

無音映像に対する効果音挿入システムのGUI設計と実装

福知山公立大学 情報学部情報学科

32245085 松永流翠

指導教員 橋田光代 准教授

提出日 2025年1月31日

改訂日 2025年2月7日

— 目次 ——

1 はじめに	1
1.1 研究背景	1
2 GUI システムの設計	1
2.1 システム全体の概要	1
2.1.1 ユーザが行う操作	
2.1.2 システムが行う処理	
2.2 使用した技術	1
2.3 実装の詳細	2
2.3.1 使用ライブラリのインポート	
2.3.2 GUI のレイアウト設計	
2.3.3 動画ファイルのアップロード	
2.3.4 動画の無音化	
2.3.5 動画の再生	
2.3.6 シーンの分割	
2.3.7 シーン動画の保存	
2.3.8 シーン画像の保存	
2.3.9 シーン情報の HTML 形式での表示	
2.3.10 シーン選択および再生	
2.3.11 複数のシーン選択および再生	
2.3.12 効果音の検索	
2.3.13 効果音の再生	
2.3.14 シーンへの効果音挿入	
2.3.15 複数シーンへの効果音挿入	
2.3.16 効果音付きシーンの選択と再生	
2.3.17 音付けされた動画のプレビュー再生	
2.3.18 音付け完了後のエクスポート	
3 GUI システムの動作例	7
3.1 動画ファイルのアップロード	7
3.2 アップロード動画の再生	8
3.3 動画の無音化	8
3.4 動画のシーン分割	9
3.5 シーン動画の保存	9
3.6 シーン情報の HTML 出力	10
3.7 シーンの選択と再生	10
3.8 複数シーンの選択と再生	11
3.9 効果音の検索および再生	11
3.9.1 効果音の検索	
3.9.2 効果音の再生	
3.10 シーンへの効果音挿入	12
3.11 複数シーンへの効果音挿入	13
3.12 効果音付きシーンの再生	14
3.13 効果音付き動画のプレビュー再生	14
3.14 効果音付き動画のエクスポート	14
4 GUI システムの課題	15

4.1 FFmpeg の処理時間	15
4.2 リストファイルのエラー	15
4.3 複数の効果音の挿入	15

5 まとめと今後の展望	15
--------------------	-----------

— 図目次 ——

1 使用ライブラリのインポートプログラム	2
2 GUI のレイアウト設計プログラム	2
3 動画ファイルのアップロードプログラム	2
4 動画の無音化プログラム	2
5 動画の再生プログラム	3
6 シーンの分割プログラム	3
7 シーン動画の保存プログラム	3
8 シーン画像の保存プログラム	4
9 シーン情報の HTML 形式での表示プログラム	4
10 シーン選択および再生プログラム	4
11 複数のシーン選択および再生プログラム	4
12 効果音の検索プログラム	4
13 効果音データベース(抜粋)	5
14 効果音の再生プログラム	5
15 シーンへの音声挿入プログラム	5
16 複数シーンへの音声挿入プログラム	6
17 音声付きシーンの選択と再生プログラム	6
18 音付けされた動画のプレビュー再生プログラム	6
19 音付け完了後のエクスポートプログラム	7
20 GUI システムの全体像	7
21 ?ボタン(黄色い枠線部分)	7
22 動画ファイルのアップロード手順	8
23 アップロード動画の再生手順	8
24 動画の無音化手順	9
25 動画のシーン分割手順	9
26 シーン動画の保存手順	10
27 シーン情報の HTML 出力手順	10
28 シーンの選択と再生手順	11
29 複数シーンの選択と再生手順	11
30 効果音の検索手順	12
31 効果音の再生手順	12
32 シーンへの効果音挿入手順	13
33 複数シーンへの効果音挿入手順	13
34 効果音付きシーンの再生手順	14
35 効果音付き動画のプレビュー再生手順	14
36 効果音付き動画のエクスポート手順	15

— 表目次 ——

1 システム実装に使用した技術	1
-----------------	---

1. はじめに

1.1 研究背景

舞台演出における照明は、音楽雰囲気を“見える形”に変えてくれる存在であり、観客の感情にダイレクトに作用する重要な要素だといえる。今のコンサートやライブパフォーマンスでは、照明はもはや脇役ではなく、演奏と一体となって「体験」をつくりあげる中心的な要素になっている。特にポピュラー音楽やクラシック音楽の演奏においては、曲の構造や演奏者の動き、さらに即興的な表現までが照明と結びつくことで、観客は深い没入感を味わうことができる。

従来の舞台照明は、オペレーターが手動で操作するか、事前に仕込んだキュー(cue)に従うのが一般的だった。この方式は大規模公演では安定する一方で、演奏中に生まれる即興的な変化には柔軟に対応しにくい。その結果、音楽のダイナミクスと照明演出との間にズレが生じることもある。また、小規模なライブや個人のパフォーマンスでは、照明オペレーターを雇うコストや人手の問題から、十分な演出ができない場合も少なくない。

こうした課題を背景に、最近では音響信号処理やコンピュータ性能の進歩によって、音声の特徴を解析し、照明や映像を自動でコントロールする研究が進んでいる。たとえば、音の強さを示す RMS (Root Mean Square)、音の変化を捉える ZCR (Zero Crossing Rate)、音色の特徴を表す MFCC (Mel Frequency Cepstral Coefficients)、そしてテンポ推定などの指標は、曲の勢いやリズムを数値化するのに有効だと知られている。これらを用いれば、演奏に合わせたリアルタイムの照明制御も可能になる。

さらに、機械学習の発展によって、音響特徴量と視覚的な演出との関係をモデル化できるようになった。これまでには照明デザイナーの経験に頼って決められていたルールを、データに基づいて獲得できるようになり、より柔軟で高精度な演出が実現可能になっている。こうした背景から、本研究では音響信号処理と機械学習を組み合わせた、新しい舞台演出システムを提案する。

2. GUI システムの設計

2.1 システム全体の概要

本研究では、読み込んだ映像に対するシーン分割をもとに、効果音やBGMを簡単に付加できるGUIシステムを構築した。以下に、ユーザが行う操作とシステムが実行する処理を区別して説明する。

2.1.1 ユーザが行う操作

(1) 動画ファイルのアップロード

ユーザは、無音または既存の音声が含まれた動画ファイルをアップロードする。

(2) シーンの分割確認

シーン分割後のプレビューを確認し、シーンの内容をチェックする。

(3) 効果音の検索と選択

キーワードを入力して効果音を検索し、表示されたリストから音源を選択する。検索結果を試聴し、挿入する効果音を決定する。

(4) 効果音の配置

分割された各シーンに対し、効果音を選択操作で配置する。

(5) プレビューの確認

編集内容をプレビューで確認し、必要に応じて配置を調整する。

(6) 動画のエクスポート

完成した動画をエクスポートボタンを押して保存する。

2.1.2 システムが行う処理

(1) 動画の読み込みとシーン分割

アップロードされた動画を解析し、映像フレームの特徴量を基に自動的にシーンを分割する。各シーンの開始時刻、終了時刻を記録し、リストとして GUI に表示する。

(2) シーン分割情報の HTML 出力

シーンごとの開始時刻や終了時刻などの情報を HTML 形式で出力する。

(3) 既存音声の無音化

動画に既存の音声トラックがある場合、FFmpeg を用いて音声トラックを無効化し、映像を無音化する。

(4) 効果音データベースの検索

ユーザが入力したキーワードに基づき、事前に用意された CSV 形式の効果音データベースを検索し、一致する効果音のリストを表示する。

(5) 効果音の再生

ユーザがリスト内の効果音をクリックすると、その音源を Pygame を用いて再生する。

(6) 編集結果のプレビュー生成

効果音の配置情報を基に、編集内容を反映した動画のプレビューを作成、再生する。

(7) 編集結果の動画エクスポート

効果音を挿入した後の動画を FFmpeg を用いてエクスポートする。

2.2 使用した技術

システムの実装には主に以下の 3 つの技術を用いた(表 1)。

プログラミング言語	Python[1]
GUI 構築ライブラリ	Tkinter[2]
映像処理ツール	FFmpeg[3]

