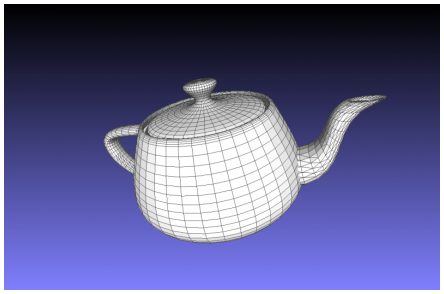
mesh 2 svg 2 paper

Meshlab : https://www.meshlab.net/ Rien tiré de meshlab pour transformer un mesh (stl, obj) en svg

Premier essai concluant avec https://www.svgai.org/convert/stl-to-svg, le fichier s'ouvre bien avec inkscape, l'épaisseur des traits est bien trop élevée mais ça s'arrange facilement. Aucune face n'est cachée

Conseil de Laurent : utiliser «ln» de Michael Fogleman : https://github.com/fogleman/ln C'est programmé en Go, jamais utilisé

Pour la suite j'utilise l'objet teapot.obj extrait du newell_teaset.zip



Conversion de formats 3D en ligne de commande

Avec OpenCTM (https://sourceforge.net/projects/openctm/)

```
sudo apt install openctm-tools
```

Ensuite on peut utiliser **ctmconv** qui permet de convertir les formats suivants :

- OpenCTM (.ctm),
- Stanford triangle format (.ply),
- Stereolitography (.stl),
- 3D Studio (.3ds),
- COLLADA 1.4/1.5 (.dae),
- Wavefront geometry file (.obj),
- LightWave object (.lwo),
- Geomview object file format (.off),
- VRML 2.0 - export only (.wrl).

Exemple :

```
ctmconv parasect.obj parasect.stl
```

Infos sur un objet 3D en ligne de commande

Nombre de points, de faces, etc.

Avec **assimp-utils**

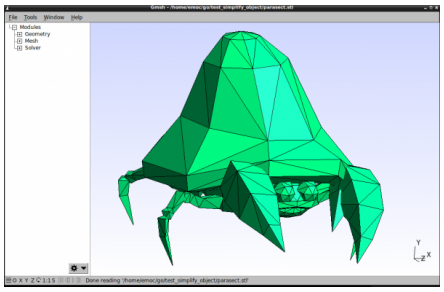
```
sudo apt install assimp-utils
assimp info teapot.obj
```

Assimp pour Open Asset Import Library

- https://github.com/assimp/assimp
- https://the-asset-importer-lib-documentation.readthedocs.io/en/latest/

Affichage d'objets STL

Avec GMSH : https://gmsh.info/ qui est aussi capable d'une multitude d'autres choses (en GUI ou CLI)



Installation de Go

```
# ************************************     # installation du langage Go sur Debian 12  @ tenko
sudo apt update
sudo apt install golang
go version                                 # go version go1.19.8 linux/amd64
go env GOPATH                              # ok : /home/emoc/go
```

Helloworld en Go

Créer un fichier vide helloworld.go

```
nano helloworld.go
```

Le fichier helloworld.go contient

```
package main

import "fmt"

func main() {
    fmt.Println("HelloWorld, Golang!")
}
```

Puis

```
go run hello.go
```

Comment compiler ce programme pour qu'il puisse être utilisé comme une commande ?

Il faut le transformer en module

```
go mod init example/helloworld      # donner un nom et chemin au module
go mod tidy                         # récupérer les dépendances
go build -o helloworld              # créer le binaire «helloworld»
mv ./helloworld ../bin/helloworld
```

