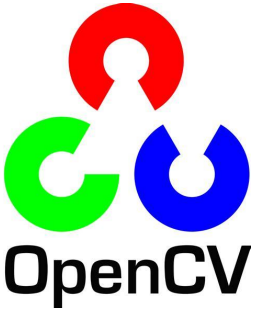
CIS25 FINAL

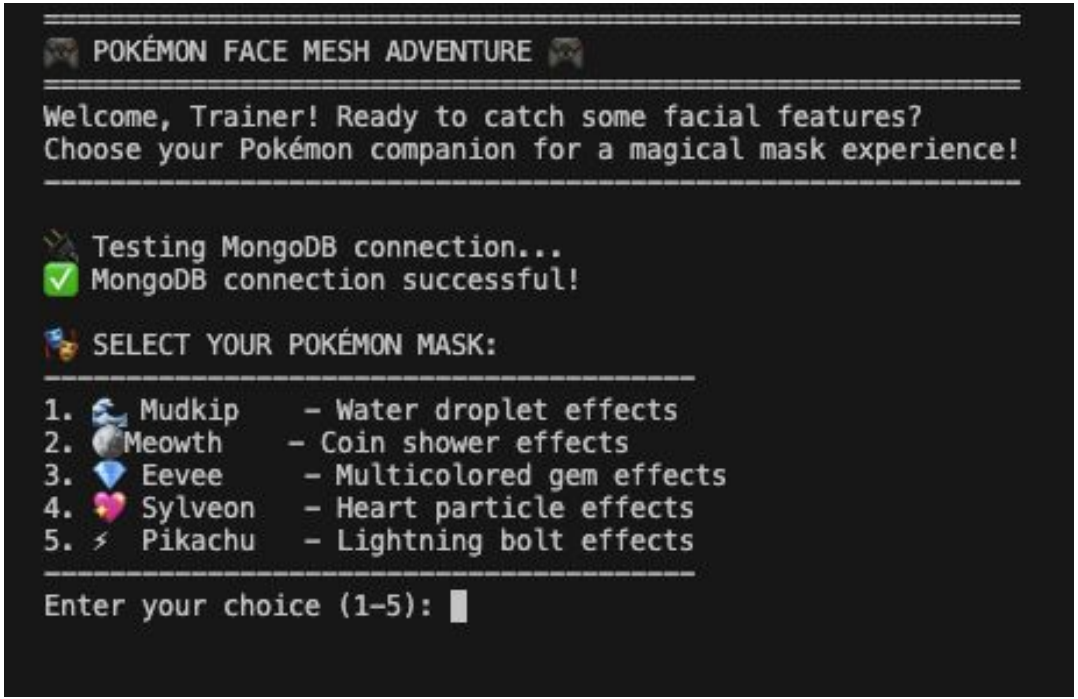
By Luis Ibarra

Pokemon Face Mesh Adventure

The **Pokémon Face Mesh Adventure** is an interactive C++ application that combines real-time computer vision, object-oriented programming, and database integration. The application uses OpenCV for face detection, applies Pokémon-themed mask overlays, generates dynamic particle effects triggered by mouth movement, and stores user data in MongoDB.



Welcome Screen + GUI



- Initializes MongoDB storage to save screenshots.

-Prompts user to select 1 of 5 pokemon to create a mask.

