Program Studi Teknik Informatika
Institut Teknologi Sumatera

Nama: **Vania(121140042), Natasya(121140052), Marchell(121140145)** Tugas Ke:**004**
Mata Kuliah: **Sistem/Teknologi Multimedia (IF4021)** Tanggal: 23 December 2024

1 Pendahuluan

Perkembangan teknologi multimedia telah membuka peluang baru dalam menciptakan pengalaman interaktif yang imersif. BLUBY hadir sebagai solusi inovatif yang mengintegrasikan computer vision dan pemrosesan audio untuk menghadirkan simulasi bawah laut yang interaktif. Dengan memanfaatkan teknologi OpenCV untuk manipulasi video real-time dan MediaPipe untuk deteksi fitur wajah, program ini mampu menghasilkan efek visual yang responsif terhadap gerakan pengguna. Sistem ini juga mengadopsi pemrosesan audio menggunakan Pydub/Librosa untuk menciptakan soundscape yang memperkuat pengalaman bawah laut. Keunikan BLUBY terletak pada kemampuannya menghasilkan gelembung dinamis yang merespons pergerakan mulut pengguna, menciptakan interaksi natural antara pengguna dan lingkungan virtual [1] [2] [3].

2 Deskripsi Sistem dan Algoritma

Sistem yang dikembangkan terdiri dari tiga komponen utama:

- **Deteksi Wajah dan Mulut**
  - Menggunakan Mediapipe Face Mesh untuk mendeteksi titik-titik fitur wajah, termasuk bibir atas dan bawah.
  - Implementasi algoritma Euclidean Distance untuk mendeteksi apakah mulut terbuka berdasarkan jarak antara bibir atas dan bawah.
- **Efek Audio**
  - Low-pass filter untuk memberikan efek suara lembut khas bawah air.
  - Penambahan reverb untuk menciptakan kesan ruang yang luas.
  - Audio delay untuk meningkatkan realisme

  Program ini memanfaatkan beberapa pustaka Python untuk fungsionalitas yang spesifik:
- **OpenCV** untuk menangkap video dari kamera dan menerapkan efek visual seperti distorsi bawah air.
- **MediaPipe** untuk mendeteksi fitur wajah, seperti bibir, yang digunakan untuk menentukan apakah mulut terbuka.
- **PyAudio** untuk memproses dan memodifikasi audio secara real-time, termasuk penambahan efek seperti low-pass filter, reverb, dan delay.
- **Pydub** untuk memutar efek suara latar dalam format MP3.

- **NumPy dan SciPy** untuk manipulasi data dan penerapan filter pada sinyal audio.

  Program ini bekerja dengan membaca umpan kamera menggunakan OpenCV. Setelah itu, wajah dianalisis untuk mendeteksi posisi dan apakah mulut terbuka, menggunakan MediaPipe. Di layar, efek visual seperti distorsi air dan gelembung ditambahkan ke video secara real-time. Sambil itu, ada audio latar yang diputar menggunakan Pydub, dan input suara dari mikrofon diproses dengan tambahan efek melalui PyAudio.

3 Desain Filter dan Pemrosesan Sinyal

3.1 Visual

- Membuat peta distorsi menggunakan fungsi sinusoidal.
- Menggunakan remap dari OpenCV untuk menerapkan peta distorsi ke setiap frame kamera.
- Penambahan gelembung dilakukan dengan memanfaatkan gambar PNG transparan yang di-overlay ke frame.

3.2 Audio

- **Low-pass Filter:**
  - Menggunakan filter Butterworth untuk membatasi frekuensi tinggi di atas 200 Hz, menciptakan efek suara yang lebih lembut.
- **Reverb:**
  - Menambahkan pantulan suara sederhana dengan delay sekitar 200 ms.
- **Delay:**
  - Menambahkan efek gema kecil untuk menambah kedalaman audio.

Penjelasan Kode

Kode berikut adalah program Python yang menggabungkan berbagai elemen seperti pemrosesan video, audio, dan efek visual untuk menciptakan simulasi interaktif bertema bawah air. Program ini juga dilengkapi fitur deteksi bibir terbuka menggunakan beberapa library populer seperti OpenCV, Mediapipe, dan Pydub. Setiap komponen dalam kode ini dirancang untuk bekerja sama, menghasilkan pengalaman interaktif yang unik dan menarik.

