

プログラミング演習3 (Python) ⑪ 動画処理

1. 【AI】MP4 ファイルを生成する

ここでは、Gemini と相談して、イラストに効果を付けた動画ファイルを生成するファイルを作ってもらおう。

自分で用意したイラストを回転させた動画を WMV ファイルとして書き出すプログラムを Python で作って。

とすると、WMV は直接サポートされていないとし、MP4 で作成するプログラムを作成した。

今回は

Pillow (画像の回転) と**MoviePy** (動画作成)、NumPy (高速な数値計算や多次元配列の操作) を使用して、回転動画を MP4 ファイルとして作成した。

事前にライブラリのインストールをする。

```
pip install Pillow moviepy numpy
```

`py11-1.py` MP4 ファイルの生成 (静止画を回転させた動画を作る)

```
001 import os
002 from PIL import Image
003 from moviepy.editor import ImageSequenceClip
004
005 def create_rotating_video(input_image_path, output_video_path, duration_seconds,
006                           fps=30, total_rotation_degrees=360):
007     """
008     指定された画像を回転させながら動画を生成する関数。
009
010     Args:
011         input_image_path (str): 入力画像ファイルへのパス。
012         output_video_path (str): 出力動画ファイルへのパス (例: output.mp4)。
013         duration_seconds (int): 動画の長さ (秒)。
014         fps (int): フレームレート (Frames Per Second)。
015         total_rotation_degrees (int): 動画全体で回転させる合計角度 (度)。
016
017     """
018     try:
019         # 1. 画像を開く
020         original_image = Image.open(input_image_path)
```

```

019     except FileNotFoundError:
020         print(f"エラー: 画像ファイルが見つかりません - {input_image_path}")
021         return
022     except Exception as e:
023         print(f"エラー: 画像の読み込み中にエラーが発生しました - {e}")
024         return
025
026     # 2. 全フレーム数と各フレームの回転角度を計算
027     total_frames = duration_seconds * fps
028     # 1 フレームあたりの回転角度（反時計回り）
029     degree_per_frame = total_rotation_degrees / total_frames
030
031     print(f"動画の生成を開始します: {total_frames} フレーム,
032           {degree_per_frame:.2f}度/フレーム")
033
034     # 3. フレームのリストを作成
035     frames = []
036     for i in range(total_frames):
037         # 現在の回転角度（Pillowのrotateは反時計回りを正とする）
038         current_angle = i * degree_per_frame
039
040         # 画像を回転（expand=Trueで回転後の全体を保持、fillcolor='white'で余白を
041         # expand=Falseにすると、回転によって画像がクロップされます
042         # fillcolorは背景の色を設定します（元の画像が透過でない場合、影響がない
043         # ことがあります）
044         rotated_image = original_image.rotate(
045             angle=current_angle,
046             resample=Image.Resampling.BICUBIC, # 高品質な補間
047             expand=True,
048             fillcolor=(255, 255, 255) # 白（RGB）
049         )
050
051         # MoviePyが処理できる形式（PIL Imageオブジェクト）でリストに追加
052         frames.append(rotated_image)
053
054     # 4. フレームから動画クリップを作成
055     # PIL ImageのリストをImageSequenceClipに渡す

