```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
import csv
import numpy as np
import pandas as pd
import math
import time

```

```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

```

```python
import os
import torch
import numpy as np
import pickle
import logging
import random
from tqdm import tqdm
#from torch_geometric.data import Dataset
```

Read data

```python
import sys
sys.path.append('/content/drive/MyDrive/action/data/processed/mmr_kp')
```

```python
filename = "/content/drive/MyDrive/action/data/processed/mmr_kp/data.pkl"
```

```python
import os
import numpy as np
import pandas as pd
import pickle
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

class MMRDataAnalyzer:
    def __init__(self, data_path):
        """Initialize analyzer with data path"""
```

```
        self.data_path = data_path
        self.data = self.load_processed_data()
        self.train_data = self.data['train']
        self.full_df = None

    def load_processed_data(self):
        """Load processed MMR data from pickle file"""
        with open(self.data_path, 'rb') as f:
            return pickle.load(f)

    def get_frame_data(self, data_point):
        """Convert a single data point to DataFrame"""
        x = data_point['new_x']
        df = pd.DataFrame({
            'frame_num': np.zeros(len(x)),
            'x': x[:, 0],
            'y': x[:, 1],
            'z': x[:, 2]
        })
        if 'cluster_labels' in data_point:
            df['cluster'] = data_point['cluster_labels']
        return df

    def analyze_dataset(self):
        """Analyze entire dataset"""
        all_points = []
        all_frames = []
        frame_count = 0

        for data_point in self.train_data:
            x = data_point['new_x']
            num_points = len(x)
            all_points.append(x)
            all_frames.extend([frame_count] * num_points)
            frame_count += 1

        all_points = np.concatenate(all_points, axis=0)
        self.full_df = pd.DataFrame({
            'frame_num': all_frames,
            'x': all_points[:, 0],
            'y': all_points[:, 1],
            'z': all_points[:, 2]
        })
        return self.full_df

    def visualize_frame(self, frame_idx=0, title="Point Cloud Visualization'
        """Visualize a single frame of point cloud data"""
        data_point = self.train_data[frame_idx]
        x = data_point['new_x']

        fig = plt.figure(figsize=(12, 8))
```

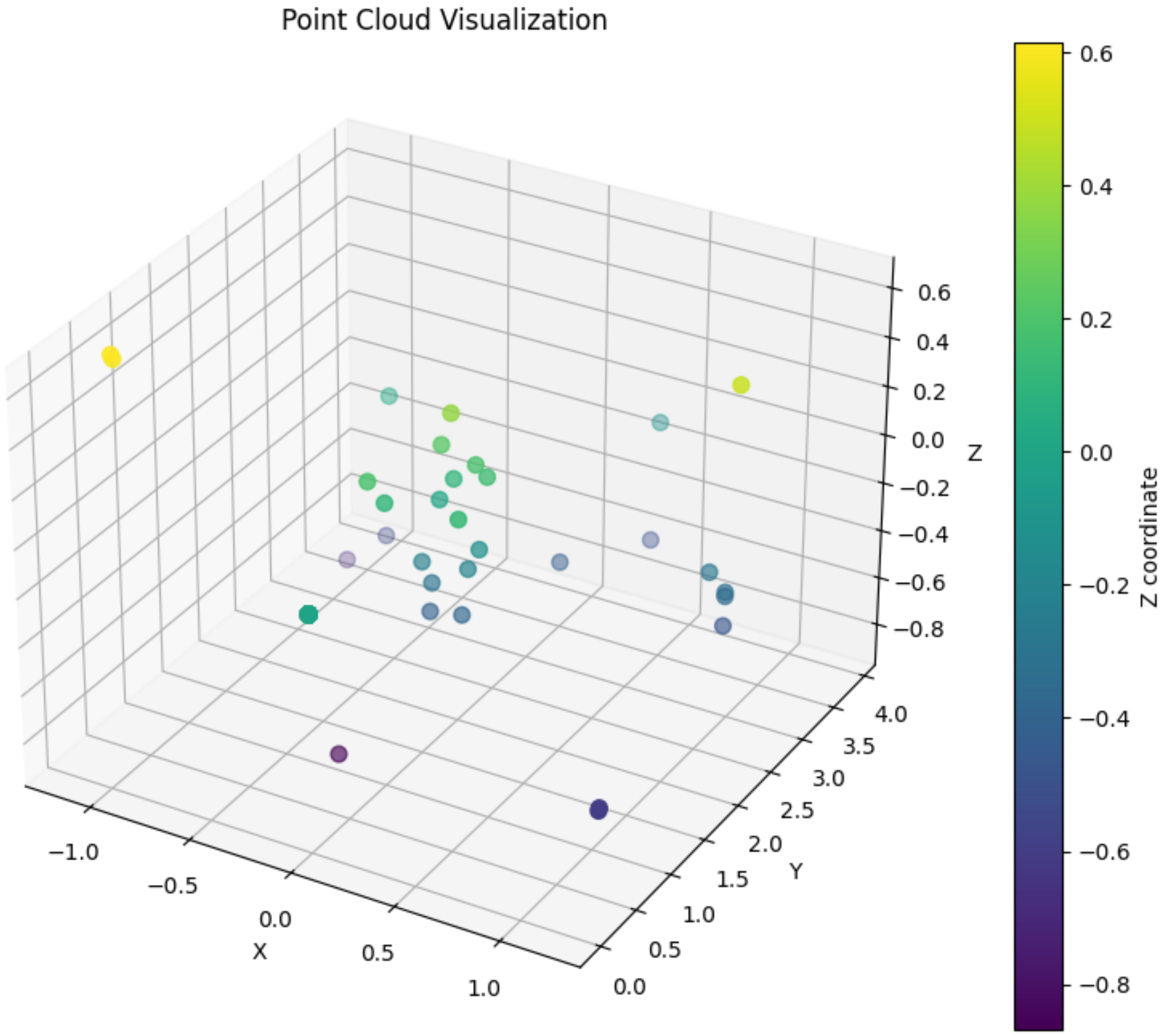
```python
        ax = fig.add_subplot(111, projection='3d')
        scatter = ax.scatter(x[:, 0], x[:, 1], x[:, 2],
                             c=x[:, 2],
                             cmap='viridis',
                             marker='o',
                             s=50)

        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        ax.set_title(title)
        plt.colorbar(scatter, label='Z coordinate')
        plt.show()

    def analyze_by_action(self):
        """Analyze data grouped by action labels"""
        if 'y' not in self.train_data[0]:
            return None

        action_data = {}
        action_labels = [d['y'] for d in self.train_data]

        for data_point, action in zip(self.train_data, action_labels):
            if action not in action_data:
                action_data[action] = []
            action_data[action].append(data_point['new_x'])

        stats = {}
        for action, points in action_data.items():
            points = np.concatenate(points, axis=0)
            df = pd.DataFrame({
                'x': points[:, 0],
                'y': points[:, 1],
                'z': points[:, 2]
            })
            stats[action] = df.describe()
        return stats

    def plot_coordinate_distributions(self):
        """Plot distributions of x, y, z coordinates"""
        if self.full_df is None:
            self.analyze_dataset()

        fig, axes = plt.subplots(1, 3, figsize=(15, 5))

        for i, coord in enumerate(['x', 'y', 'z']):
            axes[i].hist(self.full_df[coord], bins=50)
            axes[i].set_title(f'{coord.upper()} Distribution')
            axes[i].set_xlabel(f'{coord.upper()} Coordinate')
            axes[i].set_ylabel('Frequency')

```

```python
        plt.tight_layout()
        plt.show()

    def print_detailed_statistics(self):
        """Print detailed statistics for each coordinate"""
        if self.full_df is None:
            self.analyze_dataset()

        # Percentile analysis
        for coord in ['x', 'y', 'z']:
            print(f"\nDetailed {coord.upper()} coordinate analysis:")
            percentiles = [0, 1, 5, 25, 50, 75, 95, 99, 100]
            for p in percentiles:
                value = np.percentile(self.full_df[coord], p)
                print(f"{p}th percentile: {value:.6f}")

            print(f"Mean: {self.full_df[coord].mean():.6f}")
            print(f"Std: {self.full_df[coord].std():.6f}")
            print(f"Skewness: {self.full_df[coord].skew():.6f}")
            print(f"Kurtosis: {self.full_df[coord].kurtosis():.6f}")

        # Non-zero analysis
        print("\nAnalysis of non-zero points:")
        for coord in ['x', 'y', 'z']:
            non_zero = self.full_df[self.full_df[coord] != 0][coord]
            print(f"\n{coord.upper()} coordinate (non-zero):")
            print(f"Count: {len(non_zero)}")
            print(f"Mean: {non_zero.mean():.6f}")
            print(f"Std: {non_zero.std():.6f}")
            print(f"Min: {non_zero.min():.6f}")
            print(f"Max: {non_zero.max():.6f}")

    def analyze_points_per_frame(self):
        """Analyze and visualize points per frame distribution"""
        points_per_frame = [len(d['new_x']) for d in self.train_data]

        print("\nPoints per frame:")
        print(f"Mean: {np.mean(points_per_frame):.2f}")
        print(f"Std: {np.std(points_per_frame):.2f}")
        print(f"Min: {np.min(points_per_frame)}")
        print(f"Max: {np.max(points_per_frame)}")

        plt.figure(figsize=(10, 5))
        plt.hist(points_per_frame, bins=50)
        plt.title('Distribution of Points per Frame')
        plt.xlabel('Number of Points')
        plt.ylabel('Frequency')
        plt.show()

def main():
    # Initialize analyzer
```

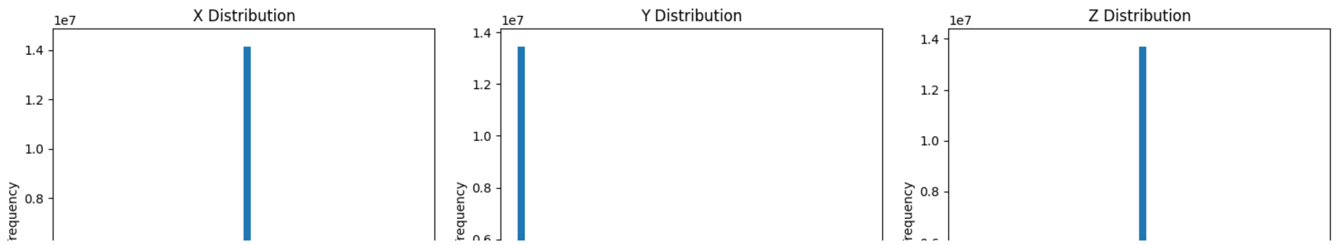
```
    data_path = '/content/drive/MyDrive/action/data/processed/mmr_action/dat
    analyzer = MMRDataAnalyzer(data_path)

    # Run analyses
    print("\nAnalyzing training data...")

    # Single frame analysis
    print("\nSingle Frame Analysis:")
    frame_df = analyzer.get_frame_data(analyzer.train_data[0])
    print("\nFrame Statistics:")
    print(frame_df.describe())

    # Visualize first frame
    print("\nVisualizing first frame...")
    analyzer.visualize_frame(0)

    # Full dataset analysis
    print("\nFull Dataset Analysis:")
    full_df = analyzer.analyze_dataset()
    print("\nDataset Statistics:")
    print(full_df.describe())

    # Plot distributions
    analyzer.plot_coordinate_distributions()

    # Action analysis
    action_stats = analyzer.analyze_by_action()
    if action_stats:
        print("\nAnalysis by Action:")
        for action, stats in action_stats.items():
            print(f"\nAction {action} Statistics:")
            print(stats)

    # Points per frame analysis
    analyzer.analyze_points_per_frame()

    # Detailed statistics
    analyzer.print_detailed_statistics()

if __name__ == "__main__":
    main()
```

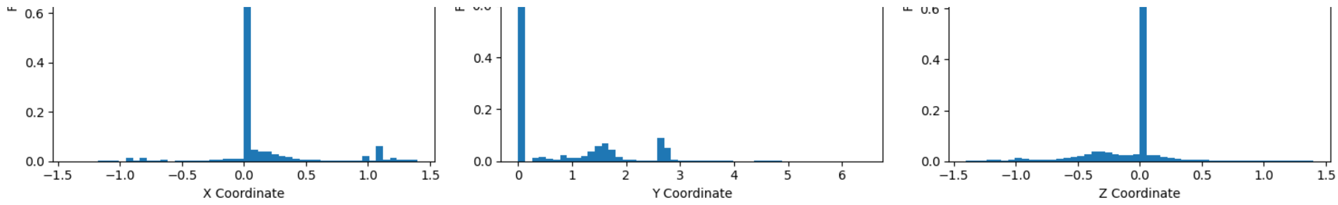
```
Analyzing training data...

Single Frame Analysis:

Frame Statistics:
       frame_num           x           y           z
count      110.0  110.000000  110.000000  110.000000
mean         0.0    0.047783    0.536345   -0.038962
std          0.0    0.349468    0.975685    0.212701
min          0.0   -1.158141    0.000000   -0.868396
```

```
min            0.0   -1.158141   0.000000   -0.868596
25%            0.0    0.000000   0.000000    0.000000
50%            0.0    0.000000   0.000000    0.000000
75%            0.0    0.000000   0.577146    0.000000
max            0.0    1.202685   3.890844    0.614703

Visualizing first frame...
```




```
Full Dataset Analysis:

Dataset Statistics:
          frame_num             x             y             z
count  1.873696e+07  1.873696e+07  1.873696e+07  1.873696e+07
mean   8.516750e+04  7.687180e-02  5.204076e-01 -5.673060e-02
std    4.917177e+04  3.291378e-01  9.468305e-01  2.585428e-01
min    0.000000e+00 -1.399950e+00  0.000000e+00 -1.399992e+00
25%    4.258375e+04  0.000000e+00  0.000000e+00  0.000000e+00
50%    8.516750e+04  0.000000e+00  0.000000e+00  0.000000e+00
75%    1.277512e+05  0.000000e+00  8.577272e-01  0.000000e+00
max    1.703350e+05  1.399950e+00  6.434681e+00  1.399989e+00
```





Analysis by Action:

Action 31 Statistics:

	x	y	z
count	378400.000000	378400.000000	378400.000000
mean	0.074577	0.514265	-0.061778
std	0.335568	0.939826	0.261655
min	-1.371951	0.000000	-1.398615
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.825487	0.000000
max	1.399950	6.349585	1.399224

Action 11 Statistics:

	x	y	z
count	468820.000000	468820.000000	468820.000000
mean	0.077840	0.527854	-0.056401
std	0.317026	0.960928	0.250767
min	-1.389769	0.000000	-1.398883
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.854694	0.000000
max	1.393587	6.434681	1.398990

