II.1. Introduction

Nous avons vu dans le chapitre précédent des généralités sur le domaine de la CFAO. Maintenant, nous allons vous présenter dans ce chapitre notre démarche pour extraire les données relatives à la trajectoire de l'outil à partir d'un fichier G-code et de la génération de cette trajectoire ainsi que les outils qui nous ont permis de réaliser notre travail.

II.2. Objectif de notre travail

L'objectif de notre travail est de développer une application à partir de la technologie OpenCASCADE (référence bibliographique), qui est une technologie libre (Open Source) proposant toutes les fonctionnalités d'un modeleur volumique de type CAO, qui nous permet d'extraire les données géométriques de la trajectoire de l'outil à partir d'un fichier G-code et de les superposer avec le modèle 3D de la pièce correspondante à partir de son fichier au format IGES.

Notre travail permettra à l'ingénieur du bureau de méthode d'avoir un outil d'aide à la conception capable de visualiser la pièce et les trajectoires incluses dans le fichier G-code de sa fabrication par une MOCN, afin qu'il puisse déterminer visuellement les problèmes de cohérence des trajectoires et de collision éventuelles.

II.3. Méthodologie de travail

Afin d'atteindre notre objectif cité dans le paragraphe précédent, nous avons fait le choix de nous baser sur le logiciel de modélisation « FreeCAD » qui est développé à partir de la technologie OpenCASCADE, et qui permet plus facilement l'exploitation des fonctionnalités avancées de la modélisation grâce aux nombreuses fonctionnalités qui possède.

Etant donné que le logiciel « FreeCAD » est développé avec le langage de programmation « Python », nous avons été obligé d'apprendre ce langage afin d'intégrer notre programme dans le logiciel « FreeCAD » et pouvoir bénéficier de toutes ses ressources.

Dans notre démarche on considère que l'utilisateur a déjà ouvert le logiciel de modélisation et qu'il a une pièce active sauvegardée au format standard IGES et du fichier G-code permettant d'usiner cette même pièce.

A partir de là, notre application une fois exécutée va :

- récupérer le nom du fichier ouvert ;
- localiser les fichiers aux formats IGES et G-code de la pièce ;
- ouvrir le fichier G-code et le charger dans la mémoire pour le traitement ;
- lire chaque ligne du contenu textuel du fichier G-code et extraire les données géométriques des trajectoires (type de trajectoire plus coordonnées) ;
- **sauvegarder le résultat dans un fichier texte pour tracer les trajectoires de l'outil ;**
- **ouvrir le fichier résultant du traitement précédent ;**
- lire chaque ligne et tracer la trajectoire indiquée par celle-ci ;
- visualiser la pièce avec les trajectoires du fichier G-code.

Les étapes 5 et 6 en gras ont étaient utilisé pour développer la version sur python indépendamment du logiciel FreeCAD pour démontrer la possibilité de l'extraction des données du fichier G-code. La version finale de notre programme n'a pas besoin de sauvegarder ces données, mais nous les utilisons directement pour tracer les trajectoires en toute transparence pour l'utilisateur.

Pour aider le lecteur à comprendre la suite de notre travail, nous allons vous présenter le logiciel de modélisation « FreeCAD » ainsi que le langage de programmation « Python ». Et pour finir, nous allons détailler l'utilisation du langage « Python » pour dessiner des trajectoires dans le logiciel « FreeCAD » à travers quelques exemples.

II.4. Présentation du logiciel de modélisation « FreeCAD »

II.4.1. Définition

FreeCAD est un modeleur 3D d'application générale à base de technologies OpenCASCADE. FreeCAD est orienté vers le génie mécanique et la conception de produits, mais vise également d'autres disciplines, telles que l'architecture ou d'autres champs d'activité d'ingénierie.

FreeCAD propose des fonctionnalités similaires à Catia, SolidWorks ou Solid Edge et s'inscrit donc aussi dans les catégories CFAO/IAO/PLM. Il vise la conception paramétrique, avec une architecture modulaire qui permet l'ajout facile de fonctionnalités supplémentaires sans modifier le cœur du programme.

Comme plusieurs logiciels de CAO modernes, il offrira une composante 2D pour extraire des vues de dessin du modèle 3D et produire des mises en plan. Mais le dessin 2D direct n'est pas l'objectif visé.

