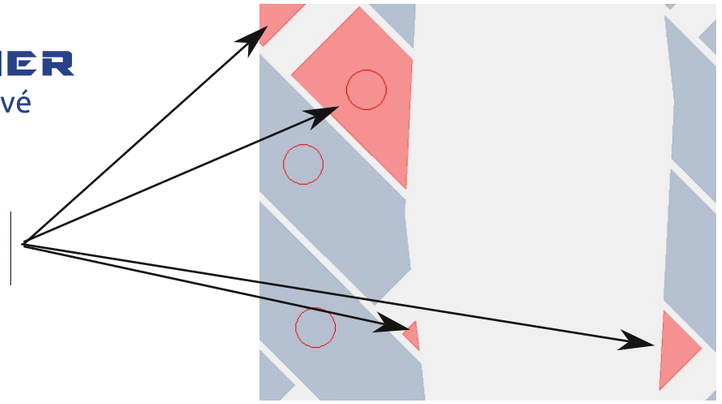
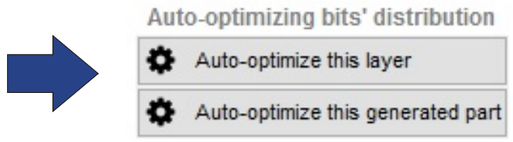
AUTO-OPTIMISATION

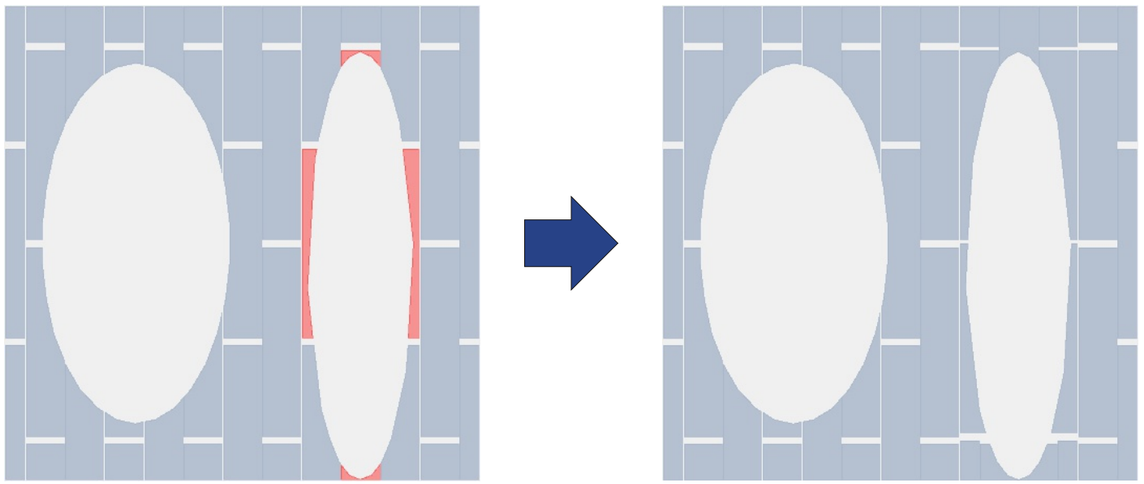
BIT IRREGULIER

Il ne peut pas être soulevé
par le préhenseur



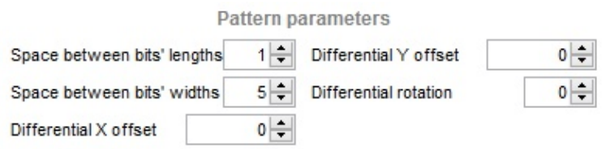


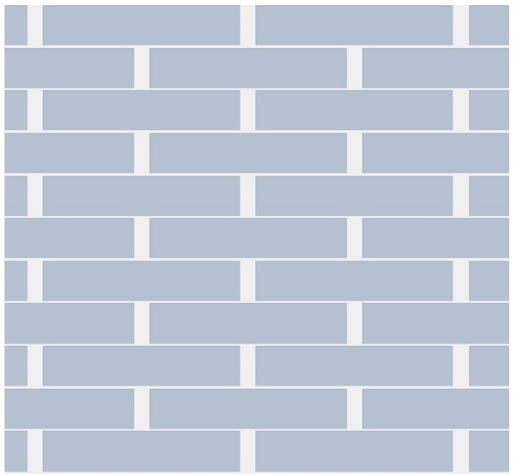
Essaye d'éliminer tous les bits irréguliers
en déplaçant ou coupant les bits
de manière **AUTOMATIQUE**



