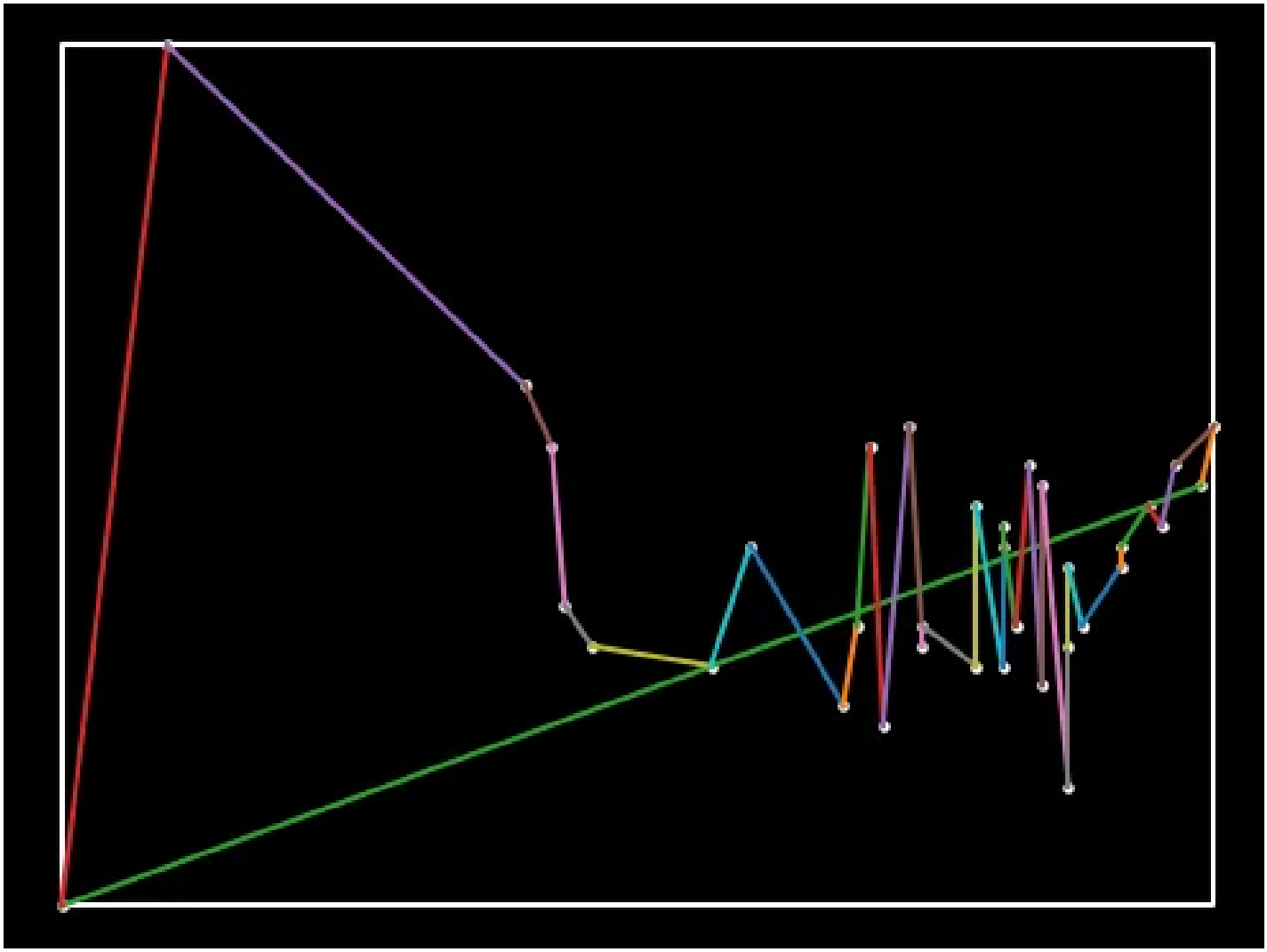
r/ThingsYouDidntKnow • 1 mo. ago
TheStocksGuy

