Activité – Python

Scanner laser : Désépaississement d'une ligne de pixels

I. Contexte

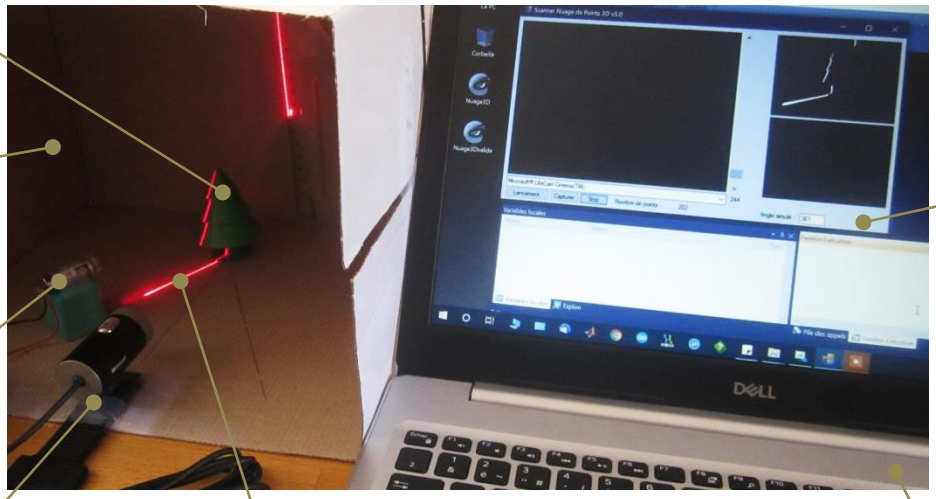
Un scanner 3D permet d'obtenir le modèle 3D d'un objet sous la forme d'un nuage de points.

Un nuage de points 3D est un ensemble de points dans un système de coordonnées qui servent à représenter le profil extérieur d'un objet :

Opération de scan 3D

Objet réel à scanner *Nuage de points obtenu*

Le scanner se compose des éléments suivants :



Vous disposez de ces différents éléments matériels mais pas de l'application logicielle complète.

Remarque : Le scanner ci-dessus permet uniquement de scanner des objets à révolution cylindrique car il est dépourvu de motorisation pour entrainer en rotation l'objet.

On vous propose de participer au début de l'élaboration logicielle du scanner 3D.