表 1: システム実装に使用した技術

2.3 実装の詳細

本節では、GUI システムの具体的な実装プログラムコードについて順に解説する。

2.3.1 使用ライブラリのインポート

```
import pandas as pd
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
import os
import threading
import pygame
import subprocess
import re
import webbrowser
```

図 1: 使用ライブラリのインポートプログラム

(1) pandas

効果音データベースの管理と検索を行うために使用した。CSV 形式で保存された効果音データを読み込み、検索機能を実現するために利用する。

(2) tkinter

GUI の構築に使用した。ユーザーが直感的に操作できるインターフェースを提供する。

(3) threading

複数の処理を同時に実行し、システム全体の応答性を向上させるために利用している。

(4) pygame

効果音の再生機能を実装するために使用した。音声の再生を簡便に行えるライブラリであり、リアルタイムで音声を再生し、ユーザーが選択した効果音を確認できる。

(5) subprocess

映像処理に FFmpeg を呼び出すために使用した。FFmpeg を利用することで、動画の編集や音声の挿入が可能となる。

(6) re(正規表現ライブラリ)

映像処理のログからシーンの開始時刻を抽出するために利用している。

(7) webbrowser

Python プログラム内からユーザーの既定のウェブブラウザを開き、指定した URL を表示するために利用している。

2.3.2 GUI のレイアウト設計

```
self.frame1 = tk.Frame(root, bg="#808080", bd=2, relief=tk.RIDGE)
self.frame1.place(x=80, y=35, width=1380, height=100)

self.frame2 = tk.Frame(root, bg="#808080", bd=2, relief=tk.RIDGE)
self.frame2.place(x=80, y=200, width=250, height=330)

self.frame3 = tk.Frame(root, bg="#808080", bd=2, relief=tk.RIDGE)
self.frame3.place(x=400, y=200, width=300, height=550)

self.frame4 = tk.Frame(root, bg="#808080", bd=2, relief=tk.RIDGE)
self.frame4.place(x=780, y=200, width=300, height=550)

self.frame5 = tk.Frame(root, bg="#808080", bd=2, relief=tk.RIDGE)
self.frame5.place(x=1160, y=200, width=300, height=550)
```

図 2: GUI のレイアウト設計プログラム

本 GUI システムでは、主要な操作パネルやデータ表示領

域を分離するために 5 つの Frame が設置されている。

(1) Frame1

X 座標 80px、Y 座標 35px の位置に幅 1380px、高さ 100px の領域が設置されている。システムのタイトルや基本情報の表示領域として利用される。

(2) Frame2

X 座標 80px、Y 座標 200px の位置に幅 250px、高さ 330px の領域が設置されている。動画のアップロードや再生、シーン分割などを行う領域である。

(3) Frame3

X 座標 400px、Y 座標 200px の位置に幅 300px、高さ 550px の領域が設置されている。シーンリストの表示やシーン動画の再生、シーンに対する音付けを行う領域である。

(4) Frame4

X 座標 780px、Y 座標 200px の位置に幅 300px、高さ 550px の領域が設置されている。効果音の検索や再生を行う領域である。

(5) Frame5

X 座標 1160px、Y 座標 200px の位置に幅 300px、高さ 550px の領域が設置されている。効果音付きのシーンの再生や、音付けが完了した動画のエクスポートなどを行う領域である。

2.3.3 動画ファイルのアップロード

```
def upload_video(self):
    ① self.video_file_path = filedialog.askopenfilename(
        title="動画ファイルを選択",
        filetypes=[("MP4ファイル", "*.mp4")]
    )
    ② if self.video_file_path:
        messagebox.showinfo("動画アップロード",
                            "動画が正常にアップロードされました。")
```

図 3: 動画ファイルのアップロードプログラム

(1) ファイル選択ダイアログを通じて動画ファイルを選択する。

(2) ファイルが正常に選択された場合、アップロード成功のメッセージを通知する。

2.3.4 動画の無音化

```
def mute_video(self):
    ① if not self.video_file_path:
        messagebox.showwarning("警告", "最初に動画をアップロードしてください。")
        return
    threading.Thread(target=self._mute_video_in_background).start()

def _mute_video_in_background(self):
    ② try:
        output_file = "output_no_audio.mp4"
        command = [
            "ffmpeg", "-i", self.video_file_path, "-an", output_file
        ]
        subprocess.run(command, check=True)
        messagebox.showinfo(
            "処理完了", f"無音化された動画が作成されました:{output_file}")
    except subprocess.CalledProcessError as e:
        messagebox.showerror("エラー", f"無音化処理に失敗しました: {e}")
```

図 4: 動画の無音化プログラム

- (1) 動画をアップロードしていない場合に警告を表示する。
動画の無音化処理は計算負荷が高く、実行中に GUI がフリーズする可能性がある。そのため、バックグラウンドで処理を実行するよう設計した。
- (2) FFmpeg を用いて入力動画ファイルから音声トラックを除去し、新しい無音化済み動画ファイルを生成する。処理が正常に完了した場合、成功メッセージを表示する。

2.3.5 動画の再生

```
def play_video(self):
    ① if not self.video_file_path:
        messagebox.showwarning("警告", "最初に動画をアップロードしてください。")
        return
    ② try:
        os.startfile(self.video_file_path)
        messagebox.showinfo("再生開始", "動画が再生されます。")
    except Exception as e:
        messagebox.showerror("エラー", f"動画の再生に失敗しました: {e}")
```

図 5: 動画の再生プログラム

- (1) 動画ファイルがアップロードされているかを確認する。アップロードされていない場合、警告を表示し、処理を中断する。
- (2) os.startfile メソッドを用いて、システムの既定のメディアプレーヤーで動画ファイルを再生することにした。元々は GUI 内にプレビュー画面を作成する予定であったが、GUI がフリーズしてしまうため、メディアプレーヤーで再生する手法を採用した。動画が正常に再生された場合は、再生の開始を通知する。

2.3.6 シーンの分割

```
def split_scenes(self):
    ① if not self.video_file_path:
        messagebox.showwarning("警告", "最初に動画をアップロードしてください。")
        return
    self._run_scene_split()

def _run_scene_split(self):
    ② try:
        self.scene_listbox.delete(0, tk.END)
        self.scene_times.clear()
        self.scene_images.clear()
        command = [
            "ffmpeg",
            "-i", self.video_file_path,
            "-vf", "select='gt(scene,0.7)',showinfo",
            "-vsync", "vfr",
            "-f", "null", ""
        ]
        result = subprocess.run(command, capture_output=True, text=True)
        self._extract_scenes(result.stderr)
    except subprocess.CalledProcessError as e:
        messagebox.showerror("エラー", f"シーン分割に失敗しました: {e}")

def _extract_scenes(self, ffmpeg_output):
    ③ self.scene_times = re.findall(r'pts_time:(\d+\.\d+)', ffmpeg_output)
    if not self.scene_times:
        messagebox.showinfo("結果", "シーン分割の結果が見つかりませんでした。")
        return
    for idx, time in enumerate(self.scene_times):
        scene_info = f"シーン {idx + 1}: 開始時刻 {float(time):.2f} 秒"
        self.scene_listbox.insert(tk.END, scene_info)
    threading.Thread(target=self._save_scene_images).start()
```

図 6: シーンの分割プログラム

- (1) 動画ファイルが選択されているかを確認した後、内部的なシーン分割処理を呼び出す。
- (2) シーン間の変化が 0.7 以上の閾値を超える場合に、フ

レームを抽出する条件を示す。このフィルタにより、視覚的な大きな変化（カットなど）を特定可能となる。

- (3) FFmpeg の出力から、シーン切り替わり時刻を正規表現で抽出する。抽出された情報はシーンリストボックスに追加され、ユーザーが確認可能な形式で表示される。

