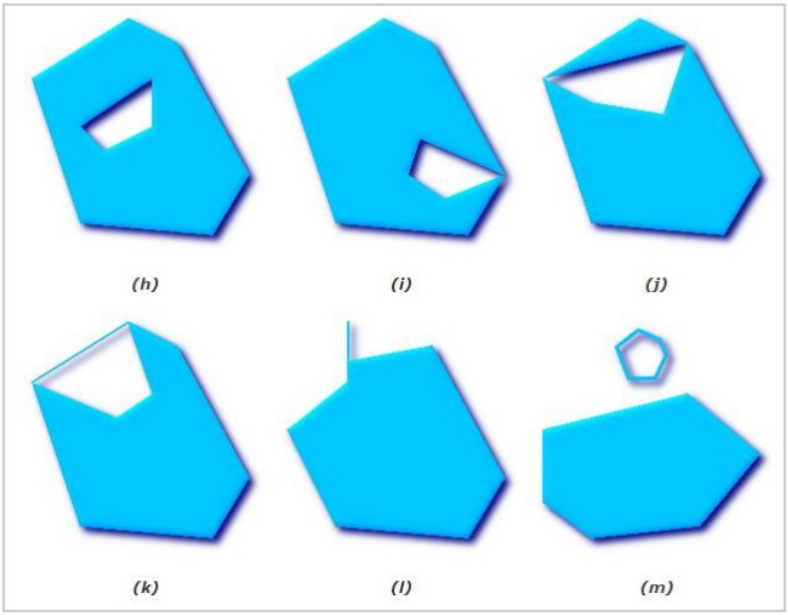
(e) et (f) sont des multipolylignes simples, mais pas (g)

Les limites d'un polygone peuvent être constituées par un unique anneau extérieur (polygone plein) ou par un anneau extérieur et un ou plusieurs anneaux intérieurs (polygone à trous). Un polygone est valide s'il ne comporte pas d'anneaux se croisant.

Un anneau peut intersecter la limite mais seulement en un point (pas le long d'un segment).

Un polygone ne doit pas comporter de lignes interrompues (les limites doivent être continues) ou de point de rebroussement (pic).

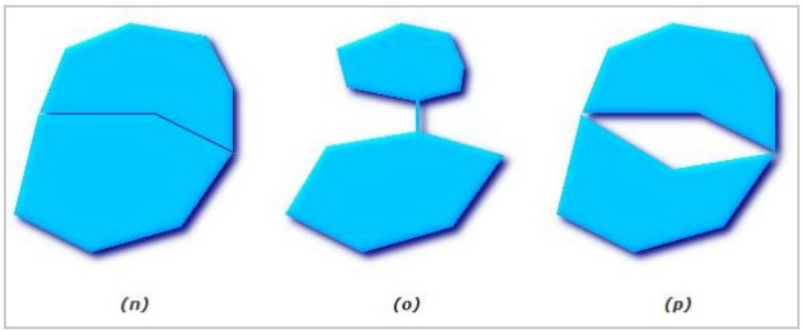
Les anneaux intérieurs doivent être entièrement contenus dans la limite extérieure du polygone.



(h) et (i) sont des polygones valides, (j), (k), (l), (m) sont des polygones ni simples ni valides mais (j) et (m) sont des multi-polygones valides

Un multi-polygone est valide si et seulement si tous les polygones le composant sont valides et si aucun intérieur d'un polygone ne croise celui d'un autre.

Les limites de 2 polygones peuvent se toucher, mais seulement par un nombre fini de points (pas par une ligne)



(n) et (o) ne sont pas des multi-polygones valides, par contre (p) est valide

nb : Par défaut, PostGIS n'applique pas le test de validité géométrique lors de l'import d'entités géométriques, parce que le test de validité géométrique consomme beaucoup de temps processeur pour les géométries complexes, notamment les polygones.

On peut mettre en œuvre des méthodes pour vérifier la validité de la géométrie des entités, soit a priori avec différents outils que nous verrons, soit a posteriori avec les méthodes de PostGIS (st_isvalid(),...).

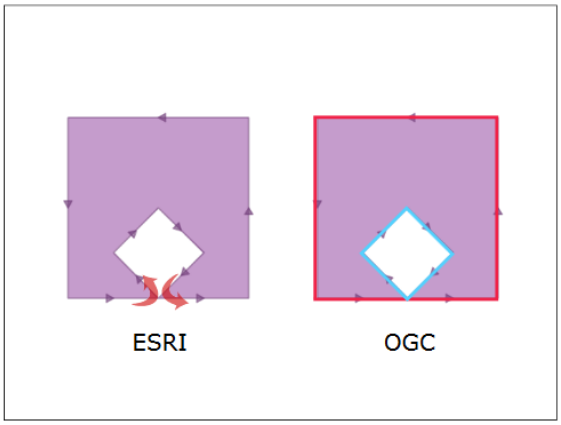
2.2.2 - ESRI

Un polygone valide n'a pas d'anneaux chevauchants, pas d'auto-intersections sauf éventuellement aux sommets, pas de segments pendants et, en général, un point arbitraire peut toujours être classé sans ambiguïté soit à l'extérieur, soit à l'intérieur, soit à la limite du polygone. Un polygone valide est dit simple. Un polygone simple (au sens ESRI) peut ne pas être compatible OGC.

Pour plus de précision : http://desktop.arcgis.com/fr/arcmap/10.3/manage-data/using-sql-with-gdbs/geometry-validation.htm

La commande *ST_Isvalid()* de PostGIS comporte un flag (paramètre optionnel) permettant d'indiquer que l'on considère les anneaux auto-intersectant formant des trous comme valides (ce qui n'est pas le cas au sens du modèle OGC). Les deux façons constituent chacune un standard distinct (modèle OGC et modèle ESRI).

Voir par exemple, page 15 du document http://2010.foss4g.org/presentations/3369.pdf



This polygon is invalid because it consists of just one ring that loops around and touches itself at the bottom. I call it a "banana polygon" because it is like a banana that has been bent until the ends touch.
The correct way to construct this shape is with an exterior and an interior ring that touch at one point.
There is no "right" way to do this. ESRI actually considers the first case valid and the second one invalid. They aren't wrong, their internal standard is just different.

On peut reconstruire un polygone valide au sens ESRI en un polygone valide au sens GEOS en utilisant une solution type *st_buffer(geom, 0.0)* que nous détaillerons plus loin.

2.2.3 - Qgis

2.2.3.a - L'algorithme 'Vérifier la validité' (check validity)

QGIS propose deux méthodes de validation : soit GEOS, soit QGIS. Une méthode peut-être imposée pour vérification interactive lors de la création de nouvelle géométrie (Préférences → onglet numérisation).

La documentation de QGIS indique que GEOS est plus rapide mais n'indique que la première erreur rencontrée pour chaque objet.

La méthode GEOS renvoie un fichier 'Sortie invalide' qui est une couche de polygones complétée par une colonne une colonne *_errors* :

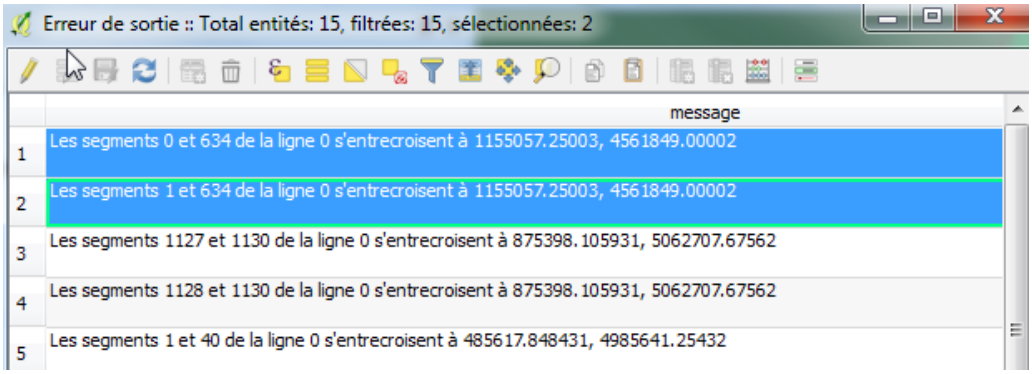
_errors
Erreur GEOS : Duplicate Rings
Erreur GEOS : Duplicate Rings
Erreur GEOS : Duplicate Rings
Erreur GEOS : Interior is disconnected
Erreur GEOS : Self-intersection
Erreur GEOS : Self-intersection

Elle renvoie également un fichier de points '*erreur de sortie*' localisant les erreurs.

La méthode Qgis peut renvoyer plusieurs erreurs par objet. Cette méthode renvoie dans le fichier '*Sortie invalide*' une couche de polygones complétée par une colonne *_errors* avec un message moins normalisé que par la méthode GEOS.

_errors
Les segments 5 et 20 de la ligne 0 s'entrecroisent à 196397.283, 6776906.7204 Les segments 6 et 20 de la ligne 0 s'entrecroisent à 196397.283, 6776906.7204
Le polygone 1 est à l'intérieur du polygone 0
Les segments 11 et 13 de la ligne 0 s'entrecroisent à 196216.099255, 6832808.34458
Les segments 161 et 163 de la ligne 0 s'entrecroisent à 145895.9958, 6802554.8754
Les segments 36 et 42 de la ligne 0 s'entrecroisent à 140126.8098, 6801062.6388 Les segments 37 et 42 de la ligne 0 s'entrecroisent à 140126.8098, 6801062.6388
Les segments 74 et 76 de la ligne 0 s'entrecroisent à 173046.610472, 6845632.85293 Les segments 74 et 77 de la ligne 0 s'entrecroisent à 173051.528774, 6845633.17517
Le polygone 1 est à l'intérieur du polygone 0

Elle renvoie également le fichier de points '*erreur de sortie*' localisant les erreurs dans lequel on ne retrouve que le message d'erreur moins normalisé, mais avec des duplications pour une même erreur. Exemple :

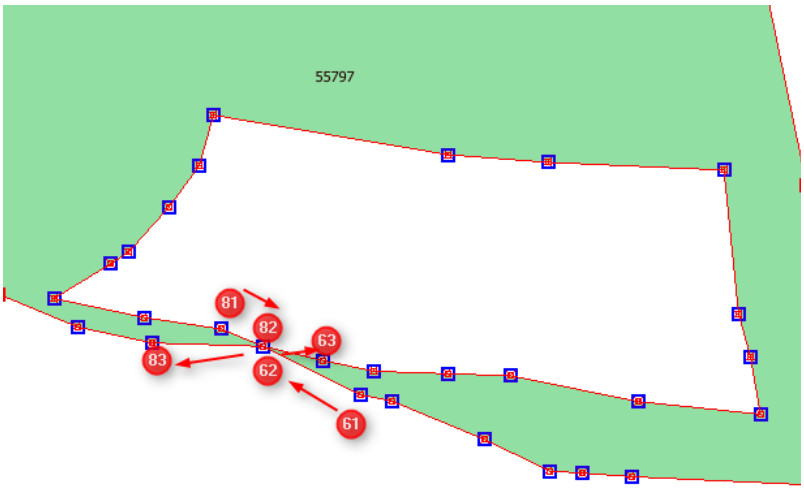


Les spécifications de la méthode QGIS ne sont pas documentées à ce jour (documentation QGIS 2.14). Jürgen Fischer, l'auteur de l'implémentation nous a indiqué qu'il l'avait implémenté pour l'aide à la numérisation (voir l'aide de QGIS), à partir d'un code réalisé pour une autre application sans se préoccuper à l'époque de la compatibilité avec les spécifications GEOS. On peut donc conclure que cette méthode est présente pour des raisons historiques, mais ne devrait pas être utilisée.

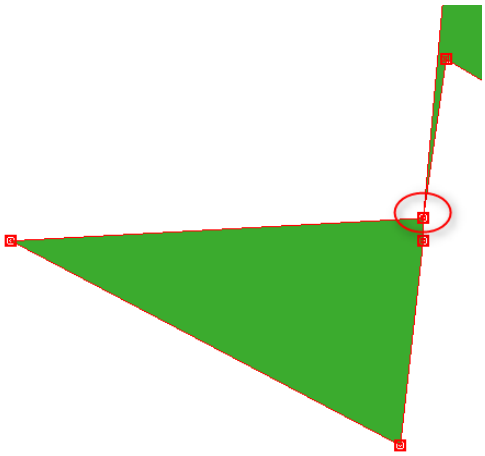
La méthode QGIS peut servir de complément au diagnostic en cas d'erreur complexe, elle détecte également les nœuds en doubles, mais comme ce n'est pas une méthode normalisée, ce n'est pas celle à utiliser en priorité.

Nous avons également pu constater que l'implémentation de la méthode GEOS n'est pas complète (QGIS 2.16) puisqu'elle ne détecte pas les auto-intersections (point double) avec anneau interne (Ring self- intersection).

Exemple (couche tempo.shp) :



Par contre elle détecte bien une 'île' relié par un point comme une erreur de type 'Ring self intersection' (Interior is disconnected).



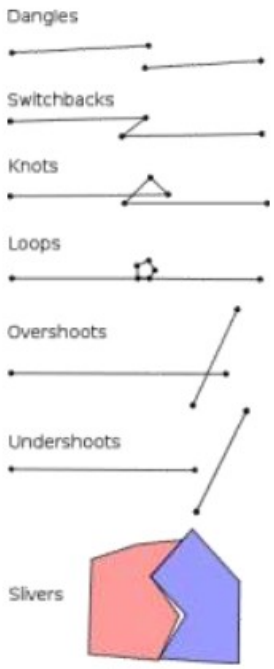
3 - ORIGINES DES PROBLEMES

Les erreurs de géométries peuvent avoir différentes origines ou causes, on peut par exemple évoquer ;

3.1 - Les erreurs humaines

Pour minimiser les erreurs humaines de saisie, le cahier des charges de numérisation, outre les spécifications de saisie, doit décrire une méthodologie de saisie et décrire les vérifications qui sont exécutées au moment de la recette.

Voici quelques exemples d'erreurs :



Dangles : Noeuds pendants

switchback : retours arrières

Knots : nœuds (enchevêtrements)

Loops : boucles

Overshoots : dépassement (de la cible)

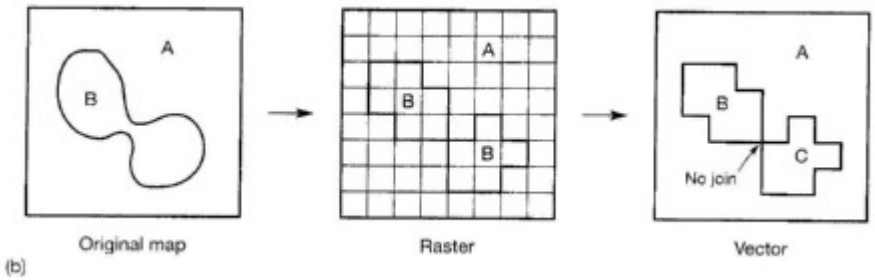
undershoot : non accrochage (à la cible) en dessous d'un seuil de tolérance

sliver : micro polygones

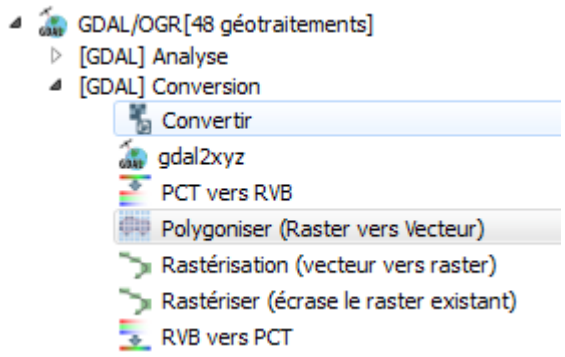
Certaines erreurs (comme la duplication de nœuds) peuvent ne pas générer de problèmes de calcul, mais elles vont le plus souvent les ralentir inutilement.

3.2 - Numérisation automatique

Les méthodes de numérisation automatiques peuvent conduire à des erreurs :



Par exemple, des problèmes de résolutions lors des conversions raster ↔ vecteurs. Ainsi il semblerait que parfois l'algorithme gdal_polygonize



disponible sous QGIS puisse générer des géométries invalides.

La numérisation des aléas PPR est souvent automatisée avec des logiciels propres aux calculs d'hydrologie (ppri) ou d'effets (pprt) dont la fonction première n'est pas d'avoir des géométries valides, mais plutôt d'avoir un état lisible des aléas (avec des superpositions, des auto-intersections…). Les calculs géométriques sur ces couches ainsi que la Covadisation peuvent être compliquées (voir la solution de rasterisation étudiée plus loin).

3.3 - opérations de traitements géométriques

Les objets peuvent être déduits par traitement géométrique tels que les tampons, intersections ou unions. Le résultat peut générer comporter des incohérences géométriques plus ou moins importantes

L'union de géométries complexes mêmes valides génèrent parfois des erreurs comme des polygones à l'intérieur d'autres polygones

ValidateGeometry()erreur: une ou plusieurs entités ont une géométrie invalide: Le polygone 50 est à l'intérieur du polygone 2
ValidateGeometry()erreur: une ou plusieurs entités ont une géométrie invalide: Le polygone 52 est à l'intérieur du polygone 2
ValidateGeometry()erreur: une ou plusieurs entités ont une géométrie invalide: Le polygone 85 est à l'intérieur du polygone 2
ValidateGeometry()erreur: une ou plusieurs entités ont une géométrie invalide: Le polygone 88 est à l'intérieur du polygone 2
ValidateGeometry()erreur: une ou plusieurs entités ont une géométrie invalide: La géométrie a 4 erreurs.
ValidateGeometry()erreur: une ou plusieurs entités ont une géométrie invalide: Le polygone 2 est à l'intérieur du polygone 4
ValidateGeometry()erreur: une ou plusieurs entités ont une géométrie invalide: Le polygone 7 est à l'intérieur du polygone 4
ValidateGeometry()erreur: une ou plusieurs entités ont une géométrie invalide: Le polygone 26 est à l'intérieur du polygone 4
ValidateGeometry()erreur: une ou plusieurs entités ont une géométrie invalide: Le polygone 31 est à l'intérieur du polygone 4
ValidateGeometry()erreur: une ou plusieurs entités ont une géométrie invalide: Le polygone 33 est à l'intérieur du polygone 4
ValidateGeometry()erreur: une ou plusieurs entités ont une géométrie invalide: Le polygone 40 est à l'intérieur du polygone 4
ValidateGeometry()erreur: une ou plusieurs entités ont une géométrie invalide: Le polygone 41 est à l'intérieur du polygone 4

Ce qui est détecté comme incorrect par la méthode QGIS, mais est correct au sens de GEOS.

Nous avons, par exemple, constaté que l'algorithme 'contour lines' de SAGA pouvait générer des géométries invalides (*Invalid : Toxic geometry … too few points*) qui causent des problèmes de disparition d'entité lors de zoom,.... Il faut absolument faire un *makevalid* sur la couche obtenue pour travailler correctement par la suite.

3.4 - les conversions de format

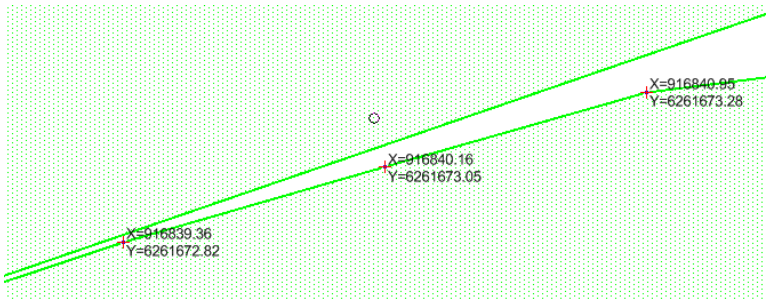
Les conversions de format peuvent aussi être génératrices d'erreurs. En effet chaque format a sa propre définition de la géométrie ainsi que sa propre façon de gérer la précision.

3.4.1 - A partir du format TAB de Mapinfo

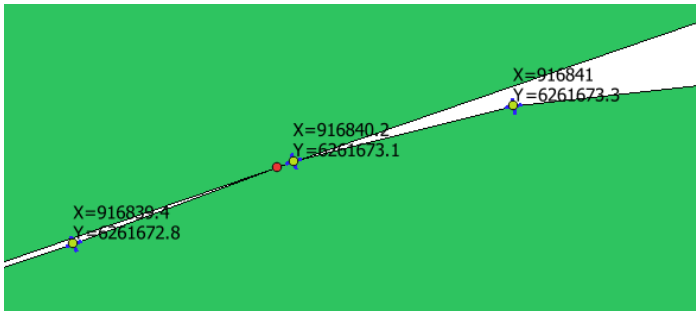
Mapinfo gère les points et donc les nœuds des polygones en les accrochant à une grille. Ainsi, chaque nœud du polygone ne pourra pas avoir une précision supérieure à la grille. Voir par exemple ce document qui reste une référence.

Ceci peut conduire à des artefacts, comme dans l'exemple ci-dessous avec un très fort grossissement sur une zone :

Sous MapInfo il n'y a pas de problème :



Mais sous QGIS on observe que le vérificateur détecte à cet endroit un problème d'entrecroisement de segments :



Une analyse en affichant les coordonnées des sommets impliqués montre qu'il y a un décalage d'arrondi des sommets.

Dans le cas étudié, la couche d'origine était en **Lambert 93 non borné**. Dans ce cas QGIS en ouverture utilise la même précision que lorsqu'on exporte la couche en MIF/MID depuis MapInfo, soit un seul chiffre après la virgule. Ceci suffit à générer des erreurs qui normalement n'existent pas.

La bonne solution dans notre cas a été d'enregistrer la couche sous MapInfo en **Lambert 93 Bornes Europe**. La précision passe alors à 3 chiffres après la virgule.

Une autre solution est d'utiliser le convertisseur de MapInfo (traducteur universel) pour réaliser la conversion au format SHP.

Il est conseillé de procéder à une vérification de la conformité de la géométrie, après une conversion de format.

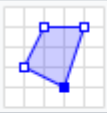
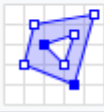
4 - RECHERCHE D'ERREURS

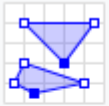
4.1 - visualisation / édition en WKT

Lorsqu'on s'intéresse aux erreurs de géométrie, il peut être utile de transformer la géométrie dans un format compréhensible par les humains, pour cela il existe le format Well Know Text ou WKT (texte bien lisible).

Cette technique n'est cependant utile que dans un objectif de compréhension profonde des anomalies, elle n'est pas utile pour des personnes recherchant une méthode rapide de détection des anomalies, qui peuvent donc ne pas lire ce paragraphe.

Nous nous intéresserons, pour les exemples, qu'aux POLYGON et MULTIPOLYGON. Les tableaux ci-dessous donnent des exemples :

Polygon		POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))
		POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10), (20 30, 35 35, 30 20, 20 30))

MultiPolygon		MULTIPOLYGON (((30 20, 45 40, 10 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))
		MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)), ((20 35, 10 30, 10 10, 30 5, 45 20, 20 35), (30 20, 20 15, 20 25, 30 20)))

Le plugin **Plain Geometry Editor** permet de montrer et d'éditer directement une géométrie sous QGIS en WKT.

Pour les MULTIPOLYGON la syntaxe est celle_ci :

(

((anneau externe polygone1), (anneau interne1 polygone1), (anneau interne2 polygone 1), …), ((anneau externe polygone2) (anneau interne polygone2) …)

)

Ainsi

MultiPolygon (

(

(465030 6700000, 465040 6700000, 465040 6700010, 465030 6700010, 465030 6700000),

(465032 6700002, 465032 6700008, 465038 6700008, 465038 6700002, 465032 6700002),

(465033 6700003, 465033 6700007, 465037 6700007, 465037 6700003, 465033 6700003)

)

soit un polygone composé de 3 anneaux,

Donne un polygone invalide avec trous imbriqués (la représentation sous QGIS peut prêter à confusion avec une coloration du trou central)

Alors que

MultiPolygon (

(

(465030 6700000, 465030 6700010, 465040 6700010, 465040 6700000, 465030 6700000),

(465032 6700002, 465038 6700002, 465038 6700008, 465032 6700008, 465032 6700002)

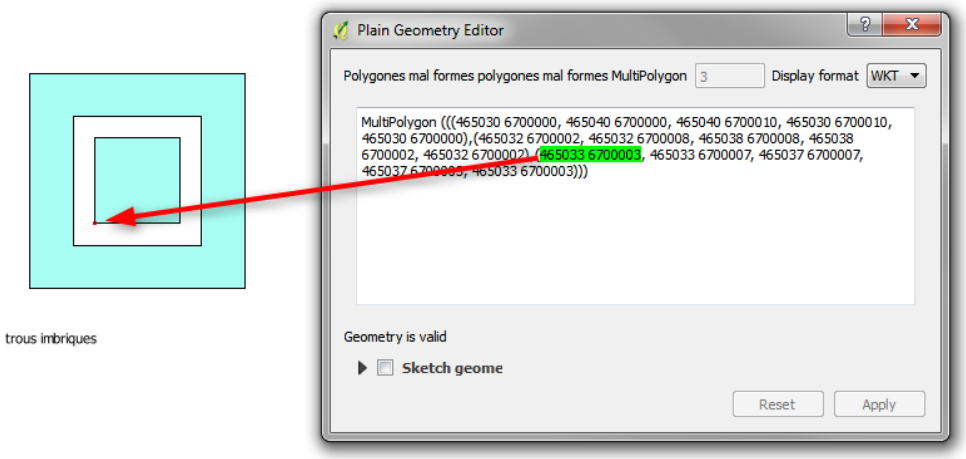
),

((465033 6700003, 465033 6700007, 465037 6700007, 465037 6700003, 465033 6700003))

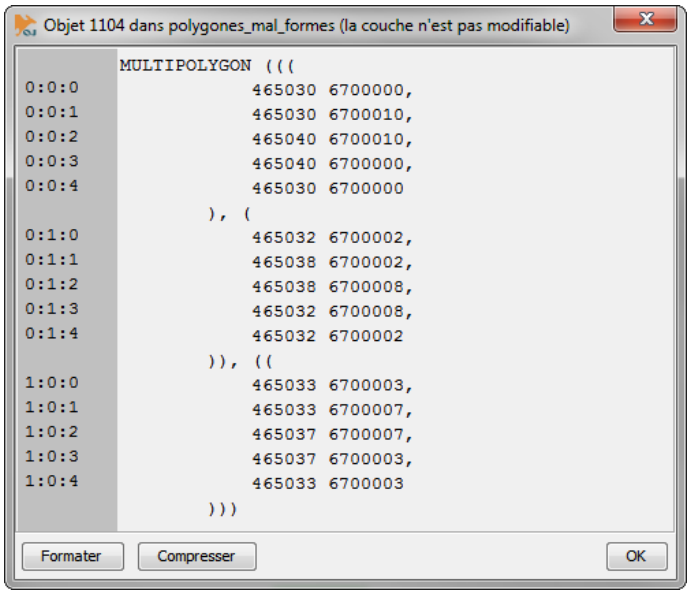
)

Soit un multipolygon composé de 2 polygones dont un avec 2 anneaux.

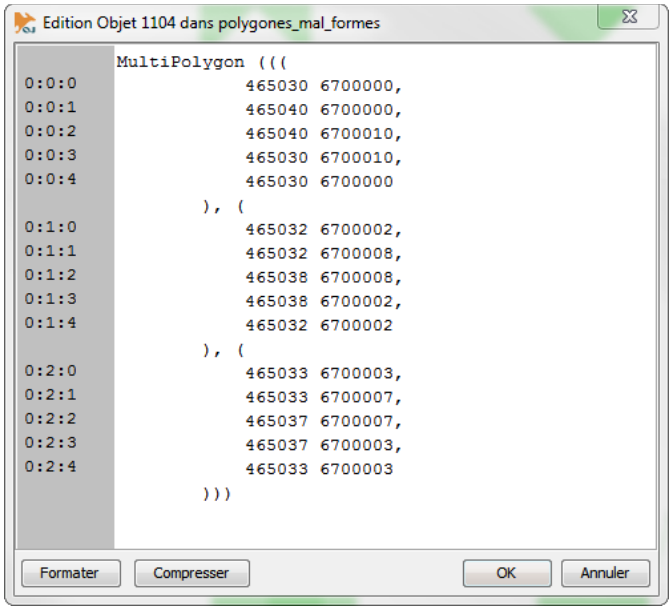
Donne un polygone valide (ce n'est plus un trou au centre mais un polygone rempli). Seul une visualisation en WKT permet de comprendre réellement l'erreur. Le plugin Plain geometry Editor permet de plus de visualiser l'emplacement des sommets.



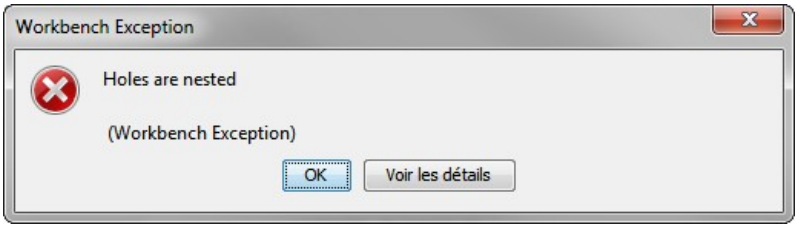
A noter que sous OpenJump (que nous présentons plus loin comme un outil intéressant pour les administrateurs de données) il est très facile de visualiser la géométrie en WKT formatée par clic droit sur l'objet → Visualiser / Modifier la géométrie sélectionnée.



Sous OpenJump une tentative pour générer un polygone décrit comme suit :



génère le message d'erreur :



C'est-à-dire : trous imbriqués.

4.2 - Afficher les coordonnées des sommets

Pour comprendre l'origine des erreurs, il peut être utile d'afficher les coordonnées des sommets concernés.