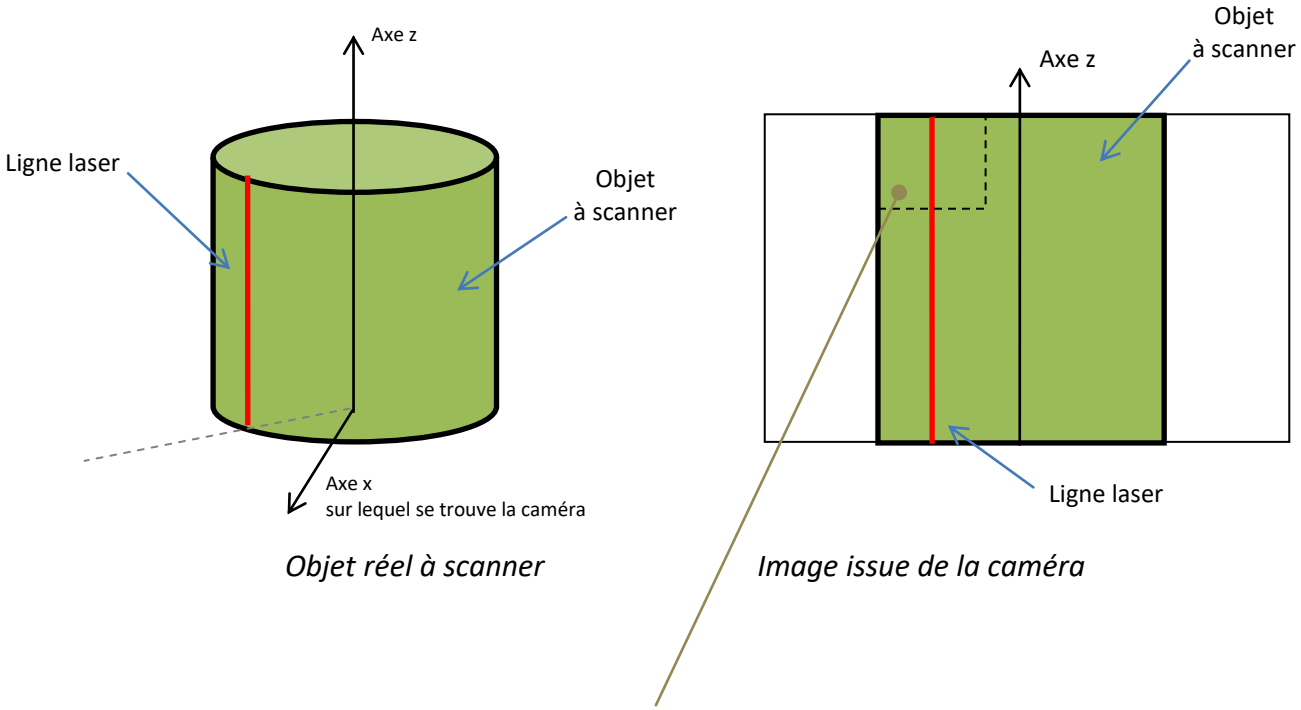
II. Etape du traitement logiciel

L'acquisition 3D d'un objet nécessite plusieurs phases de traitement logiciel :

a. Phase 1 : Acquisition de l'image

L'objectif de cette phase est de récupérer logiciellement l'image issue de la caméra.

Voici un exemple simplifié :



Objet réel à scanner *Image issue de la caméra*

	colonne 0	colonne 1	colonne 2	colonne 3	colonne 4	colonne 5	colonne 6	colonne 7	colonne 8	colonne 9
ligne 0										
ligne 1										
ligne 2										
ligne 3										
ligne 4										
ligne 5										
ligne 6										
ligne 7										
ligne 8										
ligne 9										

Zoom sur une petite portion de l'image issue de la caméra

La ligne laser apparait à la caméra comme une ligne verticale de plusieurs pixels d'épaisseur.

Voici un visuel réel :

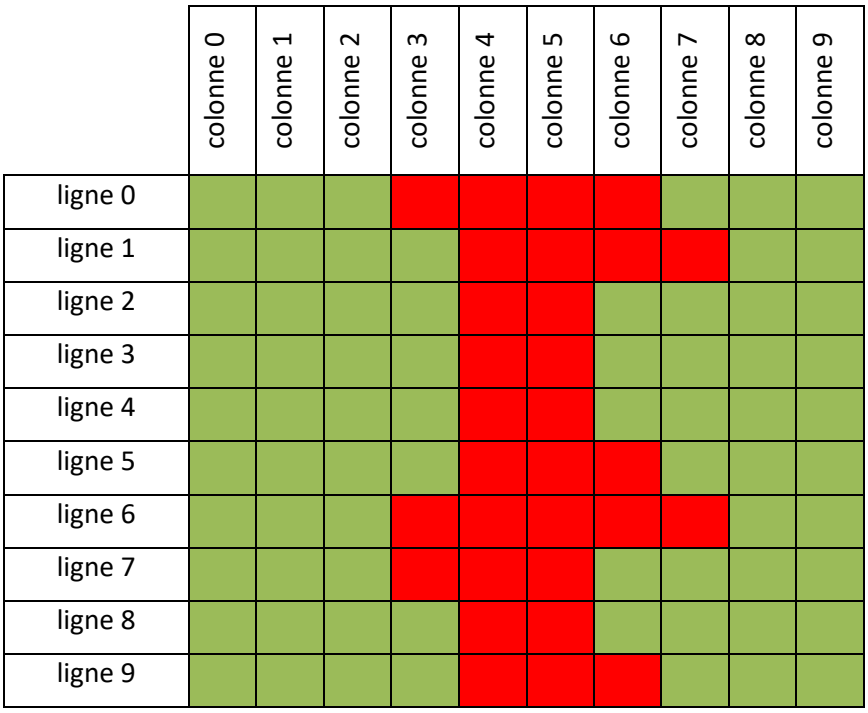
Voici le code python qui permet de faire cela :

```
import cv2
import time

cap = cv2.VideoCapture(0)
time.sleep(3)   # Attente focus caméra

angleObjet = 0

while (angleObjet < 360):   # Boucle principale

    _, frame = cap.read()
    cv2.imshow('frame', frame)

    angleObjet = angleObjet + 1
    print(angleObjet)

    if cv2.waitKey(1) == ord('q'):  # Un appui sur la touche q permet de quitter l'application
        break

cv2.destroyAllWindows()
cap.release()
```

