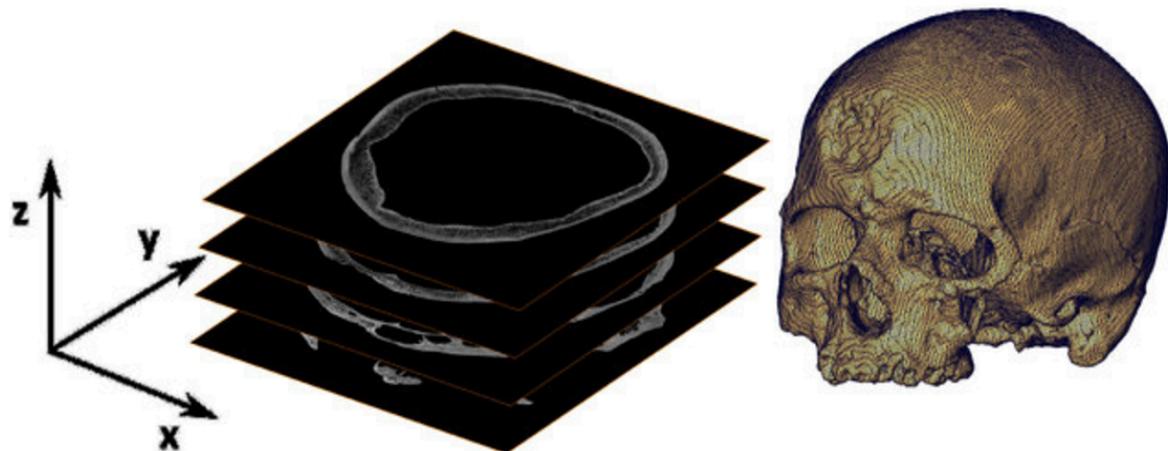




+ΣΙΓΗ +ΟΙΟΛΩΟΤ Ι ΩΗΗΗΟ Σ +ΜΩΟΣΕΙΛΣΙ Λ  
+ΟΩΩΩΗΣΙ Χ ΟΘΟΕ

*Projet intégré :*

## **MODÉLISATION 3D BASÉE SUR LES COUPES 2D D'UN CT-SCAN**



*Réalisé par : Aïssam HAMIDA  
Année universitaire : 2024-2025*

## **Résumé:**

Ce projet explore les méthodes de reconstruction volumétrique en 3D à partir de coupes 2D issues d'un CT-Scan, en mettant un accent particulier sur l'algorithme de Marching Cubes. Une étude comparative initiale des différentes techniques de reconstruction, incluant les approches Cuberille, Marching Cubes, Dual Contouring, Surface Nets et Volume Rendering, a permis de comprendre leurs principes fondamentaux, avantages et limitations. Après cette analyse, l'algorithme de Marching Cubes a été choisi comme méthode principale en raison de sa capacité à générer des modèles de surface 3D précis et optimisés. La phase suivante du projet a consisté à implémenter le Marching Cubes pour construire un modèle 3D à partir d'une fonction mathématique définissant les isosurfaces. Cette implémentation a ensuite été appliquée à des données réelles, à savoir des images 2D issues de CT-Scans, pour générer un modèle tridimensionnel du volume étudié. Les résultats obtenus démontrent la capacité de cette méthode à produire des visualisations 3D fidèles et exploitables pour des applications médicales, tout en offrant une base solide pour une analyse future ou une segmentation automatisée.

**Mots clés :** CT-Scan, reconstruction volumétrique en 3D, l'algorithme de Marching Cubes

## **Abstract:**

This project explores 3D volumetric reconstruction methods from 2D slices obtained from a CT-Scan, with a particular focus on the Marching Cubes algorithm. An initial comparative study of different reconstruction techniques, including the Cuberille, Marching Cubes, Dual Contouring, Surface Nets, and Volume Rendering approaches, provided insights into their fundamental principles, advantages, and limitations. Following this analysis, the Marching Cubes algorithm was selected as the primary method due to its ability to generate precise and optimized 3D surface models. The next phase of the project involved implementing the Marching Cubes algorithm to construct a 3D model based on a mathematical function defining isosurfaces. This implementation was then applied to real data, specifically 2D images obtained from CT-Scans, to generate a three-dimensional model of the studied volume. The results demonstrated the capability of this method to produce accurate and exploitable 3D visualizations for medical applications while providing a robust foundation for future analysis or automated segmentation.

**Keywords:** CT-Scan, 3D volumetric reconstruction, Marching Cubes algorithm

<b>1.Introduction:</b>	<b>4</b>
<b>2.Les images en 3D :</b>	<b>4</b>
<b>3.Algorithmes de modélisation 3D :</b>	<b>4</b>
<b>4.Comparaison entre les différents algorithmes de construction d'image :</b>	<b>5</b>
<b>Marching cubes</b>	<b>6</b>
<b>1.Introduction :</b>	<b>6</b>
<b>2.Les données d'entrée :</b>	<b>6</b>
<b>2.1.Isovalue (thershold) :</b>	<b>7</b>
<b>2.2.Surface de résultante :</b>	<b>7</b>
<b>3.Organigramme du Marching cubes :</b>	<b>8</b>
<b>3.1Création des sommets :</b>	<b>8</b>
<b>3.2Génération des triangles :</b>	<b>9</b>
<b>3.3Lien avec le concept des pixels :</b>	<b>9</b>
<b>4.Application pratique</b>	<b>10</b>
<b>5.Socle technique du projet :</b>	<b>10</b>
<b>6.Modélisation Du Flux :Création D'un Masque</b>	<b>11</b>
<b>7.Organigramme de la conversion 3D</b>	<b>11</b>
<b>8.Modélisation 3D :</b>	<b>12</b>
<b>8.1.Verts:</b>	<b>13</b>
<b>8.2.Faces:</b>	<b>13</b>
<b>8.3.La Fonction Meshgrid :</b>	<b>14</b>
<b>9.Visualisation de l'objet :</b>	<b>15</b>
<b>RECONSTRUCTION 3D DES IMAGES 2D DU CT-SCAN</b>	<b>16</b>
<b>1.Le flux de travail d'acquisition d'images :</b>	<b>16</b>
<b>2.Implémentation en Python :</b>	<b>17</b>
<b>2.1.Importation des bibliothèques nécessaires :</b>	<b>17</b>
<b>2.3.Chargement de la base de données</b>	<b>17</b>
<b>2.4.Pourquoi la Conversion en niveaux de gris ?</b>	<b>17</b>
<b>2.5.Vérification des dimensions des images :</b>	<b>17</b>
<b>2.6.Empiler une séquence d'images 2D :</b>	<b>17</b>
<b>2.7.Normalisation des donnes :</b>	<b>18</b>
<b>2.8.Extraction de la surface 3D (marching cubes) :</b>	<b>18</b>
<b>2.9.Création du fichier STL(stéréolithographie):</b>	<b>18</b>
<b>3.Visualisation de l'objet :</b>	<b>18</b>
<b>3.1Ajustement des Paramètres de Spacing</b>	<b>19</b>
<b>4.Amélioration du modèle :</b>	<b>20</b>

