GGCAD/CAM 2.1

Manuel d'utilisation

Table des matières

Note : le terme «advanced» se rapporte à des fonctionnalités présentes uniquement dans la version advanced.

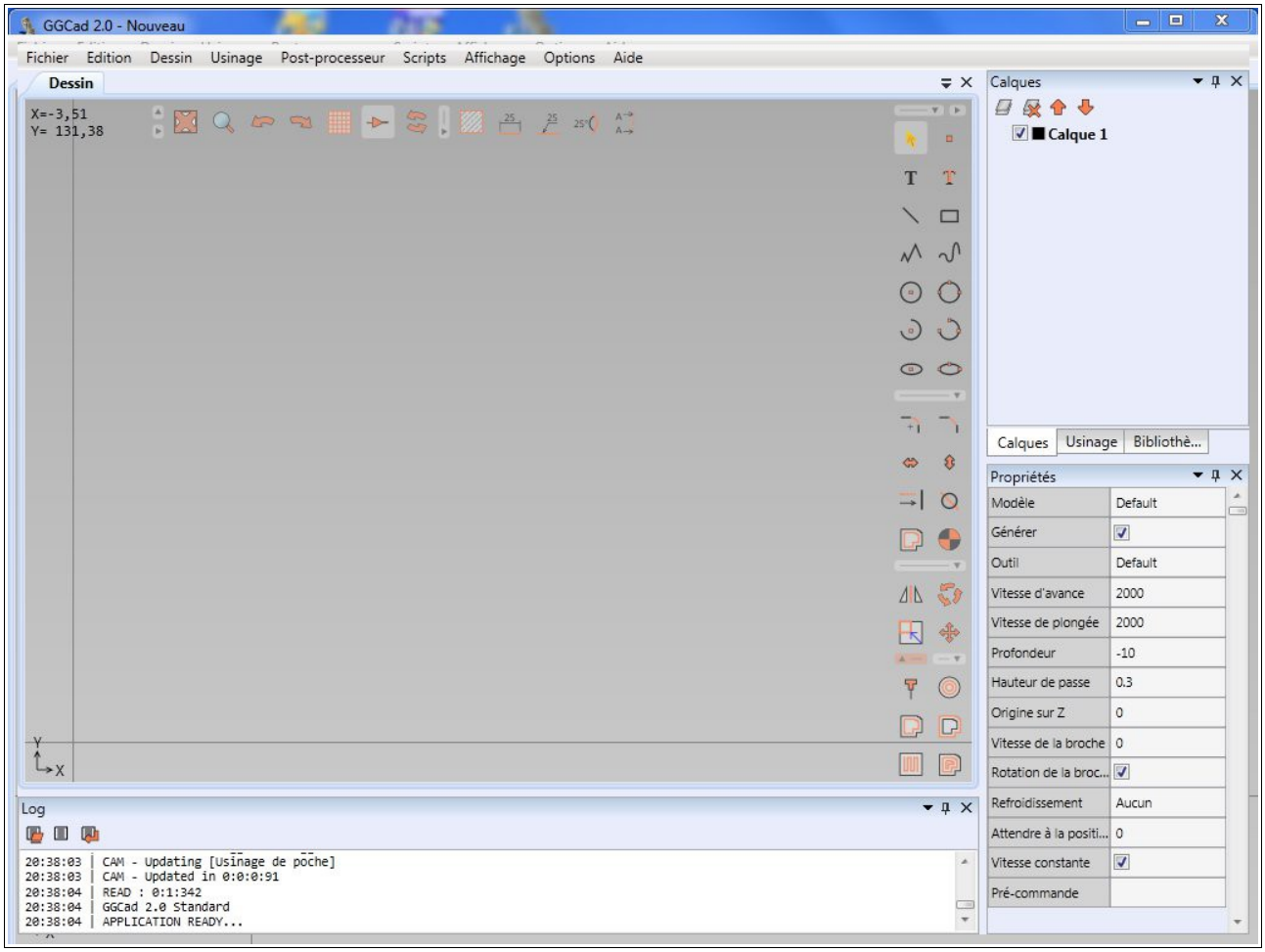
Note : le terme « professional » se rapporte à des fonctionnalités présentes uniquement dans la version professionnelle.

1 Interface graphique

1.1 Générale

GGCam 2.0 intègre un système d'interface graphique avancé. Il se compose d'un ensemble de fenêtres que l'on peut déplacer ou détacher au gré de sa volonté.

Nous l'appellerons **Espace de travail**.



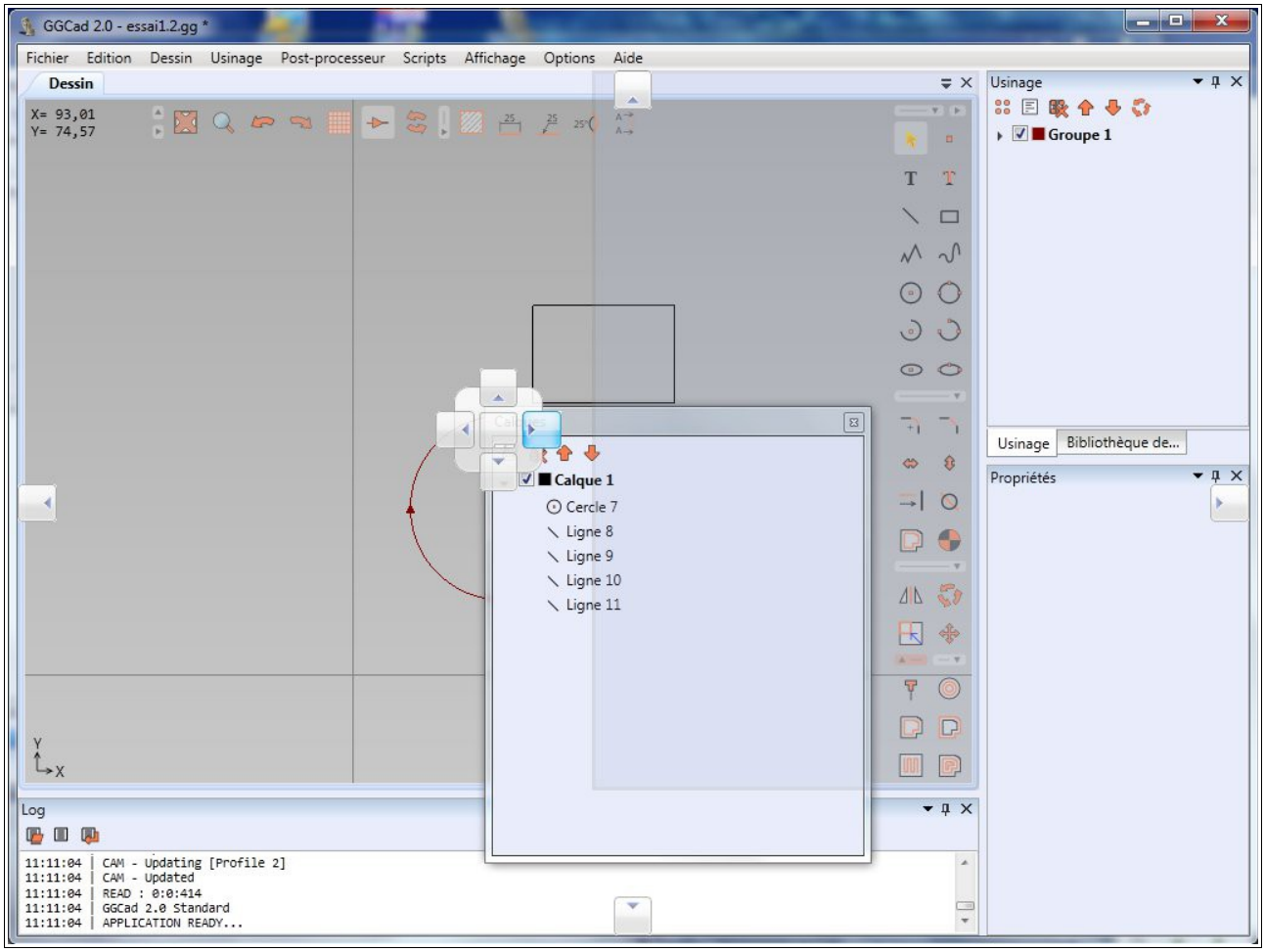
Au démarrage:

vous avez un environnement par défaut

- avec la fenêtre de dessin en haut à gauche.
- une fenêtre affichant le journal en bas au milieu.
- une fenêtre comportant les arbres en haut à droite
- une fenêtre comportant les propriétés en bas à droite.

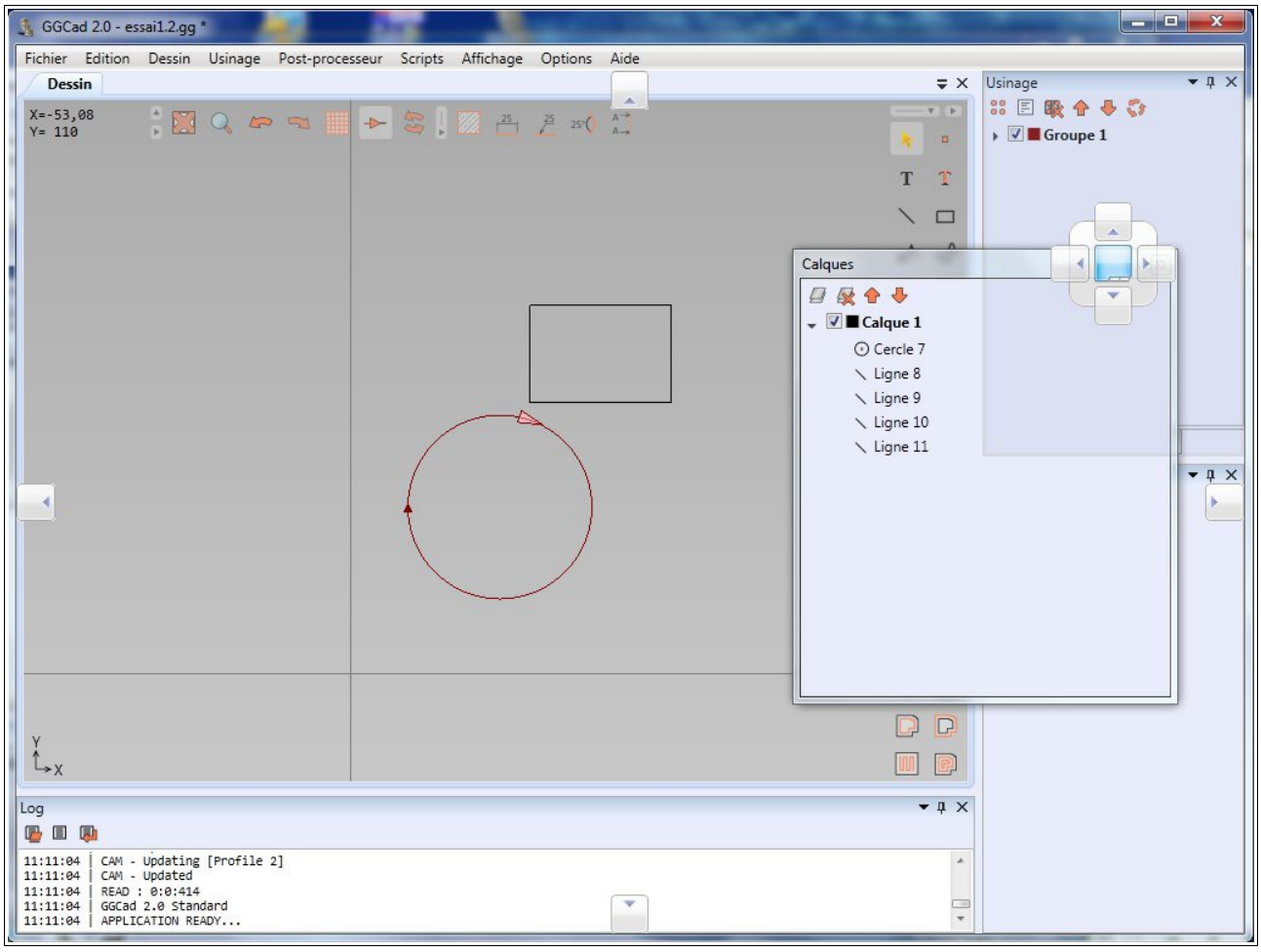
Ces fenêtres peuvent s'afficher et se fermer par le menu `[Affichage]`.

En maintenant le bouton gauche de la souris enfoncé sur la barre des titres ou sur l'onglet correspondant, vous pouvez les déplacer librement



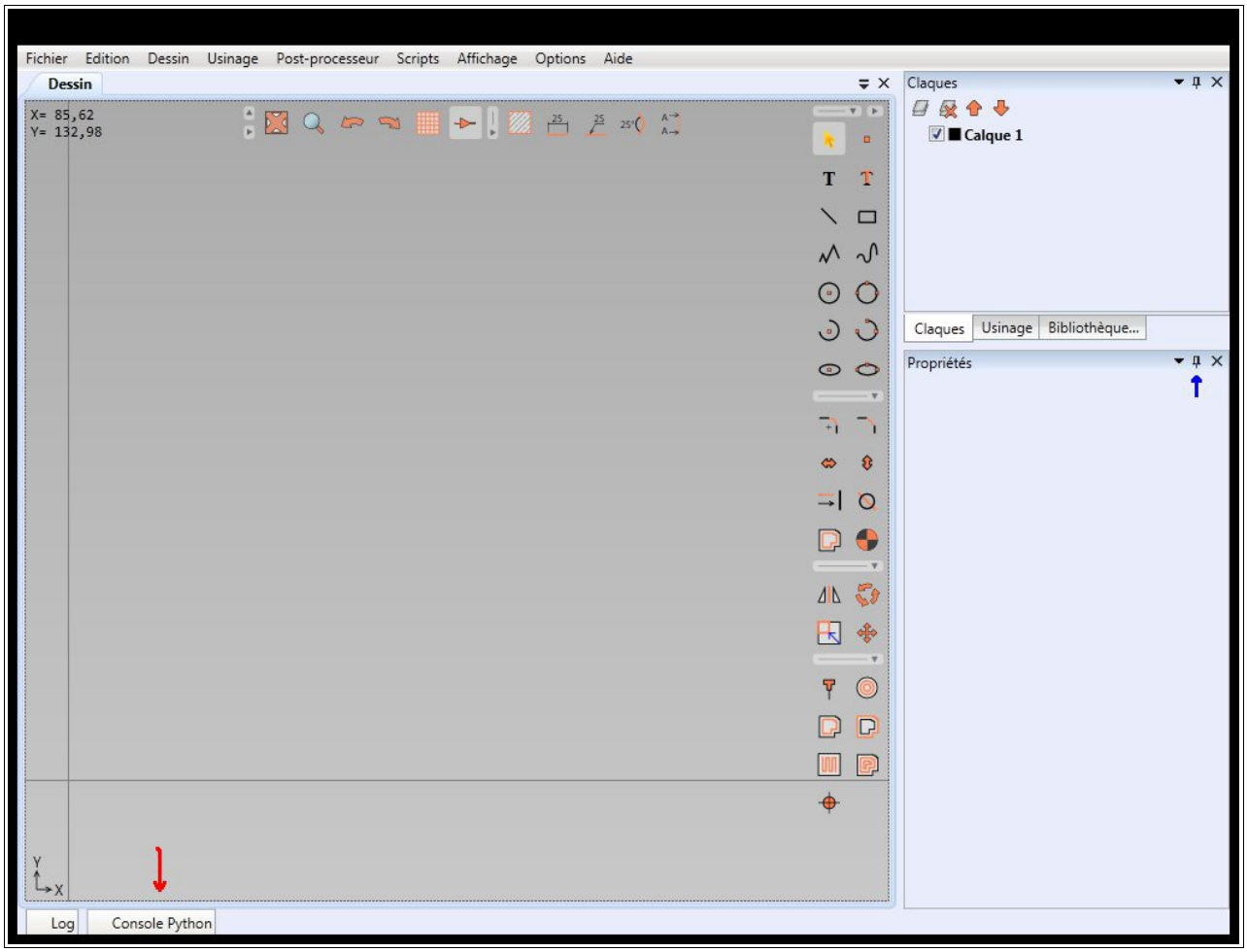
Lorsque vous déplacez une fenêtre, vous pouvez la lâcher pour la laisser flottante ou survoler une zone de l'interface.

Des images en forme de flèches apparaissent au centre et sur les bords. Si vous déplacez la souris sur une de ces images, vous pourrez alors accrocher la fenêtre dans la zone qui s'affichera en surbrillance.

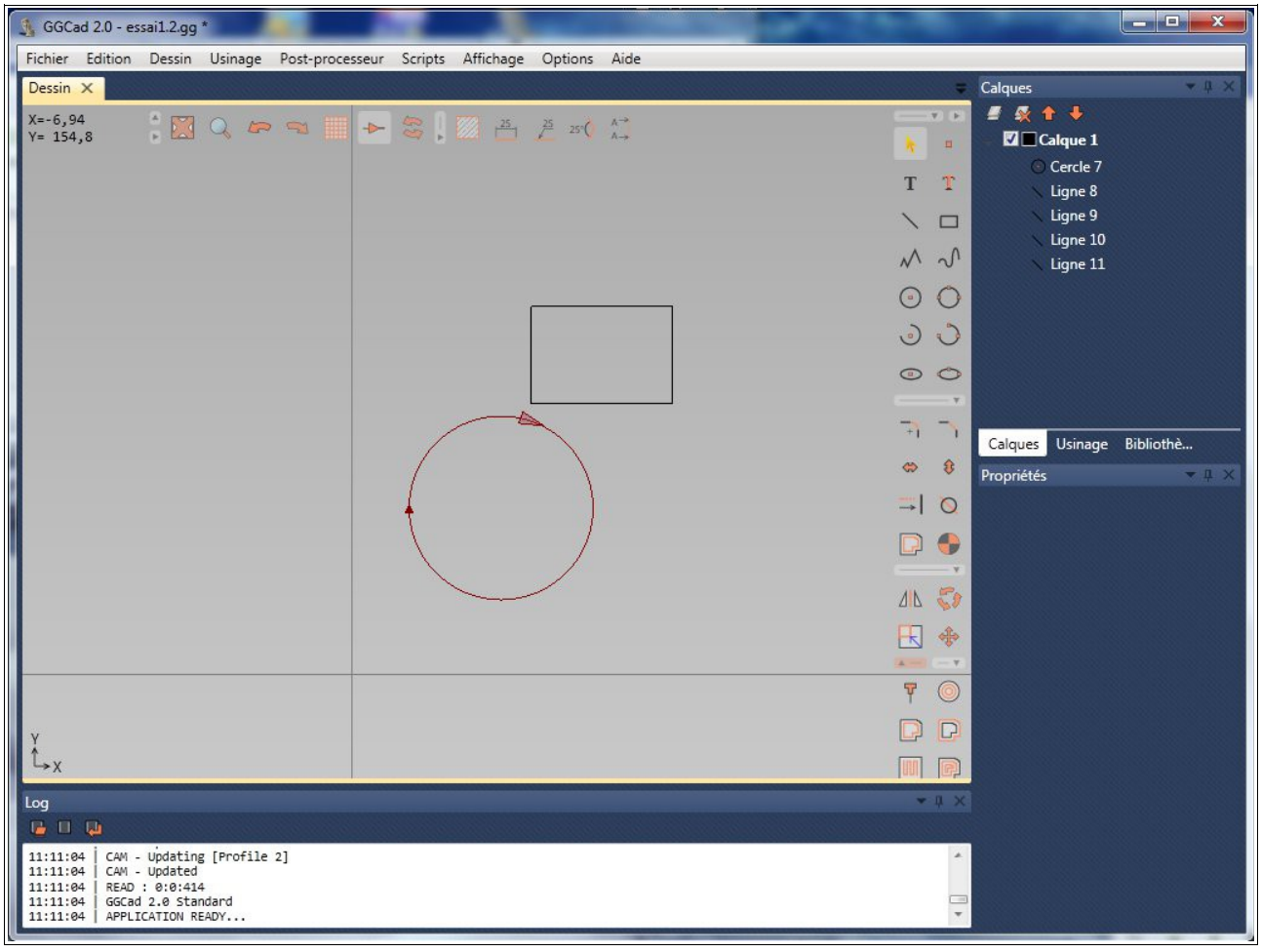


Si vous cliquez sur le symbole au-dessus de la flèche bleu à droite de la barre de titre d'une fenêtre, elle va se mettre en mode `[AutoHide]`, qui permet de la cacher automatiquement. Un onglet se crée alors sur l'un des bords de l'espace de travail. Il suffira de le survoler pour faire réapparaître la fenêtre.