Mise en oeuvre 1 : **Tester** le code ci-dessus et faire une capture de l'image obtenue. **Indiquer** ce qu'affiche le programme.

b. Phase 2 : Seuillage du niveau de rouge

A présent, l'image issue de la caméra doit être seuillée :



Zoom sur une petite portion de l'image issue de la caméra AVANT seuillage du niveau de rouge

Après seuillage du niveau de rouge, l'image est constituée uniquement de couleur blanche et noire. La zone blanche correspond à la zone où la couleur rouge était présente. Le reste devient noir.

	colonne 0	colonne 1	colonne 2	colonne 3	colonne 4	colonne 5	colonne 6	colonne 7	colonne 8	colonne 9
ligne 0	0	0	0	1	1	1	1	0	0	0
ligne 1	0	0	0	0	1	1	1	1	0	0
ligne 2	0	0	0	0	1	1	0	0	0	0
ligne 3	0	0	0	0	1	1	0	0	0	0
ligne 4	0	0	0	0	1	1	0	0	0	0
ligne 5	0	0	0	0	1	1	1	0	0	0
ligne 6	0	0	0	1	1	1	1	1	0	0
ligne 7	0	0	0	1	1	1	0	0	0	0
ligne 8	0	0	0	0	1	1	0	0	0	0
ligne 9	0	0	0	0	1	1	1	0	0	0

Zoom sur une petite portion de l'image issue de la caméra AVANT seuillage du niveau de rouge

Voici un visuel réel du seuillage :

Voici la fonction Python qui permet de seuiller l'image selon le niveau de rouge :

```
import cv2
import time

def etape1_seuillage(framecourante):
    framecouranteRGB = cv2.cvtColor(framecourante, cv2.COLOR_BGR2RGB)
    hauteur_image, largeur_image, canaux_image = framecouranteRGB.shape

    for ligne in range(hauteur_image):
        for colonne in range(largeur_image):
            if framecouranteRGB.item(ligne, colonne, 2) > 200:
                framecouranteRGB.itemset((ligne, colonne, 2), 255)
                framecouranteRGB.itemset((ligne, colonne, 1), 255)
                framecouranteRGB.itemset((ligne, colonne, 0), 255)
            else:
                framecouranteRGB.itemset((ligne, colonne, 2), 0)
                framecouranteRGB.itemset((ligne, colonne, 1), 0)
                framecouranteRGB.itemset((ligne, colonne, 0), 0)
    cv2.imshow('Image seuillee', framecouranteRGB)
    return framecouranteRGB

cap = cv2.VideoCapture(0)
time.sleep(3)   # Attente focus caméra

angleObjet = 0

while (angleObjet < 360):   # Boucle principale

    _, frame = cap.read()
    cv2.imshow('frame', frame)

    frameseuillee = etape1_seuillage(frame)

    angleObjet = angleObjet + 1
    print(angleObjet)

    if cv2.waitKey(1) == ord('q'):  # Un appui sur la touche q permet de quitter l'application
        break

cv2.destroyAllWindows()
cap.release()
```

✎ *Mise en oeuvre 2 :* **Ajouter** la fonction `etape1_seuillage()`. **Tester** le code et faire une capture de l'image obtenue.

c. CONCEPTION - Phase 2 : Désépaississement de la ligne de pixels

A présent, vous allez concevoir la fonction liée au désépaississement de la ligne de pixels.