2.3.7 シーン動画の保存

```
def save_scenes(self):
    ① if not self.scene_times:
        messagebox.showwarning("警告", "シーンが抽出されませんでした。")
        return
    save_dir = filedialog.askdirectory(title="保存先フォルダを選択")
    if not save_dir:
        return
    save_thread = threading.Thread(target=self._save_scenes_thread, args=(save_dir,))
    save_thread.start()

def _save_scenes_thread(self, save_dir):
    ② for idx, start_time in enumerate(self.scene_times):
        start_time = float(start_time)
        end_time = float(self.scene_times[idx + 1]) if idx + 1 < len(self.scene_times) else None
        output_file = os.path.join(save_dir, f"scene_{idx}.mp4")
        self._save_scene(start_time, end_time, output_file)
        messagebox.showinfo("完了", "シーンの保存が完了しました。")
    except Exception as e:
        messagebox.showerror("エラー", f"シーンの保存中にエラーが発生しました: {e}")

def _save_scene(self, start_time, end_time, output_file):
    ③ command = [
        "ffmpeg",
        "-i", self.video_file_path,
        "-ss", str(start_time),
    ]
    if end_time:
        command.extend(["-to", str(end_time)])
    command.extend([
        "-c:v", "libx264",
        "-preset", "fast",
        "-ac", "aac",
        "-strict", "experimental",
        output_file
    ])
    subprocess.run(command, check=True)
```

図 7: シーン動画の保存プログラム

- (1) シーンが分割されていない場合、警告メッセージを表示する。シーンが分割されれば、次に保存先のフォルダを選択するダイアログが表示される。そして、保存先フォルダを選択させ、フォルダが選択されなかった場合は、処理を中断する。
- (2) シーンの開始時刻を保存し、それをもとに、各シーンの終了時刻を計算する。各シーンごとに、計算された時間範囲で動画を切り出して保存する。保存処理中にエラーが発生した場合、エラーメッセージを表示する。エラーが発生しなければ、保存完了のメッセージが表示される。
- (3) 各シーンは、FFmpeg を使用して切り出される。コマンドには、入力ファイル、開始時刻、終了時刻、および出力ファイルの情報が含まれる。生成したFFmpeg コマンドを実行し、シーン動画の保存処理を実行する。

2.3.8 シーン画像の保存

```
def _save_scene_images(self):
    ① for idx, start_time in enumerate(self.scene_times):
        output_image_path = f"scene_{idx + 1:03d}.png"
        try:
            command = [
                "ffmpeg",
                "-i", self.video_file_path,
                "-ss", str(float(start_time)),
                "-vframes", "1",
                output_image_path
            ]
            subprocess.run(command, check=True)
            self.scene_images.append(output_image_path)
        ② except subprocess.CalledProcessError as e:
            messagebox.showerror("エラー", f"シーン画像の抽出に失敗しました: {e}")
    self.view_html_button.config(state=tk.NORMAL)
    messagebox.showinfo("完了", "シーン画像の保存が完了しました!")
```

図 8: シーン画像の保存プログラム

- (1) 各シーンの開始時刻を利用して、その時刻で動画から 1 フレーム（静止画像）を抽出する。
- (2) 実行中にエラーが発生した場合、エラーメッセージを通知する。すべてのシーン画像が保存された後、「完了」のメッセージを表示する。

2.3.9 シーン情報の HTML 形式での表示

```
def view_scenes_as_html(self):
    ① if not self.scene_times:
        messagebox.showwarning("警告", "シーンが抽出されていません。")
        return
    scene_data = []
    for start_time in enumerate(self.scene_times):
        start_time = float(start_time)
        end_time = float(self.scene_times[idx + 1]) if idx + 1 < len(self.scene_times) else start_time + 5
        scene_data.append((start_time, end_time))
    html_content += "<html><head><title>シーン分割</title></head><body>"
    ② html_content += "<table border='1'><tr><th>シーン番号</th><th>開始時刻 (秒)</th><th>終了時刻 (秒)</th><th>プレビュー</th></tr>"
```

- ③ for idx, (start_time, end_time) in enumerate(scene_data):
 image_path = self.scene_images[idx] if idx < len(self.scene_images) else ""
 html_content += f"<tr><td>{idx}</td><td>:{start_time:.2f}</td><td>:{end_time:.2f}</td><td></td></tr>"
 else:
 html_content += f"<tr><td colspan=4></td></tr>"
 html_content += "</table></body></html>"
 output_file = filedialog.asksaveasfilename(defaultextension=".html", filetypes=[("HTML ファイル", ".html")])
 ④ if output_file:
 with open(output_file, "w", encoding="utf-8") as f:
 f.write(html_content)
 webbrowser.open(f"file:///{os.path.abspath(output_file)}")

図 9: シーン情報の HTML 形式での表示プログラム

- (1) シーン分割が実行されていないと、シーン情報が表示できないため、シーン分割が実行されているかどうかを確認する。シーンが分割されていない場合、警告メッセージを表示する。
- (2) シーン情報を HTML で表示するために、HTML 文書の基本構造を作成する。ここでは、タイトル、ヘッダー、表を設定し、シーン番号、開始時刻、終了時刻、プレビュー画像を含む列を用意している。
- (3) 事前に計算された開始時刻と終了時刻を、HTML の表に行として挿入する。各行には、対応するシーンの開始時刻、終了時刻、プレビュー画像が表示される。
- (4) 完成した HTML コンテンツは、ユーザーが指定した場所に保存される。保存後、そのファイルを Web ブラウザで自動的に開くため、ユーザーが生成されたシーン情報を即座に確認できるようになる。

2.3.10 シーン選択および再生

```
def play_selected_scene(self):
    ① selected_index = self.scene_listbox.curselection()
    if not selected_index:
        messagebox.showwarning("警告", "再生するシーンを選択してください。")
        return
    ② scene_index = selected_index[0]
    if scene_index < len(self.saved_scene_paths):
        scene_path = self.saved_scene_paths[scene_index]
        os.startfile(scene_path)
    ③ else:
        messagebox.showwarning("警告", "保存されたシーンが見つかりません。")
```

図 10: シーン選択および再生プログラム

- (1) GUI 上のシーンリストボックスから再生したい 1 つシーンを選択する。
- (2) シーンのインデックスがリスト内に存在する場合、保存したシーン動画のリストから該当するシーンのパスを取得する。
- (3) 選択されたシーン動画を再生する。

2.3.11 複数のシーン選択および再生

```
def merge_and_play_selected_scenes(self):
    ① threading.Thread(target=self._merge_and_play_selected_scenes).start()

def _merge_and_play_selected_scenes(self):
    try:
        selected_indices = self.scene_listbox.curselection()
        if not selected_indices:
            messagebox.showwarning("警告", "統合するシーンを選択してください。")
            return
    ② selected_scene_paths = [self.saved_scene_paths[i] for i in selected_indices]
    temp_output_path = "temp_merged_scene.mp4"
    list_file_path = "temp_merge_list.txt"
    with open(list_file_path, "w") as f:
        for scene_path in selected_scene_paths:
            f.write(f"file '{scene_path}'\n")
    command = [
        "ffmpeg",
        "-f", "concat",
        "-safe", "0",
        "-i", list_file_path,
        "-c", "copy",
        temp_output_path
    ]
    subprocess.run(command, check=True)
    ③ os.system(f"start {temp_output_path}")
    except subprocess.CalledProcessError as e:
        messagebox.showerror("エラー", f"シーンの統合に失敗しました: {e}")
    finally:
        os.remove(list_file_path)
```

図 11: 複数のシーン選択および再生プログラム

- (1) GUI 上でシーンリストボックスから再生したい複数のシーンを選択。
- (2) 選択されたシーンのパスを取得し、FFmpeg で使用するリストファイルを動的に作成される。リストファイルに基づきシーンを連結し、複数のシーンが統合された動画が生成される。
- (3) 統合された動画がデフォルトのメディアプレーヤーで再生される。また、不要なファイルを残さないように、一時リストファイルを削除する。

2.3.12 効果音の検索

```
def search_sounds(sound_df, keyword):
    ① results = sound_df['tag1'].str.contains(keyword, case=False, na=False) & sound_df['tag2'].str.contains(keyword, case=False, na=False)
    return results

def update_sound_list(self, event):
    ② keyword = self.sound_search_box.get()
    filtered_sounds = search_sounds(self.sound_df, keyword)
    self.sound_listbox.delete(0, tk.END)
    for index, row in filtered_sounds.iterrows():
        self.sound_listbox.insert(tk.END, row['name'])
```