File: `background.py`

```
1
2  import cv2
3  import mediapipe as mp
4  import math
5  import time
6  import random
7  import numpy as np
8  import pyaudio
9  import scipy.signal as signal
10 from pydub import AudioSegment
11 from pydub.playback import play
12 import threading
```

```
13
```

Kode 1: Library yang digunakan

Adapun penjelasan library yang digunakan, yaitu:

- **cv**Untuk pemrosesan video dan gambar.
- **mediapipe** Digunakan untuk deteksi wajah dan fitur mesh wajah.
- **math** Untuk operasi matematika seperti menghitung jarak Euclidean.
- **time** Untuk mengatur waktu dan interval.
- **random** Untuk menghasilkan elemen acak (misalnya posisi gelembung).
- **numpy** Untuk manipulasi array.
- **pyaudio** Untuk streaming audio secara real-time.
- **scipy.signal** Untuk membuat filter audio.
- **pydub** Untuk memutar file audio MP3.
- **threading** Untuk menjalankan proses audio dan video secara paralel.

```
# Flag to indicate when to stop the application
stop_flag = threading.Event()

# Constants
SAMPLING_RATE = 44100
CHUNK_SIZE = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 1  # Mono audio
LOW_PASS_CUT_OFF = 200  # Low pass filter cutoff frequency (Hz)
Q_FACTOR = 0.71  # Quality factor for the low pass filter
REVERB_EFFECT_MIX = 0.5  # Reverb effect mix (50%)
REVERB_EFFECT_LEVEL = 8  # Level of reverb (in dB)
DELAY = 0.03  # Small delay in seconds (e.g., 30ms)

```

Kode 2: Konstanta Audio

Bagian kode ini menggunakan beberapa konstanta dan pengaturan untuk mengatur bagaimana audio diproses dan bagaimana alur aplikasi dikendalikan. Setiap konstanta punya peran penting, mulai dari mengatur kualitas suara hingga memastikan aplikasi berjalan sesuai yang diharapkan. Semuanya dirancang agar sistem dapat bekerja lebih efisien dan stabil.

- `stop_flag = threading.Event()`: Ini adalah objek yang digunakan untuk mengontrol aliran eksekusi dalam program multithreading. `stop_flag` digunakan untuk memberi sinyal bahwa aplikasi atau thread tertentu harus berhenti. Misalnya, pada akhir pemrosesan audio atau video, `stop_flag` dapat diatur untuk menghentikan stream atau proses lainnya.
- `SAMPLING_RATE = 44100`: Ini adalah laju pengambilan sampel (sampling rate) untuk pemrosesan audio. Nilai 44100 Hz adalah standar dalam audio digital, yang berarti 44.100 sampel diambil setiap detik. Ini adalah kualitas audio yang umum digunakan dalam musik dan rekaman suara.
- `CHUNK_SIZE = 1024`: Ini menentukan ukuran buffer audio yang diproses pada setiap iterasi. `CHUNK_SIZE` mengatur jumlah sampel audio yang diambil dan diproses dalam satu waktu. Dengan ukuran 1024 sampel per chunk, proses audio lebih efisien dan responsif.

- `FORMAT = pyaudio.paInt16`: Menentukan format data audio yang digunakan. `pyaudio.paInt16` berarti data audio yang digunakan dalam pemrosesan adalah 16-bit integer, yang umum dalam rekaman audio standar dengan kualitas yang baik.
- `CHANNELS = 1`: Ini mengatur jumlah saluran audio. Nilai 1 berarti audio yang diproses adalah audio mono, yang hanya memiliki satu saluran. Dalam konteks ini, ini berarti suara akan diproses dalam format mono, tidak stereo.
- `LOW_PASS_CUT_OFF = 200`: Ini adalah frekuensi cutoff untuk filter low-pass yang diterapkan pada audio. Filter low-pass digunakan untuk mengurangi frekuensi tinggi yang tidak diinginkan. Dengan cutoff di 200 Hz, hanya frekuensi di bawah 200 Hz yang akan diteruskan, sementara frekuensi lebih tinggi akan disaring.
- `Q_FACTOR = 0.71`: Ini adalah faktor kualitas untuk filter low-pass. Faktor ini mempengaruhi lebar pita filter, yang menentukan seberapa tajam atau halus filter tersebut. Nilai 0.71 menghasilkan filter yang tidak terlalu tajam, memberikan respons yang lebih alami.
- `REVERB_EFFECT_MIX = 0.5`: Ini mengontrol seberapa kuat efek reverb (gaung) yang dicampurkan ke dalam suara asli. Nilai 0.5 berarti campuran yang setara antara suara asli dan efek reverb, sehingga efek tersebut tidak terlalu dominan.
- `REVERB_EFFECT_LEVEL = 8`: Ini adalah level kekuatan reverb dalam desibel (dB). Nilai 8 dB menunjukkan kekuatan reverb yang cukup kuat, menambah kedalaman dan ruang pada suara.
- `DELAY = 0.03`: Ini adalah waktu tunda (delay) yang diterapkan pada audio dalam detik. Nilai 0.03 berarti ada penundaan sekitar 30 milidetik antara suara yang dimainkan dan efek delay yang diterapkan, menciptakan efek echo yang lembut.