IMPROVED BRICK PATTERN

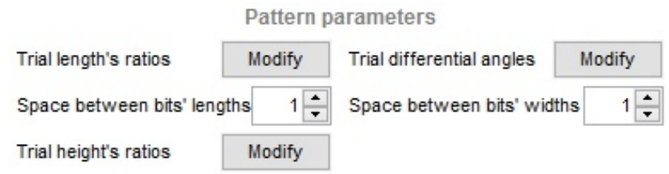
- Basé sur Classic Brick Pattern
- Plus personnalisable
- Auto-opmisation implémentée





ECONOMIC PATTERN

- Similaire à Classic Brick Pattern
- Optimisé à chaque placement de bit



MeshineBIS

VERSION 0.2

Etudiant: TRAN Quoc Nhat Han - ISI2 P17 (quoc_nhat_han.tran@utt.fr)
Encadrant: Laurent DANIEL (laurent.daniel@utt.fr)

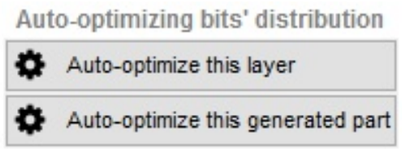
AUTRES

- Les paramètres sont fonction du pattern
- Exporter & Importer configurations d'un pattern
- Créer de nouveaux patterns

AUTO-OPTIMISATION

Qu'est-ce que l'auto-optimisation?

Une tâche qui permet d'éliminer tous les bits **irréguliers**
pour une couche remplie ou pour tout l'objet généré.



Quand on utilise?

Après la phase **Generate Layers** et
avant d'exporter la distribution des bits.

Comment ça marche?

Le(s) développeur(s) de Pattern (Template)
propose(nt) un algorithme spécial afin de traiter ces bits.

Comment on utilise?

- Pour détecter les bits irréguliers: Review > Show irregular bits
- Pour optimiser:
  + Pour tout l'objet:
    Review > Auto-optimise this generated part
  + Pour une seule couche générée:
    Review > Choisir la couche voulue > Auto-optimise this layer

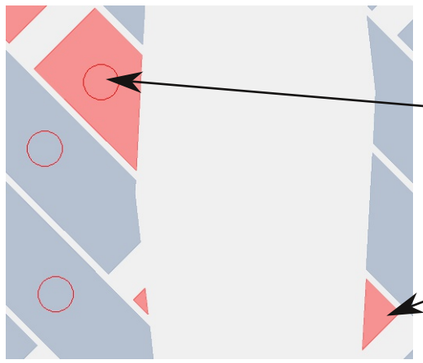
Pour Dev

Implémenter la méthode **optimise** qui va retourner
le nombre de bits non résolus; -1 si échoué.
Vous pouvez faire n'importe quoi,
mais toujours respectez la ratio de tailles d'un bit
(longeur X hauteur)
1 * 1; 1 * 1/2; 1/2 * 1; 1/2 * 1/2.

Que sont les bits "irréguliers"?

Ce sont ceux dont la surface n'est pas suffisament large pour avoir
un Lift Point à l'aide duquel le machine va tenir le bit.

Attention
Si la surface est comprise de multiple zones séparées, chaque zone
doit avoir son propre Lift Point.

