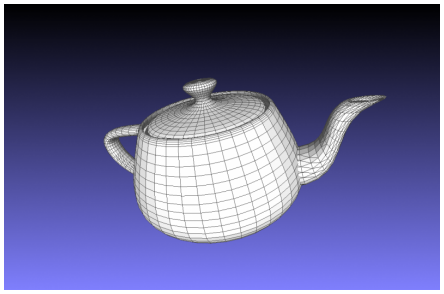
mesh 2 svg 2 paper

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).

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
```

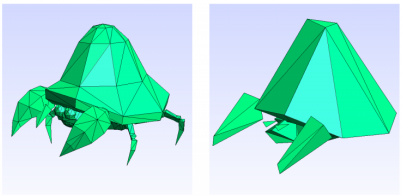
Utilisation de Simplify

Simplify est un logiciel en ligne de commande de Michael Fogleman qui permet de réduire le nombre de faces d'un objet 3D. 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
```

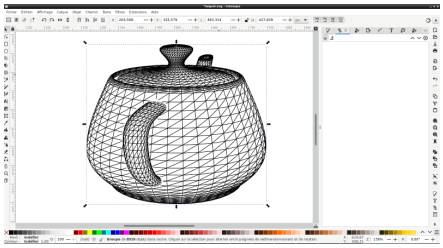
Ça marche! Le fichier svg est créé, en fonction du point de vue défini dans le script go, les faces qui doivent l'être sont cachées.

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

```
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)
}
```

cd ../test_obj2svg/ 2057 ls 2058 go mod init example/obj2svg 2059 go mod tidy 2060 go run obj2svg.go teapot.obj 2061 go build -o obj2svg 2062 mv obj2svg ../bin/obj2svg 2063 cd .. 2064 mkdir retest 2065 cd retest 2066 ~/go/bin/obj2svg teapot.obj 2067 history tail -20 2068 history |tail -20

Autres trucs intéressants à essayer

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

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

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=1762699479
Article mis à jour: **2025/11/09 15:44**