

Code final du projet

1. Correction du «bug audio»

Une erreur que beaucoup d'entre vous ont rencontré, c'est que pour certaines vidéos, le son est en mono alors que la fonction `extraction_volumes` était conçue pour un son en stéréo (pas bien, mauvaise conception).

L'explication du patch à appliquer pour corriger le bug en vidéo:

2. Le module

moduleprojet.py

```

1  #####
2  # Module contenant les fonctions nécessaires au traitement de l'image et du son#
3  #####
4
5  # Import des modules
6  import math
7  import numpy
8  import imageio
9  import scipy.io.wavfile as wave
10
11 # Partie 1 : image #####
12
13 # Conversions RGB <-> HSL
14
15 def hsl(pix: tuple) -> tuple:
16     '''
17     Prend en paramètre un tuple (r, g, b) correspond à une couleur codée en RGB,
18     avec:

```

```

19     r, g et b 3 entiers compris entre 0 et 255,
20     et renvoie sa conversion (h, s, l) en HSL, avec
21     h: la teinte, compris entre 0 et 360 degrés
22     s: la saturation, entier compris entre 0 et 100
23     l: la luminosité, entier compris entre 0 et 100
24     '''
25     r = pix[0]/255
26     g = pix[1]/255
27     b = pix[2]/255
28
29     M, m = max(r, g, b), min(r, g, b)
30     C = M - m
31     print(C)
32     if C == 0:
33         h = 0
34         print('h=0')
35     elif M == r:
36         h = ((g - b)/C) % 6
37     elif M == g:
38         h = ((b - r)/C + 2) % 6
39     else:
40         h = ((r - g)/C + 4) % 6
41     h = 60 * h
42     l = 0.5 * (M+m)
43     if l == 1 or l == 0:
44         s = 0
45     else:
46         s = C / (1 - abs(2*l-1))
47     return (int(h), int(100*s), int(100*l))
48
49 def rgb(pix: tuple) -> tuple:
50     '''
51     Prend en paramètre un tuple (h, s, l) correspond à une couleur codée en HSL,
52     avec:
53     h: la teinte, compris entre 0 et 360 degrés
54     s: la saturation, entier compris entre 0 et 100
55     l: la luminosité, entier compris entre 0 et 100
56     et renvoie sa conversion (h, s, l) en HSL, avec r, g et b 3 entiers compris
57     entre 0 et 255.
58     '''
59     h = pix[0]
60     s = pix[1]/100
61     l = pix[2]/100
62
63     C = (1 - abs(2*l-1)) * s
64     h = h / 60
65     X = C * (1-abs(h%2 - 1))
66     m = l - 0.5*C
67     if h == 0:
68         r, g, b = 0, 0, 0
69     elif h < 1:
70         r, g, b = C, X, 0
71     elif h < 2:
72         r, g, b = X, C, 0
73     elif h < 3:
74         r, g, b = 0, C, X
75     elif h < 4:
76         r, g, b = 0, X, C
77     elif h < 5:
78         r, g, b = X, 0, C
79     else:

```

```

80     r, g, b = C, 0, X
81     return (int(255* (r+m)), int(255* (g+m)), int(255* (b+m)))
82
83 # Extraction de la couleur dominante
84
85 def extraction_couleur(im: list) -> tuple:
86     '''
87     Renvoie le pixel majoritaire dans une image im.
88     On crée un dictionnaire dont les clés sont les pixels et les valeurs
89     leur nombre d'occurences dans im, puis on parcourt le dictionnaire
90     pour rechercher et le pixel majoritaire (en RGB)
91     '''
92     couleurs = {}
93     for i in range(im.shape[0]):
94         for j in range(im.shape[1]):
95             c = tuple(im[i][j])
96             if c in couleurs:
97                 couleurs[c] += 1
98             else:
99                 couleurs[c] = 1
100     c_max = 0
101     for c, n in couleurs.items():
102         if n > c_max:
103             c_max = n
104             dominante = c
105     return dominante
106
107 # Extraction de la zone de l'image
108
109 def extraction_zone(im: list, sat: int, V_son: float, V_video: float) -> list:
110     w = max(10, int(500 * sat / 100))
111     x = int((500-w) * V_son)
112     y = int((500-w) * V_video)
113     zone = numpy.zeros((w, w, im.shape[2]), dtype=numpy.uint8)
114     for i in range(w):
115         for j in range(w):
116             zone[i][j] = im[y+i][x+j]
117     return zone
118
119 # Application du filtre
120
121 def f(a: int, b: int) -> int:
122     '''
123     Fonction d'overlay à appliquer sur chaque composante d'un pixel.
124     a est la composante de l'image de base.
125     b est la composante du filtre à appliquer.
126     '''
127     a = a/255
128     b = b/255
129     if a < 0.5:
130         return int(255*2*a*b)
131     else:
132         return int(255*(1-2*(1-a)*(1-b)))
133
134 def filtre(img: list, couleur: tuple) -> None:
135     '''
136     Crée et renvoie une image composée du filtre de couleur couleur appliqué
137     à l'image img.
138     '''
139     img_filtree = numpy.zeros((img.shape[0], img.shape[1], 3), dtype=numpy.uint8)
140     for i in range(img.shape[0]):