Maintenant on peut déclencher la commande avec

```
~/go/bin/helloworld
```

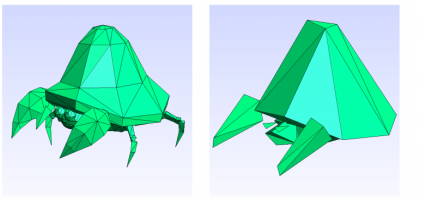
Utilisation de Simplify

Simplify est un logiciel en ligne de commande de Michael Fogleman qui permet de réduire le nombre de faces d'un objet 3D **au format .STL**. Simplify est programmé en Go

https://github.com/fogleman/simplify

```
# installer Go (voir ci-dessus)
mkdir ~/go/bin
go install github.com/fogleman/simplify/cmd/simplify@latest
# réduction à 10% des faces de l'objet (652 faces -> 64 faces)
~/go/bin/simplify -f 0.1 parasect.stl parasect-0.1.stl
```

Comparaison (objet original : parasect)



Utilisation de ln

Pour transformer un objet 3D au format .OBJ en fichier .SVG

```
git clone https://github.com/fogleman/ln.git
cd ln
go mod init ln/ln
go mod tidy
```

placer le fichier teapot.obj dans le dossier et créer le fichier teapot.go :

```
package main

import "github.com/fogleman/ln/ln"

func main() {
    scene := ln.Scene{}
    mesh, err := ln.LoadOBJ("teapot.obj")
    if err != nil {
        panic(err)
    }
    mesh.UnitCube()
    scene.Add(ln.NewTransformedShape(mesh, ln.Rotate(ln.Vector{0, 1, 0}, 0.5)))
    // scene.Add(mesh)
    eye := ln.Vector{-0.5, 0.5, 2}
    center := ln.Vector{}
    up := ln.Vector{0, 1, 0}
    width := 1024.0
    height := 1024.0
    paths := scene.Render(eye, center, up, width, height, 35, 0.1, 100, 0.01)
    paths.WriteToPNG("teapot.png", width, height)
    paths.WriteToSVG("teapot.svg", width, height)
}
```

Puis

```
go run teapot.go
```

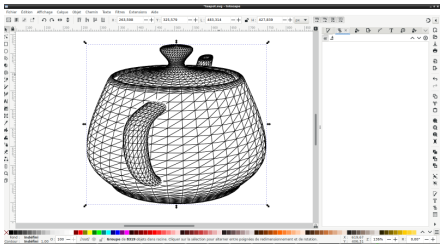
Ça marche! Le fichier svg est créé, en fonction du point de vue défini dans le script go, les faces qui doivent l'être sont cachées.

Transformer en exécutable.

La commande est lancée depuis le répertoire courant dans lequel se trouve le fichier teapot.obj, les fichiers résultants (teapot.png et teapot.svg) sont créés dans le répertoire courant.

```
go build -o teapot            # construire le binaire
mv teapot ../bin/teapot       # déplacer dans le dossier ~/go/bin
~/go/bin/teapot               # lancer la commande depuis le répertoire courant
```

On obtient



Extrait du fichier svg

```
<svg width="1024.000000" height="1024.000000" version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
<g transform="translate(0,1024.000000) scale(1,-1)">
<polyline stroke="black" fill="none" points="628.113702,626.372774 630.057369,626.470582" />
<polyline stroke="black" fill="none" points="630.057369,626.470582 612.007059,629.402582" />
```

```
<polyline stroke="black" fill="none" points="646.867425,619.146177 645.594080,623.083557" />
<polyline stroke="black" fill="none" points="645.594080,623.083557 641.262088,622.941587" />
<polyline stroke="black" fill="none" points="639.714178,622.890858 645.594080,623.083557" />
<polyline stroke="black" fill="none" points="645.594080,623.083557 630.057369,626.470582" />
<polyline stroke="black" fill="none" points="646.867425,619.146177 659.738739,615.250381" />
<polyline stroke="black" fill="none" points="659.738739,615.250381 658.331336,619.276179" />
... etc.
```

En manipulant, on dirait bien que les tracés sont doublés

obj2svg

Je cherche à créer une commande qui soit accessible de n'importe où qui permette de transformer un objet 3D au format .OBJ en image png et fichier SVG du maillage