Secara keseluruhan, konstanta-konstanta ini sebenarnya berfungsi untuk mengatur cara aplikasi memproses audio, mulai dari pengaturan efek suara seperti filter low-pass, reverb, hingga delay. Selain itu, konstanta ini juga digunakan untuk mengontrol agar aplikasi bisa berjalan lebih efisien dengan memanfaatkan thread, sehingga prosesnya bisa dilakukan secara paralel.

```
# Disable TensorFlow and Mediapipe logs
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # '2' to show only errors, '3' to disable logs completely

# Initialize Mediapipe Face Mesh
mp_face_mesh = mp.solutions.face_mesh
mp_drawing = mp.solutions.drawing_utils
face_mesh = mp_face_mesh.FaceMesh(min_detection_confidence=0.2, min_tracking_confidence=0.2)

# Create PyAudio instance
p = pyaudio.PyAudio()

# Load the MP3 file using pydub
mp3_file = AudioSegment.from_mp3("underwater-ambience-heavy-rumbling-ftus-1-00-17.mp3")
mp3_file = mp3_file.set_frame_rate(SAMPLING_RATE).set_channels(1).set_sample_width(2)  # Convert to
    mono, 16-bit


# Function to calculate Euclidean distance between two points
def euclidean_distance(p1, p2):
    return math.sqrt((p2[0] - p1[0]) ** 2 + (p2[1] - p1[1]) ** 2)

```

Kode 3: Mediapipe dan PyAudio

Pada bagian kode ini, beberapa pengaturan dan inisialisasi dilakukan untuk mempersiapkan aplikasi dalam memproses audio dan video. Pertama, log dari TensorFlow dan Mediapipe dimatikan dengan mengatur variabel lingkungan `TF_CPP_MIN_LOG_LEVEL` ke nilai '2', yang hanya menampilkan pesan kesalahan, atau '3' untuk menonaktifkan log sepenuhnya. Ini bertujuan untuk membuat output lebih bersih tanpa pesan log yang tidak diperlukan. Selanjutnya, Mediapipe diinisialisasi untuk mendeteksi dan melacak wajah menggunakan `FaceMesh`, dengan pengaturan untuk memastikan deteksi dan pelacakan wajah hanya dilakukan jika kepercayaan deteksi dan pelacakan mencapai ambang batas tertentu (0.2). Kemudian, objek `pyaudio.PyAudio()` dibuat untuk menangani pemrosesan audio. Terakhir, file audio MP3 dimuat menggunakan pustaka `pydub`, lalu disetel untuk memiliki laju pengambilan sampel (sampling rate) 44100 Hz, dikonversi menjadi mono, dan disesuaikan dengan lebar sampel 16-bit, memastikan kompatibilitas dengan pemrosesan audio yang lebih lanjut dalam aplikasi ini.

```python
# Function to detect if the mouth is open
def is_mouth_open(landmarks):
    upper_lip = landmarks[13]  # Upper lip point
    lower_lip = landmarks[14]  # Lower lip point
    distance = euclidean_distance((upper_lip.x, upper_lip.y), (lower_lip.x, lower_lip.y))
    threshold = 0.02  # Threshold for detecting open mouth
    return distance > threshold

# Function to overlay PNG image (bubble) onto the frame
def overlay_png(frame, bubble_png, x, y, size):
    bubble_resized = cv2.resize(bubble_png, (size, size), interpolation=cv2.INTER_AREA)
    b_h, b_w, _ = bubble_resized.shape
    alpha = bubble_resized[:, :, 3] / 255.0
    bubble_rgb = bubble_resized[:, :, :3]

    y1, y2 = max(0, y - b_h // 2), min(frame.shape[0], y + b_h // 2)
    x1, x2 = max(0, x - b_w // 2), min(frame.shape[1], x + b_w // 2)
    b_y1 = max(0, -y + b_h // 2)
    b_y2 = b_y1 + (y2 - y1)
    b_x1 = max(0, -x + b_w // 2)
    b_x2 = b_x1 + (x2 - x1)

    if y1 < y2 and x1 < x2 and b_y1 < b_y2 and b_x1 < b_x2:
        for c in range(3):
            frame[y1:y2, x1:x2, c] = (
                frame[y1:y2, x1:x2, c] * (1 - alpha[b_y1:b_y2, b_x1:b_x2]) +
                bubble_rgb[b_y1:b_y2, b_x1:b_x2, c] * alpha[b_y1:b_y2, b_x1:b_x2]
            )
```

Kode 4: Landmark with euclidean distance

Bagian kode ini berisi tiga fungsi yang masing-masing berfungsi untuk menghitung jarak Euclidean antara dua titik, mendeteksi apakah mulut terbuka, dan menampilkan gambar PNG (seperti gelembung) pada frame video.

Fungsi pertama, `euclidean_distance`, digunakan untuk menghitung jarak antara dua titik dalam ruang dua dimensi. Fungsi ini menerima dua parameter, yaitu `p1` dan `p2`, yang masing-masing adalah koordinat titik dalam bentuk tuple (x, y). Fungsi ini menghitung selisih antara kedua titik pada sumbu x dan y, kemudian menerapkan rumus Pythagoras untuk mendapatkan jarak antara keduanya.