Sous QGIS, on peut, pour quelques points, créer une nouvelle couche de points et utiliser un accrochage sur les sommets pour numériser les points concernés.

On utilisera ensuite dans **les étiquettes** une expression comme :

'X=' || $x || '\n' || 'Y=' || $y

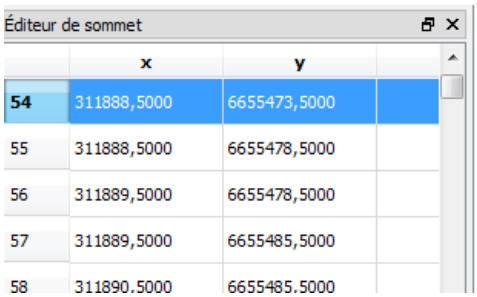
pour afficher les coordonnées.

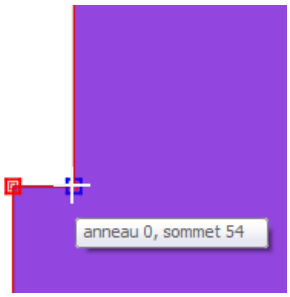
Si on souhaite créer une couche comportant tous les sommets on utilisera

Vecteurs → outils de géométrie → extraction de nœuds.

On peut aussi mettre la couche en modification puis utiliser **l'outil de nœud**  qui affiche '**l 'Éditeur de sommets**'. Un clic sur un sommet, le sélectionne et affiche son numéro et ses coordonnées. Inversement un clic dans le tableau sélectionne le sommet et déplace la carte

si besoin.

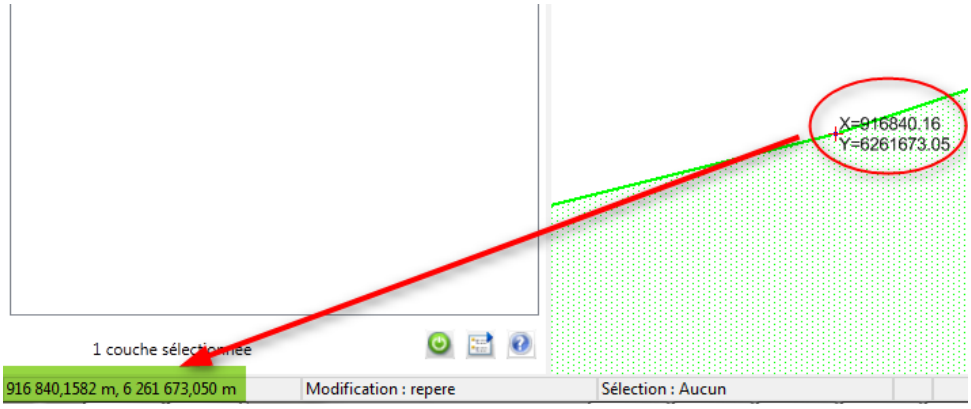




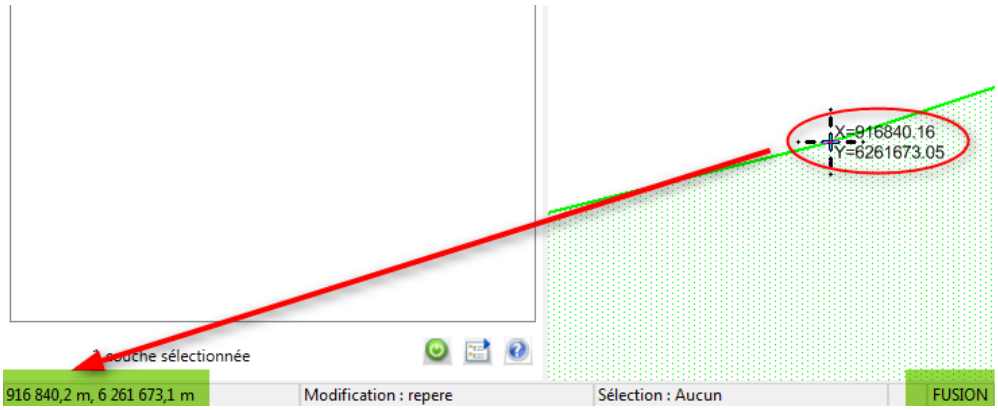
A titre d'information ; **Sous MapInfo,** on peut utiliser l'utilitaire all2pts permettant d'extraire les sommets sous forme d'une couche de points que l'on peut étiqueter avec la formule :

"X="+CentroidX(Object)+Chr$(13)+"Y="+CentroidY(Object)

Il faut cependant noter qu'un doute subsiste sur les coordonnées réelles des points car avec un très fort zoom on voit que les coordonnées étiquetées sont en cohérence avec l'afficheur de coordonnées



mais que ce n'est plus le cas si on utilise le mode Fusion (couche en Lambert 93 non bornées) :



Voir le paragraphe 3.4.1 pour les explications sur le stockage des données géométriques sous mapInfo, tenant compte des bornes de la projection.

4.3 - Recherche d'erreurs sous QGIS

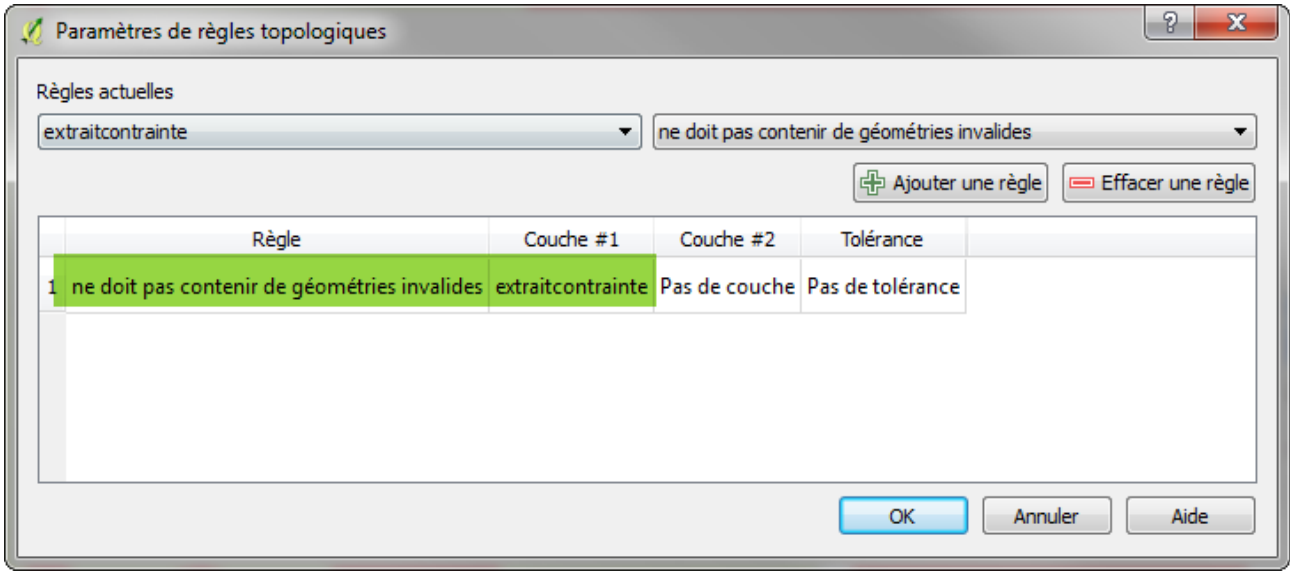
QGIS dispose historiquement de plusieurs solutions pour rechercher les erreurs.

4.3.1 - vérificateur de topologie (topology checker)

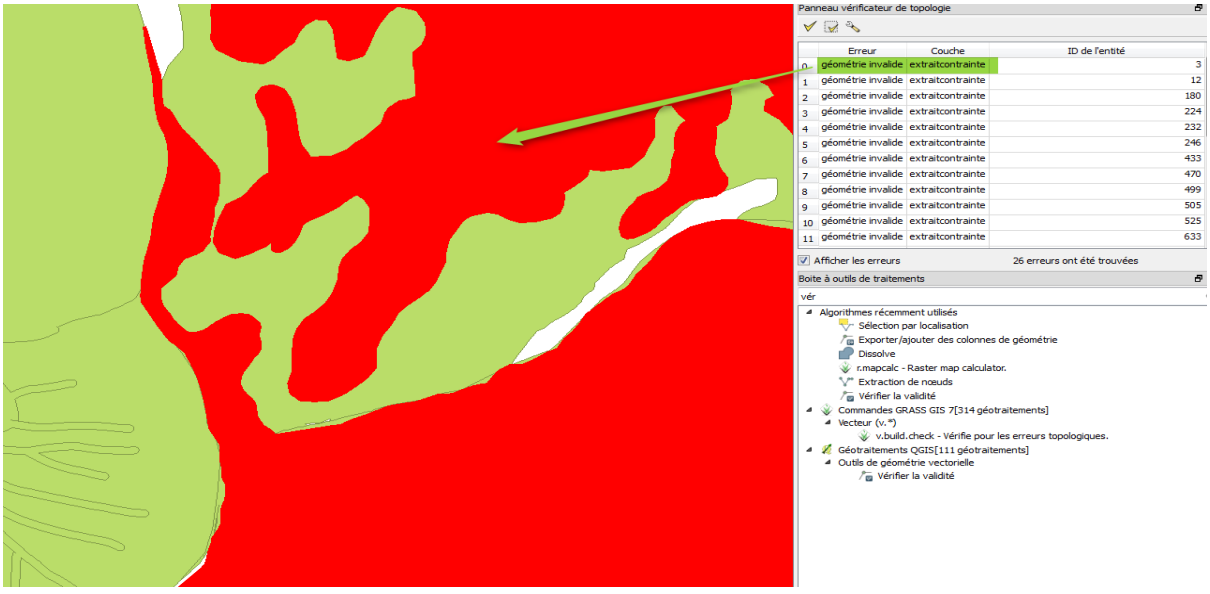
Cet outil permet avant tout de détecter des erreurs de topologie (voir la documentation). Ces possibilités dans ce domaine sont plus larges que d'autres outils que nous citons ci-après, il conserve donc de l'intérêt pour ces autres possibilités.

Pour ce qui est de la vérification de la validité de la géométrie d'une couche, il faut le paramétrer avec la règle : ***ne doit pas contenir de géométries invalides*.**

Exemple :



Le résultat est toutefois difficile à interpréter, car l'erreur n'est pas typée et toute l'entité fautive est mise en surbrillance :

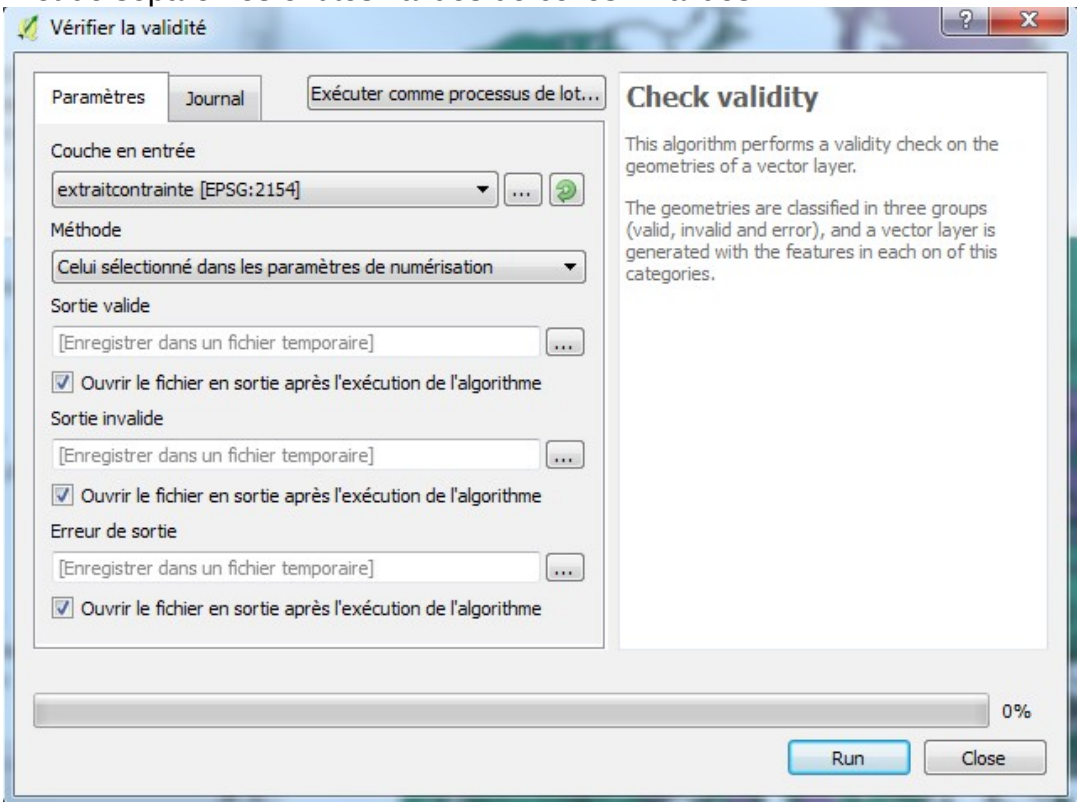


Nous ne le recommandons pas pour cet usage.

4.3.2 - Algorithme 'vérifier la validité' (check validity)

Comme nous l'avons vu (cf. 2.2.3.1), cet algorithme est dédié à la vérification de la validité de la géométrie avec la méthode GEOS ou QGIS.

Il permet de séparer les entités valides de celles invalides.



Cet algorithme est un peu lent d'exécution surtout avec la méthode Qgis.

Les comparaisons que nous avons pu faire sur quelques cas particuliers (voir des exemples dans le chapitre 6), nous ont permis de mettre en exergue des différences entre QGIS et GEOS :

- non détection de '*ring self intersection*', ni par QGIS, ni par GEOS (voir 2.2.3)
- non détection par QGIS de '*duplicate ring*'
- non détection par QGIS de '*Interior is deconnected*' (voir ci-dessous)
- non détection par QGIS de rebroussement sur arc pendant (*self intersection*)
- détection par QGIS de *self-intersection* non reconnues comme erreurs sous GEOS
- détection des points en double par la méthode QGIS.
- Détection incorrecte de 'partie à l'intérieur de partie' pour les multi-polygones (voir 5.1.4)

Comme indiquée, la méthode QGIS n'est pas documentée, il est donc difficile d'établir un bilan exhaustif des différences.

Exemple sur un cas particulier;

Dans cet exemple, la méthode GEOS détecte 26 polygones en erreur (couche sortie invalide) et 1112 valides. Une couche de points « Erreur de sortie » permet de localiser les erreurs .

GEOS détecte 3 types d'erreurs : Self-intersection (auto intersection), too few points in geometry component (pas assez de point pour l'entité géométrique) et Interior is deconnected (intérieur non connecté).

	message
10	Erreur GEOS : Self-intersection
11	Erreur GEOS : Self-intersection
12	Erreur GEOS : Self-intersection
13	Erreur GEOS : Self-intersection
14	Erreur GEOS : Too few points in geometry component
15	Erreur GEOS : Too few points in geometry component
16	Erreur GEOS : Too few points in geometry component
17	Erreur GEOS : Too few points in geometry component
18	Erreur GEOS : Too few points in geometry component

La méthode QGIS détecte 43 polygones en erreur et 143 erreurs sur ces polygones(couche Erreur de sortie)

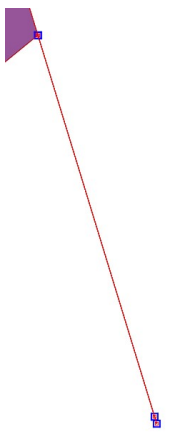
Message	Nombre
Erreur GEOS : Interior is disconnected	1
Erreur GEOS : Self-intersection	12
Erreur GEOS : Too few points in geometry component	13

La couche sortie invalide continent le champ « _errors » qui détaille les erreurs pour chaque polygone.

_errors	nombre
anneaux s'intersectent	1
nœuds en double	36
segments s'entrecroisent	6

La comparaison du résultat des 2 méthodes permet de voir que Qgis détecte les points en double ce que ne fait pas GEOS.

A l'inverse, Qgis ne détecte pas l'erreur GEOS *Interior is deconnected* (intérieur non connecté) qui correspond ici à ce qui ressemble de loin à un arc pendant.

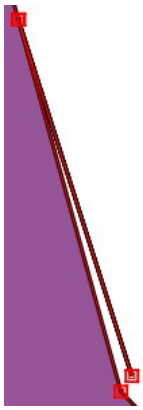


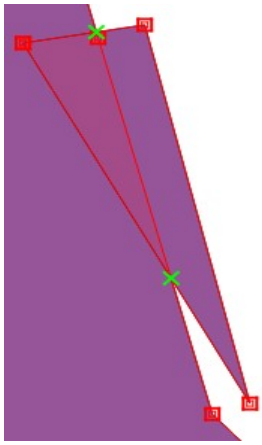
17413	921805,7058	6389439,2094
17414	921806,1618	6389437,7172
17415	921806,1702	6389437,6896
17416	921805,7058	6389439,2094

Avec un point double à la connexion avec le reste du polygone.

L'erreur QGIS « anneaux s'intersectent » correspond à une erreur classée « self intersection » par GEOS. En fait, on est face à un cas compliqué constitué d'un nœud pendant et d'une duplication de points à la base de celui-ci.

Pour la compréhension, on a décalé l'arc pendant. :



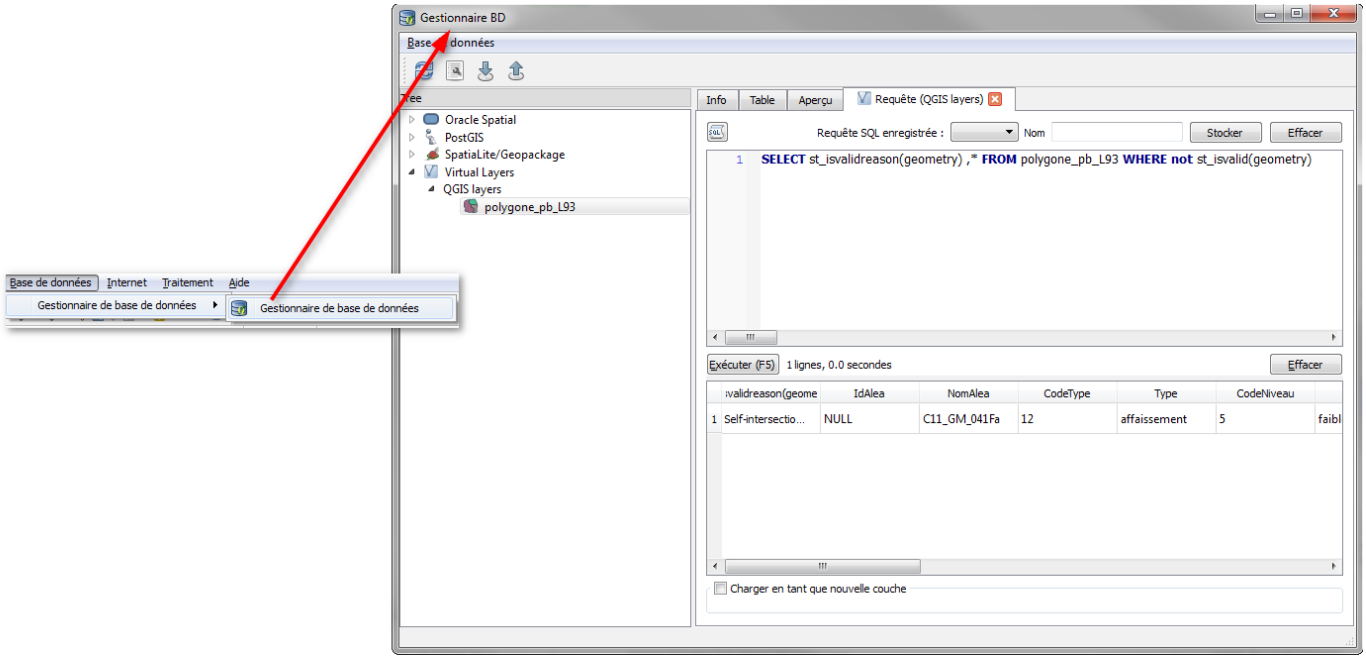


Le sommet de l'arc comporte en fait 3 nœuds (décalés ici)

On retiendra que la méthode QGIS n'est présente que pour des raisons historiques et que la méthode GEOS n'est, au moins dans la version QGIS 2.16.3, pas complète puisque ne détectant pas les anneaux en auto-intersection.

4.3.3 - SQL sous DBManager

La détection d'erreurs peut se faire avec une requête SQL sous DbManager à partir du fournisseur 'Virtual Layer' (pour toutes les couches ouvertes dans QGIS), ou des fournisseurs spatialite, ou PostGIS.



Avec les virtual layers ont peut utiliser les fonctions de vérification de géométries de spatialite (qui sont quasiment les mêmes que celles de PostGIS).

Une requête comme :

SELECT st_isvalidreason(geometry) , FROM extraitcontrainte WHERE not st_isvalid(geometry)*

permet de trouver, dans notre exemple, les 26 polygones détectés par la méthode GEOS de l'algorithme 'vérifier la validité'.

La fonction *st_isvalidreason()* renvoie un texte mêlant la raison de l'erreur et sa localisation

	st_isvalidreason(geometry)	INSEE
1	Self-intersection[925846.5138 6384425.29261251]	05060
2	Too few points in geometry component[926829.7134 6385703.229]	05060
3	Self-intersection[933998.933553813 6382687.66931792]	05035
4	Too few points in geometry component[927751.9092 6394942.28...	05042
5	Too few points in geometry component[927942.2598 6395146.85...	05042
6	Self-intersection[927344.534571426 6394721.3454]	05042
7	Self-intersection[936816.703949749 6397926.9590625]	05139
8	Self-intersection[939246.0492 6390497.2794]	05123

Pour avoir une localisation des erreurs, on peut utiliser la fonction st_isValidDetail(geometry):

SELECT st_isvalidreason(geometry) as raison, st_isValidDetail(geometry) as geometry
FROM extraitcontrainte WHERE not st_isvalid(geometry)

qui renvoie les points

	raison	geometry
1	Self-intersection[925846.5138 6384425.29261251]	Point (925846.5137999999569729 6384425.29261...
2	Too few points in geometry component[926829.7134 6385703....	Point (926829.71340000000782311 6385703.2290...
3	Self-intersection[933998.933553813 6382687.66931792]	Point (933998.93355381302535534 6382687.6693...
4	Too few points in geometry component[927751.9092 6394942....	Point (927751.90919999999459833 6394942.2857...
5	Too few points in geometry component[927942.2598 6395146....	Point (927942.25979999999981374 6395146.8528...
6	Self-intersection[927344.534571426 6394721.3454]	Point (927344.53457142598927021 6394721.3453...

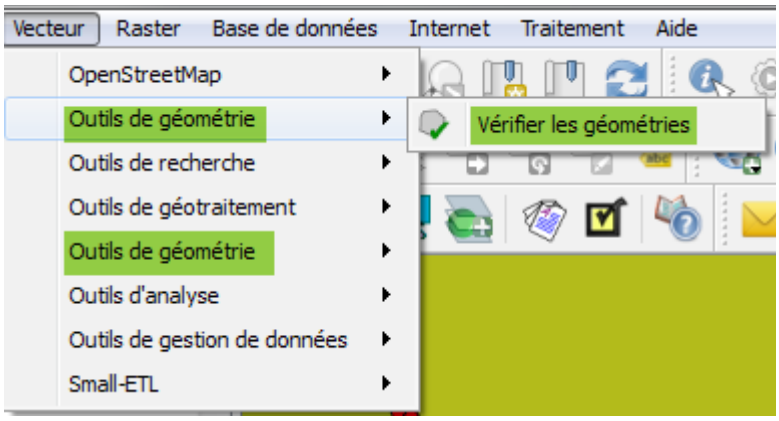
On peut alors comparer, par localisation, les messages d'erreurs entre DbManager et l'algorithme 'vérifier la géométrie avec les méthodes GEOS et Qgis. Ceci nous a permis de montrer que toutes les erreurs ont un message concordant sauf celles vues dans le paragraphe précédent : « Ring self intersection » pour DbManager et « Interior is disconnected » pour Check Validity

	raison	INSEE	niveau_de_	message
1	Self-intersection[941839.148910672 6394938.89251161]	05112	inconstructible	Erreur GEOS : Self-intersection
2	Too few points in geometry component[936291.9084 6384479.2782]	05075	contructible avec...	Erreur GEOS : Too few points in geometry component
3	Self-intersection[939246.0492 6390497.2794]	05123	inconstructible	Erreur GEOS : Self-intersection
4	Ring Self-intersection[921805.7058 6389439.2094]	05010	inconstructible	Erreur GEOS : Interior is disconnected
5	Self-intersection[925846.5138 6384425.29261251]	05060	inconstructible	Erreur GEOS : Self-intersection
6	Self-intersection[923832.708415335 6381501.39423552]	05099	inconstructible	Erreur GEOS : Self-intersection
7	Too few points in geometry component[930388.482 6387510.2046]	05087	contructible avec...	Erreur GEOS : Too few points in geometry component
8	Self-intersection[925333.979071366 6394671.19846008]	05087	contructible avec...	Erreur GEOS : Self-intersection

Cette méthode est donc celle que nous recommandons sous QGIS.

4.3.4 - Outil 'vérifier les géométries' (geometry checker)

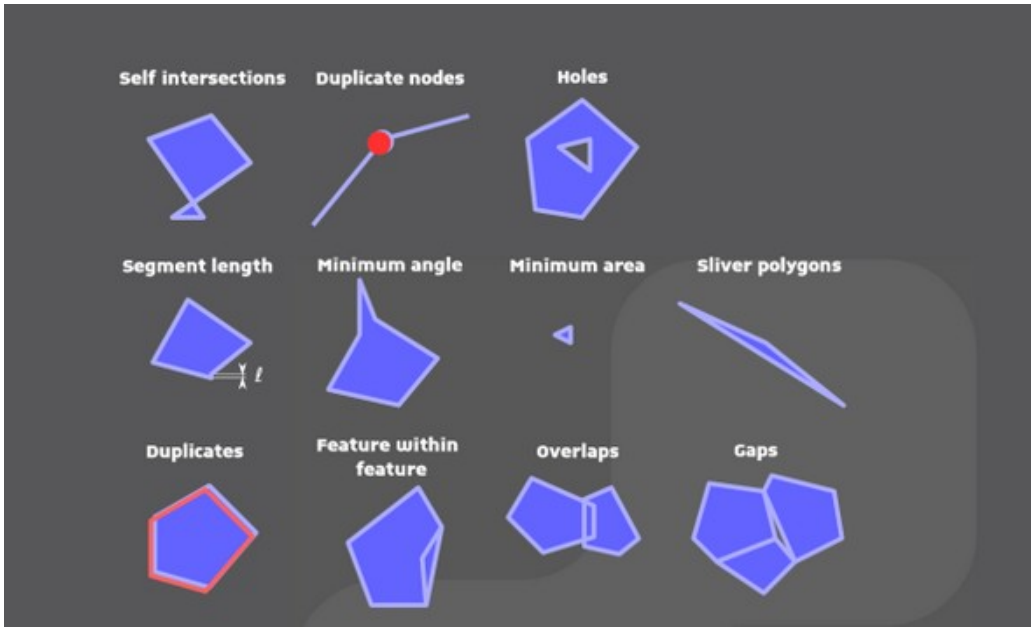
Ce nouvel outil est disponible dans le menu vecteur :



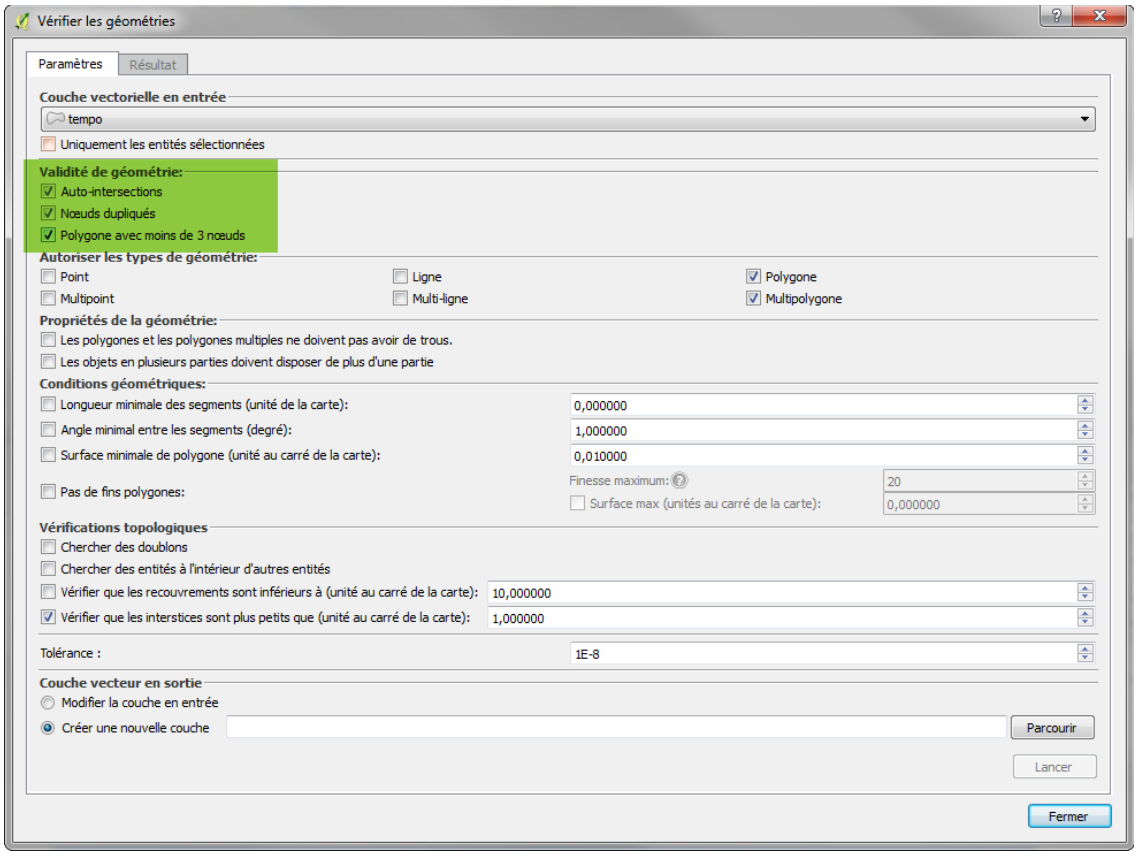
Attention car dans ce menu l'item 'outils de géométrie' est utilisé deux fois sans avoir le même ensemble de fonctionnalités. Si on recherche des informations sur Internet sur ce plugin, il est bon de connaître son nom en anglais 'geometry checker'.

