# Drone Light Show Simulation

Visualization of Pokémon Drone Formations

**Submitted by:**

**T.L.Megha Vardhan**

**T.Dwarakanath**

Department of Computer Science and Engineering

Vellore Institute of Technology

**Date:** October 26, 2025


Under the Guidance of

Dr. V.MuthuManikandan

**Abstract**

This project simulates a **Drone Light Show** using the PyBullet physics engine and OpenCV. Multiple virtual drones represented by glowing spheres dynamically rearrange themselves to form various Pokémon characters such as Pikachu, Charmander, Charizard, and more. The simulation uses real image data to generate 3D positions for the drones, creating a visually appealing animation. Additionally, the drones exhibit dynamic RGB color transitions, simulating the glowing effect observed in real drone light shows. The project demonstrates the potential of physics-based simulation for visual design and synchronization in swarm robotics.

# 1 Introduction

Drone light shows are large-scale aerial performances using hundreds of drones equipped with LED lights that form coordinated patterns and animations in the night sky. Instead of relying on physical hardware, this project replicates the concept virtually using the **PyBullet** physics engine, **OpenCV** for image processing, and **NumPy** for data computation.

The simulation forms different Pokémon figures by extracting coordinates from images and assigning those as target positions to a fleet of simulated drones. Each drone smoothly transitions into position while its color changes dynamically, producing an engaging light effect.

# 2 Objective

The main objectives of this simulation are:

- To simulate a fleet of drones forming various Pokémon-shaped patterns.
- To apply OpenCV image processing for coordinate extraction.
- To visualize dynamic RGB color transitions in a dark sky environment.
- To demonstrate smooth formation transitions and synchronization.

# 3 Tools and Libraries Used

- **PyBullet**: For real-time 3D simulation and visualization of drones.
- **OpenCV**: For image thresholding and coordinate extraction from Pokémon silhouettes.

- **NumPy**: For efficient numerical operations and scaling of image data.
- **Python**: For integrating all modules and handling the simulation flow.

# 4 Implementation Details

The following sections explain each part of the simulation process in detail.

## Step 1: Environment Setup and Sky Creation

The simulation begins with:

- Connecting to PyBullet GUI.
- Setting gravity to $-9.8\,m/s^2$.
- Disabling the default control panel for a clean view.

A dark background is created using a modified plane model:

- The `plane.urdf` file provides a flat surface.
- Its visual color is changed to $[0.03, 0.03, 0.05, 1]$ — a near-black tone, representing a night sky.

This gives the impression that the drones are flying against a dark sky backdrop, similar to real-world night drone shows.

## Step 2: Image Processing with OpenCV

Each Pokémon image (e.g., `pikachu.png`, `charmendor.png`) is processed as follows:

1. The image is read in grayscale.
2. It is resized to $300 \times 300$ pixels.
3. A binary threshold converts the image into black and white.
4. White pixels represent positions where drones will be placed.

Only a subset (400 points) is selected to represent each shape. The coordinates are scaled and centered so that the formation fits within the camera view.

## Step 3: Drone Creation

Instead of loading a complex drone model, each drone is represented by a glowing **sphere**:

- The sphere has a radius of 0.06.
- The color is initialized as bright yellow $[1, 1, 0, 1]$, mimicking a light-emitting drone.

This simplifies rendering and improves performance when simulating hundreds of drones.

## Step 4: Random Start Positions

Before the show begins, each drone starts from a random 3D position:

$$x, y \in [-5, 5], \quad z \in [3, 6]$$

This randomness makes the transition visually appealing as drones converge into the target shape.

## Step 5: Transition to Shape

The function `transition_to_shape()` performs smooth interpolation between each drone's starting position and its target position:

$$\text{new\_pos} = (1 - t) \times \text{start} + t \times \text{target}$$

where $t$ increases from 0 to 1 over 400 steps, producing a fluid motion.