```

```

054 clip = ImageSequenceClip(frames, fps=fps)
055
056 # 5. 動画ファイルを書き出す (MP4 形式)
057 print(f"動画を {output_video_path} として書き出しています...")
058 # 'libx264' は一般的な MP4 コーデックです
059 clip.write_videofile(
060     output_video_path,
061     codec='libx264',
062     audio_codec='aac',
063     fps=fps
064 )
065 print("動画の生成が完了しました!")
066
067 # --- 設定 ---
068 # 以下の値をあなたの環境に合わせて変更してください
069
070 # 1. 入力画像ファイル名
071 INPUT_IMAGE = "py11-1.png" # 例: 自分で用意したイラストのファイル名
072
073 # 2. 出力動画ファイル名 (一旦 MP4 で作成)
074 OUTPUT_VIDEO_MP4 = "rotating_video.mp4"
075
076 # 3. 動画の設定
077 VIDEO_DURATION_SECONDS = 5 # 動画の長さ (秒)
078 VIDEO_FPS = 30 # フレームレート
079 ROTATION_DEGREES = 360 * 2 # 合計回転角度 (360 * 2 で 2 回転)
080
081 # --- 実行 ---
082 create_rotating_video(
083     input_image_path=INPUT_IMAGE,
084     output_video_path=OUTPUT_VIDEO_MP4,
085     duration_seconds=VIDEO_DURATION_SECONDS,
086     fps=VIDEO_FPS,
087     total_rotation_degrees=ROTATION_DEGREES
088 )

```

```
管理: コマンドプロンプト
C:\Users\teacher\Desktop\py>py py11-1.py
動画の生成を開始します: 150フレーム, 4.80度/フレーム
動画を rotating_video.mp4 として書き出しています...
Moviepy - Building video rotating_video.mp4.
Moviepy - Writing video rotating_video.mp4

Moviepy - Done !
Moviepy - video ready rotating_video.mp4
動画の生成が完了しました！
C:\Users\teacher\Desktop\py>
```



【動画 URL】

py11-1 rotating video <https://youtube.com/shorts/bmz4r28zb94>

次は、画像にランダムに歪みの効果を入れる動画を作るためのプログラムを作るように指示をした。

うまくできた。次は、画像を回転させずに制止させて、画像は500×500ピクセルなんだけど、その範囲でランダムに座標をとって、半径50くらいの円の範囲が0.5秒間、波を打つような効果をつけて、それが終わったら、また違う場所が波打つような5秒くらいの動画を作りたい。

Gemini の回答

動画の生成成功、おめでとうございます！

次に、画像を静止させつつ、ランダムな座標に波紋（波を打つような効果）を時間差でつける動画を作成したいのですね。これは画像処理と動画編集のテクニックとしては少し複雑になりますが、moviepy と numpy、そして画像効果を適用するための数学的な処理を組み合わせることで実現可能です。

ご提示いただいた要件を実現するための基本的なアプローチと、そのための修正案を提示します。

🦋 アプローチの概要

1. **静止画の準備:** 元の画像を 500 × 500 ピクセルにリサイズまたはクロップします。
2. **時間の定義:** 5 秒間の動画全体をフレームレート(例: 30 FPS)で定義します。
3. **波紋のランダム生成:**
 - 動画の長さ(5 秒)の間に、波紋が発生する**時間と中心座標 (x, y)** をランダムに決定します。
 - 各波紋は 0.5 秒間持続します。
4. **フレーム生成:** 各時間 t において、現在アクティブな波紋(0.5 秒以内に始まったもの)をすべて特定し、それらの効果を合成して出力フレームを生成します。

🔧 実現のためのコード修正案

ここでは、moviepy のカスタム効果を適用する最も簡単な方法として、ImageClip と set_make_frame メソッドを使用します。波紋効果自体は、画像をピクセル単位でずらす****ディストーション(歪み)****として実装します。1. 必要なライブラリのインストール

まだインストールしていない場合は、これらが必要です。

Bash

```
pip install numpy Pillow moviepy
```

2. 波紋動画生成関数

このコードでは、wave_effect_frame 関数が時間 t と波紋の中心、半径に基づいて画像を歪ませる役割を果たします。

📌 重要な注意点

1. **計算負荷:** この種のピクセルディストーション処理は、フレームごとに $500 \times 500 = 25$ 万ピクセルに対して波紋効果の計算(平方根、三角関数、配列操作など)を行うため、**非常に計算負荷が高い**です。動画の書き出しには時間がかかることをご承知おきください。
2. **WAVE_EVENTS のランダム性:** 上記コードでは 0.25 秒間隔で波紋を発生させていますが、完全にランダムな時間にしたい場合は、`np.random.uniform(0, duration)` を使って開始時間を生成し、WAVE_EVENTS の数を調整してください。
3. **波紋の見た目:** WAVE_AMPLITUDE や spatial_decay の計算式を変更することで、波紋の広がり方(速さ、強さ)を調整できます。上記のコードは、中心から外側へ広がるようなディストーション効果を簡易的にシミュレートしています。