Traveling Statesperson Path

This is a python script you can adjust for visuals only.



```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation
import matplotlib.colors as mcolors

# Function to spread bits (Morton Order)
def spread_bits(v):
    v = (v | (v << 8)) & 0x00FF00FF
    v = (v | (v << 4)) & 0x0F0F0F0F
    v = (v | (v << 2)) & 0x33333333
```

```
# Function to compute Morton Order
def morton_order(city):
    x, y = city['x'], city['y']
    x = int(x * 10000)
    y = int(y * 10000)
    return spread_bits(x) | (spread_bits(y) << 1)

# Function to check if a path intersects with any existing paths
def path_intersects(new_path, existing_paths):
    for path in existing_paths:
        path = [np.array(p) for p in path]
        new_path = [np.array(p) for p in new_path]

        # Ensure all paths are 3D by adding a zero z-coordinate if missing
        if len(path[0]) == 2:
            path[0] = np.append(path[0], 0)
        if len(path[1]) == 2:
            path[1] = np.append(path[1], 0)
        if len(new_path[0]) == 2:
            new_path[0] = np.append(new_path[0], 0)
        if len(new_path[1]) == 2:
            new_path[1] = np.append(new_path[1], 0)

        cross1 = np.cross(path[1] - path[0], new_path[0] - path[0])
        cross2 = np.cross(path[1] - path[0], new_path[1] - path[0])
        cross3 = np.cross(new_path[1] - new_path[0], path[0] - new_path[0])
        cross4 = np.cross(new_path[1] - new_path[0], path[1] - new_path[0])

        if (np.all(cross1 * cross2 <= 0) and np.all(cross3 * cross4 <= 0)):
            return True
    return False

# Function to sort and connect cities from highest to lowest x and y values
def sort_cities(cities):
    if not len(cities):
        print("No cities to process. Please check the input data.")
        return []

    # Sort cities based on the highest value of x and y
    cities_sorted = sorted(cities, key=lambda c: (c['x'], c['y']), reverse=True)
    sorted_path = [cities_sorted.pop(0)]
    existing_paths = []

    while len(cities_sorted):
        current_city = sorted_path[-1]
        current_coords = [current_city.get('x', 0), current_city.get('y', 0)]
```

```
            min_distance = float('inf')
            closest_city_idx = -1

            for i, city in enumerate(cities_sorted):
                city_coords = [city.get('x', 0), city.get('y', 0)]
                if 'z' in city:
                    city_coords.append(city['z'])

                distance = np.linalg.norm(np.array(city_coords) - np.array(current_coords))
                if distance < min_distance:
                    new_path = [np.array(current_coords), np.array(city_coords)]
                    if not path_intersects(new_path, existing_paths):
                        min_distance = distance
                        closest_city_idx = i

            sorted_path.append(cities_sorted.pop(closest_city_idx))
            new_path = [np.array([current_city['x'], current_city['y'], current_city.get('z', 0)]),
                        np.array([sorted_path[-1]['x'], sorted_path[-1]['y'], sorted_path[-1].get(
            existing_paths.append(new_path)

        sorted_path.append(sorted_path[0])  # Complete the loop
        return sorted_path

    # Draw cities and connecting lines within a bounded container