Ce plugin C++ récent offre des fonctions interactives de détection et correction de géométrie et topologie.

La documentation indique qu'il permet de détecter et réparer les erreurs de type :

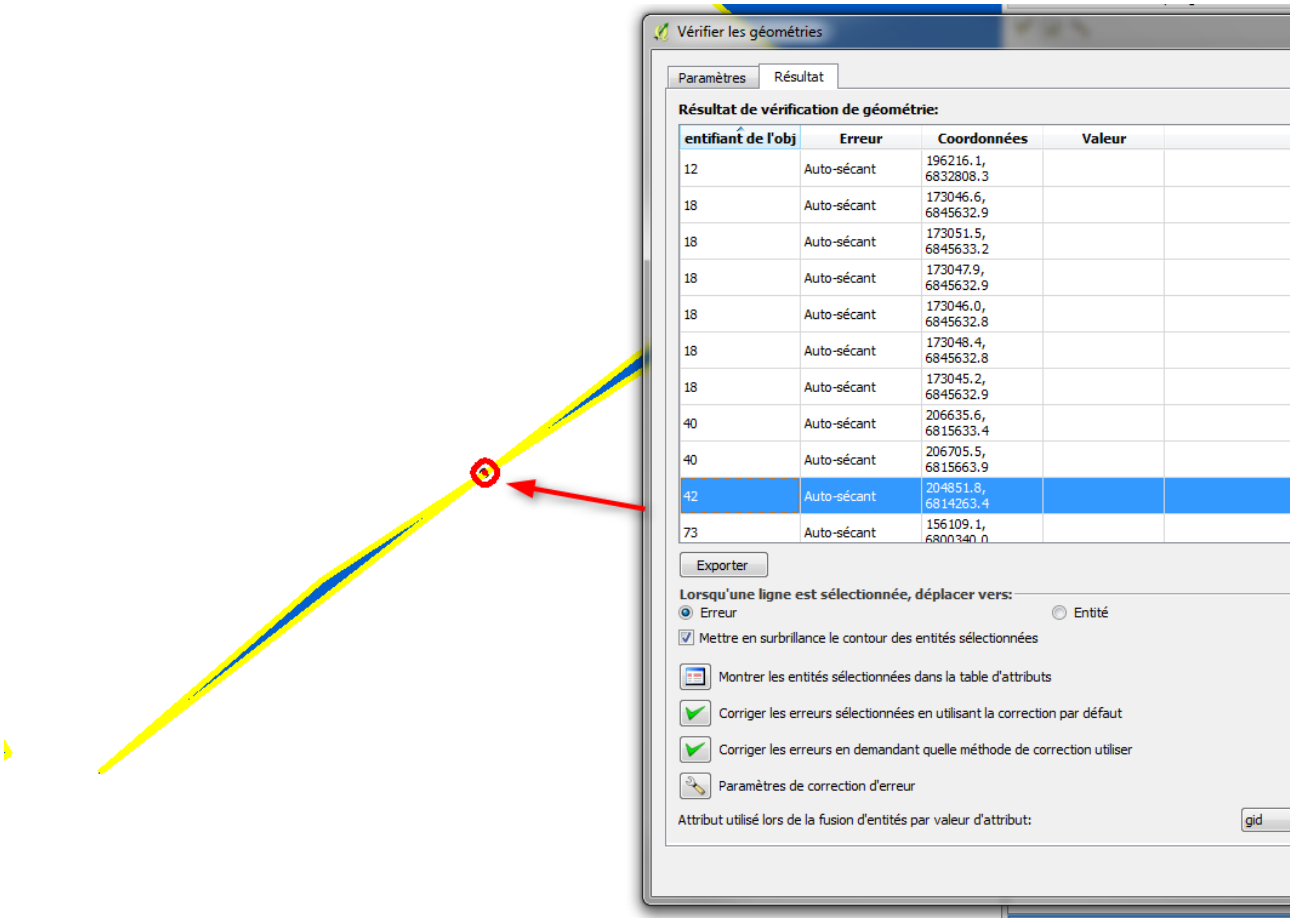


En ce qui concerne la détection des géométries invalides, il n'est pas complet (mais ajoute la détection des nœuds dupliqués et des polygones avec moins de 3 nœuds):



Comme il offre surtout d'autres fonctions qui peuvent permettre de vérifier si un lot de données répond bien aux exigences d'un cahier des charges (recette de lot de données), nous conseillons de le réserver à cette tâche.

L'interactivité pour visualiser les erreurs est un plus intéressant :



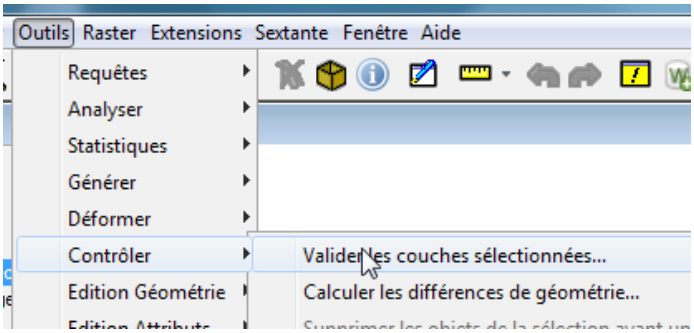
Malheureusement, la partie correction automatique semble assez instable (sous QGIS 2.16) et ce plugin provoque régulièrement des minidumps.

4.4 - recherche d'erreurs avec OPEN JUMP

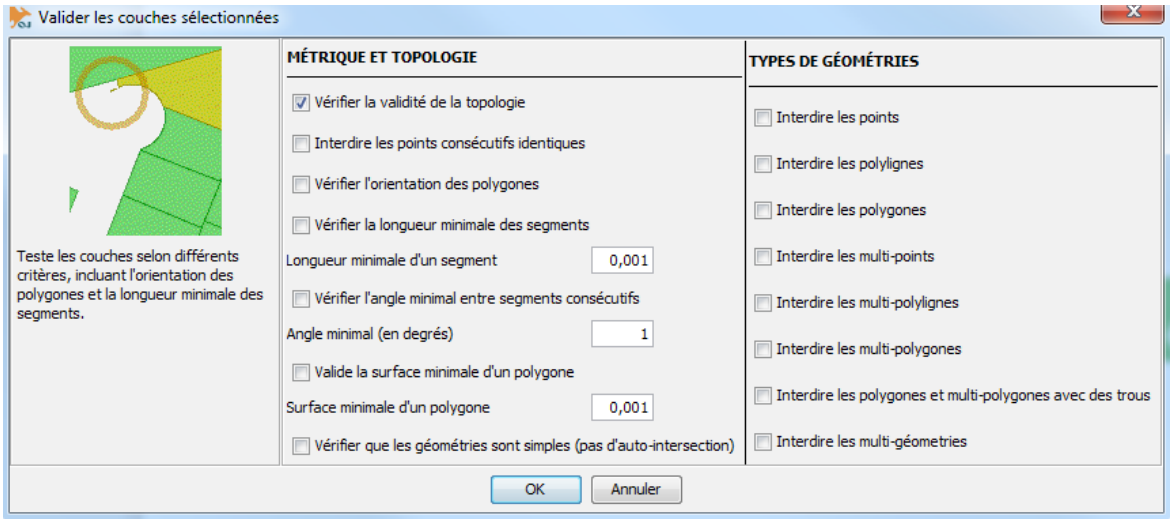
OpenJump est un outil libre comme Qgis et qui fonctionne aussi avec des extensions. Il est développé en Java mais intègre une console Python et un outil de script (BeanTools).

Il est performant pour la recherche et le traitement des erreurs géométriques et vaut le coup de s'y intéresser dans ce cadre.

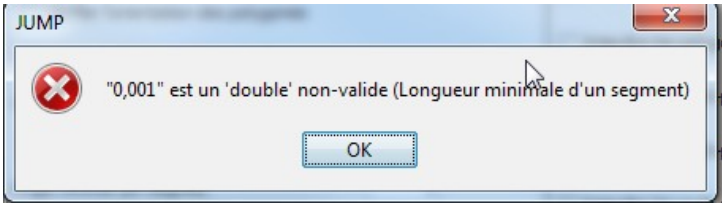
La recherche d'erreurs se fait par le menu Outils/Contrôler/Valider les couches sélectionnées



qui ouvre la fenêtre :



Attention : Il faut mettre le point comme séparateur décimal pour les longueurs et surfaces minimales même si on ne les utilise pas. Sinon, on a le message suivant :

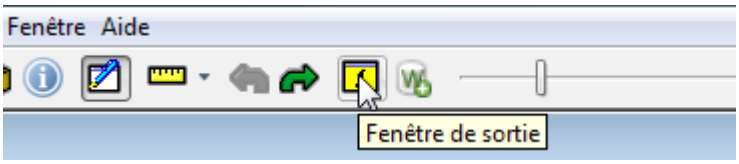


En cochant « Vérifier la validité de la topologie » on effectue en fait la vérification de la géométrie. Il existe une extension « Topologie » qui s'occupe vraiment de l'aspect topologique.

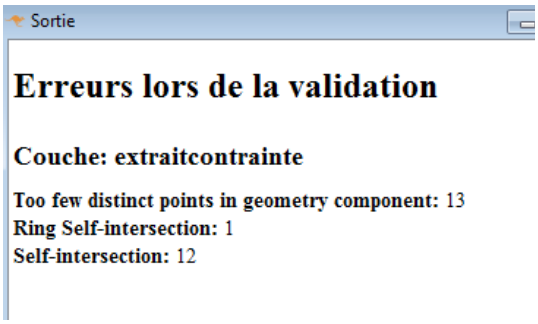
La vérification crée 2 couches temporaires (sans source de données) :

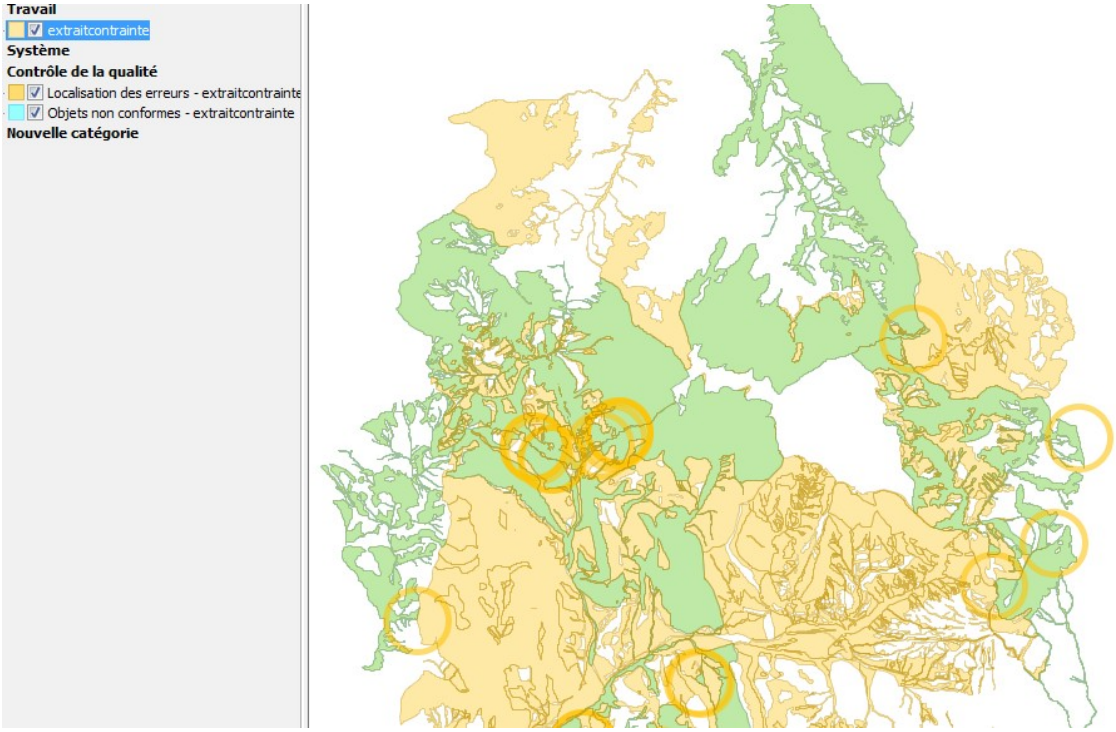
« Objets non conformes » et « Localisation des erreurs » comme dans Qgis.

De plus, la fenêtre de sortie clignote



Un clic sur celle-ci donne le résumé du résultat :





Dans notre exemple avec la couche « extraitcontrainte » , on retrouve les 26 objets non conformes trouvés par Qgis.

Le temps de détection est très rapide. Sur cette couche de 1138 objets, la détection prend une seconde.

OpenJump permet, en plus des actions présentes dans QGIS, de vérifier l'orientation des polygones.

4.5 - Conclusions sur la recherche d'erreurs

4.5.1 - détection des invalidités

Comme indiqué dans le paragraphe 4.3.2 nous ne recommandons pas d'utiliser la méthode QGIS avec l'algorithme 'vérifier la géométrie' qui n'est pas documentée et ne tient pas compte du standard GEOS.

Dans l'état actuel, nous ne recommandons pas, plus généralement, d'utiliser l'algorithme 'vérifier la géométrie' avec la méthode GEOS, car elle est incomplètement implémentée (QGIS 2.16).

Nous recommandons d'utiliser les requêtes SQL sous DbManager avec la fonction st_isvalidreason(), ou en alternative pour ceux qui le souhaitent Openjump qui est rapide et donne les mêmes résultats.

4.5.2 - détections des autres erreurs de géométrie

Pour les erreurs de contraintes de géométrie, '*vérifier les géométries*' est l'outil à utiliser sous QGIS, il est malheureusement assez instable (plantages fréquents).

OpenJump offre en alternative des outils intéressants.

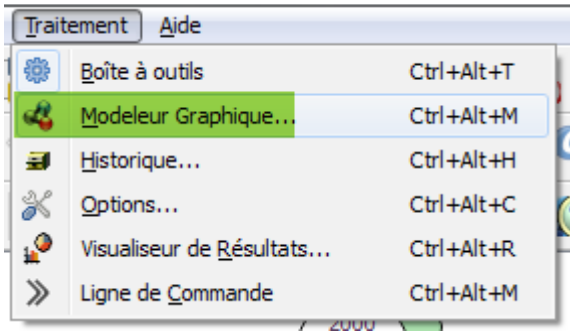
4.5.3 - Détection des erreurs de topologie

Le **vérificateur de topologie** est l'outil à utiliser sous QGIS.

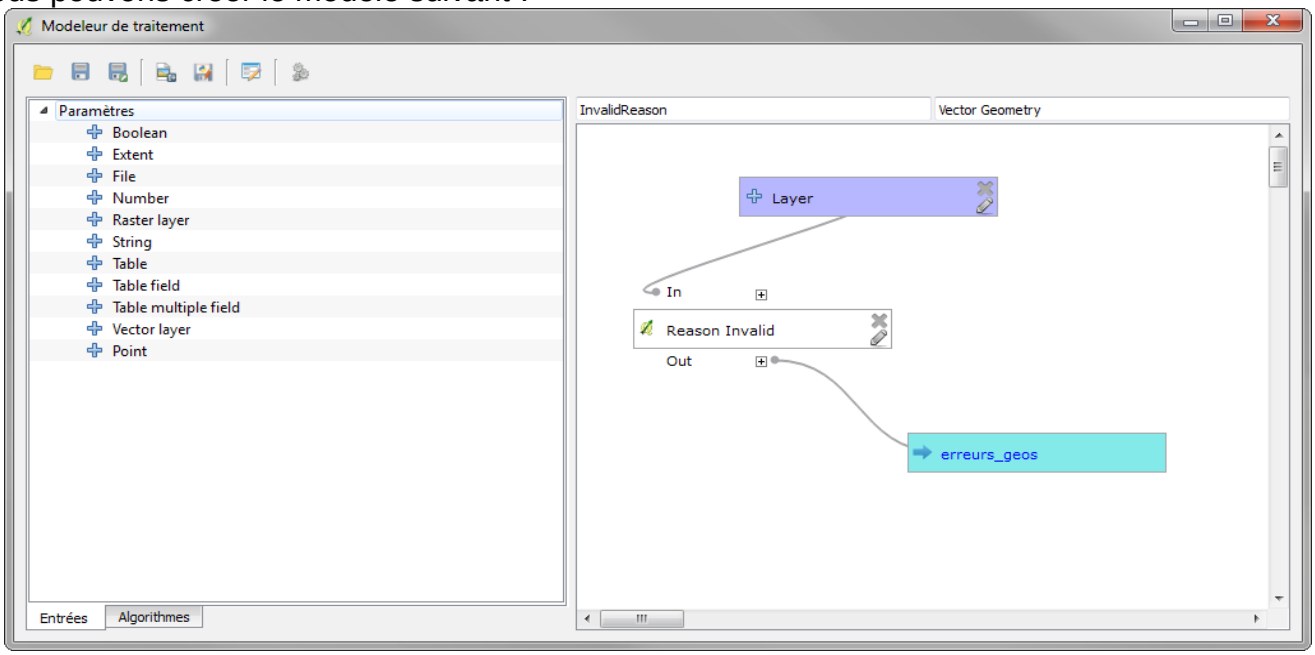
Openjump offre également des fonctions que nous survoleront dans le chapitre sur les corrections.

4.5.4 - Automatisation de la détection des géométries invalides par un modèle de traitement

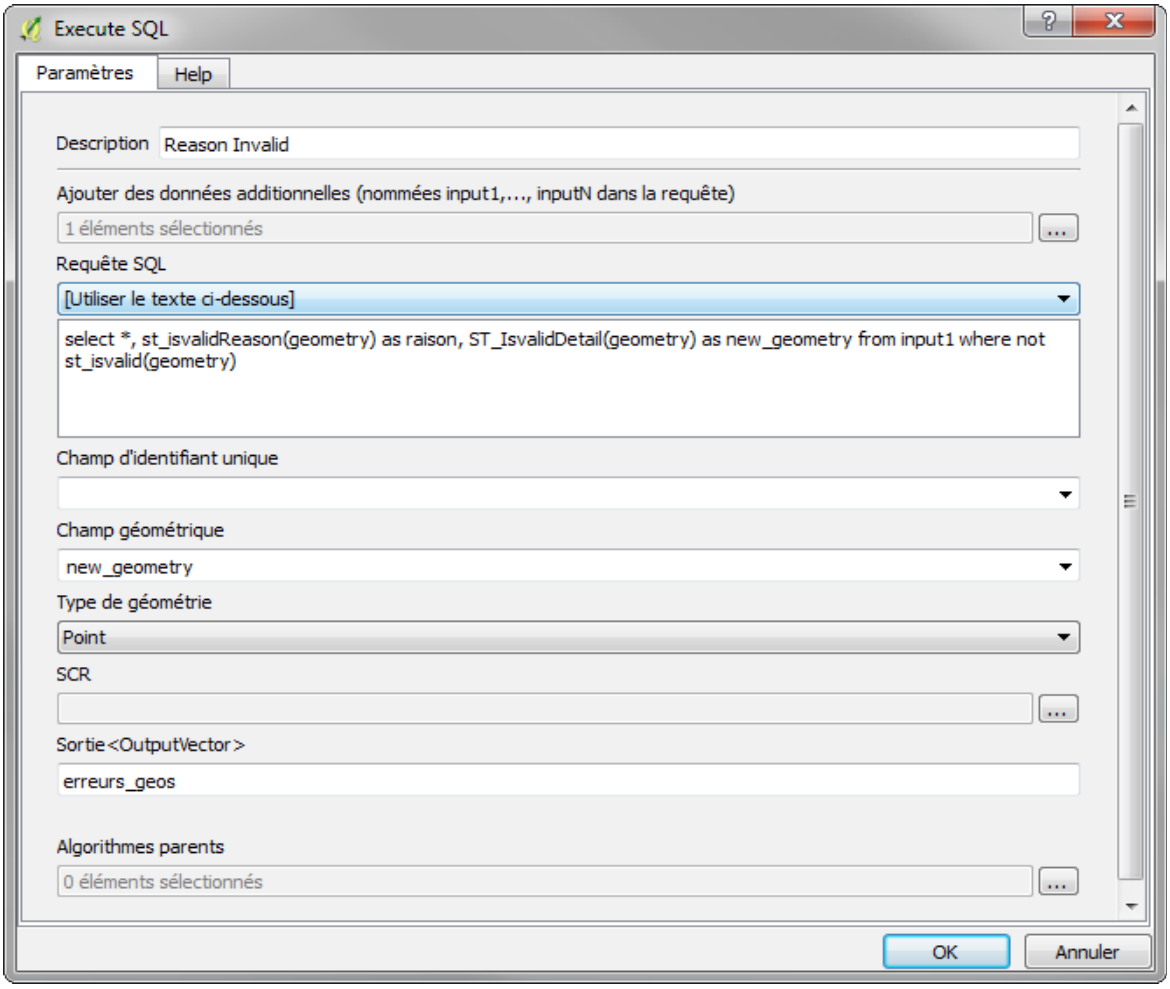
QGIS permet de créer des modèles de traitement à partir du modeleur graphique disponible dans le menu traitement :



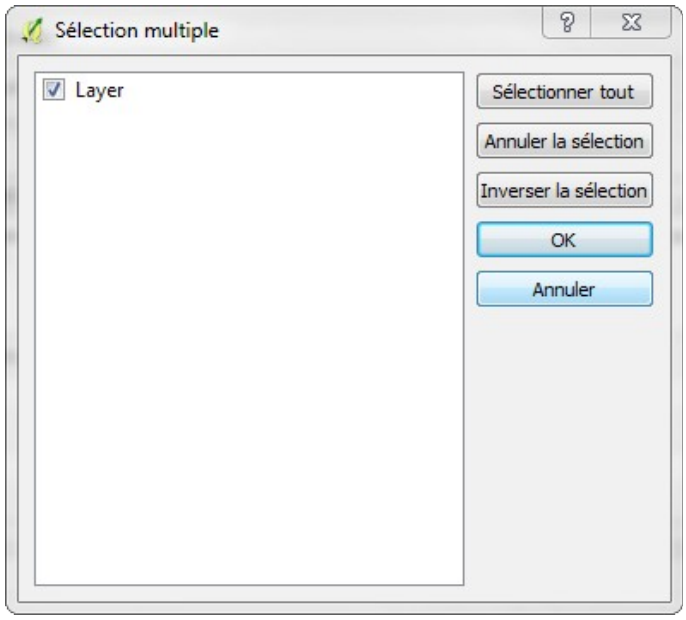
Nous pouvons créer le modèle suivant :



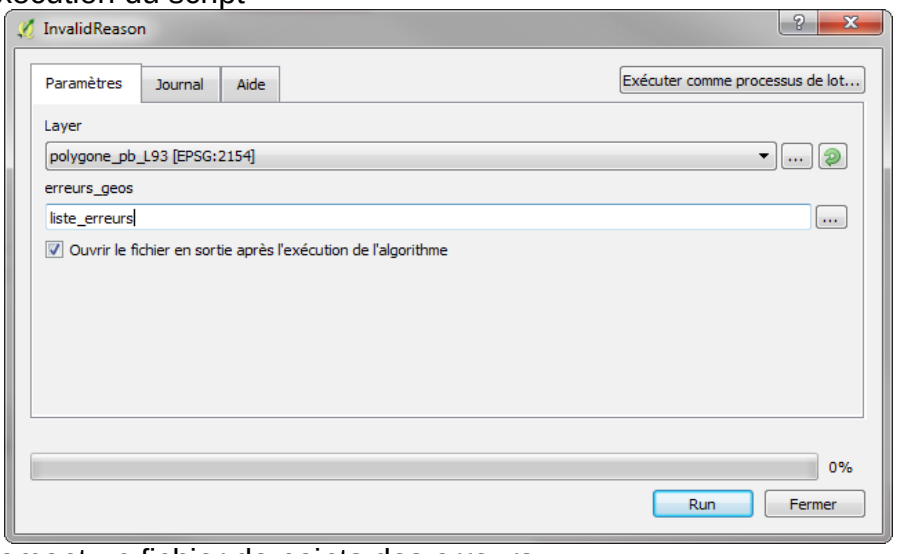
le cœur du modèle étant l'algorithme 'Exécuter SQL' avec , par exemple, les paramètres suivants :



il faut sélectionner le ‘input1’ avec le bouton ...
et choisir ‘layer1’



Ceci permet à l’exécution du script



de générer directement un fichier de points des erreurs.

Si on lance ce script sur une couche sans erreur, il va générer un message d’erreur.

On peut aussi se créer un script *layer_diagnostic* qui va simplement lister le statut des entités et leur nombre avec une requête :

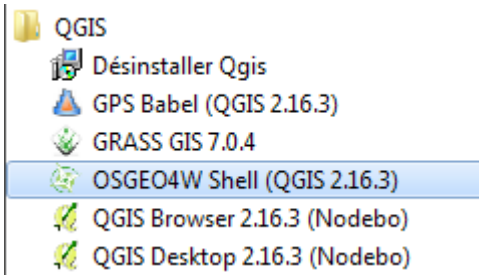
select substr(st_IsValidReason(Geometry), 1, instr(st_IsValidReason(Geometry), '[')-1) as raison, count(*) as nb_entites from input1 GROUP BY raison

On obtiendra alors un diagnostic comme celui-ci :

	raison	nb_entites
1	Ring Self-intersection	42
2	Valid Geometry	3528

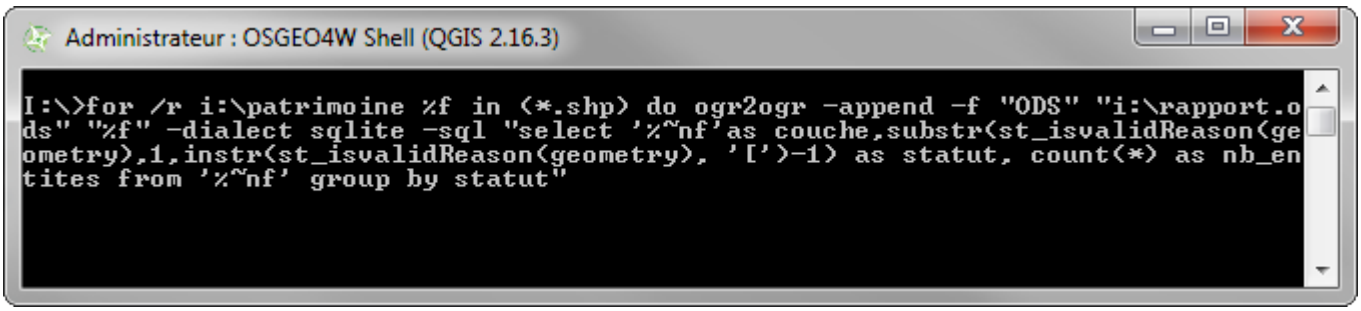
4.5.5 - diagnostic d'un ensemble de couches

On peut utiliser la console Osgeo4W pour utiliser ogr2ogr dans le but de diagnostiquer un patrimoine de couches.
Osgeo4w est accessible par le menu :



On peut utiliser par exemple la commande suivante pour générer un rapport (*i:\rapport.ods*) dans cet exemple)

for /r i:\patrimoine %f in (.shp) do ogr2ogr -append -f "ODS" "i:\rapport.ods" "%f" -dialect sqlite -sql "select '%~nf' as couche, substr(st_IsValidReason(Geometry), 1, instr(st_IsValidReason(Geometry), '[')-1) as statut, count(*) as nb_entites from '%~nf' GROUP BY statut"*



Pour quelques explications sur cette ligne de commande on pourra consulter ce site.
Cette commande parcourt le répertoire *i :\patrimoine* et tous ses sous-répertoires et génère un fichier ods qui indique un statut vide pour les entités valides et sinon les causes d'invalidité et le nombre d'objets correspondant. Le fichier *rapport.ods* doit être supprimé avant de lancer la commande (mode append)

Exemple de résultat :

	A	B	C
1	couche	statut	nb_entites
2	invalidgeometry		3
3	invalidgeometry	Interior is disconnected	1
4	invalidgeometry	Ring Self-intersection	1
5	invalidgeometry	Self-intersection	1
6	invalidgeometry_apres_makevalid		6
7	polygones_mal_formes		5
8	polygones_mal_formes	Duplicate Rings	1
9	polygones_mal_formes	Interior is disconnected	1
10	polygones_mal_formes	Ring Self-intersection	1
11	polygones_mal_formes	Self-intersection	4
12	alea_fluvial_apres_makevalid		3570
13	Alea_fluvial_original_06		3528
14	Alea_fluvial_original_06	Ring Self-intersection	42

Dans cet exemple on constate que la couche *alea_fluvial_apres_makevalid* est correcte, alors que la couche *alea_fluvial_original_06* a 42 objets en erreur de type '*Ring Self intersection*'.