図 12: 効果音の検索プログラム

- (1) GUI 内の検索ボックスからユーザーが入力した検索キーワードを取得する。効果音ラボ [4] 等から収集した約 4000 種類の音声ファイルを基に作成した効果音データベース (図 14) の tag1 列 (その効果音に関連するキーワード 例: 爆発音→爆発、斬撃音→剣) と tag2 列 (その効果音のオノマトペ) をもとに、キーワードに一致する効果音を抽出する。

id	name	file_path	tag1	tag2
1	爆破・爆発01.mp3	C:\Users\%saru	爆発	ドーン
2	爆破・爆発02.mp3	C:\Users\%saru	爆発	パン
3	爆破・爆発03.mp3	C:\Users\%saru	爆発	バーン
4	爆破・爆発04.mp3	C:\Users\%saru	爆発	ズン
5	爆破・爆発05.mp3	C:\Users\%saru	爆発	ドゴーン
6	爆破・爆発06.mp3	C:\Users\%saru	爆発	ドーン
7	爆破・爆発07.mp3	C:\Users\%saru	爆発	ドーン
8	爆破・爆発08.mp3	C:\Users\%saru	爆発	ドゴン
9	爆破・爆発09.mp3	C:\Users\%saru	爆発	ズドン
10	爆破・爆発10.mp3	C:\Users\%saru	爆発	ドンッ
11	爆破・爆発11.mp3	C:\Users\%saru	爆発	バンッ
12	爆破・爆発12.mp3	C:\Users\%saru	爆発	ドーン
13	爆破・爆発13.mp3	C:\Users\%saru	爆発	ズシン
14	爆破・爆発14.mp3	C:\Users\%saru	爆発	ドコン
15	爆破・爆発15.mp3	C:\Users\%saru	爆発	ドゴン

図 13: 効果音データベース (抜粋)

- (2) 検索結果を GUI 内の効果音リストボックスに表示するため、既存のリストを初期化した後、新たな検索結果を挿入する。

2.3.13 効果音の再生

```
def open_selected_sound_in_player(self, event=None):
    ① selected_index = self.sound_listbox.curselection()
    ② if selected_index:
        sound_file = self.sound_df.iloc[selected_index[0]]['file_path']
        os.startfile(sound_file)
```

図 14: 効果音の再生プログラム

- (1) 効果音リストボックスでユーザーが選択した項目のインデックスを取得する。
(2) 効果音データベースの file_path 列 (図 14) を参照し、選択インデックスに対応する音素材のファイルパスを抽出する。抽出されたファイルパスをもとに、音素材が再生される。

2.3.14 シーンへの効果音挿入

```
def insert_audio_with_timing(self):
    ❶ audio_file = filedialog.askopenfilename(initialdir=self.default_audio_folder, filetypes=[("MP3ファイル", "*.mp3")])
    if not audio_file:
        return
    selected_indices = list(self.scene_listbox.curselection())
    if not selected_indices:
        messagebox.showwarning("警告", "音声を挿入するシーンを選択してください。")
    ❷ timing_offset = simpledialog.askfloat("音声タイミング", "音声を開始する時間(秒単位)を入力してください", minvalue=0)
    if timing_offset is None:
        return
    for index in selected_indices:
        scene_path = self.saved_scene_paths[index]
        output_file = scene_path.replace(".mp4", f"_audio_{(index + 1)}.mp4")
        self.insert_audio_with_timing(scene_path, audio_file, output_file, timing_offset)
        self.audio_listbox.insert(index, output_file)
        display_text += f"シーン {index + 1} ({os.path.basename(audio_file)}) (開始 {timing_offset}s)"
```

```
self.audio_files_for_scenes.append(os.path.basename(audio_file))
self.play_audio_scene_button.config(state=tk.NORMAL)
self.play_audio_scene_button["text"] = "再生(クリア:音声を削除する)"
self.play_audio_scene_button["command"] = lambda: self.insert_audio_with_timing(self, video_file, audio_file, offset)
❸ try:
    command = [
        "ffmpeg",
        "-i",
        video_file,
        "-i",
        offset_file,
        "-i",
        audio_file,
        "-c:a",
        "copy",
        "-c:v",
        "aac",
        "-map",
        "0:v:0",
        "-map",
        "1:a:0",
        "-shortest",
        output_file
    ]
    subprocess.run(command, check=True)
except subprocess.CalledProcessError as e:
    messagebox.showerror("エラー", f"音声の導入に失敗しました: {e}")
```

図 15: シーンへの音声挿入プログラム

- (1) ファイル選択ダイアログを使用して音声ファイル (MP3 形式) を選択。この選択がなければ、処理は中断される。そして、シーンリストボックスから、音声を挿入したいシーンを選択する。もしシーンが選択されていない場合、警告メッセージが表示される。
(2) 音声を開始するタイミング (秒単位) を入力。このオフセットにより、音声の開始位置が調整される。各シーンに対して、指定されたタイミングで音声が挿入され、音声付きの新しいシーンファイルが生成される。音声付きのシーンファイルのパスが追加され、音声ファイル名が登録される。そして、効果音付きシーンが GUI 内の効果音付きリストボックスに表示される。
(3) 音声挿入は、FFmpeg を利用して行われる。指定されたオフセット時間後に音声が開始されるように設定される。映像の映像ストリームと音声ストリームをマッピングし、最短のメディア長に合わせて出力される。

2.3.15 複数シーンへの効果音挿入

```

def insert_audio_across_scenes(self):
    audio_file = filedialog.askopenfilename(initialdir=self.default_audio_folder, filetypes=[("MP3ファイル", "*.mp3")])
    ① if not audio_file:
        return
    selected_indices = list(self.audio_scene_listbox.curselection())
    if len(selected_indices) < 2:
        messagebox.showwarning("警告", "複数のシーンを選択してください。")
        return
    start_index, end_index = selected_indices[0], selected_indices[1]
    audio_start_time = simpledialog.askstring("音声開始タイミング", "音声を再生する時間(秒)を入力してください (例: 5.5):")
    try:
        audio_start_time = float(audio_start_time)
        if audio_start_time < 0:
            raise ValueError("負の値は無効です。")
        except (ValueError, TypeError):
            messagebox.showwarning("エラー", "音声の値が不正です。")
    merged_scenes_path = self._merge_scenes(scenes_to_merge)
    if not merged_scenes_path:
        return
    output_file = merged_scenes_path.replace(".mp4", "_with_audio.mp4")
    self._insert_audio_with_timing(merged_scenes_path, audio_file, audio_start_time, output_file)
    self.audio_scene_listbox.insert(END, os.path.basename(output_file)) ②
    self.audio_scene_listbox.insert(END, display_text)
    self.play_audio_scenes_button.config(state=tk.NORMAL)
    messagebox.showinfo("完了", "複数のシーンに音声が挿入されました。")
    self._merge_scenes(self, scenes_to_merge)
    ③ try:
        temp_list_file = "temp_scenes_list.txt"
        with open(temp_list_file, "w") as f:
            for scene in scenes_to_merge:
                f.write(f"file '{scene}'\n")
        merged_scene_path = "merged_scenes.mp4"
        command = [
            "ffmpeg",
            "-f", "concat",
            "-safe", "0",
            "-i", temp_list_file,
            "-c:v", "copy",
            "-c:a", "aac",
            merged_scene_path
        ]
        subprocess.run(command, check=True)
        os.remove(temp_list_file)
        return merged_scene_path
    except subprocess.CalledProcessError as e:
        messagebox.showerror("エラー", f"シーンの結合に失敗しました: {e}")
    return None

def _insert_audio_with_timing(self, video_file, audio_file, start_time, output_file):
    """音声を挿入する際、指定された時間タイミングを表示します。"""
    ④ try:
        command = [
            "ffmpeg",
            "-i", video_file,
            "-itsoffset", str(start_time),
            "-t", audio_file,
            "-c:v", "copy",
            "-c:a", "aac",
            "-shortest",
            output_file
        ]
        subprocess.run(command, check=True)
    except subprocess.CalledProcessError as e:
        messagebox.showerror("エラー", f"音声の挿入に失敗しました: {e}")