Créer le dossier et le fichier

```
mkdir test_obj2svg
cd test_obj2svg
touch obj2svg.go   # puis l'éditer
```

obj2svg.go (cliquer pour afficher le code)

obj2svg.go

```
<code go>
package main

import (
    "fmt"
    "flag"

    "github.com/fogleman/ln/ln"
)

func main() {

    // Parsing des arguments

    flag.Parse()
    args := flag.Args()
    if len(args) != 1 {
        fmt.Println("Usage: obj2svg input.obj -> créera 2 fichiers input.obj.png et input.obj.svg")
        return
    }

    pngfilename := args[0] + ".png"
    svgfilename := args[0] + ".svg"

    fmt.Printf("pngfilename %s\n", pngfilename)
    fmt.Printf("svgfilename %s\n", svgfilename)

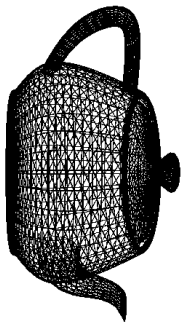
    scene := ln.Scene{}
    fmt.Printf("Loading %s\n", args[0])
    mesh, err := ln.LoadOBJ(args[0])
    if err != nil {
        panic(err)
    }
    mesh.UnitCube()
    scene.Add(ln.NewTransformedShape(mesh, ln.Rotate(ln.Vector{0, 1, 0}, 0.5)))
    // scene.Add(mesh)
    eye := ln.Vector{-0.5, 0.5, 2}
    center := ln.Vector{}
    up := ln.Vector{0, 1, 0}
    width := 1024.0
    height := 1024.0
    paths := scene.Render(eye, center, up, width, height, 35, 0.1, 100, 0.01)
    paths.WriteToPNG(pngfilename, width, height)
    paths.WriteToSVG(svgfilename, width, height)
}
```

Puis

```
go mod init example/obj2svg               # initialiser le module
go mod tidy                               # charger les dépendances
go run obj2svg.go teapot.obj              # ok, tout fonctionne
go build -o obj2svg                       # construire l'exécutable
mv obj2svg ../bin/obj2svg                 # le placer dans le bon dossier
# Maintenant on peut exécuter la commande suivante dans n'importe quel dossier
~/go/bin/obj2svg teapot.obj
```

TODO : permettre la rotation de la vue

rendu wireframe avec blender CLI + gif



Script python blender à utiliser en ligne de commande avec

```
blender --background --python blender_teapot_wireframe_views.py
```

blender_teapot_wireframe_views.py (cliquer pour afficher le code)