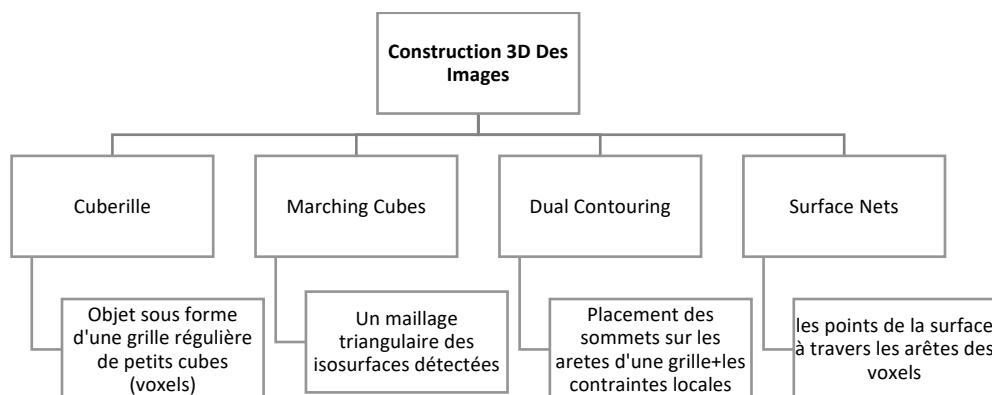
## **1.Introduction:**

La modélisation 3D en imagerie médicale regroupe un ensemble de techniques et d'outils utilisés pour le prétraitement des données issues des examens médicaux. Contrairement à l'imagerie en deux dimensions, qui offre une vision linéaire limitée, l'imagerie en 3D ajoute une dimension de profondeur, permettant une représentation plus riche dans les trois dimensions (hauteur, largeur et profondeur : x, y, z). Cette profondeur facilite une meilleure visualisation des détails anatomiques grâce à des technologies telles que le scanner (CT-scan), l'IRM (MRI) et les ultrasons. Bien que les méthodes d'imagerie en 2D, comme les radiographies ou les ultrasons, soient efficaces, elles manquent de la précision et de la crédibilité que les images en 3D peuvent offrir.

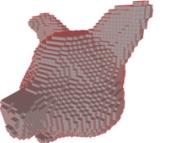
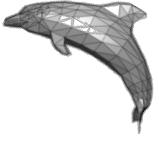
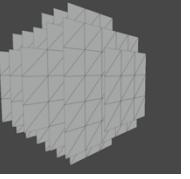
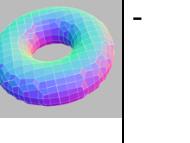
## **2.Les images en 3D :**

La création des images 3dimentions en imagerie médicale est un processus complexe qui dépend de plusieurs facteurs. D'une part la qualité des images produites joue un rôle crucial dans la construction d'image car il est reliée directement au bruit autrement dit des données médiocres produisent des résultats médiocres ce qui signifie que la précision et la résolution des données initiales influence directement la qualité de l'image 3D ,des éléments clés comme la haute résolution , l'amélioration du contraste et la minimisation des artefacts affectent considérablement la qualité des images obtenues. En d'autre part , le traitement des données brutes repose sur des algorithmes avancés de modélisation parmi ces algorithmes basés sur les méthodes de triangulation et les techniques d'interpolation volumique. Marching cubes, par exemple, permettent de reconstruire une surface 3D à partir d'images 2D en créant un maillage triangulaire des isosurfaces détectées. L'évolution récente des technologies d'intelligence artificielle et d'apprentissage automatique a également ouvert la voie à des approches plus automatisées et précises. Les réseaux neuronaux convolutifs (CNN) jouent un rôle clé dans la segmentation et l'extraction des structures anatomiques d'intérêt, tandis que les modèles U-Net sont fréquemment utilisés pour améliorer la qualité de la segmentation volumique.

## **Algorithmes de modélisation 3D :**

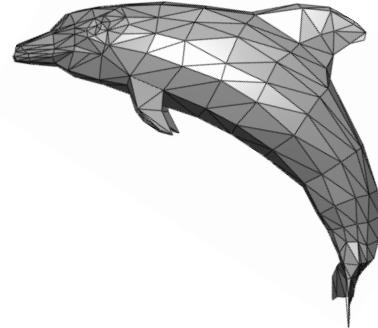


## Comparaison entre les différents algorithmes de construction d'image :

Critère	Cuberille	Marching Cubes	Dual Contouring	Surface Nets	Volume Rendering
<b>Qualité des surfaces :</b>	Faible	Moyenne	Haute	Moyenne	-
<b>Complexité de calcul :</b>	Faible	Moyenne	Élevée	Moyenne	Élevée
<b>Préservation des détails :</b>	Moyenne	Moyenne	Élevée	Faible	Élevée
<b>Utilisation typique :</b>	Visualisation Rapide	Application médicale	-	-	Application médicale
<b>Limites :</b>	<ul style="list-style-type: none"> <li>• Résolution de la surface</li> <li>• Ambiguités</li> <li>• Complexité</li> </ul>	<ul style="list-style-type: none"> <li>• Complexité élevée</li> <li>• Ambiguïtés(ex : deux cornes opposées)</li> <li>• Absence de bords et coins tranchants</li> </ul>	-	-	-
<b>Exemple :</b>					-

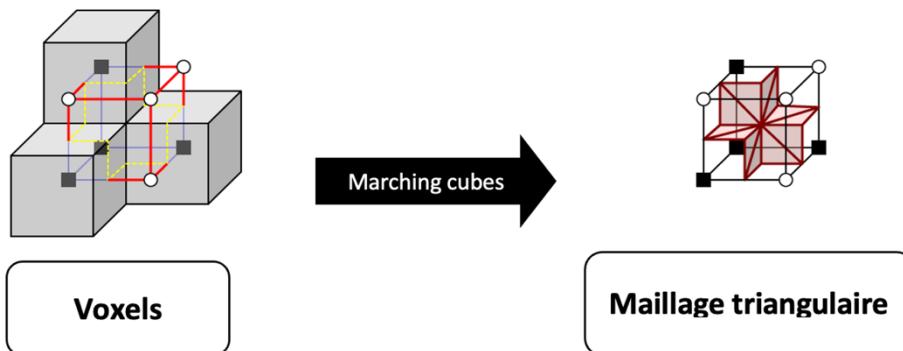
# Chapitre 1 :

# Marching cubes



## Introduction :

Le *Marching Cubes* est un algorithme fondamental utilisé en traitement d'images médicales pour la reconstruction de surfaces 3D à partir de données volumétriques. Cet algorithme parcourt le volume voxel par voxel, identifiant les intersections entre une surface d'isovaleur donnée et les arêtes des cubes formés par les voxels. Grâce à une table de correspondance prédéfinie, il génère des triangles pour représenter localement la surface, permettant ainsi de reconstruire des modèles 3D précis et détaillés tout en maintenant une grande efficacité.