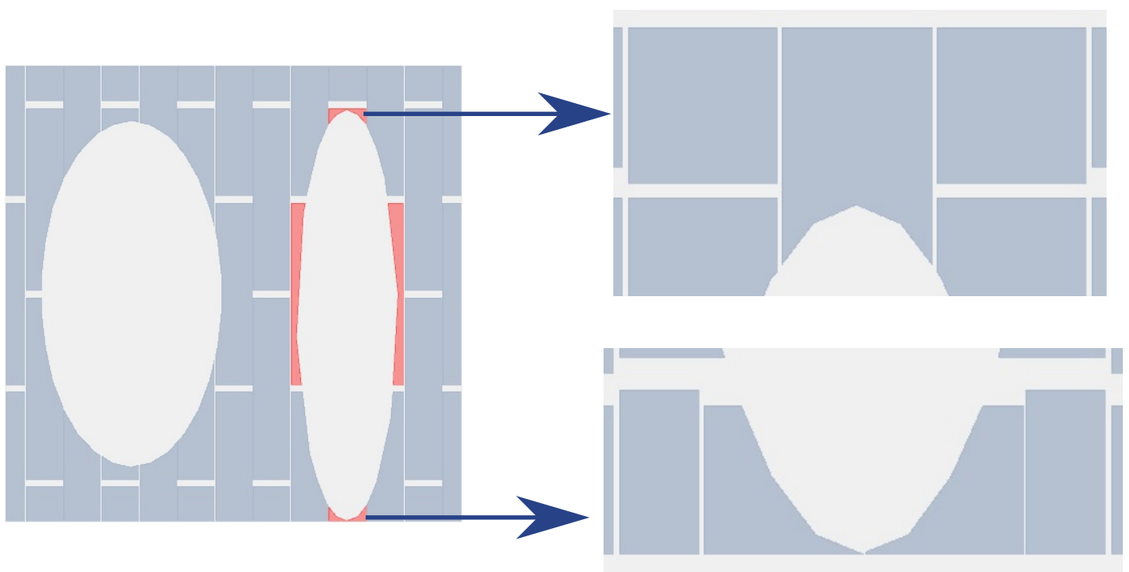


Exemple:
La surface de ce bit est séparé
en 2 morceaux.
Bien que l'un a un Lift Point,
l'autre n'en a pas.
Alors, ce bit est **irrégulier**.

Pour Dev

Utiliser **DetectorTool.checkIrregular**

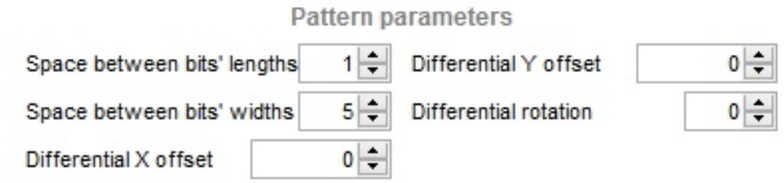
Exemple [zoom]



Improved Brick Pattern

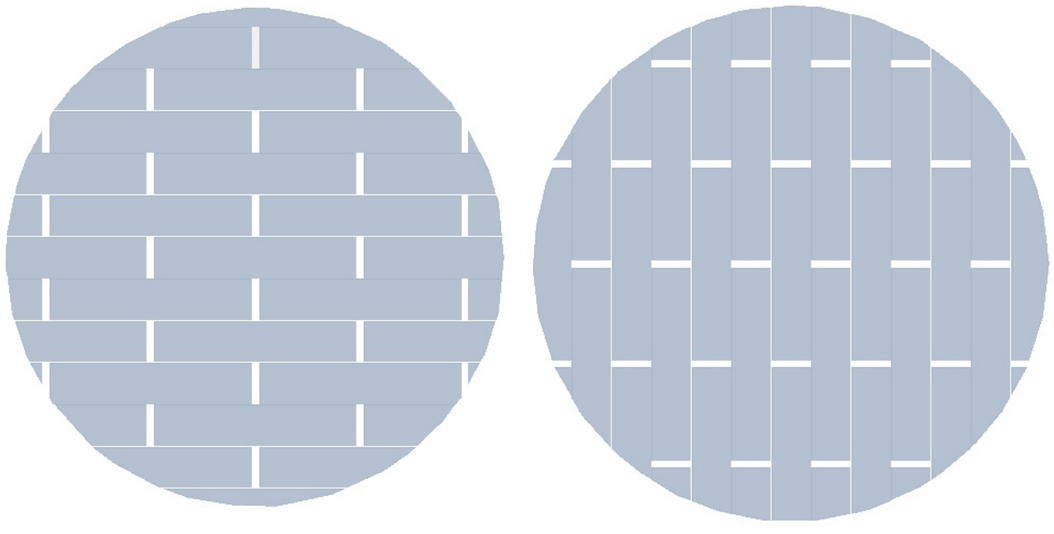
C'est quoi?