5 - METHODES CORRECTION

Confronté à une couche non valide, et en l'absence de possibilité de demander les corrections au producteur (ce qui doit être le premier réflexe) on peut être tenté de corriger soi-même la couche. La correction des géométries invalides est la plus pertinente pour permettre une utilisation correcte sous QGIS ou PostGIS. D'autres corrections sont possibles mais doivent être envisagées avec beaucoup de prudence (cf résumé de ce document)

5.1 - Correction des géométries invalides

5.1.1 - Makevalid

Pour corriger les géométries invalides au sens GEOS, l'outil préférentiel à utiliser est la fonction st_makevalid disponible sous spatialite ou PostGIS.

Cet outil n'est pas documenté sur Internet, pour comprendre ce qu'il fait il faudrait examiner le code. On peut cependant constater ses effets sur les erreurs courantes par exemple de polygones.

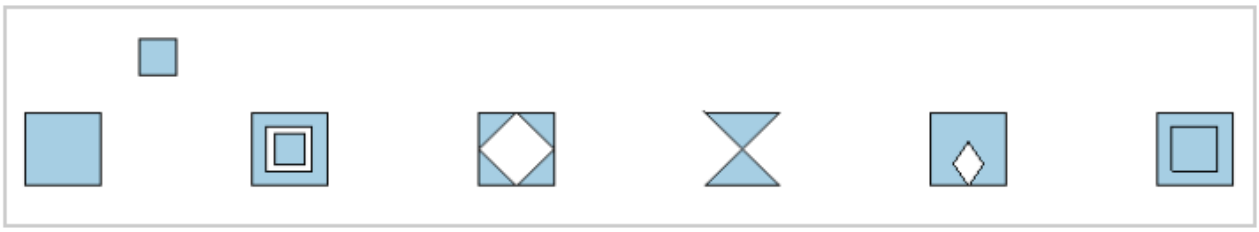
Nb : la lecture de ce paragraphe n'est utile que pour ceux qui veulent en savoir plus...

Le script SQL ci-dessous extrait du mdoule4 de la formation PostgreSQL/PostGIS permet de générer des polygones invalides dans une table invalidgeometry du schema travail (couche postgis):

```
CREATE TABLE travail.invalidgeometry (id serial, type varchar(20), geom geometry(MULTIPOLYGON, 2154), PRIMARY KEY(id));

INSERT INTO travail.invalidgeometry (type, geom) VALUES ('Hole Outside Shell', ST_multi(ST_GeomFromText('POLYGON((465000 6700000, 465010 6700000, 465010 6700010, 465000 6700010, 465000 6700000), (465015 6700015, 465015 6700020, 465020 6700020, 465020 6700015, 465015 6700015))',2154)));

INSERT INTO travail.invalidgeometry (type, geom) VALUES ('Nested Holes', ST_multi(ST_GeomFromText('POLYGON((465030 6700000, 465040 6700000, 465040 6700010, 465030 6700010, 465030 6700000), (465032 6700002, 465032 6700008, 465038 6700008, 465038 6700002, 465032 6700002), (465033 6700003, 465033 6700007, 465037 6700007, 465037 6700003, 465033 6700003))',2154)));

INSERT INTO travail.invalidgeometry (type, geom) VALUES ('Dis. Interior', ST_Multi(ST_GeomFromText('POLYGON((465060 6700000, 465070 6700000,465070 6700010, 465060 6700010, 465060 6700000), (465065 6700000, 465070 6700005, 465065 6700010, 465060 6700005, 465065 6700000))', 2154)));

INSERT INTO travail.invalidgeometry (type, geom) VALUES ('Self Intersect.', ST_multi(ST_GeomFromText('POLYGON((465090 6700000, 465100 6700010, 465090 6700010, 465100 6700000, 465090 6700000))',2154)));

INSERT INTO travail.invalidgeometry (type, geom) VALUES ('Ring Self Intersect.', ST_multi(ST_GeomFromText('POLYGON((465125 6700000, 465130 6700000, 465130 6700010, 465120 6700010, 465120 6700000, 465125 6700000, 465123 6700003, 465125 6700006, 465127 6700003, 465125
```

6700000))',2154)));

INSERT INTO travail.invalidgeometry (type, geom) VALUES ('Nested Shells',
ST_multi(ST_GeomFromText('MULTIPOLYGON(((465150 6700000, 465160 6700000, 465160 6700010, 465150 6700010, 465150 6700000)),((465152 6700002, 465158 6700002, 465158 6700008, 465152 6700008, 465152 6700002)))',2154)));



Chacun de ces objets est invalide, ce que l'on peut vérifier avec la requête suivante :

SELECT id, type, ST_IsValidReason(geom) FROM travail.invalidgeometry WHERE NOT ST_IsValid(geom);

	id integer	type character varying(20)	st_isvalidreason text
1	1	Hole Outside Shell	Hole lies outside shell[465015 6700015]
2	3	Nested Holes	Holes are nested[465033 6700003]
3	4	Discon. Interior	Interior is disconnected[465070 6700005]
4	5	Self Intersect.	Self-intersection[465095 6700005]
5	6	Ring Self Intersect.	Ring Self-intersection[465125 6700000]
6	7	Nested Shells	Nested shells[465152 6700002]

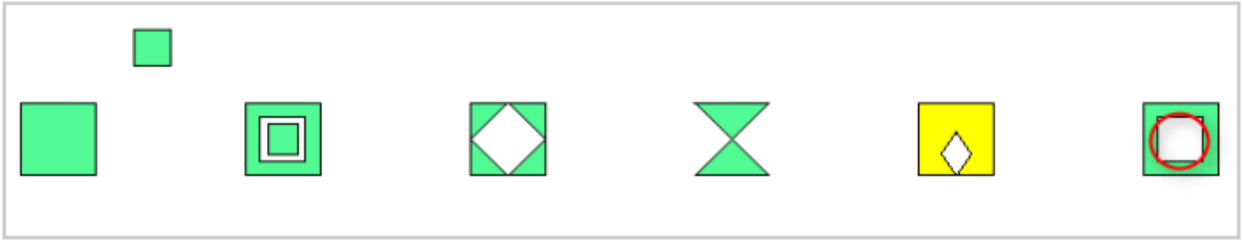
On peut tenter de corriger la géométrie avec la requête suivante :

```
CREATE TABLE travail.makevalidgeometry AS
(SELECT id, type, ST_MULTI(ST_MakeValid(geom))::geometry(MULTIPOLYGON, 2154) AS geom FROM travail.invalidgeometry WHERE NOT St_IsValid(geom));
```

ou avec les virtuals layers :

```
SELECT id, type, ST_MULTI(ST_MakeValid(geometry)) AS geom FROM polygon_error;
```

La table corrigée ressemble à :



Dans laquelle seule le dernier élément ne s'affiche pas de la même façon.

Constater que la requête :

```
SELECT id, type, ST_IsValidReason(geom) FROM travail.makevalidgeometry WHERE NOT ST_IsValid(geom);
```

Ne trouve plus d'erreur.

Si on analyse les nouveaux polygones (avec les outils décrit au paragraphe 4.1) on constate que :

'Hole Outside Shell' (trou extérieur à l'enveloppe) : un seul polygone composé d'un trou en dehors du polygone de départ est devenu un multi-polygone composé de deux polygones.

'Nested Holes' (trous imbriqués) : est devenu un multipolygone composé d'un polygone avec trou et d'un deuxième polygone qui est au centre (le plus petit).

'Disconnected Interior' : (le trou touche le polygone en plus de 1 point) : est devenu un multi-polygone composé de 4 polygones en triangle.

'Self Intersection' (auto-intersection) : un multi-polygone composé de deux polygones en triangle.

'Ring Self Intersection'(anneau auto-intersectant, avec ici réduction de l'anneau en un point) : est devenu un polygone à trou (trou en contact en 1 point avec l'enveloppe ce qui est correct au sens de geos).

'Nested shell' (polygones imbriqués) : est devenu un polygone à trou.

La première et la dernière correction peuvent porter à discussion, il faudrait examiner la pertinence de la correction à partir des données originales.

Une vérification intéressante est de demander le calcul du périmètre et de la surface avant et après correction (par exemple avec Vecteur → Exporter / ajouter des colonnes de géométries).

On constate que pour le polygone 4 (self intersection) la surface passe de 0 (incorrect) à 50 (correct).

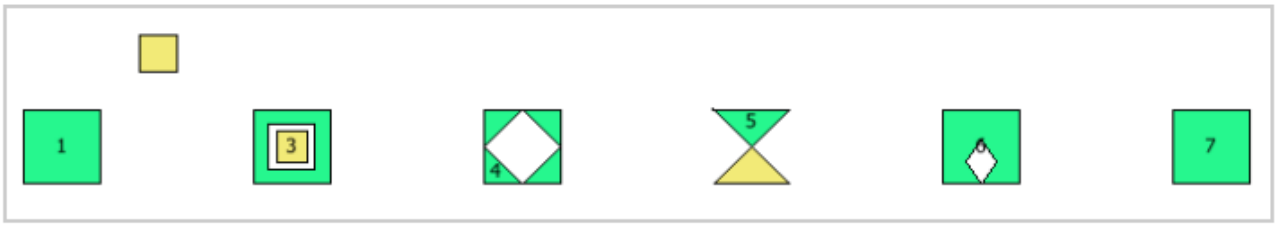
5.1.2 - ST_Buffer(geom, 0)

Une autre solution souvent proposée sur les forums est de corriger les erreurs de géométrie en réalisant un buffer nul.

Exécuter le script suivant :

```
CREATE TABLE travail.geometryvalidbuffer AS
(SELECT id, type, ST_Multi(ST_Buffer(geom,0))::geometry(MULTIPOLYGON, 2154) FROM travail.invalidgeometry WHERE NOT ST_IsValid(geom));
```

Charger cette couche dans QGIS :



Constater que les parties de polygones en jaune ont disparu et que le polygone 7 a été corrigé différemment.

Avant d'utiliser cette méthode (qui donne un résultat éventuellement plus satisfaisant pour le polygone 1 et le polygone 7) il faut obligatoirement avoir corrigé autrement les polygones de type 'self intersection'.

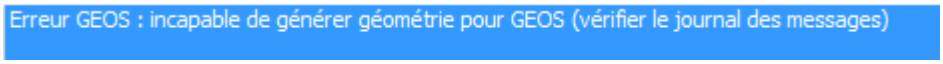
En terme de rapidité, st_Buffer est plus rapide que makevalid :

Sur une table de 1138 objets, makevalid se termine en 6 secondes 78 et st_buffer en 2 secondes 38 soit environ 3 fois plus vite .

5.1.3 - géométrie nulle

Il peut arriver qu'une couche contienne des objets auxquelles ne sont pas associé de géométrie (en particulier parce que la géométrie a été supprimée intentionnellement ou non.

Dans ce cas le message de vérification de l'algorithme '**vérifier la validité**' est avec la méthode GEOS :



et avec la méthode QGIS :

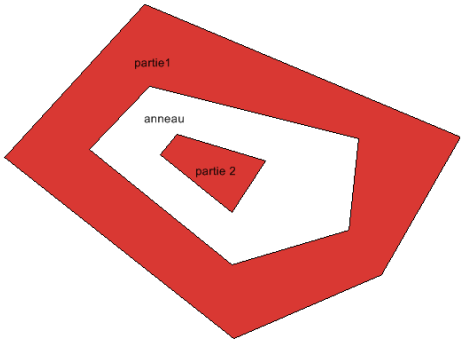
La vérification avec st_isvalidreason donne :

On peut décider de supprimer ces 'objets' ou de les associer à une géométrie avec 'ajouter une partie'.

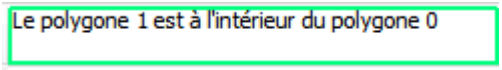
5.1.4 - Multipolygone : Partie à l'intérieur d'une autre partie

L'algorithme vérifier la validité au sens QGIS détecte comme une anomalie une partie à l'intérieur d'une autre partie pour un multipolygone.

Exemple :



Ce Multipolygone contient un polygone qui est dans l'anneau appartenant au polygone1. Ce n'est pas correct au sens de QGIS qui affiche :



Cette non-conformité ne semble pas avoir de conséquences, et en réalité semble plutôt une erreur de l'algorithme.

5.1.5 - conclusions corrections géométries invalides

Les recommandations que l'on peut dégager sont :

- détecter le type d'erreur par st_isvalidreason() sous Qgis ou en utilisant OpenJump.
- utiliser Makevalid
- On peut ajouter la suppression des points en double, par st_simplify() à 0 (voir ci-dessous 5.2.1) , ce qui permet de corriger la plupart des problèmes.

Pour cela, Sous Postgis on utilisera :

```
update ma_table SET geom =
st_multi(st_simplify(ST_Multi(ST_CollectionExtract(ST_ForceCollection(ST_MakeValid(geo
m)),3)),0)) WHERE ST_GeometryType(geom) = 'ST_MultiPolygon'
```

Les fonctions St_ForceCollection et st_CollectionExtract permettent de gérer les cas où Makevalid génère des objets autres que polygones (polylignes ou points).

On peut utiliser la même requête sur un table de polylignes en changeant le paramètre 3 en 2

```
st_multi(st_simplify(ST_Multi(ST_CollectionExtract(ST_ForceCollection(ST_MakeValid(geo
m)),2)),0)) WHERE ST_GeometryType(geom) = 'ST_MultiLinestring'
```

Attention

Ne pas utiliser directement

UPDATE matable SET geom=ST_Makevalid(geom)

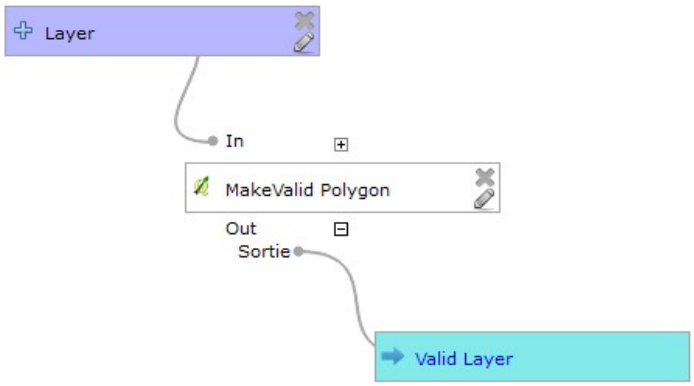
car la fonction st_makevalid peut décomposer les objets complexes… le update ne retiendra que le 1er objet de la décomposition pour mettre à jour la table.

Sur les très grandes tables, st_makevalid peut échouer. On tentera alors de passer par une méthode alternative ;

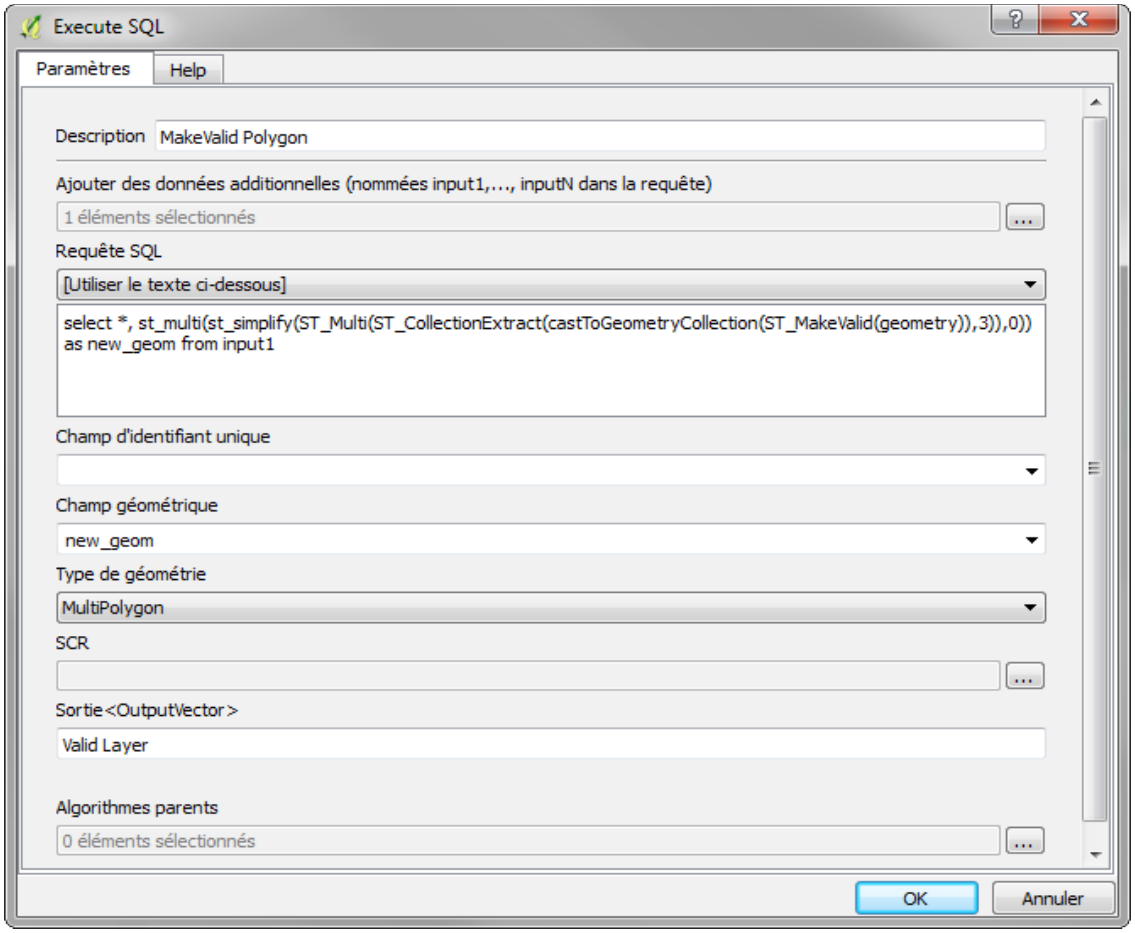
- Tenter Openjump
- st_buffer(geom, 0) si on a corrigé les 'papillons',
- rastérisation si applicable (voir plus loin)
- pprepair qui dans l'état est seulement disponible sous système LINUX et que nous ne détaillons pas dans cette version du rapport. Pour plus de détails voir ici.

5.1.6 - Automatisation de la réparation des géométries invalides

Comme décrit au paragraphe 4.5.4 on peut également automatiser la correction géométrique à l'aide d'un modèle de traitement.

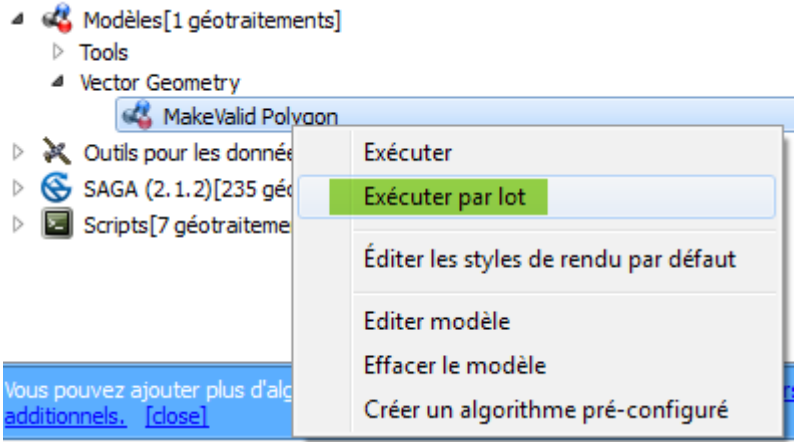


Le cœur de l'algorithme sera un algorithme 'Exécuter SQL' avec les paramètres suivants :



L'algorithme 'Exécuter SQL' utilise comme moteur SQL celui de spatialite (même principe que pour les couches virtuelles de QGIS), ainsi les fonctions et les limites sont celles de la version de spatialite utilisée avec la version de QGIS que l'on trouve dans le 'a propos' de QGIS. Par exemple spatialite 4.3.0 pour QGIS 2.16.3 (c'est pourquoi on utilise par exemple *castToGeometryCollection(geometry)* au lieu de *st_ForceCollection(geometry)* qui n'existe pas dans cette version de spatialite).

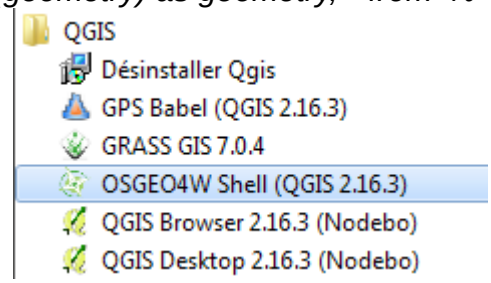
A noter que les algorithmes peuvent être exécutés sur un lot de données :

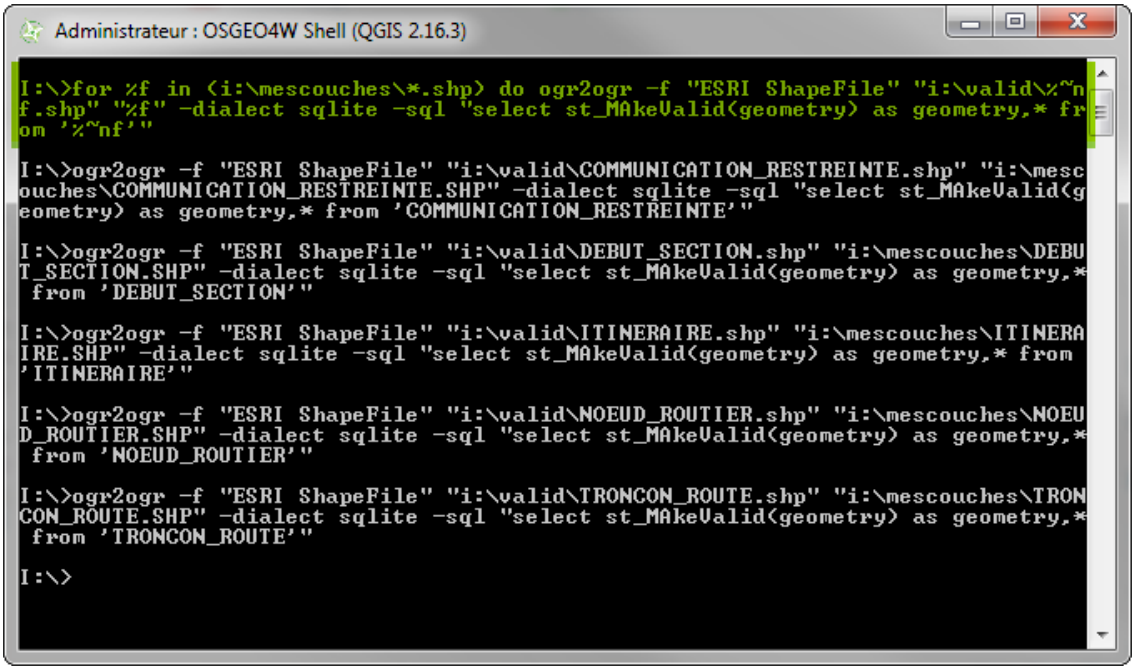


Il est ainsi possible de traiter tout un répertoire.

Une autre solution possible pour les automatisations est de passer directement par un script dans la console Osgeo4w utilisant ogr2ogr avec une syntaxe de type :

for %f in (i:\mescouches.shp) do ogr2ogr -f "ESRI Shapefile" "i:\valid\%~nf.shp" "%f" -dialect sqlite -sql "select st_makevalid(geometry) as geometry, * from '%~nf'"*





Attention, cependant à faire un minimum de vérifications avant de considérer les lots de données comme corrects.

5.2 - Corrections géométriques autres

Même si les entités sont valides au sens d'une norme (par exemple GEOS), elles peuvent être invalides par rapport au cahier des charges ou au standard métier. Nous examinons ici quelques cas de corrections possibles :

5.2.1 - Points en doubles

Les points en doubles sont souvent des erreurs de saisies qui provoquent des ralentissements de calcul ainsi des blocages de traitements spatiaux.

Nb : L'algorithme **vérifier la validité** avec la méthode QGIS détecte les nœuds en double (ce que ne fait pas la méthode GEOS).

Plusieurs méthodes permettent de les supprimer :

avec DbManager Postgis/ Spatialite :

- Le **buffer à zéro** (Cf. 5.1.2) le fait automatiquement.
- La **simplification à zéro** fonctionne mais supprime certains polygones non valides

```
1  select  st_astext(st_simplify(geometry,0)) as geom_simpl, geometry
2  from extraitcontrainte
3  where INSEE='05131'
```

Exécuter (F5) 4 lignes, 0.0 secondes

	geom_simpl	geometry
1	POLYGON((925982.4714 6383638.2438, 925972.4328 6...	Polygon ((925982.47139999992214143 63836...
2	POLYGON((925885.6278 6383517.1554, 925886.9976 6...	Polygon ((925885.62779999990016222 63835...
3	POLYGON((925535.2122 6383377.6452, 925541.3946 6...	Polygon ((925535.21219999995082617 63833...
4	POLYGON()	Polygon ((925742.19479999993927777 63805...

Il faut donc auparavant corriger la couche par exemple par un st_makevalid(). On peut aussi utiliser la fonction spatialite *sanitizeGeometry(geometry)* qui supprime les points en double.

Algorithme simplifier les géométries (ou Vecteur/outils de géométrie /simplifier les géométries)

Une simplification à 0 peut ne pas supprimer tous les points en double (refaire une vérification a posteriori). En particulier sur des géométries proche d'une 'aiguille'.

outil **vérifier les géométries**

Les corrections des nœuds en double peuvent se terminer par un mimi-dump. L'outil plante notamment lorsque la suppression du nœud en double entraîne la destruction de la géométrie (objets à 3 nœuds dont un en double..).

OpenJump.

On peut détecter les points en double par 'Controler → valider les couches sélectionnées et cocher 'interdire les points consécutifs identiques'.

La suppression des points en doubles peut se faire par Outils → contrôler → Réparer les géométries invalides et cocher 'Supprimer les points doubles'.

Conclusion pour les nœuds en double :

La suppression des nœuds en double est à effectuer sur des objets valides.

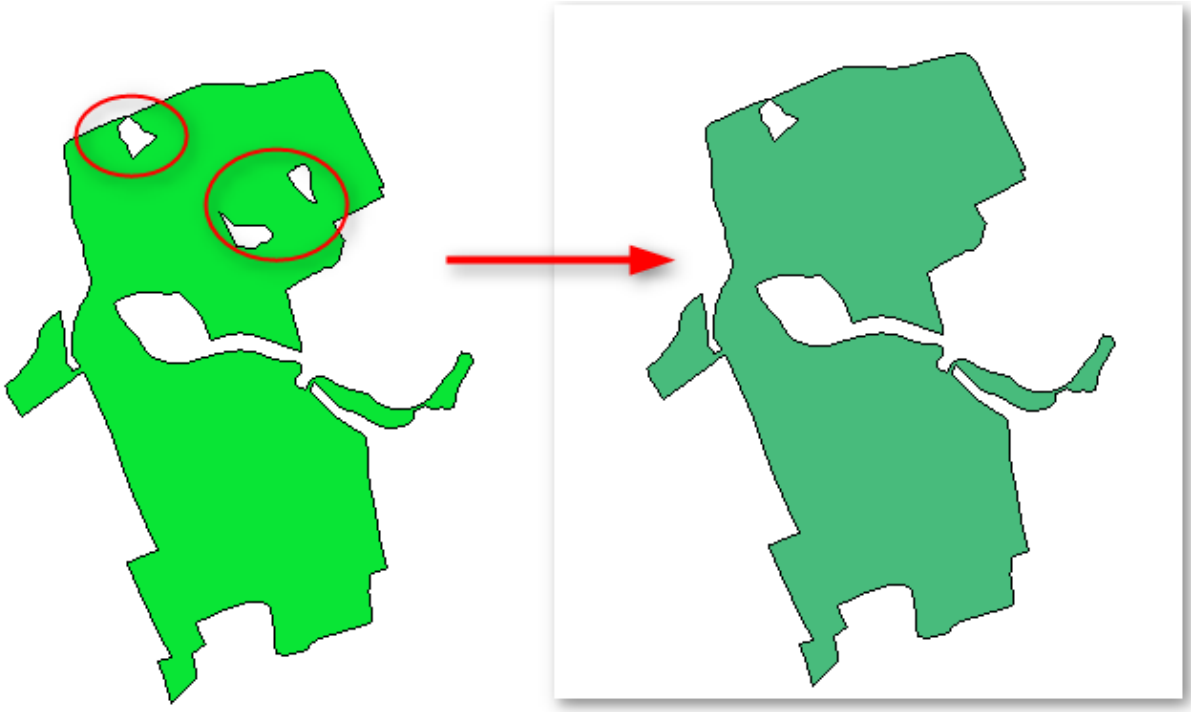
5.2.2 - trous internes : algorithmes 'fill holes' et 'delete holes'

Nous avons vu que les méthodes makevalid() et st_buffer(0) différent dans le traitement de certains polygones.

Il existe les algorithmes *fill holes* et *delete holes* (supprimer les trous) qui permettent de supprimer les trous. Fill holes permet d'indiquer une taille maximum pour la suppression des trous alors que delete holes s'applique sur toute la couche.