## Step 6: Dynamic Color Animation

Once drones reach the target shape, their colors are continuously changed over time using sinusoidal RGB wave functions:

$$\begin{aligned} R &= 0.5 + 0.5 \sin(0.03(t + \text{cycle})) \\ G &= 0.5 + 0.5 \sin(0.03(t + \text{cycle}) + \frac{2\pi}{3}) \\ B &= 0.5 + 0.5 \sin(0.03(t + \text{cycle}) + \frac{4\pi}{3}) \end{aligned}$$

This gives a glowing, shifting color effect as if the drones are pulsating with light.

# 5 Simulation Flow and Output

During execution, the console displays progress messages showing which Pokémon image the drones are forming. The sequence is as follows:

```
Transitioning to pikachu.png...
Holding pikachu.png shape...
Transitioning to charmendor.png...
Holding charmendor.png shape...
Transitioning to charmelon.png...
Holding charmelon.png shape...
Transitioning to charizard.png...
Holding charizard.png shape...
Transitioning to bulbasur.png...
Holding bulbasur.png shape...
Transitioning to ivysur.png...
Holding ivysur.png shape...
Transitioning to sqartle.png...
Holding sqartle.png shape...
```

Each transition lasts approximately 8–10 seconds, followed by a 6-second color animation hold phase.

# 6 Code Implementation

```
import pybullet as p
import pybullet_data
import numpy as np
import cv2, time, os, random

p.connect(p.GUI)
p.setGravity(0, 0, -9.8)
p.configureDebugVisualizer(p.COV_ENABLE_GUI, 0)
p.resetDebugVisualizerCamera(cameraDistance=15, cameraYaw=45,
    cameraPitch=-40, cameraTargetPosition=[0, 0, 2])
p.setAdditionalSearchPath(pybullet_data.getDataPath())

plane_id = p.loadURDF("plane.urdf")
p.changeVisualShape(plane_id, -1, rgbaColor=[0.03, 0.03, 0.05, 1])

image_files = ["pikachu.png","charmendor.png","charmelon.png",
```

```
                "charizard.png","bulbasur.png","ivysur.png","sqartle.png
                    "]

def extract_positions_from_image(image_file, num_points=400, scale
    =0.035):
    if not os.path.exists(image_file): return []
    img = cv2.imread(image_file, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (300, 300))
    _, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
    ys, xs = np.nonzero(thresh)
    coords = list(zip(xs, ys))
    if len(coords) > num_points:
        idx = np.linspace(0, len(coords) - 1, num_points, dtype=int)
        coords = [coords[i] for i in idx]
    return [[(x - 150) * scale, -(y - 150) * scale, 2] for x, y in
        coords]

num_drones = 400
drones = []
for _ in range(num_drones):
    sphere = p.createVisualShape(p.GEOM_SPHERE, radius=0.06, rgbaColor
        =[1, 1, 0, 1])
    drone_id = p.createMultiBody(baseMass=0, baseVisualShapeIndex=
        sphere, basePosition=[0, 0, 5])
    drones.append(drone_id)

start_positions = [[random.uniform(-5, 5), random.uniform(-5, 5),
    random.uniform(3, 6)] for _ in range(num_drones)]

def transition_to_shape(target_positions, steps=400, delay=0.02):
    global start_positions
    while len(target_positions) < num_drones:
        target_positions.append([random.uniform(-5, 5), random.uniform
            (-5, 5), 5])
    for step in range(steps):
        t = (step + 1) / steps
        for i, drone_id in enumerate(drones):
            start = start_positions[i]
            target = target_positions[i]
            new_pos = [
                start[0]*(1-t)+target[0]*t,
                start[1]*(1-t)+target[1]*t,
                start[2]*(1-t)+target[2]*t
            ]
            p.resetBasePositionAndOrientation(drone_id, new_pos, [0, 0,
```

```
                0, 1])
        p.stepSimulation()
        time.sleep(delay)
    start_positions = target_positions.copy()

try:
    cycle = 0
    while True:
        for image in image_files:
            print(f"      Transitioning to {image}...")
            targets = extract_positions_from_image(image, num_drones)
            if not targets: continue
            transition_to_shape(targets, steps=400, delay=0.025)
            print(f"   Holding {image} shape...")
            for t in range(300):
                r = 0.5 + 0.5*np.sin(0.03*(t+cycle))
                g = 0.5 + 0.5*np.sin(0.03*(t+cycle)+2*np.pi/3)
                b = 0.5 + 0.5*np.sin(0.03*(t+cycle)+4*np.pi/3)
                for drone_id in drones:
                    p.changeVisualShape(drone_id, -1, rgbaColor=[r,g,b
                        ,1])
                p.stepSimulation()
                time.sleep(0.02)
            cycle += 1
except KeyboardInterrupt:
    p.disconnect()
```