Un autre objectif majeur de FreeCAD est l'usage intensif des importantes bibliothèques open source qui existent dans le domaine du calcul numérique, notamment : OpenCASCADE, un puissant noyau géométrique, ainsi que Python, un des meilleurs langages de script actuels. FreeCAD peut être lui-même utilisé comme bibliothèque par d'autres programmes.

FreeCAD est aussi multiplateforme, et fonctionne présentement parfaitement sous Windows, Linux/Unix et Mac OSX, avec la même apparence et les mêmes fonctionnalités sous toutes les plateformes.

II.4.2. OpenCASCADE Technology (OCCT)

II.4.2.1. Historique

Son développement est intimement lié au logiciel de CAO Euclid que commercialise Matra Datavision (MDTV) au début des années 1980, suivi de Euclid-IS en 1987 (le premier système de CAO/CFAO). Euclid était à cette époque le concurrent direct de CATIA. En 1993, MDTV sort Euclid 3 et développe parallèlement CAS.CADE (*Computer Aided Software for Computer Aided Design and Engineering*), ce dernier étant à l'origine une boîte à outils logiciels utilisée pour les technologies objets.

En 1996, sort Euclid Quantum dont l'infrastructure repose entièrement sur CAS.CADE. Fin 1998, le portefeuille de logiciels de MDTV (Euclid Quantum, Styler, Machinist et Strim) est racheté par Dassault Systèmes. MDTV, qui comptait alors 700 employés, se mue alors en société de service spécialisée dans le développement de logiciels. S'ensuit la publication sous licence *Open Source* en 1999 de CAS.CADE qui prend à l'occasion le nom d'Open CASCADE. Ce changement brutal de culture d'entreprise va précipiter le départ de certains ingénieurs de Matra Datavision. En tant que filiale d'EADS, le 7 décembre 2000, la société Open CASCADE SAS est créée afin de gérer la suite du projet en *Open Source*, mais aussi pour développer des activités commerciales (création d'applications spécifiques, formation et conseil) autour de cette infrastructure.

En 2003, la société Principia R&D, spécialisée dans les solveurs éléments finis et l'ingénierie scientifique, acquiert Open CASCADE SAS auprès d'EADS.

Le logiciel change de nom et devient Open CASCADE Technology (OCCT). Par la suite, la société est reprise par Euriware (filiale d'AREVA) le 3 février 2006.

II.4.2.2. Définition

Open CASCADE est un framework logiciel *Open Source* pour la CAO/CFAO, l'IAO et la modélisation 3D. Il est composé d'une vaste bibliothèque objets utilisables en C++ ou depuis Tcl-Tk et d'outils annexes pour le développement d'applications spécifiques. Il est disponible sous la licence Open CASCADE Technology Public License (OCCTPL).