Fungsi kedua, `is_mouth_open`, bertugas mendeteksi apakah mulut terbuka atau tidak berdasarkan posisi titik-titik wajah yang dikenali oleh model FaceMesh. Fungsi ini menerima parameter `landmarks`, yang berisi koordinat titik-titik wajah. Dalam hal ini, titik yang digunakan untuk mendeteksi keadaan mulut terbuka adalah titik pada bibir atas (`landmarks[13]`) dan bibir bawah (`landmarks[14]`). Fungsi ini menghitung jarak antara kedua titik tersebut menggunakan fungsi `euclidean_distance` dan membandingkannya dengan ambang batas (threshold) 0.02 untuk menentukan apakah mulut terbuka (jika jaraknya lebih besar dari threshold).

Fungsi terakhir, `overlay_png`, digunakan untuk menampilkan gambar PNG dengan transparansi

(misalnya, gelembung teks) di atas frame video. Fungsi ini menerima beberapa parameter, termasuk `frame` (gambar video), `bubble_png` (gambar PNG yang akan ditampilkan), `x` dan `y` (posisi gambar), dan `size` (ukuran gambar). Gambar PNG pertama-tama diubah ukurannya sesuai dengan parameter `size`, kemudian transparansi gambar ditangani menggunakan saluran alpha dari gambar PNG. Selanjutnya, posisi gambar di dalam frame dihitung dengan memastikan gambar tidak keluar dari batas frame, dan akhirnya gambar ditambahkan ke frame video dengan memadukan warna dari gambar dan frame menggunakan saluran alpha untuk efek transparansi yang halus.

Secara keseluruhan, fungsi-fungsi ini bekerja bersama untuk mendeteksi gerakan mulut dan menampilkan efek visual dalam aplikasi pengolahan video atau pengenalan wajah secara real-time.

```
# Bubble class to manage individual bubbles
class Bubble:
    def __init__(self, x, y, radius, delay):
        self.x = x
        self.y = y
        self.radius = radius
        self.delay = delay  # Time delay for the bubble to appear
        self.start_time = time.time()  # Record the start time
        self.active = False  # Whether the bubble is active/visible

    def move(self):
        if not self.active and (time.time() - self.start_time) > self.delay:
            self.active = True
        if self.active:
            self.y -= random.randint(1, 5)
            if self.y < -self.radius:
                self.reset()

    def reset(self):
        self.y = random.randint(480, 500)
        self.x = random.randint(0, 640)
        self.radius = random.randint(5, 15)
        self.delay = random.uniform(0.5, 3)
        self.start_time = time.time()
        self.active = False
```

Kode 5: Inisiasi Pembuatan bubble

Kode ini mendefinisikan kelas `Bubble`, yang digunakan untuk mengelola gelembung individual dalam efek visual, seperti yang mungkin terlihat dalam simulasi gelembung terapung di dalam air. Kelas ini memiliki beberapa atribut dan metode untuk mengontrol perilaku gelembung.

Pada bagian konstruktor , atribut `x` dan `y` menetapkan posisi awal gelembung di layar, sementara `radius` menentukan ukuran gelembung dan `delay` mengatur waktu tunda sebelum gelembung muncul. Atribut `start_time` mencatat waktu saat objek `Bubble` dibuat, dan `active` menunjukkan apakah gelembung tersebut aktif atau terlihat.

Metode `move` bertanggung jawab untuk menggerakkan gelembung di layar. Jika gelembung belum aktif dan waktu tunda telah tercapai (dihitung dengan membandingkan waktu saat ini dengan `start_time`), maka gelembung akan menjadi aktif. Setelah aktif, gelembung bergerak ke atas dengan kecepatan acak (menggunakan fungsi `random.randint`). Jika gelembung bergerak keluar dari layar (`y < -radius`), maka gelembung akan direset menggunakan metode `reset`.

Metode `reset` mengatur ulang posisi dan ukuran gelembung secara acak, serta menetapkan waktu tunda baru untuk muncul kembali. Setelah reset, gelembung menjadi tidak aktif kembali dan menunggu hingga waktu tunda tercapai untuk muncul lagi.

