République Algérienne Démocratique et Populaire

Ministère de l'enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

VISION ARTIFICIELLE

RAPPORT PARTIE II

Reconstruction 3D photométrique

Sujet :

Trinôme:

TEBBANI Mohamed Walid

SEDKAOUI Amine

ADMANE Hocine

Etape 1. Préparation de données :

- Création de la fonction « ***load_lightSources » qui*** a pour objectif de chargé le fichier « ***light_directions.txt »*** dans une liste de listes (matrice), chaque ligne du fichier est stockée dans une liste.

```
def load_light_sources():
    global light_diections
    file = open(light_diections, 'r')
    arr = []
    while True:
        line = file.readline()
        if not line:
            break
        temp = line.strip().split(' ')

        temp = [float(x) for x in temp] # Change from str to float

        arr.append(temp)
    file.close()
    return arr
```

- Création de la fonction « ***load_intensSources*** » qui a pour objectif de chargé le fichier « ***light_intensities.txt*** » dans une liste de listes (matrice), chaque ligne du fichier est stockée dans une liste.

```
def load_intense_sources():
    global light_intensities
    file = open(light_intensities, 'r')
    arr = []

    while True:
        line = file.readline()
        if not line:
            break
        temp = line.strip().split(' ')
        temp = [float(x) for x in temp]  # replace str with float
        arr.append(temp)
    file.close()
    return arr
```

- Création de la fonction « ***load_objMask »,*** cette fonction lit le fichier image « ***mask.png*** » , parcourt ses pixels de niveau de gris et compare leurs valeurs à la valeur plafond « ***ceil*** » , si la valeur du pixel est inférieur à « ***ceil*** » alors la valeur correspondante dans la matrice binaire de retour reçoie 0 (pixel de fond) , sinon elle reçoie 1 (pixel de l'objet).

```
def load_obj_mask():
    global mask
    mat = cv.imread(mask, cv.IMREAD_GRAYSCALE)
    rows, columns = mat.shape
    ceil = 255 // 2      # default

    for i in range(columns):
        for j in range(rows):
            if mat[j, i] < ceil:
                mat[j, i] = 0
            else:
                mat[j, i] = 1
    return mat
```

- Création de la fonction « ***load_images*** » qui permet de charger les 96 images du dataset. Cette fonction aura besoin de trois autres fonctions :

```
def load_images(flag):
    if flag == 1:
        return create_matrix_e()  # creates the file (E matrix)
    else:
        return load_matrix_e()  # loades the file if already created
```

  - La fonction « ***treat_image(img_file , light_intensity)*** » qui a pour but de :
    1. Changer l'intervalle des valeurs de ***uint16*** [0, 2^{16} -1] à ***float32*** [0,1] à l'aide de la fonction « ***normalize*** » de OpenCV :

```
def treat_image(img_file, intense_row):
    # Read
    image = cv.imread(img_file, cv.IMREAD_UNCHANGED)
    # Normalize float 32 from 0 to 1
    image = cv.normalize(image, None, alpha=0, beta=1, norm_type=cv.NORM_MINMAX, dtype=cv.CV_32F)
```

    2. Diviser chaque pixel de « ***img_file*** » sur l'intensité de la source :

```
for i in range(columns):
    for j in range(rows):
        # image(BGR) is divide by intensity (RGB)
        image[j, i] = image[j, i, 0] / intense_row[2], image[j, i, 1] / intense_row[1], image[j, i, 2] /intense_row[0]
```

    3. Convertir chaque pixel de « ***img_file*** » en niveau de gris :

```
# Change to grey image
image_gray[j, i] = ((image[j, i, 0]*.3) + (image[j, i, 1]*.59) + (image[j, i, 2]*.11)) / 3
```

    4. Redimensionner l'image telle que chaque image est représentée dans une seule ligne :

```
# Reshape into 1 row and return
return image_gray.reshape(1, rows * columns)[0]
```