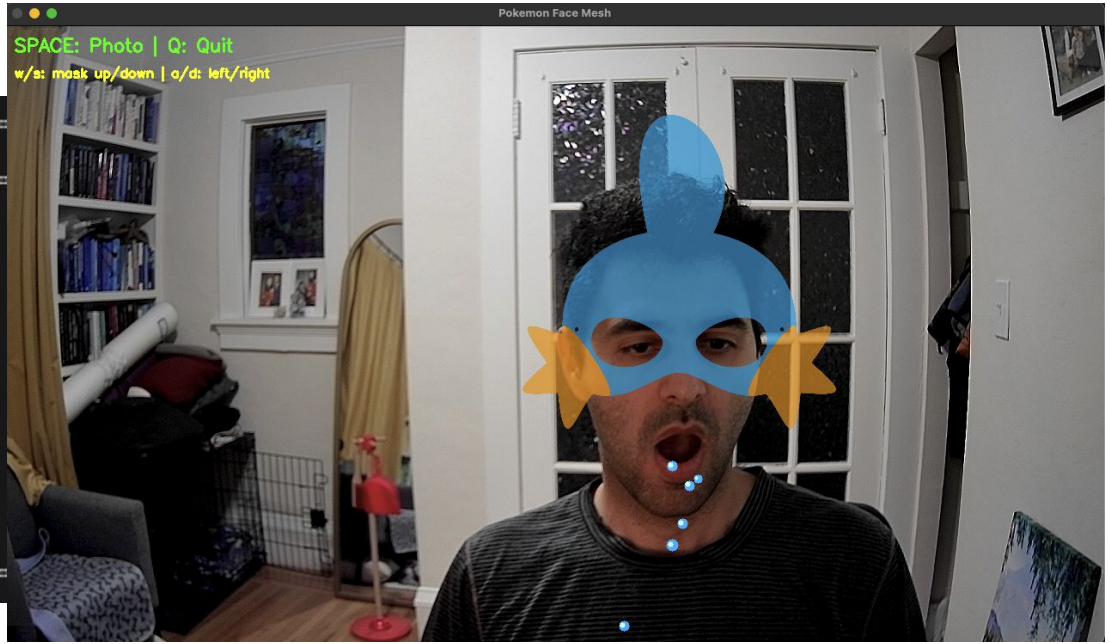
Selecting your Mask + Particle Attack

-OpenCV loads face and mouth tracking algorithm.
-OpenCV creates new window and access camera.
-Loads chosen mask from local storage

Real-Time Window

```
FACE MESH & MASK OVERLAY APP

REAL CAMERA MODE:
  SPACE or ENTER - Take photo with mask overlay
INFO:
  'i' or 'I' - Show captured faces database
  'S' - Show app statistics
MASK CONTROLS:
  w/s - Move mask up/down
  a/d - Move mask left/right
  r - Reset mask position
MOUTH DETECTION:
  Open/close mouth to see emoji changes
EXIT:
  'q' or ESC - Quit application
```