Secara keseluruhan, kelas `Bubble` ini memungkinkan simulasi gelembung yang muncul secara acak, bergerak ke atas, dan direset setelah keluar dari layar, menciptakan efek visual dinamis.

```
# Apply the underwater effect with floating bubbles and more pronounced camera distortion
def add_underwater_effect(frame, bubbles, bubble_png, time_factor):
```

```
    overlay = frame.copy()

    # Apply a basic blue filter: Boosting the blue channel
    overlay[:, :, 0] = cv2.add(overlay[:, :, 0], 100)  # Enhance blue (B)
    overlay[:, :, 1] = cv2.subtract(overlay[:, :, 1], 50)  # Reduce green (G)
    overlay[:, :, 2] = cv2.subtract(overlay[:, :, 2], 50)  # Reduce red (R)

    # Ensure that values stay within the valid range (0-255)
    overlay = np.clip(overlay, 0, 255)

    # Create the distortion effect using a slow, sinusoidal wave
    rows, cols, _ = frame.shape
    distortion_map_x = np.tile(np.linspace(0, cols - 1, cols), (rows, 1)).astype(np.float32)
    distortion_map_y = np.tile(np.linspace(0, rows - 1, rows), (cols, 1)).T.astype(np.float32)

    # Sinusoidal distortion (more pronounced wave effect)
    wave_amplitude = 15  # Increase amplitude for more pronounced distortion
    wave_frequency = 0.01  # Frequency of the wave
    wave_speed = 0.05  # Speed of the wave over time (keep the same speed)

    distortion_map_x += wave_amplitude * np.sin(wave_frequency * distortion_map_y + time_factor *
    wave_speed).astype(np.float32)
    distortion_map_y += wave_amplitude * np.cos(wave_frequency * distortion_map_x + time_factor *
    wave_speed).astype(np.float32)

    # Apply the distortion map to the frame
    distorted_frame = cv2.remap(overlay, distortion_map_x, distortion_map_y, interpolation=cv2.
    INTER_LINEAR)

    # Add bubbles to the distorted frame
    for bubble in bubbles:
        bubble.move()
        if bubble.active:
            overlay_png(distorted_frame, bubble_png, bubble.x, bubble.y, bubble.radius * 2)

    # Apply blur to the frame after adding bubbles
    distorted_frame = cv2.GaussianBlur(distorted_frame, (5, 5), 0)

    return distorted_frame
```

Kode 6: Underwater effect (floatting bubbles

Fungsi `add_underwater_effect` digunakan untuk menambahkan efek visual yang menyerupai pemandangan di bawah air pada sebuah frame gambar. Pertama, fungsi ini membuat salinan dari frame asli untuk diproses lebih lanjut. Kemudian, fungsi ini menerapkan filter biru dasar pada gambar dengan meningkatkan saluran biru (`blue channel`) dan mengurangi saluran hijau (`green`) serta merah (`red`), menciptakan nuansa biru yang lebih dominan, yang sering dikaitkan dengan efek visual di bawah air. Setelah itu, fungsi memastikan bahwa nilai warna dalam gambar tetap dalam rentang yang valid (0-255) menggunakan `np.clip`.

Selanjutnya, fungsi ini membuat efek distorsi menggunakan gelombang sinusoidal yang bergerak perlahan. Dengan memanipulasi peta distorsi (`distortion_map_x` dan `distortion_map_y`), gelombang sinusoidal ini menghasilkan efek distorsi visual yang menyerupai gelombang air. Amplitudo, frekuensi, dan kecepatan gelombang diatur untuk memberikan efek distorsi yang lebih nyata dan dinamis. Distorsi ini diterapkan ke frame menggunakan fungsi `cv2.remap`.

Setelah itu, gelembung-gelembung ditambahkan ke frame yang sudah terdistorsi. Untuk setiap objek gelembung dalam daftar `bubbles`, posisi dan status aktifnya diperiksa, dan jika gelembung aktif, gambar PNG gelembung (yang ditangani oleh fungsi `overlay_png`) akan ditempatkan pada posisi yang sesuai di frame. Terakhir, efek blur diterapkan pada frame yang telah dimodifikasi dengan menggunakan Gaussian blur untuk memberikan efek kabur yang lebih halus pada gambar.

Secara keseluruhan, fungsi ini menciptakan efek visual yang realistis, termasuk distorsi gelombang dan gelembung terapung, yang memberikan kesan visual seperti berada di bawah air.

```
# Low-pass filter function
def low_pass_filter(data, cutoff, fs, Q):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = signal.butter(1, normal_cutoff, btype='low', analog=False)
    return signal.filtfilt(b, a, data)

# Manual Reverb Effect (simple echo effect)
def add_reverb(audio, mix, level, fs):
    delay_samples = int(0.2 * fs)  # 200ms delay for reverb effect
    reverb_audio = np.zeros_like(audio)

    reverb_audio[delay_samples:] = audio[:-delay_samples]

    return mix * reverb_audio + (1 - mix) * audio

# Audio delay effect
def apply_delay(audio, delay_time, fs):
    delay_samples = int(delay_time * fs)
    delayed_audio = np.zeros_like(audio)
    delayed_audio[delay_samples:] = audio[:-delay_samples]
    return audio + delayed_audio

# Function to play the MP3 file in a separate thread
def play_bubbling_sound():
    while not stop_flag.is_set():  # Check for the stop flag
        play(mp3_file)  # This will loop the bubbling sound indefinitely


# Stream callback function
def callback(in_data, frame_count, time_info, status):
    audio_data = np.frombuffer(in_data, dtype=np.int16).astype(np.float32)
    filtered_audio = low_pass_filter(audio_data, LOW_PASS_CUT_OFF, SAMPLING_RATE, Q_FACTOR)
    reverb_audio = add_reverb(filtered_audio, REVERB_EFFECT_MIX, REVERB_EFFECT_LEVEL, SAMPLING_RATE
    )
    final_audio = apply_delay(reverb_audio, DELAY, SAMPLING_RATE)
    out_data = final_audio.astype(np.int16).tobytes()
    return (out_data, pyaudio.paContinue)

# Function to process audio
def process_audio():
    # Open PyAudio stream
    stream = p.open(format=FORMAT,
                    channels=CHANNELS,
                    rate=SAMPLING_RATE,
                    input=True,
                    output=True,
                    frames_per_buffer=CHUNK_SIZE,
                    stream_callback=callback)
    # Start the audio stream
    stream.start_stream()

    # Keep the stream running until the stop flag is set
    while not stop_flag.is_set():
        time.sleep(0.1)

    # Stop the stream and close PyAudio
    stream.stop_stream()
    stream.close()
```

```
    p.terminate()
```