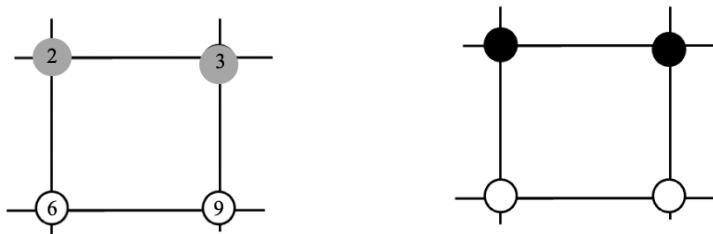


## Les données d'entrée :

L'entrée de l'algorithme MC est une voxélisation représentant un champ scalaire

$$v = f(x, y, z)$$

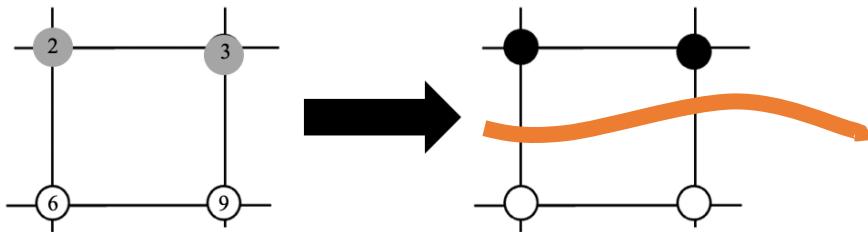
Le champ scalaire en entrée peut être de nature binaire ou non binaire.



### Isovalue (threshold) :

Lorsque les données d'entrée ne sont pas binaires, il convient d'ajouter un paramètre de classification supplémentaire, désigné par le terme isovalue.

**Exemple :**

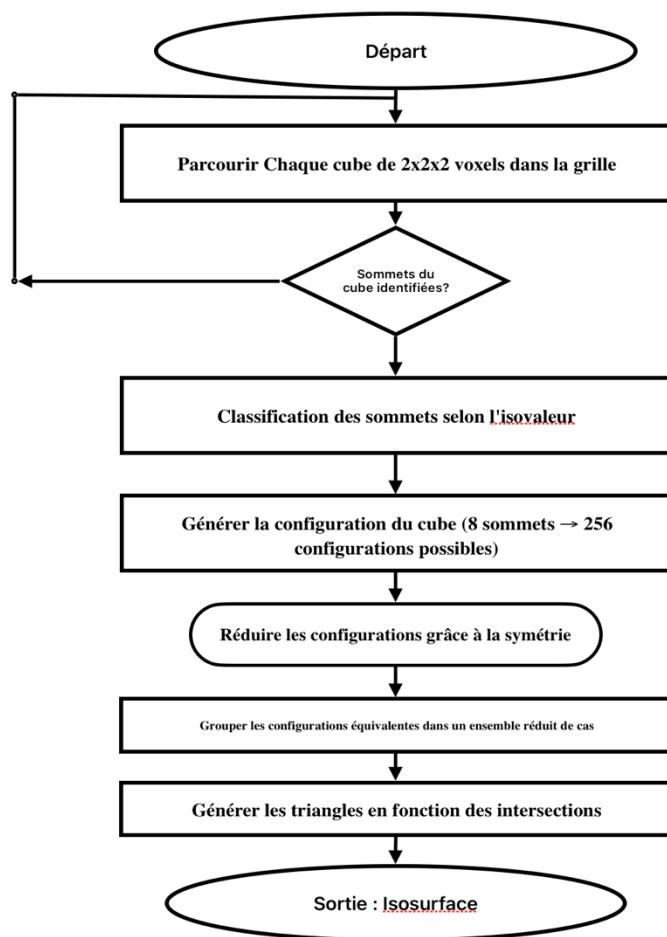


Isovalue=5

### Surface de résultante :

Dans le cas d'un modèle d'entrée binaire, l'objectif est de définir une surface séparant les points internes des points externes. Dans le cas d'un modèle d'entrée non binaire, il s'agit de déterminer l'iso surface reliant l'ensemble des points correspondant à la valeur iso spécifiée. La surface résultante doit distinguer clairement les points intérieurs des points extérieurs. En conséquence, elle doit être orientée et complètement fermée.

## Organigramme du Marching cubes :



### Remarque :

Pour chaque cube traité il faut extraire deux paramètres :

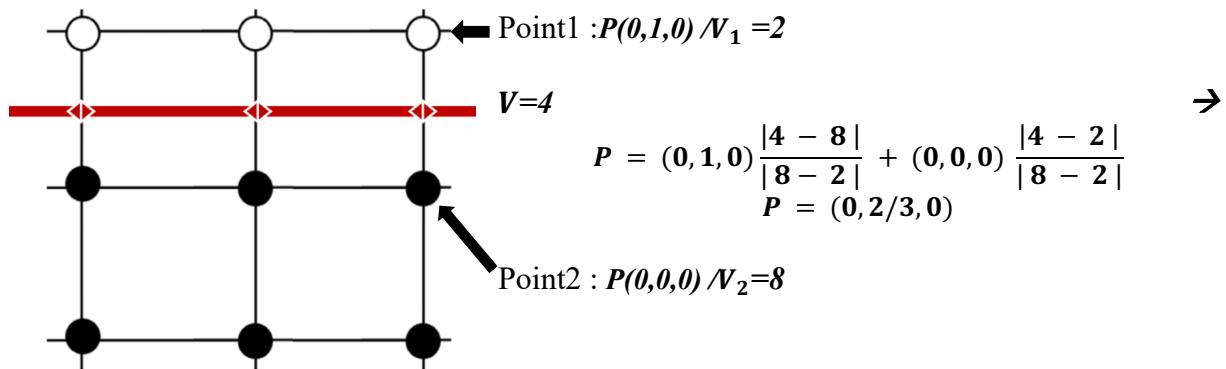
- La géométrie :Les sommets de l'isosurface
- La topologie : Les triangles connectent les sommets

En supposant que le champ est continu(variation sans interruption, exemple de distribution température), les arêtes présentant un changement de signe doivent être intersectées par l'isosurface.