Pour l'accrocher de nouveau, cliquez encore sur la même icône.



Enregistrez et ouvrez ces espaces de travail par le menu `[Options][Espace de travail]`. Changez l'apparence par le menu `[Options][Thèmes]`. Il existe 4 thèmes différents pour changer l'apparence.



1.2 Dessin 2D

Les actions se font avec le bouton gauche de la souris et les terminent ou s'annulent avec le bouton droit.

Le zoom s'effectue à la molette.

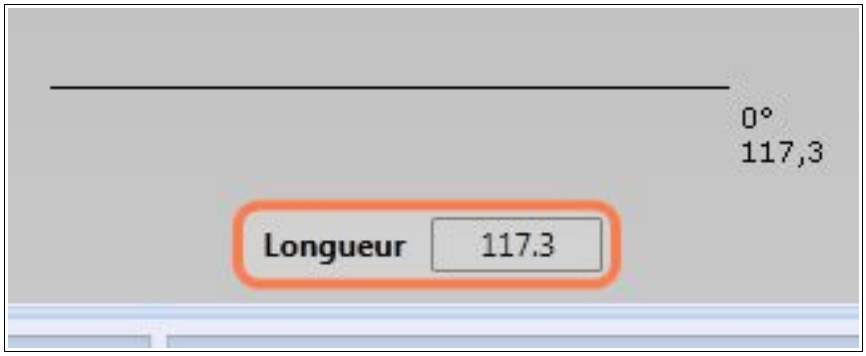
Le déplacement de l'écran implique de presser le bouton droit puis de déplacer la souris.

La touche `[Echap]` annule l'opération courante.

La barre d'icône sur le côté droit vous donne accès à tous les outils de dessin.

Aide au dessin:

Des messages s'affichent en bas de page



Dans l'exemple précédent, l'outil `[Ligne]` affiche un message longueur. Vous pouvez l'ignorer, ou taper une valeur puis `[Entrer]` pour forcer la longueur recherchée.

Affichage d'une grille grâce à l'icône sur la barre supérieure.

L'intervalle de la grille est définit dans les préférences de la CAO 2D: menu `[Options]` `[Préférences]`.

GGCam supporte nativement les éléments de dessin suivant:

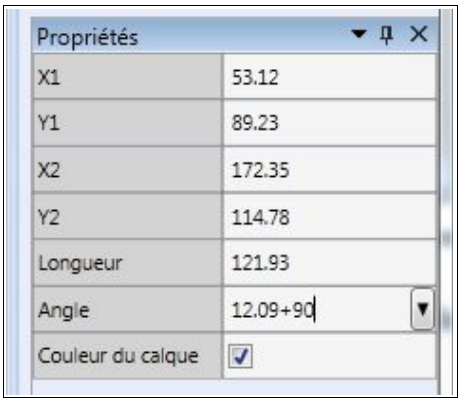
- point
- ligne
- cercle
- arc
- ellipse
- polygone
- courbe
- texte
- contour de texte

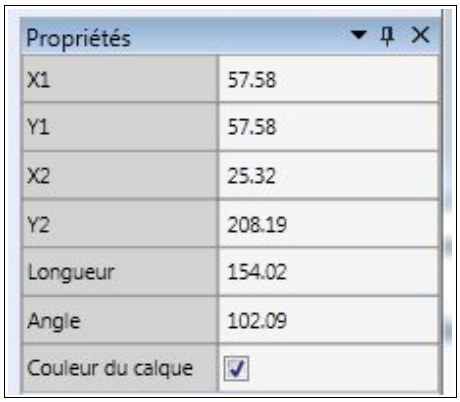
Ainsi que les éléments spécifiques au dessin industriel :

- cotation
- annotation
- cotation angulaire
- section
- hachures

Important: pour les arcs, sans en déplacer le centre, vous pouvez les modifier en maintenant la touche `[Contrôle]` enfoncée et en déplaçant une des extrémités.

Les propriétés telles que `[Angle]`, `[Longueur]`... peuvent être modifiées grâce à une opération arithmétique dans la fenêtre des propriétés.





Sélection multiple

Pour sélectionner plusieurs éléments, maintenez la touche `[Contrôle]` enfoncée.

Pour dé-sélectionner une partie, maintenez la touche `[Shift]` enfoncée .

Document

A tout moment, les propriétés du document peuvent être modifiées par le menu `[Fichier]` `[Propriétés du document]`.

Vous pouvez également y changer l'unité `[Millimètre]` ou `[Pouce]`.

Calques

Les éléments de dessin sont créés et rangés dans des calques. Les calques sont accessibles dans la fenêtre `[Calques]`. Lorsque vous en sélectionnez un dans l'arbre, ses paramètres sont affichés

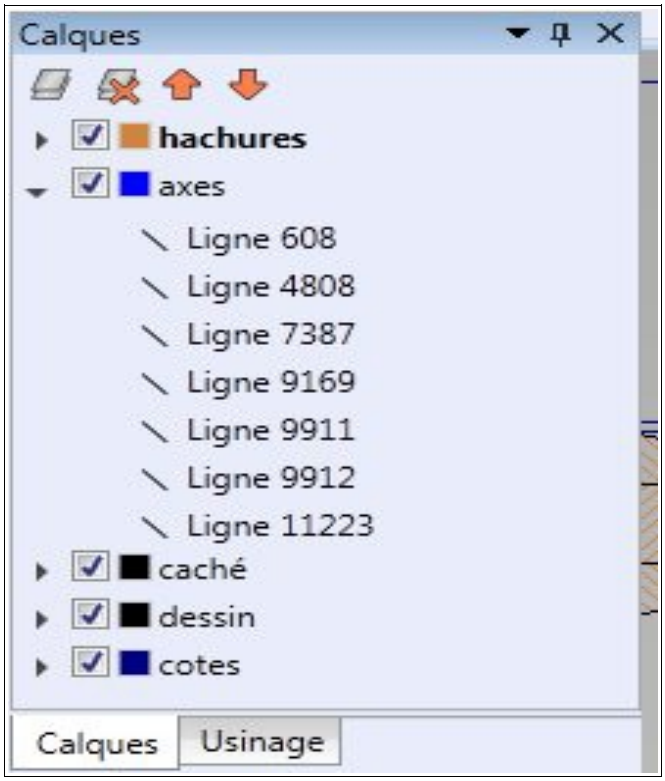
automatiquement dans la fenêtre des propriétés.

Vous pouvez changer:

- la couleur
- l'épaisseur du trait
- le style de trait

Le blocage du calque rend impossible la modification de ses éléments

Vous pouvez rendre les calques visibles ou invisibles en cochant la case de leur nom



Le dessin à l'écran, ainsi qu'à l'impression se fera toujours dans le même ordre.

Vous pouvez déplacer les éléments d'un calque à un autre par le menu `[Dessin][Changer les éléments de calque]`.

Opérations

Plusieurs outils sont disponibles afin de faciliter le dessin:

- Congé : relie 2 traits par un arc
- Chanfrein : relie 2 traits par un trait oblique
- Déplacement horizontal : déplace horizontalement l'élément choisi à une distance donnée par rapport à un autre élément, sur l'axe horizontal.
- Déplacement vertical : identique au Déplacement horizontal, mais sur axe vertical.
- Joindre : prolonge un élément jusqu'à l'intersection d'un autre élément
- Couper : permet de sectionner un élément
- Décaler : décale un objet fermé (intérieur ou extérieur)
- Déplacer l'origine : déplace le point de coordonnées (0,0)
- Symétrie : crée les éléments symétriques par rapport à un axe
- Rotation : effectue une rotation sur les éléments sélectionnés
- Échelle : permet d'appliquer un facteur d'échelle aux éléments sélectionnés
- Déplacement :déplace de (n,m) les éléments sélectionnés

D'autres opération sont disponibles par le menu `[Dessin]` :

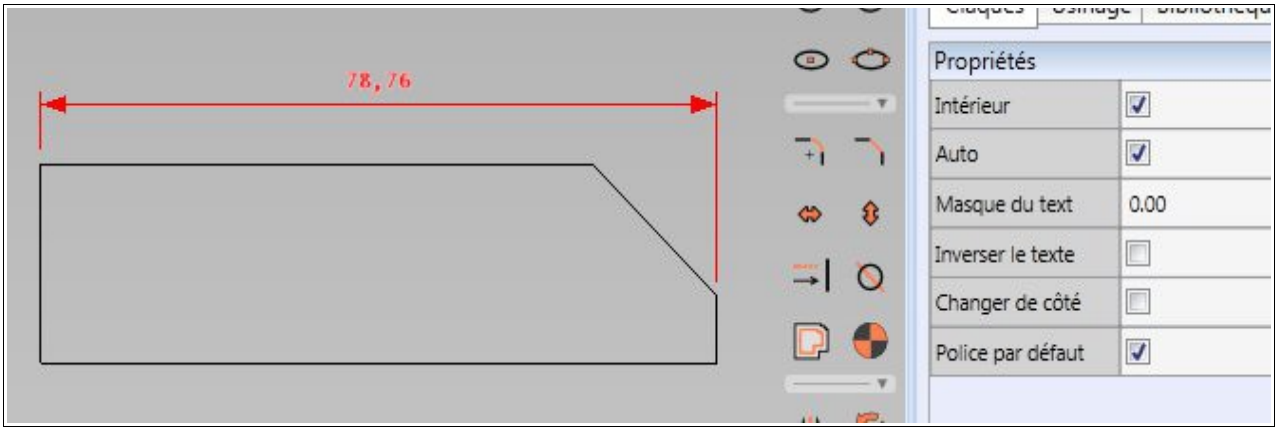
- Intersection : réalise l'intersection entre plusieurs formes fermées
- Union : réalise l'union entre plusieurs formes fermées
- Différence : réalise la différence entre plusieurs formes fermées
- Xor : réalise l'opération xor entre plusieurs formes fermées
- Couper aux intersections : coupe tous les éléments à leurs intersections
- Miroir horizontal : effectue un retournement par rapport à l'axe vertical
- Miroir vertical : effectue un retournement par rapport à l'axe horizontal

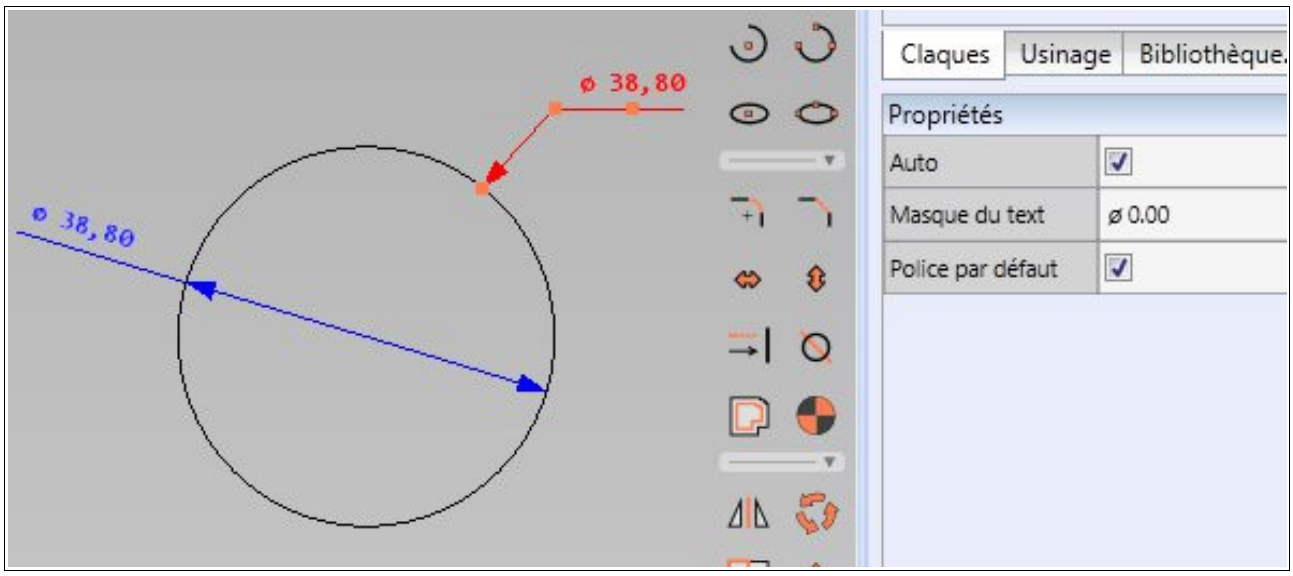
Cotations, annotations

Pour définir une cotation, il suffit de cliquer sur l'icône cotation, puis sur l'élément à coter. Si la cotation doit se faire entre 2 éléments distincts, il suffit de cliquer sur un point du premier élément, puis sur un point du deuxième élément.

Pour déplacer le texte de la cotation, cliquez sur le texte avec le bouton gauche de la souris, maintenez-le enfoncé, puis déplacez la souris. Pour déplacer la position de la cotation, cliquez sur le trait de cotation avec le bouton gauche de la souris, maintenez-le enfoncé, puis déplacez la souris.

Dans la fenêtre de propriétés, vous pouvez changer l'orientation et la position du texte par rapport au trait. Vous pouvez aussi modifier la police de caractère, le masque, ou bien mettre un texte libre. Enfin, vous pouvez changer les flèches, intérieur ou extérieur.





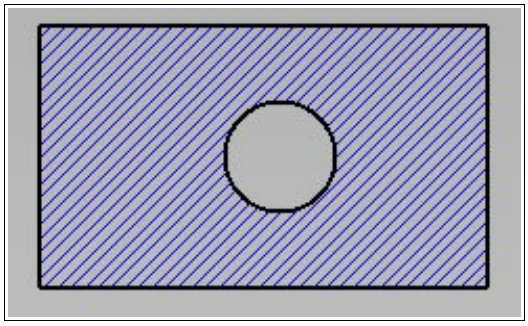
Astuce: Vous pouvez utiliser l'annotation pour faire une flèche. Enlevez simplement le texte dans ses propriétés.

▪ Hachures

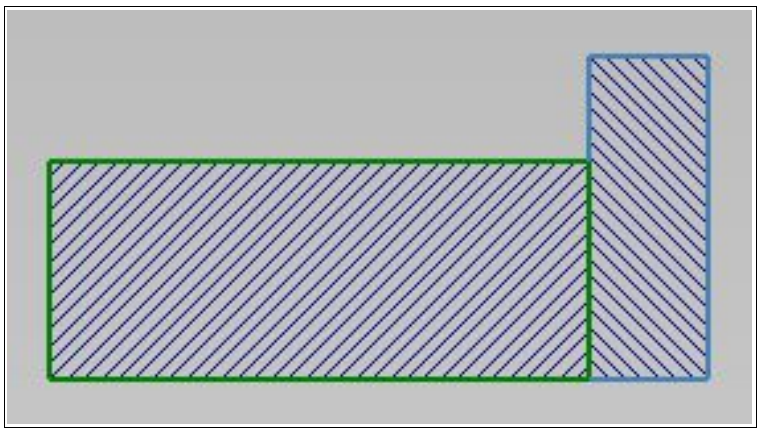
Pour créer une zone de hachure, il faut que les lignes créant le contour soient connectées entre elles. S'ils ne sont pas connectés, des cercles rouges apparaîtront aux positions posant problème.

Un fois les éléments, connectés, ils vous suffit de cliquer sur l'outil `[Hachure]` sur la barre d'icône en haut.

Vous avez 2 manières de décrire une zone.



Si la zone est distincte comme sur la figure précédente, vous pouvez sélectionner un trait du rectangle puis le cercle, et cliquer sur le bouton `[Auto]` qui s'affiche.



Cas particulier: Lorsque 2 zones se touchent, comme sur la figure précédente où des traits bleus viennent sur la zone verte, il faut alors sélectionner tous les traits de la zone à hachurer puis cliquer sur le bouton `[Manuel]`.

Note: si vous n'utilisez qu'un seul modèle de hachures, vous pouvez sélectionner plusieurs zones lors de la création d'une hachures.

Possibilité de modifier le dessin des hachures dans la fenêtre de propriétés.

Création de vos propres styles de hachures.

aller dans le répertoire : `[Mes Documents\GGSoft\GGCad2\Hatches]`

Vous y trouverez un fichier exemple `[sample.hatch]`.

Copier le puis modifiez-le. Sauvez-le avec un encodage UTF-8. Le fichier se compose d'une suite de style de ligne qui seront répétés.

Exemple 1: hachures à 45°, avec 2 mm entre chaque lignes :

```
name=45° \\ 2mm          # Nom
[LINE]                   # Déclaration d'une ligne
width=0.35               # Épaisseur de la ligne (en mm)
angle=45                 # angle en degrés
offset=0                 # départ par rapport au côté gauche (en mm)
```

```
over=2                  # espace entre 2 lignes (en mm)
space=False             # True si la première valeur de data est un espace
data=1                  # succession de valeurs trait,espace,trait,espace...
```

Exemple 2: hachures à 30°, avec une ligne continue et une ligne en pointillé :

```
name=45° \\ 2mm dot     # Nom
[LINE]                  # Déclaration d'une ligne
width=0.35              # Épaisseur de la ligne (en mm)
angle=30                # Angle en degrés
offset=0                # Départ par rapport au côté gauche (en mm)
over=4                  # Espace entre 2 lignes (en mm)
space=False             # True si la première valeur de data est un espace
data=1                  # Succession de valeurs trait,espace,trait,espace...

[LINE]                  # Déclaration d'une ligne
width=0.35              # Épaisseur de la ligne (en mm)
angle=30                # Angle en degrés
offset=2                # Départ par rapport au côté gauche (en mm)
over=4                  # Espace entre 2 lignes (en mm)
space=False             # True si la première valeur de data est un espace
data=0.72,0.35          # Succession de valeurs trait,espace,trait,espace...
```

Attention: Obligation de redémarrer GGCam pour prendre en compte le nouveau style.

▪ Style de trait

Les styles de traits sont appliqués sur chaque calque.

Création de vos propres styles de traits.

allez dans le répertoire : `[Mes Documents\GGSoft\GGCad2\Lines]`

Vous y trouverez un fichier exemple `[sample.line]`.

Le copier puis modifier et le sauvegarder avec un encodage UTF-8.

```
name=Sample             # Nom
space=False             # True si la première valeur de data est un espace
centered=False          # Non utilisé
data=3.5 3.5 7.5 3.5    # Succession de valeurs trait,espace,trait,espace...
```

Attention: Obligation de redémarrer GGCam pour prendre en compte le nouveau style.