Kode 7: Audio Processing

Kode yang diberikan mendefinisikan beberapa fungsi untuk efek pemrosesan audio, termasuk filter low-pass, reverb, dan efek delay, bersama dengan fungsionalitas untuk memutar dan memproses audio menggunakan pustaka PyAudio.

Fungsi pertama, `low_pass_filter`, menerapkan filter low-pass pada sinyal audio input. Fungsi ini menerima data audio, frekuensi cutoff, sampling rate (`fs`), dan faktor kualitas (`Q`) sebagai input. Frekuensi Nyquist dihitung, dan frekuensi cutoff dinormalisasi. Menggunakan metode `butter` dari modul `scipy.signal`, filter low-pass dirancang, dan fungsi `filtfilt` digunakan untuk menerapkan filter ini pada data audio, menghilangkan komponen frekuensi tinggi di atas cutoff.

Fungsi kedua, `add_reverb`, mengimplementasikan efek reverb (gema) sederhana dengan memperkenalkan penundaan pada sinyal audio. Fungsi ini menggeser audio dengan penundaan yang telah ditentukan ($0.2 \times fs$ atau 200 milidetik), lalu mencampur sinyal asli dengan sinyal yang ditunda. Rasio campuran antara audio asli dan yang ditunda dikendalikan oleh parameter `mix`, dan parameter `level` menentukan kekuatan efek tersebut.

Fungsi `apply_delay` menciptakan efek delay audio dengan menggeser audio input berdasarkan waktu penundaan yang ditentukan dalam detik (`delay_time`). Fungsi ini membuat versi audio yang tertunda dan menambahkannya ke audio asli, menghasilkan efek delay.

Fungsi `play_bubbling_sound` dirancang untuk memutar file MP3 di thread terpisah. Fungsi ini secara terus-menerus memutar suara gelembung hingga `stop_flag` disetel, memastikan bahwa audio diputar di latar belakang tanpa memblokir operasi lainnya.

Fungsi `callback` digunakan oleh PyAudio untuk memproses data audio secara real-time. Fungsi ini menerima data audio input, menerapkan filter low-pass, efek reverb, dan efek delay secara berurutan, lalu mengonversi audio yang diproses kembali ke format yang sesuai untuk output. Audio yang diproses dikembalikan dalam bentuk byte untuk diputar.

Terakhir, fungsi `process_audio` mengatur aliran audio PyAudio untuk pemrosesan audio secara real-time. Fungsi ini membuka aliran audio untuk input dan output, dengan fungsi callback yang memproses audio. Aliran dimulai dan tetap berjalan hingga `stop_flag` disetel, setelah itu aliran dihentikan dan ditutup.