Un pattern amélioré à partir de la version Classic, avec de nombreux nouveaux paramètres qui nous permettent de personnaliser plus facilement.



Comment on utilise?

Template > Pattern > Improved Brick Pattern
Personnaliser les paramètres à droite, notamment:
+ **Space between bits' lengths**: l'espace entre 2 longueurs
+ **Space between bits' height**: l'espace entre 2 hauteurs
+ **Differential rotation**: la rotation d'une couche par rapport à celle précédente
Ensuite, Generate Layers et voilà!



Un exemple montrant la rotation entre-couche

Comment on optimise?

Faites selon la guide d'auto-optimisation.
Note important: L'algorithme implémenté n'a pas toujours 100% de succès. Et le temps augmente de manière **exponentielle** en fonction du nombre de bits.

Pour Dev

Pour ce pattern, on utilise l'algorithme **Glouton** (partiel).

1, Copier le pavage.

2, Détecter tous les bits irréguliers.
S'il n'y en a aucun, arrêter l'algorithme.

3, Prendre-en 1 bit non "essayé".
Si tous les bits sont "essayés", arrêter l'algorithme.

4, Essayer de le déplacer dans 4 sens en creusant les bits en face pour l'espace (et en remplissant)

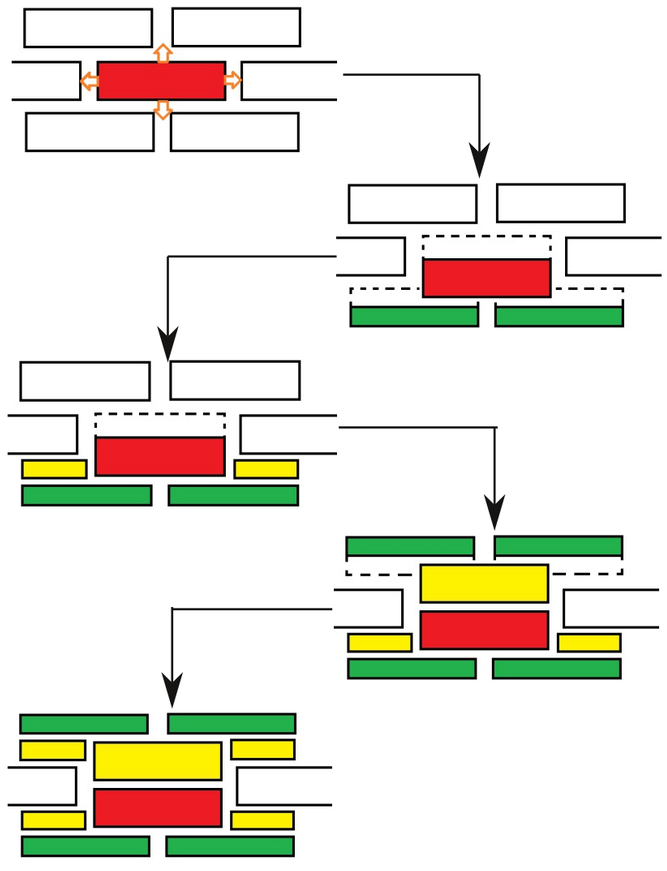
5, Recouvrir l'espace laissé par un bit complet (et creuser les bits en derrière).

6, Recalculer la surface des bits

7, Vérifier si le nombre de bits irréguliers décrémente.
Si oui, appliquer la modification sur le pavage original. Retourne à **1**.
Sinon, retourne à **3**.