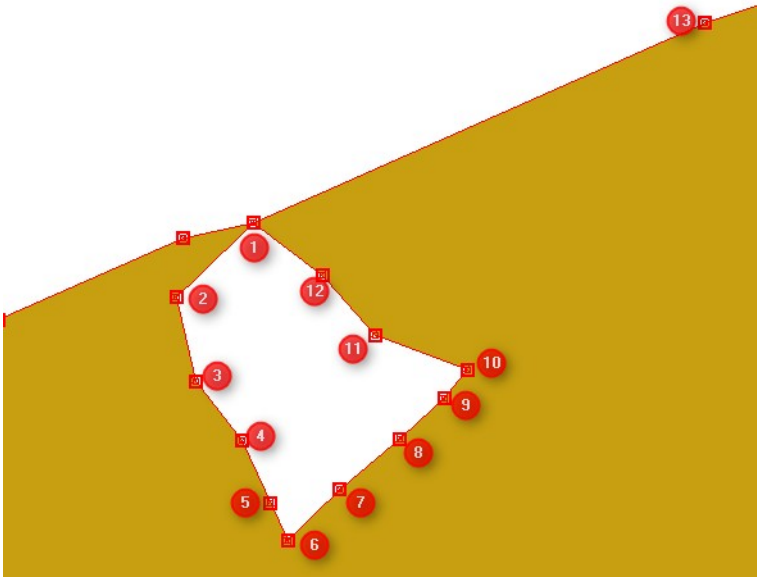
Exemple :

supprimer les trous sur ce polygone donne :



Dans ce cas particulier le 'trou' du haut n'est pas considéré par l'algorithme de nettoyage comme un trou, car il est le résultat de la description du contour (ce n'est pas un anneau interne).

En effet, la description du contour commence comme indiquée :



et se termine par le point 248 qui est le même que le point 1.

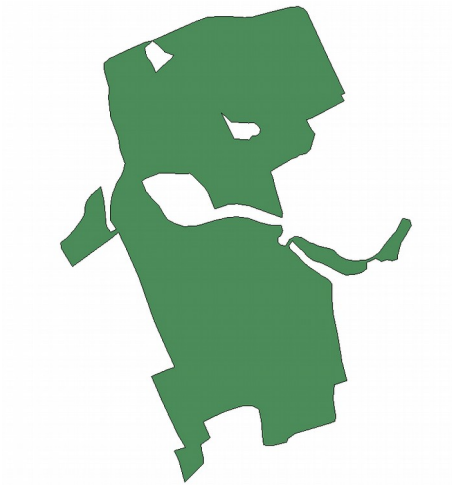
Cette description (ring-self intersection) n'est pas détectée comme une erreur par l'algorithme 'vérifier la géométrie' (méthode GEOS) de QGIS, mais comme déjà signalé, elle est pourtant incorrecte au sens de GEOS.

Une vérification sous DBManager (avec st_isvalid) permet de le vérifier :

	st_isvalidreason(geometry)	gid
1	Ring Self-intersection[156404.2188 6841721.139]	64565

Il faut d'abord corriger le polygone avec st_makevalid, pour que le trou soit bien pris en considération par l'algorithme 'fil hole'. On vérifie de nouveau ici la nécessité absolue de travailler avec des géométries valides au sens de GEOS.

Si on utilise fill holes avec un seuil de 700 on ne corrige que le trou dont la surface est inférieure :



Ceci permet d'éliminer les micro-trous internes à des entités.

Remarque : on trouvera ici des informations pour réaliser la suppression des trous en SQL et sur ce lien en anglais une solution avec la réalisation d'une fonction filter_rings :

```
CREATE OR REPLACE FUNCTION filter_rings(geometry, DOUBLE PRECISION)
  RETURNS geometry AS
$BODY$
SELECT ST_BuildArea(ST_Collect(b.final_geom)) AS filtered_geom
  FROM (SELECT ST_MakePolygon((/* Get outer ring of polygon */
    SELECT ST_ExteriorRing(a.the_geom) AS outer_ring /* ie the outer ring */
    ),  ARRAY(/* Get all inner rings > a particular area */
     SELECT ST_ExteriorRing(b.geom) AS inner_ring
       FROM (SELECT (ST_DumpRings(a.the_geom)).*) b
      WHERE b.path[1] > 0 /* ie not the outer ring */
        AND ST_Area(b.geom) > $2
    ) ) AS final_geom
         FROM (SELECT ST_GeometryN(ST_Multi($1),/*ST_Multi converts any Single Polygons to MultiPolygons */
                                   generate_series(1,ST_NumGeometries(ST_Multi($1)))
                                   ) AS the_geom
               ) a
       ) b
$BODY$
  LANGUAGE 'sql' IMMUTABLE;
```

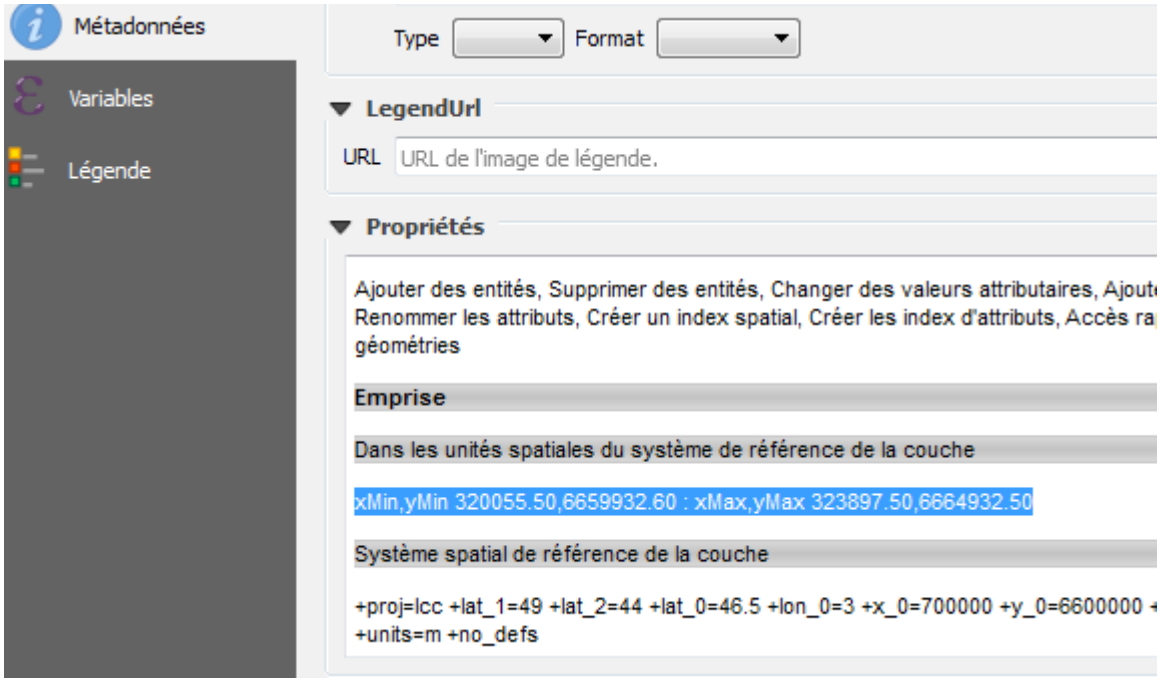
5.2.3 - La rastérisation /polygonisation

Les couches qui comportent beaucoup d'erreurs de géométries et des recouvrements sont souvent difficiles à corriger. La Rastérisation / Polygonisation permet de corriger la plupart des erreurs lors de la polygonisation, mais n'est utilisable que dans le cas où les vecteurs sont déjà issus d'un raster. Sinon, il faudrait rastériser avec une précision telle que la couche pourrait devenir inutilisable.
La méthode permet de corriger les recouvrements si besoin, en utilisant un attribut discriminant. Le cas d'une couche d'aléas de PPR Inondation, avec d'éventuels recouvrements, permet d'utiliser cette méthode.

5.2.3.a - Rastérisation

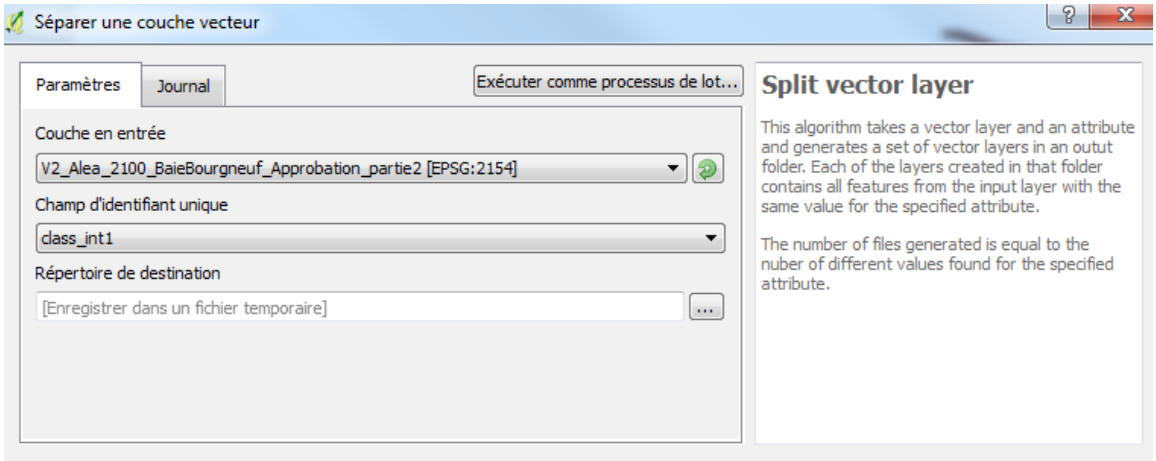
En préalable, on vérifiera que l'attribut choisi est de type numérique, sinon il faudra le convertir avec par exemple to_int(string). Cet attribut n'est utile que pour gérer les recouvrements.

Dans tous les cas, il faut noter l'emprise de (chaque) table(s) de manière à ne pas avoir de décalage en sortie. On la trouve dans l'onglet Métadonnées des propriétés de la couche :

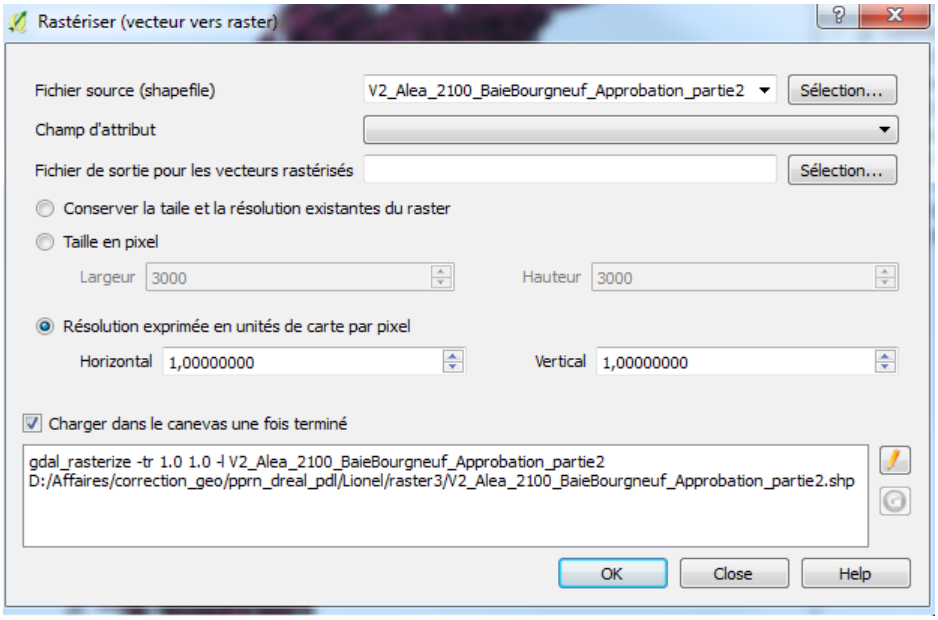


Pour gérer les recouvrements, on scinde la couche pour chaque occurrence de l'attribut discriminant.

[Vecteur/ Outil de gestion de données / séparer une couche vecteur]



- Rasteriser chaque couche. Le pixel de la couche en sortie prend la valeur du champ. [Menu Raster / Conversion / Rastériser] Il faut gérer la commande en modification pour ajouter l'étendue et le champ qui porte la valeur (si la longueur du champ >9):



Options de la ligne de commande :

-tr 1.0 1.0 (taille du pixel d'origine trouvé sur la couche vecteur. Ici, 1 mètre)

-a class_int (champ portant la valeur)

-**te 320055.50 6659932.60 323897.50 6664932.50** (étendue de la couche globale, sinon on risque d'avoir des décalages surtout avec plusieurs les couches)

-l couche (nom de la couche Shape sans extension)

couche.shp (nom de la couche Shape avec extension et chemin)

image.tiff (nom du raster avec extension et chemin)

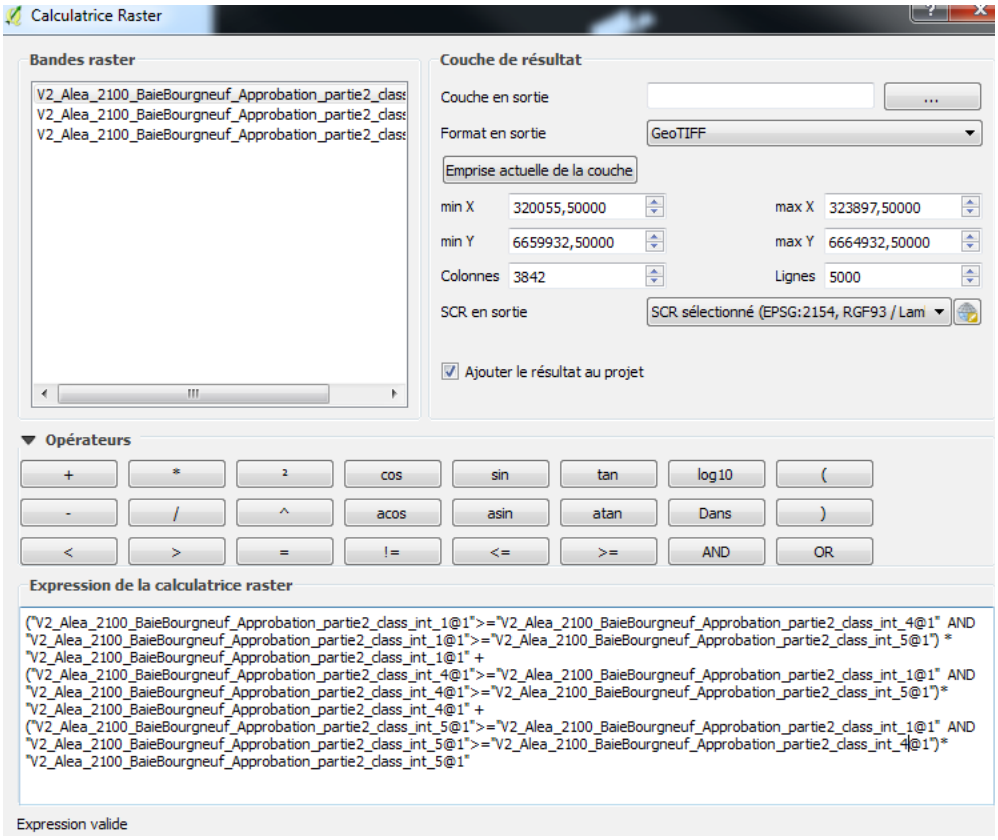
Il est plus rapide de lancer en ligne de commande les 3 commandes par : Menu Démarrer / Qgis/ OSGEO4W Shell…

5.2.3.b - Gestion des superpositions

On va créer une nouvelle couche raster qui porte la valeur maximale de chaque pixel, autrement dit l'aléa le plus fort dans le cas des PPR innondation)

Entrer dans la **calculatrice raster,** une formule du type :

```
("couche1@1" >= "couche2@1" and "couche1@1" >= "couche3@1")* "couche1@1" +
("couche2@1" >= "couche1@1" and "couche2@1" >= "couche3@1")* "couche2@1" +
("couche3@1" >= "couche1@1" and "couche3@1" >= "couche2@1")* "couche3@1"
```

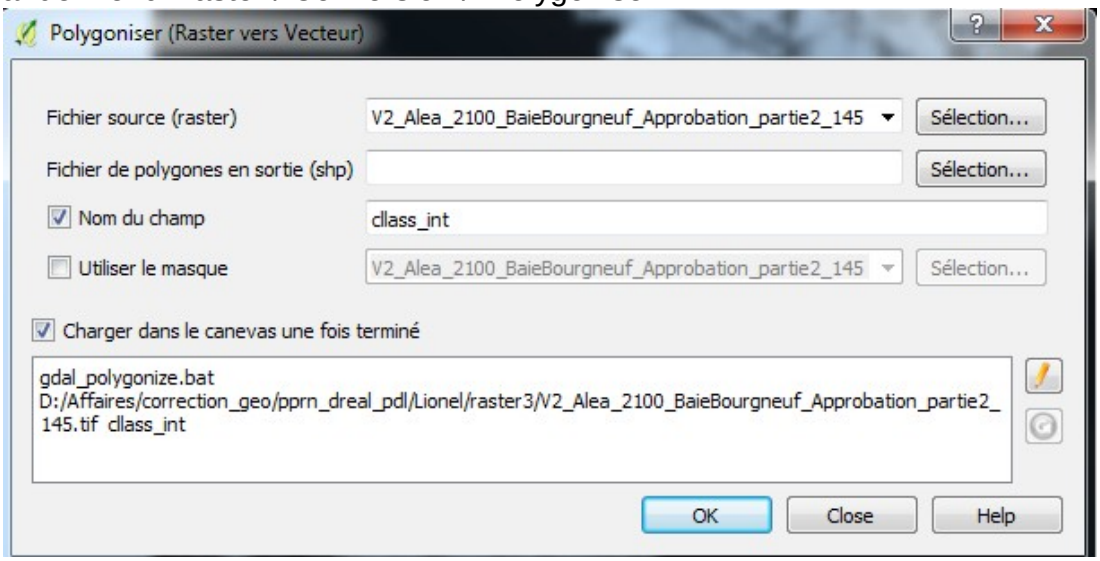


Un clic sur chaque couche copie la valeur "couche1@1" dans la calculatrice.

Ne pas oublier de spécifier l'emprise (min X, min Y, max X et max Y) et indiquer le nom de la couche en sortie.

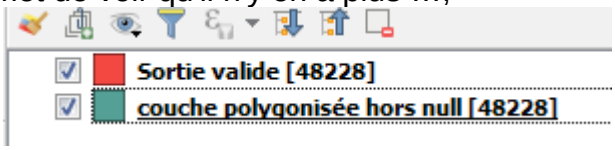
5.2.3.c - Polygoniser

Commande:Menu Raster / Conversion / Polygoniser



La couche corrigée est crée .
La rastérisation couvrant entièrement l'emprise y compris les trous, la polygonisation créé des polygones sur ces zones. On peut les supprimer après avoir sélectionné les objets avec une valeur 0 (ou nulle) pour retrouver uniquement les objets de la couche initiale .

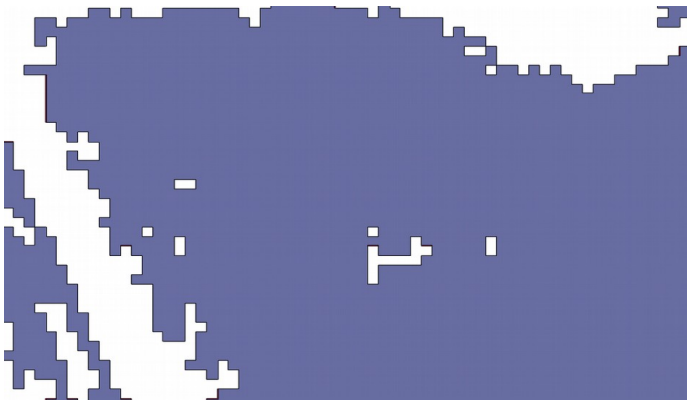
La recherche d'erreurs permet de voir qu'il n'y en a plus …,



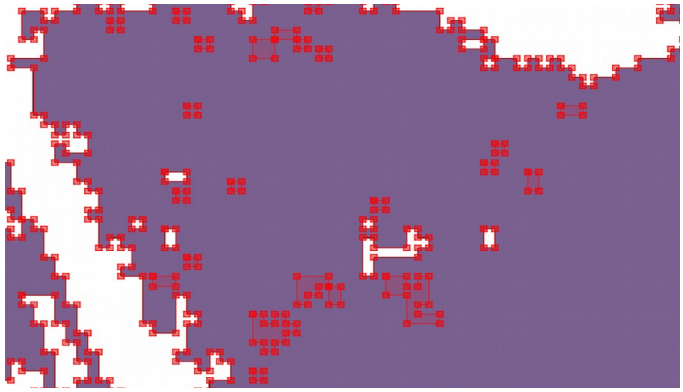
sauf pour le cas ci-dessous .

Sur certaines zones particulièrement complexes, on peut trouver des incohérences avec la couche d'origine, dues à la présence de trous imbriqués (holes nested) . Ceux-ci sont transformés en vrais trous alors qu'ils ne devraient pas.

Couche d'origine

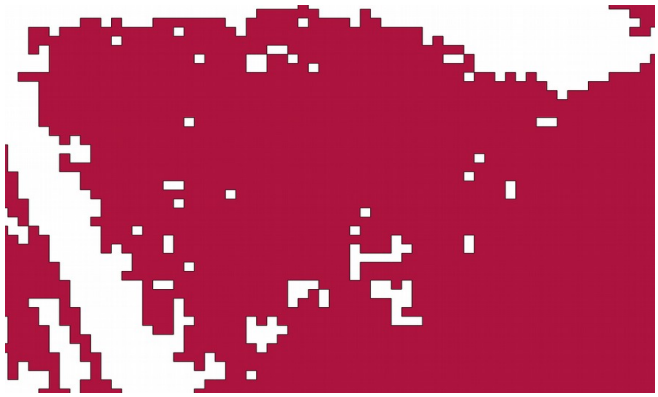


Affichage des nœuds montrant les trous imbriqués



Il faut alors commencer par décomposer les objets de multiples en uniques pour éviter ce phénomène.

Résultat par rastérisation sans décomposition des objets. Les trous imbriqués apparaissent



Résultat par rastérisation avec décomposition des objets. Les trous disparaissent

Il est intéressant de décomposer (st_dump ou morceaux multiples vers morceaux unique) les couches avant tout traitement.

Par contre, l'assemblage des polygones n'est pas performant sous Qgis (dissolve) et plante sur des grandes surfaces.

Il y a deux façons de gérer ce problème :

- a priori, en décomposant uniquement les objets avec des trous imbriqués et les ré-assemblant après la polygonisation.
- à posteriori, grâce à l'outil web Mapshapper qui permet d'assembler les polygones très rapidement,

5.3 - Corrections topologiques sur une couche

Il s'agit de résoudre, lorsque c'est possible, des problèmes de non-respect des contraintes topologiques, propre à une couche, imposées par le cahier des charges. Attention les corrections de topologie modifient les objets. Les corrections automatiques peuvent ne pas être pertinentes.

5.3.1 - Correction des interstices

Une contrainte souvent posée est de disposer d'une couverture en continuité géométrique du territoire. Dans ce cas il ne doit pas y avoir de trous (interstices) entre les objets au sein d'une même couche.

La détection des interstices peut se faire, sous QGIS, par le plugin '**vérificateur de topologie**' avec la règle '*ne doit pas avoir de trou*', toutefois cet outil rapide ne permet pas de fixer des seuils de tolérance.

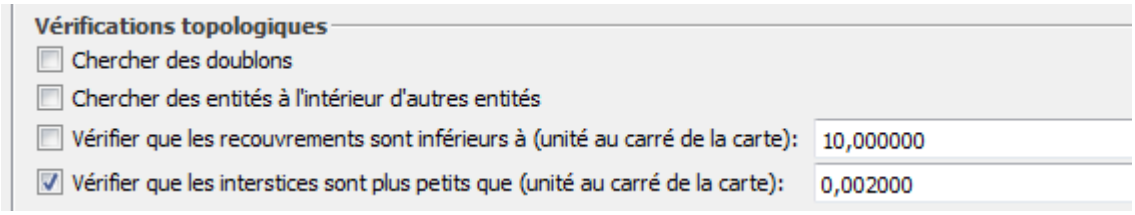
Correction manuelle

S'il n'y a pas trop de polygones en cause, on peut remodeler manuellement les entités avec un accrochage objet sur les sommets et segments du polygone voisin.

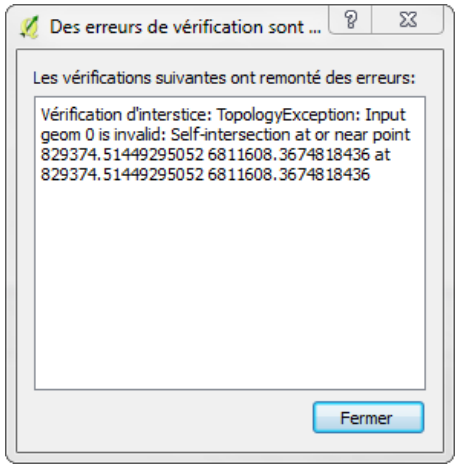
Le plugin vérifier les géométries avec la condition

'vérifier que les interstices sont plus petits que' permet de fixer un seuil.

Sur une couche test 'N_ZONE_URBA_52088_52' nous testons avec un seuil de 0,002.

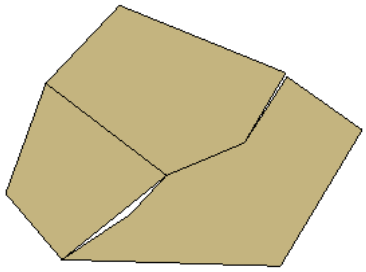


le plugin affiche un message d'erreur :

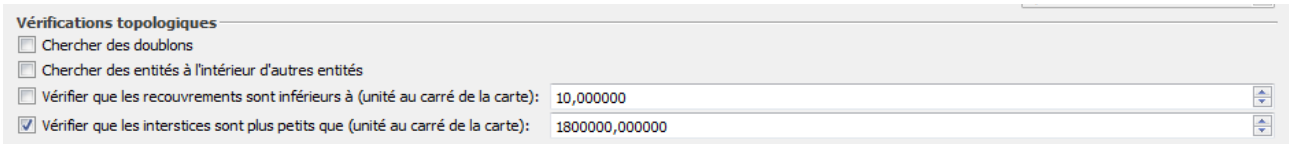


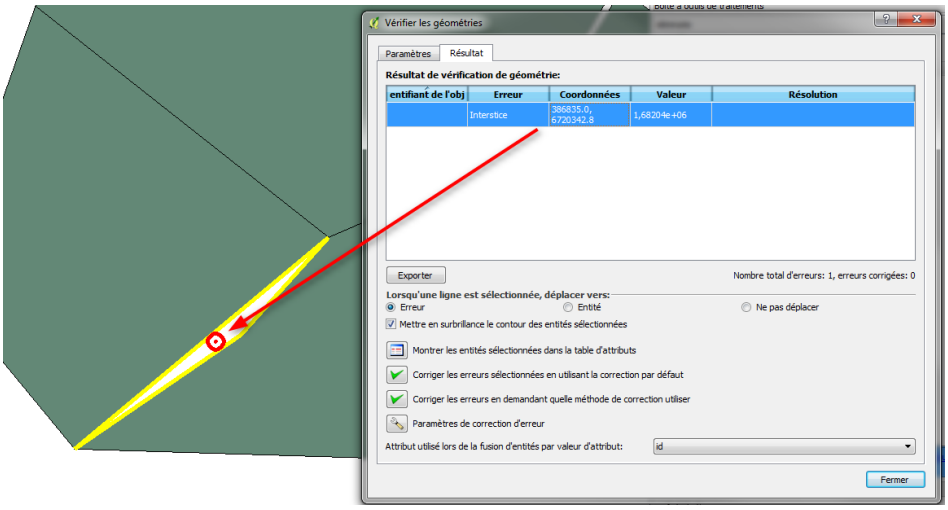
Alors que la couche est valide et qu'il n'y a aucune erreur sur ce sommet. Nous en concluons que ce plugin n'est pas encore fiable à 100 %. Il provoque d'ailleurs régulièrement des minidumps de QGIS.

Sur une couche très simple constituée pour l'occasion (couchea):

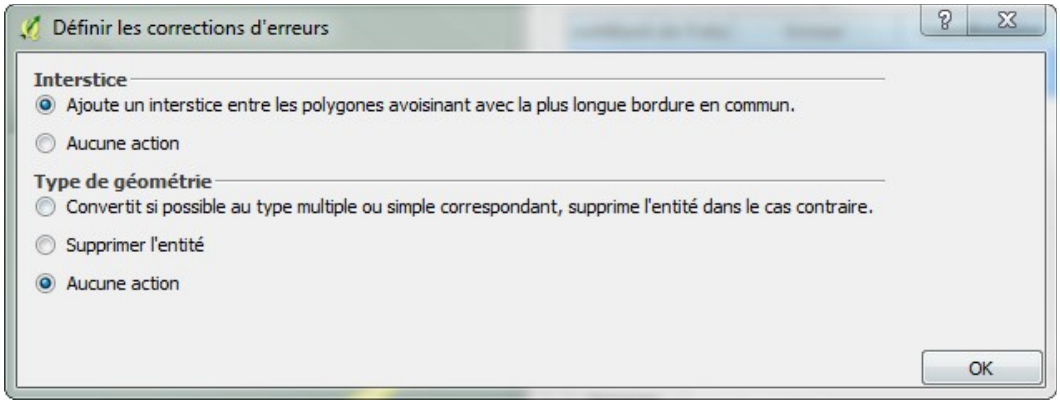


'Vérifier les géométries' détecte l'interstice (qui a une surface de 1 683 000 m2)

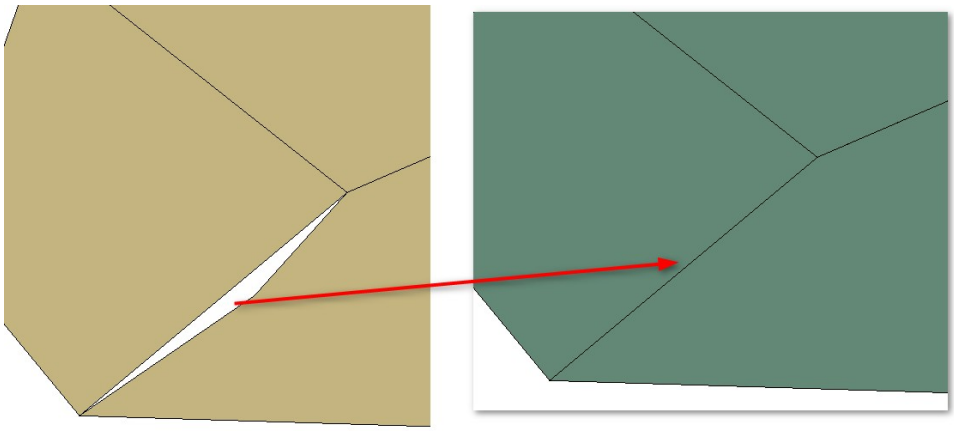




La correction proposée par défaut est :



Ce qui signifie que le polygone qui a la plus longue bordure commune va être fusionné avec un polygone recouvrant l'interstice pour remplir le trou :



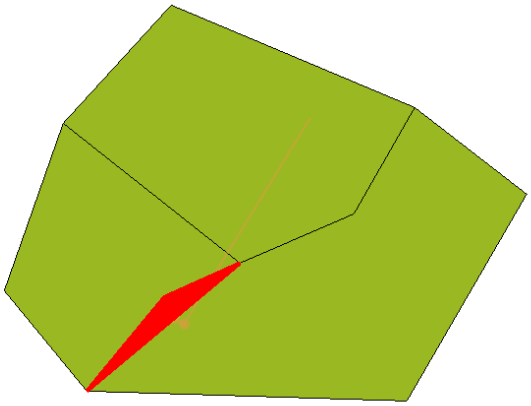
Une méthode permettant de convertir les trous en polygones est donnée ici

Ce qui permet de se ramener au cas d'élimination des polygones fins (sliver polygones).