Le désépaississement consiste à obtenir une ligne verticale constituée d'un seul pixel par ligne :

	colonne 0	colonne 1	colonne 2	colonne 3	colonne 4	colonne 5	colonne 6	colonne 7	colonne 8	colonne 9
ligne 0	0	0	0	1	1	1	1	0	0	0
ligne 1	0	0	0	0	1	1	1	1	0	0
ligne 2	0	0	0	0	1	1	0	0	0	0
ligne 3	0	0	0	0	1	1	0	0	0	0
ligne 4	0	0	0	0	1	1	0	0	0	0
ligne 5	0	0	0	0	1	1	1	0	0	0
ligne 6	0	0	0	1	1	1	1	1	0	0
ligne 7	0	0	0	1	1	1	0	0	0	0
ligne 8	0	0	0	0	1	1	0	0	0	0
ligne 9	0	0	0	0	1	1	1	0	0	0

Zoom sur une petite portion de l'image issue de la caméra AVANT désépaississement de la ligne

	colonne 0	colonne 1	colonne 2	colonne 3	colonne 4	colonne 5	colonne 6	colonne 7	colonne 8	colonne 9
ligne 0	0	0	0	0	0	1	0	0	0	0
ligne 1	0	0	0	0	0	0	1	0	0	0
ligne 2	0	0	0	0	0	1	0	0	0	0
ligne 3	0	0	0	0	0	1	0	0	0	0
ligne 4	0	0	0	0	0	1	0	0	0	0
ligne 5	0	0	0	0	0	1	1	0	0	0
ligne 6	0	0	0	0	0	1	1	0	0	0
ligne 7	0	0	0	0	1	0	0	0	0	0
ligne 8	0	0	0	0	0	1	0	0	0	0
ligne 9	0	0	0	0	0	1	1	0	0	0

Zoom sur une petite portion de l'image issue de la caméra APRES désépaississement de la ligne

Logiciellement parlant, le désépaississement de la ligne de pixels consiste à considérer successivement les lignes issues de la caméra (image seuillée).

Considérons uniquement la ligne 0 avant désépaississement :

	colonne 0	colonne 1	colonne 2	colonne 3	colonne 4	colonne 5	colonne 6	colonne 7	colonne 8	colonne 9
ligne 0	0	0	0	1	1	1	1	0	0	0

Pour chacune des lignes (ici la ligne 0) :

Etape A : on parcourt les colonnes à partir de la colonne 0 tant que le pixel rencontré est de couleur noire (valeur de 0).

Programme en pseudo-langage :

```
# image(ligne,colonne) est une fonction qui retourne la couleur du pixel (0 noir et 1 blanc)

DEBUT
ligne_courante := 0                                  # indice de la ligne courante
colonne := 0                                         # indice de la colonne
Tant que ( image( ligne_courante, colonne ) = 0 )    # test si le pixel est noir
        colonne := colonne + 1                       # on incrémente de 1 l'indice de colonne
Fin Tant que
debut_blanc := colonne
FIN
```

Question 3 : Que valent les variables `colonne` et `debut_blanc` à l'issue de l'étape A ?

Etape B : arrivé au niveau du premier pixel blanc, on parcourt les colonnes (donc ici à partir de la colonne 3) tant que le pixel rencontré est blanc (valeur de 1). On profite de ce parcours pour mettre à 0 (en blanc) les pixels de valeurs 1 parcourus.

Programme en pseudo-langage :

```
# imagemodif() est une fonction qui permet de modifier la couleur du pixel (0 noir et 1 blanc)
# debutblanc = 3 à l'issue de l'étape A

DEBUT
colonne := 0                                                        # indice de la colonne
Tant que ( image( ligne_courante, debut_blanc + colonne ) = 1 )     # test si le pixel est blanc
        imagemodif( ligne_courante, debut_blanc + colonne , 0 )     # mise à zéro du pixel
        colonne := colonne + 1                                      # incrémente de 1 colonne
Fin Tant que
FIN
```

A la fin de l'étape 2, la ligne de pixels ressemble à cela :

	colonne 0	colonne 1	colonne 2	colonne 3	colonne 4	colonne 5	colonne 6	colonne 7	colonne 8	colonne 9
ligne 0	0	0	0	0	0	0	0	0	0	0

Question 4 : Que vaut la variable `colonne` à l'issue de l'étape B ?