Action 12 Statistics:

	x	y	z
count	471350.000000	471350.000000	471350.000000
mean	0.078124	0.485886	-0.059419
std	0.338254	0.917437	0.281582
min	-1.359224	0.000000	-1.399949
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.513116	0.000000
max	1.393587	6.399363	1.398434

Action 46 Statistics:

	x	y	z
count	157080.000000	157080.000000	157080.000000
mean	0.076483	0.431295	-0.043606
std	0.309860	0.886490	0.219364
min	-1.309590	0.000000	-1.398176
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000
max	1.393587	6.285223	1.391910

Action 10 Statistics:

	x	y	z
count	468930.000000	468930.000000	468930.000000
mean	0.072497	0.508579	-0.056327
std	0.323098	0.925929	0.264549

```
min       -1.374497       0.000000      -1.399579
25%        0.000000       0.000000       0.000000
50%        0.000000       0.000000       0.000000
75%        0.000000       0.849805       0.000000
max        1.399950       6.319821       1.399556

Action 5 Statistics:
                   x              y              z
count  467720.000000  467720.000000  467720.000000
mean        0.076633       0.516826      -0.047413
std         0.343420       0.954382       0.253734
min        -1.374497       0.000000      -1.395063
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.777470       0.000000
max         1.399950       6.379820       1.397185

Action 40 Statistics:
                   x              y              z
count  343640.000000  343640.000000  343640.000000
mean        0.078217       0.534478      -0.058406
std         0.336558       0.955172       0.264493
min        -1.389769       0.000000      -1.399087
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.894839       0.000000
max         1.399950       6.383748       1.399817

Action 28 Statistics:
                   x              y              z
count  403040.000000  403040.000000  403040.000000
mean        0.076906       0.549277      -0.068857
std         0.345861       0.969178       0.269449
min        -1.374497       0.000000      -1.399150
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.929228       0.000000
max         1.399950       6.346957       1.398393

Action 37 Statistics:
                   x              y              z
count  374440.000000  374440.000000  374440.000000
mean        0.078752       0.522425      -0.066635
std         0.315767       0.955456       0.264576
min        -1.394860       0.000000      -1.399194
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.934513       0.000000
max         1.399950       6.434008       1.399444

Action 48 Statistics:
                  x             y             z
count  90420.000000  90420.000000  90420.000000
mean       0.078932      0.436856     -0.056474
std        0.316602      0.868073      0.238996
min       -1.276500      0.000000     -1.391721
25%        0.000000      0.000000      0.000000
50%        0.000000      0.000000      0.000000
```

```
50%        0.000000      0.000000      0.000000
75%        0.000000      0.000000      0.000000
max        1.399950      6.269374      1.395553

Action 9 Statistics:
                   x              y              z
count  462990.000000  462990.000000  462990.000000
mean        0.073573       0.554251      -0.058386
std         0.330726       0.967275       0.261660
min        -1.389769       0.000000      -1.399277
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       1.167069       0.000000
max         1.399950       6.417338       1.397870

Action 18 Statistics:
                   x              y              z
count  464200.000000  464200.000000  464200.000000
mean        0.076618       0.537162      -0.042527
std         0.327533       0.959041       0.258339
min        -1.399950       0.000000      -1.398597
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       1.082601       0.000000
max         1.399950       6.393705       1.398511

Action 3 Statistics:
                   x              y              z
count  467390.000000  467390.000000  467390.000000
mean        0.077899       0.568150      -0.071368
std         0.348494       0.986090       0.279974
min        -1.397405       0.000000      -1.398430
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       1.091420       0.000000
max         1.399950       6.383855       1.399461

Action 4 Statistics:
                   x              y              z
count  470030.000000  470030.000000  470030.000000
mean        0.083634       0.573610      -0.073397
std         0.360205       1.006856       0.290792
min        -1.399950       0.000000      -1.398132
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.952551       0.000000
max         1.399950       6.405042       1.399950

Action 19 Statistics:
                   x              y              z
count  466180.000000  466180.000000  466180.000000
mean        0.076934       0.500493      -0.049903
std         0.324334       0.937175       0.255963
min        -1.374497       0.000000      -1.399920
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.646444       0.000000
max         1.399950       6.424575       1.399923
```

Action 32 Statistics:

	x	y	z
count	382580.000000	382580.000000	382580.000000
mean	0.075206	0.479069	-0.048354
std	0.346054	0.900482	0.249378
min	-1.378315	0.000000	-1.397405
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.534835	0.000000
max	1.397405	6.147089	1.399785

Action 2 Statistics:

	x	y	z
count	470470.000000	470470.000000	470470.000000
mean	0.075689	0.579819	-0.063618
std	0.346258	1.001837	0.275747
min	-1.399950	0.000000	-1.399992
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	1.155810	0.000000
max	1.399950	6.408193	1.399882

Action 35 Statistics:

	x	y	z
count	343970.000000	343970.000000	343970.000000
mean	0.065449	0.479796	-0.041497
std	0.302885	0.897658	0.235044
min	-1.351588	0.000000	-1.399636
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.508344	0.000000
max	1.393587	6.342800	1.399777

Action 17 Statistics:

	x	y	z
count	437250.000000	437250.000000	437250.000000
mean	0.080128	0.562249	-0.044236
std	0.332713	0.988308	0.257440
min	-1.374497	0.000000	-1.398787
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	1.174127	0.000000
max	1.399950	6.434319	1.399356

Action 33 Statistics:

	x	y	z
count	408430.000000	408430.000000	408430.000000
mean	0.081345	0.529303	-0.059197
std	0.346095	0.949509	0.266641
min	-1.378315	0.000000	-1.399966
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.849508	0.000000
max	1.399950	6.434681	1.398520

Action 44 Statistics:

```
                   x              y              z
count  180730.000000  180730.000000  180730.000000
mean        0.068136       0.405919      -0.045828
std         0.330267       0.848062       0.217462
min        -1.389769       0.000000      -1.398622
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.000000       0.000000
max         1.393587       6.163586       1.385596

Action 30 Statistics:
                   x              y              z
count  407000.000000  407000.000000  407000.000000
mean        0.082047       0.489071      -0.073012
std         0.325658       0.896445       0.262465
min        -1.374497       0.000000      -1.399925
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.712131       0.000000
max         1.399950       6.279720       1.396795

Action 26 Statistics:
                   x              y              z
count  375210.000000  375210.000000  375210.000000
mean        0.076987       0.559731      -0.061462
std         0.347518       0.974459       0.283168
min        -1.397405       0.000000      -1.399356
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       1.024297       0.000000
max         1.399950       6.432427       1.399950

Action 13 Statistics:
                   x              y              z
count  469040.000000  469040.000000  469040.000000
mean        0.071510       0.505082      -0.053410
std         0.340723       0.928638       0.264821
min        -1.374497       0.000000      -1.399950
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.801803       0.000000
max         1.399950       6.296864       1.399496

Action 36 Statistics:
                   x              y              z
count  349250.000000  349250.000000  349250.000000
mean        0.079881       0.528928      -0.063319
std         0.320422       0.956455       0.263092
min        -1.389769       0.000000      -1.397785
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.948160       0.000000
max         1.393587       6.311101       1.399598

Action 16 Statistics:
                   x              y              z
count  466400.000000  466400.000000  466400.000000
mean        0.073473       0.502445      -0.040275
```

```
std         0.325360       0.954712       0.252284
min        -1.389769       0.000000      -1.399508
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.523012       0.000000
max         1.399950       6.393885       1.398629

Action 20 Statistics:
                   x              y              z
count  409420.000000  409420.000000  409420.000000
mean        0.079091       0.513077      -0.053115
std         0.317295       0.940046       0.256633
min        -1.389769       0.000000      -1.399572
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.846486       0.000000
max         1.399950       6.398951       1.398761

Action 1 Statistics:
                   x              y              z
count  467390.000000  467390.000000  467390.000000
mean        0.073163       0.567240      -0.066120
std         0.349041       1.006595       0.294535
min        -1.399950       0.000000      -1.399929
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.898223       0.000000
max         1.399950       6.429296       1.399778

Action 25 Statistics:
                   x              y              z
count  402490.000000  402490.000000  402490.000000
mean        0.072725       0.451899      -0.048821
std         0.316602       0.883519       0.227775
min        -1.374497       0.000000      -1.399848
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.000000       0.000000
max         1.399950       6.392045       1.397190

Action 47 Statistics:
                   x              y              z
count  149820.000000  149820.000000  149820.000000
mean        0.071963       0.479796      -0.067595
std         0.302505       0.915010       0.254481
min        -1.389769       0.000000      -1.399089
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.509499       0.000000
max         1.393587       6.269374       1.398368

Action 24 Statistics:
                   x              y              z
count  404470.000000  404470.000000  404470.000000
mean        0.079350       0.533825      -0.069417
std         0.325634       0.950116       0.262742
min        -1.389769       0.000000      -1.399982
```

```
25%          0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000
75%          0.000000       1.025297       0.000000
max          1.399950       6.381793       1.399262

Action 43 Statistics:
                    x              y              z
count   179740.000000  179740.000000  179740.000000
mean         0.061394       0.457617      -0.069759
std          0.287983       0.883638       0.222477
min         -1.256137       0.000000      -1.398671
25%          0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000
75%          0.000000       0.000000       0.000000
max          1.393587       6.240797       1.394823

Action 8 Statistics:
                    x              y              z
count   469040.000000  469040.000000  469040.000000
mean         0.084970       0.522866      -0.055848
std          0.319263       0.944904       0.258823
min         -1.374497       0.000000      -1.399687
25%          0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000
75%          0.000000       0.882933       0.000000
max          1.399950       6.432488       1.399646

Action 23 Statistics:
                    x              y              z
count   374550.000000  374550.000000  374550.000000
mean         0.074803       0.548074      -0.053603
std          0.332200       0.971611       0.241057
min         -1.374497       0.000000      -1.399892
25%          0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000
75%          0.000000       1.062627       0.000000
max          1.399950       6.422101       1.397105

Action 0 Statistics:
                    x              y              z
count   451770.000000  451770.000000  451770.000000
mean         0.070893       0.547336      -0.055559
std          0.321600       0.992501       0.260741
min         -1.399950       0.000000      -1.398672
25%          0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000
75%          0.000000       0.906558       0.000000
max          1.399950       6.431149       1.399431

Action 45 Statistics:
                    x              y              z
count   158840.000000  158840.000000  158840.000000
mean         0.077193       0.410146      -0.036707
std          0.313086       0.858483       0.210863
min         -1.389769       0.000000      -1.397101
25%          0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000
75%          0.000000       0.000000       0.000000
```

```
max          1.393587       6.186941       1.398242

Action 21 Statistics:
                    x              y              z
count  376750.000000  376750.000000  376750.000000
mean        0.078417       0.541347      -0.065800
std         0.334038       0.950795       0.270160
min        -1.389769       0.000000      -1.398807
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.991691       0.000000
max         1.393587       6.413658       1.390261

Action 6 Statistics:
                    x              y              z
count  466950.000000  466950.000000  466950.000000
mean        0.083345       0.542249      -0.055615
std         0.327001       0.977326       0.260309
min        -1.389769       0.000000      -1.399695
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.858658       0.000000
max         1.399950       6.429239       1.399429

Action 34 Statistics:
                    x              y              z
count  372900.000000  372900.000000  372900.000000
mean        0.078543       0.504012      -0.050469
std         0.319428       0.918820       0.245226
min        -1.399950       0.000000      -1.398905
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       0.855156       0.000000
max         1.399950       6.414887       1.394047

Action 39 Statistics:
                    x              y              z
count  376640.000000  376640.000000  376640.000000
mean        0.072222       0.548899      -0.071456
std         0.326717       0.960930       0.276787
min        -1.384678       0.000000      -1.399840
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       1.043640       0.000000
max         1.397405       6.356513       1.397997

Action 15 Statistics:
                    x              y              z
count  469260.000000  469260.000000  469260.000000
mean        0.076804       0.539083      -0.043463
std         0.311330       0.953184       0.237928
min        -1.399950       0.000000      -1.399516
25%         0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000
75%         0.000000       1.158413       0.000000
max         1.393587       6.414969       1.399753
```

Action 27 Statistics:

	x	y	z
count	405130.000000	405130.000000	405130.000000
mean	0.080284	0.547597	-0.053680
std	0.322925	0.947382	0.242774
min	-1.356679	0.000000	-1.399793
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	1.279008	0.000000
max	1.399950	6.390528	1.398357

Action 14 Statistics:

	x	y	z
count	440550.000000	440550.000000	440550.000000
mean	0.084835	0.504918	-0.049325
std	0.328976	0.937369	0.261018
min	-1.389769	0.000000	-1.399944
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.808561	0.000000
max	1.393587	6.176641	1.399989

Action 22 Statistics:

	x	y	z
count	375870.000000	375870.000000	375870.000000
mean	0.073941	0.568886	-0.058043
std	0.322990	0.966665	0.253392
min	-1.389769	0.000000	-1.399231
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	1.260711	0.000000
max	1.393587	6.431538	1.399225

Action 29 Statistics:

	x	y	z
count	380490.000000	380490.000000	380490.000000
mean	0.082675	0.440326	-0.052754
std	0.323891	0.851155	0.209361
min	-1.221775	0.000000	-1.398298
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.308037	0.000000
max	1.399950	5.861362	1.399806