### Création des sommets :

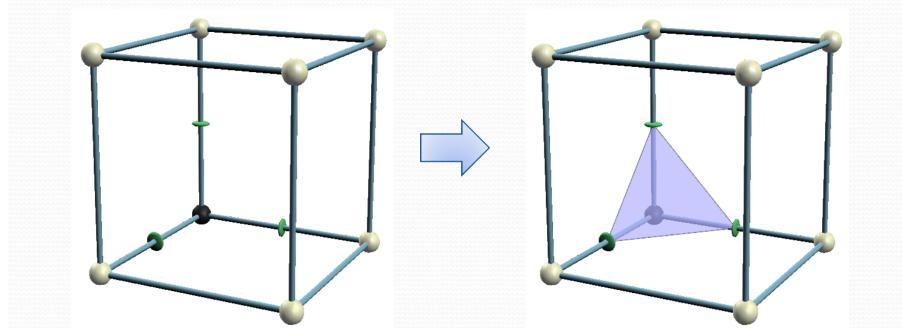
Lorsque l'isovaleur traverse une arête (parce que les valeurs changent de signe), un **nouveau sommet** est créé à l'endroit exact où l'isovaleur coupe l'arête selon la méthode d'interpolation linéaire :

$$P = P_1 \frac{|V - V_2|}{|V_2 - V_1|} + P_2 \frac{|V - V_1|}{|V_2 - V_1|}$$

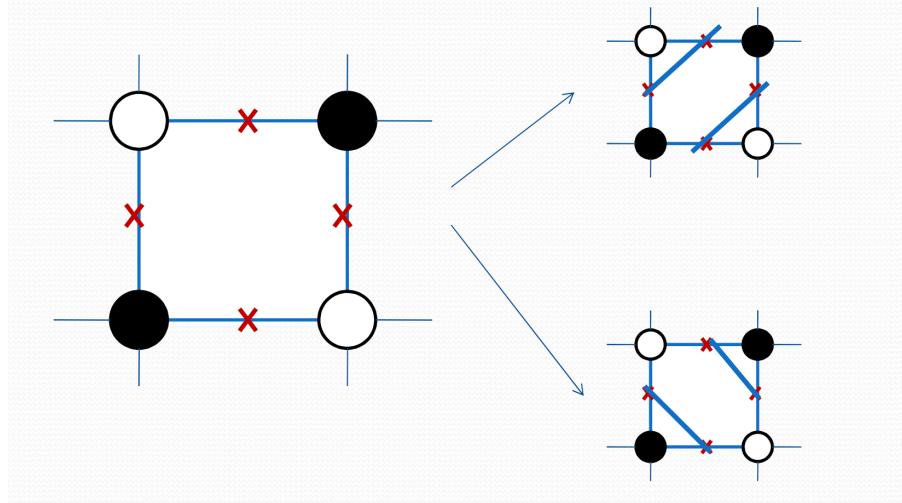


### Génération des triangles :

La plupart des cubes présentent un seul agencement possible pour les triangles. En fonction de la configuration des voxels à l'intérieur et à l'extérieur de la surface, un modèle de triangle unique est généré. Ce cas est appelé *cas non ambigu*.



Cependant, il existe des configurations spécifiques de voxels, appelées *cas ambigus*, dans lesquelles plusieurs agencements de triangles peuvent être créés. Ces configurations se produisent lorsqu'une surface traverse plusieurs voxels de manière complexe.



### Lien avec le concept des pixels :

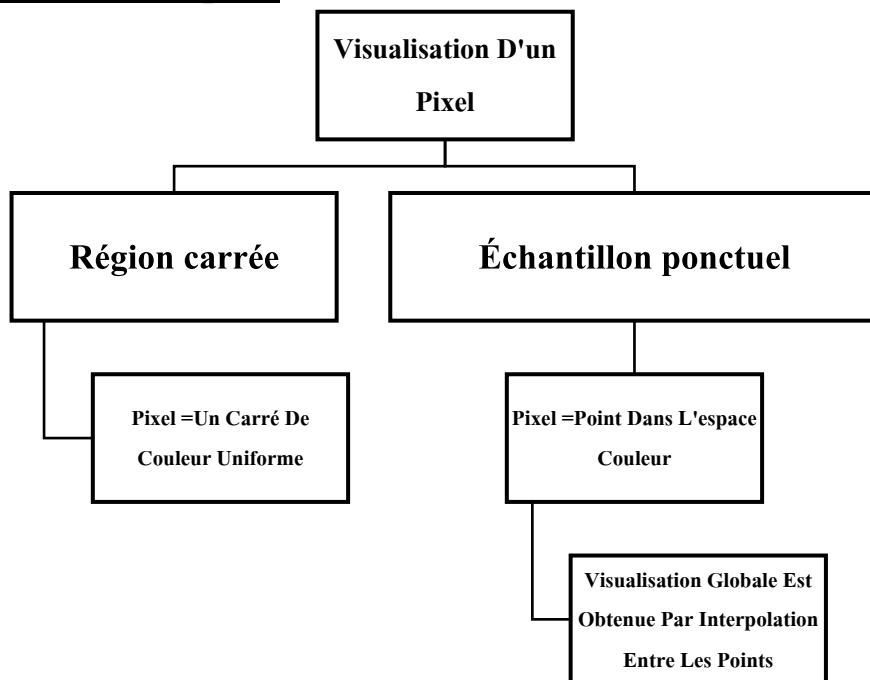
#### **Composition d'une image:**

Une image numérique est une grille composée de pixels. Chaque pixel possède des coordonnées (ligne et colonne) qui définissent sa position dans cette grille.

#### **Représentation numérique:**

Chaque pixel est associé à une valeur numérique qui correspond à sa couleur. Cette valeur peut être représentée de différentes manières, mais l'idée générale est de coder l'intensité des couleurs rouge, verte et bleue (système RVB) pour obtenir toutes les nuances possibles.

### Interprétations d'un pixel



### Application pratique

#### Socle technique du projet :

##### Opencv :

OpenCV est une bibliothèque open-source spécialisée dans la vision par ordinateur et l'apprentissage automatique. Elle offre des outils pour traiter et analyser des images.