- La fonction « ***create_matrix_e*** », cette fonction permet de faire le traitement de ***« treat_image*** » vue précédemment sur chacune des images du dataset et stocké ensuite le résultat dans un fichier « ***matrix_e*** » puis dans un tableau pour former une matrice de N lignes et (h*w) colonnes où chaque ligne représente une image :

```
def create_matrix_e():  # save matrix E
    global matrix_e, filenames, resource
    file = open(matrix_e, "w")
    # Read intensity file into list
    intense = load_intense_sources()
    # Read pictures filenames
    filenames = open(filenames)
    # output list: contains list of (N, w*h) of one lined images
    images_list = list()
    itr = 0
    while True:
        filename = filenames.readline().strip()
        if not filename:
            # end of file (filenames.txt)
            break
        treat = treat_image(resource + filename, intense[itr])
        itr += 1
        print(itr)
        for elem in treat:
            file.write(str(elem) + " ")
        file.write("\n")
        images_list.append(treat)
    file.close()
    filenames.close()
    return images_list
```

- La fonction « ***load_matrix_e*** » qui permet de charger le fichier « ***matrix_e*** » dans un tableau de N lignes et (h*w) , le fichier « ***matrix_e*** » doit être préalablement crée par la fonction « ***create_matrix_e*** » :

```
def load_matrix_e():
    global matrix_e
    file = open(matrix_e, 'r')
    arr = []
    while True:
        line = file.readline()
        if not line:
            break
        temp = line.strip().split(' ')
        temp = [float(x) for x in temp] # Change from str to float
        arr.append(temp)
    file.close()
    return arr
```

Remarque : sachant que le traitement de toutes les images du dataset prend un temps considérable à être réaliser, les deux fonctions « create_matrix_e » et « load_matrix_e » on étaient crées dans le but d'éviter de refaire le traitement plusieurs fois

Etape 2. Calcul des normales :

- Création de la fonction « ***calcul_needle_map*** », cette fonction calcule les normales de l'objet et retourne une matrice de h lignes et w colonnes, chaque élément de la matrice contient trois composantes x, y , z .Avant d'aboutir à ce résultat la fonction doit passer par les étapes suivantes :

    - Calcul de la matrice pseudo inverse à partir de la matrice des positions des sources lumineuses « ***light_sources*** » à l'aide de la fonction « ***pinv*** » de « ***numpy.linalg*** » :

```
def calcul_needle_map():    # creates n wich is S^-1 * E
    global matrix_e, normales
    #obj_images = load_images(2)
    light_sources = load_light_sources()
    #obj_masques = load_obj_mask()
    file = open(normales, "w")
    file2 = open(matrix_e, "r")

    matE = list()
    # create inverted S
    s_inv = numpy.linalg.pinv(light_sources)
```