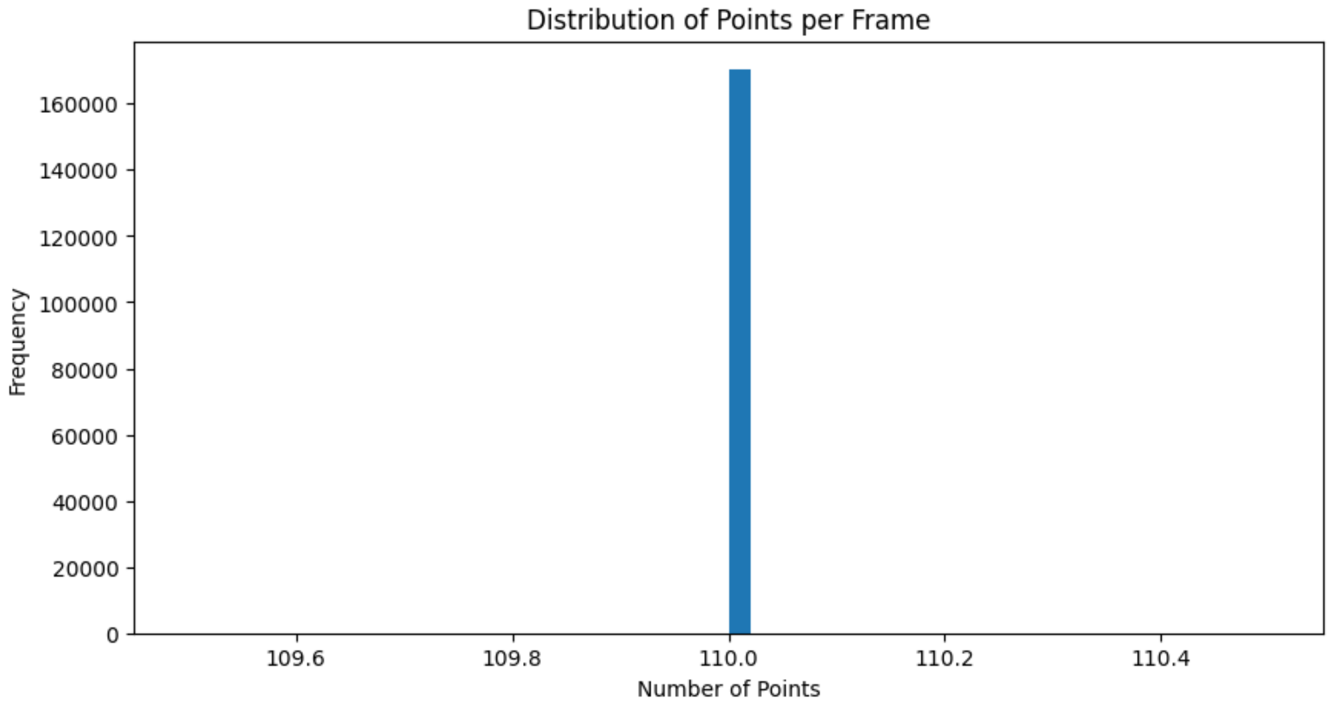
Action 42 Statistics:

	x	y	z
count	174240.000000	174240.000000	174240.000000
mean	0.071184	0.474571	-0.048048
std	0.310809	0.923397	0.244577
min	-1.374497	0.000000	-1.399148
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000
max	1.399950	6.420762	1.399556

Action 38 Statistics:

	x	y	z
count	375210.000000	375210.000000	375210.000000

```
count  [illegible]
mean         0.072853       0.501803      -0.058829
std          0.311792       0.929197       0.238388
min         -1.389769       0.000000      -1.399300
25%          0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000
75%          0.000000       0.659137       0.000000
max          1.393587       6.399340       1.399584

Action 41 Statistics:
                    x              y              z
count   340560.000000  340560.000000  340560.000000
mean         0.082395       0.511885      -0.051959
std          0.338019       0.919939       0.256697
min         -1.389769       0.000000      -1.399533
25%          0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000
75%          0.000000       0.860548       0.000000
max          1.399950       6.267469       1.395117

Action 7 Statistics:
                    x              y              z
count   469920.000000  469920.000000  469920.000000
mean         0.076454       0.489076      -0.063943
std          0.321035       0.931456       0.259399
min         -1.389769       0.000000      -1.399851
25%          0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000
75%          0.000000       0.422368       0.000000
max          1.393587       6.386649       1.398395

Points per frame:
Mean: 110.00
Std: 0.00
Min: 110
Max: 110
```




```
Detailed X coordinate analysis:
```

```
0th percentile: -1.399950
1th percentile: -0.916331
5th percentile: -0.058543
25th percentile: 0.000000
50th percentile: 0.000000
75th percentile: 0.000000
95th percentile: 1.065235
99th percentile: 1.207775
100th percentile: 1.399950
Mean: 0.076872
Std: 0.329138
Skewness: 1.463836
Kurtosis: 5.981577

Detailed Y coordinate analysis:
0th percentile: 0.000000
1th percentile: 0.000000
5th percentile: 0.000000
25th percentile: 0.000000
50th percentile: 0.000000
75th percentile: 0.857727
95th percentile: 2.688806
99th percentile: 3.350385
100th percentile: 6.434681
Mean: 0.520408
Std: 0.946831
Skewness: 1.786532
Kurtosis: 2.654240

Detailed Z coordinate analysis:
0th percentile: -1.399992
1th percentile: -1.075179
5th percentile: -0.538880
25th percentile: 0.000000
50th percentile: 0.000000
75th percentile: 0.000000
95th percentile: 0.148700
99th percentile: 0.693422
100th percentile: 1.399989
Mean: -0.056731
Std: 0.258543
Skewness: -1.148323
Kurtosis: 8.433358

Analysis of non-zero points:

X coordinate (non-zero):
Count: 4988251
Mean: 0.288747
Std: 0.587996
Min: -1.399950
Max: 1.399950

Y coordinate (non-zero):
Count: 5281486
Mean: 1.846233
Std: 0.855954
Min: 0.300078
```

```
Max: 6.434681

Z coordinate (non-zero):
Count: 5276678
Mean: -0.201445
Std: 0.456296
Min: -1.399992
Max: 1.399989
```

Small tool functions

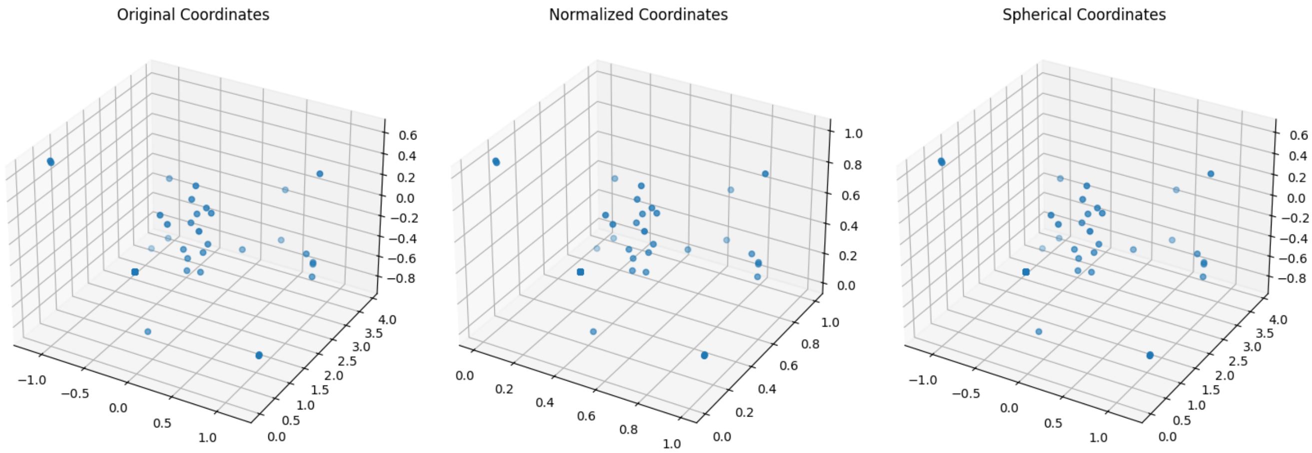
```
import numpy as np
from sklearn.preprocessing import MinMaxScaler

class MMRUtils:
    @staticmethod
    def normalize(x, x_min=None, x_max=None):
        """Normalize data to [0,1] range"""
        if x_min is None:
            x_min = np.min(x)
        if x_max is None:
            x_max = np.max(x)
        return (x - x_min) / (x_max - x_min)

    @staticmethod
    def normalize_points(points):
        """Normalize point cloud data"""
```

```
        normalized = np.zeros_like(points)
        for i in range(points.shape[1]):
            normalized[:, i] = MMRUtils.normalize(points[:, i])
        return normalized

    @staticmethod
    def weight_by_intensity(data_point):
        """Convert intensity values to weights in [0,1] range"""
        if 'intensity' in data_point:
            intensities = data_point['intensity']
            return MinMaxScaler().fit_transform(
                np.asarray(intensities).reshape(-1,1)).reshape(1,-1)[0]
        return None

    @staticmethod
    def to_vertical(array):
        """Convert array to vertical format"""
        return array.reshape(-1, 1)

    @staticmethod
    def to_horizontal(array):
        """Convert array to horizontal format"""
        return array.reshape(1, -1)

    @staticmethod
    def vector_angle(vec1, vec2):
        """Calculate normalized angle between two vectors"""
        return np.abs(np.dot(vec1, vec2)) / (np.linalg.norm(vec1) * np.linal

    @staticmethod
    def cartesian_to_spherical(points):
        """Convert cartesian coordinates to spherical
        Args:
            points: Array of shape (N, 3) containing [x, y, z] coordinates
        Returns:
            Array of shape (N, 3) containing [r, theta, phi] coordinates
        """
        x, y, z = points[:, 0], points[:, 1], points[:, 2]
        r = np.sqrt(x**2 + y**2 + z**2)
        theta = np.arctan2(y, x)
        theta[theta < 0] += 2 * np.pi
        phi = np.arctan2(np.sqrt(x**2 + y**2), z)
        return np.stack([r, theta, phi], axis=1)

    @staticmethod
    def spherical_to_cartesian(points):
        """Convert spherical coordinates to cartesian
        Args:
            points: Array of shape (N, 3) containing [r, theta, phi] coordir
        Returns:
            Array of shape (N, 3) containing [x, y, z] coordinates
```

```
        """
        r, theta, phi = points[:, 0], points[:, 1], points[:, 2]
        x = r * np.cos(theta) * np.sin(phi)
        y = r * np.sin(theta) * np.sin(phi)
        z = r * np.cos(phi)
        return np.stack([x, y, z], axis=1)

    @staticmethod
    def get_cluster_colors(labels):
        """Get colors for different clusters/labels"""
        colors = ['r', 'b', 'g', 'c', 'm', 'darkorange', 'deepskyblue',
                  'blueviolet', 'crimson', 'orangered', 'k']
        return [colors[label % len(colors)] for label in labels]

    @staticmethod
    def process_frame(data_point):
        """Process a single frame of point cloud data
        Returns normalized coordinates and spherical coordinates"""
        points = data_point['new_x']

        # Normalize cartesian coordinates
        normalized = MMRUtils.normalize_points(points)

        # Convert to spherical coordinates
        spherical = MMRUtils.cartesian_to_spherical(points)

        return {
            'original': points,
            'normalized': normalized,
            'spherical': spherical
        }

# Enhanced MMRDataAnalyzer class with utility functions
class EnhancedMMRAnalyzer(MMRDataAnalyzer):
    def __init__(self, data_path):
        super().__init__(data_path)
        self.utils = MMRUtils()

    def analyze_frame_geometry(self, frame_idx=0):
        """Analyze geometric properties of a frame"""
        data_point = self.train_data[frame_idx]
        processed = self.utils.process_frame(data_point)

        # Calculate geometric properties
        points = processed['original']
        center = np.mean(points, axis=0)
        distances = np.linalg.norm(points - center, axis=1)

        print("\nGeometric Analysis:")
        print(f"Center of mass: {center}")
        print(f"Mean distance from center: {np.mean(distances):.4f}")
```

```python
        print(f"Max distance from center: {np.max(distances):.4f}")

        # Visualize in both coordinate systems
        fig = plt.figure(figsize=(15, 5))

        # Original coordinates
        ax1 = fig.add_subplot(131, projection='3d')
        ax1.scatter(points[:, 0], points[:, 1], points[:, 2])
        ax1.set_title('Original Coordinates')

        # Normalized coordinates
        ax2 = fig.add_subplot(132, projection='3d')
        norm_points = processed['normalized']
        ax2.scatter(norm_points[:, 0], norm_points[:, 1], norm_points[:, 2])
        ax2.set_title('Normalized Coordinates')

        # Spherical coordinates
        ax3 = fig.add_subplot(133, projection='3d')
        sph_points = processed['spherical']
        ax3.scatter(sph_points[:, 0] * np.sin(sph_points[:, 2]) * np.cos(spł
                    sph_points[:, 0] * np.sin(sph_points[:, 2]) * np.sin(sph_
                    sph_points[:, 0] * np.cos(sph_points[:, 2]))
        ax3.set_title('Spherical Coordinates')

        plt.tight_layout()
        plt.show()

        return processed

# Example usage
if __name__ == "__main__":
    data_path = '/content/drive/MyDrive/action/data/processed/mmr_action/dat
    analyzer = EnhancedMMRAnalyzer(data_path)

    # Analyze a frame with new utilities
    frame_data = analyzer.analyze_frame_geometry(0)

    # Example of using utilities
    points = analyzer.train_data[0]['new_x']
    normalized = analyzer.utils.normalize_points(points)
    spherical = analyzer.utils.cartesian_to_spherical(points)

    print("\nPoint Cloud Statistics:")
    print(f"Original range: [{points.min():.4f}, {points.max():.4f}]")
    print(f"Normalized range: [{normalized.min():.4f}, {normalized.max():.4f
    print(f"Spherical coordinates shape: {spherical.shape}")
```

```
Geometric Analysis:
Center of mass: [ 0.04778343  0.53634458 -0.03896176]
Mean distance from center: 0.8495
Max distance from center: 3.3704
```




```
Point Cloud Statistics:
Original range: [-1.1581, 3.8908]
Normalized range: [0.0000, 1.0000]
Spherical coordinates shape: (110, 3)
```

Plot data points

```python
import os
import numpy as np
import pandas as pd
import pickle
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

class MMRUtils:
    @staticmethod
    def normalize_points(points):
        """Normalize point cloud data to [0,1] range"""
        min_vals = np.min(points, axis=0)
        max_vals = np.max(points, axis=0)
        normalized = (points - min_vals) / (max_vals - min_vals)
        return normalized

