Function Documentation:`is_contour_partially_inside`

1 Description

The `is_contour_partially_inside` function checks whether any part of `contour1` is inside `contour2`. It does this by checking each point of `contour1` as well as points along the segments connecting consecutive points of `contour1`. It uses OpenCV's `cv2.pointPolygonTest` for the containment checks and considers a width around the contour segments for partial inclusion.

2 Function Definition

```python
def is_contour_partially_inside(contour1, contour2, side_width):
    # Check if all points of contour1 are inside contour2
    last_point = (int(contour1[-1][0][0]), int(contour1[-1][0][1]))
    for i in range(len(contour1)):
        # Using cv2.pointPolygonTest to check each point of contour1 against contour2
        point = (int(contour1[i][0][0]), int(contour1[i][0][1]))
        if cv2.pointPolygonTest(contour2, point, False) > 0:
            return True
        points = generate_points(last_point, point, int(side_width/2)+1)
        for new_point in points:
            if cv2.pointPolygonTest(contour2, new_point, False) > 0:
                return True
        last_point = point
    return False
```

3 Function Explanation

3.1 Step-by-Step Breakdown

Function 1: Initialize Last Point

Initialize `last_point` with the coordinates of the last point of `contour1`.

```python
last_point = (int(contour1[-1][0][0]), int(contour1[-1][0][1]))
```

Explanation: The function starts by setting `last_point` to the last point of `contour1`. This point will be used to generate intermediate points along the contour segments.

Function 2: Check Point Inside

Check if each point of `contour1` is inside `contour2`.

```
for i in range(len(contour1)):
    point = (int(contour1[i][0][0]), int(contour1[i][0][1]))
    if cv2.pointPolygonTest(contour2, point, False) > 0:
        return True
```

Explanation: The function iterates over each point in `contour1` and uses `cv2.pointPolygonTest` to determine if the point is inside `contour2`. If any point is found inside, the function returns `True`.

Function 3: Generate Intermediate Points

Generate intermediate points along the line segment between `last_point` and `point`.

```
points = generate_points(last_point, point, int(side_width/2)+1)
```

Explanation: Intermediate points are generated along the line segment connecting `last_point` and `point` to account for partial inclusion. The number of intermediate points is determined by `side_width`.

Function 4: Check Intermediate Points

Check if any of the generated intermediate points are inside `contour2`.

```
for new_point in points:
    if cv2.pointPolygonTest(contour2, new_point, False) > 0:
        return True
```

Explanation: The function checks each of the generated intermediate points to see if they lie inside `contour2`. If any intermediate point is found inside, the function returns `True`.

Function 5: Update Last Point

Update `last_point` to the current `point` and continue checking the next segment.

```
last_point = point
```

Explanation: The `last_point` is updated to the current `point` after processing, so that the next segment can be checked.

4 Conclusion

The `is_contour_partially_inside` function checks if any part of `contour1` is inside `contour2`. It considers both direct point inclusion and partial inclusion along the segments connecting points of `contour1`, making it suitable for more comprehensive geometric containment checks.