##### Scikits learn :

Scikit-learn est une bibliothèque Python open-source spécialisée dans l'apprentissage automatique. Elle fournit des outils et des algorithmes pour des tâches telles que la classification, la régression, la réduction de dimensionnalité, ainsi que l'évaluation de modèles.



##### Numpy :

une bibliothèque Python pour le calcul scientifique, offrant des tableaux multidimensionnels et des routines pour des opérations rapides, telles que des calculs mathématiques, logiques, statistiques, et de l'algèbre linéaire.



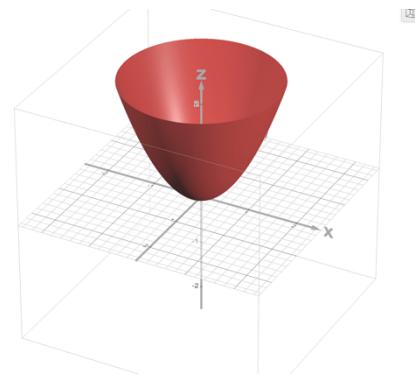
**NumPy**

## Modélisation Du Flux :Création D'un Masque

Le but principal de cette partie consiste à créer une représentation visuelle tridimensionnelle

on considère le masque suivant :

$$z \leq x^2 + y^2 \leq 1,5z$$

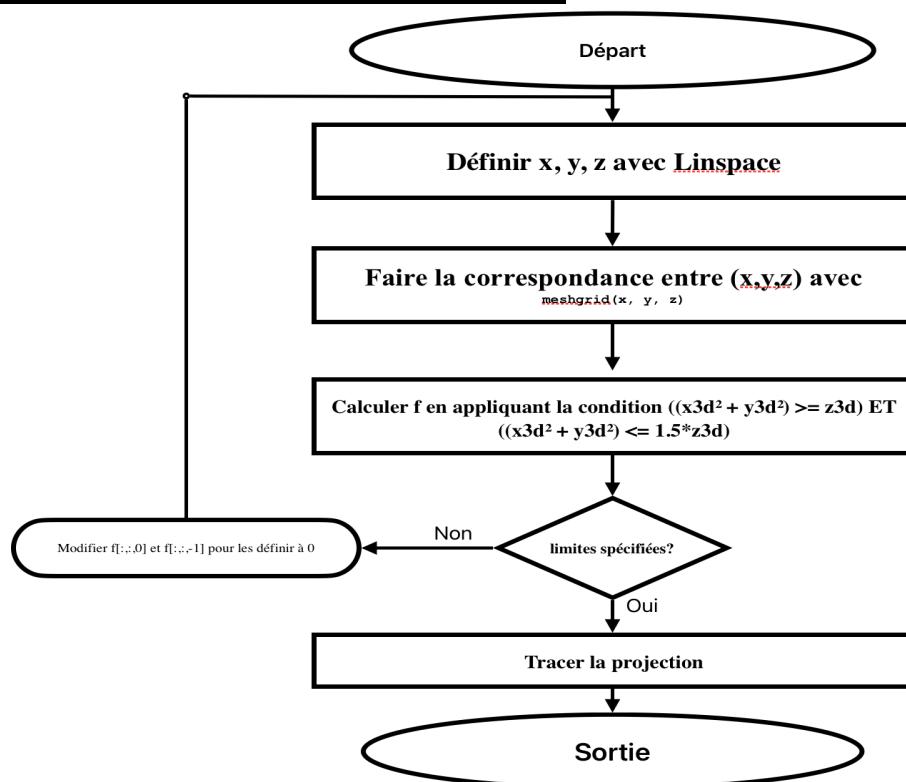


## Importation Des Bibliothèques

[1]

```
import numpy as np
import matplotlib.pyplot as plt
```

## Organigramme de la conversion 3D



## Code correspondant :

[2]

```
import numpy as np
import matplotlib.pyplot as plt
x=y= np.linspace(-1,1,100)
z=np.linspace(0,0.5,100)
x3d, y3d, z3d= np.meshgrid(x,y,z)
f=((x3d**2+y3d**2) >= z3d)*((x3d**2+y3d**2) <= 1.5*z3d)#selon la logique de boole
plt.pcolormesh(x3d,y3d,f[:, :, 50])
  
```

Selon z

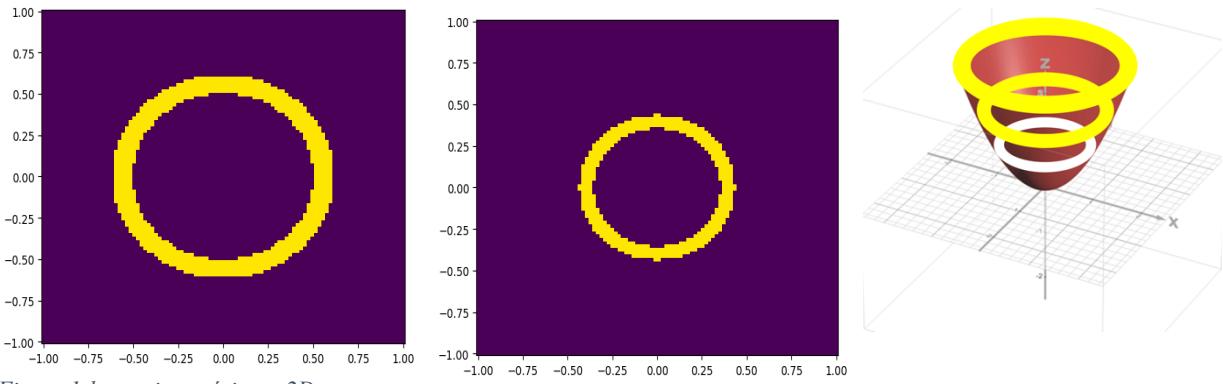


Figure 1: la partie supérieure 2D

L'utilisation de cette approche permet de faire une simulation des coupes 2D du CT-scanne qui vont être utilisée pour la reconstruction 3D

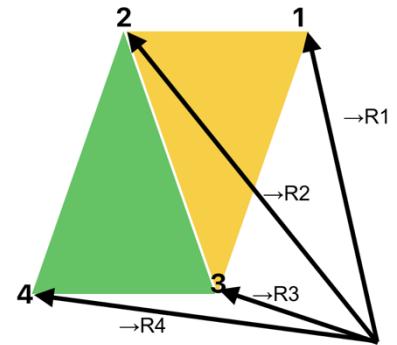
### **Modélisation 3D :**

La modélisation d'un objet nécessite l'utilisation des triangles selon des visages (l'algorithme du machine cubes ).

**Faces:** correspond à une surface plane qui constitue une partie de l'objet. Dans ce cas précis, le triangle est composé d'une seule face, définie par trois points (ou sommets).