```

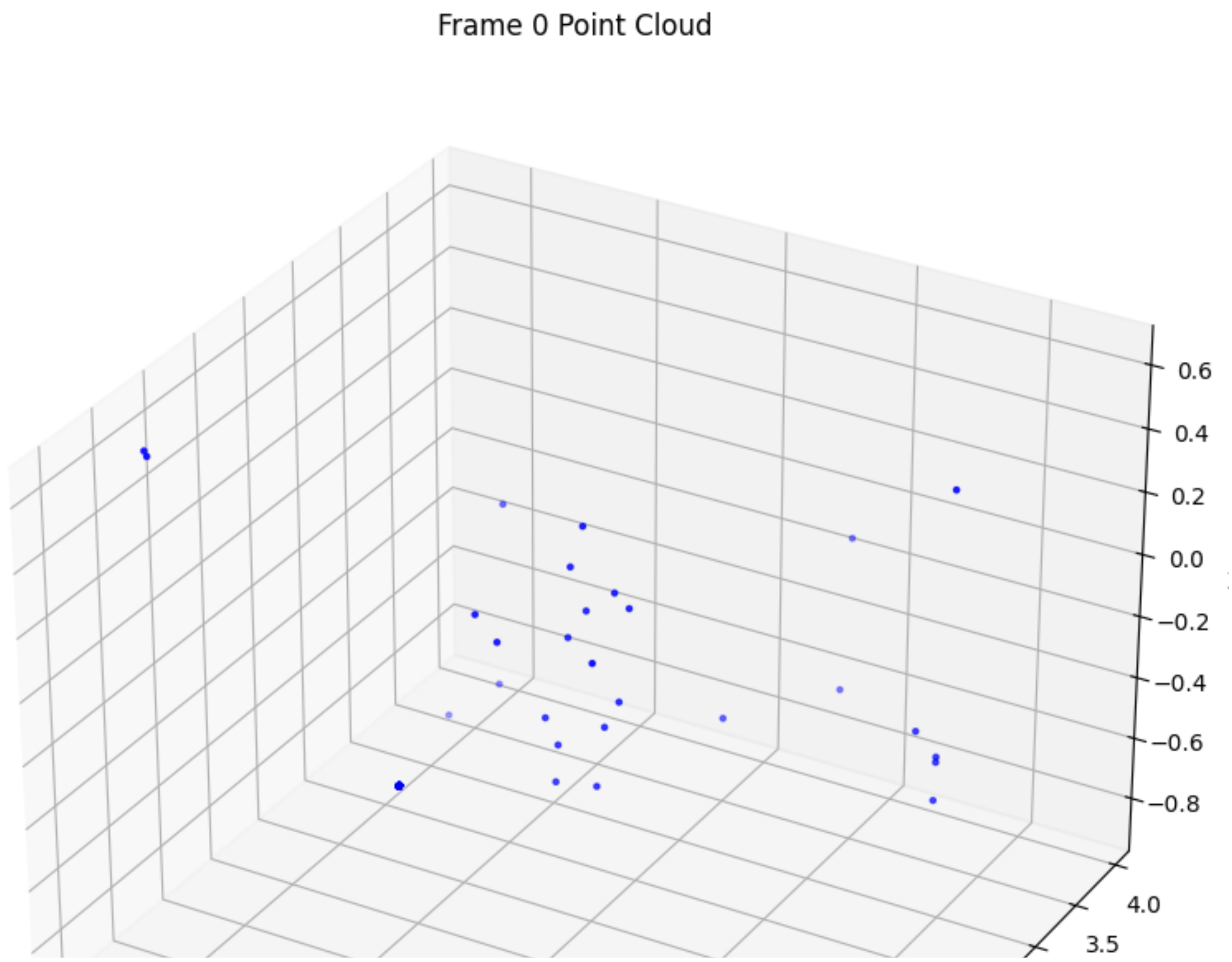
```
    @staticmethod
    def cartesian_to_spherical(points):
        """Convert cartesian to spherical coordinates"""
        x, y, z = points[:, 0], points[:, 1], points[:, 2]
        r = np.sqrt(x**2 + y**2 + z**2)
        theta = np.arctan2(y, x)
        phi = np.arccos(z/r)
        return np.stack([r, theta, phi], axis=1)

    @staticmethod
    def spherical_to_cartesian(points):
        """Convert spherical to cartesian coordinates"""
        r, theta, phi = points[:, 0], points[:, 1], points[:, 2]
        x = r * np.sin(phi) * np.cos(theta)
        y = r * np.sin(phi) * np.sin(theta)
        z = r * np.cos(phi)
        return np.stack([x, y, z], axis=1)

class EnhancedMMRAnalyzer:
    def __init__(self, data_path):
        self.utils = MMRUtils()
        self.data_path = data_path
        self.load_data()

    def load_data(self):
        """Load and verify data"""
        try:
            with open(self.data_path, 'rb') as f:
                self.data = pickle.load(f)
            self.train_data = self.data['train']
            print(f"Loaded {len(self.train_data)} training samples")
        except Exception as e:
            print(f"Error loading data: {e}")
            raise

    def visualize_frame(self, frame_idx=0):
        """Basic frame visualization"""
        try:
            data_point = self.train_data[frame_idx]
            points = data_point['new_x']

            # Create simple 3D scatter plot
            fig = plt.figure(figsize=(10, 10))
            ax = fig.add_subplot(111, projection='3d')

            # Plot points
            scatter = ax.scatter(points[:, 0], points[:, 1], points[:, 2],
                                 c='blue', marker='.')

            ax.set_xlabel('X')
            ax.set_ylabel('Y')
```

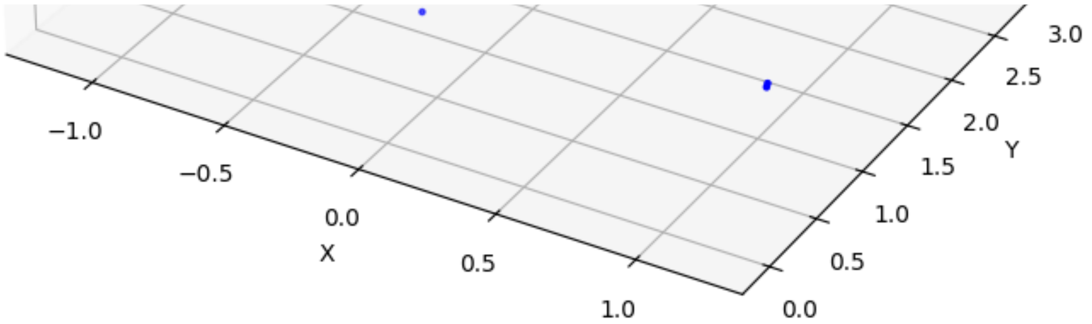
```
            ax.set_zlabel('Z')
            ax.set_title(f'Frame {frame_idx} Point Cloud')

            # Print statistics
            print("\nFrame Statistics:")
            print(f"Number of points: {len(points)}")
            print(f"X range: [{points[:, 0].min():.4f}, {points[:, 0].max():
            print(f"Y range: [{points[:, 1].min():.4f}, {points[:, 1].max():
            print(f"Z range: [{points[:, 2].min():.4f}, {points[:, 2].max():

            plt.show()
            return points

        except Exception as e:
            print(f"Error visualizing frame: {e}")
            raise

    def visualize_frame_enhanced(self, frame_idx=0):
        """Enhanced visualization with multiple views"""
        try:
            data_point = self.train_data[frame_idx]
            points = data_point['new_x']

            # Calculate geometric properties
            center = np.mean(points, axis=0)
            distances = np.linalg.norm(points - center, axis=1)

            # Print analysis
            print("\nGeometric Analysis:")
            print(f"Center of mass: {center}")
            print(f"Mean distance from center: {np.mean(distances):.4f}")
            print(f"Max distance from center: {np.max(distances):.4f}")

            # Create figure with three views
            fig = plt.figure(figsize=(15, 5))

            # Original coordinates
            ax1 = fig.add_subplot(131, projection='3d')
            ax1.scatter(points[:, 0], points[:, 1], points[:, 2], c='blue',
            ax1.set_title('Original Coordinates')
            ax1.set_xlabel('X')
            ax1.set_ylabel('Y')
            ax1.set_zlabel('Z')

            # Normalized coordinates
            norm_points = self.utils.normalize_points(points)
            ax2 = fig.add_subplot(132, projection='3d')
            ax2.scatter(norm_points[:, 0], norm_points[:, 1], norm_points[:,
                       c='red', marker='.')
            ax2.set_title('Normalized Coordinates')
            ax2.set_xlabel('X')
```

```
            ax2.set_ylabel('Y')
            ax2.set_zlabel('Z')

            # Spherical coordinates view
            sph_points = self.utils.cartesian_to_spherical(points)
            sph_cart = self.utils.spherical_to_cartesian(sph_points)
            ax3 = fig.add_subplot(133, projection='3d')
            ax3.scatter(sph_cart[:, 0], sph_cart[:, 1], sph_cart[:, 2],
                       c='green', marker='.')
            ax3.set_title('Spherical Coordinates')
            ax3.set_xlabel('X')
            ax3.set_ylabel('Y')
            ax3.set_zlabel('Z')

            plt.tight_layout()
            plt.show()

            # Print ranges
            print("\nPoint Cloud Statistics:")
            print(f"Original range: [{points.min():.4f}, {points.max():.4f}]
            print(f"Normalized range: [{norm_points.min():.4f}, {norm_points
            print(f"Spherical coordinates shape: {sph_points.shape}")

            return points

        except Exception as e:
            print(f"Error in enhanced visualization: {e}")
            raise

    def compare_frames(self, frame_indices):
        """Compare multiple frames"""
        try:
            n_frames = len(frame_indices)
            fig = plt.figure(figsize=(5*n_frames, 5))

            for i, idx in enumerate(frame_indices):
                ax = fig.add_subplot(1, n_frames, i+1, projection='3d')
                points = self.train_data[idx]['new_x']

                ax.scatter(points[:, 0], points[:, 1], points[:, 2],
                          c='blue', marker='.')
                ax.set_title(f'Frame {idx}')
                ax.set_xlabel('X')
                ax.set_ylabel('Y')
                ax.set_zlabel('Z')

            plt.tight_layout()
            plt.show()

        except Exception as e:
            print(f"Error comparing frames: {e}")
```

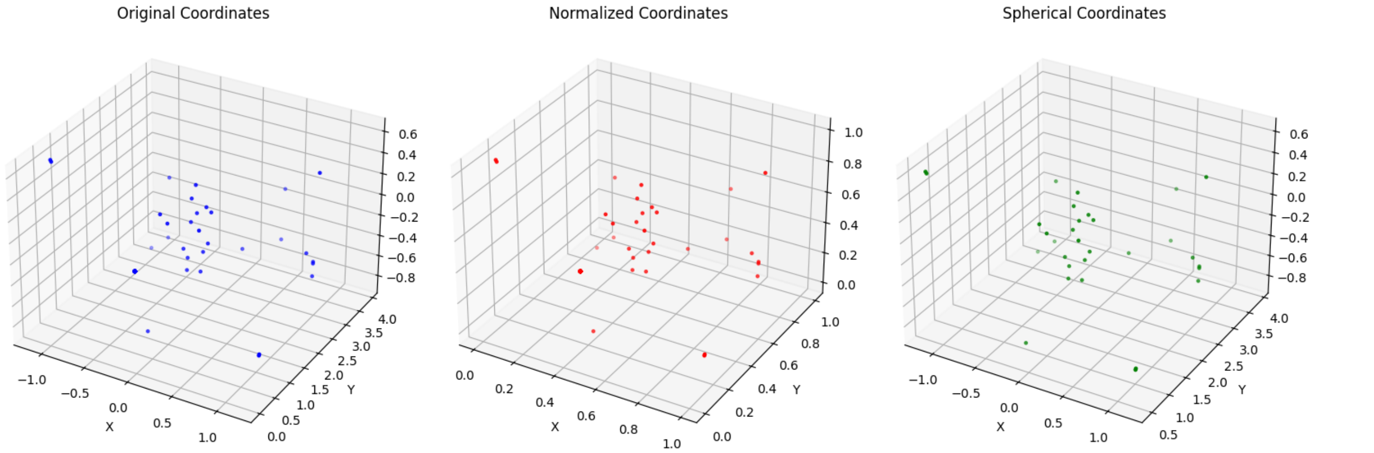
```
                raise

# Example usage
if __name__ == "__main__":
    data_path = '/content/drive/MyDrive/action/data/processed/mmr_action/dat
    analyzer = EnhancedMMRAnalyzer(data_path)

    # Basic visualization
    print("\nBasic Visualization:")
    analyzer.visualize_frame(0)

    # Enhanced visualization
    print("\nEnhanced Visualization:")
    analyzer.visualize_frame_enhanced(0)

    # Compare multiple frames
    print("\nComparing Multiple Frames:")
    analyzer.compare_frames([0, 1, 2])
```

```
Loaded 170336 training samples

Basic Visualization:

Frame Statistics:
Number of points: 110
X range: [-1.1581, 1.2027]
Y range: [0.0000, 3.8908]
Z range: [-0.8684, 0.6147]
```

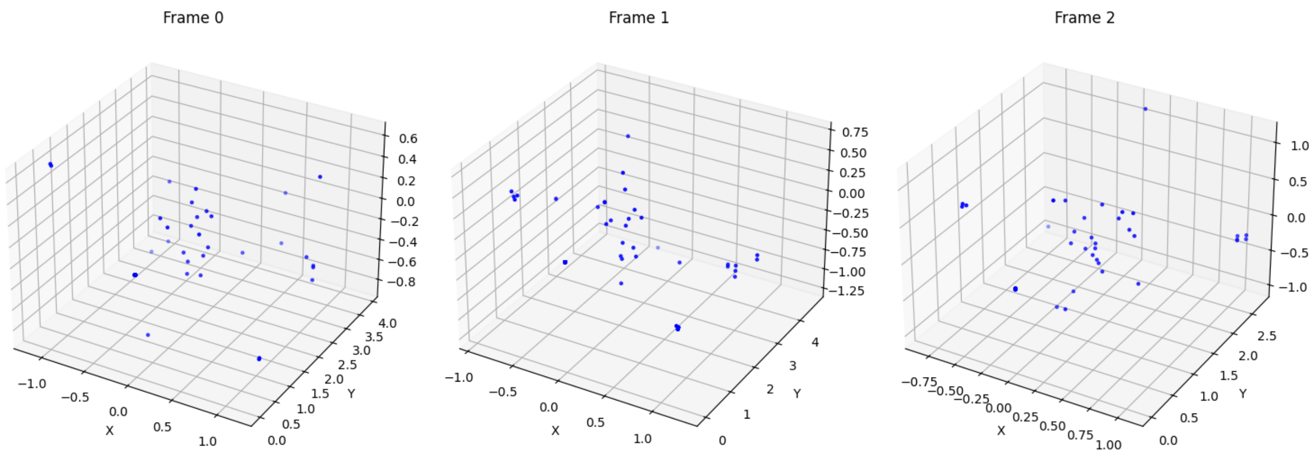





```
Enhanced Visualization:

Geometric Analysis:
Center of mass: [ 0.04778343  0.53634458 -0.03896176]
Mean distance from center: 0.8495
Max distance from center: 3.3704
<ipython-input-9-f8f139a9df25>:23: RuntimeWarning: invalid value encountere
  phi = np.arccos(z/r)
```




```
Point Cloud Statistics:
Original range: [-1.1581, 3.8908]
Normalized range: [0.0000, 1.0000]
Spherical coordinates shape: (110, 3)

Comparing Multiple Frames:
```