Listing 1: Drone Light Show Simulation Code

# 7 Results and Discussion

The simulation produces a vivid, dynamic drone light show. Each Pokémon image is formed accurately using the pixel-based drone coordinates extracted from OpenCV. The dark sky enhances the visual contrast, while the RGB sinusoidal transitions provide a glowing, pulsating color effect.

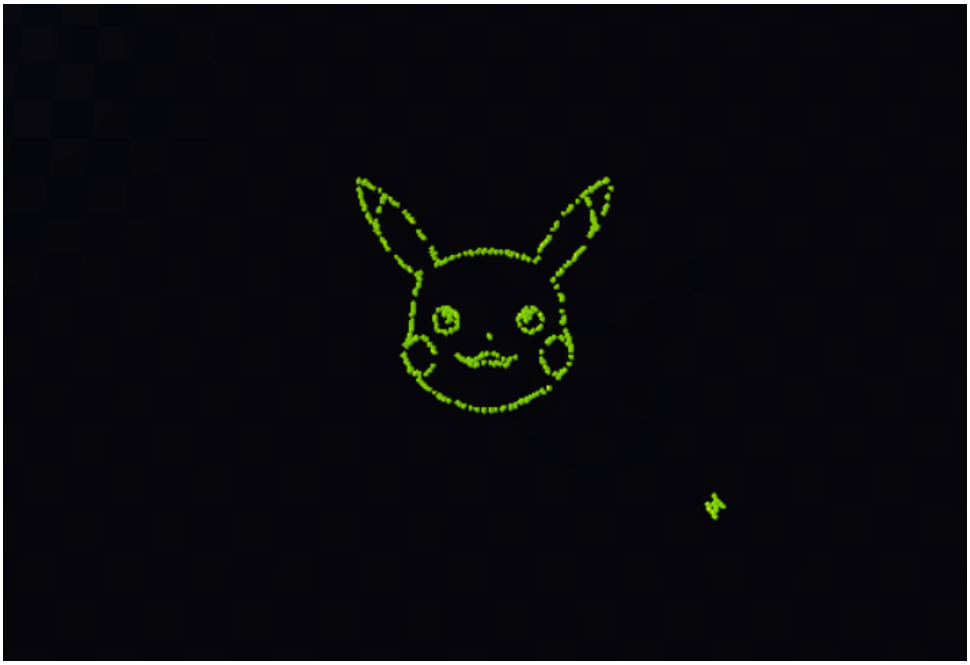

Figure 1: Drones forming Pikachu pattern in a dark sky environment.

### 7.1 Video

video link:-`https://drive.google.com/file/d/1uZHP1iBv5cbSBzg1IwP2hNasdzqNQMiM/view?usp=sharing`

### 7.2 Note:-

The Video is kept on fast track for time and memory constraints.on consedring real world drones light show we need to display the pattern atlest for a notable minutes on using the code we have provided the transition and formation is upto a notable time.and we have added the effects of transition between pokemons for a better experince.

## 8 Conclusion

This project demonstrates how physics simulation and computer vision can together emulate a drone light show. By converting image pixels into 3D coordinates, we can form virtually any shape or animation. Dynamic RGB color modulation gives the illusion of light-emitting drones, while the dark environment adds realism. This approach can serve as a testbed for real-world drone choreography and synchronization research.

# 9 Future Enhancements

- Adding collision avoidance and path planning for smoother transitions.
- Incorporating actual drone models with rotors and physics.
- Implementing multi-image transitions and text-based formations.
- Integrating ROS or MAVLink for real drone deployment.

# References


1. PyBullet Physics Engine: `https://pybullet.org`
2. OpenCV Library: `https://opencv.org`
3. Drone Swarm Simulation Research – IEEE Xplore