#!/usr/bin/env python3

import os

import numpy as np

from moviepy.editor import ImageClip, CompositeVideoClip, vfx

from PIL import Image

import cv2

# Calea imaginii

IMAGE\_PATH = r"d:\family-hugging.jpg"

OUTPUT\_PATH = r"d:\family-hugging-animation.mp4"

# Durata animației

DURATION = 10 # 10 secunde

def apply\_ken\_burns\_effect(clip, duration, zoom\_factor=1.2):

"""Aplică efectul Ken Burns: zoom și panoramare simulată"""

print("Aplic efect Ken Burns...")

width, height = clip.size

original\_w, original\_h = clip.size

def make\_frame(t):

# Zoom progresiv

scale = 1 + (zoom\_factor - 1) \* (t / duration)

new\_w, new\_h = int(original\_w \* scale), int(original\_h \* scale)

# Calculăm poziția pentru panoramare (stânga-dreapta)

max\_shift = (new\_w - original\_w) / 2

x\_shift = max\_shift \* np.sin((t / duration) \* 2 \* np.pi) # Mișcare sinusoidală

x\_center = (new\_w - original\_w) / 2 - x\_shift

y\_center = (new\_h - original\_h) / 2

# Redimensionăm și decupăm imaginea pentru a simula zoom și panoramare

resized = clip.resize((new\_w, new\_h))

cropped = resized.crop(

x\_center=x\_center,

y\_center=y\_center,

width=original\_w,

height=original\_h

)

return cropped.get\_frame(t)

# Creăm un clip nou cu cadrele generate

return clip.set\_duration(duration).set\_make\_frame(make\_frame)

def create\_light\_flare\_effect(width, height, duration):

"""Creează un efect de lumină care se mișcă peste imagine"""

print("Creez efect de lumină...")

def flare\_frame(t):

frame = np.zeros((height, width, 3), dtype=np.uint8)

flare\_center\_x = int(width \* (t / duration))

flare\_center\_y = height // 2 + int(100 \* np.sin((t \* 2 \* np.pi) / duration))

for y in range(height):

for x in range(width):

distance = np.sqrt((x - flare\_center\_x)\*\*2 + (y - flare\_center\_y)\*\*2)

if distance < 150:

intensity = 255 \* (1 - distance / 150)

frame[y, x] = [int(intensity), int(intensity), 100]

return frame

flare\_clip = ImageClip(flare\_frame(0), duration=duration)

flare\_clip = flare\_clip.set\_make\_frame(lambda t: flare\_frame(t))

return flare\_clip.set\_opacity(0.3)

def create\_particles\_effect(width, height, duration, num\_particles=50):

"""Creează particule animate (scântei sau petale)"""

print("Creez efect de particule...")

particles = []

for \_ in range(num\_particles):

x = np.random.randint(0, width)

y = np.random.randint(0, height)

particle\_duration = np.random.uniform(2, duration)

speed = np.random.uniform(30, 70)

# Captăm valorile în closure

x\_start, y\_start = x, y

p\_duration, p\_speed = particle\_duration, speed

def make\_particle\_frame(t, x=x\_start, y=y\_start,

particle\_duration=p\_duration, speed=p\_speed):

frame = np.zeros((height, width, 3), dtype=np.uint8)

if t < particle\_duration:

x\_pos = x + 50 \* np.sin((t \* 2 \* np.pi) / particle\_duration)

y\_pos = y - t \* speed

if 0 <= y\_pos < height and 0 <= x\_pos < width:

cv2.circle(frame, (int(x\_pos), int(y\_pos)), 2, (255, 255, 200), -1)

return frame

particle = ImageClip(make\_particle\_frame(0), duration=duration)

particle = particle.set\_make\_frame(lambda t: make\_particle\_frame(t))

particles.append(particle.set\_opacity(0.7))

return particles

def create\_color\_shift\_effect(clip, duration):

"""Aplică o tranziție subtilă de luminozitate"""

print("Aplic efect de tranziție de luminozitate...")

def make\_frame(t):

# Obținem cadrul curent

frame = clip.get\_frame(t)

# Calculăm factorul de luminozitate

factor = 0.8 + 0.2 \* np.sin((t / duration) \* 2 \* np.pi) # Variază între 0.8 și 1.0

# Aplicăm factorul direct și asigurăm că valorile rămân între 0-255

return np.clip(frame \* factor, 0, 255).astype('uint8')

# Creăm un clip nou cu funcția de transformare

return clip.fl\_image(lambda img: make\_frame(clip.to\_seconds(img)))

def create\_color\_tone\_effect(clip, duration):

"""Aplică o tranziție de tonuri de culoare (cald-rece)"""

print("Aplic efect de tranziție de tonuri de culoare...")

def tone\_transform(get\_frame, t):

frame = get\_frame(t)

factor = np.sin((t / duration) \* 2 \* np.pi)

# Ajustăm canalele RGB pentru a simula tranziția cald-rece

r\_factor = 1 + 0.2 \* factor # Accentuăm roșul pentru tonuri calde

b\_factor = 1 - 0.2 \* factor # Accentuăm albastrul pentru tonuri reci

new\_frame = frame.copy()

new\_frame[:, :, 0] = np.clip(frame[:, :, 0] \* r\_factor, 0, 255) # Canalul R

new\_frame[:, :, 2] = np.clip(frame[:, :, 2] \* b\_factor, 0, 255) # Canalul B

return new\_frame

return clip.fl(tone\_transform)

def main():

print(f"Încarc imaginea: {IMAGE\_PATH}")

try:

image = Image.open(IMAGE\_PATH)

width, height = image.size

print(f"Dimensiune imagine: {width}x{height}")

base\_clip = ImageClip(np.array(image), duration=DURATION)

print("Imaginea a fost convertită în clip video")

# Aplicăm efectul Ken Burns

ken\_burns\_clip = apply\_ken\_burns\_effect(base\_clip, DURATION)

# Creăm efectul de lumină

flare\_clip = create\_light\_flare\_effect(width, height, DURATION)

# Creăm particulele

particle\_clips = create\_particles\_effect(width, height, DURATION)

# Aplicăm efectul de luminozitate

enhanced\_clip = create\_color\_shift\_effect(ken\_burns\_clip, DURATION)

# Aplicăm efectul de tonuri de culoare

final\_base\_clip = create\_color\_tone\_effect(enhanced\_clip, DURATION)

print("Combin efectele...")

all\_clips = [final\_base\_clip, flare\_clip] + particle\_clips

final\_video = CompositeVideoClip(all\_clips)

print(f"Salvez animația: {OUTPUT\_PATH}")

final\_video.write\_videofile(

OUTPUT\_PATH,

fps=24,

codec='libx264',

audio\_codec='aac',

bitrate="2000k",

ffmpeg\_params=["-crf", "23"],

threads=4

)

print("\n✅ Animația a fost creată cu succes!")

except Exception as e:

print(f"❌ Eroare: {str(e)}")

import traceback

traceback.print\_exc()

if \_\_name\_\_ == "\_\_main\_\_":

main()