    def draw_cities(ax, sorted_path, zoom_level=1, offset_x=0, offset_y=0):
        ax.clear()
        ax.set_facecolor('black')
        ax.set_xticks([])
        ax.set_yticks([])

        # Find min and max of x and y to define the container
        min_x = min(city.get('x', 0) for city in sorted_path)
        max_x = max(city.get('x', 0) for city in sorted_path)
        min_y = min(city.get('y', 0) for city in sorted_path)
        max_y = max(city.get('y', 0) for city in sorted_path)

        # Draw container outline in white
        container_coords = [
            (min_x, min_y), (min_x, max_y), (max_x, max_y), (max_x, min_y), (min_x, min_y)
        ]
        ax.plot([c[0] + offset_x for c in container_coords],
                [c[1] + offset_y for c in container_coords], color='white')

        colors = list(mcolors.TABLEAU_COLORS.values())

        for index in range(1, len(sorted_path)):
```

```
            x_coords = [prev_city.get('x', 0) + offset_x, city.get('x', 0) + offset_x]
            y_coords = [prev_city.get('y', 0) + offset_y, city.get('y', 0) + offset_y]
            if 'z' in city and 'z' in prev_city:
                z_coords = [prev_city.get('z', 0), city.get('z', 0)]
                ax.plot(x_coords, y_coords, zs=z_coords, color=color)
            else:
                ax.plot(x_coords, y_coords, color=color)

        if 'z' in sorted_path[0]:
            ax.scatter([city.get('x', 0) + offset_x for city in sorted_path],
                       [city.get('y', 0) + offset_y for city in sorted_path],
                       [city.get('z', 0) for city in sorted_path],
                       c='white', s=5)
        else:
            ax.scatter([city.get('x', 0) + offset_x for city in sorted_path],
                       [city.get('y', 0) + offset_y for city in sorted_path],
                       c='white', s=5)

    # Animation loop
    def animate(frame, sorted_path, ax):
        ax.clear()
        draw_cities(ax, sorted_path[:frame + 1])

    # Example USA States data
    usa_states = [
        {'x': -97.0, 'y': 35.0, 'name': 'Oklahoma', 'size_x': 69.0, 'size_y': 47.0},
        {'x': -98.0, 'y': 31.0,  'name': 'Texas', 'size_x': 268.0, 'size_y': 278.0},
        {'x': -88.0, 'y': 41.0,  'name': 'Illinois', 'size_x': 57.0, 'size_y': 55.0},
        {'x': -81.0, 'y': 27.0,  'name': 'Florida', 'size_x': 53.0, 'size_y': 58.0},
        {'x': -119.0, 'y': 36.0,  'name': 'California', 'size_x': 163.0, 'size_y': 156.0},
        {'x': -86.0, 'y': 40.0,  'name': 'Indiana', 'size_x': 36.0, 'size_y': 35.0},
        {'x': -93.0, 'y': 45.0,  'name': 'Minnesota', 'size_x': 87.0, 'size_y': 84.0},
        {'x': -105.0, 'y': 39.0,  'name': 'Colorado', 'size_x': 104.0, 'size_y': 103.0},
        {'x': -81.0, 'y': 38.0,  'name': 'West Virginia', 'size_x': 24.0, 'size_y': 31.0},
        {'x': -77.0, 'y': 39.0,  'name': 'Maryland', 'size_x': 12.0, 'size_y': 32.0},
        {'x': -92.0, 'y': 34.0,  'name': 'Arkansas', 'size_x': 53.0, 'size_y': 55.0},
        {'x': -81.0, 'y': 34.0,  'name': 'South Carolina', 'size_x': 32.0, 'size_y': 33.0},
        {'x': -83.0, 'y': 32.0,  'name': 'Georgia', 'size_x': 59.0, 'size_y': 60.0},
        {'x': -80.0, 'y': 35.0,  'name': 'North Carolina', 'size_x': 54.0, 'size_y': 55.0},
        {'x': -74.0, 'y': 40.0,  'name': 'New Jersey', 'size_x': 9.0, 'size_y': 21.0},