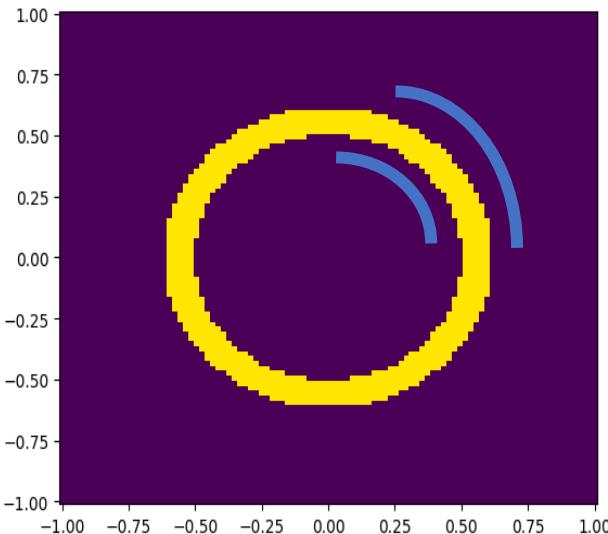
**Indices (1,2,3,4):** Ces nombres servent à identifier chaque point (ou sommet) du triangle. Par exemple, le point 1 est un sommet, le point 2 en est un autre, etc.

**Vecteurs (->R1, ->R2, ->R3, ->R4):** Un vecteur est une quantité qui possède à la fois une direction et une magnitude (une longueur). les vecteurs r<sub>1</sub>, r<sub>2</sub>, r<sub>3</sub> et r<sub>4</sub> représentent les positions des points 1, 2, 3 et 4 dans l'espace. Ils indiquent à quelle distance et dans quelle direction chaque point se trouve par rapport à un point de référence.



➤ **Donc le premier objectif pour pouvoir détecter ou spécifier l'objet dans l'espace consiste à déterminer les indices 1,2,3,4...**

Solution : **Machine\_cubes algorithm** de la bibliothèque scikit-image



Dans notre cas cette fonction va pouvoir trouver tous les points de la surface intérieur et extérieur pour pouvoir créer une image en trois dimensions , puis construire un volume en 3 dimensions .

[3]

```
import skimage as ski
from skimage import measure
verts, faces, normals, values = measure.marching_cubes(f)
```

Dans le cadre du **machring\_cubes** on ignore les deux facteurs **normals et values**

### 1.Verts:

```
>> verts
```

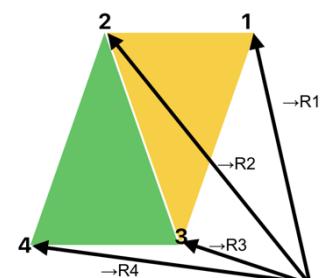
[4]

```
array([[ 6.5, 44. , 99. ],
       [ 7. , 44. , 98.5],
       [ 7. , 43.5, 99. ],
       ...,
       [92.5, 53. , 99. ],
       [92.5, 54. , 99. ],
       [92.5, 55. , 99. ]], dtype=float32)
```

Exemple :

[ 6.5, 44. , 99. ] correspond au premier point 1

[ 7., 44. , 98.5 ] correspond au point 2..



### 2.Faces:

Spécifie le triangle est composé d'une seule face, définie par trois points (ou sommets) par exemple dans notre cas il existe un triangle entre 1,2,et3 mais pas de triangle entre 1,3et 4.

```
>> faces
```

[5]

```
array([[ 2, 1, 0],
       [ 0, 1, 3],
       [ 3, 1, 4],
       ...,
       [50987, 50952, 50986],
       [50986, 50952, 50947],
       [50987, 50956, 50952]], dtype=int32)
```

Exemple :

[ 2, 1, 0]

Existence d'une relation entre 2,1 et 0 autrement dit un triangle constituée des points 2,1 et 0

➤ Ca c'est uniquement dans le cadre de Z = 50 que nous avons déjà spécifier au début

**Notre Modélisation 3D Utilise Uniquement Les Deux Entrées Faces Et Verts.**

### **La Fonction Meshgrid :**

#### **Principe de fonctionnement :**

Deux vecteurs d'entrée : Vous fournissez à **meshgrid** deux vecteurs, un pour les valeurs de x et un pour les valeurs de y.

Ces vecteurs définissent les points le long des axes x et y.

#### **Création de matrices :**

**meshgrid** répète chaque élément de ces vecteurs pour former deux matrices de la même taille.

La première matrice contient des copies horizontales du vecteur y.

La deuxième matrice contient des copies verticales du vecteur x.

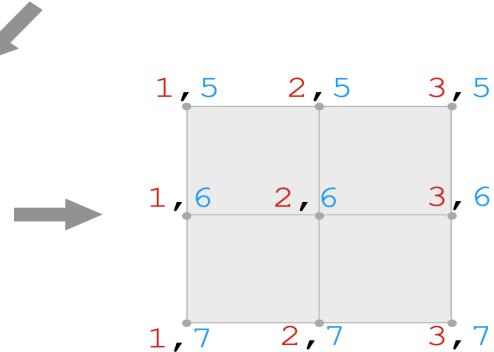
**Combinaison des matrices :** Ces deux matrices représentent les coordonnées de tous les points de la grille. En combinant les éléments correspondants de ces matrices, vous obtenez les coordonnées (x, y) de chaque point.

```
np.meshgrid([1, 2, 3], [5, 6, 7])
```

1	2	3
1	2	3
1	2	3

5	5	5
6	6	6
7	7	7



#### **Exemple :**

[6]

```
import numpy as np
# Créer deux vecteurs
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

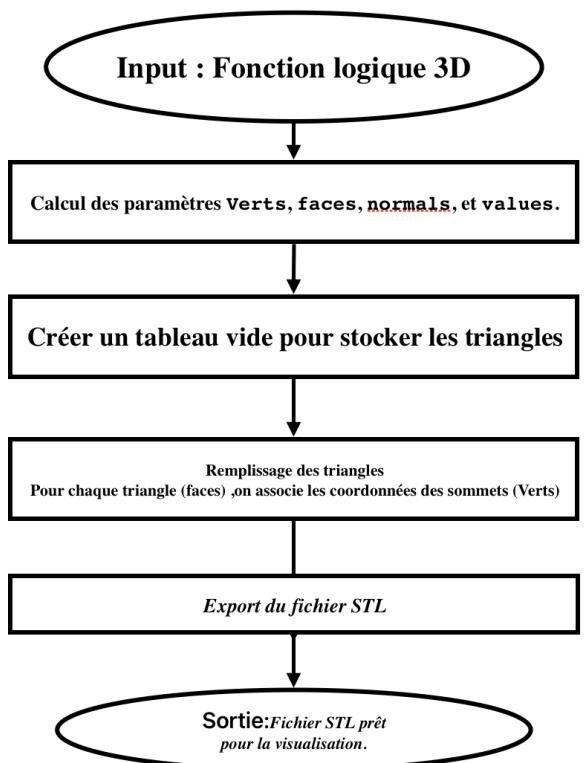
# Créer la grille
X, Y = np.meshgrid(x, y)

print(X)
print(Y)
```