blender_teapot_wireframe_views.py

```python
# Blender 3.4.1
# Debian 12 @ tenko
# 20251109, résidence polygones @ Fablab des portes logiques

import bpy
import math

# --------------------------------
# Rendu wireframe "propre" 600x600
# --------------------------------

# Supprimer tous les objets existants
bpy.ops.wm.read_factory_settings(use_empty=True)

# Importer le STL
bpy.ops.import_mesh.stl(filepath="teapot.stl")
obj = bpy.context.selected_objects[0]

# Supprimer tous les matériaux existants
obj.data.materials.clear()

# Ajouter un modifier wireframe
mod = obj.modifiers.new(name="WireframeMod", type='WIREFRAME')
mod.thickness = 0.02  # épaisseur des lignes

# Créer un matériau noir shadeless pour le wireframe
mat = bpy.data.materials.new(name="WireMat")
mat.diffuse_color = (0, 0, 0, 1)
mat.use_nodes = True
bsdf = mat.node_tree.nodes.get("Principled BSDF")
bsdf.inputs['Base Color'].default_value = (0, 0, 0, 1)
bsdf.inputs['Specular'].default_value = 0
bsdf.inputs['Roughness'].default_value = 1
obj.data.materials.append(mat)

# Ajouter une caméra
cam_data = bpy.data.cameras.new(name="Camera")
cam_object = bpy.data.objects.new("Camera", cam_data)
bpy.context.collection.objects.link(cam_object)
bpy.context.scene.camera = cam_object

# Paramètres de rendu
scene = bpy.context.scene
scene.render.image_settings.file_format = 'PNG'
scene.render.resolution_x = 600
scene.render.resolution_y = 600
scene.render.film_transparent = False  # fond blanc
# scene.render.film_transparent_glass = False

# Désactiver l'anti-aliasing
# scene.render.use_antialiasing = False
scene.render.engine = 'BLENDER_EEVEE'  # moteur Eevee plus simple
# Eevee anti-aliasing quasi désactivé
```

```
scene.eevee.taa_render_samples = 1

# Récupérer la scène
scene = bpy.context.scene

# Créer un monde si nécessaire
if scene.world is None:
    world = bpy.data.worlds.new("World")
    scene.world = world

# Couleur de fond blanc
scene.world.use_nodes = True
bg = scene.world.node_tree.nodes['Background']
bg.inputs['Color'].default_value = (1, 1, 1, 1)  # blanc

# Centrer la caméra autour de l'objet
center = obj.location

# Paramètres rotation
n_views = 30
radius = 10   # distance caméra
elevation = 5

for i in range(n_views):
    angle = 2 * math.pi * i / n_views
    cam_object.location.x = center.x + radius * math.cos(angle)
    cam_object.location.y = center.y + radius * math.sin(angle)
    cam_object.location.z = center.z + elevation

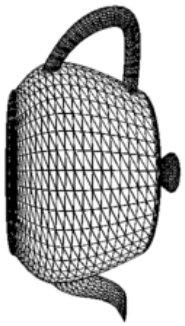
    # Orienter la caméra vers le centre
    direction = center - cam_object.location
    rot_quat = direction.to_track_quat('-Z', 'Y')
    cam_object.rotation_euler = rot_quat.to_euler()

    # Nom du fichier
    scene.render.filepath = f"teapot_wire_{i:02d}.png"

    # Rendu
    bpy.ops.render.render(write_still=True)
```

Ensuite on peut assembler les images avec

```
convert teapot_wire_*.png -threshold 50% -colors 2 -resize 600x600 teapot_wire.gif
```



Version alternative qui affiche également les faces (et masque les faces cachées)

```
blender --background --python blender_teapot_facewire.py                                    # calculer les rendus d'image
convert teapot_facewire_*.png -threshold 50% -colors 2 -resize 300x300 teapot_facewire.gif  # préparer l'animation
```

blender_teapot_facewire.py (cliquer pour afficher le code)