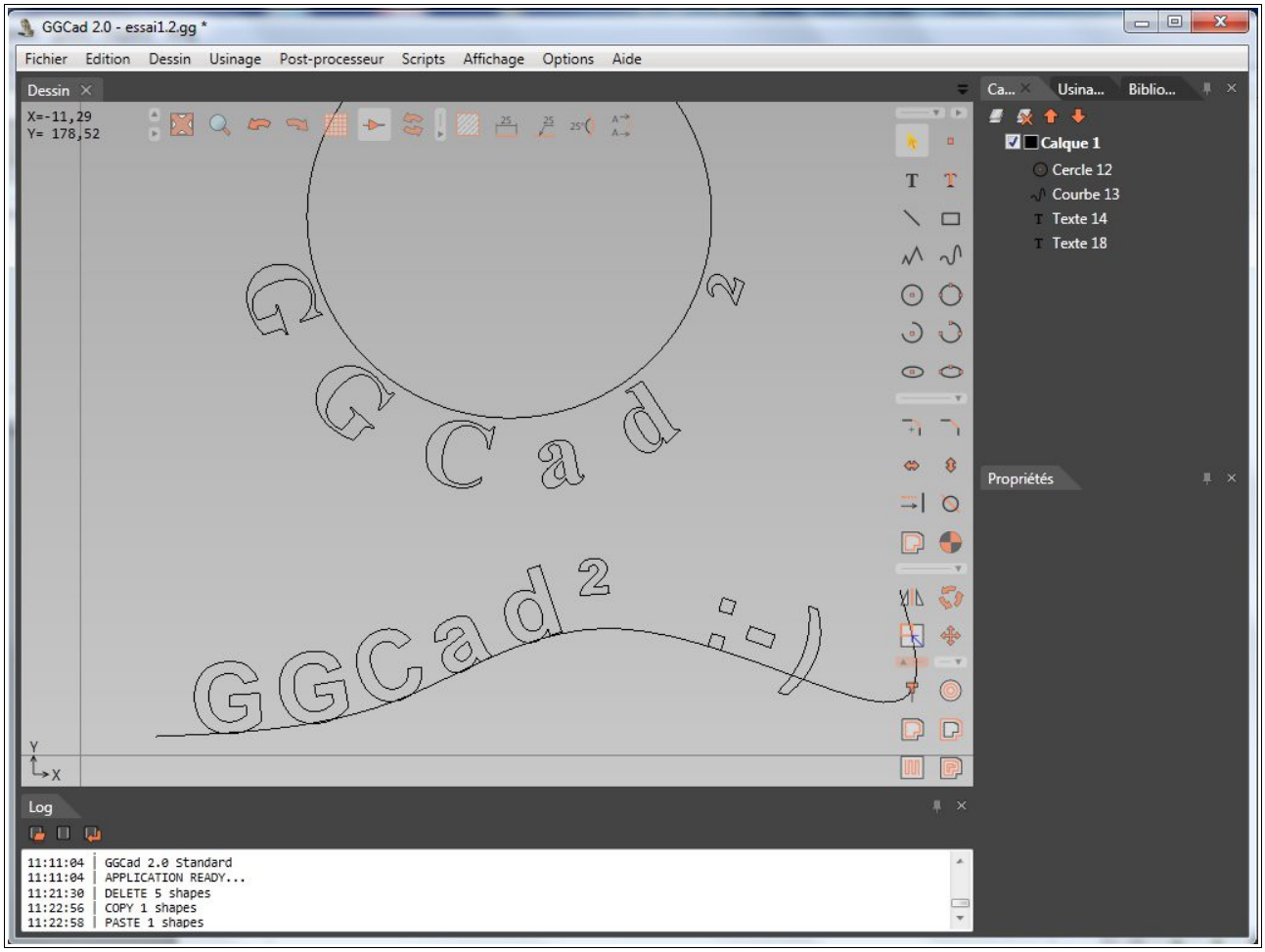
▪ Police

Les polices TrueType et Autocad SHP normales, bigfont et unicodes sont supportées nativement. Les polices Autocad SHX normales et unicodes sont supportées. Pour utiliser les poilces TrueType, cliquez sur l'outil [Texte]. Pour utiliser les polices SHP/SHX, cliquez sur l'outil [Contour de texte], puis parcourez la liste jusqu'aux polices commençant pas [SHP]. Les polices SHP/SHX sont à stoquer dans le dossier [Mes documents\GGSoft\GGCad2\SHp]

Contour de texte incurvé

L'écriture d'un texte peut suivre la forme de n'importe quelle ligne géométrique Il suffit de sélectionner le texte, de cliquer sur l'icône `[Lier]` sur la barre d'icône supérieur et cliquer cliquer sur la forme. Le texte peut alors être déplacé en cliquant sur son attache et en la déplaçant le long de la courbe.

La position du texte, dessus ou dessous la courbe, ainsi que le sens de lecture sont modifiables.



Groupes d'éléments

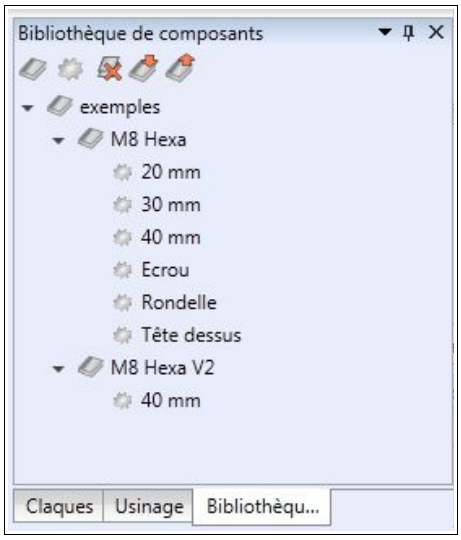
Possibilité de grouper/dégrouper les éléments entre eux:

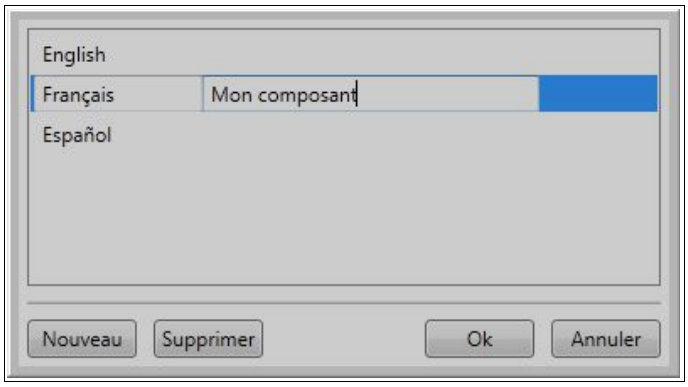
Sélectionner les éléments,et cliquer sur l'icône `[Grouper]` dans la barre d'icône supérieur.

Bibliothèque de composants

La fenêtre `[bibliothèque de composants]` vous permet de gérer les composants. Elle est

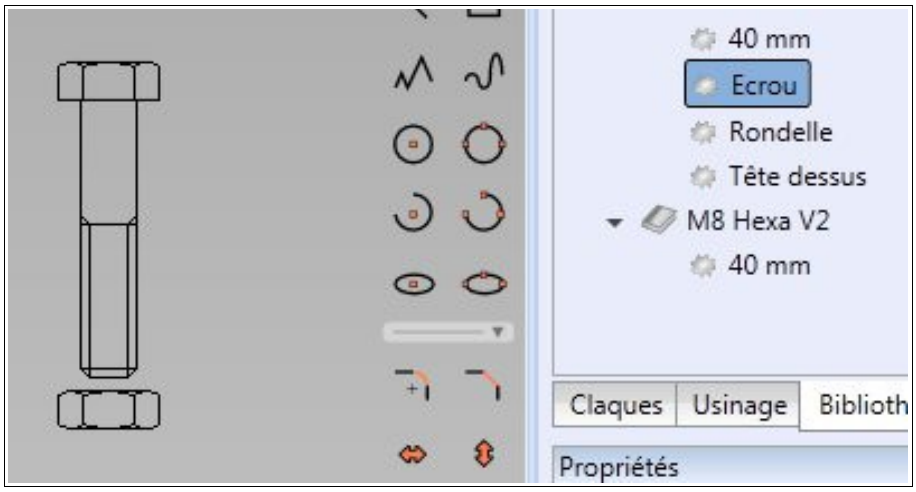
composées de dossiers. Chaque dossier peut contenir des composants et des dossiers. Pour ajouter un dossier, cliquez sur l'icône `[Créer un dossier]` dans la barre d'icône de la fenêtre. Une boîte de dialogue apparaitra pour le préciser le nom. Le nom est associé à une langue. Vous pouvez déclarer plusieurs nom/description en fonction de la langue.





Pour créer un composant, cliquez sur l'icône approprié, sélectionnez les éléments, sélectionnez le point de référence, puis entrez le nom.

Pour insérer un composant dans le dessin, il suffit de glisser/déposer le composant sur le dessin.



Vous pouvez importer/exporter les dossiers pour pouvoir les transférer d'un poste à un autre.

▪ Modèle

Pour créer un modèle de dessin, enregistrer le dessin par le menu `[Fichier][Enregistrer comme modèle...]`. Ainsi possibilité du choix du modèle lors d'un nouveau document. Les modèles sont stockés dans le dossier `[Mes Documents\GGSoft\GGCad2\Templates]`.

▪ Script

Vous pouvez exécuter des scripts écrits en langage Python. Les scripts sont accessibles par le menu [Script]. Vous pouvez les exécuter, modifier, créer de nouveaux. Voir le chapitre sur les scripts.

1.3 Impression

Pour imprimer, il faut créer des pages en y accédant par le menu `[Fichier][Gérer les pages d'impression]`. L'aperçu avant impression nécessite un lecteur de fichier PDF. Créer et centrer les pages sur le dessin. Visualiser les pages avec l'icône `[Pages]` sur la barre d'icônes du haut. Ajuster les paramètres.

Possibilité d'ajouter un facteur d'échelle pour imprimer par exemple un dessin fait pour le format A2 en A4 (facteur 0,5).

1.4 Importation

- DXF: les entités DIMENSION et HATCH ne sont pas supportés
- SVG: les remplissages ne sont pas supportés
- HPGL: support complet

1.5 Exportation

- DXF: format R12. Les dimensions ne sont pas exportés.
- SVG: seuls les contours et traits sont exportés
- HPGL: seuls les contours et traits sont exportés
- PDF

1.6 Dessin 3D

L'interface 3D est uniquement disponible dans la version professionnelle. Pour passer en mode 3D, il faut importer un modèle 3D. Différents formats sont supportés tel que le STL, 3DS, Collada...

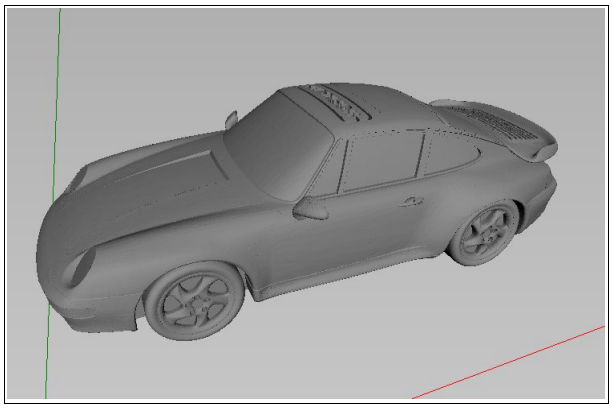
Dans l'espace 3D, il y a 2 types d'éléments :

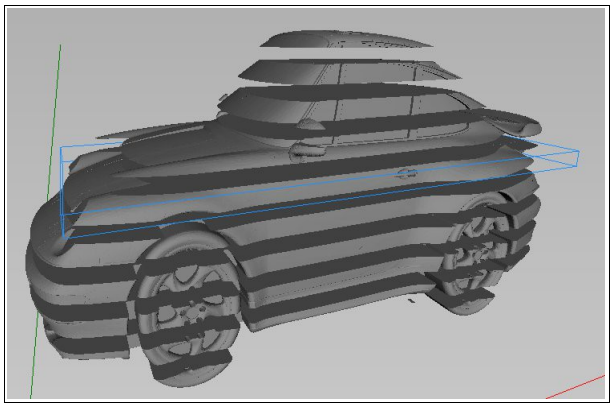
- Les éléments de type Face
- Les éléments de type Polygone/Polyligne

Un menu [Dessin] met à disposition les fonctions de translation, mise à l'échelle, rotation, mirroir.

Les autres fonctionnalités sont :

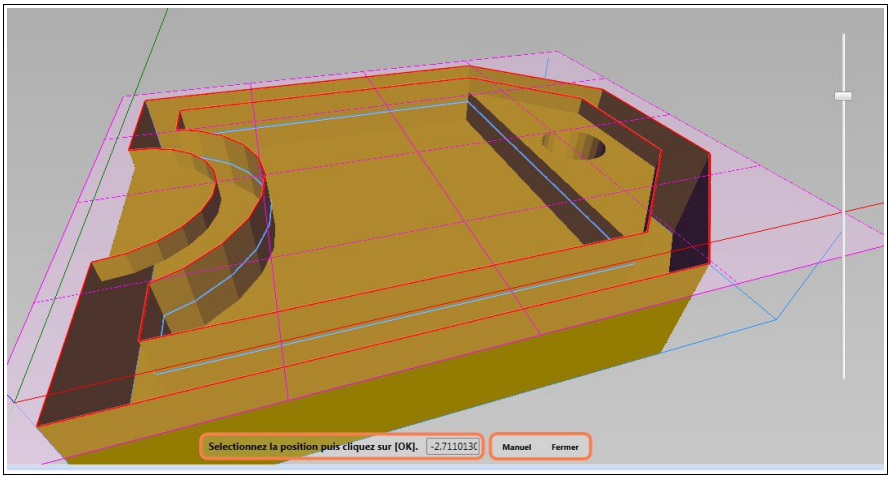
- Découper en tranches : permet de découper un objet en plusieurs tranches/strates, pour un usinage en plusieurs morceaux qui pourront être assembler.





- Ajouter un cube/cylindre : permet de rajouter ces types d'éléments
- Ajouter des attaches : permet d'ajouter des attaches sur un élément que l'on pourra déplacer autour grâce à un curseur. Ces attaches permettent de laisser la pièce fixée à la pièce brute après l'usinage.
- Séparer les polylignes : explose un élément polygon possédant plusieurs polylignes en plusieurs polygon, chacun possédant une seule polyligne.
- Fusionner les polylignes : permet de fusionner les polylignes en un seul polygon
- Exporter les polygones dans un fichier 2D : permet de passer les polylignes créées sur l'élément 3D pour dessiner ou usiner en mode 2D.
- Importer les polylignes d'un fichier 2D : permet de récupérer dans l'espace 3D des polylignes telles que du texte dans l'espace 3D.

Sur la barre d'outil se trouve l'icône [Créer un profil] permettant d'afficher un curseur pour créer les profils de la pièce. Déplacer le curseur pour détecter le profil souhaité.



2 Usinage

2.1 Généralités

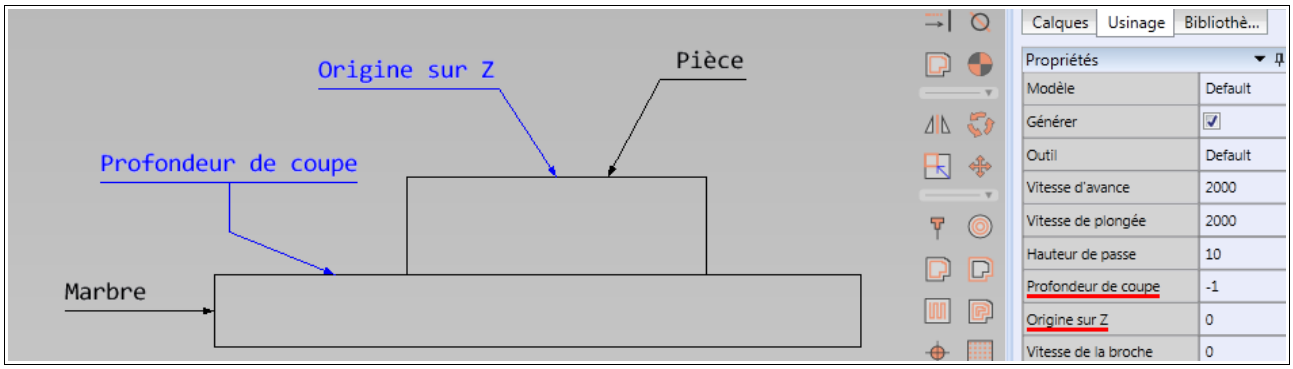
La barre d'outils de droite donne accès à plusieurs types d'usinages. Chaque usinage appartient à un groupe. Les groupes sont accessibles dans la fenêtre d'usinage. L'exécution du post-processeur se fera dans l'ordre représenté par l'arbre. L'ajout d'une opération se fait dans le groupe actif.

2.2 FAO

Groupe

- Application d'un modèle d'usinage au groupe.
- Les modèles d'usinages sont définis dans le menu `[Usinage][Modèles]`.
- Les paramètres d'un groupe sont :
- Outil (les outils sont définissables dans le menu `[Usinage][Outils]`)
- Vitesse d'avance : en unité/min
- Vitesse de plongée : en unité/min
- Vitesse de finition : en unité/min, vitesse utilisée pour la passe de finition sur le plan X-Y
- Offset de finition : en unité, distance, décalage à rajouter avant la passe de finition
- Profondeur : en unité, la position finale de l'outil sur l'axe Z

- Hauteur de passe : la hauteur maximale de l'outil dans la matière
- Origine sur Z : l'origine de référence de l'axe Z, position de la matière
- Position de retrait : position sur l'axe Z de remontée de la fraise lors des perçages
- Vitesse de la broche : vitesse de rotation
- Rotation de la broche en sens horaire/antihoraire
- Refroidissement : aucun/brouillard/liquide
- Attendre à la position Z basse : en seconde/milliseconde (suivant le contrôleur), l'outil restera ce temps de pause à la fin du perçage
- Vitesse constante : rend le déplacement plus doux
- Angle de rampe : angle en degrés de descente de l'outil dans la matière (0 = vertical)
- Répéter l'usinage sur l'axe X : répète automatiquement l'usinage n fois avec l'espacement spécifié
- Répéter l'usinage sur l'axe Y : répète automatiquement l'usinage n fois avec l'espacement spécifié
- Pre-commande : commande macro insérée par le post-processeur au début du groupe
- Post-commande : commande macro insérée par le post-processeur à la fin du groupe



Opérations

Toutes les opérations d'usinages possèdent des paramètres pre-commande et post-commande.

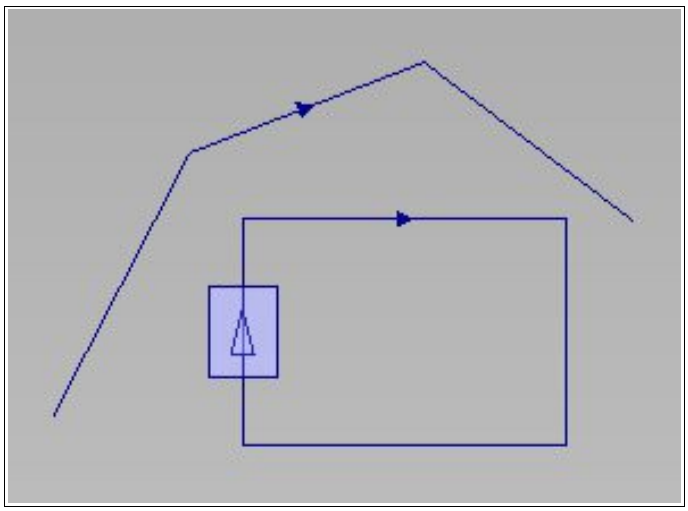
Elles sont respectivement insérées avant et après l'opération.

Pour la définition des contours, même méthodes que pour les hachures.

Gravure/Profile

L'opération de gravure/profile sur un ou plusieurs éléments définit le déplacement de l'outil le long des éléments. Si l'élément ou la forme est fermée, vous pouvez alors modifier l'origine en déplaçant la flèche le long de la forme. Si la forme est fermée, vous pouvez choisir un démarrage décalé

permettant de démarrer à côté de la pièce (par exemple pour l'amorçage au plasma). Vous pouvez aussi déclarer un nombre d'attaches pour laisser la pièce usinée attachée à la pièce brute. (voir les chapitres dédiés).



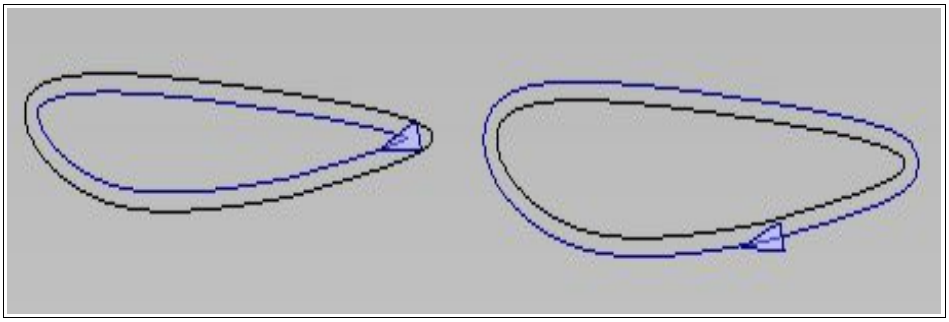
- **Spirale, Cercles**

L'opération spirale permet d'usiner l'intérieur d'un cercle suivant le tracé d'une spirale. Cela permet ainsi de garder un mouvement constant à l'outil. Le décalage de cercle permet lui de commencer dans un trou pré-percer et d'usiner avec une entrée décalée.