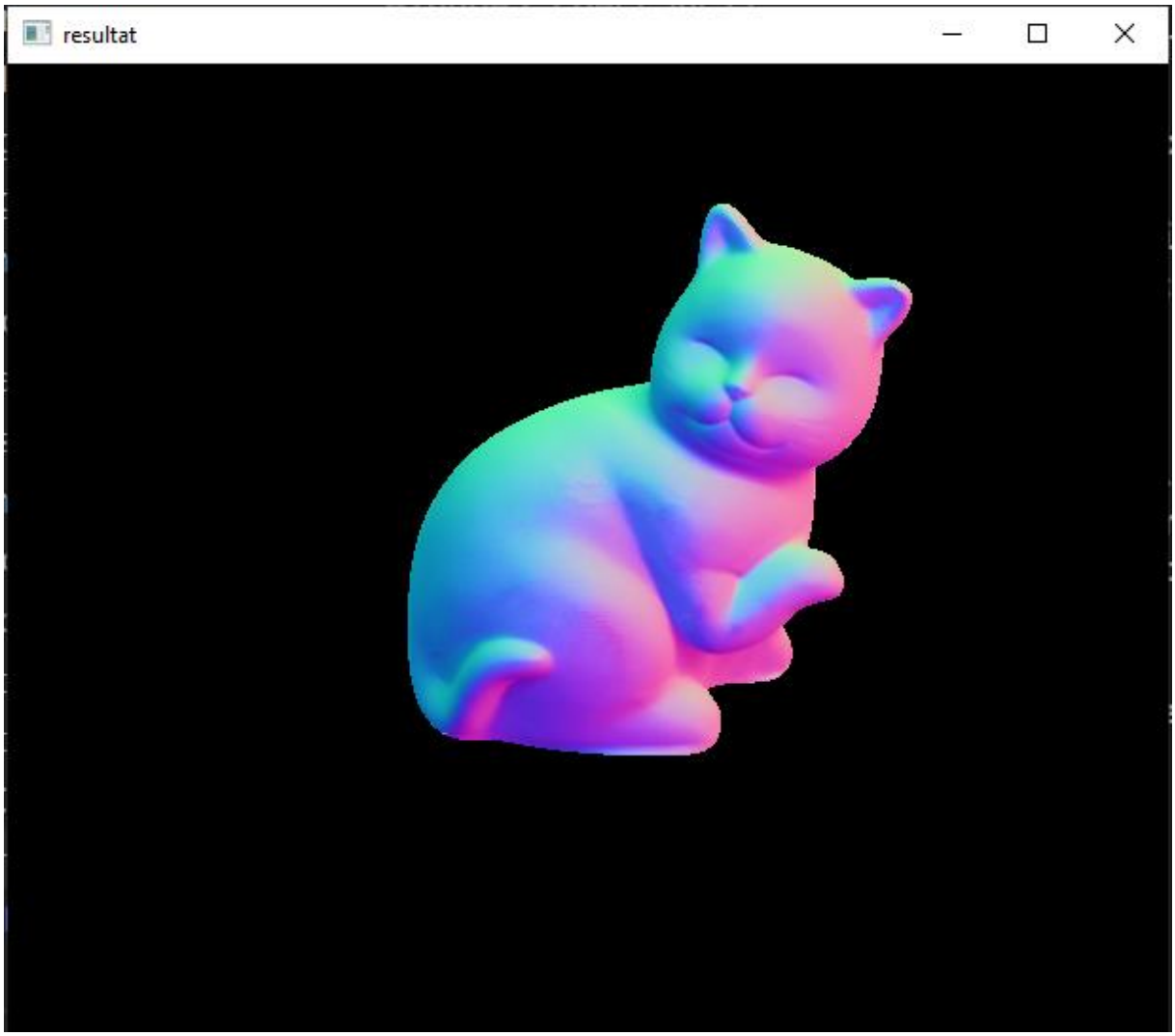
    - Calcul des normales en se basant sur la modélisation vue en cours :
      $N = S^{-1} * E$
      en utilisant la fonciton « ***matmul*** » de « ***numpy*** » puis normaliser le résultat en divisant les composantes x , y , z de chaque pixel par ||N|| :

```
# multiply s^-1 * E
n=numpy.matmul(s_inv,matE)
sum=n[0,:]*n[0,:]#calcule modulo ||N||
for i in range(1,3):
    sum+=n[i,:]*n[i,:]
s1=numpy.sqrt(sum)
for i in range(3):
    n[i,:] = n[i,:] / s1
```

    - Sauvegarde des normales dans un fichier « ***normales.txt*** » :

```
# write down n
for i in range(3):
    for j in range(612*512):
        file.write(str(n[i,j]) + " ")
    file.write("\n")
file.close()
```

Voila le résultat de l'exécution :



Etape 3. Calcul la profondeur Z :

- D’après la formule de cours :

$$N = \frac{-p, -q, 1}{\sqrt{p^2 + q^2 + 1}}$$

Avec N= (x, y, z) ;

On a déduit que le p et le q sont égaux a :

P=-x/z et q=-y/z.

- Donc on a appliqué les formules précédentes dans notre cas on a eu deux listes, pour chaque vecteur N on a eu p et q.

```
for k in range(512*612):
        p=-n[0,k]/n[2,k] #calcule de q=-y/z
        q=-n[1,k]/n[2,k]    #calcule de p=-x/z
        res_q.append(q)
        res_p.append(p)
```