Choose frame window

```
WINDOW = 5
```

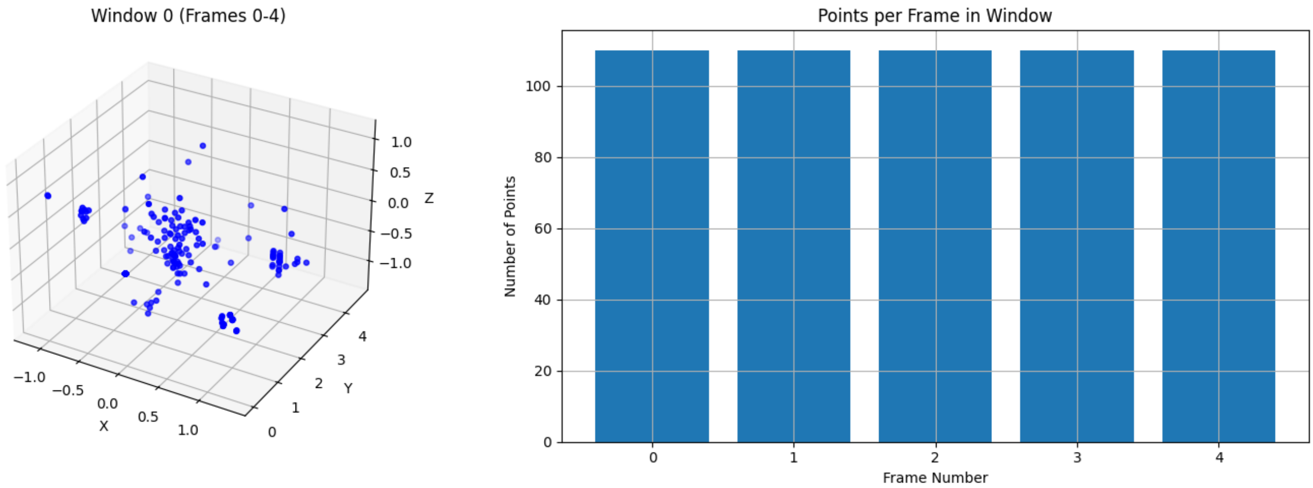
```
def analyze_mmr_windows(data_path, window_size=5):
    """Analyze MMR data with sliding windows"""

    # Load data
    with open(data_path, 'rb') as f:
        all_data = pickle.load(f)
    train_data = all_data['train']

    print(f"Analyzing {len(train_data)} frames with window size {window_size

    # Single frame analysis
    frame_points = []
    for data_point in train_data:
        points = data_point['new_x']
        frame_points.append(len(points))

    frame_series = pd.Series(frame_points)
    print("\nPoints per Frame Statistics:")
    print(frame_series.describe())

    # Window analysis
    window_points = []
```

```python
    num_windows = len(train_data) // window_size

    for i in range(num_windows):
        start_idx = i * window_size
        end_idx = start_idx + window_size
        window_count = sum(frame_points[start_idx:end_idx])
        window_points.append(window_count)

    window_series = pd.Series(window_points)
    print(f"\nPoints per {window_size}-Frame Window Statistics:")
    print(window_series.describe())

    # Visualize first window
    visualize_window(train_data, 0, window_size)

    # Compare first few windows
    compare_windows(train_data, [0, 1, 2], window_size)

    return frame_series, window_series

def visualize_window(train_data, window_idx, window_size):
    """Visualize all points in a window"""
    start_idx = window_idx * window_size
    end_idx = start_idx + window_size

    # Collect points
    all_points = []
    frame_counts = []

    for i in range(start_idx, min(end_idx, len(train_data))):
        points = train_data[i]['new_x']
        all_points.append(points)
        frame_counts.append(len(points))

    all_points = np.vstack(all_points)

    # Create visualization
    fig = plt.figure(figsize=(15, 5))

    # 3D scatter plot
    ax1 = fig.add_subplot(121, projection='3d')
    scatter = ax1.scatter(all_points[:, 0], all_points[:, 1], all_points[:,
                         c='blue', marker='.', s=50)
    ax1.set_xlabel('X')
    ax1.set_ylabel('Y')
    ax1.set_zlabel('Z')
    ax1.set_title(f'Window {window_idx} (Frames {start_idx}-{end_idx-1})')
    ax1.grid(True)

    # Point count distribution
    ax2 = fig.add_subplot(122)
```

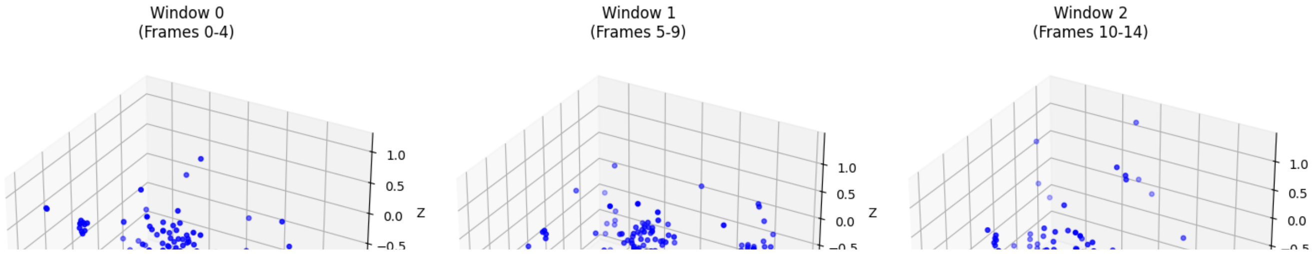
```
    ax2.bar(range(start_idx, end_idx), frame_counts)
    ax2.set_xlabel('Frame Number')
    ax2.set_ylabel('Number of Points')
    ax2.set_title('Points per Frame in Window')
    ax2.grid(True)

    plt.tight_layout()
    display(plt.gcf())
    plt.close()

    # Print statistics
    print(f"\nWindow {window_idx} Statistics:")
    print(f"Total frames: {len(frame_counts)}")
    print(f"Total points: {len(all_points)}")
    print(f"Average points per frame: {np.mean(frame_counts):.2f}")
    print(f"Point range: [{all_points.min():.4f}, {all_points.max():.4f}]")

    return all_points, frame_counts

def compare_windows(train_data, window_indices, window_size):
    """Compare multiple windows"""
    n_windows = len(window_indices)
    fig = plt.figure(figsize=(5*n_windows, 5))

    for i, idx in enumerate(window_indices):
        start_idx = idx * window_size
        end_idx = start_idx + window_size

        # Collect window points
        all_points = []
        for j in range(start_idx, min(end_idx, len(train_data))):
            points = train_data[j]['new_x']
            all_points.append(points)
        all_points = np.vstack(all_points)

        # Plot
        ax = fig.add_subplot(1, n_windows, i+1, projection='3d')
        ax.scatter(all_points[:, 0], all_points[:, 1], all_points[:, 2],
                   c='blue', marker='.', s=50)
        ax.set_title(f'Window {idx}\n(Frames {start_idx}-{end_idx-1})')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        ax.grid(True)

    plt.tight_layout()
    display(plt.gcf())
    plt.close()

# Example usage
if __name__ == "__main__":
```

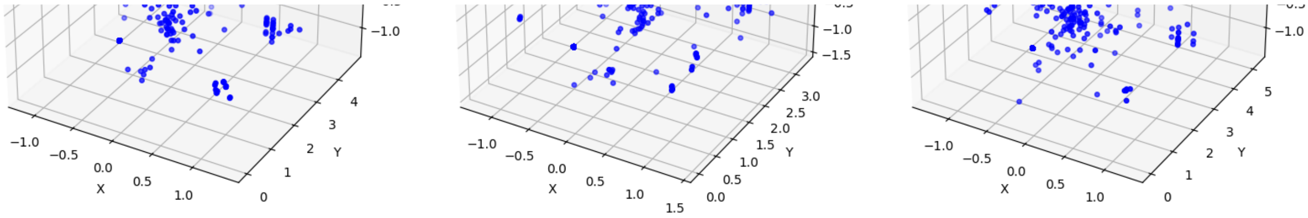
```python
# Set parameters
data_path = '/content/drive/MyDrive/action/data/processed/mmr_action/dat
WINDOW_SIZE = 5

# Run analysis
frame_stats, window_stats = analyze_mmr_windows(data_path, WINDOW_SIZE)
```

```
Analyzing 170336 frames with window size 5

Points per Frame Statistics:
count    170336.0
mean        110.0
std           0.0
min         110.0
25%         110.0
50%         110.0
75%         110.0
max         110.0
dtype: float64

Points per 5-Frame Window Statistics:
count    34067.0
mean       550.0
std          0.0
min        550.0
25%        550.0
50%        550.0
75%        550.0
max        550.0
dtype: float64
```




```
Window 0 Statistics:
Total frames: 5
Total points: 550
Average points per frame: 110.00
Point range: [-1.3165, 4.7211]
```





DBSCAN clustering

```
from sklearn.cluster import DBSCAN
```

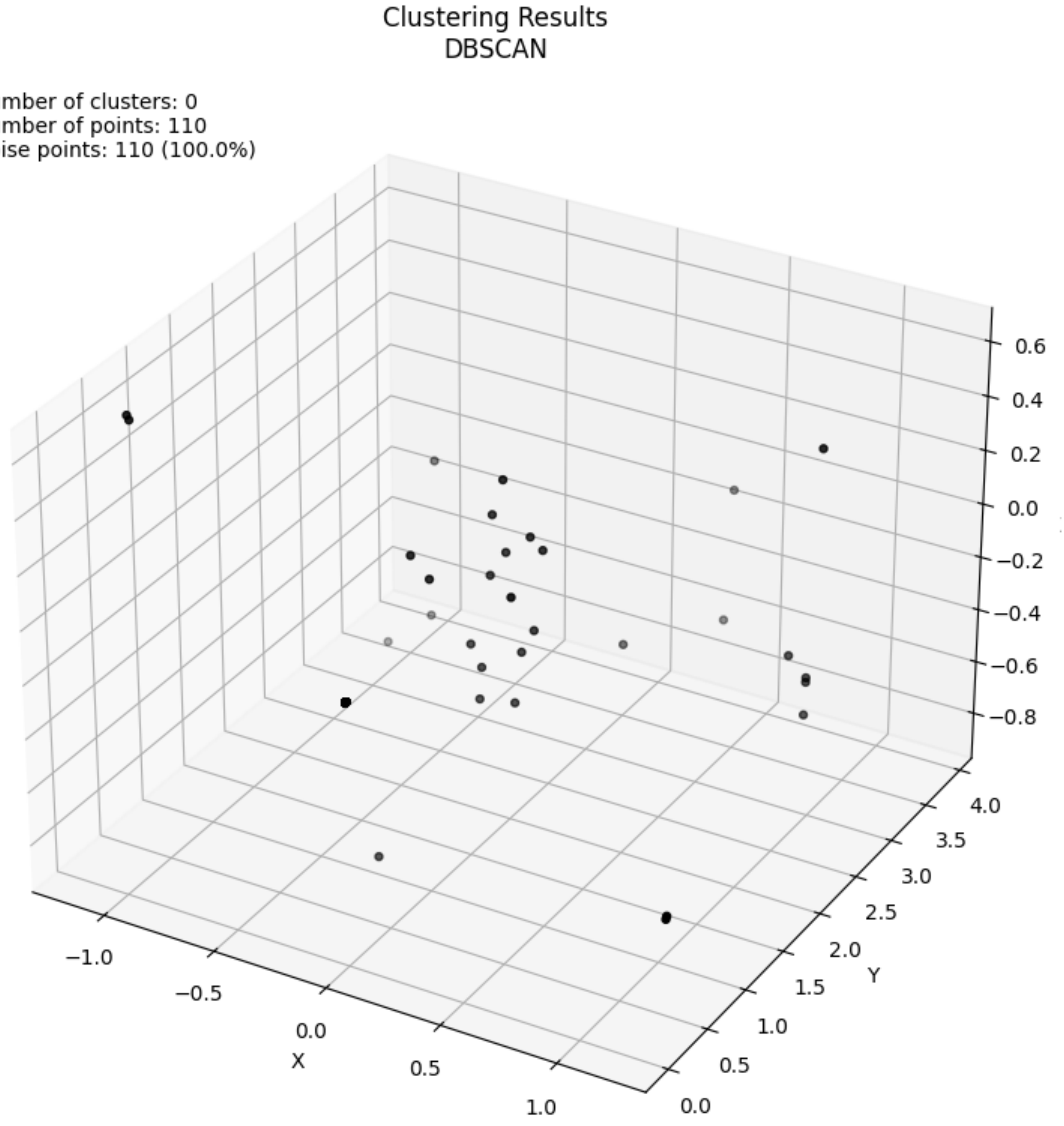
```
# Define global variables
DBSCAN_EPS = 0.3
DBSCAN_SAMPLES = 6
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

def get_colors(label_list):
    """Get colors for cluster labels"""
    color_list = ['r', 'b', 'g', 'c', 'm', 'darkorange', 'deepskyblue',
                  'blueviolet', 'crimson', 'orangered', 'k']
    return [color_list[label % len(color_list)] for label in label_list]

def normalize_weights(points):
    """Normalize point weights to [0,1]"""
    return MinMaxScaler().fit_transform(
        np.linalg.norm(points, axis=1).reshape(-1, 1)
    ).ravel()
```

```python
def model_cluster(points, model, use_weights=True, show_plot=True, return_clu
    """
    Cluster point cloud data using the provided model

    Args:
        points: numpy array of shape (N, 3) containing point cloud coordinate
        model: clustering model (e.g., DBSCAN, KMeans)
        use_weights: whether to use point distances as weights
        show_plot: whether to show clustering visualization
        return_cluster: whether to return the clustering model

    Returns:
        clustering model if return_cluster is True
    """
    # Prepare weights if needed
    if use_weights:
        sample_weights = normalize_weights(points)
    else:
        sample_weights = None

    # Perform clustering
    clustering = model.fit(points, sample_weight=sample_weights)

    # Visualize if requested
    if show_plot:
        fig = plt.figure(figsize=(12, 8))
        ax = fig.add_subplot(111, projection='3d')

        # Plot points colored by cluster
        scatter = ax.scatter(points[:, 0], points[:, 1], points[:, 2],
                           c=get_colors(clustering.labels_),
                           marker='.',
                           s=50)

        # Add cluster centers if available (e.g., for KMeans)
        if hasattr(model, 'cluster_centers_'):
            centers = model.cluster_centers_
            ax.scatter(centers[:, 0], centers[:, 1], centers[:, 2],
                      c='black',
                      marker='*',
                      s=200,
                      label='Cluster Centers')

        # Configure plot
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        ax.set_title(f'Clustering Results\n{model.__class__.__name__}')
        ax.grid(True)

        # Add statistics annotation
```