II.4.3. Dessiner sur FreeCAD

II.4.3.1. Atelier Sketcher

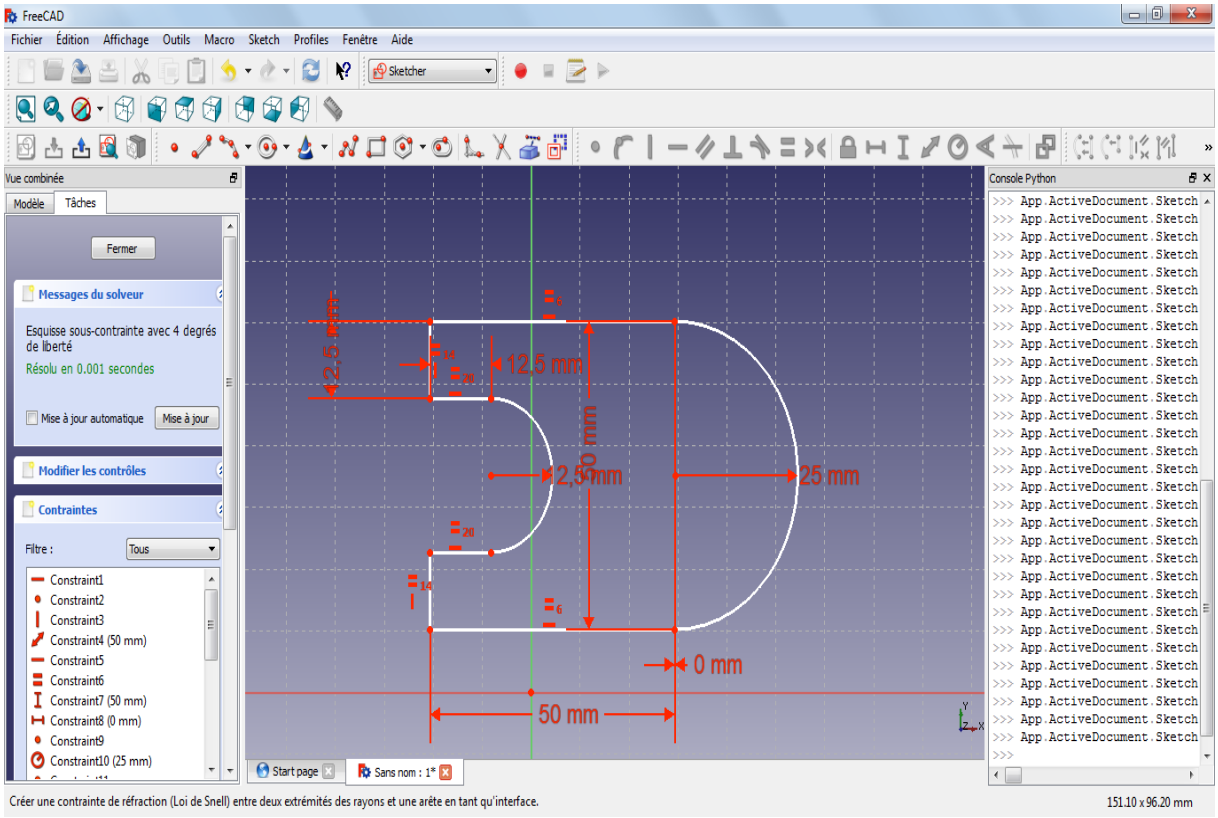


Figure II.1. Exemple de dessin avec l'atelier Sketcher

L'**atelier Sketcher** permet de créer des géométries 2D nommées **esquisses** (voir figure II.1), qui seront principalement utilisées par l'**Atelier PartDesign**, mais potentiellement par d'autres ateliers tels que l'atelier Architectural.

En général, l'esquisse est le point de départ de la plupart des modèles de CAO - une esquisse simple peut être 'extrudée' en une forme 3D, une autre esquisse peut être créée pour creuser une cavité à la surface de cette forme, ou encore générer une extrusion. Avec les opérations booléennes, l'atelier Sketcher est au cœur de la conception 3D solide.

L'atelier Sketcher met à l'avant les **contraintes**, qui permettent de définir des formes 2D selon des critères géométriques précis. Un solveur mathématique calcule le niveau de contrainte de l'esquisse et permet l'exploration interactive des degrés de liberté.

II.4.3.2. Processus d'esquisse

Une esquisse est toujours en deux dimensions (2D). Pour créer un solide, on crée une esquisse 2D d'une aire simple fermée et on lui applique une extrusion ou une révolution pour lui ajouter la troisième dimension.

Si l'esquisse possède des segments qui se croisent, ou un point non directement positionné sur un segment, ou encore des écarts entre des points terminaux ou des segments adjacents, l'extrusion ou la pièce de révolution ne créera pas un solide.

II.4.3.3. Les outils

Les outils **Atelier Esquisse** sont tous situés dans le menu **Sketch** qui s'affiche lorsque vous chargez votre plan de travail.

a) Géométries d'esquisse

Ces outils permettent de créer des objets, parmi les on compte :

-  Point : dessine un point.
-  Ligne : dessine une ligne entre 2 points.
-  Arc : dessine un segment d'arc à partir du centre, rayon, angle de départ et angle d'arrivée.
-  Arc par 3 points : dessine un arc de cercle sur deux points et un troisième point pour la circonférence.
-  Cercle : dessine un cercle à partir de son centre et du rayon.

-  Cercle par 3 points : dessine un cercle à partir de trois points.
-  Polyligne : dessine une ligne composée de plusieurs segments connectés entre eux.
-  Rectangle : dessine un rectangle à partir de 2 points opposés.
-  Triangle : dessine un triangle équilatéral inscrit dans un cercle.
-  Carré : dessine un carré inscrit dans un cercle.
-  Congé : crée un congé entre deux lignes connectées en un point. Sélectionnez les deux lignes, ou cliquez sur le sommet commun, puis activez l'outil.

b) Contraintes d'esquisse

Les contraintes sont utilisées pour définir des règles entre les éléments d'esquisse et pour verrouiller l'esquisse le long des axes vertical et horizontal.