```

図 16: 複数シーンへの音声挿入プログラム

- 挿入する音声ファイルを選択する。選択が行われない場合、処理は中断される。シーンリストボックスから複数のシーンを選択することができる。選択されたシーンが 2 つ以上でなければならず、2 つ以上選択されていない場合、警告メッセージが表示され、処理は終了する。
- 音声の開始タイミングを指定する。ユーザーに秒単位での開始時間を入力させる。
- 音声を複数のシーンに挿入するため、まずシーンの結合が行われる。選択されたシーンのパスはリスト化され、これらを 1 つの動画に統合する。この際、シーンを結合した新たな動画ファイルが作成される。
- シーンの結合が完了した後、音声の挿入が行われる。音声ファイルの開始時間は、前述のユーザー入力に基づいて決定される。音声は動画ファイルに挿入される。ここでは、ffmpeg コマンドが呼び出され、指定された開始タイミングで音声が挿入されるように設定される。また、音声のオフセット時間を指定し、音声と映像の同期が取れるようにする。音声の挿入が完了すると、音声を含む新たな動画ファイルが保存され、効果音付きシーンリストに追加される。

2.3.16 効果音付きシーンの選択と再生

```

def play_selected_audio_scene(self):
    ① selected_index = self.audio_scene_listbox.curselection()
    if not selected_index:
        messagebox.showwarning("警告", "再生する音声付きシーンを選択してください。")
        return
    scene_index = selected_index[0]
    if scene_index < len(self.audio_inserted_scene_paths):
        scene_path = self.audio_inserted_scene_paths[scene_index]
        ② try:
            os.startfile(scene_path)
        except Exception as e:
            messagebox.showerror("エラー", f"シーンの再生中にエラーが発生しました: {e}")
    else:
        messagebox.showwarning("警告", "保存されたシーンが見つかりません。")

```

図 17: 音声付きシーンの選択と再生プログラム

- 効果音付きシーンリストボックスから効果音付きシーンが選択されているかどうかを確認する。選択されたインデックスを取得し、選択が行われていない場合は、警告メッセージを表示する。
- 選択されたインデックスに基づいて、効果音付きのシーンファイルのリストから該当するシーンのパスを取得する。もしインデックスが無効であれば、”保存されたシーンが見つかりません”という警告メッセージが表示され、処理は終了する。
- 有効なシーンのパスが見つかった場合、デフォルトのメディアプレーヤーにてそのシーンを開く。再生中にエラーが発生した場合、エラーメッセージが表示される。

2.3.17 音付けされた動画のプレビュー再生

```

def preview_merge(self):
    ① if not self.audio_inserted_scene_paths:
        messagebox.showwarning("警告", "プレビューするシーンがありません。")
        return
    self.preview_merge_button.config(state=tk.DISABLED)
    threading.Thread(target=self._create_and_play_preview, daemon=True).start()

def _create_and_play_preview(self):
    ② sorted_scenes = sorted(self.audio_inserted_scene_paths)
    list_file_path = "scene_preview_list.txt"
    with open(list_file_path, "w") as f:
        for scene_path in sorted_scenes:
            f.write(f"file '{scene_path}'\n")
    preview_file = "preview_output.mp4"
    ③ try:
        command = [
            "ffmpeg",
            "-f", "concat",
            "-safe", "0",
            "-i", list_file_path,
            "-c:v", "copy",
            preview_file
        ]
        subprocess.run(command, check=True)
        os.startfile(preview_file)
    ④ except subprocess.CalledProcessError as e:
        messagebox.showerror("エラー", f"プレビューに失敗しました: {e}")
    finally:
        if os.path.exists(list_file_path):
            os.remove(list_file_path)
    self.preview_merge_button.config(state=tk.NORMAL)

```

図 18: 音付けされた動画のプレビュー再生プログラム

- 効果音付きのシーンが存在しているかどうかが確認される。シーンが存在していない場合、警告メッセージが表示され、処理が終了する。効果音付きのシーンが存在する場合、プレビュー作成と再生が別スレッドで非同期に実行される。
- 音声付きのシーンファイルのリストに格納されたシ

- ンファイルパスをシーン番号順にソートし、その後、FFmpeg に渡すための入力ファイルリストを作成する。リストは一時的にテキストファイルに書き込まれる。
- (3) 作成されたリストファイルを基に、複数のシーンを 1 つのプレビュー動画に統合する。FFmpeg はリストファイル内のシーンを順番に結合し、映像と音声を再エンコードせずにコピーする。
 - (4) 統合が成功した後、生成されたプレビュー動画をデフォルトのメディアプレーヤーにて再生する。プレビュー作成後、リストファイルは削除される。

2.3.18 音付け完了後のエクスポート

```

def merge_scenes(self):
    ① if not self.audio_inserted_scene_paths:
        messagebox.showwarning("警告", "統合するシーンがありません。")

    sorted_scenes = sorted(self.audio_inserted_scene_paths)
    list_file_path = "scene_merge_list.txt"
    with open(list_file_path, "w") as f:
        for scene_path in sorted_scenes:
            f.write(f"file '{scene_path}'\n")
    output_file = filedialog.asksaveasfilename(defaultextension=".mp4", filetypes=[("MP4 ファイル", "*.mp4")])
    if not output_file:
        return

    ③ try:
        command = [
            "ffmpeg",
            "-f",
            "concat",
            "-safe",
            "0",
            "-i",
            list_file_path,
            "-c",
            "copy",
            output_file
        ]
        subprocess.run(command, check=True)
        messagebox.showinfo("成功", "シーンを統合してエクスポートしました。")
    except subprocess.CalledProcessError as e:
        messagebox.showerror("エラー", f"シーンの統合に失敗しました: {e}")
    finally:
        os.remove(list_file_path)

```

図 19: 音付け完了後のエクスポートプログラム

- (1) 効果音付きのシーンが存在しているかどうかが確認される。シーンが存在していない場合、警告メッセージが表示され、処理が終了する。
- (2) 音声付きのシーンファイルのリストに格納されたシーンファイルパスを昇順にソートし、その順序で FFmpeg に渡すための入力ファイルリストを作成する。このリストは、一時的にテキストファイルに書き込まれる。リストファイルには、統合したいシーンのパスが file '...' という形式で記載される。
- (3) 作成されたリストファイルを基に、複数のシーンを 1 つのプレビュー動画に統合する。FFmpeg はリストファイル内のシーンを順番に結合し、映像と音声を再エンコードせずにコピーする。
- (4) 統合処理中にエラーが発生した場合、適切なエラーメッセージをユーザーに表示する。そして、統合処理が完了した後、一時的に作成したリストファイルが削除される。

3. GUI システムの動作例

ユーザがどのようにシステムを利用して動画に音付を行うか、具体的な例を手順ごとに紹介する。GUI システムの全体像は以下(図 20)のようになっている。また、各機能の詳細については、各ボタンの右にある「?」ボタン(図 21)をクリックすることで確認可能である。

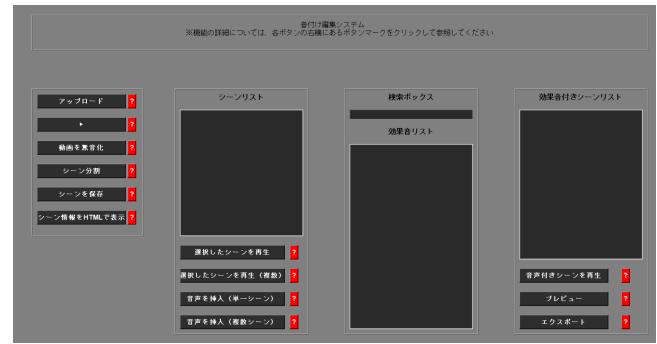


図 20: GUI システムの全体像

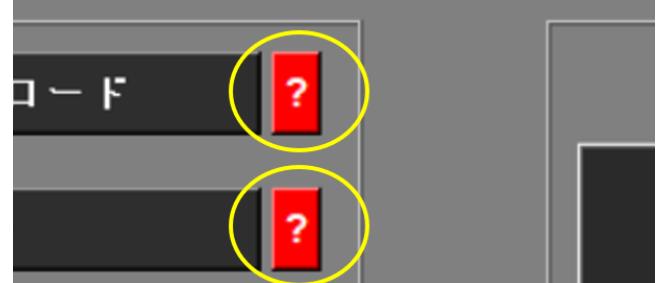


図 21: ?ボタン (黄色い枠線部分)

