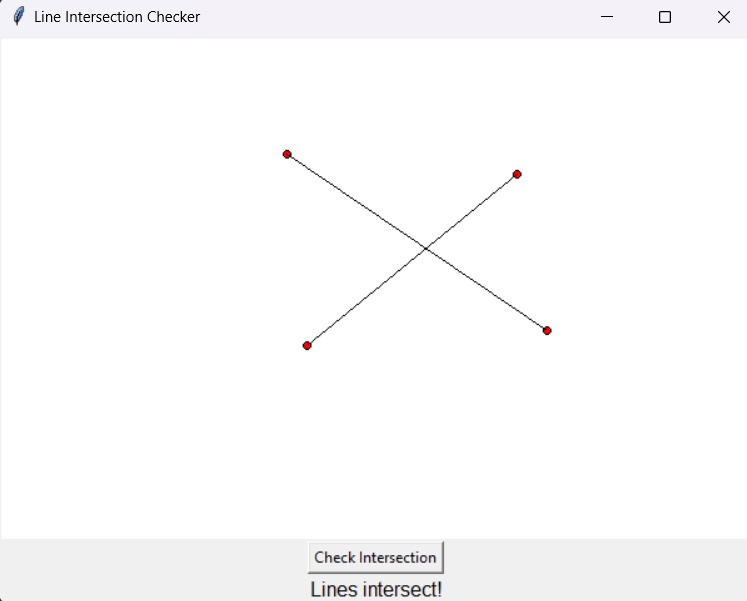
# LINE INTERSECTION

## ORIENTATION:

import tkinter as tk  
  
class Point:  
 def \_\_init\_\_(self, x, y):  
 self.x = x  
 self.y = y  
  
def orientation(p, q, r):  
 val = (q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y)  
 if val == 0:  
 return 0 # collinear  
 return 1 if val > 0 else 2 # 1 for clockwise, 2 for counterclockwise  
  
def on\_segment(p, q, r):  
 return (  
 (q.x <= max(p.x, r.x))  
 and (q.x >= min(p.x, r.x))  
 and (q.y <= max(p.y, r.y))  
 and (q.y >= min(p.y, r.y))  
 )  
  
def do\_intersect(p1, q1, p2, q2):  
 o1 = orientation(p1, q1, p2)  
 o2 = orientation(p1, q1, q2)  
 o3 = orientation(p2, q2, p1)  
 o4 = orientation(p2, q2, q1)  
  
 if (o1 != o2) and (o3 != o4):  
 return True  
  
 if (o1 == 0) and on\_segment(p1, p2, q1):  
 return True  
  
 if (o2 == 0) and on\_segment(p1, q2, q1):  
 return True  
  
 if (o3 == 0) and on\_segment(p2, p1, q2):  
 return True  
  
 if (o4 == 0) and on\_segment(p2, q1, q2):  
 return True  
  
 return False  
  
class LineIntersectionApp:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Line Intersection Checker")  
  
 self.canvas = tk.Canvas(root, width=600, height=400, bg="white")  
 self.canvas.pack()  
  
 self.points = []  
 self.canvas.bind("<Button-1>", self.on\_canvas\_click)  
  
 tk.Button(root, text="Check Intersection", command=self.check\_intersection).pack()  
 self.result\_label = tk.Label(root, text="", font=("Helvetica", 12))  
 self.result\_label.pack()  
  
 def on\_canvas\_click(self, event):  
 x, y = event.x, event.y  
 self.points.append(Point(x, y))  
 self.canvas.create\_oval(x - 3, y - 3, x + 3, y + 3, fill="red")  
  
 if len(self.points) == 4:  
 self.canvas.create\_line(self.points[0].x, self.points[0].y, self.points[1].x, self.points[1].y, fill="black")  
 self.canvas.create\_line(self.points[2].x, self.points[2].y, self.points[3].x, self.points[3].y, fill="black")  
  
 def check\_intersection(self):  
 if len(self.points) == 4:  
 intersect = do\_intersect(self.points[0], self.points[1], self.points[2], self.points[3])  
  
 if intersect:  
 self.result\_label.config(text="Lines intersect!")  
 else:  
 self.result\_label.config(text="Lines do not intersect")  
  
 self.points = [] # Reset points for the next round  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = LineIntersectionApp(root)  
 root.mainloop()

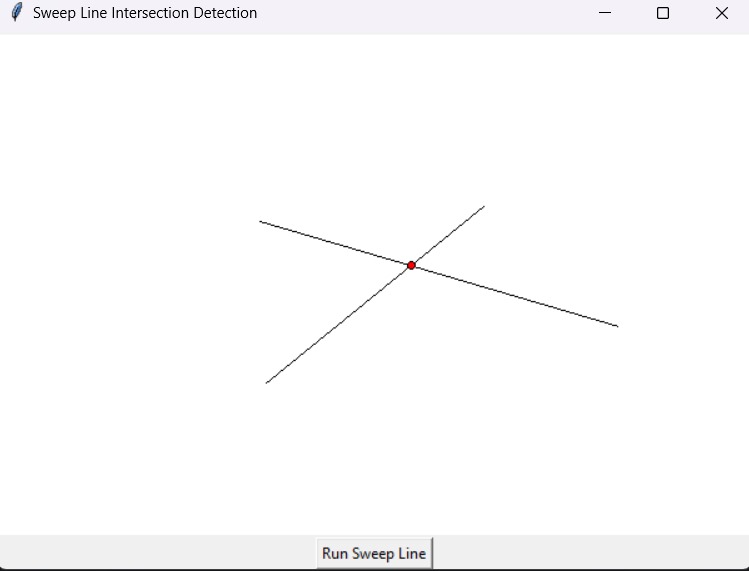


## SWEEPLINE ALGORITHM

"Computational Geometry: Algorithms and Applications" by Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars.