Etape C : à cette étape on cherche à placer un pixel blanc le plus au centre possible de la zone où les pixels blancs étaient présents.

Programme en pseudo-langage :

```
# imagemodif(ligne,colonne,v) est une fonction modifiant la couleur du pixel (v=0 noir et v=1 blanc)
# debutblanc = 3 et colonne = 4 à l'issue de l'étape B

DEBUT
colonne_pixel_central = debutblanc + ( colonne // 2 )        # // est une division entière
imagemodif( ligne_courante, colonne_pixel_central , 1 )      # mise à un du pixel (blanc)
FIN
```

Question 5 : Que vaut la variable `colonne_pixel_central` dans le cas de la ligne ? **Justifier** par un calcul.

Question 6 : **Indiquer** ci-dessous à quoi ressemble la ligne de pixels à la fin de l'étape C.

	colonne 0	colonne 1	colonne 2	colonne 3	colonne 4	colonne 5	colonne 6	colonne 7	colonne 8	colonne 9
ligne 0	0	0	0	0	0	1	0	0	0	0

Question 7 : Que vaut la variable `colonne_pixel_central` dans le cas des lignes de 1 à 9 ? **Justifier** par un calcul.

On se propose de concevoir la fonction `etape2_desepaisse()` qui va permettre le désépaississement de la ligne de pixels.

Le gabarit de la fonction `etape2_desepaisse()` sera le suivant :

Programme Python :

```
def etape2_desepaisse(framecouranteRGB):

    hauteur_image, largeur_image, canaux_image = framecouranteRGB.shape

    for ligne in range(hauteur_image):

    # VOTRE CODE SERA A AJOUTER ICI

    cv2.imshow('Image seuillee et ligne desepaissie', framecouranteRGB)
    return framecouranteRGB
```

L'objet `framecouranteRGB` est représentatif de l'image seuillée.

La variable `hauteur_image` contient la hauteur de l'image.

La variable `largeur_image` contient la largeur de l'image.

La boucle `for ligne in range(hauteur_image):` permet de parcourir les lignes de l'image.

La ligne `cv2.imshow('Image seuillee et ligne desepaissie', framecouranteRGB)` permettra d'afficher l'image à l'issue du désépaississement.

La ligne `return framecouranteRGB` renvoie l'objet représentatif de l'image à l'issue du désépaississement.

Chaque pixel de l'image est codé par trois octets représentant le niveau de rouge (sous-pixel rouge), vert (sous-pixel vert) et bleu (sous-pixel bleu) du pixel.

Le noir est codé en décimal de la façon suivante : 0 0 0

Le blanc est codé en décimal de la façon suivante : 255 255 255

En Python, pour mettre en noir un pixel situé à la ligne 8 et la colonne 9 on écrit :

```
framecouranteRGB.itemset((8, 9, 2), 0)   # Niveau de rouge à 0
framecouranteRGB.itemset((8, 9, 1), 0)   # Niveau de vert à 0
framecouranteRGB.itemset((8, 9, 0), 0)   # Niveau de bleu à 0
```

En Python, pour mettre en blanc un pixel situé à la ligne 8 et la colonne 9 on écrit :

```
framecouranteRGB.itemset((8, 9, 2), 255)   # Niveau de rouge à 255
framecouranteRGB.itemset((8, 9, 1), 255)   # Niveau de vert à 255
framecouranteRGB.itemset((8, 9, 0), 255)   # Niveau de bleu à 255
```

Comme l'image est constituée de pixels de couleur noire ou blanche, en Python pour tester si un pixel est de couleur blanche il suffit de tester si un des sous-pixels a une valeur de 255.