Si le groupe utilise la vitesse et l'offset de finition, un décalage supplémentaire sera effectué.

- **Décalage, offset**

L'opération de décalage permet de générer un parcours décalé à l'extérieur ou à l'intérieur d'une forme. La distance du décalage reste constante. Le décalage peut-être répété plusieurs fois.

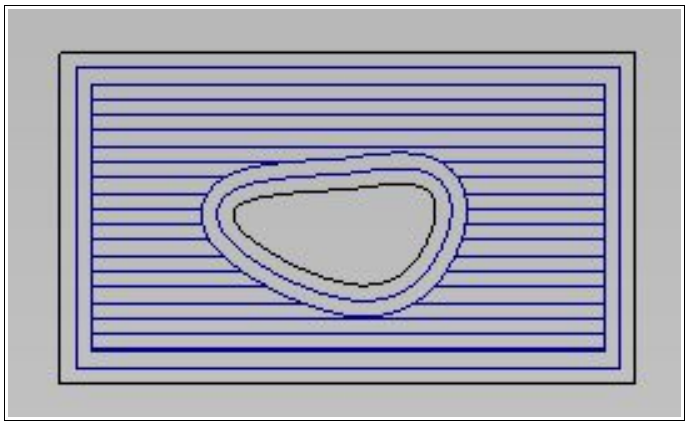


- **Zigzag**

L'opération zigzag permet un usinage de poche ou un surfaçage à l'aide d'aller-retours. Les

paramètres sont:

- Vertical: si coché, les zigzags sont verticaux, sinon horizontaux
- Récursif: applique l'usinage aux îlots dans les îlots
- Segments parallèles: si coché, l'outil se lève à chaque intersection avec les contours
- Inclure les contours: inclut l'usinage des contours à la fin des zigzags
- Offset outil: si coché, utilise les valeurs de l'outil sélectionnée

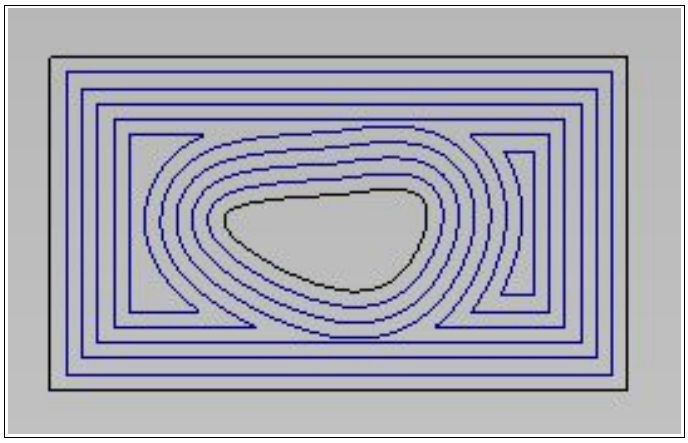


Si le groupe utilise la vitesse et l'offset de finition, un double décalage sera effectué.

Usinage de poche

L'opération usinage de poche génère les parcours par une succession de décalage des contours. Les paramètres sont:

- Récursif: applique l'usinage aux îlots dans les îlots
- Offset outil: si coché, utilise les valeurs de l'outil sélectionnée



Si le groupe utilise la vitesse et l'offset de finition, un double décalage sera effectué.

Perçage

L'opération de perçage génère les commandes de perçage. Si votre contrôleur ne peut pas interpréter les commandes standards, utilisez la commande manuelle.

Un perçage va faire descendre l'outil à une position Z, peut marquer une pause puis remonter à la position de retrait (R).

- Perçage: l'outil va descendre à Z puis remonter à R. Si le groupe a une hauteur de passe inférieur à Z, l'opération sera répétée sur l'axe Z, commande G81 ou G82.
- Perçage avec débourrage: même que précédemment, commande G83.
- Manuel: même que précédemment mais commande G0 et G1.
- Taraudage: Commande G84
- Alésage: Commande G85

Le perçage se fait sur les éléments de dessin de type Point, Cercle, Arc.

Grille de perçage

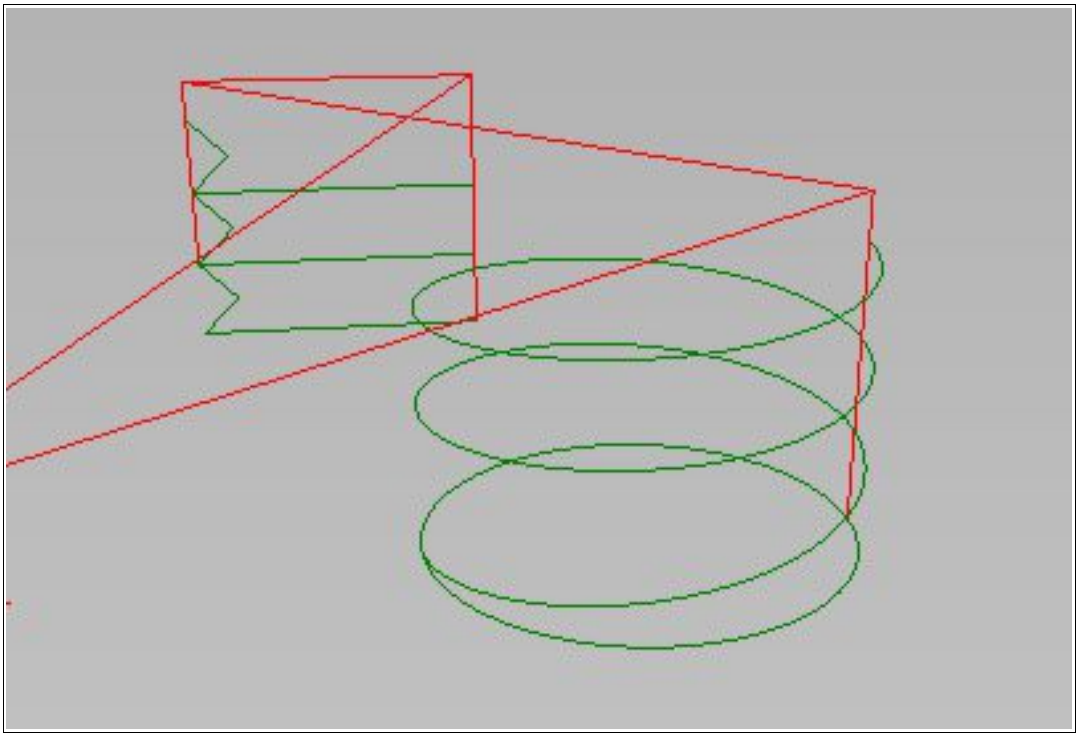
La grille de perçage permet de créer une liste de perçage en suivant un profil ou en remplissant un contour.

Gravure globale

Par le menu `[Usinage][Graver le calque courant]`, création de l'opération qui génère un parcours sur tous les éléments du même niveau.

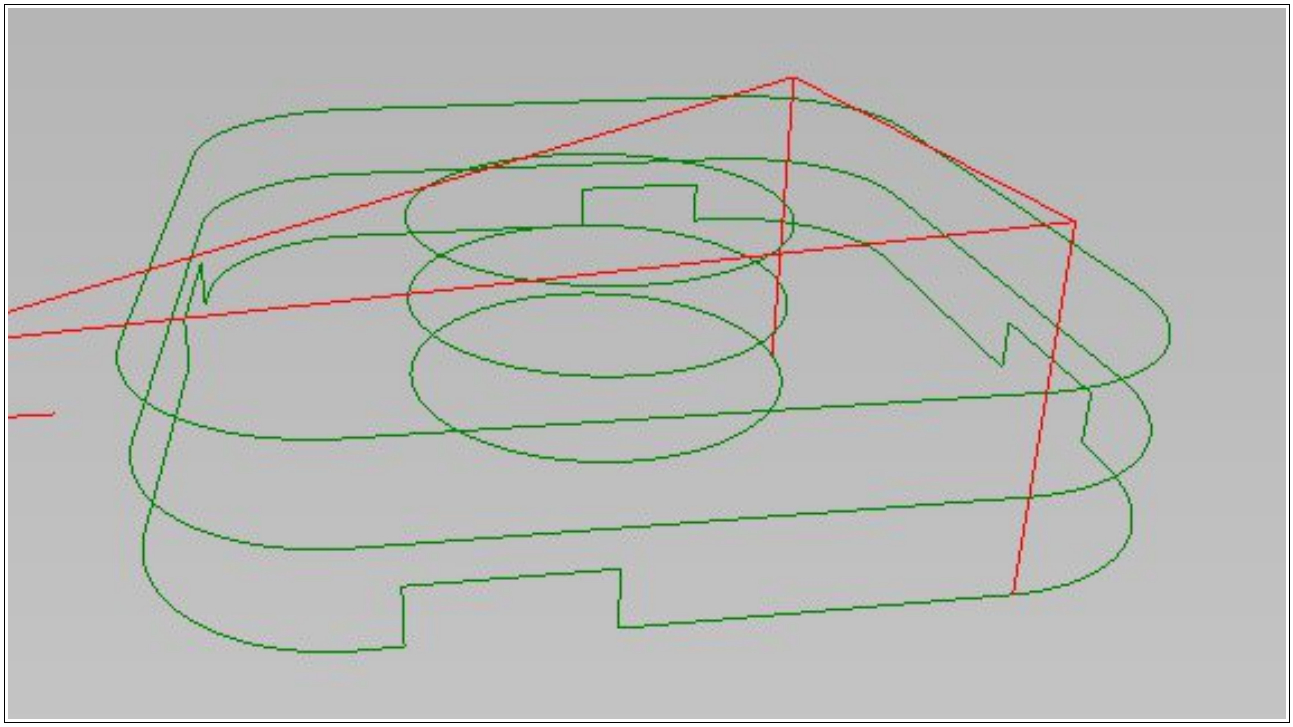
Rampe

Dans les propriétés du groupe, le paramètre [Angle de rampe] permet de faire pénétrer l'outil avec un angle (en degré) dans la matière. Si le chemin est fermé, il descendra alors régulièrement avec une pente constante entre le début et la fin, puis terminera par un tour complet à la profondeur de passe.

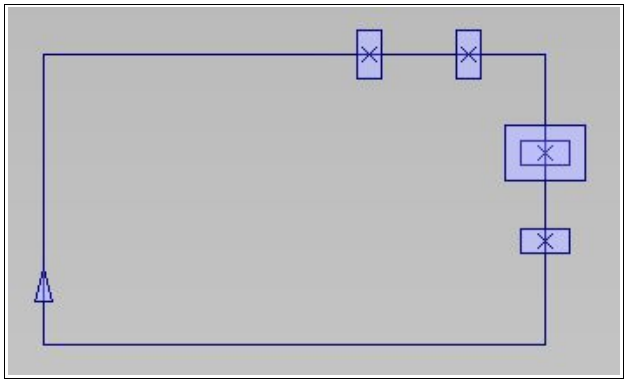


- **Attaches**

L'usinage de certaines pièces nécessitent de laisser la pièce attachée à la pièce brute. Pour cela, nous allons laisser des attaches entre le brute et la pièce usinée. Les usinages type profil, offset, cercles possèdent un paramètre [Attaches]. Définissez le nombre d'attaches, la hauteur, la longueur.

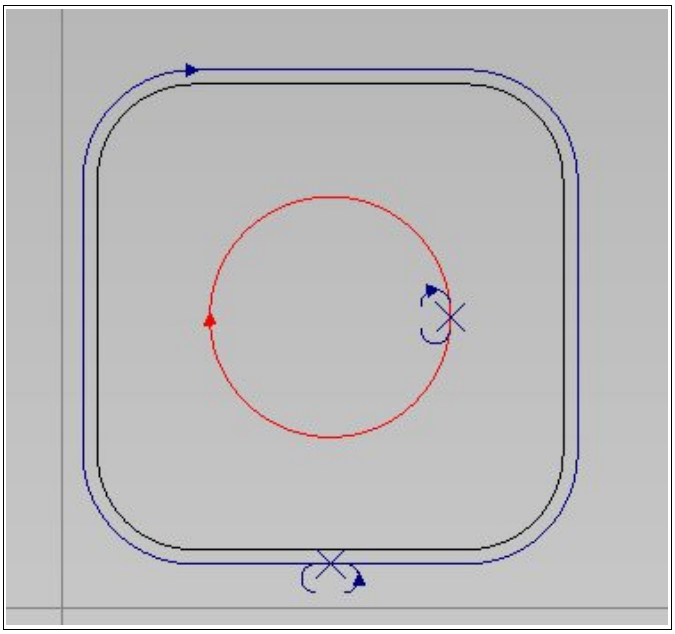


Advanced, Professional : vous pouvez déplacer les attaches en cliquant sur celles-ci puis en déplaçant la souris.



- **Départ décalé**

Certains outils comme les torches d'oxycoupage, de plasma laissent des bavures lors de l'amorçage. Pour avoir un bon résultat, le paramètre [Amorçage décalé] permet de décaler l'outil sur le côté pour amorcer, puis de revenir sur le tracer.

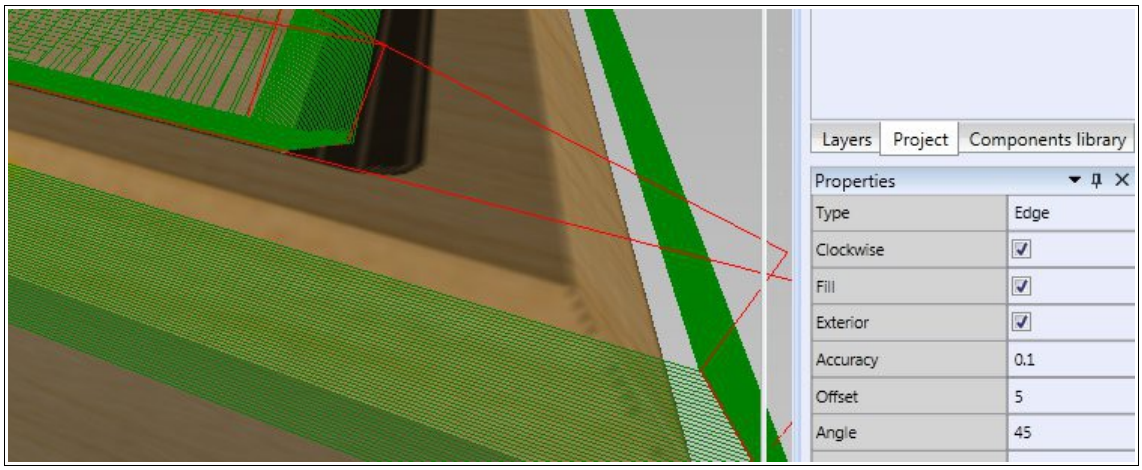


Advanced, Professional : vous pouvez déplacer les attaches en cliquant sur celles-ci puis en déplaçant la souris.

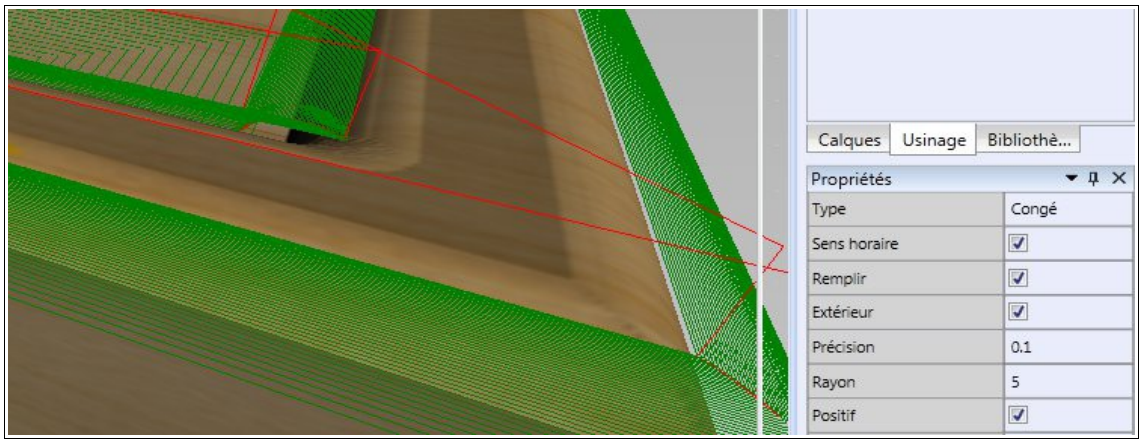
Attention : vous ne pouvez pas faire un départ décalé en même temps que des attaches. Si vous renseignez le paramètre « Nombre d'attaches » au dessus de 0, le paramètre « Départ décalé » disparaîtra. Inversement si Si vous renseignez le paramètre « Départ décalé » au dessus de 0, le paramètre « Nombre d'attaches » disparaîtra des propriétés de l'usinage.

▪ Chanfrein / Congé (Advanced, Professional)

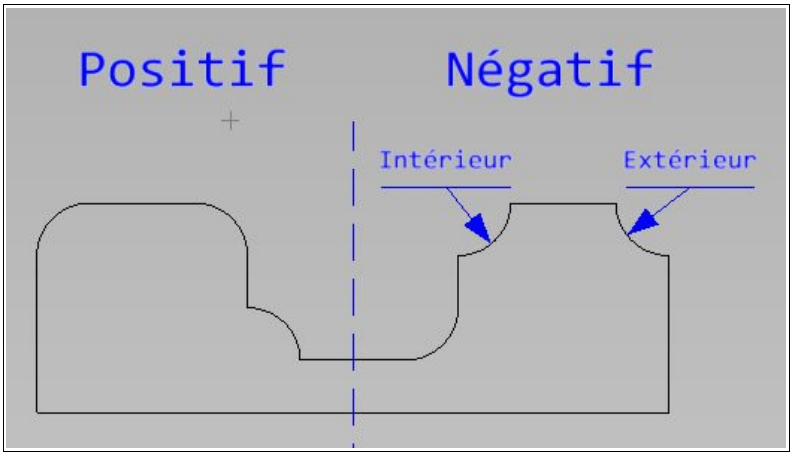
L'opération `[Chanfrein]` permet de créer un chanfrein ou un congé en 3 dimensions. Le chanfrein peut être intérieur ou extérieur par rapport au contour donné. Le congé peut-être extérieur ou intérieur, positif ou négatif.



Le paramètre `[Précision]` définit le déplacement sur l'axe Z entre 2 courbes. Ici 0,1 mm. `[Offset]` est la distance du bord de la pièce, ex : 5 mm, et `[Angle]` est l'angle du chanfrein en degrés.