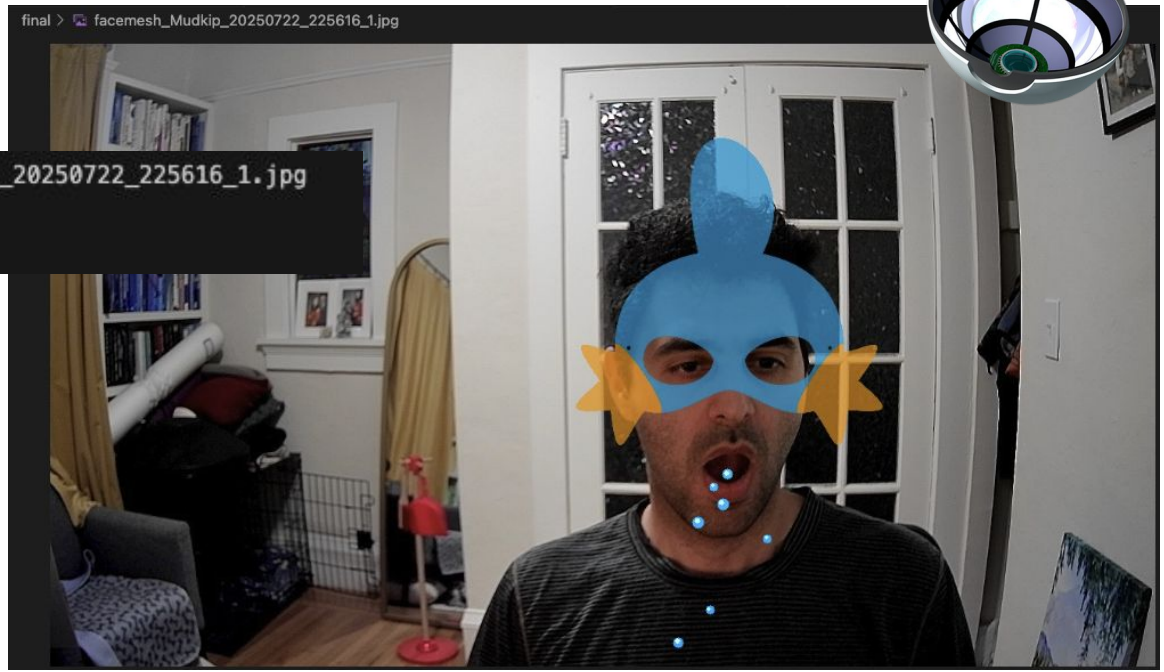


-OpenCV draws mask in real-time.
-Mask can be adjusted with keyboard.
-OpenCV draws particles from mouth in real time.

```
Mask moved up. Vertical offset: -0.05
Mask moved up. Vertical offset: -0.1
Mask moved up. Vertical offset: -0.15
Mask moved up. Vertical offset: -0.2
Mask moved up. Vertical offset: -0.25
Mask moved up. Vertical offset: -0.3
Mask moved down. Vertical offset: -0.25
Mask moved up. Vertical offset: -0.3
Mask moved down. Vertical offset: -0.25
```

Saving Screenshots to MongoDB

```
Face capture saved: facemesh_Mudkip_20250722_225616_1.jpg
  Faces detected: 1
  Data saved to MongoDB
```



-Pressing SPACE saves screenshot to both MongoDB and local folder.

Gotta Try ‘Em All!