この修正案で、ご希望の「ランダムな座標で波を打つような効果」を持つ動画が作成できるはずです。お試しください。

生成されたプログラムでは、効果が小さかったので、修正を要求した後のコード

`py7-2.py` ランダムに半径 100 ピクセルの円の範囲で歪ませる

```
001 import os
002 import numpy as np
003 from PIL import Image
004 from moviepy.editor import ImageClip
005 # MoviePy のバージョンエラー対策のため、音声関連のインポートは削除したまま
006
007 # --- 設定 ---
008 INPUT_IMAGE = "py11-1.png"
009 OUTPUT_VIDEO_MP4 = "wavy_effect_video.mp4"
010 VIDEO_DURATION_SECONDS = 5
011 VIDEO_FPS = 30
012
013 # 画像と動画のサイズ（要件に合わせる）
014 W, H = 500, 500
015
016 # ★ 修正点 1: 波紋の半径を 100 ピクセルに拡大
017 WAVE_RADIUS = 100
018 WAVE_DURATION = 0.5 # 0.5 秒間波紋を発生させる
019 # ★ 修正点 2: 波紋の強さを 15 ピクセルに調整
020 WAVE_AMPLITUDE = 15 # ピクセル単位での最大の歪みの強さ
021
022 # --- ランダムな波紋の生成 ---
023 # イベント頻度を増やし、より多くの波紋が重なるように調整
024 num_waves = int(VIDEO_DURATION_SECONDS * 4) # 4 倍に増やして、5 秒間に約 20 個のイベントを発生
025 WAVE_EVENTS = []
026
027 for i in range(num_waves):
028     # 0.25 秒ごと（1/4 秒ごと）に新しい波紋イベントを開始
029     start_time = i * 0.25
030     if start_time >= VIDEO_DURATION_SECONDS:
031         break
```

```

032
033     # 波紋の中心座標 (x, y) をランダムに決定
034     # 中心が画像の端から WAVE_RADIUS 以上離れていることを保証
035     center_x = np.random.randint(WAVE_RADIUS, W - WAVE_RADIUS)
036     center_y = np.random.randint(WAVE_RADIUS, H - WAVE_RADIUS)
037
038     WAVE_EVENTS.append({
039         'start_time': start_time,
040         'center': (center_x, center_y)
041     })
042     print(f"生成された波紋イベント数: {len(WAVE_EVENTS)}")
043
044
045     def create_wave_video(input_image_path, output_video_path, duration, fps):
046
047         try:
048             # 1. 元画像を開き、ターゲットサイズにリサイズ
049             original_image_pil = Image.open(input_image_path).convert('RGB')
050             original_image_pil = original_image_pil.resize((W, H))
051             original_image_np = np.array(original_image_pil)
052         except FileNotFoundError:
053             print(f"エラー: 画像ファイルが見つかりません - {input_image_path}")
054             return
055         except Exception as e:
056             print(f"エラー: 画像の読み込み中にエラーが発生しました - {e}")
057             return
058
059         # 2. ピクセル座標のメッシュグリッドを事前に作成
060         xx, yy = np.meshgrid(np.arange(W), np.arange(H))
061
062         def wave_effect_frame(t):
063             """
064             時間 t におけるフレームを生成する関数。
065             """
066             dx = np.zeros((H, W), dtype=np.float32)
067             dy = np.zeros((H, W), dtype=np.float32)
068
069             # 3. アクティブな波紋効果を計算し、歪み (dx, dy) を累積する