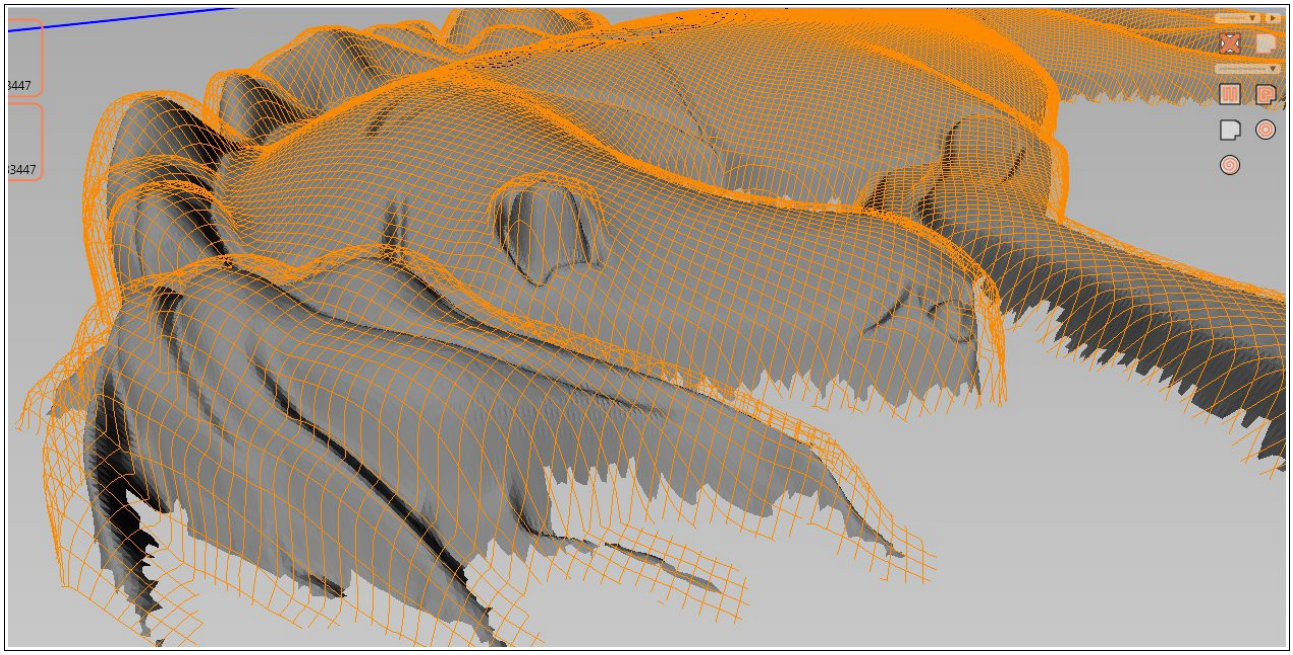


Le congé peut-être positif ou négatif.



Les outils suivants concernent la version **Professional**

Zigzag



L'opération `[Zigzag]` génère le parcours d'outil suivant une trajectoire en zigzag. Si un ou plusieurs éléments de type Face sont sélectionnés, le parcours se fait à l'intérieur d'un espace cubique. Cet espace est par défaut l'espace englobant les éléments. Il peut-être modifier en changeant les paramètres `[Dimensions]` et `[Offset]`.

- `Dimensions` : largeur, longueur, hauteur.
- `Offset` :position de l'origine de l'espace sur x, y, z.
- `Étendre sur (x,y)` : permet de décaler automatiquement à l'extérieur ou à l'intérieur en fonction du rayon de l'outil ou de valeurs personnelles sur x et y.

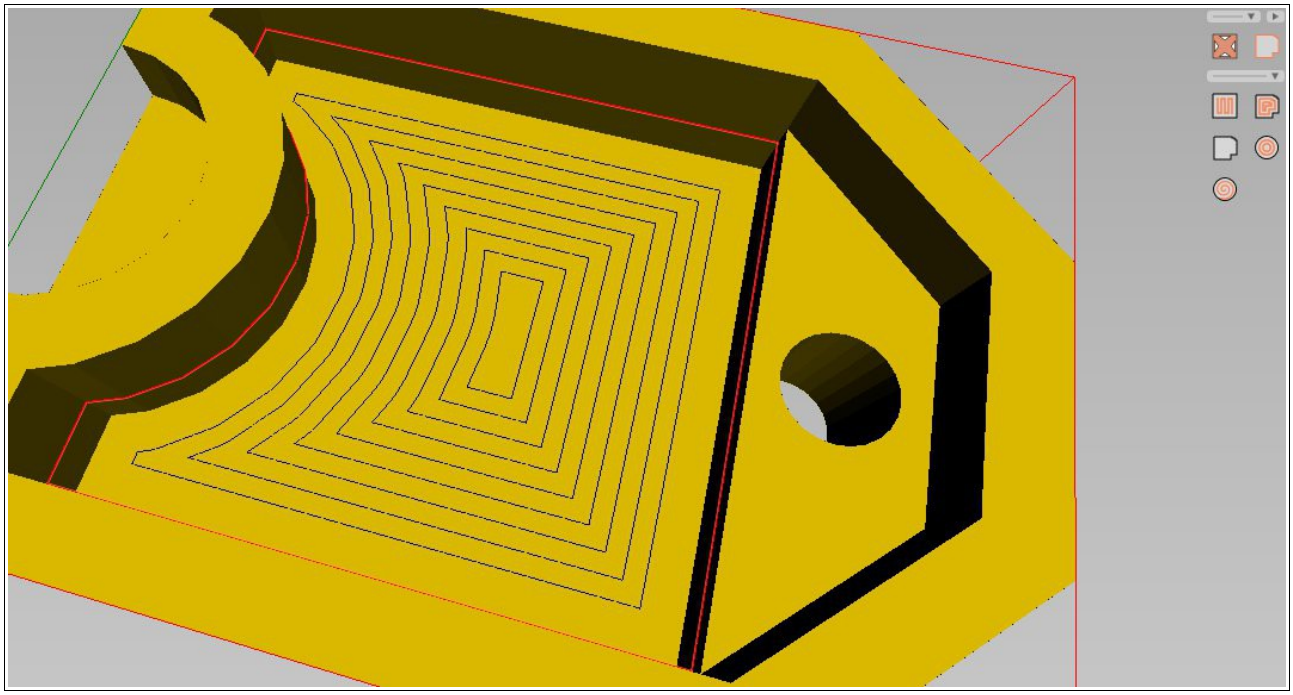
Si l'opération s'effectue sur un ou plusieurs éléments Face et Polygon, l'usinage va crée un offset du ou des Polygons puis faire le parcours en fonction de ces offsets.

- `Récursif` : s'il coché, calcul sur tous les îlots impairs imbriqués.
- `Inclure les contours` : si coché, ajoute le parcours des contours pour l'usinage.
- `Offset double` : si coché, double l'offset pour avoir un meilleur état de surface des contours.
- `% offset double` : pourcentage du rayon de l'outil pour le 2ème offset.

Paramètres communs aux 2 types de zigzag :

- `Mode` : vertical ou horizontal.
- `Segments parallèles` : si oui, il n'y a pas de va et vient.
- `Surfaçage` : si coché, usine les parties sans éléments.
- `Hauteur de surface` : limite la hauteur d'usinage au dessus de la pièce (ex : 1, l'outil se déplacera à une hauteur constante de 1mm au dessus de la pièce).
- `Recouvrement` : l'outil se décale de la valeur de recouvrement.

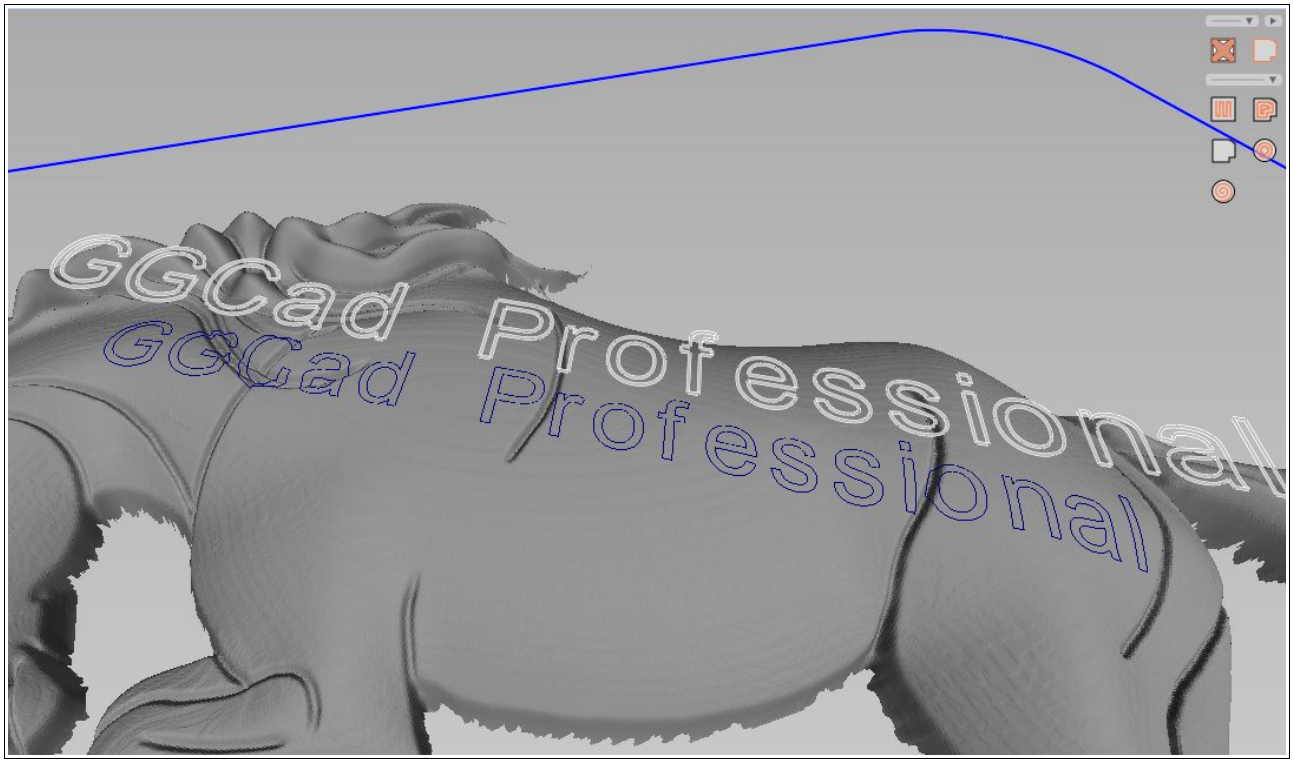
Offset, décalage



L'opération `[Décalage]` permet de faire un décalage successif d'un Polygon. Si la méthode utilisée est la `[Projection]`, le parcours est projeté sur la surface. Si la méthode est `[Multipasse]`, le parcours est calculé sans prise en compte des éléments Faces.

- `Sens horaire` : sens d'usinage.
- `Hauteur de surface` : limite la hauteur d'usinage au dessus de la pièce (ex : 1, l'outil se déplacera à une hauteur constante de 1mm au dessus de la pièce).
- `Recouvrement` : l'outil se décale de la valeur de recouvrement.
- `Étendre sur (x,y)` : permet de décaler automatiquement à l'extérieur ou à l'intérieur en fonction du rayon de l'outil ou d'une valeur personnelle.

- **Profil**



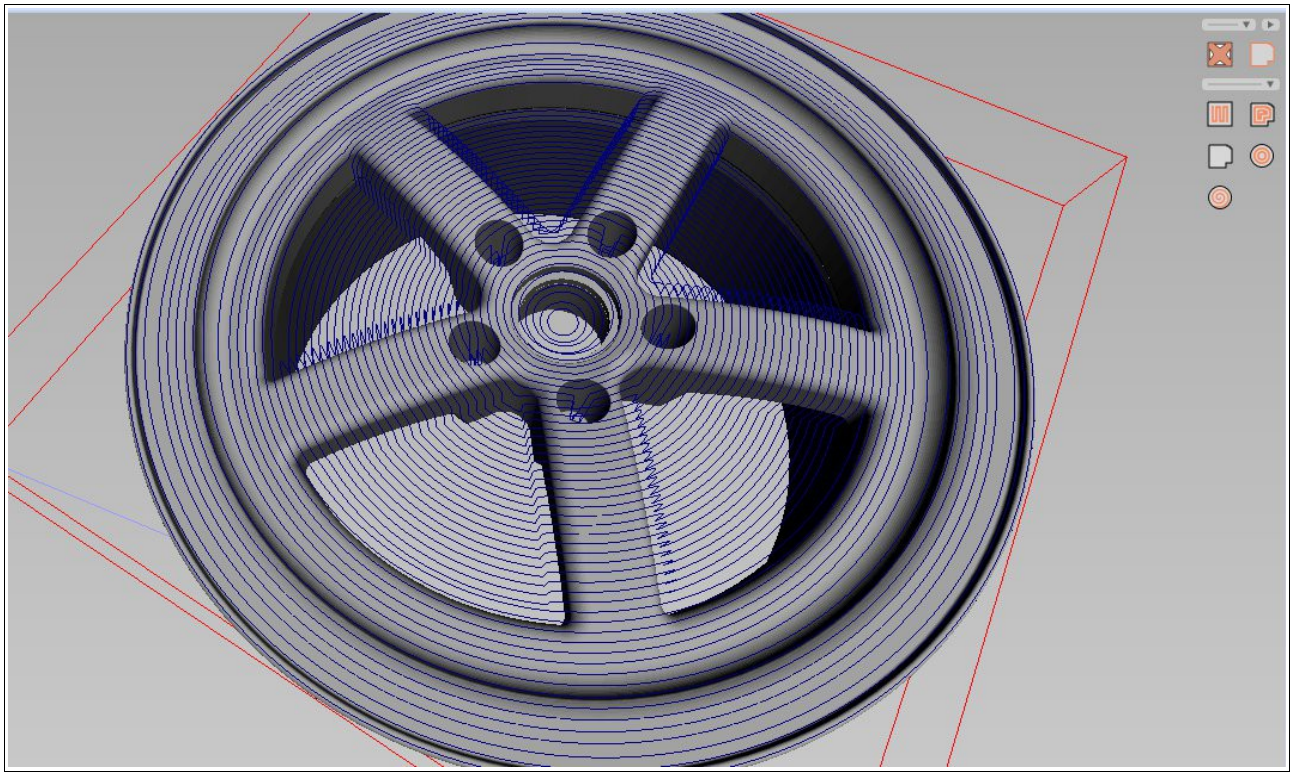
L'usinage de type `[Profil]` s'effectue sur un élément Polygon. Il dispose de 2 méthodes :

- Si seul des Polygons sont sélectionnés : l'outil suit le profil, ou le décale à l'intérieur ou à l'extérieur
- Si Polygons et Faces : le parcours du profil est projeté sur la surface.

Les paramètres :

- `Profondeur de coupe` : la profondeur est soustraite au parcours de l'outil. Par exemple, si la profondeur est 2mm, l'outil descendra au parcours calculé, puis descendra de 2mm sur toutes les positions sur Z du parcours.
- `Étendre sur (x,y)` : permet de décaler automatiquement à l'extérieur ou à l'intérieur en fonction du rayon de l'outil ou d'une valeur personnelle.

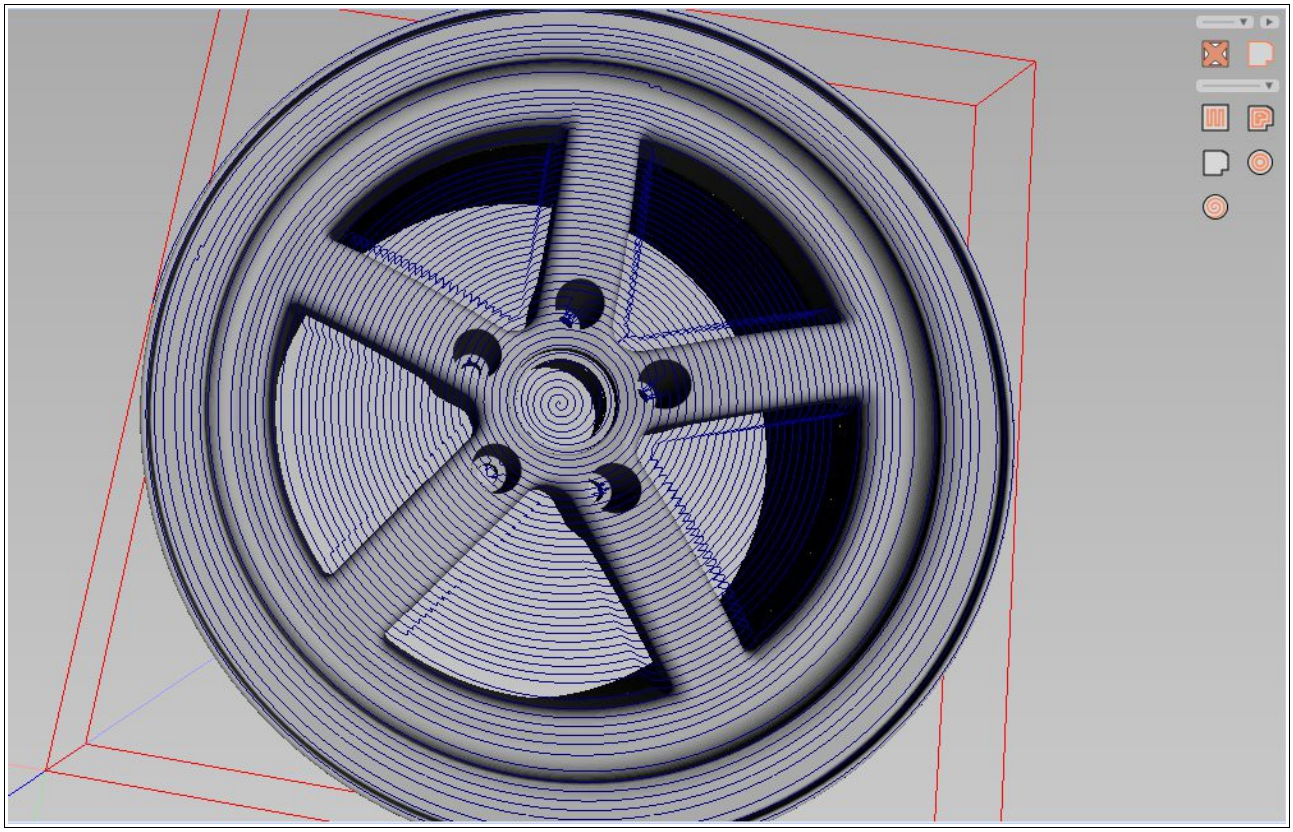
Cercle



L'usinage de type `[Cercle]` permet de créer les parcours d'outil de façon circulaire.

- `Dimensions` : largeur, longueur, hauteur.
- `Offset` :position de l'origine de l'espace sur x, y, z.
- `Étendre sur (x,y)` : permet de décaler automatiquement à l'extérieur ou à l'intérieur en fonction du rayon de l'outil ou de valeurs personnelles sur x et y.
- `Surfaçage` : si coché, usine les parties sans éléments.
- `Hauteur de surface` : limite la hauteur d'usinage au dessus de la pièce (ex : 1, l'outil se déplacera à une hauteur constante de 1mm au dessus de la pièce).
- `Recouvrement` : l'outil se décale de la valeur de recouvrement.
- Rayon central : permet de limiter le centre à un rayon de départ.

- **Spirale**

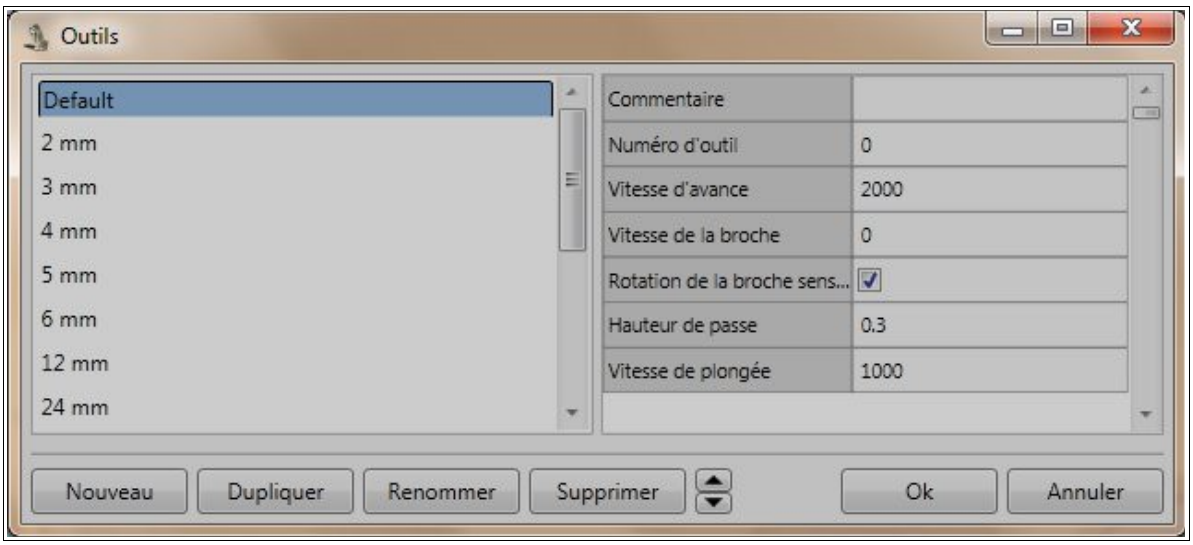


L'usinage de type `[Spirale]` permet de créer les parcours d'outil en suivant une spirale.