-  Coïncident : crée une contrainte coïncidente (point sur point) entre deux sommets sélectionnés.
-  Point sur objet : crée une contrainte point-sur-objet sur les éléments sélectionnés. L'un des éléments doit être un sommet, l'autre une ligne, un arc ou un cercle.
-  Vertical : crée une contrainte de verticalité sur les lignes ou segments de polylignes sélectionnés. Plus d'un élément peut être sélectionné.
- Horizontal : crée une contrainte d'horizontalité sur les lignes ou segments de polylignes sélectionnés. Plus d'un élément peut être sélectionné.
-  Parallèle : crée une contrainte de parallélisme entre deux lignes sélectionnées.
-  Perpendiculaire : crée une contrainte de perpendicularité entre deux lignes sélectionnées.

-  Tangente : crée une contrainte de tangence entre deux éléments sélectionnés, ou de colinéarité entre deux lignes.
-  Égalité : crée une contrainte d'égalité entre au moins deux éléments sélectionnés. Contraindra la longueur pour des lignes et le rayon pour des cercles et des arcs.
-  Symétrie : crée une contrainte symétrique entre deux points par rapport à une ligne.
-  Fixe : crée une contrainte fixe sur le sommet sélectionné en ajoutant des dimensions horizontale et verticale relatives à l'origine (les dimensions peuvent être éditées par la suite).

II.4.3.3. Python script dans FreeCAD

FreeCAD a été programmé dès la première ligne de code dans le but d'être totalement contrôlé par des scripts écrits en Python. Presque toutes les procédures de FreeCAD, telles que l'interface, le contenu des scènes, même la représentation du contenu des vues 3D, sont accessibles à partir de l'interpréteur Python ou de vos propres scripts. Par conséquence, FreeCAD est probablement l'une des applications d'ingénierie la plus profondément personnalisable et évolutive disponible actuellement.

L'une des utilisations du python les plus répandues est comme langage de script, car il est facilement à intégrable dans d'autres applications. C'est exactement de cette manière qu'il est utilisé dans FreeCAD.

A partir de la console python, ou à partir d'un propre script, nous pouvons programmer FreeCAD, et lui faire exécuter des commandes très complexes pour lesquelles il n'y a pas encore d'outils disponibles dans l'interface graphique.

Par exemple, à partir d'un script python, vous pouvez:

- créer de nouveaux objets ;
- modifier les objets existants ;
- modifier la représentation 3D de ces objets ;
- modifier l'interface de FreeCAD.

II.5. Langages de programmation « Python »

II.5.1. Introduction

Un programme informatique est une succession d'instructions exécutable par l'ordinateur. Toutefois, l'ordinateur ne sait manipuler que du binaire, c'est-à-dire une succession de 0 et de 1. Il est donc nécessaire d'utiliser un langage de programmation pour écrire de façon lisible, c'est-à-dire avec des instructions compréhensibles par l'humain car proches de son langage, les instructions à exécuter par l'ordinateur.

Les langages peuvent être plus ou moins verbeux, plus ou moins expressifs. Certains langages sont très répandus, ou au contraire ne sont utilisés que par une poignée de personnes.

Les langages peuvent être jeunes ou vieillissant, et, lorsque le côté affectif s'en mêle, ils peuvent être agréables à utiliser ou au contraire agaçants... La tâche qui consiste à choisir un langage de programmation pour illustrer un cours d'algorithmique et d'informatique, à destination d'un public hétérogène est donc... difficile.

Dans la suite, c'est le langage **Python** qui sera utilisé. Il est : généraliste, assez répandu, multi-paradigmes, assez jeune, expressif et agréable à utiliser. Il n'est cependant pas le seul à posséder toutes ces qualités.

II.5.2. Définition

Python est un langage de programmation généraliste, facile à apprendre et rapide à mettre en œuvre. Grâce à sa syntaxe claire, cohérente et concise, Python permet aux développeurs de produire du code de qualité, lisible et maintenable. Écrire du code Python est un exercice agréable, même en respectant les conventions de codage.

II.5.3. Utilisation

Python est un langage qui peut s'utiliser dans de nombreux contextes et s'adapter à tout type d'utilisation grâce à des bibliothèques spécialisées. Il est cependant particulièrement utilisé comme langage de script pour automatiser des tâches simples mais fastidieuses,

Comme un script qui récupérerait la météo sur Internet ou qui s'intégrerait dans un logiciel de conception assistée par ordinateur afin d'automatiser certains enchaînements d'actions répétitives.