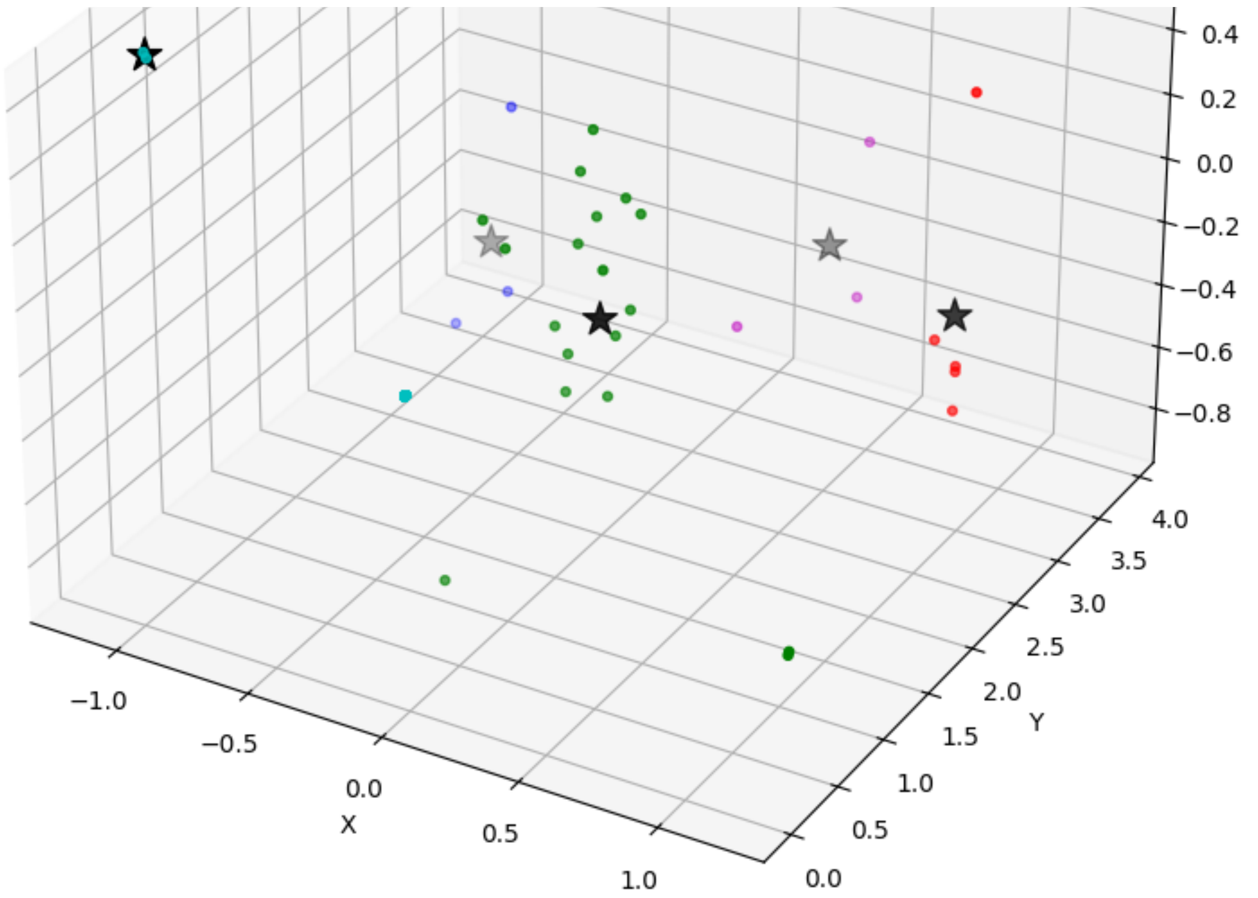
```python
        # Add statistics annotation
        n_clusters = len(set(clustering.labels_)) - (1 if -1 in clustering.la
        stats_text = f'Number of clusters: {n_clusters}\n'
        stats_text += f'Number of points: {len(points)}\n'
        if -1 in clustering.labels_:
            n_noise = np.sum(clustering.labels_ == -1)
            stats_text += f'Noise points: {n_noise} ({n_noise/len(points)*100
        ax.text2D(0.02, 0.98, stats_text,
                  transform=ax.transAxes,
                  verticalalignment='top')

        plt.tight_layout()
        display(plt.gcf())
        plt.close()

        # Print cluster sizes
        print("\nCluster Sizes:")
        unique_labels = sorted(set(clustering.labels_))
        for label in unique_labels:
            count = np.sum(clustering.labels_ == label)
            if label == -1:
                print(f"Noise points: {count} ({count/len(points)*100:.1f}%)"
            else:
                print(f"Cluster {label}: {count} points ({count/len(points)*1

    if return_cluster:
        return clustering

# Example usage
if __name__ == "__main__":
    # Load your data
    data_path = '/content/drive/MyDrive/action/data/processed/mmr_action/data
    with open(data_path, 'rb') as f:
        data = pickle.load(f)

    # Get points from first frame
    points = data['train'][0]['new_x']

    # Try different clustering models
    from sklearn.cluster import DBSCAN, KMeans

    # DBSCAN clustering
    dbscan = DBSCAN(eps=0.3, min_samples=5)
    print("\nDBSCAN Clustering:")
    dbscan_result = model_cluster(points, dbscan, use_weights=True, return_cl

    # KMeans clustering
    kmeans = KMeans(n_clusters=5, random_state=42)
    print("\nKMeans Clustering:")
    kmeans_result = model_cluster(points, kmeans, use_weights=True, return_cl
```

DBSCAN Clustering:



Cluster Sizes:
Noise points: 110 (100.0%)

KMeans Clustering:




```
Cluster Sizes:
Cluster 0: 5 points (4.5%)
Cluster 1: 3 points (2.7%)
Cluster 2: 18 points (16.4%)
Cluster 3: 81 points (73.6%)
Cluster 4: 3 points (2.7%)
```

Compare with and without sample_weight

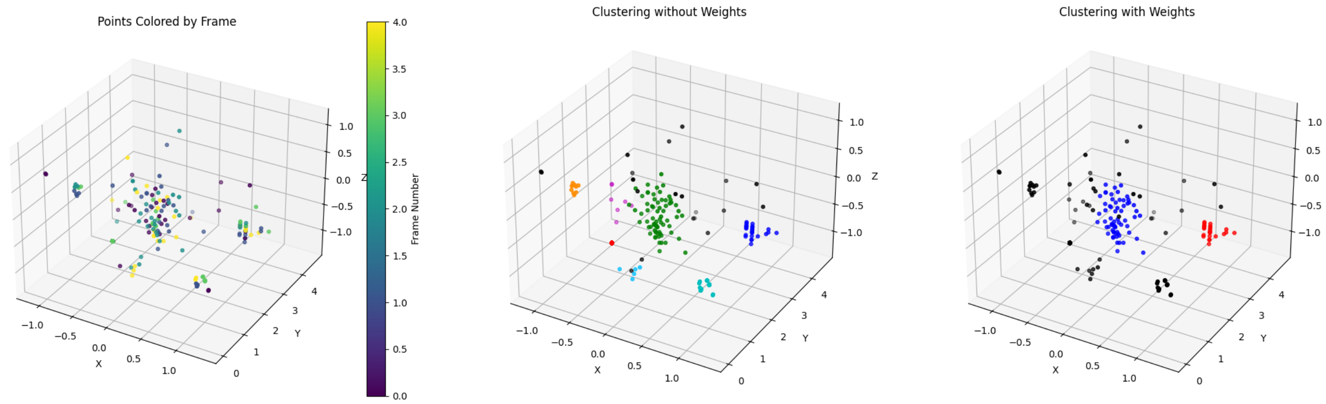
```
def cluster_window(data_points, start_frame, window_size, eps=0.3, min_sampl
    """
```

```
    Cluster points in a specific frame window

    Args:
        data_points: List of frame data points
        start_frame: Starting frame index
        window_size: Number of frames in window
        eps: DBSCAN epsilon parameter
        min_samples: DBSCAN min_samples parameter
        use_weights: Whether to use intensity weights
    """
    # Collect points from window
    window_points = []
    frame_indices = []

    end_frame = start_frame + window_size
    for i in range(start_frame, end_frame):
        if i < len(data_points):
            points = data_points[i]['new_x']
            window_points.append(points)
            frame_indices.extend([i] * len(points))

    window_points = np.vstack(window_points)
    frame_indices = np.array(frame_indices)

    # Create DBSCAN model
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)

    # Prepare sample weights if needed
    if use_weights:
        weights = normalize_weights(window_points)
    else:
        weights = None

    # Perform clustering
    clustering = dbscan.fit(window_points, sample_weight=weights)

    # Visualize results
    fig = plt.figure(figsize=(20, 6))

    # Original points colored by frame
    ax1 = fig.add_subplot(131, projection='3d')
    scatter1 = ax1.scatter(window_points[:, 0], window_points[:, 1], window_
                          c=frame_indices, cmap='viridis',
                          marker='.', s=50)
    ax1.set_title('Points Colored by Frame')
    ax1.set_xlabel('X')
    ax1.set_ylabel('Y')
    ax1.set_zlabel('Z')
    plt.colorbar(scatter1, label='Frame Number')

    # Clustering without weights
```

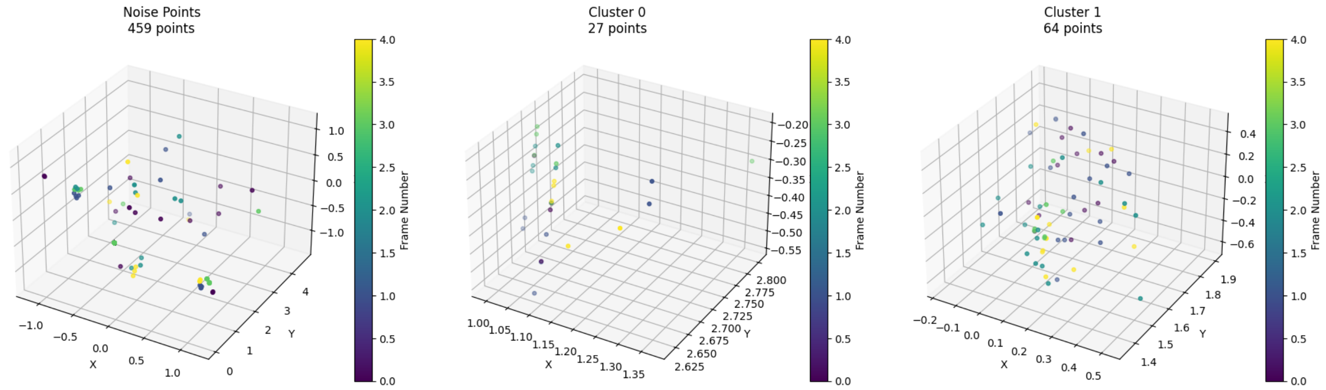
```
    dbscan_no_weights = DBSCAN(eps=eps, min_samples=min_samples)
    labels_no_weights = dbscan_no_weights.fit_predict(window_points)

    ax2 = fig.add_subplot(132, projection='3d')
    scatter2 = ax2.scatter(window_points[:, 0], window_points[:, 1], window_
                          c=get_colors(labels_no_weights),
                          marker='.', s=50)
    ax2.set_title('Clustering without Weights')
    ax2.set_xlabel('X')
    ax2.set_ylabel('Y')
    ax2.set_zlabel('Z')

    # Clustering with weights
    ax3 = fig.add_subplot(133, projection='3d')
    scatter3 = ax3.scatter(window_points[:, 0], window_points[:, 1], window_
                          c=get_colors(clustering.labels_),
                          marker='.', s=50)
    ax3.set_title('Clustering with Weights')
    ax3.set_xlabel('X')
    ax3.set_ylabel('Y')
    ax3.set_zlabel('Z')

    plt.tight_layout()
    display(plt.gcf())
    plt.close()

    # Print statistics
    print(f"\nWindow Statistics (Frames {start_frame}-{end_frame-1}):")
    print(f"Total points: {len(window_points)}")

    print("\nClustering without weights:")
    n_clusters_no_weights = len(set(labels_no_weights)) - (1 if -1 in labels
    print(f"Number of clusters: {n_clusters_no_weights}")
    if -1 in labels_no_weights:
        n_noise = np.sum(labels_no_weights == -1)
        print(f"Noise points: {n_noise} ({n_noise/len(window_points)*100:.1f

    print("\nClustering with weights:")
    n_clusters = len(set(clustering.labels_)) - (1 if -1 in clustering.label
    print(f"Number of clusters: {n_clusters}")
    if -1 in clustering.labels_:
        n_noise = np.sum(clustering.labels_ == -1)
        print(f"Noise points: {n_noise} ({n_noise/len(window_points)*100:.1f

    return clustering, window_points, frame_indices

def analyze_window_clusters(clustering, window_points, frame_indices):
    """Analyze clusters in a window"""
    unique_labels = sorted(set(clustering.labels_))

    # Create visualization
```

```
    n_clusters = len(unique_labels)
    fig_cols = min(3, n_clusters)
    fig_rows = (n_clusters + fig_cols - 1) // fig_cols

    fig = plt.figure(figsize=(6*fig_cols, 5*fig_rows))

    for i, label in enumerate(unique_labels):
        mask = clustering.labels_ == label
        cluster_points = window_points[mask]
        cluster_frames = frame_indices[mask]

        ax = fig.add_subplot(fig_rows, fig_cols, i+1, projection='3d')
        scatter = ax.scatter(cluster_points[:, 0],
                             cluster_points[:, 1],
                             cluster_points[:, 2],
                             c=cluster_frames,
                             cmap='viridis',
                             marker='.',
                             s=50)

        title = "Noise Points" if label == -1 else f"Cluster {label}"
        title += f"\n{len(cluster_points)} points"
        ax.set_title(title)
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.colorbar(scatter, label='Frame Number')

    plt.tight_layout()
    display(plt.gcf())
    plt.close()

# Example usage
if __name__ == "__main__":
    # Parameters
    WINDOW = 5
    DBSCAN_EPS = 0.3
    DBSCAN_SAMPLES = 5
    START_FRAME = 0

    # Load data
    data_path = '/content/drive/MyDrive/action/data/processed/mmr_action/dat
    with open(data_path, 'rb') as f:
        data = pickle.load(f)

    # Perform clustering
    clustering, points, frames = cluster_window(
        data['train'],
        start_frame=START_FRAME,
        window_size=WINDOW,
        eps=DBSCAN_EPS,
```

```python
        min_samples=DBSCAN_SAMPLES
    )

    # Analyze clusters
    analyze_window_clusters(clustering, points, frames)
```




```
Window Statistics (Frames 0-4):
Total points: 550

Clustering without weights:
Number of clusters: 7
Noise points: 20 (3.6%)

Clustering with weights:
Number of clusters: 2
Noise points: 459 (83.5%)
```