Eevee + Gems

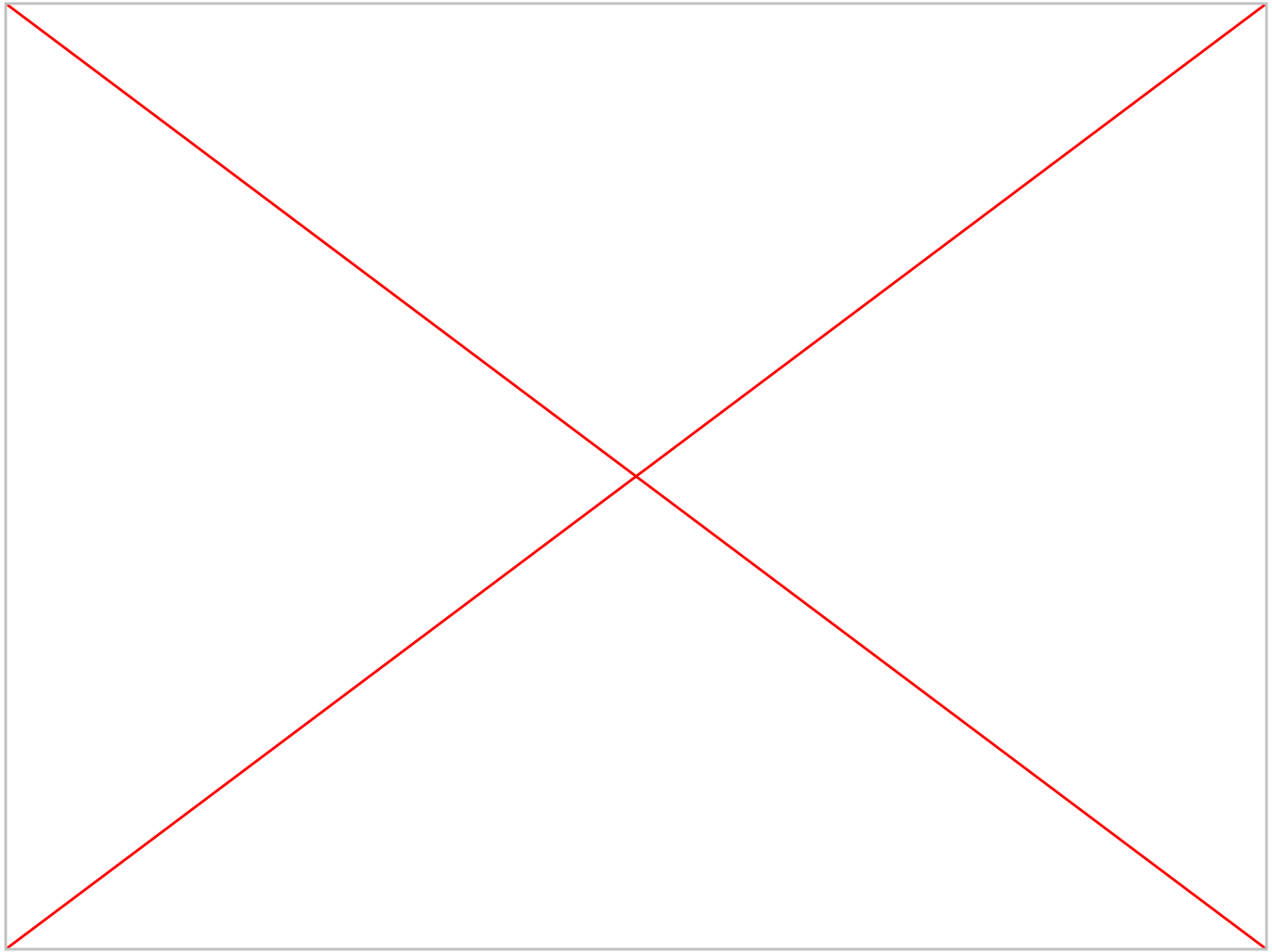
Meowth + Coins

Pikachu + Lightning

Sylveon + Hearts

Mudkip + Water

Demo



Classes

1. FaceMeshApp (Main Controller)

```
class FaceMeshApp {
private:
    VideoCapture camera;                    // Camera capture
    CascadeClassifier face_cascade;         // Face detection
    ParticleSystem particle_system;         // Particle effects
    unique_ptr<MongoDBHandler> mongo_handler; // Database handler

public:
    void run();                             // Main application loop
    vector<DetectedFace> detectFacesWithMesh(const Mat& image);
    bool detectMouthOpenSimple(const Mat& face_roi);
};
```

2. CLIInterface (User Interaction)

```
namespace CLIInterface {
    void displayWelcomeBanner();
    void displayPokemonMenu();
    pair<string, string> getPokemonSelection();
    bool testMongoDBConnection(const string& connection_string);
}
```

3. ParticleSystem (Visual Effects)

```
class ParticleSystem {
private:
    vector<unique_ptr<BaseParticle>> particles;
    string particle_type;

public:
    void setParticleType(const string& type);
    void startEmission();
    void update();
    void draw(Mat& image);
};
```

4. MongoDBHandler (Data Persistence)

```
class MongoDBHandler {
public:
    bool saveFaceData(const vector<DetectedFace>& faces,
                      const string& filename,
                      const string& pokemon,
                      const string& mask_file);
    pair<int, int> getStatistics();
    void showFaceDatabase();
};
```

Data Structures

1. DetectedFace Structure

```
struct DetectedFace {
    Rect rect;                          // Face bounding rectangle
    Point2f center;                     // Center point of face
    float confidence;                   // Detection confidence score
    vector<FaceLandmark> landmarks;     // 68 facial landmark points
    vector<Point2f> face_mesh;          // Generated mesh points
    double face_angle;                  // Face rotation angle
    bool mouth_open;                    // Mouth state detection
    Point2f mouth_center;               // Center of mouth for particles
};
```

2. FaceLandmark Structure

```
struct FaceLandmark {
    Point2f point;                      // 2D coordinate (x, y)
    int landmark_id;                    // Unique identifier (0-67)
    string landmark_name;               // Descriptive name
};
```

3. Landmark Mapping System

The application generates 68 facial landmarks organized as:

- **Jaw line**: 17 points (indices 0-16)
- **Eyebrows**: 10 points (17-26)
- **Nose**: 9 points (27-35)
- **Eyes**: 12 points (36-47)
- **Mouth**: 20 points (48-67)

4. Particle Inheritance Hierarchy

```
class BaseParticle {
protected:
    Point2f position, velocity;
    Scalar color;
    float life, max_life;
public:
    virtual void update() = 0;
    virtual void draw(Mat& image) = 0;
};

// Derived classes: WaterParticle, CoinParticle, GemParticle,
//                  HeartParticle, LightningParticle
```

Computer Vision Algorithm

1. Face Detection

- Uses OpenCV's Haar Cascade Classifier
- Processes frames at 30 FPS
- Selects largest detected face for stability
- Applies histogram equalization for better detection

2. Landmark Generation

```
// Procedural landmark generation based on face rectangle
vector<FaceLandmark> generateFacialLandmarks(const Rect& face_rect) {
    // Generate 68 landmarks using mathematical formulas
    // Jaw: curved line using sine function
    // Eyes: circular patterns around centers
    // Nose: vertical line with variations
    // Mouth: elliptical pattern around center
}
```