On l'utilise également comme langage de développement de prototype lorsqu'on a besoin d'une application fonctionnelle avant de l'optimiser avec un langage de plus bas niveau. Il est particulièrement répandu dans le monde scientifique, et possède de nombreuses extensions destinées au calcul numérique.

II.5.4. Caractéristiques du langage

- **Portable**

Python fonctionne sous différentes variantes d'Unix, Windows, Mac OS, BeOs, NextStep, Et par le biais de différentes implémentations.

- **Facile à intégrer**

Un programme écrit en Python s'intègre très facilement avec d'autres composants logiciels. Il est possible par exemple d'utiliser directement des bibliothèques externes ou encore d'intégrer du code C ou C++.

- **Hautement productif**

La conception d'applications en Python est très rapide car certains aspects de programmation sont gérés automatiquement, comme la gestion des ressources mémoire et le typage des données.

- **Dynamique**

Python est un langage dynamique : dans la plupart des implémentations, le code source n'est pas compilé contrairement à des langages comme C ou Pascal, mais exécuté à la volée.

II.5.5. Exemple de programme : Editer un fichier par Python

Dans la plupart des travaux de programmation, on doit lire ou écrire dans un fichier. Pour éditer un fichier en python on utilise la fonction « **open** ».

- **Lecture de fichier**

En programmation, la lecture d'un fichier se fait en trois temps :

- l'ouverture du fichier,
- la lecture proprement dite,
- la fermeture du fichier.

- **Exemple de code du programme :**

```
1. f = open("test.txt", "r")      # ouverture
2. texte = f.read()               # récupération du texte dans une variable
3. print(texte)                   # affichage du texte récupéré
4. f.close()                      # fermeture
```

- **Ouverture d'un fichier**

L'ouverture d'un fichier se réalise grâce à la fonction open qui a deux paramètres :

- Le premier paramètre est une chaîne de caractères contenant le nom (avec suffixe éventuel) du fichier.
- Le deuxième paramètre est aussi une chaîne de caractères contenant le mode de traitement du fichier. La valeur de ce paramètre est "r" pour la lecture d'un fichier.

La fonction **open** retourne un descripteur de fichiers qui est un objet particulier.

- **Les types d'ouverture**

Il existe plusieurs modes d'ouverture:

r, pour une ouverture en lecture (READ)

w, pour une ouverture en écriture (WRITE), à chaque ouverture le contenue du fichier est écrasé. Si le fichier n'existe pas python le crée.

a, pour une ouverture en mode ajout à la fin du fichier(APEND). Si le fichier n'existe pas python le crée.

- **Fermeture**

Comme tout élément ouvert, il faut le refermer une fois les instructions terminées. pour cela là on utilise la méthode **close()** .

- **Lecture**

L'intégralité d'un fichier peut être récupérée dans une unique chaîne de caractères grâce à la méthode **read()**.

```
1. f = open ( "test.txt" , "r" )
2. texte = f . read ( )
3. texte
4. print ( texte  )
```

➢ **Ecrire dans un fichier**

Syntaxe pour écrire dans un fichier:

```
1. f  = open ( "test.txt" , "a" )
2.  f .write( "vitesse de coupe=" )
3. f.close()
```

II.6. Création des objets sur FreeCAD avec langage python

II.6.1. Importer les modules nécessaires

Nous avons d'abord besoin d'importer le module **Part** afin que nous puissions utiliser son contenu Python.

Nous allons également importer le module Base à l'intérieur du module de FreeCAD:

```
import Part
from FreeCAD import Base
```

II.6.2. Création des formes simples

Vous pouvez créer facilement des formes topologiques avec "**make...()**" qui est une méthode du "**Module Part**":

```
b = Part.makeBox(100,100,100)
Part.show(b)
```

La combinaison de **make...()** avec d'autres méthodes sont disponibles:

- **makeBox(l,w,h)**: Construit un cube et pointe sur **p** dans la direction **d** et de dimensions (**longueur,largeur,hauteur**)
- **makeCircle(radius)**: Construit un cercle de rayon (**r**).
- **make Cone(radius1,radius2,height) :** Construit un cône de (rayon1,rayon2,hauteur).
- **makeCylinder(radius,height) :** Construit un cylindre de (**rayon,hauteur**).
- **makeLine((x1,y1,z1),(x2,y2,z2)) :** Construit une ligne aux coordonnées (**x1,y1,z1**),(**x2,y2,z2**) dans l'espace 3D.
- **makePlane(length,width) :** Construit un rectangle de (**longueur,largeur**).
- **makePolygon(list) :** Construit un polygone (**liste de points**).
- **makeSphere(radius) :** Construit une sphère de (**rayon**).