```

```

070         for event in WAVE_EVENTS:
071             t_start = event['start_time']
072             cx, cy = event['center']
073
074             # 波紋が現在アクティブかチェック (t_start <= t < t_start +
075 WAVE_DURATION)
076             if t_start <= t < t_start + WAVE_DURATION:
077
078                 t_wave = t - t_start
079                 R = np.sqrt((xx - cx)**2 + (yy - cy)**2)
080
081                 # 距離 R を WAVE_RADIUS (100) に正規化
082                 R_norm = R / WAVE_RADIUS
083
084                 # 効果の減衰: 距離が WAVE_RADIUS を超えるピクセルには効果なし
085                 mask = R_norm < 1.0
086
087                 # 時間による波紋の強さ (0.5 秒かけて現れて消える)
088                 t_wave_norm = t_wave / WAVE_DURATION
089                 strength_time = WAVE_AMPLITUDE * np.sin(np.pi * t_wave_norm)
090
091                 # 空間的な減衰 (中心で強く、端で弱く)
092                 spatial_decay = np.cos(R_norm * np.pi / 2)
093
094                 total_distortion = strength_time * spatial_decay * mask
095
096                 # 歪みの方向 (中心からの単位ベクトル)
097                 angle = np.arctan2(yy - cy, xx - cx)
098
099                 # x 方向と y 方向の歪みを累積
100                 dx += total_distortion * np.cos(angle)
101                 dy += total_distortion * np.sin(angle)
102
103             # 4. 参照座標を計算 (元のピクセル位置 + 歪み)
104             x_ref = np.clip(xx + dx, 0, W - 1).astype(int)
105             y_ref = np.clip(yy + dy, 0, H - 1).astype(int)
106
107             # 5. 歪んだ座標からピクセル値をサンプリング

```



```

108         new_frame = original_image_np[y_ref, x_ref]
109
110         return new_frame
101
102     # 6. MoviePy クリップの作成
103     clip = ImageClip(original_image_np, duration=duration).set_fps(fps)
104     clip = clip.set_make_frame(wave_effect_frame)
105
106     # 7. 動画ファイルを書き出す
107     print(f"動画を {output_video_path} として書き出しています... ({W}x{H}px,
108 {duration}s)")
109
110     clip.write_videofile(
101         output_video_path,
102         codec='libx264',
103         audio_codec='aac',
104         fps=fps,
105         preset='medium'
106     )
107     print("動画の生成が完了しました！")
108
109 # --- 実行 ---
110 create_wave_video(
101     input_image_path=INPUT_IMAGE,
102     output_video_path=OUTPUT_VIDEO_MP4,
103     duration=VIDEO_DURATION_SECONDS,
104     fps=VIDEO_FPS
105 )

```

【実行結果】



【動画】

py11-2 wavy effect video <https://youtube.com/shorts/-wxMauxBw5s>

音に合わせてゆがむようにする。

まず、挿入する音楽を作成する。

Gemini が「Suno」や「Stable Audio」を紹介してくれたので、学校の環境でアカウントがとれた「Stable Audio」を使って作る。ちなみに、Suno はドラマチックな展開が得意で、「Stable Audio」は精密な音響・質感が得意だそう。

Stable Audio で作る(精密な音響・質感が得意)

こちらはより「音の質感」にこだわった、プロっぽいシネマティックな音を作るのが得意です。

- URL: stableaudio.com
- 始め方: Google アカウント等でサインイン。
- 手順:
 1. プロンプト入力欄に以下の文章を貼り付けます。
 2. [Duration](長さ)を 10 秒～20 秒程度に設定。
 3. [Generate] をクリック！

Stable Audio 用プロンプト: *Experimental Industrial noise, heavy distorted bass drops, metallic screeching, robotic glitching sounds, cinematic impact, dark futuristic city atmosphere, high definition, 44.1kHz.*

の指示通り、20秒間の音楽を作り、py11-3.mp3 という名前にする。

画像をゆがませるタイミングを一定の時間間隔ではなく、プログラムと一緒ににおいてある MP3 ファイル (py11-3.mp3) の音が鳴るタイミングになるように修正してほしい。で、生成された動画にはそのMP3 ファイルの音が演奏されるようにしてほしい。