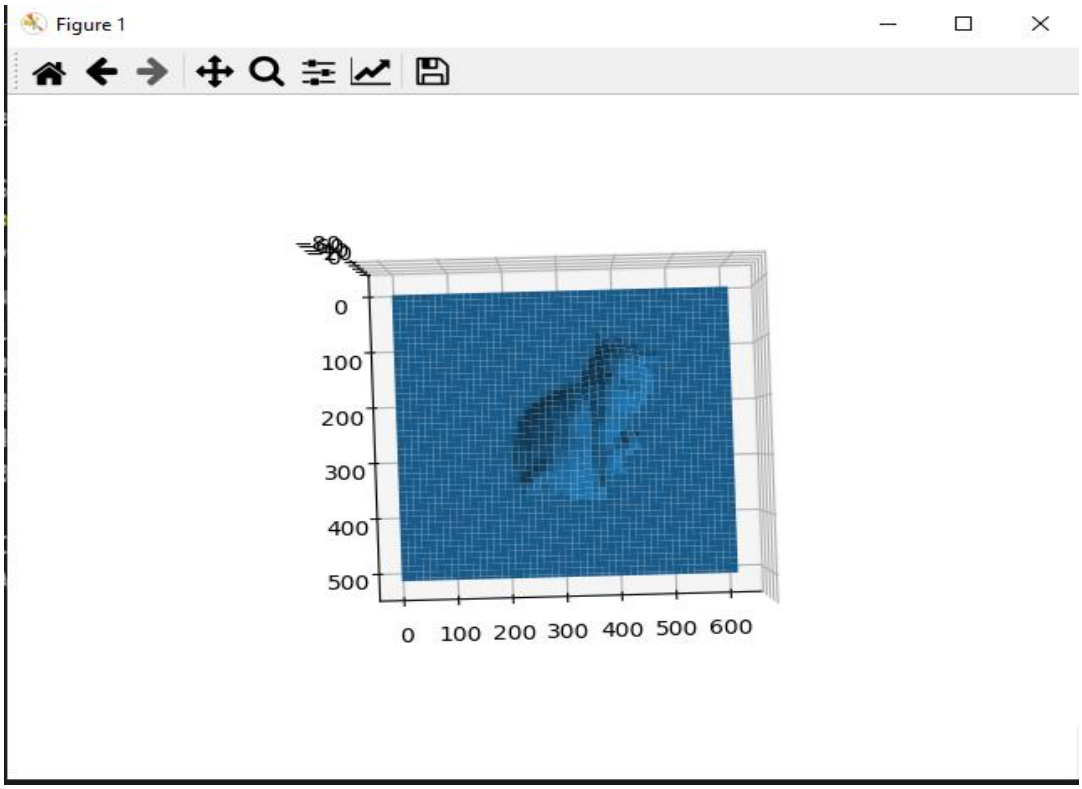
- Après avoir eu les deux listes précédentes de p et q, on a multiplié les deux listes par le Mask, après on a calculer le Zn pour l'axe X et l'axe Y. Avec la formule suivante :
  On fixe Z0=0 ; et on calcule ZN =Z(N-1) +p avec p celui qui correspond le vecteur normale N et sella pour l'axe X.
  La même chose pour l'axe Y seulement on va sommet avec q donc la formule deviens comme suit : ZN =Z(N-1) +q. Avec Z0=0.

```
znx=numpy.zeros((512, 612),numpy.float32)
zny=numpy.zeros((512, 612),numpy.float32)
res_p=numpy.reshape(res_p,(512,612))
res_q=numpy.reshape(res_q,(512,612))
res_p=mask*res_p
res_q=mask*res_q
for i in range(1,512):
        znx[i,:]=znx[i-1,:]+res_p[i,:]
for j in range(1,612):
        zny[:,j]=znx[:,j-1]+res_q[:,j]
```

- Après avoir eu les deux matrices de Zn pour l'axe X et Y pour afficher l'objet 3D il nous manque la matrice de Zn pour l'axe Z donc on a choisi la moyenne des deux matrices de l'axe X et Y comme matrice de l'axe Z après on afficher tout simplement l'objet 3D après avoir multiplié la matrice Z fois le Mask.

```
plt.figure()
x=numpy.array(znx)
y=numpy.array(zny)
x=x.reshape((512,612))
y=y.reshape((512,612))
ax = plt.axes(projection='3d')
x1=range(612)
y1=range (512)
x1,y1=numpy.meshgrid(x1,y1)
m=mask*(x+y)/2
ax.plot_surface(x1,y1,m)
plt.show()
```