- **makeTorus(radius1,radius2)**: Construit un tore de (**rayon1,rayon2**).

a) Création d'un Vecteur

Les Vecteurs sont l'une des informations les plus importantes lors de la construction des formes géométriques.

Ils contiennent habituellement 3 nombres (mais pas toujours) les coordonnées cartésiennes **x, y et z**.

On peut créez un vecteur comme ceci:

```
Vecteur = Base.Vector(3,2,0)
```

Nous venons de créer un vecteur de coordonnées **x = 3, y = 2, z = 0**.

Dans le module Part, les vecteurs sont utilisés partout.

Le module Part utilise aussi une autre façon de représenter un point, appelé Vertex, qui n'est actuellement rien d'autre qu'un conteneur pour un vecteur.

b) Création d'une arête (edge)

Une arête (bord) n'est rien d'autre qu'une ligne avec deux Vertex (sommets):

```
edge = Part.makeLine((0,0,0), (10,0,0))
edge.Vertexes
> [<Vertex object at 01877430>, <Vertex object at 014888E0>]
```

PS: On peut aussi créer une arête en donnant deux Vecteurs:

```
vec1 = Base.Vector(0,0,0)
vec2 = Base.Vector(10,0,0)
line = Part.Line(vec1,vec2)
```

c) Mise en forme à l'écran

L'écran FreeCAD n'affiche uniquement que les vues 3D que vous lui demandez d'afficher. Pour cela, nous utilisons une méthode simple**:**

```
Part.show(edge)
```

Un Objet 3D sera affiché dans notre document FreeCAD, et notre dessin sera affiché sous forme filaire.

Utilisez cette commande chaque fois que vous voudrez afficher votre forme géométrique à l'écran.

d) Création d'un contour (Wire)

Un contour est une ligne multi-arêtes, et peut être créé dans une liste d'arêtes ou même une liste de lignes (fils):

```
edge1 = Part.makeLine((0,0,0), (10,0,0))
edge2 = Part.makeLine((10,0,0), (10,10,0))
wire1 = Part.Wire([edge1,edge2])
edge3 = Part.makeLine((10,10,0), (0,10,0))
edge4 = Part.makeLine((0,10,0), (0,0,0))
wire2 = Part.Wire([edge3,edge4])
wire3 = Part.Wire([wire1,wire2])
Part.show(wire3)
```

II.6.3. Exemple : Création d'une topologie simple

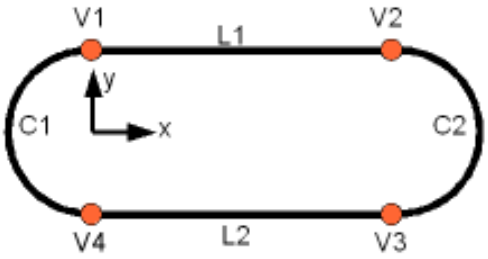


Figure II.2. Forme à réalisé

Nous allons créer une topologie avec une géométrie toute simple. Nous devrons veiller à ce que les sommets des pièces géométriques soient à la même position, quatre sommets, deux cercles et deux lignes (voir figure II.2).

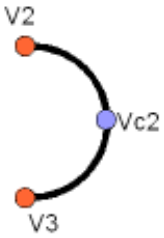
II.6.3.1. Création de la géométrie

Nous devons d'abord créer les parties distinctes géométriques en filaire. Nous devons veiller à ce que tous les sommets des pièces géométriques qui vont être raccordées soient à la même position.