blender_teapot_facewire.py

```
# Blender 3.4.1
# Debian 12 @ tenko
# 20251109, résidence polygones @ Fablab des portes logiques

# En ligne 65 on peut choisir : fond transparent ou fond monochrome (changement de couleur en ligne 77)

import bpy
import math

# ------------------------------
# Configuration de la scène
```

```
# --------------------------------

# Supprimer tous les objets existants
bpy.ops.wm.read_factory_settings(use_empty=True)

# Importer le STL
bpy.ops.import_mesh.stl(filepath="teapot.stl")
obj = bpy.context.selected_objects[0]

# Supprimer tous les matériaux existants
obj.data.materials.clear()

# --------------------------------
# Matériau blanc pour les faces
# --------------------------------
mat = bpy.data.materials.new("FaceWhite")
mat.use_nodes = True
bsdf = mat.node_tree.nodes["Principled BSDF"]
bsdf.inputs['Base Color'].default_value = (1, 1, 1, 1)  # blanc
bsdf.inputs['Specular'].default_value = 0
obj.data.materials.append(mat)

# --------------------------------
# Matériau Wireframe noir
# --------------------------------
# Ajouter un modifier wireframe
mod = obj.modifiers.new(name="WireframeMod", type='WIREFRAME')
mod.thickness = 0.02
mod.use_replace = False  # conserve faces originales

# Création d'un second matériau pour le wireframe
wire_mat = bpy.data.materials.new("WireBlack")
wire_mat.use_nodes = True
nodes = wire_mat.node_tree.nodes
bsdf_wire = nodes.get("Principled BSDF")
bsdf_wire.inputs['Base Color'].default_value = (0, 0, 0, 1)  # noir
bsdf_wire.inputs['Specular'].default_value = 0
obj.data.materials.append(wire_mat)

# Associer le modifier wireframe au matériau noir
mod.material_offset = 1  # utilise le second matériau

# --------------------------------
# Caméra
# --------------------------------
cam_data = bpy.data.cameras.new(name="Camera")
cam_object = bpy.data.objects.new("Camera", cam_data)
bpy.context.collection.objects.link(cam_object)
bpy.context.scene.camera = cam_object

# Paramètres de rendu
scene = bpy.context.scene
scene.render.image_settings.file_format = 'PNG'
scene.render.resolution_x = 600
scene.render.resolution_y = 600
scene.render.film_transparent = True  # False : fond blanc, True : fond transparent
scene.render.engine = 'BLENDER_EEVEE'
scene.eevee.taa_render_samples = 1  # anti-aliasing minimal

# Fond blanc
if scene.world is None:
    world = bpy.data.worlds.new("World")
    scene.world = world
scene.world.use_nodes = True
bg = scene.world.node_tree.nodes['Background']
bg.inputs['Color'].default_value = (1, 1, 1, 1)  # blanc

# --------------------------------
# Paramètres rotation
# --------------------------------
center = obj.location
n_views = 30
radius = 10
elevation = 5

# --------------------------------
# Générer les images
# --------------------------------
for i in range(n_views):
    angle = 2 * math.pi * i / n_views
    cam_object.location.x = center.x + radius * math.cos(angle)
    cam_object.location.y = center.y + radius * math.sin(angle)
    cam_object.location.z = center.z + elevation

    # Orienter la caméra vers le centre de l'objet
    direction = center - cam_object.location
    rot_quat = direction.to_track_quat('-Z', 'Y')
    cam_object.rotation_euler = rot_quat.to_euler()

    # Nom du fichier
    scene.render.filepath = f"teapot_facewire_{i:02d}.png"

    # Rendu
```

```
bpy.ops.render.render(write_still=True)
```

Blender Export Paper Model

Un add-on pour Blender permet de «déplier» un objet 3D : Export Paper Model

Utiliser vpype

Sur Linux Debian 12, en suivant les indications de https://vpype.readthedocs.io/en/latest/install.html#linux

```
sudo apt-get install pipx
pipx ensurepath
pipx install "vpype[all]"
vpype --version             # vpype 1.15.0
vpype random show           # ooooooooooooooooooh !
```

J'ajoute **deduplicate** un plugin vpype pour enlever les lignes en doublon dans un fichier svg
https://github.com/LoicGoulefert/deduplicate

```
pipx inject vpype deduplicate
```

Ainsi que

Autres trucs intéressants à essayer

removeduplicatelines : une extension inkscape qui enlève les segments dupliqués :
https://cutlings.datafil.no/inkscape-extension-removeduplicatelines/

occult plugin vpype pour masquer les faces cachées d'un fichier svg https://github.com/LoicGoulefert/occult

vpype «vpype is an extensible CLI pipeline utility which aims to be the Swiss Army knife for creating, modifying and/or optimizing plotter-ready vector graphics» https://vpype.readthedocs.io/en/latest/install.html#linux

Article extrait de : https://lesporteslogiques.net/wiki/ - **WIKI Les Portes Logiques**
Adresse :
https://lesporteslogiques.net/wiki/recherche/residence_polygones/mesh2svg2paper?rev=1762873086
Article mis à jour: **2025/11/11 15:58**