5.3.2 - Correction des recouvrements (overlap)

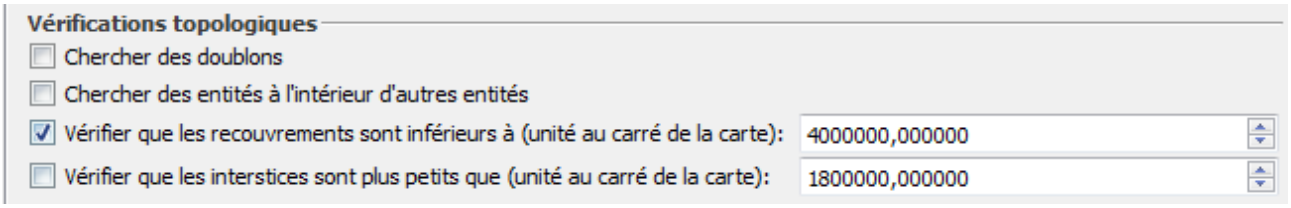
On impose avec cette condition qu'aucun polygone d'une couche ne recouvre partiellement ou totalement un autre polygone de la couche.

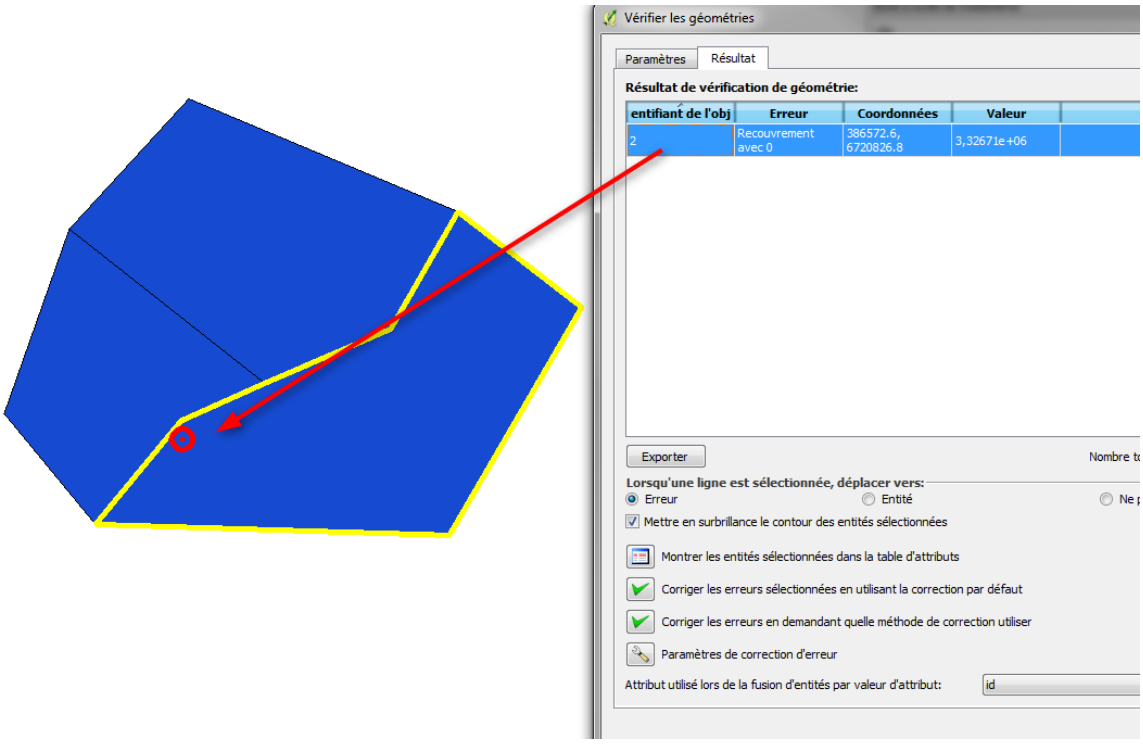
La détection des recouvrements peut se faire, sous QGIS, par le plugin '**vérificateur de topologie**' avec la règle 'ne doit pas se superposer', toutefois cet outil rapide ne permet pas de fixer des seuils de tolérance.



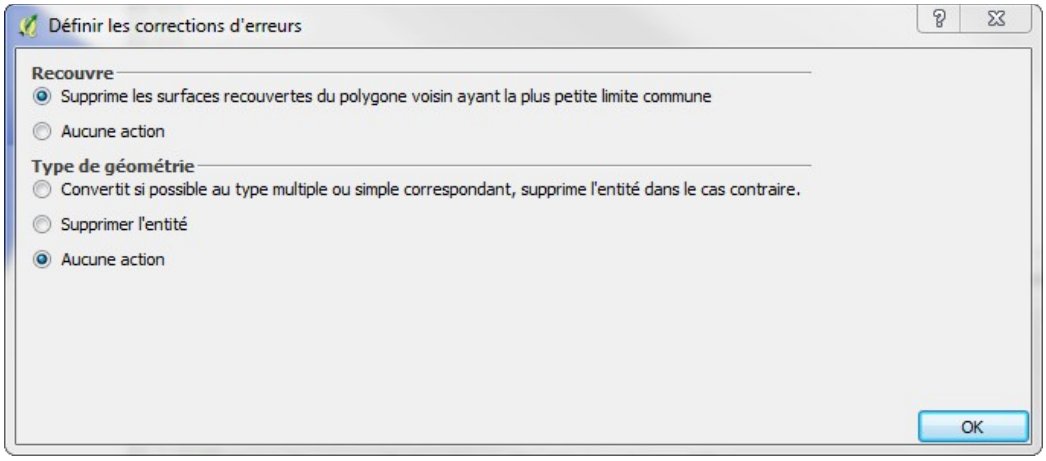
Le plugin ‘vérifier les géométries’ avec la condition

‘*vérifier que les recouvrements sont inférieurs à*’ avec un seuil.

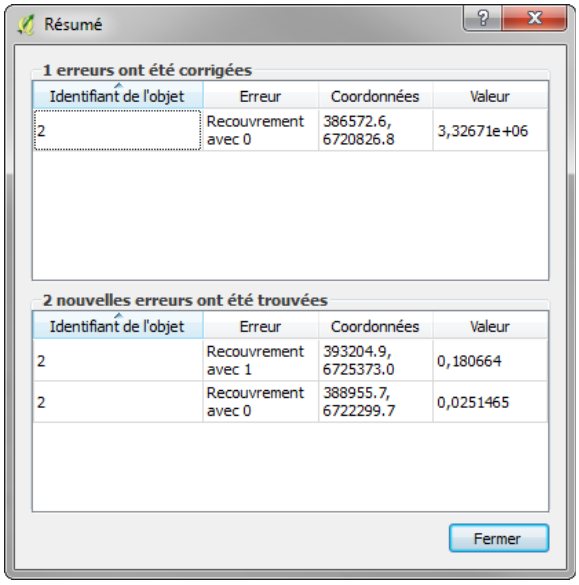




La correction proposée par défaut est :

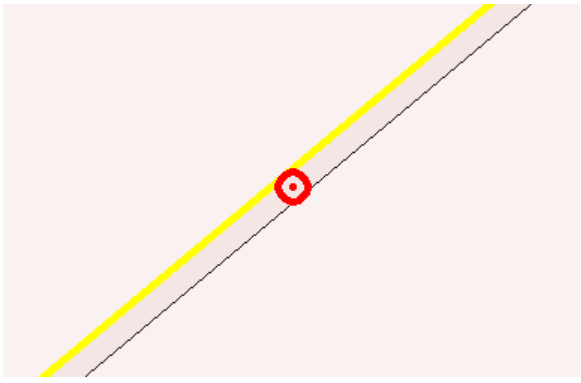


le correcteur indique cependant, dans notre cas, que la correction a créé deux nouvelles erreurs avec d'autres polygones voisins :



Malheureusement ce plugin conduit assez fréquemment à des minidumps. Dans notre cas il ne faut pas tenir compte immédiatement des deux nouvelles erreurs et valider pour relancer.

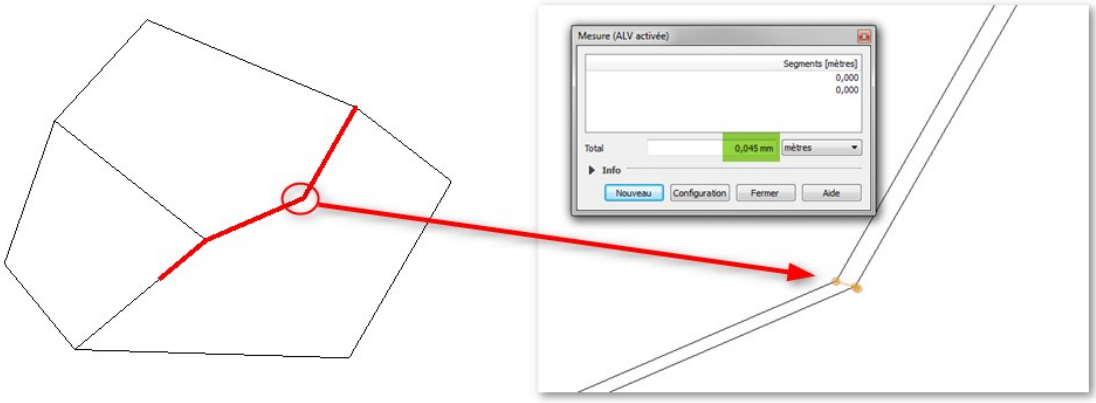
La correction à conduit à deux légers recouvrement qui ne sont visibles qu'avec un zoom très important :



Si on tente de corriger à nouveau on obtient un message d'échec :

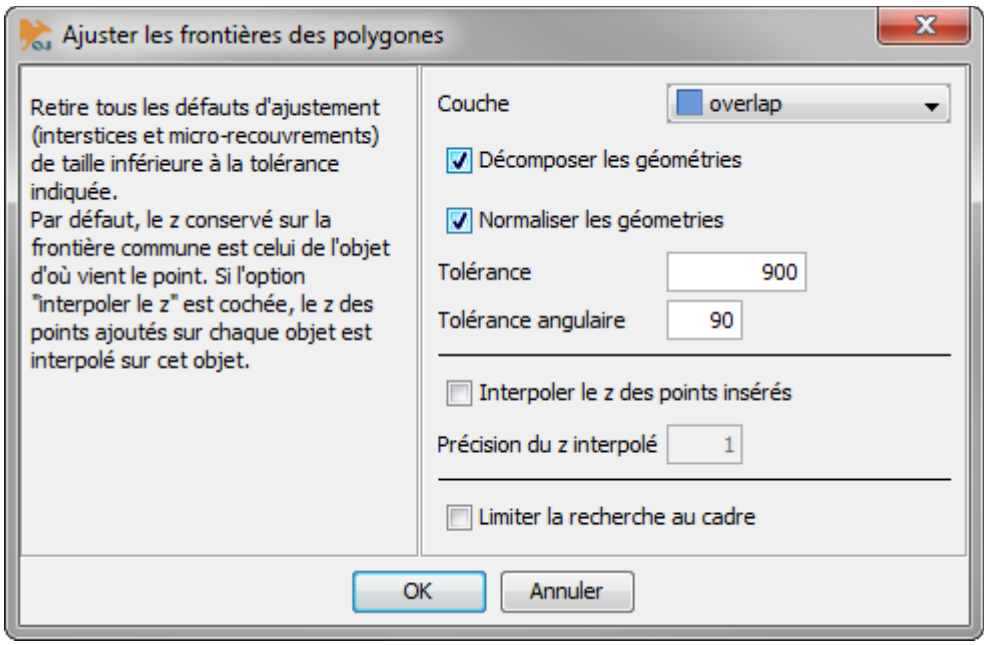
entifiant de l'obj	Erreur	Coordonnées	Valeur	Résolution
2	Recouvrement avec 0	388955.7, 6722299.7	0,0251465	Corrections échouées: Impossible de trouver des limites communes entre l'intersection et les entités recouvertes
2	Recouvrement avec 1	393204.9, 6725373.0	0,180664	

de fait, il y a toujours un léger chevauchement (0,045 mm) qui est détecté par vérificateur de topologie et qui ne peut être corrigé par ce plugin :

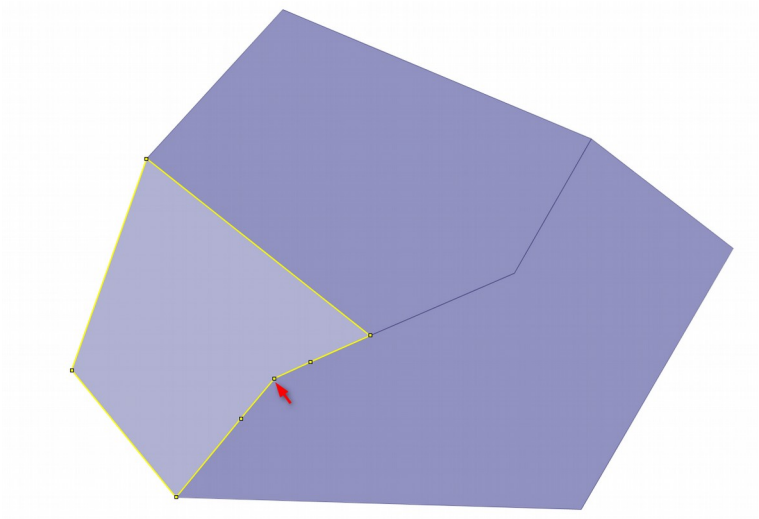


Correction avec l'**extension 'Topologie → Ajuster les frontières des polygones de Openjump'**

Cette extension permet la correction des recouvrements :



Dans notre cas on donne des valeurs élevées car notre erreur est grossière. Le résultat modifie le polygone comme indiqué ci-dessous :



La couche corrigée ne présente plus d'erreur sous QGIS.

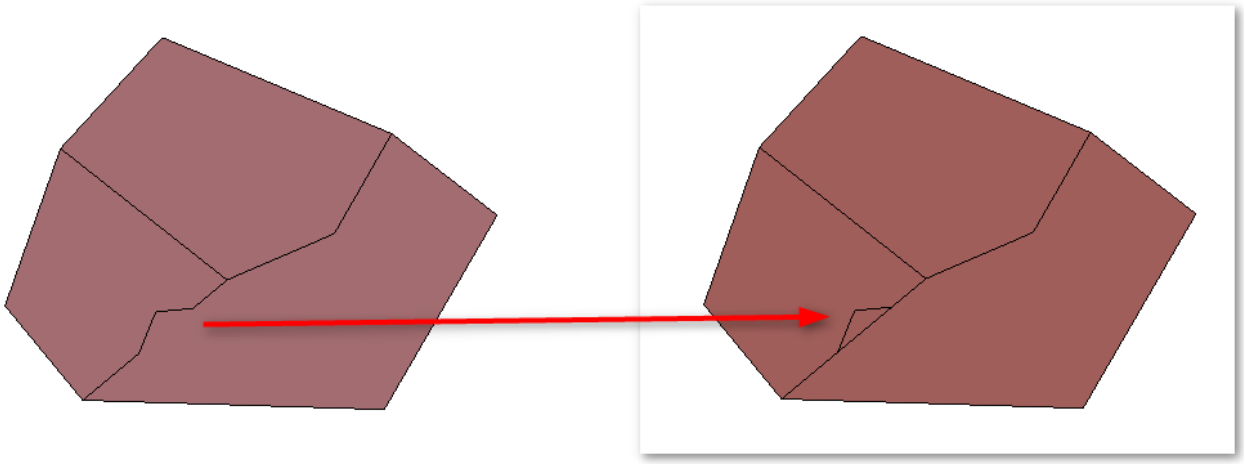
GRASS sous QGIS:

Il existe une solution alternative avec les algorithmes de GRASS :

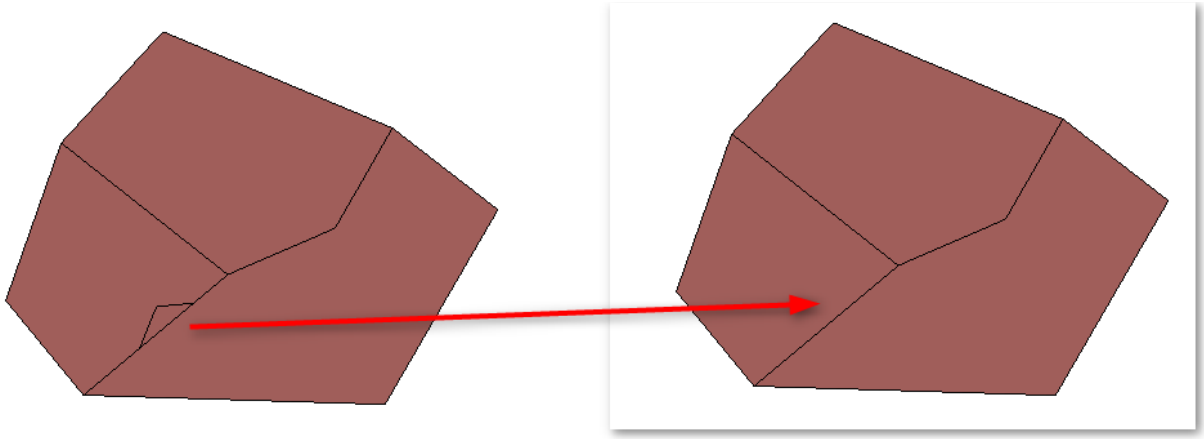
(attention avec GRASS le chemin ne doit pas contenir de caractères accentués)

la démarche est la suivante :

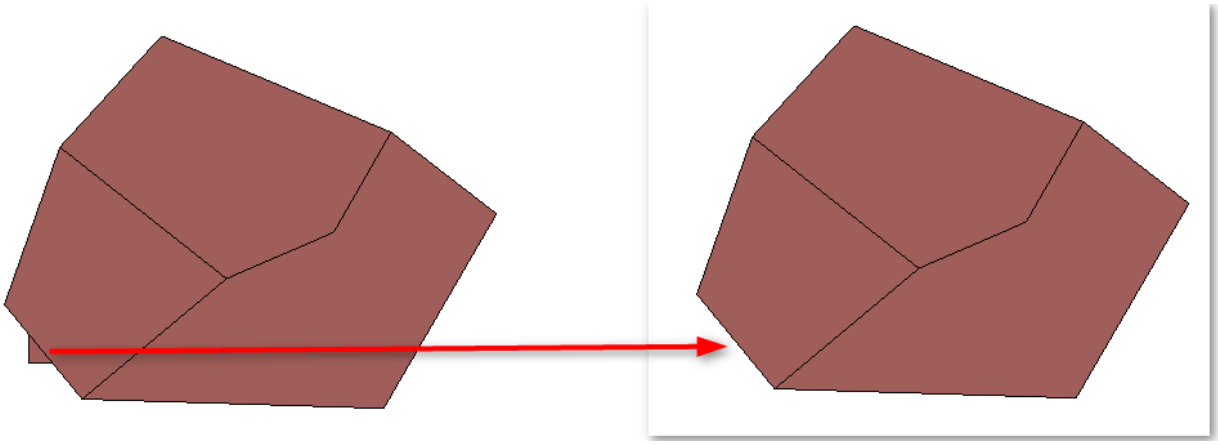
v.clean option bpol (le paramétre treshold n'est pas utilisé) :



v.clean option rmarea (suppression des petits polygones -threshold indique la valeur maxi pour supprimer les polygones en m2). Fusionne le petit polygone créé par bpol au voisin avec lequel il partage la plus longue frontière.



Attention toutefois car le rmarea fusionne avec l'objet avec le périmètre commun le plus grand et parfois celui-ci est du côté où il n'y a pas de polygone donc on peut avoir une correction incorrecte, exemple :

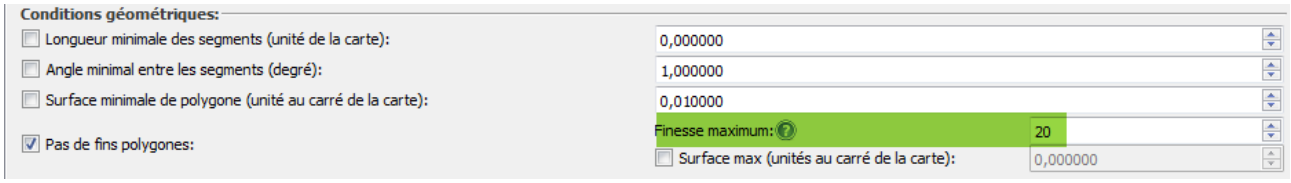


.

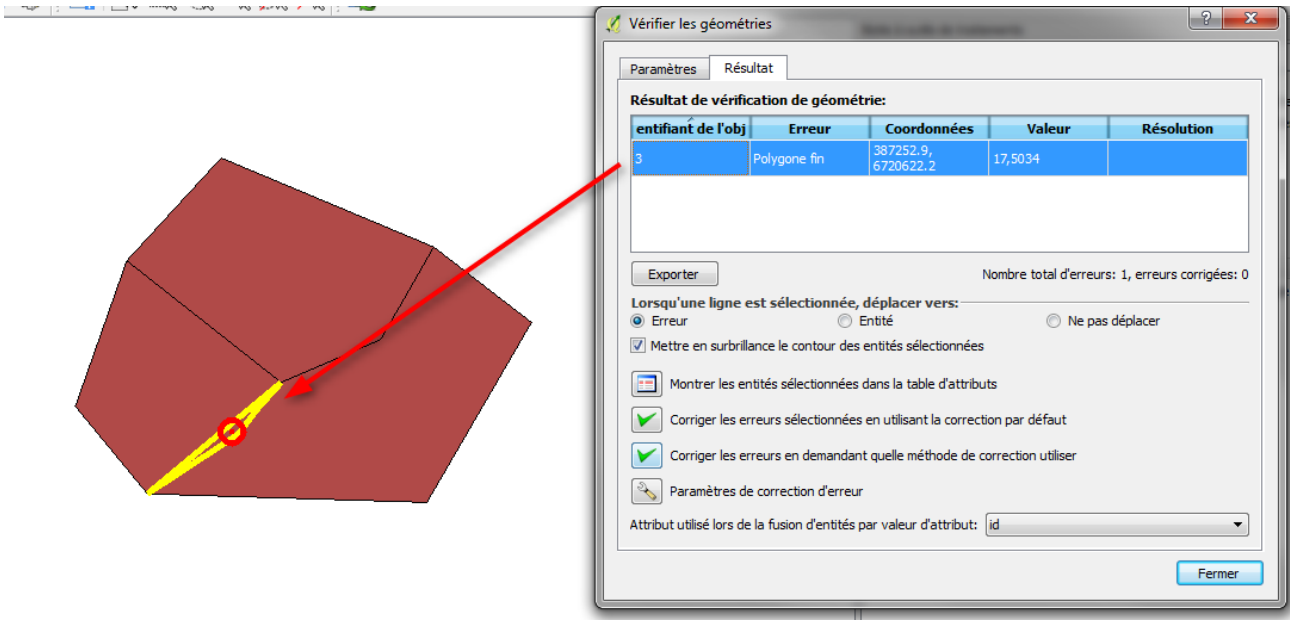
5.3.3 - Corrections des polygones fins

Le terme ‘sliver polygon’ en anglais devrait être traduit par ‘micro-zone’ ou ‘zone parasite’ en français d’après le glossaire de l’ENSG. ESRI utilise le terme ‘micro-polygone’. Cette traduction ne prend pas en compte l’aspect ‘ruban’ ou lamelle qui est celle du terme ‘sliver’. La traduction utilisée par le plugin correction de géométrie de QGIS de ‘*polygone fin*’ semble meilleure. Elle prend en compte la notion de finesse qui est le ratio entre la surface du carré minimum qui contient la géométrie (rectangle d’encombrement) et la surface du polygone.

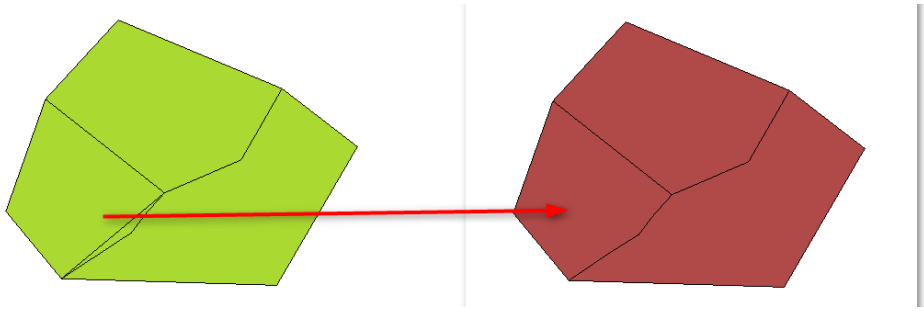
Pour régler le seuil de finesse, il convient de faire des essais :



exemple : Dans ce cas d'école nous réglons la finesse à 5 :

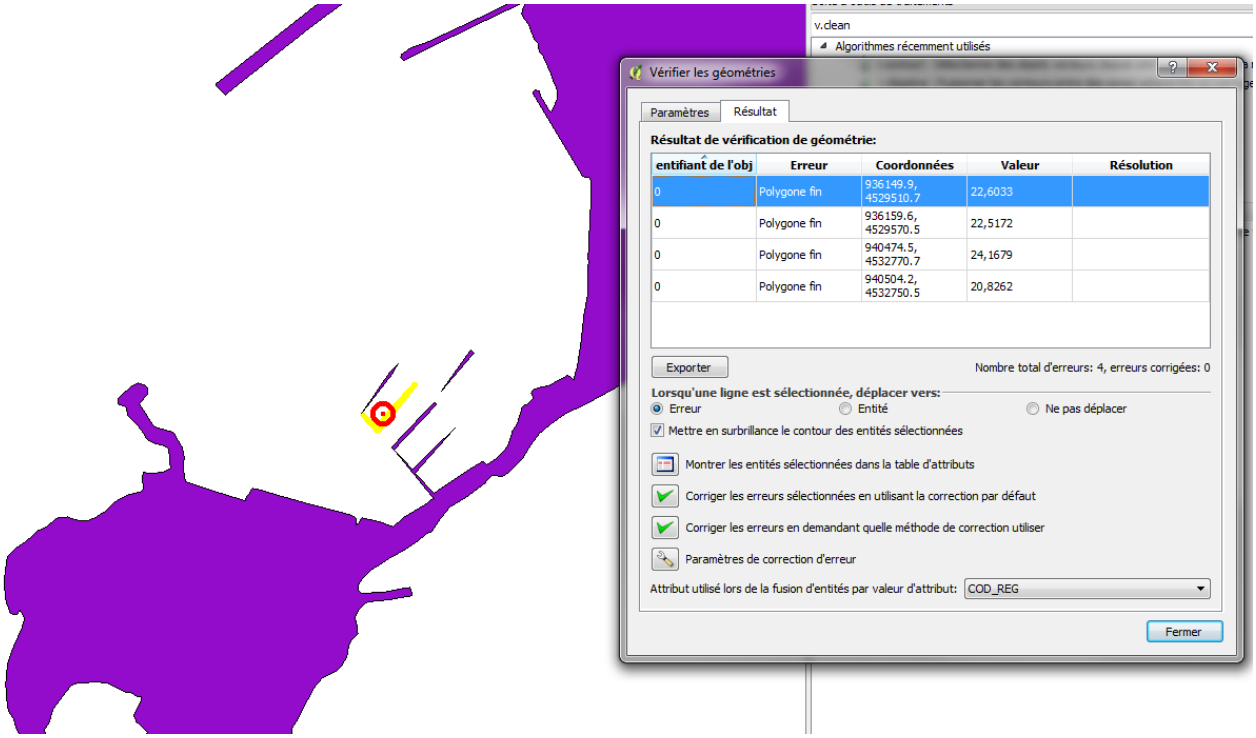


et nous pouvons demander la correction automatique avec la méthode par défaut :



Attention à utiliser le paramètre de finesse et les corrections avec prudence. Il peut exister des polygones très fins légitimes dans une couche.