Donc, nous créons d'abord les points:

```
from FreeCAD import Base
V1 = Base.Vector(0,10,0)
V2 = Base.Vector(30,10,0)
V3 = Base.Vector(30,-10,0)
V4 = Base.Vector(0,-10,0)
```

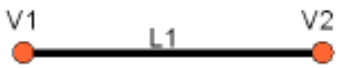
- ***Arc***



Pour créer un arc de cercle, nous créons un point de repère puis nous créons l'arc de cercle passant par trois points:

```
VC1 = Base.Vector(-10,0,0)
C1 = Part.Arc(V1,VC1,V4)
VC2 = Base.Vector(40,0,0)
C2 = Part.Arc(V2,VC2,V3)
```

- **Ligne**



La ligne peut être créée très simplement en dehors des points :

```
L1 = Part.Line(V1,V2)
L2 = Part.Line(V4,V3)
```

➢ **Tout relier**

La dernière étape consiste à relier les éléments géométriquement ensemble, et façonner une forme topologique:

```
S1 = Part.Shape([C1,C2,L1,L2])
```

➢ **Construire un prisme**

Maintenant nous allons extruder notre forme filaire dans une direction, et créer une forme en 3 Dimensions:

```
W = Part.Wire(S1.Edges)
S = Part.Face(W)
P = S.extrude(Base.Vector(0,0,10))
```

➢ **Affichons le tout**

La pièce sera visible comme dans la figure II.3 :

```
Part.show(P)
```

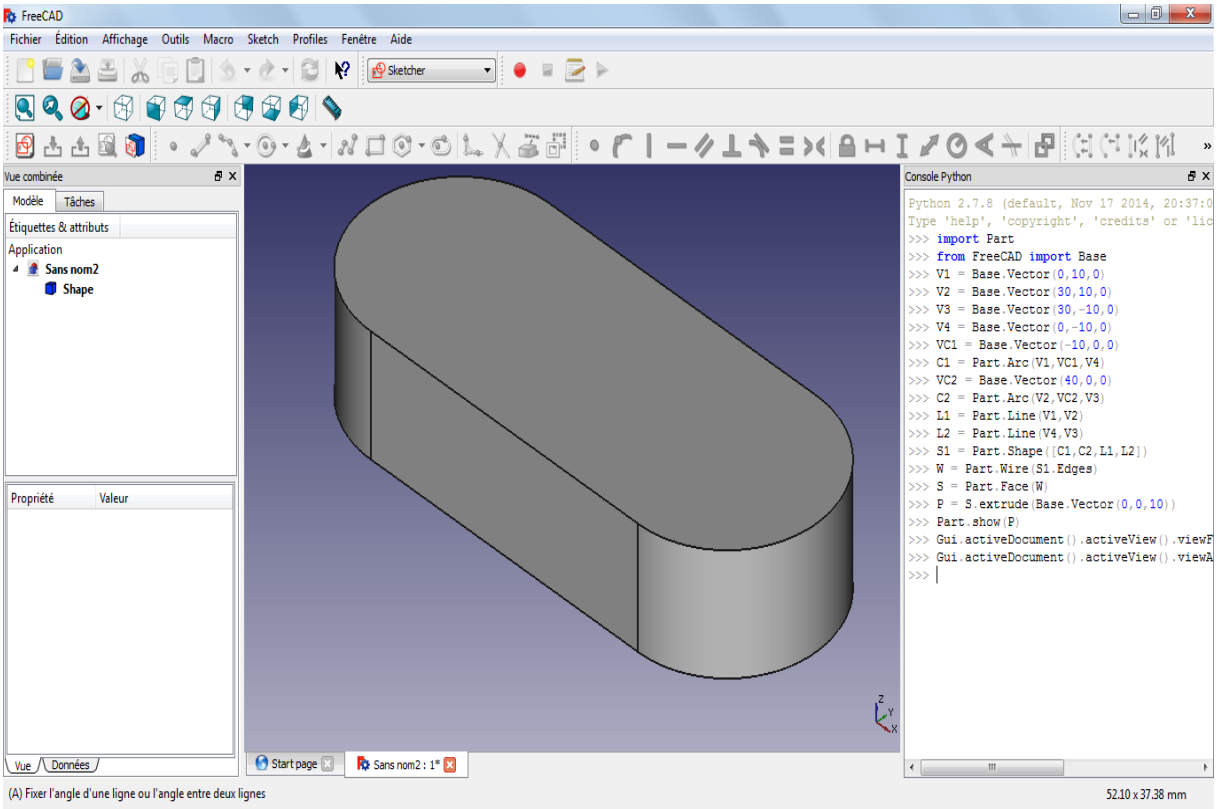


Figure II.2. Pièce affichée