Secara keseluruhan, kode ini menyediakan jalur pemrosesan audio secara real-time yang menerapkan filter low-pass, reverb, dan efek delay pada aliran audio input sambil memutar suara gelembung di latar belakang.

```
# Start the video and audio processing
# Function to process video and handle mouth detection
def face_mesh_mouth_detection_with_audio():
    # Start the audio processing thread
    audio_thread = threading.Thread(target=process_audio)
    audio_thread.start()

    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Failed to open camera!")
        stop_flag.set()  # Signal to stop the audio thread
        return

    # Load assets from the 'assets/' directory
    bubble_png = cv2.imread('assets/bubble.png', cv2.IMREAD_UNCHANGED)

    bubbles = [Bubble(random.randint(0, 640), random.randint(480, 500),
                random.randint(5, 15), random.uniform(0.5, 3)) for _ in range(30)]
    time_factor = 0
    last_bubble_time = 0
    bubble_delay = 1
```

```
22
23      while cap.isOpened():
24          success, frame = cap.read()
25          if not success:
26              print("Failed to read frame!")
27              break
28
29          frame = cv2.flip(frame, 1)  # Flip the frame horizontally
30
31          # Removed the background change (background image is no longer applied)
32
33          frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
34          results = face_mesh.process(frame_rgb)
35
36          if results.multi_face_landmarks:
37              for face_landmarks in results.multi_face_landmarks:
38
39                  landmarks = face_landmarks.landmark
40                  if is_mouth_open(landmarks):
41                      upper_lip = landmarks[13]
42                      lower_lip = landmarks[14]
43                      mouth_center = ((upper_lip.x + lower_lip.x) / 2,
44                                      (upper_lip.y + lower_lip.y) / 2)
45                      current_time = time.time()
46                      if current_time - last_bubble_time > bubble_delay:
47                          bubbles.append(Bubble(
48                              int(mouth_center[0] * frame.shape[1]),
49                              int(mouth_center[1] * frame.shape[0]),
50                              40, random.uniform(0.5, 3)))
51                          last_bubble_time = current_time
52                      cv2.putText(frame, "Mouth Open", (50, 50),
53                                  cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
54                  else:
55                      cv2.putText(frame, "Mouth Closed", (50, 50),
56                                  cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
57
58          # Add underwater effect (bubbles)
59          underwater_frame = add_underwater_effect(frame, bubbles, bubble_png, time_factor)
60          time_factor += 0.1
61
62          cv2.imshow("Face Mesh - Mouth Detection with Bubbles", underwater_frame)
63
64          # Exit if 'q' is pressed
65          if cv2.waitKey(1) & 0xFF == ord('q'):
66              stop_flag.set()  # Signal to stop the audio thread
67              break
68
69      cap.release()
70      cv2.destroyAllWindows()
71      stop_flag.set()  # Ensure the audio thread stops if it hasn't already
72
73  # Start the MP3 playback thread
74  bubbling_thread = threading.Thread(target=play_bubbling_sound)
75  bubbling_thread.start()
76
77  # Start the video processing
78  face_mesh_mouth_detection_with_audio()
79
80
```

Kode 8: Final Video and Audio Processing

Kode yang diberikan menggabungkan proses video dan audio, serta menggunakan deteksi wajah untuk

memeriksa apakah mulut seseorang terbuka. Jika mulut terbuka, efek gelembung akan muncul di layar. Dengan kata lain, kode ini memanfaatkan teknologi untuk menganalisis gambar dan suara secara bersamaan, lalu menampilkan animasi berbentuk gelembung berdasarkan kondisi tertentu, seperti apakah mulut terbuka atau tidak.

Thread Audio

Dimulai dengan memulai thread `process_audio` untuk memproses aliran audio yang berjalan secara terpisah. Ini memastikan bahwa pemrosesan audio seperti filter low-pass, reverb, dan efek delay tetap berjalan tanpa menghalangi pemrosesan video.

Pengaturan Kamera

Dibuka koneksi ke kamera menggunakan OpenCV untuk menangkap video dari kamera default (`0`). Jika kamera gagal dibuka, proses audio dihentikan dengan menyetel `stop_flag`.

Asset Gelembung

Gambar PNG yang berisi gelembung dalam folder 'assets/' dibaca menggunakan OpenCV. Setiap gelembung yang muncul kemudian didefinisikan melalui sebuah daftar yang mencatat atribut penting, seperti posisi, ukuran, dan waktu penundaan kemunculannya. Hal ini memberikan kontrol lebih atas bagaimana setiap gelembung ditampilkan, memungkinkan untuk penyesuaian waktu dan posisi secara dinamis. Pendekatan ini cukup fleksibel, namun membutuhkan pemahaman yang baik tentang pengolahan citra dan koordinasi waktu agar hasil yang diinginkan dapat tercapai dengan baik.

Loop Pemrosesan Frame

Setiap frame video diproses dengan membaliknya secara horizontal, lalu diubah ke format RGB agar dapat dikenali oleh model face mesh dari Google MediaPipe. Proses ini memastikan model bekerja optimal, tetapi juga menambah beban komputasi, terutama jika dilakukan secara real-time. Penting untuk mempertimbangkan efisiensi algoritma ini, terutama untuk aplikasi yang membutuhkan respons cepat, seperti pengenalan wajah atau augmented reality.

Deteksi Mulut Terbuka

Jika wajah terdeteksi dan mulut terbuka, sistem akan menghitung koordinat titik tengah mulut, lalu menambahkan gelembung baru setelah jeda tertentu. Saat gelembung ditambahkan, teks "Mulut Terbuka" akan ditampilkan di layar dengan warna hijau. Sebaliknya, jika mulut tertutup, teks "Mulut Tertutup" akan muncul dengan warna merah. Mekanisme ini tidak hanya sekadar memberikan visualisasi, tetapi juga memungkinkan sistem untuk secara langsung menunjukkan status mulut secara real-time. Pendekatan seperti ini cukup efektif untuk menyoroti perubahan ekspresi, meskipun ada potensi pengembangan lebih lanjut untuk menangkap detail yang lebih kompleks.

Efek Underwater

Fungsi `add_underwater_effect` digunakan untuk menerapkan efek gelembung dengan distorsi dan filter visual lainnya ke frame video.