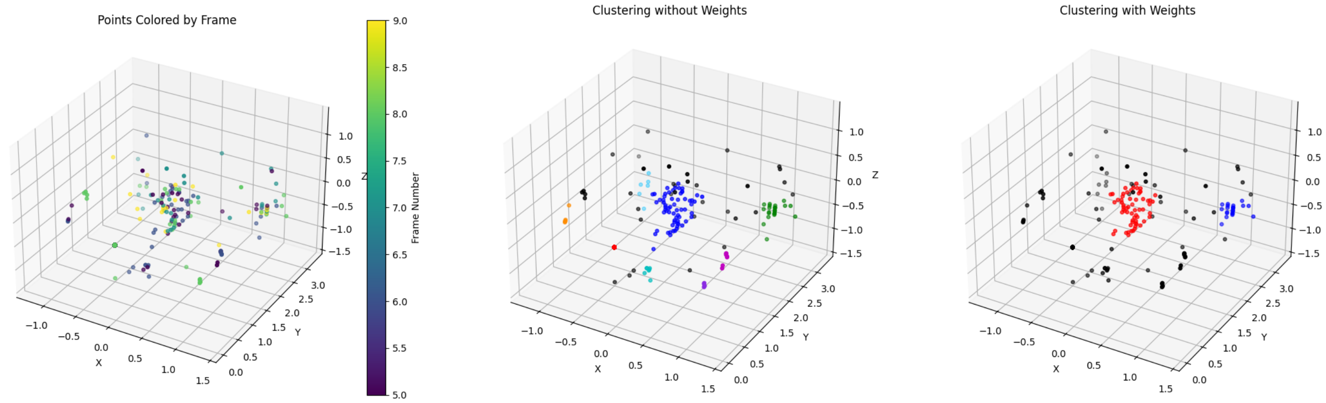


```
def cluster_window(data_points, start_frame, window_size, eps=0.3, min_sampl
    """
    Cluster points in a specific frame window

    Args:
        data_points: List of frame data points
        start_frame: Starting frame index
        window_size: Number of frames in window
        eps: DBSCAN epsilon parameter
        min_samples: DBSCAN min_samples parameter
        use_weights: Whether to use intensity weights
    """
    # Collect points from window
    window_points = []
    frame_indices = []

    end_frame = start_frame + window_size
    for i in range(start_frame, end_frame):
        if i < len(data_points):
            points = data_points[i]['new_x']
            window_points.append(points)
            frame_indices.extend([i] * len(points))

    window_points = np.vstack(window_points)
    frame_indices = np.array(frame_indices)

    # Create DBSCAN model
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)

    # Prepare sample weights if needed
    if use_weights:
        weights = normalize_weights(window_points)
    else:
        weights = None

    # Perform clustering
    clustering = dbscan.fit(window_points, sample_weight=weights)

    # Visualize results
    fig = plt.figure(figsize=(20, 6))

    # Original points colored by frame
    ax1 = fig.add_subplot(131, projection='3d')
    scatter1 = ax1.scatter(window_points[:, 0], window_points[:, 1], window_
                          c=frame_indices, cmap='viridis',
                          marker='.', s=50)
    ax1.set_title('Points Colored by Frame')
    ax1.set_xlabel('X')
    ax1.set_ylabel('Y')
    ax1.set_zlabel('Z')
    plt.colorbar(scatter1, label='Frame Number')
```

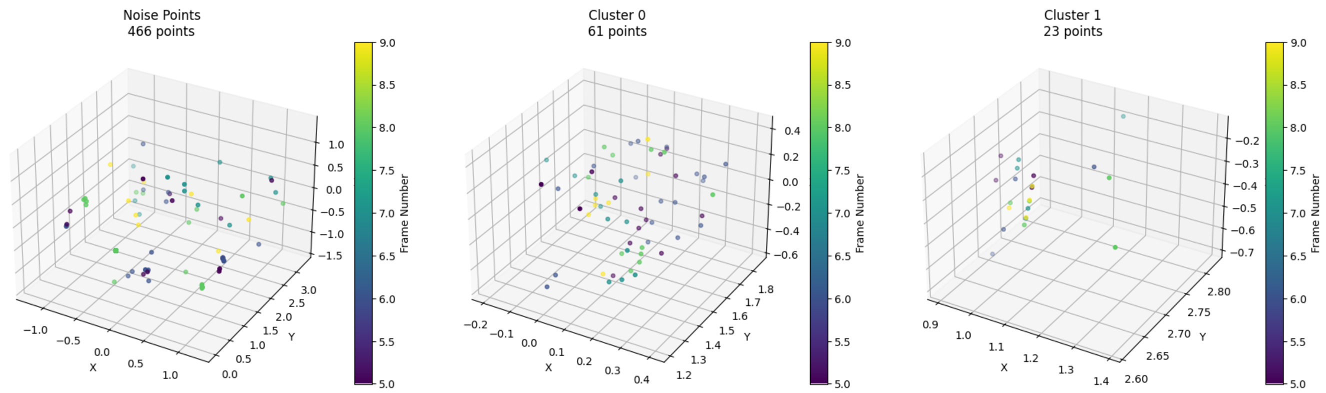
```python

    # Clustering without weights
    dbscan_no_weights = DBSCAN(eps=eps, min_samples=min_samples)
    labels_no_weights = dbscan_no_weights.fit_predict(window_points)

    ax2 = fig.add_subplot(132, projection='3d')
    scatter2 = ax2.scatter(window_points[:, 0], window_points[:, 1], window_
                          c=get_colors(labels_no_weights),
                          marker='.', s=50)
    ax2.set_title('Clustering without Weights')
    ax2.set_xlabel('X')
    ax2.set_ylabel('Y')
    ax2.set_zlabel('Z')

    # Clustering with weights
    ax3 = fig.add_subplot(133, projection='3d')
    scatter3 = ax3.scatter(window_points[:, 0], window_points[:, 1], window_
                          c=get_colors(clustering.labels_),
                          marker='.', s=50)
    ax3.set_title('Clustering with Weights')
    ax3.set_xlabel('X')
    ax3.set_ylabel('Y')
    ax3.set_zlabel('Z')

    plt.tight_layout()
    display(plt.gcf())
    plt.close()

    # Print statistics
    print(f"\nWindow Statistics (Frames {start_frame}-{end_frame-1}):")
    print(f"Total points: {len(window_points)}")

    print("\nClustering without weights:")
    n_clusters_no_weights = len(set(labels_no_weights)) - (1 if -1 in labels
    print(f"Number of clusters: {n_clusters_no_weights}")
    if -1 in labels_no_weights:
        n_noise = np.sum(labels_no_weights == -1)
        print(f"Noise points: {n_noise} ({n_noise/len(window_points)*100:.1f

    print("\nClustering with weights:")
    n_clusters = len(set(clustering.labels_)) - (1 if -1 in clustering.label
    print(f"Number of clusters: {n_clusters}")
    if -1 in clustering.labels_:
        n_noise = np.sum(clustering.labels_ == -1)
        print(f"Noise points: {n_noise} ({n_noise/len(window_points)*100:.1f

    return clustering, window_points, frame_indices

def analyze_window_clusters(clustering, window_points, frame_indices):
    """Analyze clusters in a window"""
    unique_labels = sorted(set(clustering.labels_))
```

```

    # Create visualization
    n_clusters = len(unique_labels)
    fig_cols = min(3, n_clusters)
    fig_rows = (n_clusters + fig_cols - 1) // fig_cols

    fig = plt.figure(figsize=(6*fig_cols, 5*fig_rows))

    for i, label in enumerate(unique_labels):
        mask = clustering.labels_ == label
        cluster_points = window_points[mask]
        cluster_frames = frame_indices[mask]

        ax = fig.add_subplot(fig_rows, fig_cols, i+1, projection='3d')
        scatter = ax.scatter(cluster_points[:, 0],
                             cluster_points[:, 1],
                             cluster_points[:, 2],
                             c=cluster_frames,
                             cmap='viridis',
                             marker='.',
                             s=50)

        title = "Noise Points" if label == -1 else f"Cluster {label}"
        title += f"\n{len(cluster_points)} points"
        ax.set_title(title)
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.colorbar(scatter, label='Frame Number')

    plt.tight_layout()
    display(plt.gcf())
    plt.close()

# Example usage
if __name__ == "__main__":
    # Parameters
    WINDOW = 5
    DBSCAN_EPS = 0.3
    DBSCAN_SAMPLES = 5
    START_FRAME = 5

    # Load data
    data_path = '/content/drive/MyDrive/action/data/processed/mmr_action/dat
    with open(data_path, 'rb') as f:
        data = pickle.load(f)

    # Perform clustering
    clustering, points, frames = cluster_window(
        data['train'],
        start_frame=START_FRAME,
```

```
        window_size=WINDOW,
        eps=DBSCAN_EPS,
        min_samples=DBSCAN_SAMPLES
    )

    # Analyze clusters
    analyze_window_clusters(clustering, points, frames)
```




```
Window Statistics (Frames 5-9):
Total points: 550

Clustering without weights:
Number of clusters: 8
Noise points: 35 (6.4%)

Clustering with weights:
Number of clusters: 2
Noise points: 466 (84.7%)
```



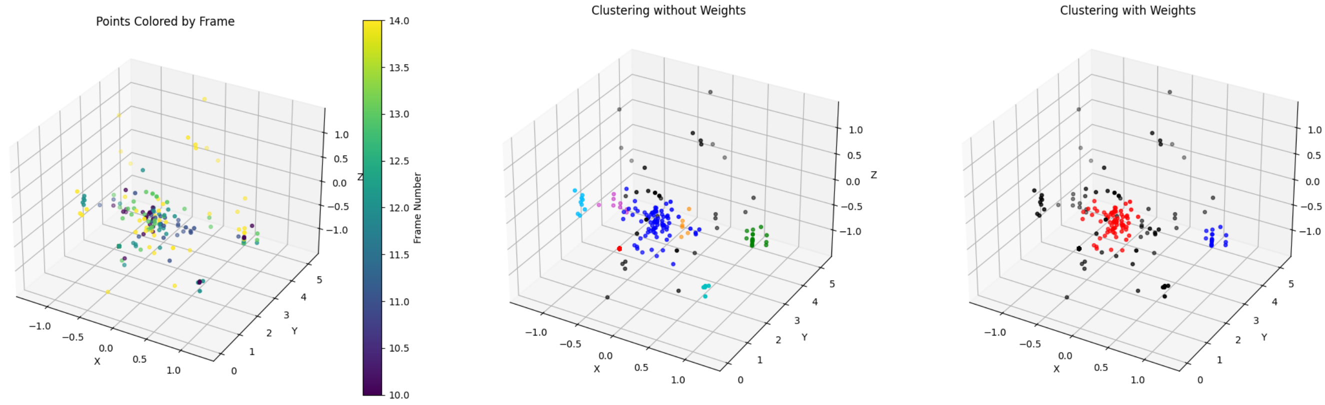

```
def cluster_window(data_points, start_frame, window_size, eps=0.3, min_sampl
    """
    Cluster points in a specific frame window

    Args:
```

```
        data_points: List of frame data points
        start_frame: Starting frame index
        window_size: Number of frames in window
        eps: DBSCAN epsilon parameter
        min_samples: DBSCAN min_samples parameter
        use_weights: Whether to use intensity weights
    """
    # Collect points from window
    window_points = []
    frame_indices = []

    end_frame = start_frame + window_size
    for i in range(start_frame, end_frame):
        if i < len(data_points):
            points = data_points[i]['new_x']
            window_points.append(points)
            frame_indices.extend([i] * len(points))

    window_points = np.vstack(window_points)
    frame_indices = np.array(frame_indices)

    # Create DBSCAN model
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)

    # Prepare sample weights if needed
    if use_weights:
        weights = normalize_weights(window_points)
    else:
        weights = None

    # Perform clustering
    clustering = dbscan.fit(window_points, sample_weight=weights)

    # Visualize results
    fig = plt.figure(figsize=(20, 6))

    # Original points colored by frame
    ax1 = fig.add_subplot(131, projection='3d')
    scatter1 = ax1.scatter(window_points[:, 0], window_points[:, 1], window_
                          c=frame_indices, cmap='viridis',
                          marker='.', s=50)
    ax1.set_title('Points Colored by Frame')
    ax1.set_xlabel('X')
    ax1.set_ylabel('Y')
    ax1.set_zlabel('Z')
    plt.colorbar(scatter1, label='Frame Number')

    # Clustering without weights
    dbscan_no_weights = DBSCAN(eps=eps, min_samples=min_samples)
    labels_no_weights = dbscan_no_weights.fit_predict(window_points)