Illustration

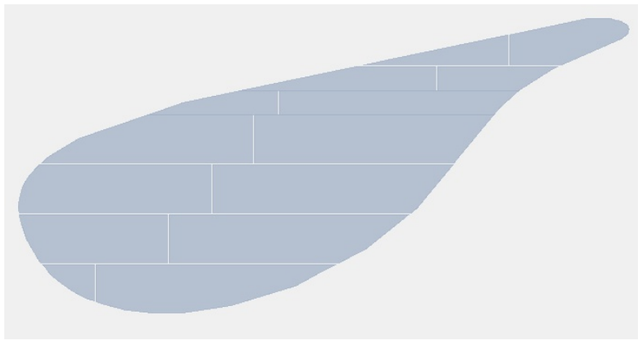
Rouge: Bit à résoudre
Verte: Bit creusé (modifié)
Jaune: Bit couvrant



Economic Pattern

C'est quoi?

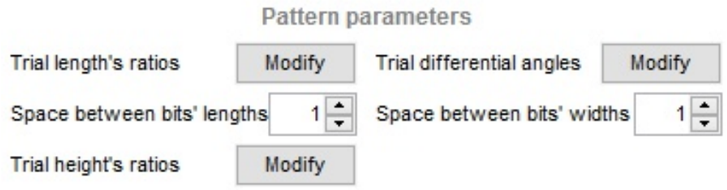
Un pattern optimisé à chaque pavage,
assurant qu'aucun bit irrégulier n'apparaît.



Comment on utilise?

!!! Attention: Pour ce pattern, le découpage est généré au moment de l'optimisation !!!

Template > Pattern > Economic Pattern
> Personnaliser les paramètres à droite > Generate Layers
> Review > Auto-optimise (this layer/this generated part)



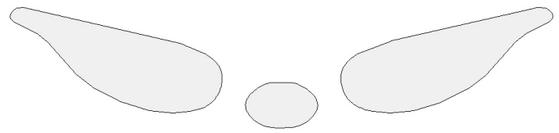
Comment on paramètre?

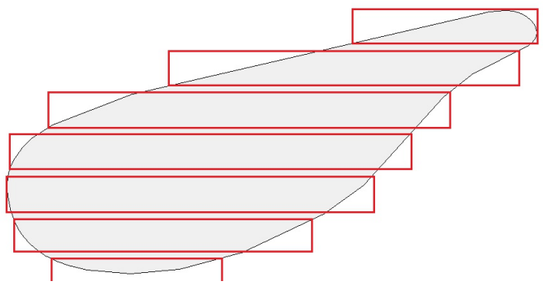
+ **Space between bits' lengths**: l'espace entre 2 longueurs
+ **Space between bits' height**: l'espace entre 2 hauteurs
+ **Trial differential angles**: les rotations possibles (en degrés) d'une couche par rapport à celle précédente. Le programme va choisir la première qui donne une bonne couche (qui n'a aucun bit irrégulier)
Exemple: 90; 45; 135; 30; 60; 120; 150
+ **Trial lengths' ratios**: les nombres réels entre 0 et 1 pour calculer et essayer le décalage horizontal lors d'un pavage d'une ligne
Exemple: 0.0; 0.125; 0.25; 0.375; 0.5; 0.625; 0.75; 0.875; 1.0
+ **Trial height's ratios**: les nombres réels entre 0 et 1 pour calculer et essayer le décalage vertical lors d'un pavage d'une zone
Exemple: 0.0; 0.125; 0.25; 0.375; 0.5; 0.625; 0.75; 0.875; 1.0

Pour Dev

Cet algorithme est **récursif**.

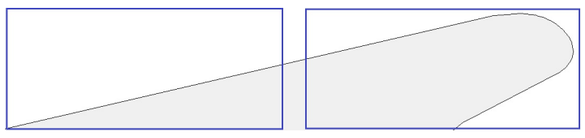
1, optimise: Paver une couche.
Déterminer tous les zones par **AreaTools.getLv0Areas**.
Pour chaque zone, appliquer **fillZone**.
+ Si **fillZone** échoue, **optimise** recommence avec l'image de zone après une rotation calculée par Trial differential angles.
+ Si **fillZone** échoue même avec tous les angles de rotation, le procès retourne un échec.

