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:
```

```
    r, g et b 3 entiers compris entre 0 et 255,
    et renvoie sa conversion (h, s, l) en HSL, avec
    h: la teinte, compris entre 0 et 360 degrés
    s: la saturation, entier compris entre 0 et 100
    l: la luminosité, entier compris entre 0 et 100
    '''
    r = pix[0]/255
    g = pix[1]/255
    b = pix[2]/255

    M, m = max(r, g, b), min(r, g, b)
    C = M - m
    print(C)
    if C == 0:
        h = 0
        print('h=0')
    elif M == r:
        h = ((g - b)/C ) % 6
    elif M == g:
        h = ((b - r)/C + 2) % 6
    else:
        h = ((r - g)/C + 4) % 6
    h = 60 * h
    l = 0.5 * (M+m)
    if l == 1 or l == 0:
        s = 0
    else:
        s = C / (1- abs(2*l-1))
    return (int(h), int(100*s), int(100*l))

def rgb(pix: tuple) -> tuple:
    '''
    Prend en paramètre un tuple (h, s, l) correspond à une couleur codée en HSL,
    avec:
    h: la teinte, compris entre 0 et 360 degrés
    s: la saturation, entier compris entre 0 et 100
    l: la luminosité, entier compris entre 0 et 100
    et renvoie sa conversion (h, s, l) en HSL, avec r, g et b 3 entiers compris
    entre 0 et 255.
    '''
    h = pix[0]
    s = pix[1]/100
    l = pix[2]/100

    C = (1 - abs(2*l-1)) * s
    h = h / 60
    X = C * (1-abs(h%2 - 1))
    m = l - 0.5*C
    if h == 0:
        r, g, b = 0, 0, 0
    elif h < 1:
        r, g, b = C, X, 0
    elif h < 2:
        r, g, b = X, C, 0
    elif h < 3:
        r, g, b = 0, C, X
    elif h < 4:
        r, g, b = 0, X, C
    elif h < 5:
        r, g, b = X, 0, C
    else:
```

```
        r, g, b = C, 0, X
    return (int(255* (r+m)), int(255* (g+m)), int(255* (b+m)))

# Extraction de la couleur dominante

def extraction_couleur(im: list) -> tuple:
    '''
    Renvoie le pixel majoritaire dans une image im.
    On crée un dictionnaire dont les clés sont les pixels et les valeurs
    leur nombre d'occurences dans im, puis on parcourt le dictionnaire
    pour rechercher et le pixel majoritaire (en RGB)
    '''
    couleurs = {}
    for i in range(im.shape[0]):
        for j in range(im.shape[1]):
            c = tuple(im[i][j])
            if c in couleurs:
                couleurs[c] += 1
            else:
                couleurs[c] = 1
    c_max = 0
    for c, n in couleurs.items():
        if n > c_max:
            c_max = n
            dominante = c
    return dominante

# Extraction de la zone de l'image

def extraction_zone(im: list, sat: int, V_son: float, V_video: float) -> list:
    w = max(10, int(500 * sat / 100))
    x = int((500-w) * V_son)
    y = int((500-w) * V_video)
    zone = numpy.zeros((w, w, im.shape[2]), dtype=numpy.uint8)
    for i in range(w):
        for j in range(w):
            zone[i][j] = im[y+i][x+j]
    return zone

# Application du filtre

def f(a: int, b: int) -> int:
    '''
    Fonction d'overlay à appliquer sur chaque composante d'un pixel.
    a est la composante de l'image de base.
    b est la composante du filtre à appliquer.
    '''
    a = a/255
    b = b/255
    if a < 0.5:
        return int(255*2*a*b)
    else:
        return int(255*(1-2*(1-a)*(1-b)))

def filtre(img: list, couleur: tuple) -> None:
    '''
    Crée et renvoie une image composée du filtre de couleur couleur appliqué
    à l'image img.
    '''
    img_filtree = numpy.zeros((img.shape[0], img.shape[1], 3), dtype=numpy.uint8)
    for i in range(img.shape[0]):
```

```
        for j in range(img.shape[1]):
            for c in range(3):
                img_filtree[i][j][c] = f(img[i][j][c], couleur[c])
    return img_filtree

# Partie 2 : son ######################################################################

# Récupération du spectre

def spectre(data: list, rate: int, debut: float, duree: float) -> list:
    '''
    Renvoie le spectre correspondant à un intervalle du signal.

    data: le signal d'un canal
    rate: la fréquence d'échantillonnage
    debut: le début de l'intervalle à étudier (en secondes)
    duree: la durée de l'intervalle à étudier (en secondes)
    '''
    start = int(debut * rate)
    stop = int((debut+duree) * rate)
    s = numpy.absolute(numpy.fft.fft(data[start:stop]))
    s = s / s.max()
    return [math.log10(i) for i in s if i != 0]

def extraction_volumes(filename: str) -> list:
    '''
    Découpe un fichier son en 25 intervalles, récupère le volume minimal (en dbA)
    du spectre de chaque intervalle et renvoie ces volumes en pourcentage de la
    plage [vmin, vmax].
    '''
    rate, echantillon = wave.read(filename)
    if type(echantillon[0]) is numpy.ndarray:
        data = [e[0] for e in echantillon] # on choisit un seul canal, ici le gauche
    else:
        data = echantillon
    duree = len(data) / rate
    volumes = []
    for k in range(25):
        sp = spectre(data, rate, k*duree/25, duree/25)
        volumes.append(min(sp))
    vmin, vmax = min(volumes), max(volumes)
    pourcentages = [(v-vmin) / (vmax-vmin) for v in volumes]
    return pourcentages
```

3. Le programme principal

main.py

```
import moduleprojet as mp
import imageio
import os

# Chargement des fichiers de données
video = 'videos/video_gr4_st2.mp4'
son = 'sons/son_gr4_st2.wav'
image = imageio.imread('images/image_gr4_el1.jpg')
audio = 'audio.wav'

```

```python
# Extraction du son de la video
os.system('rm audio.wav') # sinon ffmpeg plante...
os.system('ffmpeg -i ' + video + ' ' + audio)

# Extraction des volumes du son et de l'audio
v_son = mp.extraction_volumes(son)
v_video = mp.extraction_volumes(audio)

# Création du reader de la vidéo
reader = imageio.get_reader(video)

# Boucle principale
for k in range(25):
    # extraction de l'image de la video
    frame = reader.get_data(k * reader.count_frames() // 25)

    # extraction de la couleur dominante, convertie en hsl
    dominante = mp.hsl(mp.extraction_couleur(frame))

    # extraction de la zone
    zone = mp.extraction_zone(image, dominante[1], v_son[k], v_video[k])

    # application du filtre
    couleur_filtre = mp.rgb((dominante[0], 100, dominante[2]))
    zone_filtree = mp.filtre(zone, couleur_filtre)

    # enregistrement de l'image
    imageio.imsave(f'images/image_gif{k}.png', zone_filtree)

    # redimensionnement des images
    os.system(f'mogrify -resize 500x images/image_gif{k}.png')

# Création du GIF
with imageio.get_writer('GIF_ultime.gif', mode='I') as writer:
    for k in range(25):
        image = imageio.imread(f'images/image_gif{k}.png')
        writer.append_data(image)
```