Exemple :



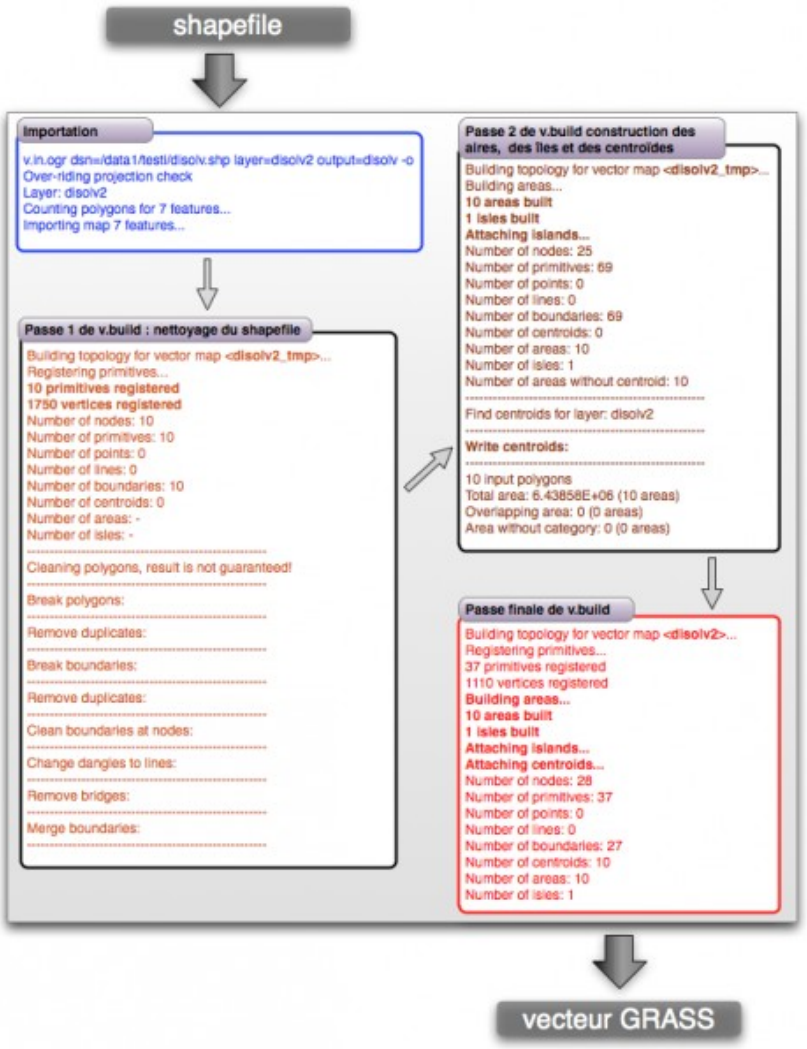
On détecte ici des polygones fins avec un ratio de finesse > 20, il faut vérifier dans les spécifications de saisie si ces polygones qui représentent visiblement des jetées sont corrects ou non.

5.3.4 - Correction avec GRASS (v.in.ogr et v.build)

GRASS est un logiciel puissant mais assez complexe à appréhender. Ce n'est pas la solution à envisager en première intention et nous ne traiterons pas de toutes ses possibilités. GRASS utilise un modèle topologique et il doit donc construire cette topologie lors de l'import d'une couche.

Voir par exemple un extrait de la traduction du manuel en français.

On trouvera ici une description des principales règles de topologie de GRASS, ainsi que la description du traitement effectué lors de l'import :



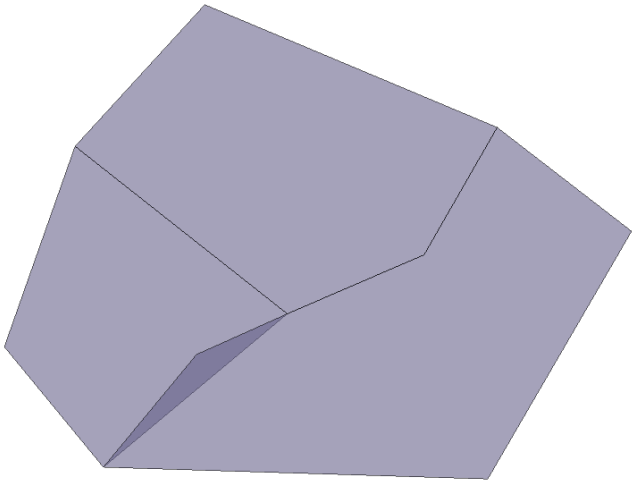
en complément on pourra lire (en anglais) le [Wiki sur le nettoyage de topologie]

On retiendra en particulier la possibilité de considérer les options (snap, bpol, rmdupl, break, rms) de v.clean.

Lors d'une importation sous GRASS, le module v.in.ogr **v.in.ogr** va essayer de construire une topologie avec le module **v.build** (nous allons voir les paramètres snap et min_area).

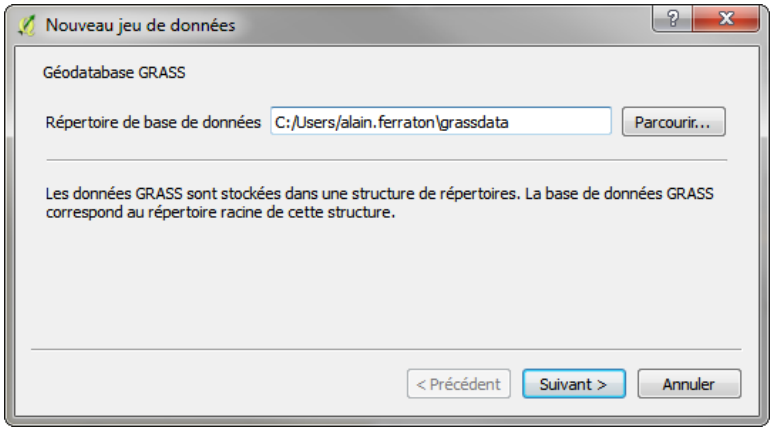
Pour corriger les erreurs on peut tenter v.clean pour les corrections automatiques (nous avons déjà vu les options *bpol* et *rmarea* au paragraphe 5.3.2).

Nous prenons comme exemple une couche *overlap* qui contient un recouvrement.

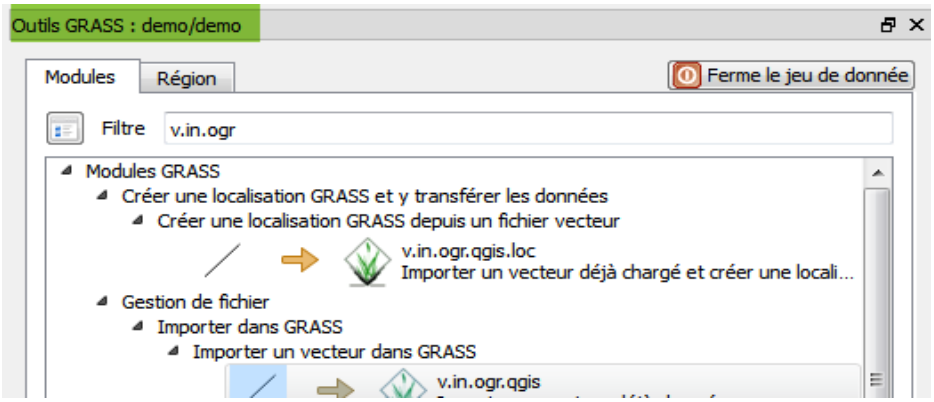


v.in.ogr n'est pas explicitement disponible dans la boîte à outils de traitement de QGIS. Pour pouvoir l'utiliser nous créons un nouveau jeu de données GRASS :

Extension → GRASS → Nouveau jeu de données :

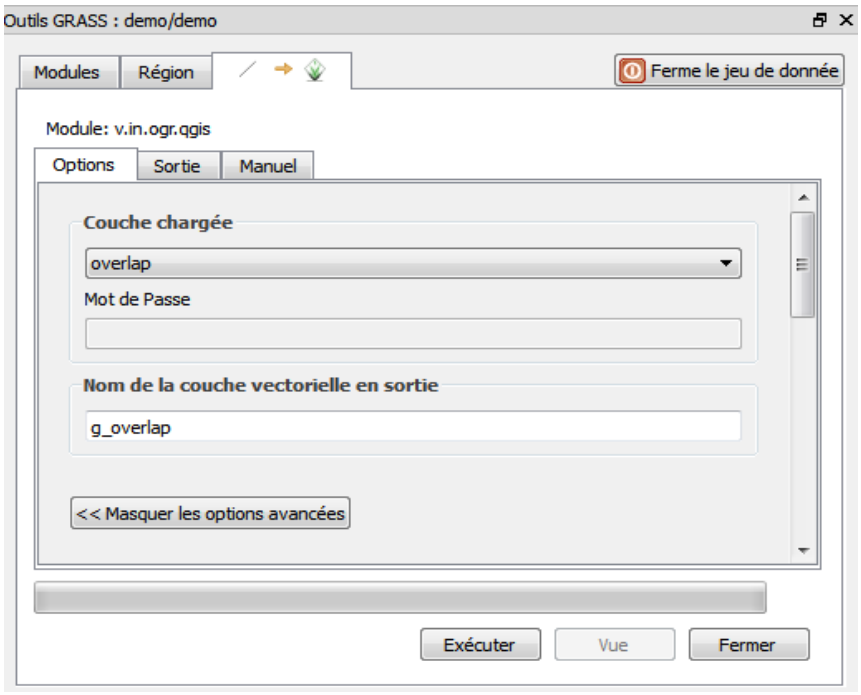


Puis créer un nouveau *secteur* et un nouveau *jeu de données.* Puis ouvrir le jeu de données. La boîte 'Outils GRASS' doit afficher (au besoin : *Extension → GRASS → ouvrir les outils GRASS*) :



du coup on peut utiliser **v.in.ogr.qgis** dans les outils de GRASS.

Réaliser une première importation sans options avancées :



Le rapport (onglet sortie) indique en particulier :

Some input polygons are overlapping each other.

If overlapping is not desired, the data need to be cleaned.

The input could be cleaned by snapping vertices to each other.

Estimated range of snapping threshold: [1e-008, 1]

Try to import again, snapping with at least 1e-008: 'snap=1e-008'

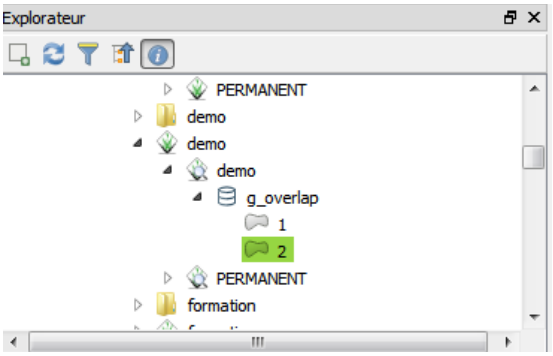
Un peu plus haut dans le rapport on trouve :

Overlapping area: 3.32671E+006 (1 areas)

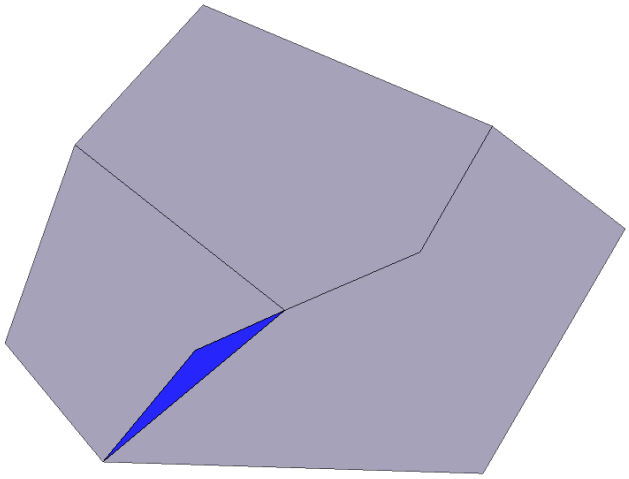
Il y a donc 1 polygone en recouvrement et GRASS nous propose de ré-importer avec un seuil compris entre 10^{-8} et 1.

Le bouton *Vue* en bas de la fenêtre permet de visualiser la couche1 (elle compte toujours 3570 objets).

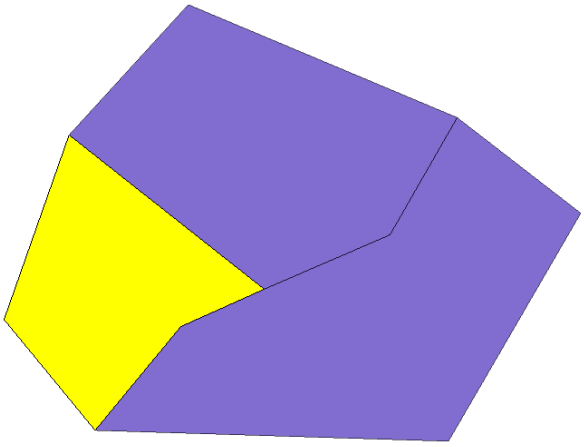
On peut charger les différentes couches de GRASS en passant par l'explorateur de QGIS positionné sur le répertoire de la géodatabase de GRASS.



On peut charger avec clic droit ‘ajouter la couche’ la couche 2 qui représente les surfaces en recouvrement. Il y a 1 polygone dans cette couche (en bleu) :

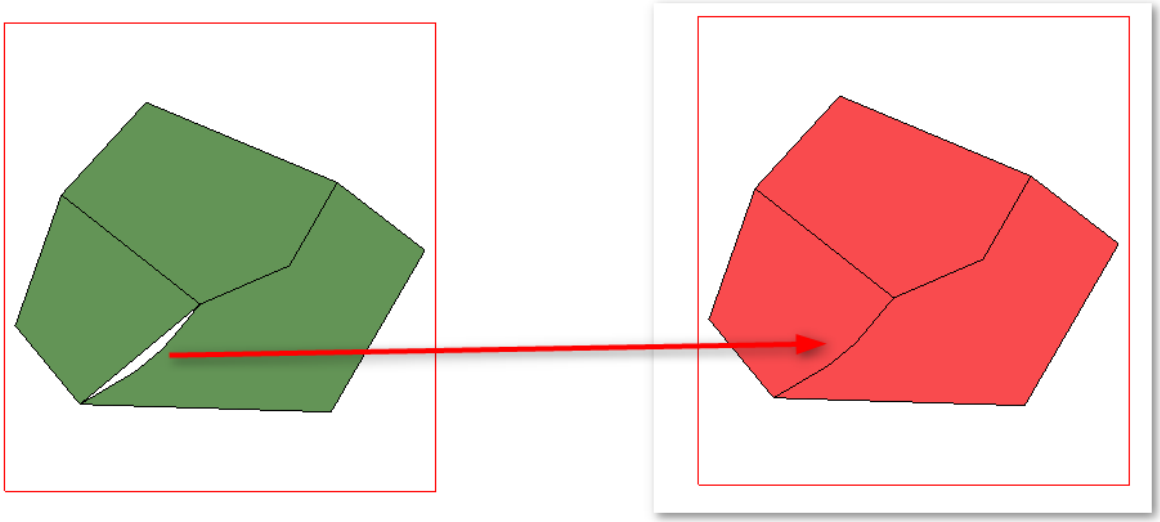


Ré-importons la couche avec le paramètre *snapping threshold of boudaries* (map units) fixé à 1500 (plus grande largeur approximative du polygone bleu) , puis visualisons le résultat (bouton Vue) :



Il n’y a plus de superposition. Le polygone en jaune a été modifié. La topologie est maintenant correcte, cependant on ne maîtrise pas la façon dont GRASS effectue son accrochage.

En effet l'option *snap* demande de réaliser un accrochage de sommet à un autre sommet le plus proche dans la limite du seuil donné en unité de carte. Ainsi l'option *snap* corrige également les interstices :



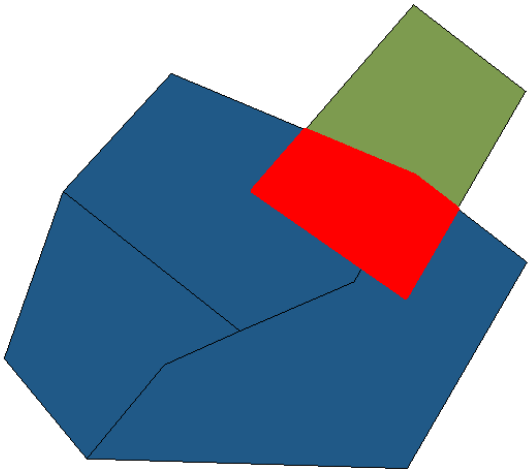
Il nous semble préférable d'identifier correctement les problèmes, puis de tenter les options de v.clean pour tenter de corriger les erreurs, Voir [la documentation de GRASS]

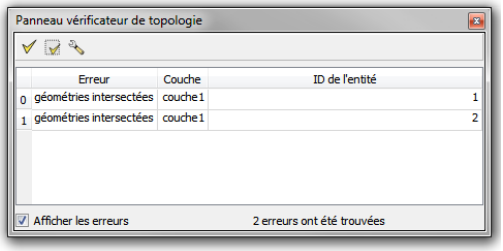
5.4 - Corrections de topologies entre couches

On peut souhaiter gérer des conditions de topologie avec une autre couche.

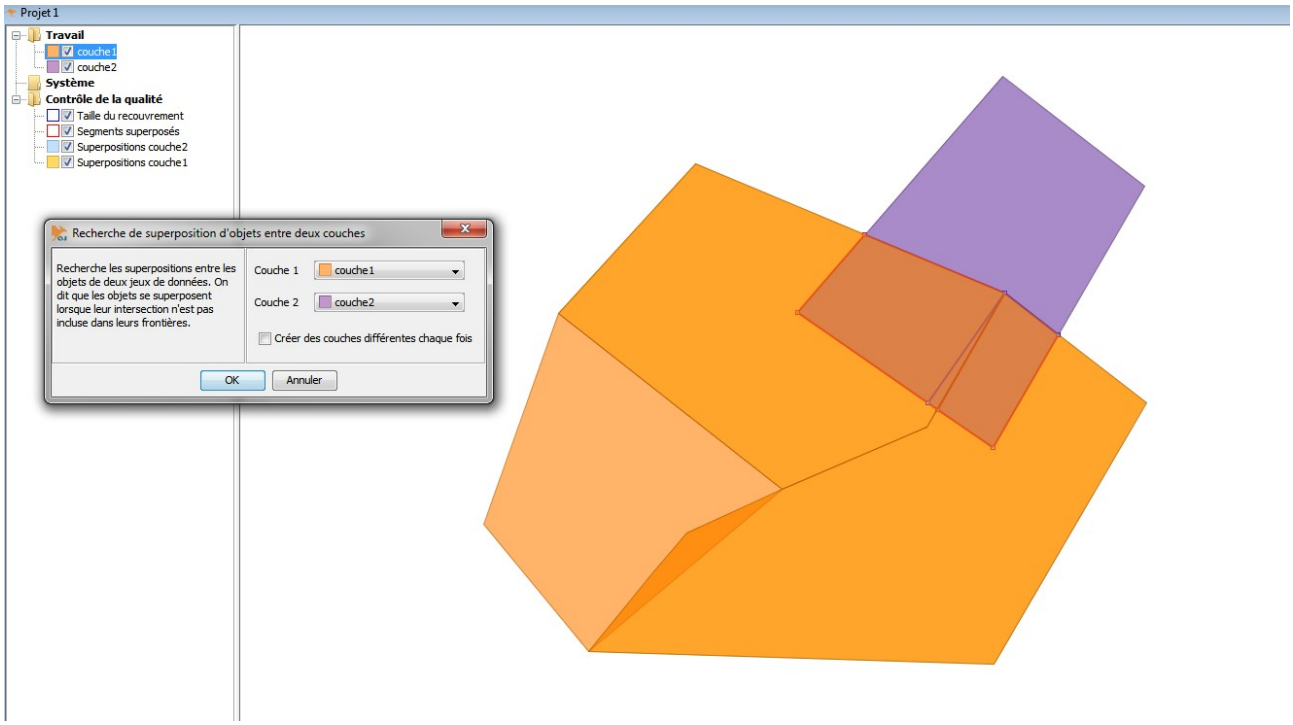
5.4.1 - Superposition entre deux couches

Le plugin **vérificateur de topologie** permet de détecter que les objets d'une couche ne doivent pas se superposer aux objets d'une autre couche.





Openjump permet également de rechercher ‘des superpositions d’objets entre deux couches’ et va créer une couche montrant la taille des recouvrements et une autre les segments superposés



Une autre façon de faire est de créer la couche d’intersection entre les deux couches :

QGIS : Vecteur → outils de géotraitement → intersection

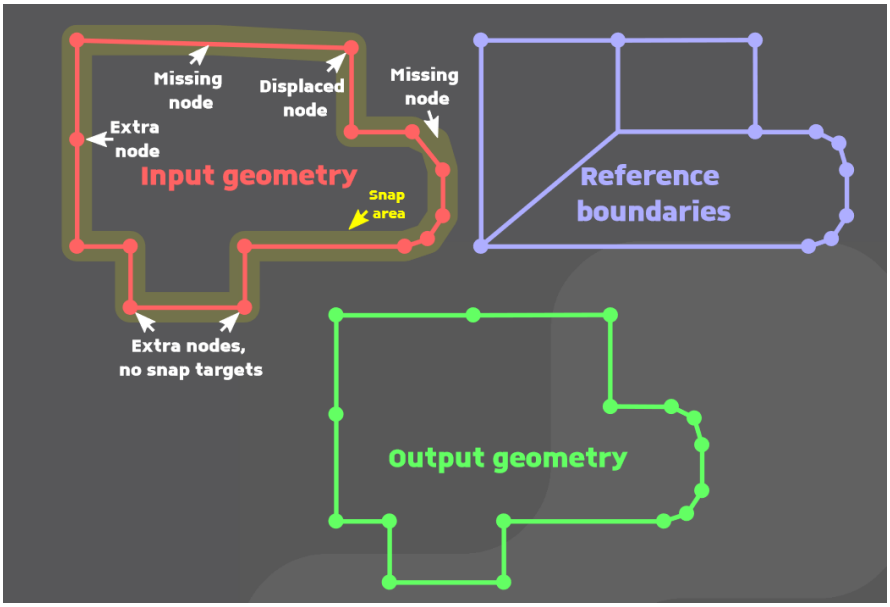
Il semble difficile de corriger automatiquement ce type de contraintes. Cependant il existe, comme nous allons le voir juste après, des solutions d’accrochage (appariement) sur une géométrie existante.

5.4.2 - Accrochage de géométries

QGIS propose un plugin d’accrochage de géométries (geometry snapper).

(sera proposé sous forme d’algorithme dans QGIS 3.0).

Ce plugin permet de modifier des objets d’une couche de polygones en conservant les attributs selon l’algorithme suivant :



Pour les nœuds qui sont aux mêmes emplacements dans les deux géométries -> pas d'action.

Pour un nœud qui ne se trouve pas exactement au même emplacement (*Displaced node* dans la figure) , mais dans le rayon de recherche défini dans la fenêtre de paramétrage (*Snap area* de la figure)-> déplacement du noeud de la géométrie à corriger pour le situer exactement à l'endroit de la géométrie de référence.

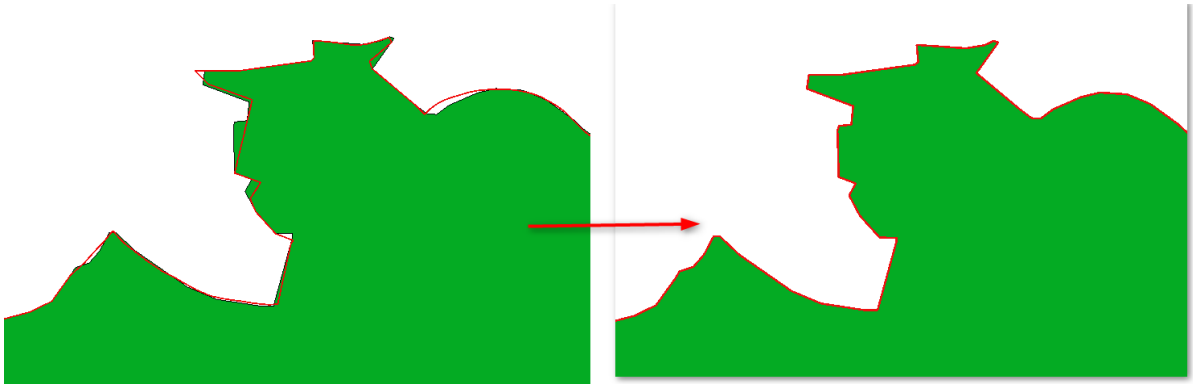
Pour un nœud présent dans la géométrie à corriger mais qui n'a pas de correspondance dans la géométrie de référence dans le rayon de recherche défini dans le paramétrage (*Extra node*) -> pas d'action, les nœuds seront gardés dans la géométrie corrigée.

Pour les nœuds de la couche de référence qui n'ont pas de correspondance dans la géométrie à corriger (Missing node), mais qui se situent dans la zone de recherche (Snap area) -> ajout des ces points dans la géométrie corrigée. (source : Blog Sig&territoires)

Exemple :

Contour de la commune de La Flèche de la Bdcarto en rouge sur polygone de la Bdtopo

Malheureusement, sous QGIS 2.16 l'utilisation de ce plugin rend QGIS instable (minidump très fréquent après utilisation).



Note : Sous **Openjump**, il existe un plugin d'appariement (matching plugin). La version 1.11 rev 5434 du 13 avril 2017 inclus le plugin '*matching*').

La documentation (en français) du plugin peut-être trouvé ici

Son objectif n'est pas le même que l'accrochage de géométrie de QGIS. Il s'agit en réalité de réaliser une jointure spatiale avec une couche de référence, mais avec des critères de comparaison avancés (voir la documentation). Les sommets de la couche initiale ne sont pas modifiés, mais certains objets répondant aux conditions seront appariés avec ceux de la couche de référence. Il est possible de gérer les transferts d'attributs.

6 - Exemples

6.1 - Couche ‘tempo.SHP’

6.1.1 - Contexte

Jeux de données fourni par Alain FELER – DTTM 29.

(couche tempo.shp).

6.1.2 - Phase 1 : Analyse de la validité du jeu de données

La couche tempo.shp compte 849 polygones.

Analyse par l’algorithme **Check validity** avec la méthode **GEOS** : 730 erreurs sur 730 polygones :

- 9 polygones en ‘duplicate rings’.
- 1 ‘ring self intersection’.
- 720 polygones en ‘self-intersection’

Avec la méthode **QGIS** : 694 erreurs sur 390 polygones (entrecroisement et polygone à l’intérieur de polygones) :

- 7 polygones en ‘duplicate rings’
- 33 ‘ring ‘self intersection’
- 350 ‘self-intersection’

Il y a des polygones qui ont plusieurs erreurs (la méthode QGIS ne s’arrêtant pas à la première erreur détectée)

SQL sous Dbmanager :

select st_isvalidreason(geometry) as erreur, from tempo where not st_isvalid(geometry) order by erreur*

Donne 849 polygones en erreurs (tous) :

- 3 ‘Duplicate rings’
- 120 ‘ring self-intersection’
- 726 ‘self-intersection’

Analyse sous **Open jump**

OpenJump (propriété de la couche) indique qu’il y a 385 multipolygones et 464 polygones)

Le contrôle donne 849 polygones en erreur :

- 3 ‘duplicate rings’
- 120 Ring Self-intersection
- 726 ‘self-intersection

(La même chose que la requête sous DbManager)

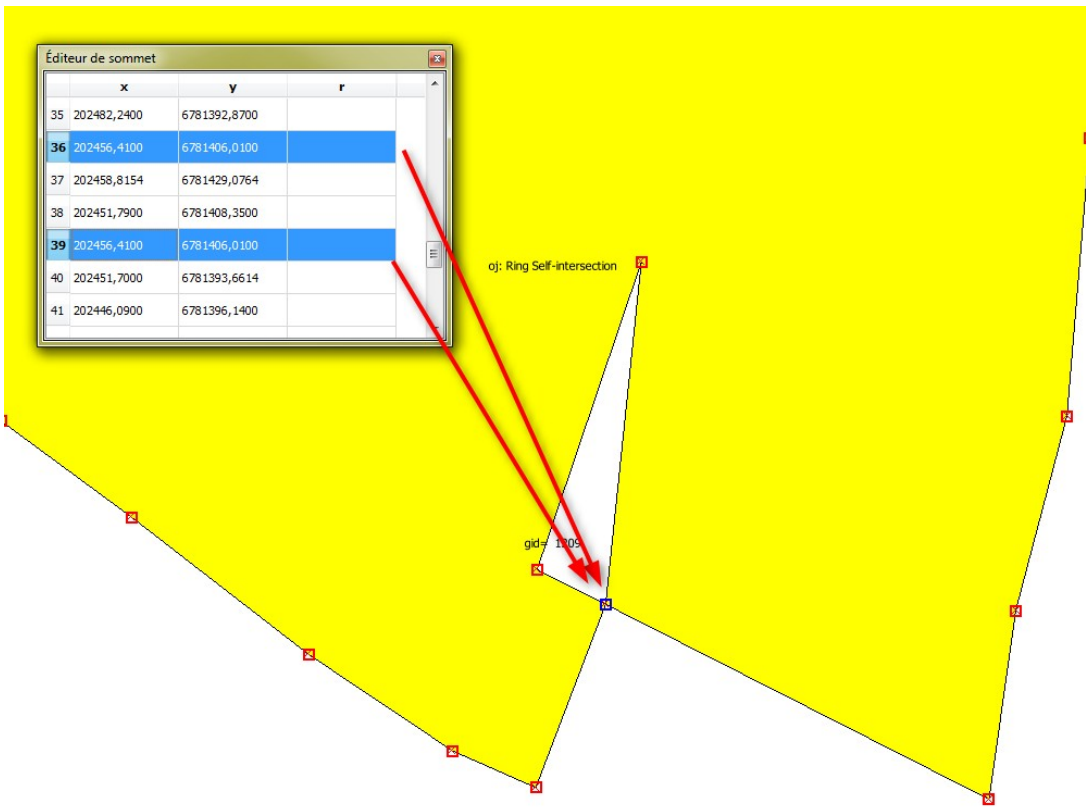
Pour un ADL les deux dernières méthodes (SQL sous DbManager ou Openjump) sont celles recommandées et cela suffit.