とすると、

音のタイミングに合わせて映像を動かすのは、非常に面白い試みですね！

これを実現するには、音声の振幅(ボリューム)を解析し、一定以上の音量がある瞬間に波紋を発生させるロジックが必要です。

実装のポイント

1. 音声解析: moviepy の AudioFileClip を使い、指定した時間(fps ごと)の音量を数値化します。

2. **トリガー判定:** 音量が設定したしきい値(Threshold)を超えた瞬間に、新しい波紋の中心座標を生成します。
3. **音声の統合:** set_audio を使い、完成した動画に元の MP3 を合成します。

● 数十回やり取りをしてプログラムが完成する

py7-3.py 音に合わせて歪ませる

```
001 import numpy as np
002 from PIL import Image
003 import sys
004
005 # ライブラリの読み込み
006 try:
007     from moviepy.editor import VideoClip, AudioFileClip
008 except ImportError:
009     try:
010         from moviepy.video.VideoClip import VideoClip
011         from moviepy.audio.io.AudioFileClip import AudioFileClip
012     except ImportError:
013         from moviepy import VideoClip, AudioFileClip
014
015 # --- 設定 ---
016 INPUT_IMAGE = "py11-3.png"
017 INPUT_AUDIO = "py11-3.mp3"
018 OUTPUT_VIDEO_MP4 = "audio_reactive_wave.mp4"
019 VIDEO_FPS = 30
020
021 W, H = 500, 500
022 WAVE_RADIUS = 250 # 波紋の影響範囲
023 WAVE_DURATION = 0.8 # 1つの歪みが持続する時間(秒)
024
025 def generate_audio_reactive_events(audio_path, fps):
026     audio = AudioFileClip(audio_path)
027     print("音声を解析中...")
028     audio_data = audio.to_soundarray(fps=fps)
029     volumes = np.abs(audio_data).mean(axis=1) if len(audio_data.shape) > 1 else
np.abs(audio_data)
```

```

030
031     # 【修正ポイント 1】 しきい値を「最大音量の 70%」まで引き上げる
032     # これにより、本当に大きな音（重低音のピーク）だけに絞ります。
033     dynamic_threshold = np.max(volumes) * 0.7
034
035     events = []
036     last_event_time = -WAVE_DURATION
037
038     for i, volume in enumerate(volumes):
039         t = i / fps
040         # 【修正ポイント 2】 反応間隔を 0.1 秒 から 0.3 秒 に広げる
041         # 連続して波紋が出すぎるのを防ぎ、メリハリをつけます。
042         if volume > dynamic_threshold and (t - last_event_time) > 0.3:
043             events.append({
044                 'start_time': t,
045                 'center': (np.random.randint(100, 400), np.random.randint(100,
400)),
046                 'volume': volume
047             })
048             last_event_time = t
049
050     # デバッグ表示：これでいくつイベントが登録されたか確認できます
051     print(f"検出された大きな音の数: {len(events)}")
052     return events, audio
053
054 def create_wave_video():
055     WAVE_EVENTS, audio = generate_audio_reactive_events(INPUT_AUDIO, VIDEO_FPS)
056     duration = audio.duration
057     print(f"解析完了: {len(WAVE_EVENTS)} 個のインパクトを検出しました。")
058
059     # 元画像を読み込み
060     img_pil = Image.open(INPUT_IMAGE).convert('RGB').resize((W, H))
061     img_np = np.array(img_pil).astype(np.float32)
062
063     xx, yy = np.meshgrid(np.arange(W), np.arange(H))
064
065     def make_frame(t):
066         dx = np.zeros((H, W), dtype=np.float32)