```

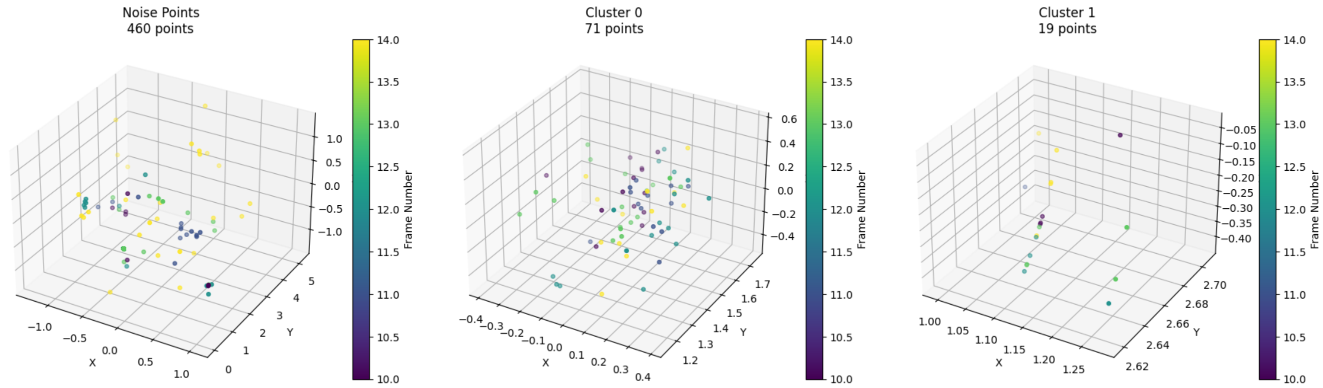
```
    ax2 = fig.add_subplot(132, projection='3d')
    scatter2 = ax2.scatter(window_points[:, 0], window_points[:, 1], window_
                          c=get_colors(labels_no_weights),
                          marker='.', s=50)
    ax2.set_title('Clustering without Weights')
    ax2.set_xlabel('X')
    ax2.set_ylabel('Y')
    ax2.set_zlabel('Z')

    # Clustering with weights
    ax3 = fig.add_subplot(133, projection='3d')
    scatter3 = ax3.scatter(window_points[:, 0], window_points[:, 1], window_
                          c=get_colors(clustering.labels_),
                          marker='.', s=50)
    ax3.set_title('Clustering with Weights')
    ax3.set_xlabel('X')
    ax3.set_ylabel('Y')
    ax3.set_zlabel('Z')

    plt.tight_layout()
    display(plt.gcf())
    plt.close()

    # Print statistics
    print(f"\nWindow Statistics (Frames {start_frame}-{end_frame-1}):")
    print(f"Total points: {len(window_points)}")

    print("\nClustering without weights:")
    n_clusters_no_weights = len(set(labels_no_weights)) - (1 if -1 in labels
    print(f"Number of clusters: {n_clusters_no_weights}")
    if -1 in labels_no_weights:
        n_noise = np.sum(labels_no_weights == -1)
        print(f"Noise points: {n_noise} ({n_noise/len(window_points)*100:.1f

    print("\nClustering with weights:")
    n_clusters = len(set(clustering.labels_)) - (1 if -1 in clustering.label
    print(f"Number of clusters: {n_clusters}")
    if -1 in clustering.labels_:
        n_noise = np.sum(clustering.labels_ == -1)
        print(f"Noise points: {n_noise} ({n_noise/len(window_points)*100:.1f

    return clustering, window_points, frame_indices

def analyze_window_clusters(clustering, window_points, frame_indices):
    """Analyze clusters in a window"""
    unique_labels = sorted(set(clustering.labels_))

    # Create visualization
    n_clusters = len(unique_labels)
    fig_cols = min(3, n_clusters)
    fig_rows = (n_clusters + fig_cols - 1) // fig_cols
```

```python

    fig = plt.figure(figsize=(6*fig_cols, 5*fig_rows))

    for i, label in enumerate(unique_labels):
        mask = clustering.labels_ == label
        cluster_points = window_points[mask]
        cluster_frames = frame_indices[mask]

        ax = fig.add_subplot(fig_rows, fig_cols, i+1, projection='3d')
        scatter = ax.scatter(cluster_points[:, 0],
                             cluster_points[:, 1],
                             cluster_points[:, 2],
                             c=cluster_frames,
                             cmap='viridis',
                             marker='.',
                             s=50)

        title = "Noise Points" if label == -1 else f"Cluster {label}"
        title += f"\n{len(cluster_points)} points"
        ax.set_title(title)
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.colorbar(scatter, label='Frame Number')

    plt.tight_layout()
    display(plt.gcf())
    plt.close()

# Example usage
if __name__ == "__main__":
    # Parameters
    WINDOW = 5
    DBSCAN_EPS = 0.3
    DBSCAN_SAMPLES = 5
    START_FRAME = 10

    # Load data
    data_path = '/content/drive/MyDrive/action/data/processed/mmr_action/dat
    with open(data_path, 'rb') as f:
        data = pickle.load(f)

    # Perform clustering
    clustering, points, frames = cluster_window(
        data['train'],
        start_frame=START_FRAME,
        window_size=WINDOW,
        eps=DBSCAN_EPS,
        min_samples=DBSCAN_SAMPLES
    )

```

```python
    # Analyze clusters
    analyze_window_clusters(clustering, points, frames)
```




```
Window Statistics (Frames 10-14):
Total points: 550

Clustering without weights:
Number of clusters: 7
Noise points: 31 (5.6%)

Clustering with weights:
Number of clusters: 2
Noise points: 460 (83.6%)
```



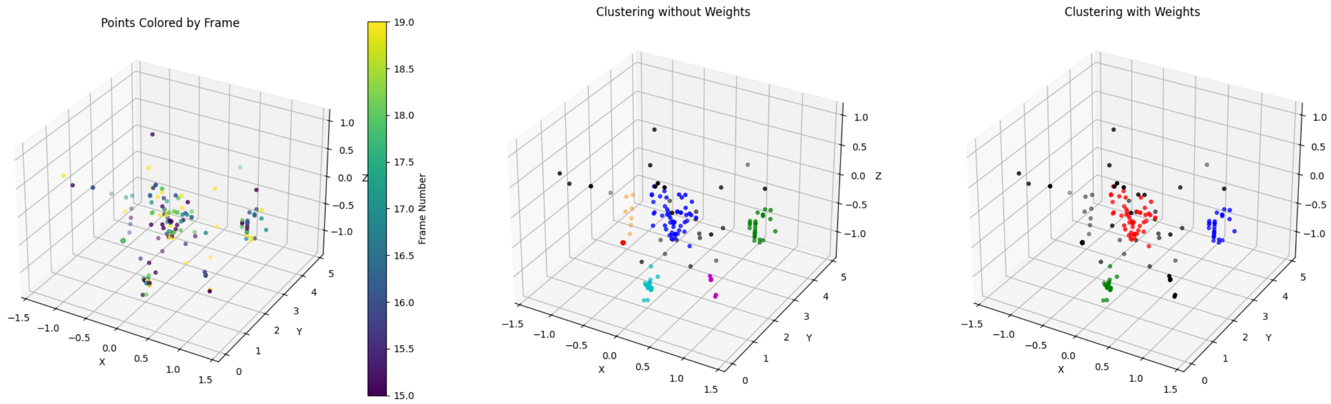

```python
def cluster_window(data_points, start_frame, window_size, eps=0.3, min_sampl
    """
```

```
    Cluster points in a specific frame window

    Args:
        data_points: List of frame data points
        start_frame: Starting frame index
        window_size: Number of frames in window
        eps: DBSCAN epsilon parameter
        min_samples: DBSCAN min_samples parameter
        use_weights: Whether to use intensity weights
    """
    # Collect points from window
    window_points = []
    frame_indices = []

    end_frame = start_frame + window_size
    for i in range(start_frame, end_frame):
        if i < len(data_points):
            points = data_points[i]['new_x']
            window_points.append(points)
            frame_indices.extend([i] * len(points))

    window_points = np.vstack(window_points)
    frame_indices = np.array(frame_indices)

    # Create DBSCAN model
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)

    # Prepare sample weights if needed
    if use_weights:
        weights = normalize_weights(window_points)
    else:
        weights = None

    # Perform clustering
    clustering = dbscan.fit(window_points, sample_weight=weights)

    # Visualize results
    fig = plt.figure(figsize=(20, 6))

    # Original points colored by frame
    ax1 = fig.add_subplot(131, projection='3d')
    scatter1 = ax1.scatter(window_points[:, 0], window_points[:, 1], window_
                          c=frame_indices, cmap='viridis',
                          marker='.', s=50)
    ax1.set_title('Points Colored by Frame')
    ax1.set_xlabel('X')
    ax1.set_ylabel('Y')
    ax1.set_zlabel('Z')
    plt.colorbar(scatter1, label='Frame Number')

    # Clustering without weights
```

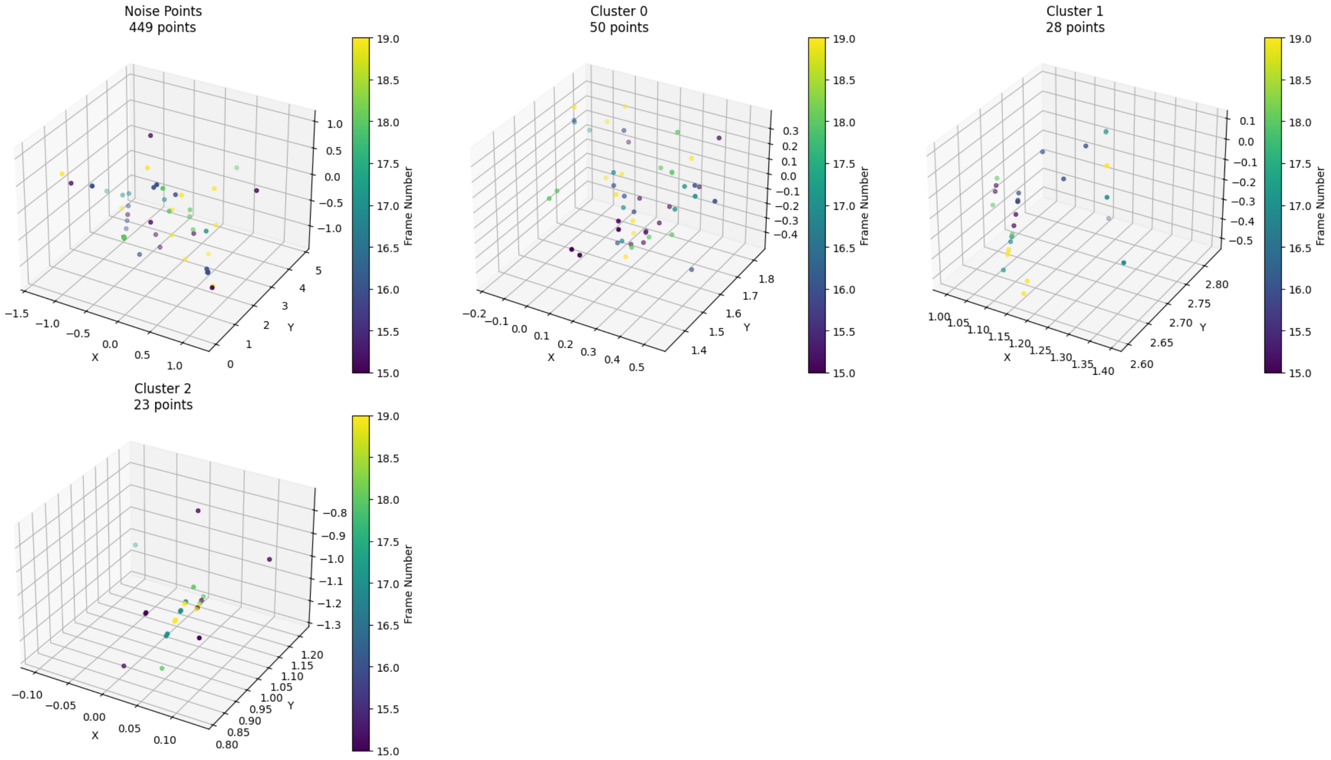
```
    dbscan_no_weights = DBSCAN(eps=eps, min_samples=min_samples)
    labels_no_weights = dbscan_no_weights.fit_predict(window_points)

    ax2 = fig.add_subplot(132, projection='3d')
    scatter2 = ax2.scatter(window_points[:, 0], window_points[:, 1], window_
                           c=get_colors(labels_no_weights),
                           marker='.', s=50)
    ax2.set_title('Clustering without Weights')
    ax2.set_xlabel('X')
    ax2.set_ylabel('Y')
    ax2.set_zlabel('Z')

    # Clustering with weights
    ax3 = fig.add_subplot(133, projection='3d')
    scatter3 = ax3.scatter(window_points[:, 0], window_points[:, 1], window_
                           c=get_colors(clustering.labels_),
                           marker='.', s=50)
    ax3.set_title('Clustering with Weights')
    ax3.set_xlabel('X')
    ax3.set_ylabel('Y')
    ax3.set_zlabel('Z')

    plt.tight_layout()
    display(plt.gcf())
    plt.close()

    # Print statistics
    print(f"\nWindow Statistics (Frames {start_frame}-{end_frame-1}):")
    print(f"Total points: {len(window_points)}")

    print("\nClustering without weights:")
    n_clusters_no_weights = len(set(labels_no_weights)) - (1 if -1 in labels
    print(f"Number of clusters: {n_clusters_no_weights}")
    if -1 in labels_no_weights:
        n_noise = np.sum(labels_no_weights == -1)
        print(f"Noise points: {n_noise} ({n_noise/len(window_points)*100:.1f

    print("\nClustering with weights:")
    n_clusters = len(set(clustering.labels_)) - (1 if -1 in clustering.label
    print(f"Number of clusters: {n_clusters}")
    if -1 in clustering.labels_:
        n_noise = np.sum(clustering.labels_ == -1)
        print(f"Noise points: {n_noise} ({n_noise/len(window_points)*100:.1f

    return clustering, window_points, frame_indices

def analyze_window_clusters(clustering, window_points, frame_indices):
    """Analyze clusters in a window"""
    unique_labels = sorted(set(clustering.labels_))

    # Create visualization
```

```
    n_clusters = len(unique_labels)
    fig_cols = min(3, n_clusters)
    fig_rows = (n_clusters + fig_cols - 1) // fig_cols

    fig = plt.figure(figsize=(6*fig_cols, 5*fig_rows))

    for i, label in enumerate(unique_labels):
        mask = clustering.labels_ == label
        cluster_points = window_points[mask]
        cluster_frames = frame_indices[mask]

        ax = fig.add_subplot(fig_rows, fig_cols, i+1, projection='3d')
        scatter = ax.scatter(cluster_points[:, 0],
                             cluster_points[:, 1],
                             cluster_points[:, 2],
                             c=cluster_frames,
                             cmap='viridis',
                             marker='.',
                             s=50)

        title = "Noise Points" if label == -1 else f"Cluster {label}"
        title += f"\n{len(cluster_points)} points"
        ax.set_title(title)
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.colorbar(scatter, label='Frame Number')

    plt.tight_layout()
    display(plt.gcf())
    plt.close()

# Example usage
if __name__ == "__main__":
    # Parameters
    WINDOW = 5
    DBSCAN_EPS = 0.3
    DBSCAN_SAMPLES = 5
    START_FRAME = 15

    # Load data
    data_path = '/content/drive/MyDrive/action/data/processed/mmr_action/dat
    with open(data_path, 'rb') as f:
        data = pickle.load(f)

    # Perform clustering
    clustering, points, frames = cluster_window(
        data['train'],
        start_frame=START_FRAME,
        window_size=WINDOW,
        eps=DBSCAN_EPS,
```

```
        min_samples=DBSCAN_SAMPLES
    )

    # Analyze clusters
    analyze_window_clusters(clustering, points, frames)
```




```
Window Statistics (Frames 15-19):
Total points: 550

Clustering without weights:
Number of clusters: 6
Noise points: 28 (5.1%)

Clustering with weights:
Number of clusters: 3
Noise points: 449 (81.6%)
```



Change DBSCAN eps, min_samples [TODO]

Estimate boundary for each cluster (extension, orientation)

```
# Clustering frame_num = 0~4
startFrame = 0
dbscan = DBSCAN(eps=0.4, min_samples=10)
datalist = data[(data.frame_num>=startFrame)&(data.frame_num<startFrame+WINDO

clustering = ModelCluster(datalist, dbscan, sample_weight=True, show_plot=Fal
clustering.labels_
```

```
# Number of points in each cluster
def ClusterAnalysis(labels):
    print('Total:', len(labels), 'points,', len(np.unique(labels))-1, 'cluste
    for i in range(np.max(np.unique(labels))+1):
        print('Cluster', i, ':', np.sum(labels==i), 'points')
    print('Noise:', np.sum(labels==-1), 'points')
```


```
# Number of points in each cluster
def ClusterAnalysis(labels):
    print('Total:', len(labels), 'points,', len(np.unique(labels))-1, 'cluste
    for i in range(np.max(np.unique(labels))+1):
        print('Cluster', i, ':', np.sum(labels==i), 'points')
    print('Noise:', np.sum(labels==-1), 'points')
```