3.1 動画ファイルのアップロード

まずは、音付け編集を行うための動画ファイルをアップロードする(図 22)。この動作例では、ワールドトリガーの戦闘シーン[5]の一場面を題材としている。



図 22: 動画ファイルのアップロード手順



図 23: アップロード動画の再生手順

- (1) 「アップロード」ボタンをクリックすることで、ファイル選択ダイアログが表示され、ユーザーは動画ファイルを選択することができる(図 22)。
- (2) ユーザーは、ファイル選択ダイアログ内で MP4 形式の動画ファイルを選択し、「開く」ボタンをクリックする(図 22)。
- (3) ファイルが正常に選択されると、画面に「動画が正常にアップロードされました。」というメッセージが表示される(図 22)。

3.2 アップロード動画の再生

続いて、アップロードした動画を再生していく(図 23)。

- (1) 「再生」ボタンをクリックするとシステム既定のメディアプレーヤーにて動画が再生される(図 23)。
- (2) 再生が開始されると、「動画が再生されます」という情報メッセージが表示され、ユーザーに再生が成功したことが通知される。もし、再生中に何らかのエラーが発生した場合(例えば、指定されたファイルが存在しない、またはファイルが破損している場合)、エラーメッセージが表示され、再生に失敗した理由がユーザーに通知される(図 23)。

3.3 動画の無音化

アップロードした動画にそのまま音付けを行うと、元の動画の音声と挿入する効果音が混ざってしまうため、動画の元の音声を消す必要がある(図 24)。また、動画の無音化処理は、FFmpeg を用いたエンコードが必要なため、処理に一定の時間を要する。動画読み込み時に自動実行すると、処理に無駄な時間がかかる可能性があるため、動画無音化ボタンを設けた。



図 24: 動画の無音化手順

- (1) 「動画無音化」ボタンをクリックすると、指定された動画ファイルに対して音声トラックを削除する。出力ファイル名はデフォルトで output_no_audio.mp4 に設定される(図 24)。
- (2) 無音化処理が正常に完了した場合、情報メッセージ「無音化された動画が作成されました: output_no_audio.mp4」が表示される。これにより、ユーザーは処理結果を確認できる。無音化処理に失敗した場合、エラーメッセージが表示され、処理失敗の原因をユーザーに通知する(図 24)。
- (3) そして最後に、作成された output_no_audio.mp4 を再度「アップロード」ボタンにて読み込む。また、一度無音化した動画は再度無音化する必要はない(図 24)。

3.4 動画のシーン分割

本 GUI システムはシーン分割(図 25)をもとに音付けしていく。



図 25: 動画のシーン分割手順

- (1) 「シーン分割」ボタンをクリックすると、アップロードされた動画ファイルに対し、シーン分割処理が開始される(図 25)。
- (2) シーンごとの情報(例:「シーン 1: 開始時刻 0.00 秒」)がシーンリストボックスに順次表示される。動画ファイルが選択されていない場合や、FFmpeg コマンドの実行に失敗した場合、警告メッセージやエラーメッセージが表示され、ユーザーに原因を通知する(図 25)。

3.5 シーン動画の保存

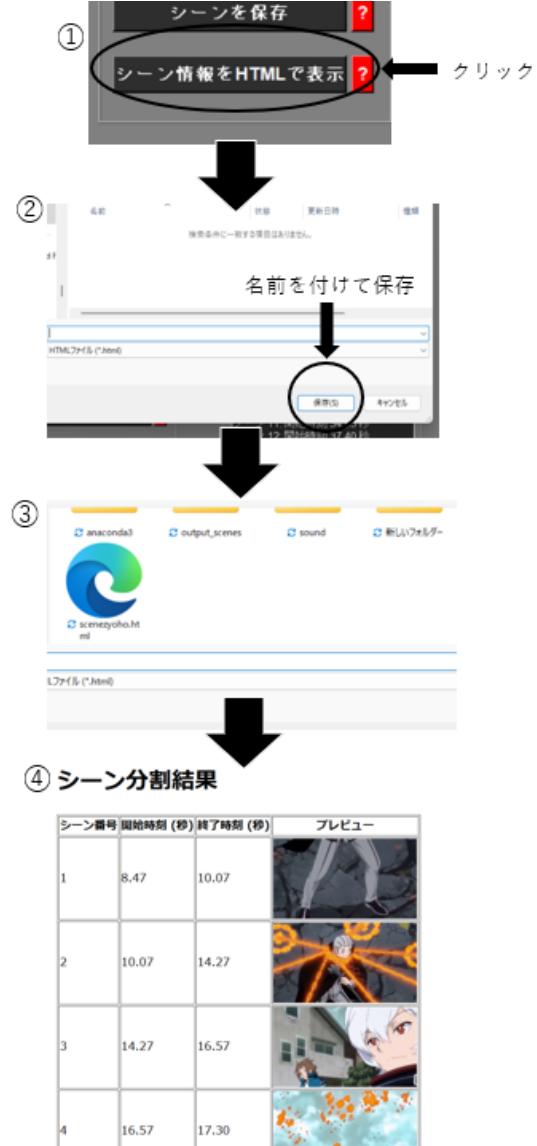
シーンの分割後、シーン動画を任意のフォルダに保存していく(図 26)。



- 「シーン保存」ボタンをクリックすると、フォルダ選択ダイアログが表示される(図 26)。
- ユーザーは、保存先のフォルダを指定し、選択する(図 26)。
- シーン動画の保存処理が実行され、選択した保存先フォルダ内に、シーン動画ファイルが保存される(図 26)。
- シーン保存が正常に完了すると、「シーンの保存が完了しました」というメッセージが表示される(図 26)。

3.6 シーン情報の HTML 出力

シーン分割の視覚的な結果を出力する機能は、ユーザーが各シーンの情報を確認する上で重要である。そのため、本 GUI システムでは、シーン分割情報を HTML 形式でエクスポートし、ブラウザでのプレビューを提供する(図 27)。



- 「シーン情報を HTML で表示」ボタンをクリックすると、保存先を指定するダイアログが表示される(図 27)。
- ユーザーは HTML ファイルの保存場所を選択する。選択後、シーン情報を基に、開始時刻・終了時刻・プレビュー画像を含む HTML コンテンツが動的に生成される(図 27)。
- 生成後、指定された場所に HTML ファイルが出力される。ファイル名にはデフォルト拡張子として.html が付加される(図 27)。
- 保存完了後、自動的にブラウザが起動し、生成された HTML ファイルが開かれる。ユーザーは表形式でシーン情報を視覚的に確認できる(図 27)。

3.7 シーンの選択と再生

シーン動画の保存後、シーン動画を再生し、各シーンの

内容を確認する(図 28)。



図 28: シーンの選択と再生手順

- (1) シーンリストボックスから再生したいシーンをクリックして選択する(図 28)。
- (2) 「選択したシーンを再生」ボタンをクリックする(図 28)。
- (3) デフォルトのメディアプレーヤーにてシーン動画が再生される(図 28)。

3.8 複数シーンの選択と再生

本 GUI システムでは、1 つのシーンだけでなく、複数のシーンの再生も可能である(例: シーン 4 からシーン 7 まで再生したい場合、シーン 4 から 7 まで複数選択)(図 29)。