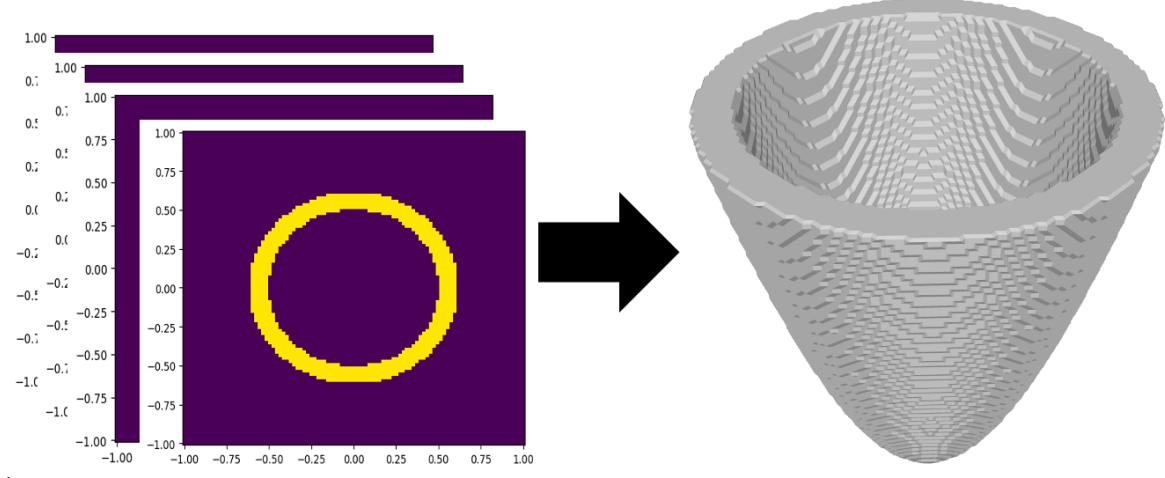
#### **Résultats :**

```
[[1 2 3]
 [1 2 3]
 [1 2 3]]
 [[4 4 4]
 [5 5 5]
 [6 6 6]]
```

### **Organigramme De Construction 3D Des Images :**



## Visualisation de l'objet :



# RECONSTRUCTION 3D DES IMAGES 2D DU CT-SCAN



## 1. Le flux de travail d'acquisition d'images :

Le principe fondamental de la technique de micro-CT repose sur l'acquisition d'un grand nombre de projections radiographiques (images 2D) d'un objet autour d'un axe de rotation. L'objet est placé entre une source de rayons X et un détecteur, et la variation de la distance entre la source et l'objet, ainsi qu'entre l'objet et le détecteur, permet d'obtenir une résolution plus fine par rapport aux scanners CT classiques utilisés en milieu clinique. Les projections, prises sous différents angles de l'échantillon, produisent des sinogrammes qui représentent des cartes d'atténuation. Ces sinogrammes sont ensuite utilisés pour générer des images de coupes transversales à l'aide d'algorithme de rétroprojection. Chaque image de projection est projetée en arrière pour reconstruire l'image, produisant des images floues appelées rétroprojections. Ce flou peut être réduit par des filtres mathématiques, et les algorithmes qui combinent rétroprojection et filtrage sont appelés "rétroprojection filtrée". En combinant ces images de rétroprojection, on peut localiser avec plus de précision la position de l'échantillon. À mesure que le nombre de projections augmente, la représentation de la forme et de la position de l'objet devient de plus en plus précise.

## Implémentation en Python :

### Importation des bibliothèques nécessaires :

[7]

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import measure
import cv2
from stl import mesh
import os
```

### Chargement de la base de données

On commence par la création d'une fonction `get_data()` de préprocessing qui permet de lire les images et les convertir en niveaux de gris.

[8]

```
def get_data(path):
    for i in range(2, 81):
        chemin = os.path.join(path, f"{i}.jpg")
        if os.path.exists(chemin):
            img_arr = cv2.imread(chemin, cv2.IMREAD_GRAYSCALE)
            if img_arr is not None:
                images.append(img_arr)
            else:
                print(f"Erreur : impossible de lire {chemin}")
        else:
            print(f"Fichier introuvable : {chemin}")
```

### Pourquoi la Conversion en niveaux de gris ?

La conversion en niveaux de gris est assuré par la fonction `IMREAD_GRAYSCALE` de la bibliothèque `open CV`, l'image est alors convertie en une matrice 2D où chaque élément représente l'intensité de lumière (luminosité) d'un pixel. Cette simplification réduit les données et simplifie les calculs .

### Vérification des dimensions des images :

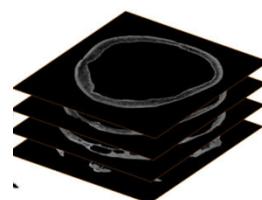
[9]

```
image_shapes = [img.shape for img in images]
```

On vérifie les dimensions de l'image pour assurer la cohérence si non on peut procéder directement à une redimensionne des images dès la première partie avec la fonction `cv2.resize()`.

### Empiler une séquence d'images 2D :

La fonction `.stack` est utilisée pour empiler plusieurs tableaux (images) 2D le long d'un nouvel axe pour former un tableau 3D.



[10]

```
volume_data = np.stack(images, axis=0).astype(np.float32)
```

## Normalisation des données :

### Extraction de la surface 3D (marching cubes) :

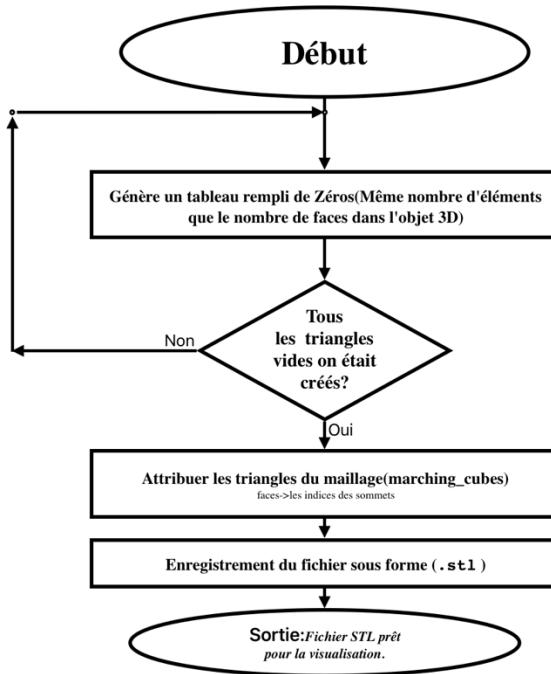
[11]