Par exemple pour tester si le pixel de la ligne 8 et de la colonne 9 est blanc, on va simplement tester si le sous-pixel rouge est à 255 en écrivant la condition suivante :

```
framecouranteRGB.item(8, 9, 2) == 255
```

Question 8 : A l'aide de toutes les indications précédentes et du programme en pseudo-langage, **écrire** le programme Python de la fonction `etape2_desepaisse()`.

Pour rappel le pseudo-langage (pour une ligne) associé à la fonction `etape2_desepaisse()` est le suivant :

```
# image(ligne,colonne) est une fonction qui retourne la couleur du pixel (0 noir et 1 blanc)

DEBUT
ligne_courante := 0                                  # indice de la ligne courante
colonne := 0                                         # indice de la colonne
Tant que ( image( ligne_courante, colonne ) = 0 )    # test si le pixel est noir
        colonne := colonne + 1                       # on incrémente de 1 l'indice de colonne
Fin Tant que
debut_blanc := colonne

# imagemodif() est une fonction qui permet de modifier la couleur du pixel (0 noir et 1 blanc)

colonne := 0                                                          # indice de la colonne
Tant que ( image( ligne_courante, debut_blanc + colonne ) = 1 )       # test si le pixel est blanc
        imagemodif( ligne_courante, debut_blanc + colonne , 0 )       # mise à zéro du pixel
        colonne := colonne + 1                                        # incrémente de 1 colonne
Fin Tant que

# imagemodif(ligne,colonne,v) est une fonction modifiant la couleur du pixel (v=0 noir et v=1 blanc)

colonne_pixel_central = debutblanc + ( colonne // 2 )              # // est une division entière
imagemodif( ligne_courante, colonne_pixel_central , 1 )           # mise à un du pixel (blanc)

FIN
```

La solution :

Question 9 : **Tester** le programme avec le matériel (il faut connecter la caméra USB avant de lancer le programme). **Commenter**. **Expliquer** d'où peut provenir le problème.

Question 10 : **Modifier** votre fonction en conséquence. **Tester** votre fonction. **Faire** une capture d'écran du résultat obtenu.

La solution :

Ici la hauteur de l'image est de 480 pixels soit 480 points maximum de données brutes par image. Pour la suite, il conviendrait de pouvoir choisir le nombre de points de données brutes à acquérir par image afin de limiter la quantité de données à traiter.

Par exemple, ne récupérer que 50% de 480 points par image.

Pour ce faire on propose d'ajouter cette ligne de code dans la fonction `etape2_desepaisse()` :

```
niveau = 255 * ((ligne % int(100/pourcentage)) == 0)
```

Question 11 : Sachant que la variable `ligne` évolue de 0 à 479 dans votre cas, **indiquer** comment la variable niveau va évoluer si la variable `pourcentage` a pour valeur 50.

Question 12 : **Ajouter** la gestion de la valeur de ce pourcentage de pixels en ajoutant les lignes de code suivantes au bon endroit. **Modifier** votre fonction en conséquence. **Tester** votre fonction. **Faire** une capture d'écran du résultat obtenu.

A placer au début du programme :

```
pourcentage_points_verticaux = int(input("Saisir le pourcentage (entre 0 et 100) de points verticaux présent sur la ligne : "))
```

A placer lors de l'appel de la fonction `etape2_desepaisse()` (bien penser à modifier la définition de la fonction en ajoutant le paramètre `pourcentage`) :

```
framedesepaisse = etape2_desepaisse(frameseuillee, pourcentage_points_verticaux)
```

A placer dans la fonction etape2_desepaisse() :

```
niveau = 255 * ((ligne % int(100/pourcentage)) == 0)
```

La solution :

Voilà, vous avez contribué au développement logiciel d'un scanner 3D, évidemment cela n'est pas terminé il reste à :

- Calculer les coordonnées 3D
- Générer le fichier de données 3D
- Mettre en place la rotation de l'objet.

Question 13 : Le programme complet est fourni en ressource, je vous invite à l'essayer !