```

```

141         for j in range(img.shape[1]):
142             for c in range(3):
143                 img_filtree[i][j][c] = f(img[i][j][c], couleur[c])
144         return img_filtree
145
146 # Partie 2 : son #####
147
148 # Récupération du spectre
149
150 def spectre(data: list, rate: int, debut: float, duree: float) -> list:
151     '''
152     Renvoie le spectre correspondant à un intervalle du signal.
153
154     data: le signal d'un canal
155     rate: la fréquence d'échantillonnage
156     debut: le début de l'intervalle à étudier (en secondes)
157     duree: la durée de l'intervalle à étudier (en secondes)
158     '''
159     start = int(debut * rate)
160     stop = int((debut+duree) * rate)
161     s = numpy.absolute(numpy.fft.fft(data[start:stop]))
162     s = s / s.max()
163     return [math.log10(i) for i in s if i != 0]
164
165 def extraction_volumes(filename: str) -> list:
166     '''
167     Découpe un fichier son en 25 intervalles, récupère le volume minimal (en dbA)
168     du spectre de chaque intervalle et renvoie ces volumes en pourcentage de la
169     plage [vmin, vmax].
170     '''
171     rate, echantillon = wave.read(filename)
172     if type(echantillon[0]) is numpy.ndarray:
173         data = [e[0] for e in echantillon] # on choisit un seul canal, ici le gauche
174     else:
175         data = echantillon
176     duree = len(data) / rate
177     volumes = []
178     for k in range(25):
179         sp = spectre(data, rate, k*duree/25, duree/25)
180         volumes.append(min(sp))
181     vmin, vmax = min(volumes), max(volumes)
182     pourcentages = [(v-vmin) / (vmax-vmin) for v in volumes]
183     return pourcentages

```

3. Le programme principal

main.py

```

1 import moduleprojet as mp
2 import imageio
3 import os
4
5 # Chargement des fichiers de données
6 video = 'videos/video_gr4_st2.mp4'
7 son = 'sons/son_gr4_st2.wav'
8 image = imageio.imread('images/image_gr4_el1.jpg')
9 audio = 'audio.wav'
10

```

```

11 # Extraction du son de la video
12 os.system('rm audio.wav') # sinon ffmpeg plante...
13 os.system('ffmpeg -i ' + video + ' ' + audio)
14
15 # Extraction des volumes du son et de l'audio
16 v_son = mp.extraction_volumes(son)
17 v_video = mp.extraction_volumes(audio)
18
19 # Création du reader de la vidéo
20 reader = imageio.get_reader(video)
21
22 # Boucle principale
23 for k in range(25):
24     # extraction de l'image de la video
25     frame = reader.get_data(k * reader.count_frames() // 25)
26
27     # extraction de la couleur dominante, convertie en hsl
28     dominante = mp.hsl(mp.extraction_couleur(frame))
29
30     # extraction de la zone
31     zone = mp.extraction_zone(image, dominante[1], v_son[k], v_video[k])
32
33     # application du filtre
34     couleur_filtre = mp.rgb((dominante[0], 100, dominante[2]))
35     zone_filtree = mp.filtre(zone, couleur_filtre)
36
37     # enregistrement de l'image
38     imageio.imsave(f'images/image_gif{k}.png', zone_filtree)
39
40     # redimensionnement des images
41     os.system(f'mogrify -resize 500x images/image_gif{k}.png')
42
43 # Création du GIF
44 with imageio.get_writer('GIF_ultime.gif', mode='I') as writer:
45     for k in range(25):
46         image = imageio.imread(f'images/image_gif{k}.png')
47         writer.append_data(image)

```