```
verts, faces, normals, values = measure.marching_cubes(volume_data)
```

### Création du fichier STL(stéréolithographie):

Algorithme de création du fichier :

1. On initialise un objet avec des triangles vides, qui seront remplis avec des données (positions des sommets, vecteurs normaux)
2. Une fois les données renseignées, on peut modifier les sommets, changer la géométrie et calculer les propriétés
3. Une fois terminé, le maillage peut être sauvegardé en tant que fichier STL pour l'impression 3D



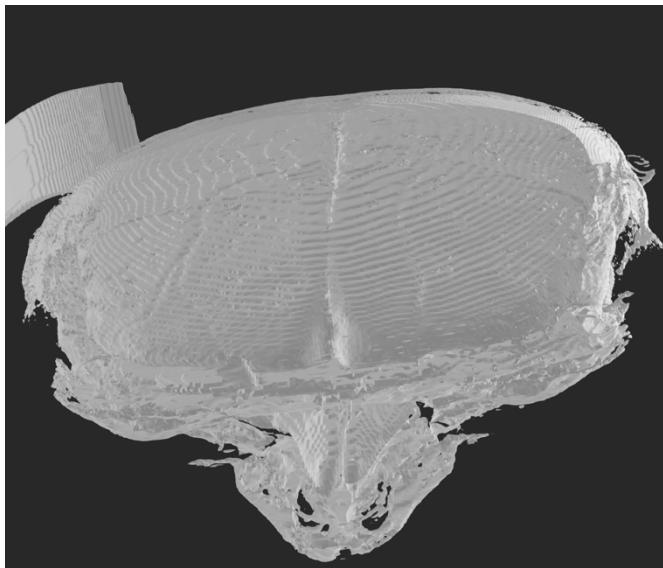
[12]

```
mesh_3d = mesh.Mesh(np.zeros(faces.shape[0], dtype=mesh.Mesh.dtype))
for i, f in enumerate(faces):
    mesh_3d.vectors[i] = verts[f]

output_file = 'mirleft1966.stl'
mesh_3d.save(filename=output_file)
print(f"Modèle 3D sauvegardé sous : {output_file}")
```

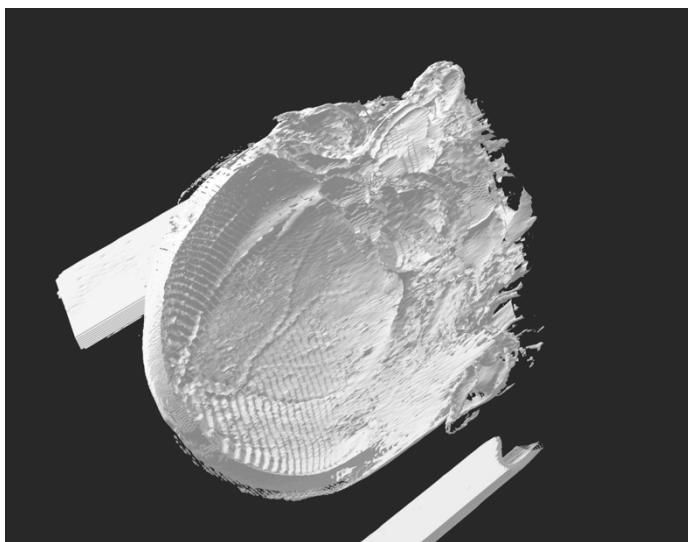
## Visualisation de l'objet :

Bien que les détails soient bien visibles, on remarque que la structure générale du modèle semble légèrement aplatie ou déformée, notamment en termes de



proportions et de profondeur. Cette apparence aplatie est due à une mauvaise échelle des axes, à une compression des données

### Ajustement des Paramètres de Spacing



Lors de la reconstruction du modèle, une déformation notable a été observée. Pour corriger cela, un ajustement des paramètres de **spacing** a été effectué. Cette modification a permis de rétablir les proportions correctes du modèle, assurant une représentation fidèle des dimensions et des détails du volume reconstruit.

[13]

```
spacing = (9, 2.0, 2.0)
verts, faces, normals, values = measure.marching_cubes(volume_data,
spacing=spacing)
```

### **Conclusion**

En conclusion, la modélisation 3D à partir d'images 2D issues d'un CT-scan représente une avancée significative dans le domaine de l'imagerie médicale. En exploitant des algorithmes tels que le *Marching Cubes* et en manipulant les paramètres essentiels comme le spacing, il devient possible de transformer des données brutes en une représentation volumétrique précise et riche en détails. Cette reconstruction en 3D dépasse les limites de l'imagerie 2D traditionnelle, en offrant une profondeur et une perspective permettant une meilleure visualisation des structures anatomiques complexes. Cette étape constitue une avancée non seulement dans le diagnostic et l'analyse clinique, mais également dans des domaines tels que la planification chirurgicale.

## **Amélioration du modèle :**

## **Ressources :**

<https://pmc.ncbi.nlm.nih.gov/articles/PMC8321043/>

https://dline.info/fpaper/jdim/v18i1/jdimv18i1\_1.pdf

https://www.sciencedirect.com/science/article/abs/pii/0146664X81901039

<https://www.kaggle.com/datasets/trainingdatapro/computed-tomography-ct-of-the-brain>

<https://www.sciencedirect.com/topics/computer-science/bilinear>

interpolation#:~:text=Bilinear%20interpolation%20is%20defined%20as,to%20the%20nearest%20neighbor%20approach.

https://ieeexplore.ieee.org/document/6746908/

<https://3dqlab.stanford.edu/what-is-3d-imaging-2/>

<https://numpy.org/>

[https://scikit-image.org/docs/stable/auto\\_examples/edges/plot\\_marching\\_cubes.html](https://scikit-image.org/docs/stable/auto_examples/edges/plot_marching_cubes.html)

<https://www.cs.montana.edu/courses/spring2005/525/students/Hunt1.pdf>

https://dl.acm.org/doi/abs/10.1145/280811.281026

Healthcare Solutions Using Machine Learning and Informatics, Dinesh Kumar Saini, Punit Gupta, Rohit Verma, Create citation