- Voici le résultat de l'objet 3D

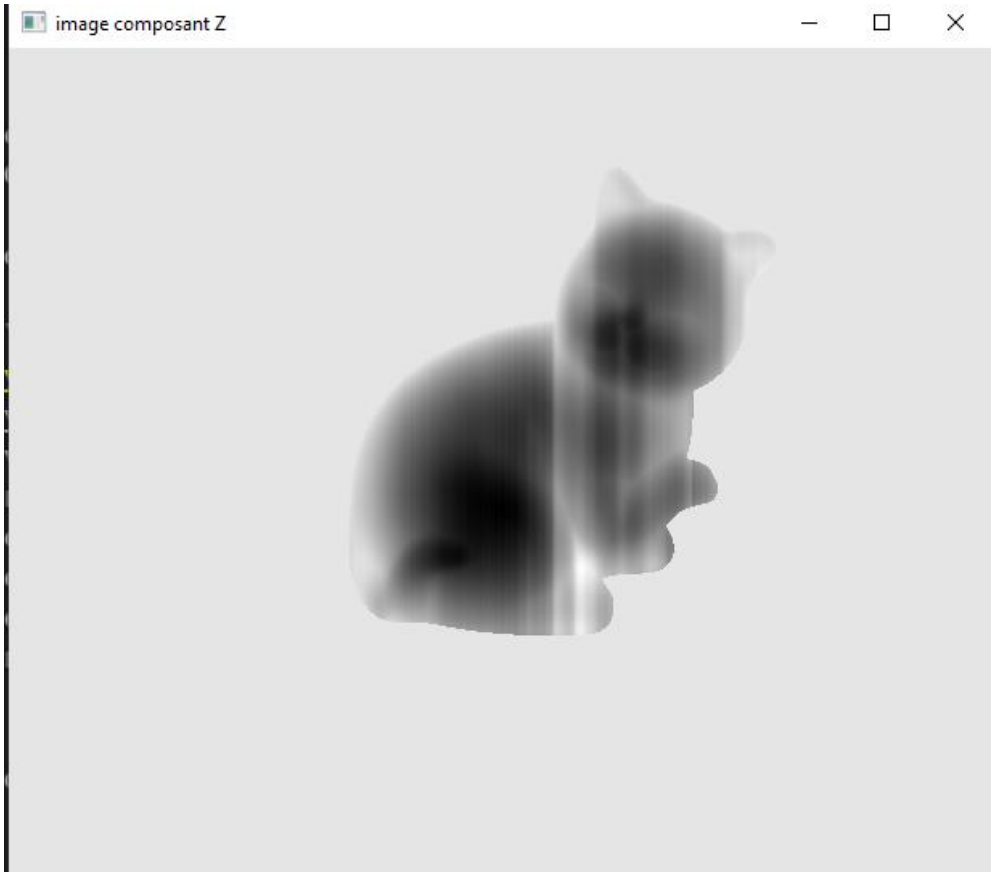


- On aussi afficher les composantes Z dans une images après l'avoir normalisé bien sûr.

```
#normalisation max min
max_ = m.max()
min_ = m.min()
m = 255*(m - min_) / (max_ - min_)

cv.imshow("image composant Z",m.astype("uint8"))
cv.waitKey(0)
```

Voilà le résultat de l'exécution.



- Pour afficher l'image d'erreur :

On récupère la matrice des vecteurs normales avec la fonction <calcul_needle_map()> et on récupère aussi la matrice de vecteurs normale réel du fichier normal.txt après on applique la formule d'erreur :

- **Calcul d'erreur d'angle PS : (projet partie 2)**
  - N_true = [x,y,z], N_est=[x,y,z] // deux vecteurs
  - dotM = dot(N_true, N_est);
  - angle = (180 .* acos(dotM)) ./ pi;

```
def bonus_calcule_erreur ():
    n_est=calcul_needle_map()
    n_true =load_normal_reel()
    dot_m = numpy.zeros((512*612),dtype=numpy.float32)

    for i in range(612*512):
        dot_m[i]=numpy.dot(n_true[:,i],n_est[:,i])
        dot_m[i]=180*math.acos(dot_m[i])/math.pi
    img_result= numpy.zeros((512, 612),numpy.uint8)
    k = 0
    for i in range(512):
        for j in range(612):
                #normalisation
            img_result[i,j]=((dot_m[k]+1)/2)*255
            k+=1
```

Voilà le résultat de l'exécution : les points noirs sont l'erreur de la matrice normale estimer par rapport au réel.