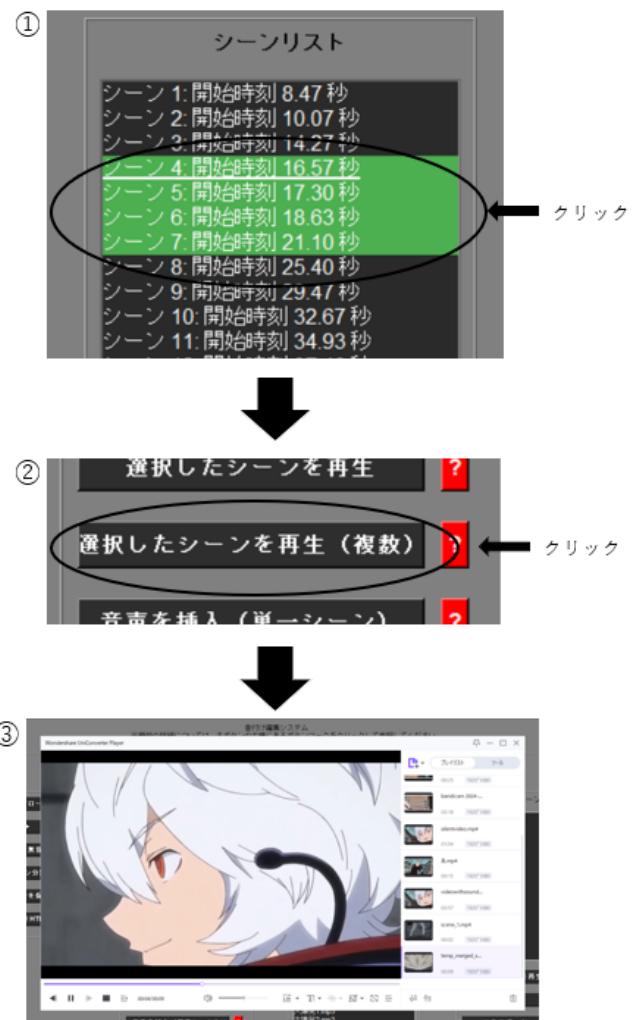


図 29: 複数シーンの選択と再生手順

- (1) シーンリストボックスから再生したいシーンをクリックして選択する(図 29)。
- (2) 「選択したシーンを再生 (複数)」ボタンをクリックする(図 29)。
- (3) デフォルトのメディアプレーヤーにて、選択した範囲のシーン動画が再生される(図 29)。

3.9 効果音の検索および再生

本 GUI システムでは、効果音リストから検索条件に基づくフィルタリングと、選択された効果音ファイルの即時再生機能を提供している(図 30)、(図 31)。

3.9.1 効果音の検索

3.9.2 効果音の再生

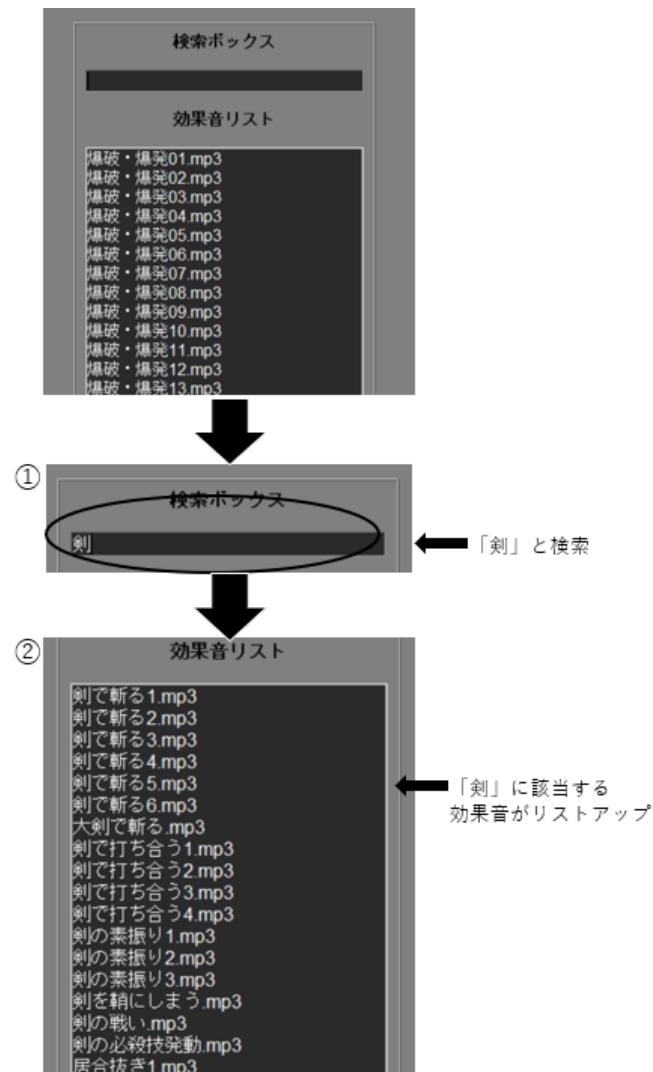


図 30: 効果音の検索手順



図 31: 効果音の再生手順

- (1) ユーザーは検索ボックスに目的の効果音に関連するキーワードを入力する(図 30)。
- (2) 入力内容が変更されるたびに効果音リストがリアルタイムで更新され、該当する効果音のみがリストボックスに表示される(図 30)。

- (1) ユーザーは更新されたリストボックスの中から目的の効果音を選択する(図 31)。
- (2) 選択した効果音をダブルクリックすることで、効果音がデフォルトのメディアプレイヤーで再生される(図 31)。

3.10 シーンへの効果音挿入

シーンと効果音を再生した後、選択したシーンに音声ファイルを挿入する。また、音声の開始タイミングは設定可能である(図 32)。



図 32: シーンへの効果音挿入手順

- (1) ユーザーはシーンリストボックスから、効果音を挿入したいシーンを選択する。シーンが選択されていない場合、警告メッセージが表示される(図 32)。
- (2) 「音声を挿入(单一シーン)」ボタンをクリックする(図 32)。
- (3) 音声ファイル選択用のファイルダイアログが表示され、ユーザーは目的の音声ファイル(MP3 形式)を選択する(図 32)。
- (4) ダイアログボックスを通じて、ユーザーは音声開始タイミング(秒単位)を指定する(例: シーン開始 2 秒後に効果音挿入したい場合は、「2」と指定)(図 32)。
- (5) 音声ファイルと選択されたシーンファイルを基に、音声挿入処理が実行され、挿入済みのシーン情報が効果音付きシーンリストに追加される(図 32 では、シーン 3 に「K.O.mp3」という音声ファイルがシーン開始 2 秒後に挿入されている)。

3.11 複数シーンへの効果音挿入

本 GUI システムでは、1 つのシーンだけでなく、複数

シーンにまたいで、音声の挿入が可能である(図 33)。



図 33: 複数シーンへの効果音挿入手順

- (1) ユーザーはシーンリストボックスから、音声を挿入したい複数シーンを選択する。シーンが選択されていない場合、警告メッセージが表示される(図 33)。
- (2) 「音声を挿入(複数シーン)」ボタンをクリックする(図 33)。
- (3) 音声ファイル選択用のファイルダイアログが表示され、ユーザーは目的の音声ファイル(MP3 形式)を選択する(図 33)。
- (4) ダイアログボックスを通じて、ユーザーは音声開始タイミング(秒単位)を指定する(図 33)。
- (5) 音声ファイルと選択されたシーンファイルを基に、音声挿入処理が実行され、挿入済みのシーン情報が効果音付きシーンリストに追加される(図 33 では、シーン 1 から 3 にまたいで、「K.O.mp3」という音声ファイルがシーン開始 2 秒後に挿入されている)。

3.12 効果音付きシーンの再生

本 GUI システムでは、ユーザーがシーンの再生を通じて音声の配置やタイミングの確認を行えるように、効果音が挿入されたシーンを選択して再生する機能を提供している(図 34)。

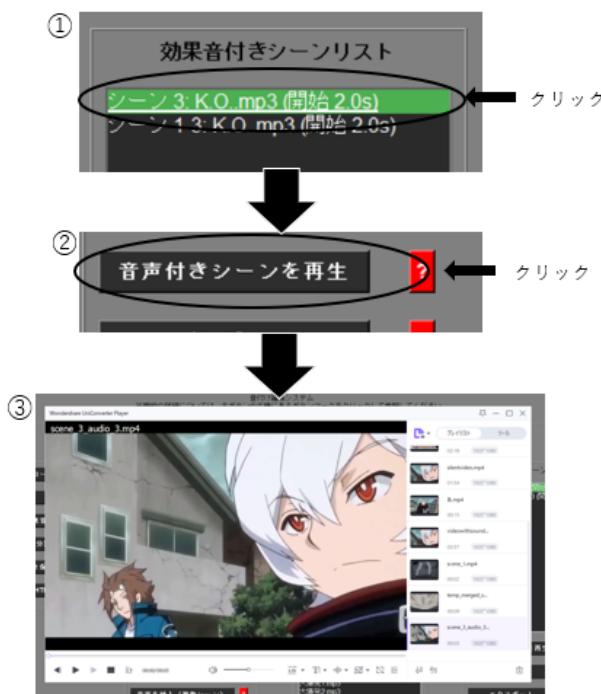


図 34: 効果音付きシーンの再生手順

- (1) ユーザーは、効果音付きシーンリストボックスから、再生したい効果音付きシーンを選択する(図 34)。
- (2) 「音声付きシーンを再生」ボタンをクリックする。シーンが選択されていない場合、警告メッセージが表示され、「再生する音声付きシーンを選択してください」と通知される(図 34)。
- (3) 選択されたシーンがデフォルトのメディアプレーヤーで再生される(図 34)。