- `Dimensions` : largeur, longueur, hauteur.
- `Offset` :position de l'origine de l'espace sur x, y, z.
- `Étendre sur (x,y)` : permet de décaler automatiquement à l'extérieur ou à l'intérieur en fonction du rayon de l'outil ou de valeurs personnelles sur x et y.
- `Surfaçage` : si coché, usine les parties sans éléments.

- `Hauteur de surface` : limite la hauteur d'usinage au dessus de la pièce (ex : 1, l'outil se déplacera à une hauteur constante de 1mm au dessus de la pièce).
- `Recouvrement` : l'outil se décale de la valeur de recouvrement.
- Rayon central : permet de limiter le centre à un rayon de départ.

2.3 Outils



Les outils sont gérés par le menu `[Usinage][Outils]`. Ils sont après accessible dans le groupe. L'outil possède les paramètres standard, vitesse d'avance, de plongée, diamètre... Il possède des paramètres particuliers:

- Vitesse de plongée initiale: l'outil descendra à cette vitesse avant d'utiliser la vitesse de plongée.
- Profondeur en vitesse initiale: la profondeur d'utilisation de la vitesse de plongée initiale.

Pour associer une image particulière à un outils, ajoutez dans le répertoire des outils une image avec le même nom que l'outil.

Vous pouvez créer plusieurs magasins d'outils. Le mode de sélection des outils vous permet de :

- Lister tous les outils dans le groupe, puis en fonction de l'outil, afficher le magasin correspondant.
- Lister tous les magasins dans le groupe, puis en fonction du magasin, afficher uniquement les outils du magasin.

2.4 Modèle d'usinage

Les modèles d'usinage sont gérés par le menu `[Usinage][Modèle]`. Vous y retrouvez les paramètres d'un groupe.

2.5 Commandes

Les commandes sont gérés par le menu `[Usinage][Commandes]`. Elles sont utilisables dans les pré et post-commandes.

Il y a 2 types de commandes:

- **Macro:** La commande contient du texte et des variables qui seront remplacées à la génération.

Les variables sont:

```
{safe}       : position de sécurité
{diameter}   : diamètre de l'outil
{radius}     : rayon de l'outil
{length}     : longueur de l'outil
{tail}       : longueur de la queue de l'outil
{plung}      : vitesse de plongée de l'outil
{number}     : numéro de l'outil
{name}       : nom de l'outil
{speed}      : vitesse de rotation de l'outil
{pass}       : hauteur de passe de l'outil
{feed}       : vitesse d'avance de l'outil
{clockwise}  : sens de rotation de l'outil
```

Exemple avec l'outil numéro 2:

```
G45 T{number} H{number}
```

Résultat :

```
G45 T2 H2
```

- **Script Python:** La commande contient un script python.

Un objet `[post]` permet d'interroger le post-processeur durant l'exécution.

Exemple:

```python
def Home():                     # déclaration de la commande
    post.comment('début home') # écrit un commentaire
    post.rapid_z(50)           # déplacement rapide sur Z=50
    post.rapid_xy(0,0)         # déplacement rapide à X=0 Y=0
    post.rapid_z(post.safe)    # déplacement rapide à la position de sécurité
    post.comment('fin home')   # écrit un commentaire
```

Consulter le chapitre sur l'interface post-processeur

2.6 Choisir son Post-processeur

Les post-processeurs sont écrits en langage Python. C'est un langage de programmation complet. Il permet donc de faire toutes les adaptations envisageables. Les post-processeurs sont accessibles par le menu `[Post-processeur]`. Vous pouvez générer, simuler, sélectionner, configurer, éditer, créer de nouveau post-processeurs. Le post-processeur génère le fichier dans le répertoire `[Mes Documents\GGSoft\GGCad2\Output]`. Vous pouvez changer ce répertoire dans les préférences de

GGCam.

▪ Générique / Fraiseuse

Ce post-processeur supporte les commandes standard FANUC et permet de générer du code pour la grande majorité des commandes numériques.

Ses options vous permettent de numéroter les lignes, de générer du code condensé (supprime les espaces), de spécifier le nombre maximum de chiffres après la virgule, de spécifier l'extension du fichier et de supprimer les commentaires.

▪ Générique / Plasma-Laser

Ce post-processeur supporte les commandes standard FANUC et permet de générer du code pour la grande majorité des commandes numériques. Il dispose des mêmes options que le post-processeur précédent. Il gère aussi des paramètres spécifiques:

- Commande d'amorçage: spécifiez la commande pour amorcer/démarrer l'appareil
- Commande d'arrêt: spécifiez la commande pour arrêter l'appareil
- Temporisation: marque une pause juste après l'amorçage
- Mode hauteur: si coché, actionne cette fonctionnalité
- Hauteur: si mode hauteur, déplace l'outil à cette hauteur avant d'amorcer, puis redescend à la position d'usinage

▪ Mach3

Les post-processeurs pour Mach3 dérive des génériques. L'extension du fichier de sortie est [*.tap].

▪ HPGL/PLT

Génère un fichier HPGL pour les tables traçantes, les tables de découpe.

2.7 Créer un Post-Processeur

Les post-processeurs livrés avec GGCam ne sont pas modifiables. Leurs fichiers se trouvent dans le dossier [Mes Documents\GGSoft\Processor]. Il est possible de copier et modifier un post-processeur comme le [PostTemplate.py] qui sert de modèle. Les fichiers sont enregistrés en UTF8. Les fichiers pré-installés sont systématiquement réécrits au lancement de GGCam.

Choisissez le post-processeur modèle, menu [Post-processeur][Spécifique][Modèle vide], éditez-le, et changez la description ligne 13:

```
self.Français = {'descriptor': 'test'}
```

Enregistrez-le sous [test.py]. Un nouveau menu c'est créé [Post-processeur][Spécifique][test]. Sélectionnez-le. Le post-processeur est actif et prêt à utiliser. Faites vos modifications, enregistrez le fichier. Il est recompilé automatiquement. Si lors de la compilation ou de la génération des erreurs se produisent, les messages correspondant apparaîtront dans le fenêtre de log (le journal). Les fonctions commencent par `def`.

Le fichier de définition

Le fichier comporte une classe toujours appelée `[Post]`. La classe comporte plusieurs fonctions appelées à l'initialisation:

`def __init__(self):`
Ne pas modifier.

`def initilization(self):`
Faites ici l'initialisation de vos variables, objets...
Un dictionnaire est pré-créé pour stocker les chaines de caractères. Remplissez le dictionnaire ici.

`def set_country(self, country_name):`
GGCam va appeler cette fonction pour lui passer en paramètre dans country_name le nom de la langue actuellement utilisée. Exemple : `English`.

`def get_description(self):`
Retourne le nom du post-processeur.

`def get_menu(self):`
Retourne le type de menu : `'MILL'`, `'PLASMA'`, `'TURN'`, `'OTHER'`

`def get_codepage(self):`
Retourne le codepage utilisé pour écrire le fichier. Généralement `'ANSI'`. Il peut être aussi `'UTF8'`, `'UTF16'`, `'ASCII'`, ou un nombre entier définissant un codepage Windows (ex : 1252).

`def get_extention(self):`
Retourne l'extension du fichier de sortie (ex : `'txt'`).

`def get_decimal(self):`
Retourne le nombre maximum de chiffre après la virgule.

Le post-processeur permet d'afficher une boîte de propriétés pour sa configuration. Pour cela, il dispose de plusieurs fonctions :

`def get_properties_count(self):`
Retourne le nombre de propriétés à configurer.

`def get_property_description(self, index):`
Retourne la description de la propriété. Le paramètre `[index]` est le numéro de la propriété. La première propriété commence à l'index `'0'`.

`def get_property(self, index):`
Retourne la valeur de la propriété à `[index]`.

`def set_property(self, index, val):`

Passe la nouvelle valeur `[val]` de la propriété `[index]`.

`def hidden(self):`
Vous pouvez cacher certaines propriétés en fonction d'autres grâce à cette fonction. Elle doit renvoyer un tableau d'entier `int` contenant les indexes à cacher.

Le post-processeur lors de la génération va appeler les fonctions qui suivent. Ces fonctions prennent les données du post-processeur, et doivent lui retourner un chaîne de caractères.

`def ijk_relative(self):`
Indique au post si les paramètres i, j, k doivent être en coordonnées relatives (`True`) ou absolue (`False`).

`def begin(self):`
Est appelé au début de la génération.

`def end(self):`
Est appelé à la fin de la génération.

`def initial_position(self):`
Est appelé pour placer l'outil à une position initiale, généralement (0,0,safe)

`def final_position(self):`
Est appelé pour placer l'outil à une position finale à la fin du fichier, généralement (0,0,safe)

`def comment(self, data):`
Retourne `[data]` sous forme de commentaire.

`def new_line(self):`
Retourne le texte en début de ligne (ex : numérotation de ligne).

`def start_process(self):`
Appelé après un positionnement rapide en (X,Y). Start et Stop encadre un cycle de déplacement à la vitesse de travail.

`def stop_process(self):`
Appelé avant un déplacement rapide.

`def rapid_position(self, z):`
Appelé juste avant une plongée.

`def rapid(self, axes, values):`
Déplacement rapide. `[Axes]` est un tableau de caractères contenant les axes X, Y, Z, A, B... et les constantes I, J, K. `[Values]` est un tableau de même dimension contenant les valeurs de type `float` associées à chaque axe. (Note : si la valeur d'un axe n'a pas bougé, il n'est pas renvoyé au déplacement suivant).

`def linear(self, axes, values):`
Déplacement linéaire.

`def clockwise(self, axes, values):`
Déplacement circulaire dans le sens horaire.

`def counter_clockwise(self, axes, values):`
Déplacement circulaire dans le sens antihoraire ou trigonométrique.

`def pause(self, time):`
Demande une pause de `[time]` unité.

`def program_pause (self):`
Demande une pause du programme.

`def tool(self, number):`
Changement d'outil. `[number]` contient le numéro d'outil.

`def feed(self, speed):`
Positionne la vitesse d'avance.

`def start_spindle_clockwise(self):`
Démarre la broche dans le sens de rotation horaire.

`def start_spindle_counter_clockwise(self):`
Démarre la broche dans le sens de rotation anti-horaire.

`def stop_spindle(self):`
Arrête la broche.

`def spindle_speed(self, speed):`
Spécifie la vitesse de la broche.

`def stop_program(self):`
Arrêt du programme.

`def exact(self):`
Déplacement exact.

`def constant(self):`
Déplacement adouci.

`def absolute(self):`
Coordonnées en absolue.

`def relative(self):`
Coordonnées en relatif. (Note : pas utilisé)

`def rate_inverse_time(self):`
Vitesse d'avance en inverse sur le temps. (Note : pas utilisé)

`def rate_minute(self):`
Vitesse d'avance en unités par minutes.

`def rate_revolution(self):`
Vitesse d'avance en tour par minutes.

`def inch(self):`
Unité en pouces

`def millimeter(self):`
Unités en millimètres.

`def tool_left_radius_compensation(self):`
Compensation de rayon à gauche.

`def tool_right_radius_compensation(self):`
Compensation de rayon à droite.

`def tool_cancel_radius_compensation(self):`
Arrêt de la compensation de rayon.

`def tool_length_compensation(self):`
Compensation de la longueur de l'outil.

`def tool_cancel_length_compensation(self):`
Arrêt de la compensation de la longueur de l'outil.

`def drill(self, axes, values, retract, pause):`
Perçage. `[retract]` contient la position de retrait. `[pause]` contient le temps de pause.

`def peck_drill(self, axes, values, retract, delta):`
Perçage avec débourrage. `[retract]` contient la position de retrait. `[delta]` contient la hauteur de passe.

`def tapping(self, axes, values, retract):`
Alésage.

`def boring(self, axes, values, retract, pause):`
Taraudage.

`def mist(self):`
Refroidissement par brouillard.

`def flood(self):`
Refroidissement par liquide.

`def stop_coolant(self):`
Arrêt du refroidissement.

L'interaction avec le moteur de génération

Le post-processeur peut-être lui-même appelé et propose une interface accessible par l'objet `[post]`. Il peut être appelé dans le fichier du post-processeur lui-même, et dans les commandes. Cela permet de travailler sur les coordonnées par le post. Ces fonctions peuvent être paramétrées. Les fonctions commencent par `def`, contrairement aux propriétés.
Si par exemple, vous implémentez une fonction `[home]` tel que:

```
def home() :
      return 'G1 X' + str( post.x + 1)
```

La fonction va déplacer l'outil sur x+1, donc la nouvelle position sera x+1 mais la position courante dans le post-processeur restera x. Il est donc préférable d'écrire:

```
def home() :
      post.linear_x(post.x + 1)
```

L'objet `[post]` publie l'interface suivante:

Propriétés

`x` : retourne la position x courante.

`y` : retourne la position y courante.

`z` : retourne la position z courante.

`a` : retourne la position a courante.

`b` : retourne la position b courante.

`c` : retourne la position c courante.

`safe` : retourne la hauteur de sécurité.

`color` : retourne la couleur du groupe.

`feed_rate` : retourne la vitesse d'avance courante.

`plung_rate` : retourne la vitesse d'avance en plongée courante.

Fonctions

`def is_millimeter()`: retourne `True` si l'unité est le millimètre.

`def is_inch()`: retourne `True` si l'unité est le pouce.

`def log(s)`: écrit dans le journal.

`def rapid_x(x)`: effectue un déplacement rapide sur x.

`def rapid_y(y)`: effectue un déplacement rapide sur y.

`def rapid_xy(x,y)`: effectue un déplacement rapide sur (x,y).

`def rapid(x,y,z)`: effectue un déplacement rapide sur (x,y,z).

`def rapid_z(z)`: effectue un déplacement rapide sur z.

`def linear_x(x)`: effectue un déplacement linéaire sur x.

`def linear_y(y)`: effectue un déplacement linéaire sur y.

`def linear_xy(x,y)`: effectue un déplacement linéaire sur (x,y).

`def linear(x,y,z)`: effectue un déplacement linéaire sur (x,y,z).

`def linear_z(z)`: effectue un déplacement linéaire sur z.

`def rapid_safe()`: effectue un déplacement rapide sur z à la position de sécurité.

`def rapid_a(a)`: effectue un déplacement rapide sur a.

`def rapid_b(b)`: effectue un déplacement rapide sur b.

`def rapid_c(c)`: effectue un déplacement rapide sur c.

`def rapid_axy(a,x,y)`: effectue un déplacement rapide sur (a,x,y).

`def rapid_axyz(a,x,y,z)`: effectue un déplacement rapide sur (a,x,y,z).

`def linear_a(a)`: effectue un déplacement linéaire sur a.

`def linear_b(b)`: effectue un déplacement linéaire sur b.

`def linear_c(c)`: effectue un déplacement linéaire sur c.

`def linear_axy(a,x,y)`: effectue un déplacement linéaire sur (a,x,y).

`def linear_axyz(a,x,y,z)`: effectue un déplacement linéaire sur (a,x,y,z).

`def append(s)`: ajoute la chaîne s.

`def feed(f)`: change la vitesse d'avance.

`def pause(time)`: effectue une pause.

`def cw_arc(sx, sy, cx, cy, dx, dy)`: effectue un déplacement circulaire dans le sens horaire sur (x,y,z). Point de départ (sx,sy), centre (cx,cy), arrivée (dx,dy).

`def ccw_arc(sx, sy, cx, cy, dx, dy)`: effectue un déplacement circulaire dans le sens anti-horaire. Point de départ (sx,sy), centre (cx,cy), arrivée (dx,dy).

`def comment(s)`: ajoute un commentaire.

`def start_spindle()`: démarre l'outil dans le sens horaire.

`def start_ccw_spindle()`: démarre l'outil dans le sens anti-horaire.

`def stop_spindle()`: arrête l'outil.

`def start_flood()`: démarre l'arrosage liquide.

`def start_mist()`: démarre l'arrosage brouillard.

`def stop_coolant()`: arrête l'arrosage.

`def tool_left_radius_compensation()`: compensation du rayon de l'outil à gauche.

`def tool_right_radius_compensation()`: compensation du rayon de l'outil à droite.

`def tool_cancel_radius_compensation()`: arrêt de la compensation du rayon de l'outil.

`def tool_length_compensation(tool_number, tool_height)`: compensation de la longueur de l'outil.

`def tool_cancel_length_compensation()`: arrêt de la compensation de la longueur de l'outil.

`def absolute()`: les coordonnées suivantes sont absolues.

`def relative()`: les coordonnées suivantes sont relatives.

`def rate_inverse_time()`: la vitesse d'avance est inverse du temps.

`def rate_minute()`: la vitesse d'avance est en unité par minute.

`def rate_revolution()`: la vitesse d'avance est en unité par tour.

`def inch()`: l'unité est le pouce.

`def millimeter()`: l'unité est le millimètre.

`def new_line()`: passe à la ligne.

`def drilling(x, y, z, retract, pause)`: perçage.

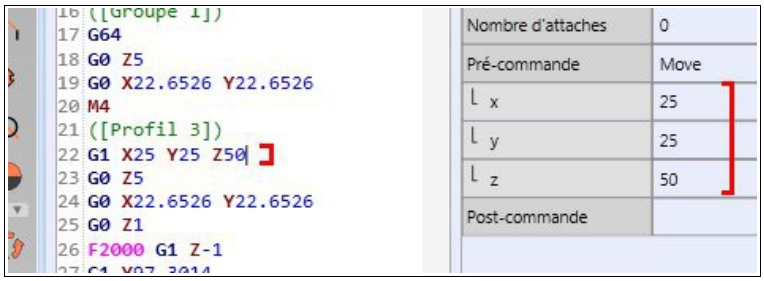
`def peck_drilling(x, y, z, retract, delta)`: perçage avec débourrage.

`def tapping(x, y, z, retract)`: taraudage.

`def boring(x, y, z, retract, pause)`: alésage.