        {'x': -71.0, 'y': 42.0,  'name': 'Massachusetts', 'size_x': 10.0, 'size_y': 27.0},
        {'x': -122.0, 'y': 47.0,  'name': 'Washington', 'size_x': 71.0, 'size_y': 70.0},
        {'x': -70.0, 'y': 45.0,  'name': 'Maine', 'size_x': 35.0, 'size_y': 34.0},
        {'x': -149.0, 'y': 64.0,  'name': 'Alaska', 'size_x': 665.0, 'size_y': 663.0},
        {'x': -157.0, 'y': 21.0,  'name': 'Hawaii', 'size_x': 11.0, 'size_y': 10.0},
        {'x': -86.0, 'y': 39.0,  'name': 'Indiana', 'size_x': 36.0, 'size_y': 35.0},
```

```
    {'x': -108.0, 'y': 33.0,  'name': 'New Mexico', 'size_x': 121.0, 'size_y': 122.0},
    {'x': -83.0, 'y': 42.0,  'name': 'Ohio', 'size_x': 45.0, 'size_y': 44.0},
    {'x': -96.0, 'y': 44.0,  'name': 'South Dakota', 'size_x': 77.0, 'size_y': 75.0},
    {'x': -85.0, 'y': 35.0,  'name': 'Tennessee', 'size_x': 42.0, 'size_y': 41.0},
    {'x': -120.0, 'y': 44.0,  'name': 'Oregon', 'size_x': 98.0, 'size_y': 96.0},
    {'x': -84.0, 'y': 43.0,  'name': 'Michigan', 'size_x': 97.0, 'size_y': 96.0},
    {'x': -73.0, 'y': 43.0,  'name': 'New York', 'size_x': 55.0, 'size_y': 54.0},
    {'x': -75.0, 'y': 41.0,  'name': 'Pennsylvania', 'size_x': 46.0, 'size_y': 47.0},
    {'x': -77.0, 'y': 38.0,  'name': 'Virginia', 'size_x': 43.0, 'size_y': 44.0},
    {'x': -86.0, 'y': 33.0,  'name': 'Alabama', 'size_x': 52.0, 'size_y': 53.0},
    {'x': -88.0, 'y': 33.0,  'name': 'Mississippi', 'size_x': 48.0, 'size_y': 49.0}
]


# Main

sorted_path = sort_cities(usa_states)

if not sorted_path:
    print("No paths were given to calculate.")
    print("Example command to issue the request:")
    print("python script_name.py --num_cities 50")
else:
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d' if 'z' in usa_states[0] else 'rectilinear')

    # Initial draw to set up plot
    draw_cities(ax, sorted_path)

    ani = FuncAnimation(fig, animate, frames=len(sorted_path), fargs=(sorted_path, ax), interv

    # Zoom functionality
    zoom_level = 1
    offset_x = 0
    offset_y = 0

    def on_scroll(event):
        global zoom_level, offset_x, offset_y
        zoom_factor = 1.1 if event.button == 'up' else 0.9
        zoom_level *= zoom_factor
        offset_x = event.xdata - (event.xdata - offset_x) * zoom_factor
        offset_y = event.ydata - (event.ydata - offset_y) * zoom_factor
        draw_cities(ax, sorted_path, zoom_level, offset_x, offset_y)
        plt.draw()

    fig.canvas.mpl_connect('scroll_event', on_scroll)
```

```
def on_click(event):
    if animation_frame[0] < len(sorted_path) - 1:
        animation_frame[0] += 1
        animate(animation_frame[0], sorted_path, ax)
        plt.draw()

fig.canvas.mpl_connect('button_press_event', on_click)

plt.show()
```

1 3 Share Approved 1 month ago

Post Insights

Only the post author and moderators can see this

547	100%	3	0
Total Views	Upvote Rate	Comments	Total Shares

Hourly views for first 48 hours

Add a comment

Sort by: Best Search Comments

TheStocksGuy OP • 1mo ago

https://github.com/BadNintendo/TravelingStatesProblem

1 Approved 1 month ago

TheStocksGuy OP • 1mo ago




Create



TheStocksGuy OP • 1mo ago • Edited 1mo ago

this github will remain the better version but this doesn't cross paths only to return home

Comment Image


1



Approved 1 month ago