6.1.3 - Pour en savoir plus...

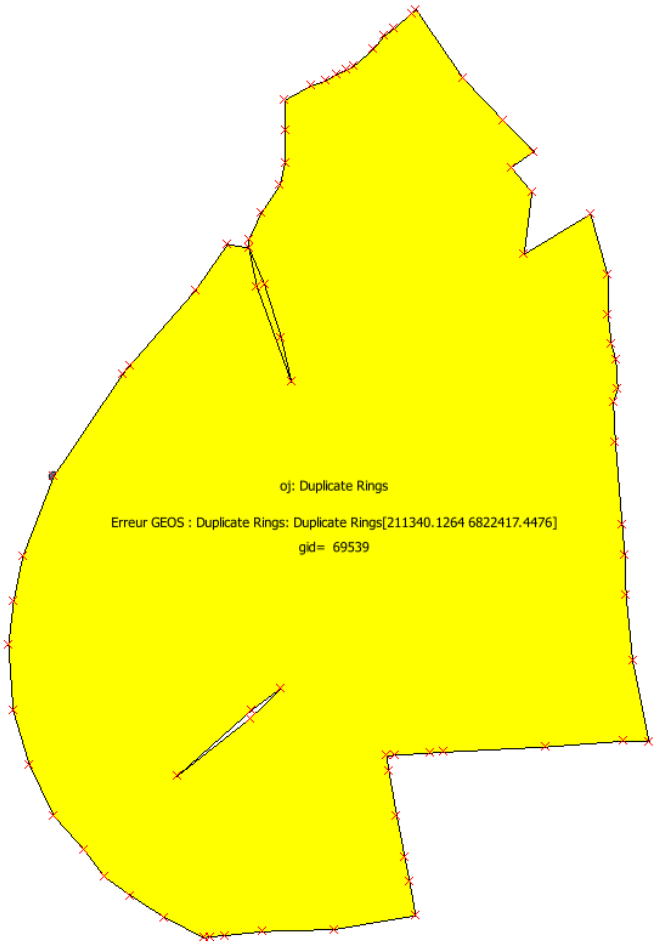
On peut charger sous QGIS les couches en erreurs fournies par les méthodes QGIS et GEOS et importer la couche fournie par OpenJump. Des étiquettes convenablement choisies permettent d’indiquer pour chaque polygone les résultats des différentes méthodes, on peut ainsi identifier les divergences entre les méthodes.

Quelques exemples :

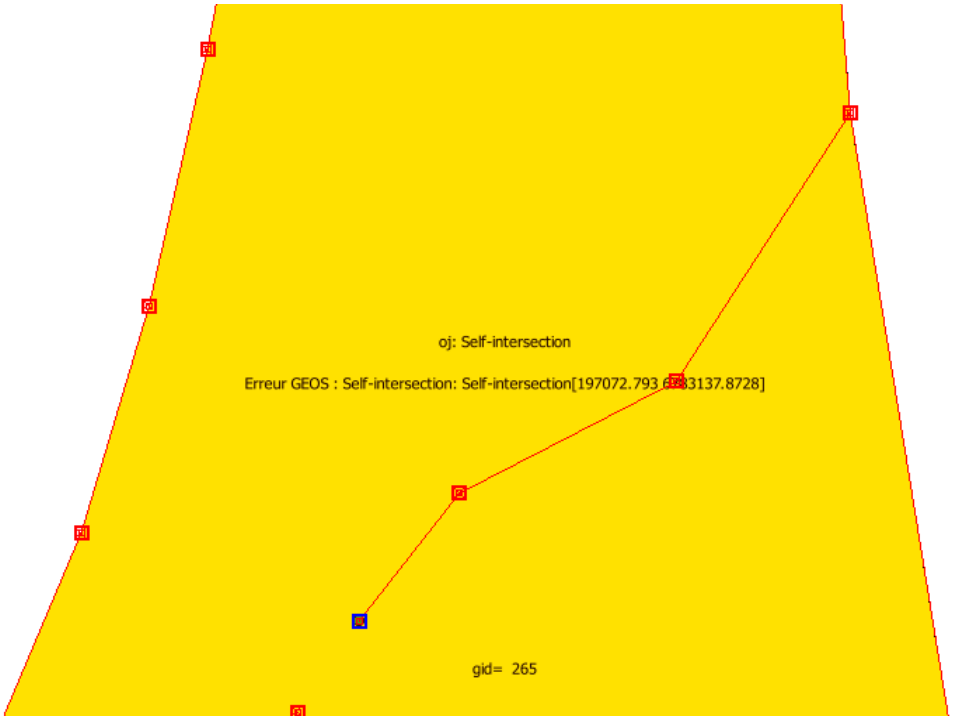
polygone : 1209 : Openjump détecte un ‘ring-self intersection’, mais pas les méthodes GEOS, ni QGIS. Il y a pourtant bien une erreur avec les sommets 36 et 39 confondus.



Polygone 69539 : la méthode QGIS ne détecte pas le ‘duplicate ring’



polygone 265 : la méthode QGIS ne détecte par ce qui est interprété par Openjump et GEOS comme une self intersection (rebroussement sur un arc pendant)



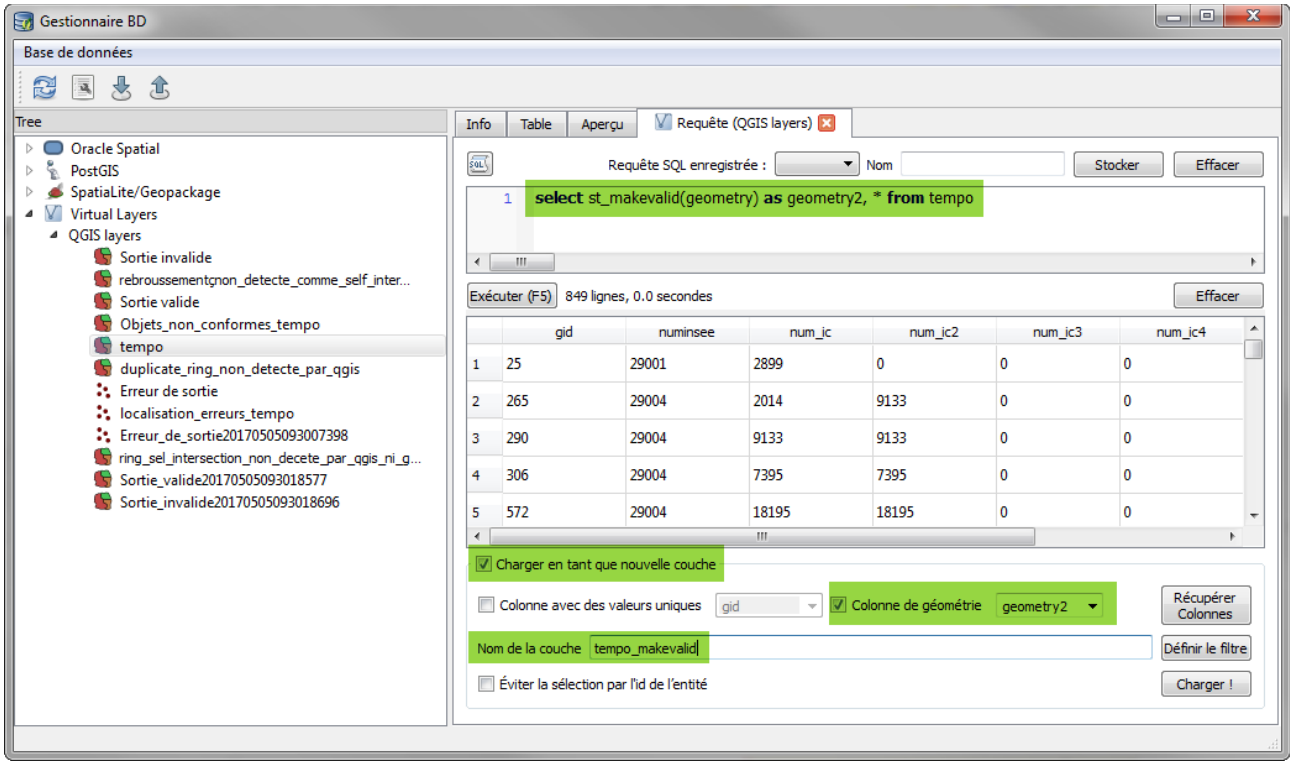
6.1.4 - Phase 2 : rendre la couche valide

SQL sous Dbmanager avec Makevalid :

select st_makevalid(geometry) as geometry, * from tempo

puis chargement dans QGIS

résultat : couche tempo_makevalid



On vérifie que la couche contient toujours 849 entités.

Travailler avec des virtual layer est assez lent sous QGIS, il est conseillé d'enregistrer la couche en SHP (par exemple) pour continuer le travail.

On peut vérifier que

select st_isvalidreason(geometry) as erreur, from tempo_makevalid where not st_isvalid(geometry) order by erreur*

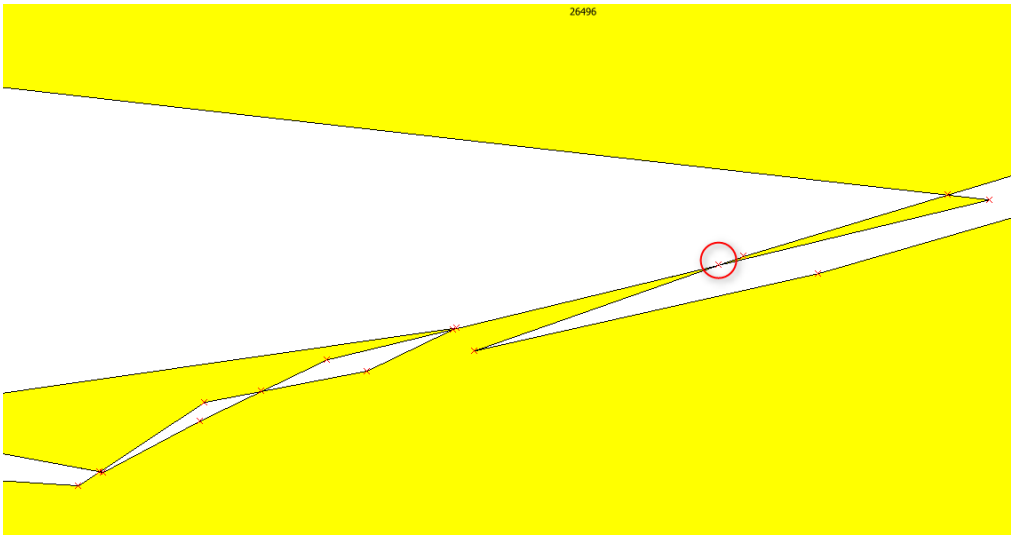
ne renvoie plus rien.

La méthode GEOS de l'algorithme 'vérifier la validité' ne renvoie également plus rien, mais la méthode QGIS renvoie 3 erreurs. Le fichier de localisation des erreurs ('erreur de sortie') associé renvoi des points qui sont tous en (0,0).

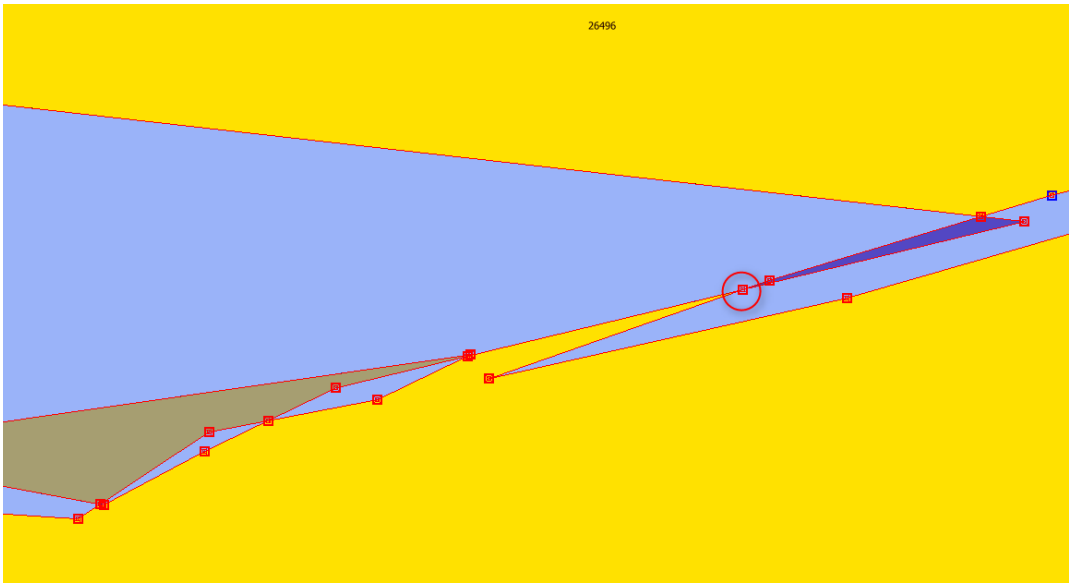
Les polygones concernés ont une géométrie complexe, exemple :

polygone 26496 :

Le point détecté en ‘self intersection’ par la méthode QGIS est celui-ci :



Mais en réalité le polygone a été recomposé par makevalid avec différents anneaux figurés ci-dessous :



Le point double détecté par la méthode QGIS appartient en réalité à des anneaux différents. Ce n'est donc pas une erreur au sens GEOS.

SQL sous Dbmanager avec buffer(geom,0):

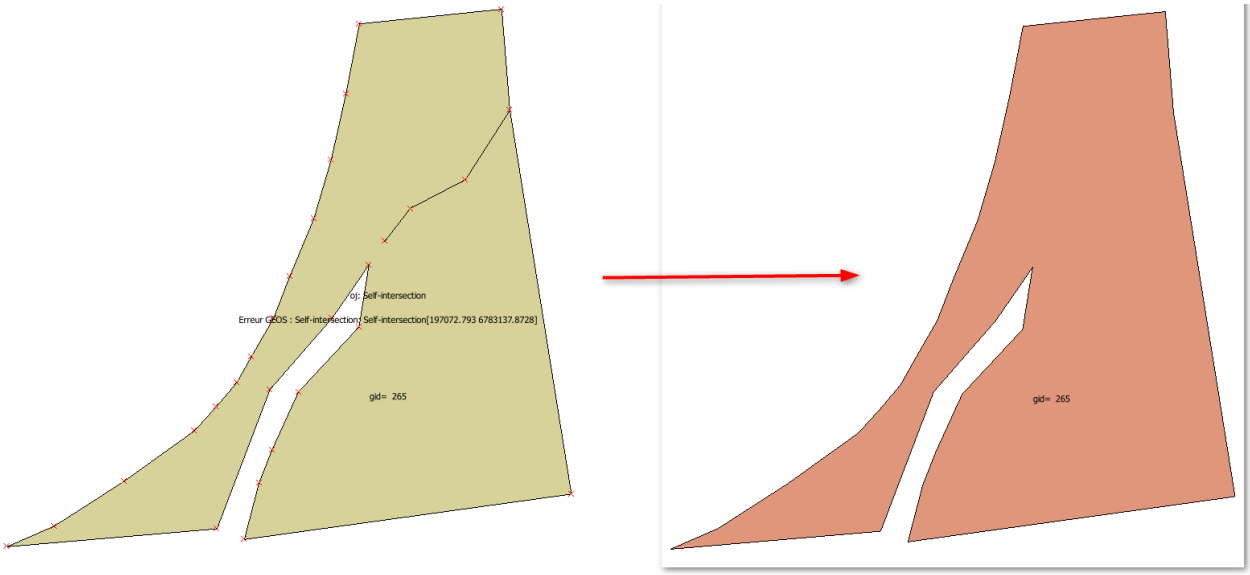
Nous avons vu que nous ne pouvons pas utiliser cette méthode, car il existe des 'self-intersection'.

Si on tente tout de même de le faire avec une requête comme :

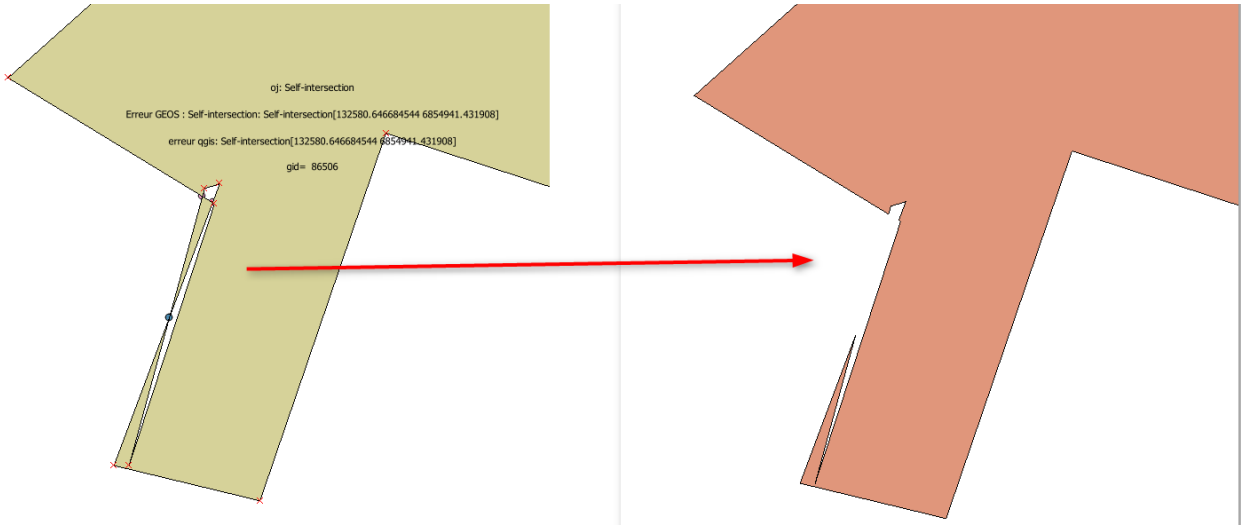
```
SELECT ST_Multi(ST_Buffer(geometry,0)) as geometry2,* FROM tempo
```

Les 'self-intersections' qui sont des arcs pendant avec rebroussement (cf ci-dessus)

sont bien corrigés :

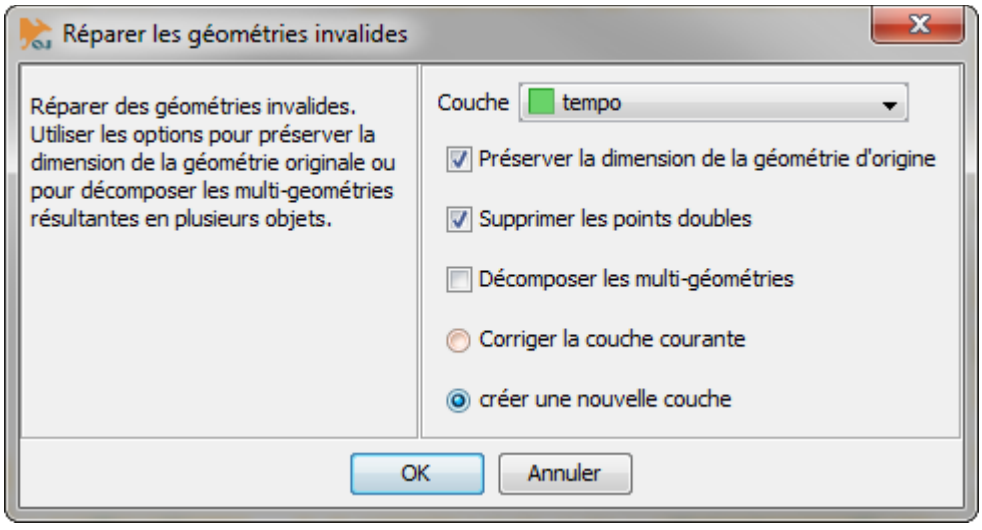


Mais on constate que pour un polygone avec une ‘self-intersection’ en papillon comme le polygone 86506, il manque un bout :



Openjump :

On peut aussi utiliser les fonctions de réparation de Openjump :

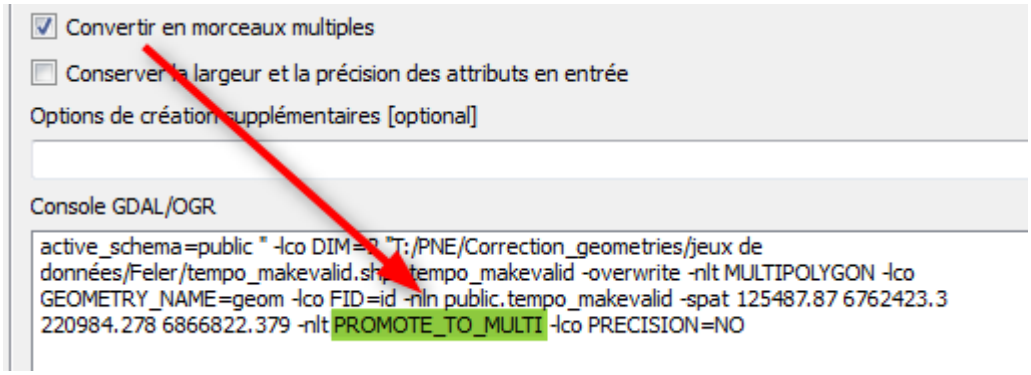


Pour ce jeu de données on retiendra que la méthode Makevalid donne des résultats valides au sens de GEOS et que la correction par Openjump est une alternative.

Nous ne chercherons pas à nettoyer outre mesure ce jeu de données, car cela reviendrait alors à modifier les géométries et nous n'avons pas de source de données de référence pour vérifier le bien fondé des résultats. Il doit maintenant fonctionner sans erreurs sous QGIS. C'est le minimum recherché.

On peut remarquer que la couche mixte des polygones et multi-polygones. Il faudrait donc faire attention lors d'un éventuel import sous PostGIS, on pourrait utiliser, par exemple avec l'algorithme Importer un vecteur vers une base de données PostGIS :

'Convertir en morceaux multiples' pour tout convertir en MULTIPOLYGON.



6.2 - Exemple : rastérisation/polygonisation

6.2.1 - Contexte

La DREAL des Pays de la Loire à fourni une couche concernant les aléas des PPRI ;
Les aléas de la baie de Bourgneuf :
V2_Alea_2100_BaieBourgneuf_Approbation.shp
273 757 enregistrements

Cette couche résulte de traitements sous des logiciels spécifiques de calcul d'aléas et comporte beaucoup d'erreurs de géométries et des recouvrements. Elle est pixélisée avec un pas d'un mètre.

Ceci nous permet d'envisager un traitement par polygonisation/rastérisation.

La couche étant très volumineuse, le test se fait sur un extrait.

6.2.2 - Principe

Chaque point étant représenté par un pixel, en rastérisant on obtient une couche de points qui porte un des attributs de la couche d'origine. Il suffit ensuite de polygoniser, chaque région homogène (polygone) récupère la valeur de l'attribut. Les erreurs de géométrie tels que les papillons sont découpés en 2 polygones distincts, les arcs pendants en polylignes , les nœuds en doubles disparaissent,..

Ce que ça ne fait pas : combler les éventuelles lacunes. Pour cela on pourra utiliser les outils de Qgis Vecteur/ Outils de géométrie / Vérifier les géométries.

Procèdure :
1- Séparer la couche d'origine en autant de valeur que le champ Class pour pouvoir gérer les superpositions.
2 - Rastériser chaque couche obtenue et repasser en vecteur pour éliminer les erreurs géométriques.

6.2.3 - Phase 1 : analyse du jeu de données

Détection des erreurs de géométrie par la méthode GEOS (« Vérifier la validité » dans la Boite à outil de traitements.)
On trouve 95 erreurs sur 44611 enregistrements (extraits de la couche) :
82 interrior disconnected
13 self intersection

6.2.4 - phase 2 : rendre la couche valide

- Ajout du champ class_int (integer) avec la formule : to_int(rigth(class,1))
- Noter l'étendue de la couche (propriété/ Métadonnées) en vue de gdal rasterize
  xMin,yMin 320055.50,6659932.60 : xMax,yMax 323897.50,6664932.50
- Scinder en couches suivant la valeur par class_int : 4 couches sur le test (1,2,4,5)
  [Vecteur/ Outil de gestion de données / séparer une couche vecteur]
  → 3 couches créées pour les valeurs 1,4 et 5
- Rasteriser chaque couche. Le pixel de la couche en sortie prend la valeur du champ class_int.
  [Menu Raster / Conversion / Rastériser] Il faut gérer la commande en modification pour ajouter le champ (bug qgis) et l'étendue :
  gdal_rasterize
  -tr 1.0 1.0 (taille du pixel, on prend 1 mètre correspondant à la taille d'origine)
  -a class_int (champ portant la valeur)
  -**te 320055.50 6659932.60 323897.50 6664932.50** (étendue de la couche globale, sinon on risque d'avoir des décalages de pixels entre les couches)
  -l couche (nom de la couche Qgis sans extension)

couche.shp (nom de la couche Qgis avec extension et chemin)
image.tiff (nom du raster avec extension et chemin)

On rastérise chaque couche correspondant à une classe. Il est plus rapide de lancer en ligne de commande par Menu Démarrer / Qgis/ OSGEO4W Shell , puis taper la commande.

- *Superposition conserver que la valeur maximale des couches*
Calculatrice raster, avec 3 valeurs a, b et c :
(a>=b AND a>=c) * a + (b>=a AND b>=c) * b + (c>=a AND c>=b) * c

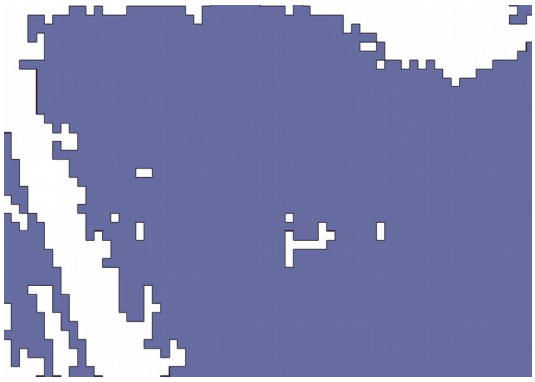
Polygoniser .
Menu Raster / Conversion / Polygoniser

Supprimer les objets avec une valeur 0 (ou nulle) correspondant aux trous de la couche. Et des objets crées pour fermer l'étendue de la couche. Sélectionner val=0 puis supprimer.

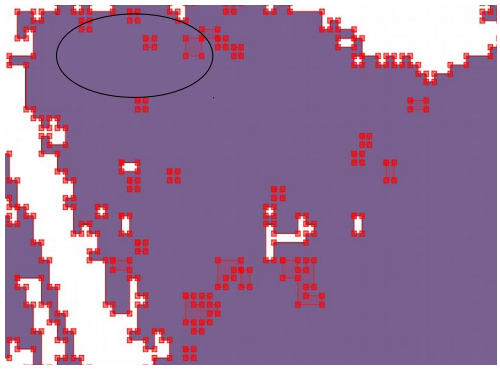
Vérifier de nouveau la validité GEOS qui doit maintenant être ok.

Cette méthode corrige mal, les trous imbriqués (holes nested). Ceux-ci se retrouvent comme des trous alors qu'ils ne devraient pas.

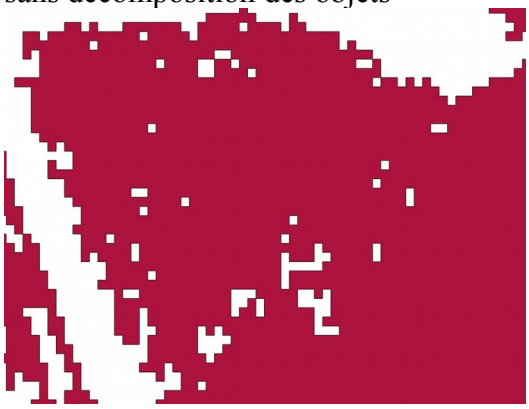
Couche Origine



Couche avec affichage des sommets



Les trous apparaissent
sans décomposition des objets



Et disparaissent avec décomposition des objets.

Il faut alors commencer par décomposer les objets de multiples en uniques pour éviter ce phénomène.
On se reportera au paragraphe 5.2.3.3 pour mettre en œuvre une méthodologie adaptée.

6.3 - Communes italiennes

6.3.1 - Contexte

Jeu de données des communes italiennes fournies par Istat, l'Institut Statistique Italien utilisée dans la page sur la validation des géométries de Spatialite :SQL functions based on liblwgeom support in version 4.0.0. et reprise dans le blog SIG et Territoires sur les outils de validation des géométries.

La table est celle des 8 094 communes italiennes en 2011.

6.3.2 - Phase 1 : analyse du jeu de données

Analyse avec la requête :

```
select rowid, ST_IsValidReason(geometry) from com2011  where not st_isvalid(geometry)
```

détecte 19 objets invalides en 'ring-self intersection'

Pour mémoire Openjump détecte les mêmes polygones en erreurs.

6.3.3 - Phase 2 : correction des géométries invalides.

Correction avec

```
select rowid, st_makevalid(geometry) as geometry, * from com2011
```

sous DBManager avec le fournisseur virtual layer de QGIS.

On peut également faire une correction avec Openjump (beaucoup plus rapide que la manipulation des virtual layers sous QGIS)


Ressources, territoires, habitats et logement
Énergie et climat
Développement durable
Prévention des risques
Infrastructures, transports et mer
Présent
pour
l'avenir