```

```

067         dy = np.zeros((H, W), dtype=np.float32)
068
069         for event in WAVE_EVENTS:
070             t_rel = t - event['start_time']
071             if 0 <= t_rel < WAVE_DURATION:
072                 cx, cy = event['center']
073                 R = np.sqrt((xx - cx)**2 + (yy - cy)**2)
074                 mask = (R < WAVE_RADIUS)
075
076                 # 【重要】音量(volume)に比例して歪みの強さを変える
077                 # 400 の部分を大きくするとより激しく、小さくすると控えめになりま
078                 strength = (event['volume'] * 400) * np.sin(np.pi * (t_rel /
す WAVE_DURATION))
079                 distort = strength * (1 - (R / WAVE_RADIUS)**2)**2 * mask
080
081                 angle = np.arctan2(yy - cy, xx - cx)
082                 dx += distort * np.cos(angle)
083                 dy += distort * np.sin(angle)
084
085                 # ピクセルを移動
086                 map_x = np.clip(xx + dx, 0, W - 1).astype(int)
087                 map_y = np.clip(yy + dy, 0, H - 1).astype(int)
088
089                 return img_np[map_y, map_x].astype('uint8')
090
091         # VideoClip の作成
092         clip = VideoClip(make_frame, duration=duration)
093
094         # FPS と音声の設定（環境に合わせて両方の書き方に対応）
095         try:
096             clip = clip.set_fps(VIDEO_FPS).set_audio(audio)
097         except:
098             clip = clip.with_fps(VIDEO_FPS).with_audio(audio)
099
100         print(f"動画の書き出しを開始します（約{duration:.1f}秒）...")
101         clip.write_videofile(
102             OUTPUT_VIDEO_MP4,

```

103	codec='libx264',
104	audio_codec='aac',
105	fps=VIDEO_FPS
106)
107	print("完了しました！出力ファイル：" + OUTPUT_VIDEO_MP4)
108	
109	if __name__ == "__main__":
110	create_wave_video()

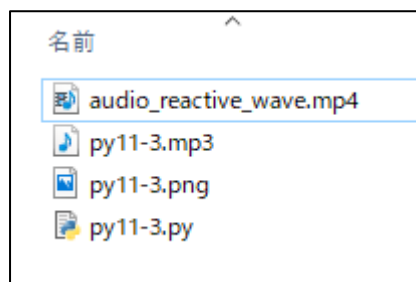
実行時の様子

```
PS C:\Users\student> cd C:\Users\student\Desktop\py3
PS C:\Users\student\Desktop\py3> py py11-3.py
音声を解析中...
検出された大きな音の数: 64
解析完了: 64 個のインパクトを検出しました。
動画の書き出しを開始します (約20.0秒) ...
MoviePy - Building video audio_reactive_wave.mp4.
MoviePy - Writing audio in audio_reactive_waveTEMP_MPY_wvf_snd.mp4
MoviePy - Done.
MoviePy - Writing video audio_reactive_wave.mp4
frame_index: 20%| | 121/601 [00:08<00:31, 15.12it/s, now=None]
```

処理終了

```
PS C:\Users\student> cd C:\Users\student\Desktop\py3
PS C:\Users\student\Desktop\py3> py py11-3.py
音声を解析中...
検出された大きな音の数: 64
解析完了: 64 個のインパクトを検出しました。
動画の書き出しを開始します (約20.0秒) ...
MoviePy - Building video audio_reactive_wave.mp4.
MoviePy - Writing audio in audio_reactive_waveTEMP_MPY_wvf_snd.mp4
MoviePy - Done.
MoviePy - Writing video audio_reactive_wave.mp4
MoviePy - Done !
MoviePy - video ready audio_reactive_wave.mp4
完了しました！出力ファイル: audio_reactive_wave.mp4
PS C:\Users\student\Desktop\py3> █
```

MP4 ファイルが生成されている



【実行結果】



【動画】

py11-3 audio reactive wave <https://youtube.com/shorts/FoOLhINJQ8k>

【感想】

歪みが大きかったので、修正した。音が小さいところでも動いているように見えるが、無音区間がある音楽に変えて試してもいいだろう。