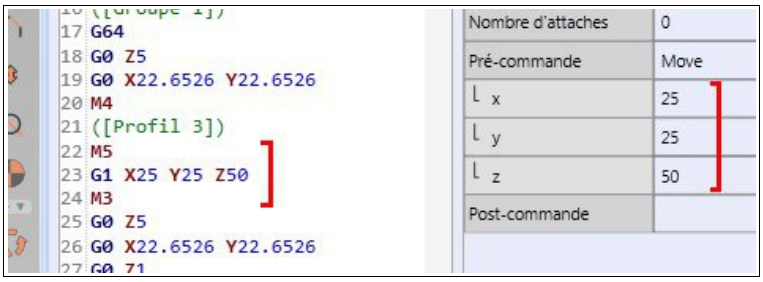
Exemple :

```
def Move(x,y,z):
    post.linear(x,y,z)
```



Dans les propriétés à droite s'affichent automatiquement les paramètres `x, y et z`. Ils seront passés à la fonction lors de la génération.

```
def Move(x,y,z):
    post.stop_spindle()
    post.linear(x,y,z)
    post.start_spindle()
```



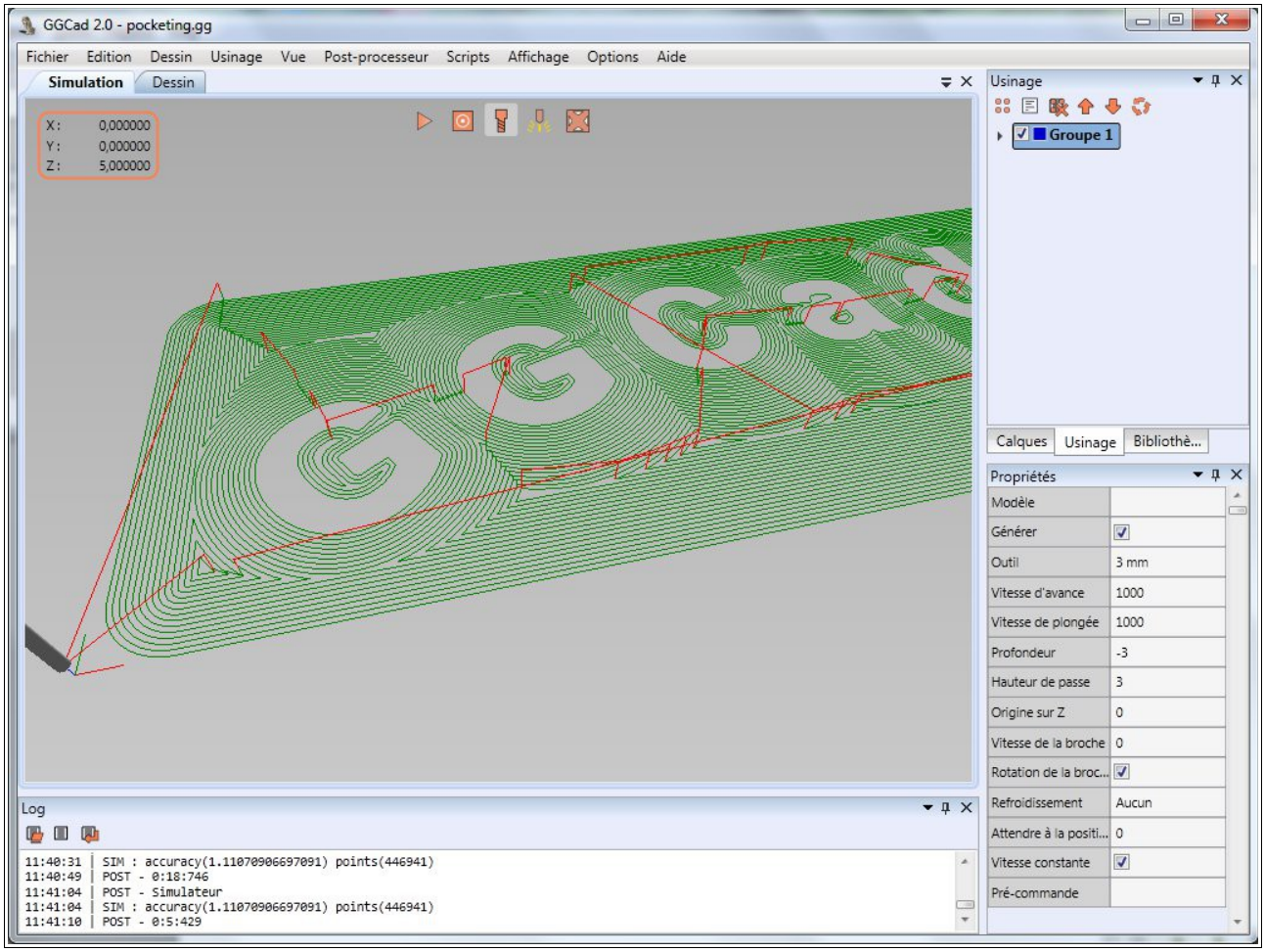
La même fonction avec les appels pour arrêter l'outil, effectuer le déplacement, puis redémarrer l'outil.

```
def RapidZ(Position):
    post.rapid_z(Position)
```

Ici, le paramètre se nomme `Position`.

2.8 Simulation

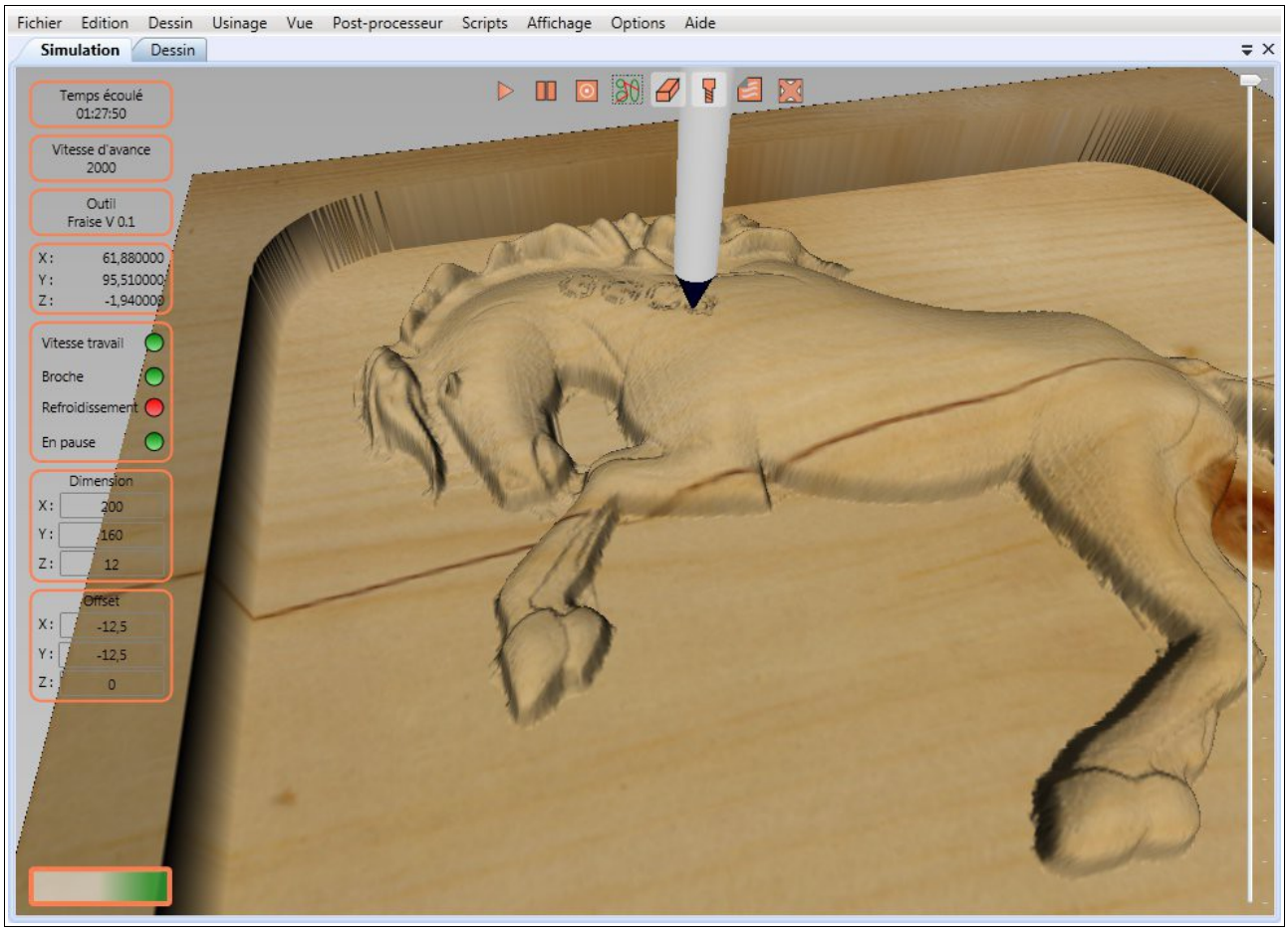
Vous pouvez lancer le simulateur par le menu `[Usinage][Simulation]`. Il dessinera alors le parcours effectué par l'outil.

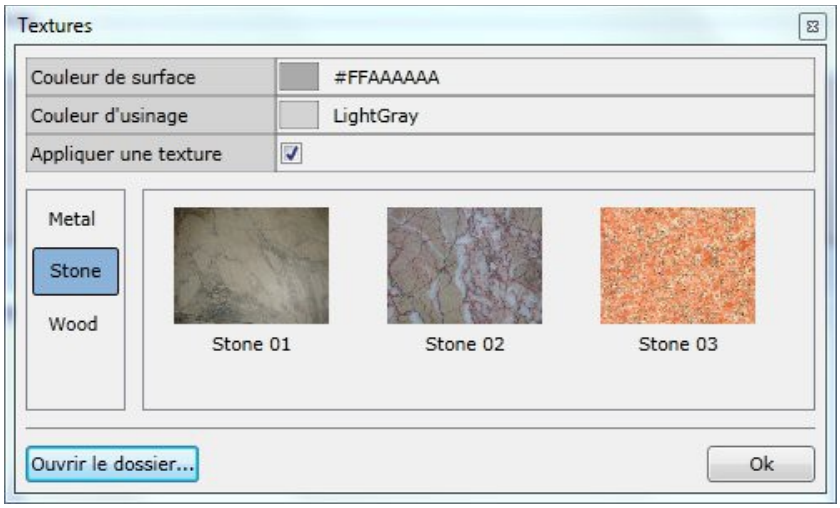


La barre d'icônes permet de lancer ou d'arrêter le simulateur.

Les déplacements se font à la souris. Bouton gauche pour la rotation, bouton droit pour la translation et la molette pour zoomer.

Professional : La version professionnelle permet de simuler l'usinage de la pièce par enlèvement de matière, augmenter/diminuer la vitesse de simulation, et donne les informations sur l'état de la commande numérique. Le matériau peut-être colorisé ou matérialisé par une texture (bois, métal, pierre). Vous pouvez ajouter vos propres textures.





2.9 Transmission port série (RS232)

La version professionnelle permet de transmettre les fichiers générés par port série. La boîte de configuration s'ouvre à travers le menu `[Post-processeur][Configurer le port série]`. La configuration doit être la même que celle du contrôleur.

- `Port série` : nom du port série utilisé.
- `Taux de transfert` : vitesse de transfert sur le port.
- `Parité` : aucune, pair, impair.
- `Bit de données` : 7, 8 ou 9.
- `Bit d'arrêt` : 1, 1.5 ou 2.
- `Contrôle` : aucun, logiciel (Xon/Xoff), matériel, les deux.
- `Timeout` : nombre de secondes maximum d'attente de la connexion.
- `Encodage` : encodage des caractères. (En général ASCII).
- `Enregistrer les données reçues` : les données renvoyées par le contrôleur sont enregistrées dans le fichier [nom du fichier de sortie.log].
- `Fin de ligne` : caractères ajoutés à la fin de chaque ligne. Les codes {cr}, {lf}, {crlf} correspondent aux caractères cr : retour chariot, lf : nouvelle ligne.

L'envoi ce fait ensuite par le menu `[Post-processeur][Envoyer par port série]`. Chaque ligne est envoyée une par une. Si le mode Xon/Xoff est sélectionné, le contrôleur indiquera quand poursuivre ou mettre en pause l'envoi.

2.10 Scripts

Les scripts sont écrits en langage Python.

Les fichiers sont stockés dans le dossier `[Mes Documents\GGSoft\GGCad2\Scripts]`. Le fichier script doit contenir une classe portant le même nom que le fichier (ex: `Coordinates.py` et `Class Coordinates:`).

Les fonctions commencent par `def`.

Initialisation

```
def __init__(self):
    pass
```

Cette fonction est obligatoire et doit être laissée en l'état.

`def initilization(self):`

Faites vos initialisation ici.

`def load_countries(self):`

Charge les chaînes de caractère.

`def set_country(self, country_name):`

Définit le nom de la langue actuelle.

`def menu(self):`

Retourne le titre du menu qui contiendra le script.

`def submenu(self):`

Retourne le sou-menu, ou la description du script.

`def use_properties(self):`

Retourne vrai si le script utilise la boîte de propriétés pour sa configuration.

`def get_properties_count(self):`
Retourne le nombre de propriétés à configurer.

`def get_property_description(self, index):`
Retourne la description de la propriété. Le paramètre `[index]` est le numéro de la propriété. La première propriété commence à l'index `'0'`.

`def get_property(self, index):`
Retourne la valeur de la propriété à `[index]`.

`def set_property(self, index, val):`
Passe la nouvelle valeur `[val]` de la propriété `[index]`.

`def hidden(self):`
Vous pouvez cacher certaines propriétés en fonction d'autres grâce à cette fonction. Elle doit renvoyer un tableau d'entier `int` contenant les indexes à cacher.

`def is_open_file(self, index):`
Renvoyer `True` si la propriété à l'index doit permettre d'ouvrir la boîte de dialogue pour choisir un fichier.

`def is_browse_directory(self, index):`
Renvoyer `True` si la propriété à l'index doit permettre d'ouvrir la boîte de dialogue pour choisir un dossier.

`def get_description(self):`
Renvoyer la description affichée au-dessus des propriétés.

`def is_configurable(self):`
Renvoyer `True` si le script peut ouvrir sa propre boîte de dialogue pour sa configuration.

`def configurate(self):`
Ouvrez ici la boîte de dialogue de configuration.

`def begin(self):`
Appelé avant la méthode run.

`def run(self):`
Execution du script.

`def end(self):`
Appelé après la méthode run.

Interface [cad] avec la CAO

GGCam expose un objet `[cad]` permettant d'effectuer les opérations avec le dessin. Ses fonctions sont les suivantes :

Souris

`def enable_mouse():`
Active la souris.

`def disable_mouse():`
Désactive la souris.

`Point MouseLeftDown :`
Coordonnée du bouton souris enfoncé.

`Point MouseLeftUp :`
Coordonnée du bouton souris relaché.

Timers

`def set_timer(index, interval, callback):`
Définit un timer. `[index]` est l'identifiant, `[intervalle]` la période en milliseconde, et callback le nom de la fonction à appeler.

`def start_timer(index):`
Démarre le timer.

`def stop_timer(index):`
Arrête le timer.

Journal

`def log(message):`
Ajoute `[message]` au journal.

Messages

`def start_wait():`
Affiche le logo d'attente.

`def do_events():`
Laisse Windows traiter ses messages.

`def do_events(milliseconds):`
Laisse Windows traiter ses messages pendant un nombre de millisecondes.

`def stop_wait():`
Ferme le logo.

Dessin

`def draw():`
Force le dessin.

`def color(a,r,g,b):`
Retourne une couleur avec les valeurs alpha, red, green, blue. Ces valeurs peuvent être au format byte (de 0 à 255) ou au format float (0.0 à 1,0).

Historique

`def start_history():`
Démarre un enregistrement de plusieurs opérations à la fois dans l'historique.

`def stop_history():`
Termine l'enregistrement.

`def enable_history():`
Active l'historique.

`def disable_history():`
Désactive l'historique.

Sélection des éléments

`def select():`
Sélectionne tous les éléments.

`def select(shapes):`
Sélectionne les éléments contenus dans `[shapes]`.

`def unselect():`
Dés-sélectionne tous.

`def unselect(shapes):`
Dés-sélectionne les éléments contenus dans `[shapes]`.

Grouper/Dégrouper

`def group():`
Crée un groupe (block) avec les éléments sélectionnés, puis retourne le block.

`def group(shapes):`
Crée un groupe (block) avec les éléments contenus dans `[shapes]`, puis retourne le block.

`def ungroup():`
Supprime les groupes sélectionnés, retourne les éléments contenu dans les groupes.

`def ungroup(shapes):`
Supprime les groupes contenus dans `[shapes]`, retourne les éléments contenu dans le groupe.

Calques

`current_layer :`
Retourne le calque actif. Il y a deux raccourcis : `clayer` et `cl`.

`def add_layer(name):`
Retourne un nouveau calque de nom `name`.

`def delete_layer(name):`
Supprime le calque de nom `name`.

`def delete_layer(layer):`
Supprime le calque `layer`.

Éléments

`selected_shapes :`
Retourne la liste des éléments sélectionnes. Il y a deux raccourcis : `sshapes` et `ss`.

`selected_shape :`
Retourne le premier élément sélectionné.

`def order_shapes(shapes):`
Retourne une liste de liste d'éléments. Ces listes sont triées de façon à obtenir les listes des éléments connectés. Par exemple, si nous avons les éléments `A,B,C` connectés et les éléments `D,E,F,G` connectés, l'appel à la méthode avec `[A,E]` va retourner `[A,B,C] [D,E,F,G]`. Vous

aurez le même résultat pour un appel avec `[B,C,F]`.

`def get_shapes(shape):`
Retourne la liste des éléments connecté à `shape`. (note : la méthode `order_shapes` fait appel à cette méthode.

`def get_points(shapes):`
Retourne la liste de `Point` représentant les coordonnées de `shapes`.

`def split(shapes):`
Éclate un éléments en sous-éléments. (ex: polyline en liste de line).

`def connect(shapes, distance):`
Connecte les éléments contenus dans `shapes` entre eux, avec une `distance` entre 2 points.

Géométrie

`def fillet(radius):`
Crée un congé de rayon radius entre les 2 éléments.

`def edge(size):`
Crée un chanfrein de dimension `size` entre les 2 éléments.

`def split_at_intersection():`
Coupe les éléments sélectionnés aux intersections et retourne les nouveaux éléments.

`def split_at_intersection(shapes):`
Coupe les éléments contenus dans `shapes` aux intersections et retourne les nouveaux éléments.

`def horizontal_mirror():`
Effectue un miroir horizontale aux éléments sélectionnés.

`def horizontal_mirror(shapes):`
Effectue un miroir horizontale aux éléments contenus dans `shapes`.

`def vertical_mirror():`
Effectue un miroir vertical aux éléments sélectionnés.

`def vertical_mirror(shapes):`
Effectue un miroir vertical aux éléments contenus dans `shapes`.

`def symmetry(p1, p2):`
Effectue une symétrie par rapport à l'axe entre les points `p1` et `p2` aux éléments sélectionnés.

`def symmetry(shapes, p1, p2):`
Effectue une symétrie par rapport à l'axe entre les points `p1` et `p2` aux éléments contenus dans `shapes`.

`def symmetry(x1, y1, x2, y2):`
Effectue une symétrie par rapport à l'axe entre les points `(x1,y1)` et `(x2,y2)` aux éléments sélectionnés.

`def symmetry(shapes, x1, y1, x2, y2):`
Effectue une symétrie par rapport à l'axe entre les points `(x1,y1)` et `(x2,y2)` aux éléments contenus dans `shapes`.

`def rotation(center, angle):`
Effectue une rotation de centre `center` et d'`angle` aux éléments sélectionnés.

`def rotation(x, y, angle):`
Effectue une rotation de centre `(x,y)` et d'`angle` aux éléments sélectionnés.

`def rotation(shapes, center, angle):`
Effectue une rotation de centre `center` et d'`angle` aux éléments contenus dans `shapes`.

`def rotation(shapes, x, y, angle):`
Effectue une rotation de centre `(x,y)` et d'`angle` aux éléments contenus dans `shapes`.

`def translation(offset):`
Effectue une translation de centre `center` aux éléments sélectionnés.

`def translation(x, y):`
Effectue une translation de centre `(x,y)` aux éléments sélectionnés.

`def translation(shapes, offset):`
Effectue une translation de centre `center` aux éléments contenus dans `shapes`.

`def translation(shapes, x, y):`
Effectue une translation de centre `(x,y)` aux éléments contenus dans `shapes`.

`def scale(width, height):`
Effectue une mise à l'échelle aux éléments sélectionnés.

`def scale(shapes, width, height):`
Effectue une mise à l'échelle aux éléments contenus dans `shapes`.

`def union(shapes) :`
Applique la transformation *Union* aux éléments contenus dans `shapes`.

`def xor(shapes) :`
Applique la transformation *XOR* aux éléments contenus dans `shapes`.

`def difference(shapes) :`
Applique la transformation *Difference* aux éléments contenus dans `shapes`.

`def intersection(shapes) :`
Applique la *transformation Intersection* aux éléments contenus dans `shapes`.

Information

`path :`
Retourne le chemin du répertoire des scripts.

Fonctions 3D (professional)

`def explode_polygon(shapes) :`
Explose les polygon3D contenus dans `shapes` en éléments uniques.

`def explode_polygon() :`
Explose les polygon3D sélectionnés en éléments uniques.

▪ Interface [Point]

La classe `[Point]` permet de créer des points et d'effectuer certaines opérations dessus.

Propriétés

`x :` contient la position x.

`y :` contient la position y.

`empty :` point vide.

`is_empty :` retourne vrai si le point est égal à `empty`.

Constructeur

`def Point():`
Retourne un nouveau `Point`.