3. Mouth Detection

- **Multi-line scanning**: Analyzes multiple horizontal lines in mouth region
- **Dynamic thresholding**: Uses cheek brightness as reference
- **Pixel classification**: Counts dark pixels indicating mouth cavity
- **Ratio analysis**: Applies statistical thresholds for open/closed determination

Particle Systems

Particle Types and Behaviors

```
// Water particles (Mudkip)
class WaterParticle : public BaseParticle {
    void update() override {
        // Gravity simulation with splash effects
        velocity.y += gravity;
        position += velocity;
        // Fade over time
    }
};

// Coin particles (Meowth)
class CoinParticle : public BaseParticle {
    void update() override {
        // Spinning motion with arc trajectory
        rotation += angular_velocity;
        // Golden shimmer effect
    }
};
```

Database Integration

MongoDB Schema

```
// Face analysis collection
{
  "_id": ObjectId,
  "timestamp": ISODate,
  "filename": "facemesh_Mudkip_20250722_225616_1.jpg",
  "pokemon": "Mudkip",
  "mask_file": "mudkip_mask.png",
  "faces_detected": 1,
  "face_data": {
    "landmarks": [...],  // 68 facial landmarks
    "confidence": 1.0,
    "face_angle": -2.3,
    "mouth_open": true,
    "face_center": {"x": 640, "y": 360}
  }
}
```

Database Operations

```
// Save face analysis data
bool saveFaceData(const vector<DetectedFace>& faces,
                  const string& filename,
                  const string& pokemon,
                  const string& mask_file);

// Retrieve statistics
pair<int, int> getStatistics(); // Returns (total_captures, total_faces)

// Display database contents
void showFaceDatabase();
```

Program Flow

1. Initialization Phase

A. Program starts
B. Display welcome banner
C. Initialize MongoDB instance
D. Load environment variables
E. Test database connection
F. Display Pokémon selection menu
G. User selects a Pokémon
H. Initialize `FaceMeshApp` instance

2. Application Setup (`FaceMeshApp` constructor)

A. Initialize 68 facial landmark names
B. Load OpenCV Haar cascade models for face detection
C. Load the selected Pokémon mask image
D. Initialize the camera for video capture
E. Set up the particle system for visual effects

3. Main Application Loop (`FaceMeshApp::run`)

A. Start main application loop
B. Capture video frame from camera
C. Detect faces and compute facial mesh
D. Draw face with mouth emoji overlay
E. Wait for user input key
F. If spacebar is pressed, process and save the current frame
G. If 'q' is pressed, exit the loop
H. Handle optional mask movement (keys: w/s/a/d)
I. Display the processed frame in a window

Program Flow Continued

4. Face Detection Pipeline

A. Convert the captured frame to grayscale
B. Apply histogram equalization
C. Detect faces using Haar cascades
D. Generate 68 facial landmarks per detected face
E. Create a face mesh using the landmarks
F. Calculate face rotation angle
G. Detect whether the mouth is open
H. Overlay the Pokémon mask on the face
I. Trigger particle effects based on facial expression

5. Mouth Detection Algorithm (`detectMouthOpenSimple`)

A. Define the mouth region (approx. 75% down the face)
B. Convert region to grayscale and apply blur
C. Measure average brightness of the cheek area as a reference
D. Count dark pixels in the mouth region
E. Calculate ratios of dark and very dark pixels
F. Return true if the mouth is open based on threshold logic

6. Save to MongoDB Flow

A. User captures a photo (e.g., by pressing `SPACE`)
B. Current frame is saved with a timestamped filename
C. Detected face data is collected from the frame
D. Application gathers metadata:
    - Pokémon name
    - Mask file used
    - Number of faces
    - Landmark and mouth data per face
E. MongoDBHandler is called with the following parameters:
    - `vector<DetectedFace> faces`
    - `string filename`
    - `string pokemon`
    - `string mask_file`
F. A MongoDB document is created with a structured schema
G. Document is inserted into the `face_analysis` collection
H. Application checks if insert operation was successful
I. On success, confirmation is logged to console
J. On failure, error is caught and displayed

Project Files

https://github.com/flowprimedesign/CIS25/tree/main/final

YouTube Demo

https://youtu.be/JupguNcl_Nc