Menampilkan Hasil

Frame yang telah diproses ditampilkan dalam jendela yang bernama "Face Mesh - Mouth Detection with Bubbles". Program keluar jika pengguna menekan tombol 'q' atau jika kamera gagal menangkap frame.

Thread Pemutaran Suara

Thread terpisah `play_bubbling_sound` dimulai untuk memutar suara gelembung di latar belakang.

Hentian dan Cleanup

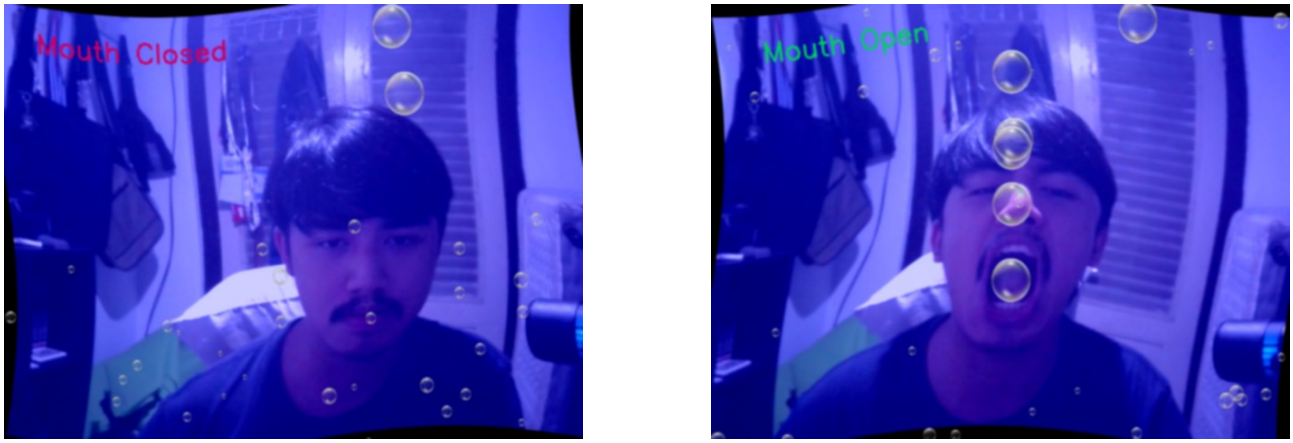
Setelah loop pemrosesan selesai, aliran kamera dilepas, jendela OpenCV dihancurkan, dan `stop_flag` disetel untuk memastikan bahwa thread pemrosesan audio juga dihentikan.

4 Analisis Hasil

Hasil yang diperoleh menunjukkan bahwa:

- Sistem mampu mendeteksi mulut terbuka secara *real-time* dengan akurasi tinggi.
- Efek visual *underwater* berhasil menciptakan pengalaman yang realistis, dengan distorsi yang dinamis dan gelembung yang bergerak alami.
- Efek audio memberikan kesan ruang bawah air yang imersif, meningkatkan pengalaman pengguna secara keseluruhan.

Uji coba dilakukan pada perangkat dengan spesifikasi menengah dan hasilnya sistem dapat berjalan dengan baik tanpa *lag* yang signifikan.



Gambar 1: Penggunaan Filter Blubby

5 Kesimpulan

Secara keseluruhan, proyek ini berhasil memadukan teknologi deteksi wajah, pemrosesan visual, dan audio untuk menciptakan sebuah sistem multimedia interaktif yang terasa hidup dan menarik. Meskipun hasilnya cukup menjanjikan, masih ada ruang untuk perbaikan, terutama dalam hal kecepatan respons dan akurasi deteksi pada kondisi cahaya atau lingkungan yang kurang ideal. Ke depan, sistem ini memiliki potensi besar untuk diterapkan dalam berbagai bidang, seperti hiburan, pendidikan, atau bahkan terapi, asalkan pengembangannya terus disempurnakan sesuai kebutuhan pengguna nyata.

6 Lampiran (Kode)

Untuk program secara keseluruhan, dapat diakses melalui Blubby Repository

7 Referensi

- Dokumentasi Mediapipe: https://google.github.io/mediapipe
- Dokumentasi OpenCV: https://docs.opencv.org
- Dokumentasi Pyaudio: https://people.csail.mit.edu/hubert/pyaudio
- Link GPT: https://chatgpt.com/share/676a7b64-4760-800c-a4d4-5bdbe5a2896c

References

[1] H. C. Jian-Guang Lou and J. Li, "Real-time interactive multimedia effects in video streams," https://www.researchgate.net/publication/221571284_A_real-time_interactive_multi-view_video_system, 2023.

[2] Y. Z. J. J. Q. X. Kai Hu, Chenghang Weng, "An overview of underwater vision enhancement: From traditional methods to recent deep learning," https://www.mdpi.com/2077-1312/10/2/241, 2022.

[3] T. Chen and R. Rao, "Audio-visual synchronization in interactive multimedia applications," https://ieeexplore.ieee.org/document/476612, 1995.