`def Point(x,y):`
Retourne un nouveau `Point` de coordonnées `(x,y)`.

`def Point(p):`
Retourne un nouveau `Point` de coordonnées `(p.x,p.y)`.

`def Clone():`
Retourne une copie du point.

Fonctions statiques

`def add(p1, p2):`
Retourne la somme de `p1` et `p2`.

`def sub(p1, p2):`
Retourne soustraction de `p2` à `p1`.

Fonctions

`def length(p):`
Retourne la distance jusqu'à `p`.

`def length2(p):`
Retourne la distance au carré jusqu'à `p`.

`def angle(p):`
Retourne l'angle en degrés ente l'horizontale et `p`, de centre point.

`def middle(p):`
Retourne le point milieu entre point et `p`.

`def position(angle,length):`
Retourne le point à l'`angle` et la distance `length` par rapport au centre point.

`def position(p,length):`
Retourne le point à l'angle avec `p` et la distance `length` par rapport au centre point.

`def rotate(angle,center):`
Retourne le point avec une rotation d'`angle` et de centre `center`.

`def rotate(angle):`
Retourne le point avec une rotation d'`angle` et de centre `(0,0)`.

Interface [Point3]

La classe `[Point3]` permet de créer des points 3D et d'effectuer certaines opérations dessus.

Propriétés

`x :` contient la position x.

`y :` contient la position y.

`z :` contient la position z.

`empty :` point vide.

`is_empty :` retourne vrai si le point est égal à `empty`.

Constructeur

`def Point3():`
Retourne un nouveau `Point`.

`def Point3(x,y,z):`
Retourne un nouveau `Point` de coordonnées `(x,y,z)`.

`def Point3(p):`
Retourne un nouveau `Point` de coordonnées `(p.x,p.y,p.z)`.

`def Clone():`
Retourne une copie du point3.

Fonctions statiques

`def add(p1, p2):`
Retourne la somme de `p1` et `p2`.

`def sub(p1, p2):`
Retourne soustraction de `p2` à `p1`.

Fonctions

`def length(p):`
Retourne la distance jusqu'à `p`.

`def length2(p):`
Retourne la distance au carré jusqu'à `p`.

`def rotate(angle,center,axis):`
Retourne le point avec une rotation d'`angle`, de centre `center` et d'axe `axis`.

`def rotate(angle,axis):`
Retourne le point avec une rotation d'`angle` et de centre `(0,0)` et d'axe `axis`.

▪ Interface [Matrix3]

La classe `[Matrix3]` permet de créer une matrice 4x4.

Propriétés

`M11 :` 1ère ligne, 1ère colonne.

`m12 :` 1ère ligne, 2ème colonne.

`m13 :` 1ère ligne, 3ème colonne.

`m14 :` 1ère ligne, 4ème colonne.

`m21 :` 2ème ligne, 1ère colonne.

`m22 :` 2ème ligne, 2ème colonne.

`m23 :` 2ème ligne, 3ème colonne.

`m24 :` 2ème ligne, 4ème colonne.

`m31 :` 3ème ligne, 1ère colonne.

`m32 :` 3ème ligne, 2ème colonne.

`m33 :` 3ème ligne, 3ème colonne.

`m34 :` 3ème ligne, 4ème colonne.

`offset_x :` 4ème ligne, 1ère colonne.

`offset_y :` 4ème ligne, 2ème colonne.

`offset_z :` 4ème ligne, 4ème colonne.

`m44 :` 4ème ligne, 4ème colonne.

`determinant :` retourne le déterminant.

`is_identity :` retourne vrai la matrice est égale à la matrice d'identité.

`is_affine :` retourne vrai la matrice est affine.

`has_inverse :` retourne vrai si la matrice peut être inversée.

Constructeur

`def Matrix3():`
Retourne une nouvelle `Matrix3`.

Fonctions statiques

`def multiply(p1, p2):`
Retourne la matrice résultat de la multiplication de `p1` et `p2`.

Fonctions

`def invert():`
Inverse la matrice.

`def prepend(matrix3):`
Ajoute `matrix3` en début de la pile.

`def append(matrix3):`
Ajoute `matrix3` en à la fin de la pile.

`def rotate(quaternion):`
Ajoute une rotation de `quaternion` et de centre `(0,0,0)` à la matrice.

`def rotate_at(quaternion,center):`
Ajoute une rotation de `quaternion` et de centre `center` à la matrice.

`def rotate_prepend(quaternion):`
Ajoute une rotation de `quaternion` et de centre `(0,0,0)` à la matrice au début.

`def rotate_at_prepend(quaternion,center):`
Ajoute une rotation de `quaternion` et de centre `center` à la matrice au début.

`def scale(point3):`
Ajoute une mise à l'échelle de `point3` à la matrice.

`def scale_at(point3,center):`
Ajoute une mise à l'échelle de `point3` et de centre `center` à la matrice.

`def scale_prepend(point3):`
Ajoute une mise à l'échelle de `point3` et de centre `(0,0,0)` à la matrice au début.

`def scale_at_prepend(point3,center):`
Ajoute une mise à l'échelle de `point3` et de centre `center` à la matrice au début.

`def translate(point3):`

Ajoute une translation de `point3` à la matrice.

`def translate_prepend(point3):`
Ajoute une translation de `point3` à la matrice au début.

`def multiply(point3):`
Multiplie `point3` par la matrice puis retourne le résultat. `Point3` peut-être un Point3 ou un tableau de Point3

Interface [Quaternion]

La classe `[Quaternion]` permet de créer un quaternion.

Propriétés

`angle :` retourne l'angle du quaternion.

`axis :` retourne l'axe du quaternion en tant que Point3.

`x :` retourne la valeur de x.

`y :` retourne la valeur de y.

`z :` retourne la valeur de z.

`w :` retourne la valeur de w.

`is_identity :` retourne vrai le quaternion est égale au quaternion d'identité.

`is_normalized :` retourne vrai la matrice est affine.

Constructeur

`def Quaternion():`
Retourne un nouveau Quaternion.

`def Quaternion(axis,angle):`
Retourne un nouveau Quaternion.

`def Quaternion(x,y,z,w):`
Retourne un nouveau Quaternion.

Fonctions statiques

`def add(p1, p2):`
Retourne le quaternion somme de `p1` et `p2`.

`def multiply(p1, p2):`
Retourne le quaternion résultat de la multiplication de `p1` et `p2`.

`def sub(p1, p2):`
Retourne le quaternion soustraction de `p1` et `p2`.

Fonctions

`def invert():`

Remplace le quaternion par son inverse.

`def normalize():`
Normalise le quaternion.

`def conjugate():`
Remplace le quaternion par son conjugué.

▪ Interface [Layer], calque

Créez, supprimez les calques par l'interface `[cad]`.

Propriétés

`handle :` retourne le numéro unique interne.

`shapes :` retourne la liste d'éléments.

`name :` retourne le nom du calque.

`color :` retourne, affecte la couleur du calque.

`visible :` retourne, affecte la propriété visible du calque.

`locked :` retourne, affecte la propriété locked du calque.

Création d'éléments

`def line(p1, p2):`
Retourne une nouvelle ligne de coordonnées `[p1,p2]`.

`def line(x1, y1, x2, y2):`
Crée et retourne une nouvelle ligne de coordonnées `[(x1,y1),(x2,y2)]`.

`def dot(p):`
Retourne un nouveau point de coordonnées `[p1]`.

`def dot(x, y):`
Retourne un nouveau point de coordonnées `(x,y)`.

`def circle(center, radius):`
Retourne un nouveau cercle de centre `center` et de rayon `radius`.

`def circle(x, y, radius):`
Retourne un nouveau cercle de centre `(x,y)` et de rayon `radius`.

`def arc(start, center, end, cw):`
Retourne un nouvel arc de centre `center` partant de `start`, arrivant à `end` et dans le sens horaire si `cw` est vrai.

`def arc(sx, sy, cx, cy, dx, dy, radius, cw):`
Retourne un nouvel arc de centre `(cx,cy)` partant de `(sx,sy)`, arrivant à `(dx,dy)` et dans le

sens horaire si `cw` est vrai.

`def ellipse(center, minor, major, angle):`
Retourne une nouvelle ellipse de centre `center`, d'axe majeur `major`, d'axe mineur `minor` et d'`angle`.

`def ellipse(x, y, minor, major, angle):`
Retourne une nouvelle ellipse de centre `(x,y)`, d'axe majeur `major`, d'axe mineur `minor` et d'`angle`.

`def polygon(points[]):`
Retourne un nouveau polygone avec `points` comme coordonnées.

`def spline(points[]):`
Retourne une courbe avec `points` comme coordonnées.

`def text(p, text, font_family, font_size):`
Retourne un nouveau texte à la coordonnée `p`, avec la chaîne `text`, le nom de police `font_family` et la taille `font_size`.

`def text(p, text, font_family, font_size, bold, italic, angle):`
Retourne un nouveau texte à la coordonnée `p`, avec la chaîne `text`, le nom de police `font_family`, la taille `font_size`, gras si `bold` est vrai, italic si `italic` est vrai et d'`angle`.

`def text(x, y, text, font_family, font_size, bold, italic, angle):`
Retourne un nouveau texte à la coordonnée `(x,y)`, avec la chaîne `text`, le nom de police `font_family`, la taille `font_size`, gras si `bold` est vrai, italic si `italic` est vrai et d'`angle`.

`def outline(p, text, font_family, font_size):`
Retourne un nouveau contour de texte à la coordonnée `p`, avec la chaîne `text`, le nom de police `font_family` et la taille `font_size`.

`def outline(p, text, font_family, font_size, bold, italic, angle):`
Retourne un nouveau contour de texte à la coordonnée `p`, avec la chaîne `text`, le nom de police `font_family`, la taille `font_size`, gras si `bold` est vrai, italic si `italic` est vrai et d'`angle`.

`def outline(x, y, text, font_family, font_size, bold, italic, angle):`
Retourne un nouveau contour de texte à la coordonnée `(x,y)`, avec la chaîne `text`, le nom de police `font_family`, la taille `font_size`, gras si `bold` est vrai, italic si `italic` est vrai et d'`angle`.

`def polygon3d():`
Retourne un nouveau **`polygon3d`**.

`def polygon3d(points):`
Retourne un nouveau **`polygon3d`** avec un polygone **`points`**.

`def face3d():`

Retourne un nouveau `face3d`.

`def face3d(points):`

Retourne un nouveau `face3d` avec les triangles `points`. `points` est un multiple de 3.

Éléments

Les éléments possèdent des propriétés et foncions communes :

Propriétés

`handle :` numéro unique interne.

`name :` nom de l'élément.

`type :` type d'élément.

`is_selected :` retourne, affecte la propriété [sélectionnée].

`layer :` retourne le calque qui possède l'élément.

`tag :` retourne/affecte un objet (à disposition).

Fonctions

`def get_points():`
Retourne la liste des points de l'élément.

`def get_point(index):`
Retourne le point à `index`.

`def first():`
Retourne le premier point.

`def last():`
Retourne le dernier point.

`def rotate(angle):`
Effectue une rotation à l'élément de `angle` degrés, de centre `(0,0)`.

`def rotate(center, angle):`
Effectue une rotation à l'élément de `angle` degrés, de centre `center`.

`def translate(diff):`
Effectue une translation à l'élément de `diff.x` et `diff.y`.

`def translate(x, y):`
Effectue une translation à l'élément de `x` et `y`.

Interface [Block], groupe

Classe représentant un groupe d'éléments.

Propriétés

`center :` retourne, affecte le centre du groupe.

Interface [Line], ligne

Propriétés

`p1 :` retourne, affecte le point de départ.

`p2 :` retourne, affecte le point d'arrivée.

`length :` retourne la longueur du segment.

`angle :` retourne l'angle par rapport à l'horizontale.

Interface [Dot], point

Propriétés

`point :` retourne, affecte le point de coordonnées.

Interface [Circle], cercle

Propriétés

`center :` retourne, affecte le centre.

`radius :` retourne, affecte le rayon.

`diameter:` retourne, affecte le diamètre.

Interface [Arc]

Propriétés

`center :` retourne, affecte le centre.

`begin :` retourne, affecte le point de départ.

`end :` retourne, affecte le point d'arrivée.

`angle :` retourne l'angle de l'arc.

`radius :` retourne, affecte le rayon.

`cw :` retourne, affecte la direction de l'arc, vrai si sens horaire, faux si sens anti-horaire.

Interface [Ellipse]

Propriétés

`center :` retourne, affecte le centre.

`minor :` retourne, affecte l'axe mineur.

`major :` retourne, affecte l'axe majeur.

`angle :` retourne l'angle de l'ellipse.

Interface [Polygon], polygone

Propriétés

`count :` retourne le nombre de points.

`is_closed :` retourne, affecte la propriété `[fermée]`.

Fonctions

`def set_point(index, p):`
Affecte le point `p` à l'`index`.

`def set_point(index, x, y):`
Affecte le point `(x,y)` à l'`index`.

`def add_points(points):`
Ajoute les `points` à la fin.

`def add_point(point):`
Ajoute le `point` à la fin.

`def add_point(x,y):`
Ajoute le point `(x,y)` à la fin.

`def insert_points(index, points):`
Insert les `points` à partir de `index`.

`def insert_point(index, point):`
Insert le `point` à `index`.

`def insert_point(index, x, y):`
Insert le point `(x,y)` à `index`.

`def delete_point(index):`
Supprime le point à `index`.

Interface [Spline], courbe

Propriétés

`count :` retourne le nombre de points.

`is_closed :` retourne, affecte la propriété `[fermée]`.

`spline_type :` retourne, affecte la propriété `[type]`, le type de courbe.

```
Bezier, Bspline, NaturalCubic
```

Fonctions

`def set_point(index, p):`
Affecte le point `p` à l'`index`.

`def set_point(index, x, y):`
Affecte le point `(x,y)` à l'`index`.

`def add_points(points):`
Ajoute les `points` à la fin.

`def add_point(point):`
Ajoute le `point` à la fin.

`def add_point(x,y):`
Ajoute le point `(x,y)` à la fin.

`def insert_points(index, points):`
Insert les `points` à partir de `index`.

`def insert_point(index, point):`
Insert le `point` à `index`.

`def insert_point(index, x, y):`
Insert le point `(x,y)` à `index`.

`def delete_point(index):`
Supprime le point à index.

▪ Interface [Outline], contour de texte

Propriétés

`point :` retourne, affecte le point de référence.

`text :` retourne, affecte la propriété `[Texte]`.

`family :` retourne, affecte la propriété `[Police]`.

`size :` retourne, affecte la propriété `[Taille]`, la taille du texte.

`angle :` retourne, affecte la propriété `[Angle]`.

`bold :` retourne, affecte la propriété `[Gras]`.

`italic :` retourne, affecte la propriété `[Italic]`.

▪ Interface [Text], texte

Propriétés

`point :` retourne, affecte le point de référence.

`text :` retourne, affecte la propriété `[Texte]`.

`family :` retourne, affecte la propriété `[Police]`.

`size :` retourne, affecte la propriété `[Taille]`, la taille du texte.

`angle :` retourne, affecte la propriété `[Angle]`.

`bold :` retourne, affecte la propriété `[Gras]`.

`italic :` retourne, affecte la propriété `[Italic]`.

- ## Exemple

Création d'un rectangle

```
    def run(self):
        cad.start_history()    # démarre une action groupée pour l'historique
        cad.unselect()         # dé-sélectionne les éléments
        p1 = Point(self.x, self.y)         # calcule les coordonnées
        p2 = Point(self.x+self.width, self.y)
        p3 = Point(self.x+self.width, self.y+self.height)
        p4 = Point(self.x, self.y+self.height)
        l1 = cad.current_layer.line(p1, p2)                  # Crée la ligne l1
        l2 = cad.current_layer.line(p2, p3)                  # Crée la ligne l2
        l3 = cad.current_layer.line(p3, p4)                  # Crée la ligne l3
        l4 = cad.current_layer.line(p4, p1)                  # Crée la ligne l4
        cad.select([l1, l2, l3, l4])       # sélection les lignes créées
        cad.stop_history()     # termine une action groupée pour l'historique
        cad.log('Rectangle done...') # écrit dans le journal
```

- ## Interface [Face3D]

(Professional)

Propriétés

`count :` retourne le nombre de triangles.

Fonctions

`def get_triangle(index):`
Retourne le triangle à l'`index` sous la forme d'un tableau de 3 Point3.

`def add_triangles(points):`
Ajoute les triangles. `points` est un tableau de Point3.
Il doit être un multiple de 3 (3 coordonnées par triangle).

`def multiply(matrix):`
Multiplication matricielle.

Exemple

Création d'un cube :

```
    def run(self):
        # we have to create 2 triangles per face
        points = []
        # bottom face
        points.append(Point3(x,y,z));
        points.append(Point3(x+size_x,y,z));
        points.append(Point3(x+size_x,y+size_y,z));
        points.append(Point3(x+size_x,y+size_y,z));
        points.append(Point3(x,y+size_y,z));
        points.append(Point3(x,y,z));
        # up face
        points.append(Point3(x,y,z+size_z));
        points.append(Point3(x+size_x,y,z+size_z));
        points.append(Point3(x+size_x,y+size_y,z+size_z));
        points.append(Point3(x+size_x,y+size_y,z+size_z));
        points.append(Point3(x,y+size_y,z+size_z));
        points.append(Point3(x,y,z+size_z));
        # left face
        points.append(Point3(x,y+size_y,z));
        points.append(Point3(x,y,z));
        points.append(Point3(x,y,z+size_z));
        points.append(Point3(x,y,z+size_z));
        points.append(Point3(x,y+size_y,z+size_z));
        points.append(Point3(x,y+size_y,z));
        # right face
        points.append(Point3(x+size_x,y+size_y,z));
        points.append(Point3(x+size_x,y,z));
        points.append(Point3(x+size_x,y,z+size_z));
        points.append(Point3(x+size_x,y,z+size_z));
        points.append(Point3(x+size_x,y+size_y,z+size_z));
        points.append(Point3(x+size_x,y+size_y,z));
        # front face
        points.append(Point3(x,y,z));
        points.append(Point3(x+size_x,y,z));
        points.append(Point3(x+size_x,y,z+size_z));
        points.append(Point3(x+size_x,y,z+size_z));
        points.append(Point3(x,y,z+size_z));
        points.append(Point3(x,y,z));
        # back face
        points.append(Point3(x,y+size_y,z));
        points.append(Point3(x+size_x,y+size_y,z));
        points.append(Point3(x+size_x,y+size_y,z+size_z));
        points.append(Point3(x+size_x,y+size_y,z+size_z));
        points.append(Point3(x,y+size_y,z+size_z));
        points.append(Point3(x,y+size_y,z));


        cube = cad.cl.face3d(points)
```

Interface [Polygon3D]

(Professional)

Propriétés

`count :` retourne le nombre de polygones.

Fonctions

`def get_polygon(index):`
Retourn le polygon à l'`index` sous la forme d'un tableau de Point3.

`def add_polygon(points):`
Ajoute le polygone. `points` est un tableau de Point3.

`def delete_polygon(index):`
Supprime le polygone à l'`index`.

`def multiply(matrix):`
Multiplication matricielle.

Exemple

Création d'une spirale 3D comme un ressort :

```
def run(self):
        l = []
        angle = 0
        inc = disc / (360.0/increment);
        z = cz
        h = cz - height
        while z > h:
            l.append( coordinates(radius, math.radians(angle), cx, cy, z) )
            angle = angle + increment
            z = z - inc
            if angle >= 360:
                angle = 0

        s = cad.cl.polygon3d()
        s.add_polygon(l)
```

3 Raccourcis clavier

CTRL + N	nouveau
CTRL + O	ouvrir
CTRL + S	enregistrer
CTRL + C	copier
CTRL + X	couper
CTRL + V	coller
CTRL + Z	annuler
CTRL + Y	refaire
CTRL + I	Réinitialisation de la camera du simulateur
SUPPR/DEL	supprimer
F11	plein écran
F5	générer
F6	simuler
H	miroir horizontal
V	miroir vertical
R	rotation
G	Grouper les éléments
U	Dégrouper les éléments
F1	Ouvrir le manuel