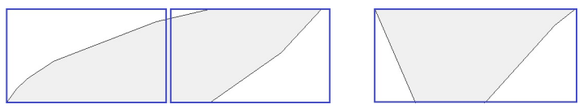




2, fillZone: Paver une zone.
Le pavage de zone se fait **verticalement** ligne par ligne, de haut en bas:
+ Pour chaque ligne, appliquer **fillBand**.
+ Si **fillBand** échoue, refaire la ligne précédente avec une demie hauteur si sa hauteur est bitWidth et puis continuer. Mais si on échoue de nouveau, relancer **fillZone** avec un nouveau décalage.
+ Si **fillBand** échoue et que la hauteur de la ligne précédente est bitWidth/2, relancer **fillZone** avec un nouveau décalage.
* Relancer **fillZone** avec un nouveau décalage: recommence avec un décalage vertical calculé à partir de Trial height's ratios.
+ Si **fillZone** échoue même avec tous les décalage possibles, **fillZone** retourne un échec.

3, fillBand: Paver une ligne
Le pavage de ligne débute **horizontalement** de gauche à droite.
+ Si le bit est irrégulier, replacer le bit précédent avec une demie longueur si sa longueur est bitLength puis continuer. Mais si le replacement ne réussit pas, relancer **fillBand avec un nouveau décalage horizontal**.
+ Si le bit suivant sera irrégulier s'il est placé et la longueur de celui précédant est bitLength/2, relancer **fillBand avec un nouveau décalage horizontal.**
* Relancer fillBand avec un nouveau décalage horizontal: recommencer avec un décalage horizontal calculé à partir de Trial length's ratios.
+ Si **fillBand** échoue même avec tous les décalages possibles, **fillBand** retourne un échec.





Fait / Reste à faire

Les paramètres sont fonction du pattern

Accepter seulement un **Double** ou un **List de Double** pour un paramètre (tous les autres valeurs sont filtrées)

Importer/Exporter les paramètres de pattern

Stocker sous le format binaire (Serializable) de Java.

Pour créer de nouveaux patterns, il faut:

+ Etendre **PatternTemplate**

+ Implémenter:
  - initiateConfig --> les paramètres spéciaux de ce pattern.
config.add(new **PatternParameterConfig**(…))
(voir le constructeur de **PatternParameterConfig**)
  - ready --> les paramètres privés de pattern après slicing et avant de générer les bits.
  - createPattern --> la façon de placer les bits dans une couche
  - optimize --> l'algorithme d'optimisation de pattern
  - moveBit --> la façon de déplacer un bit généré

+ Probablement redéfinir ces méthodes pour une meilleure description:
  - getIconName
  - getCommonName
  - getDescription
  - getHowToUse

Idées pour la version 0.3

*** Importer/Exporter le pattern développé

*** Undo/Redo, en particulier pour la déplacement manuel

*** Un pattern qui fonctionne dans tous les cas solubles sans créer des bits irréguliers

** Nouveaux détecteurs + Affichage d'irréguliers

** Texte tooltip d'aide très claire sur tous les composants

** Vue 3D --> Voir + modifier le repère

** Améliorer l'algorithme d'optimiser d'Improved Brick Pattern

* Multi-threading au niveau du choix de l'angle de rotation pour des zones séparées (**Economic Pattern**). Aujourd'hui, la rotation est la même pour toutes les zones. Cela nous donne une plus grande possibilité de trouver une solution

* Importer/Exporter la configuration du pattern sous le format XML

* Panneau de log

* Avoir possibilité d'arrêter le cours d'un traitement

* Un tutoriel

* Build logiciel automatique (par Gradle ou Maven, etc.)

MeshInBts

VERSION 0.2

Etudiant: TRAN Quoc Nhat Han - ISI2 P17 (quoc_nhat_han.tran@utt.fr)
Encadrant: Laurent DANIEL (laurent.daniel@utt.fr)