3.13 効果音付き動画のプレビュー再生

本 GUI システムでは、ユーザーが効果音を挿入した複数のシーンのプレビュー動画を生成・再生する機能を提供する(図 35)。この機能により、編集後のシーンが連続した映像としてどのように表示されるかを確認できる。

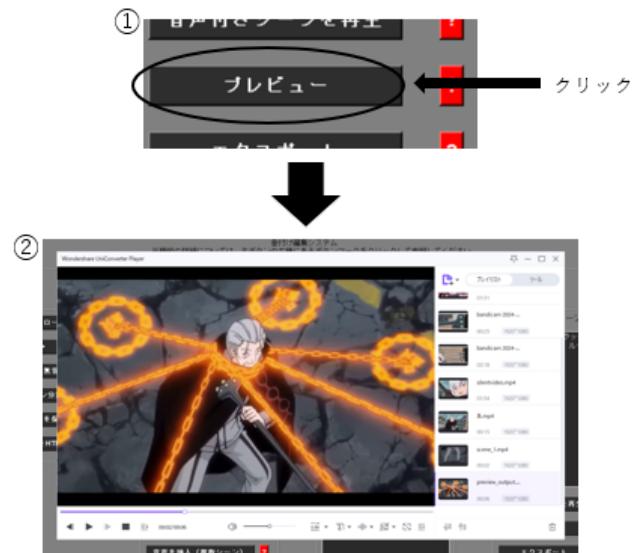


図 35: 効果音付き動画のプレビュー再生手順

- (1) ユーザーは、音付けされた映像のプレビューを開始するために「プレビュー」ボタンをクリックする。プレビュー対象のシーンが存在しない場合、警告メッセージが表示され、「プレビューするシーンがありません」と通知される。この際、処理は中断される。プレビューボタンが押下されると、プレビュー動画の生成される(図 35)。
- (2) プレビュー動画の生成後、システムのデフォルトプレーヤーを使用して動画が再生される(図 35)。

3.14 効果音付き動画のエクスポート

最後に、音付けが完了した動画をユーザー指定のフォルダへ MP4 形式でエクスポートする(図 36)。

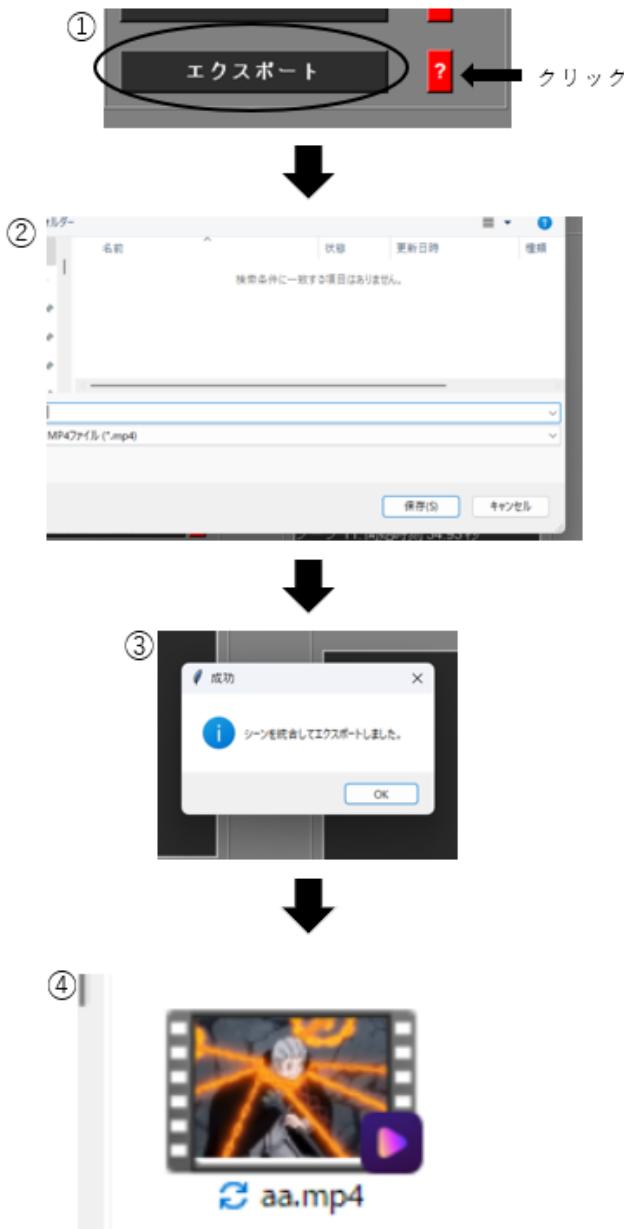


図 36: 効果音付き動画のエクスポート手順

- (1) 「エクスポート」ボタンをクリックすると、ファイル保存ダイアログが表示される(図 36)。
- (2) ユーザーはエクスポート後の動画ファイルの保存先ディレクトリおよびファイル名を指定する。デフォルトのファイル形式は MP4 である(図 36)。
- (3) エクスポート処理が正常に終了した場合、GUI 画面に「成功」のメッセージが表示される。一方、エラーが発生した場合は「エラー」メッセージが表示され、問題の詳細が提示される(図 36)。
- (4) エクスポートされたファイルは、指定されたディレクトリに保存されていることを確認できる(図 36)。

4. GUI システムの課題

本 GUI システムには、主に以下の 3 つの課題が存在する。

4.1 FFmpeg の処理時間

プレビュー動画を生成する際、複数のシーンを結合する FFmpeg 処理に時間がかかる場合があることである。特にシーン数が多い場合や個々のシーンファイルのサイズが大きい場合に、処理速度が低下する傾向が見られた。

4.2 リストファイルのエラー

2 つ目は、プレビュー動画生成時に一時的に作成されるリストファイルが、エラー発生時や予期せぬ中断で削除されない場合があることである。

4.3 複数の効果音の挿入

本 GUI システムでは、1 つのシーンに対して 1 つの効果音のみを挿入することができるが、音付け編集の現場では複数の効果音を同時に挿入する場合が多い。例えば、アクションシーンでは爆発音と銃声、背景音として風の音などが同時に再生されることが求められる場合がある。しかし、現行システムでは複数の効果音を 1 つのシーンに挿入する機能が実装されていない。

5. まとめと今後の展望

今年度は、無音にした映像に対する効果音の付加作業を効率化するための GUI システムを開発した。このシステムは、動画ファイルの読み込み、シーン分割、効果音検索、効果音再生、タイミングを指定した効果音挿入、効果音付きシーンのプレビュー再生、プレビューの生成、エクスポートを一連のプロセスとして実現している。

来年度以降は、「誰もが手軽に自身の音響のアイデアを映像へ反映できる環境」の実現を目指す。具体的には、シーンと効果音をタイムライン上で視覚的に配置・調整できる機能の実装や、効果音のパラメータ調整機能の追加を行い GUI をより直感的にする。また、AI を活用し、映像の内容やシーンの雰囲気を解析して、適切な効果音を自動で提案する機能や、ユーザーが指定した条件(例:「雨音」、「SF 的なエネルギー音」)に基づいて新しい効果音を生成する機能の実装などを目指す。このように、音付け編集のハードルを下げるだけでなく、創造的な映像制作の可能性を広げていきたいと考えている。

参考文献

- [1] Welcome to Python.org. <https://www.python.org/>.
- [2] tkinter — Python interface to Tel/Tk — Python 3.13.1 documentation. <https://docs.python.org/3/library/tkinter.html>.
- [3] python-ffmpeg · PyPI. <https://pypi.org/project/python-ffmpeg/>.
- [4] 効果音ラボ. <https://soundeffect-lab.info/>.
- [5] 【ワールドトリガー】空闊遊真戦闘シーン. <https://www.youtube.com/watch?v=i9DE8-rAPFY>.