This approach is different from the orientation and parametric representation methods. It involves maintaining a dynamic status structure and efficiently handling events along the sweep line. The complexity of the algorithm is often O((n + k) log n), where n is the number of line segments, and k is the number of intersections.

import tkinter as tk  
from shapely.geometry import LineString  
  
class Point:  
 def \_\_init\_\_(self, x, y):  
 self.x = x  
 self.y = y  
  
class SweepLineApp:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Sweep Line Intersection Detection")  
  
 self.canvas = tk.Canvas(root, width=600, height=400, bg="white")  
 self.canvas.pack()  
  
 self.lines = []  
 self.current\_line = []  
  
 self.canvas.bind("<Button-1>", self.on\_canvas\_click)  
 tk.Button(root, text="Run Sweep Line", command=self.run\_sweep\_line).pack()  
  
 def on\_canvas\_click(self, event):  
 x, y = event.x, event.y  
 point = Point(x, y)  
 self.current\_line.append(point)  
  
 if len(self.current\_line) == 2:  
 self.lines.append(list(self.current\_line))  
 self.canvas.create\_line(self.current\_line[0].x, self.current\_line[0].y,  
 self.current\_line[1].x, self.current\_line[1].y, fill="black")  
 self.current\_line = []  
  
 def run\_sweep\_line(self):  
 intersections = self.sweep\_line\_intersection()  
  
 for intersection in intersections:  
 self.canvas.create\_oval(intersection.x - 3, intersection.y - 3, intersection.x + 3, intersection.y + 3, fill="red")  
  
 def sweep\_line\_intersection(self):  
 intersections = []  
  
 for i in range(len(self.lines)):  
 for j in range(i + 1, len(self.lines)):  
 line1 = LineString([(self.lines[i][0].x, self.lines[i][0].y),  
 (self.lines[i][1].x, self.lines[i][1].y)])  
 line2 = LineString([(self.lines[j][0].x, self.lines[j][0].y),  
 (self.lines[j][1].x, self.lines[j][1].y)])  
  
 if line1.intersects(line2):  
 intersection = line1.intersection(line2)  
 if intersection.is\_empty or intersection.geom\_type == 'Point':  
 intersections.append(Point(\*list(intersection.coords)[0]))  
  
 return intersections  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = SweepLineApp(root)  
 root.mainloop()



# Slope Algorithm

import tkinter as tk  
  
class Point:  
 def \_\_init\_\_(self, x, y):  
 self.x = x  
 self.y = y  
  
def slope(p1, p2):  
 if p1.x == p2.x:  
 return float('inf') # Vertical line  
 return (p2.y - p1.y) / (p2.x - p1.x)  
  
def on\_segment(p, q, r):  
 return (q.x <= max(p.x, r.x) and q.x >= min(p.x, r.x) and  
 q.y <= max(p.y, r.y) and q.y >= min(p.y, r.y))  
  
def do\_intersect(p1, q1, p2, q2):  
 slope1 = slope(p1, q1)  
 slope2 = slope(p2, q2)  
  
 if slope1 == slope2: # Parallel lines  
 return False  
  
 intercept1 = p1.y - slope1 \* p1.x  
 intercept2 = p2.y - slope2 \* p2.x  
  
 intersection\_x = (intercept2 - intercept1) / (slope1 - slope2)  
 intersection\_y = slope1 \* intersection\_x + intercept1  
  
 intersection\_point = Point(intersection\_x, intersection\_y)  
  
 return on\_segment(p1, intersection\_point, q1) and on\_segment(p2, intersection\_point, q2)  
  
class LineIntersectionApp:  
 def \_\_init\_\_(self, master):  
 self.master = master  
 self.master.title("Line Intersection Checker")  
  
 self.canvas = tk.Canvas(self.master, width=400, height=400, bg="white")  
 self.canvas.pack()  
  
 self.points = []  
 self.lines = []  
  
 self.canvas.bind("<Button-1>", self.on\_canvas\_click)  
 self.check\_intersection\_button = tk.Button(self.master, text="Check Intersection", command=self.check\_and\_visualize\_intersection)  
 self.check\_intersection\_button.pack()  
  
 def on\_canvas\_click(self, event):  
 x, y = event.x, event.y  
 self.canvas.create\_oval(x - 3, y - 3, x + 3, y + 3, fill="black")  
 new\_point = Point(x, y)  
 self.points.append(new\_point)  
  
 if len(self.points) == 2:  
 self.lines.append(self.canvas.create\_line(self.points[0].x, self.points[0].y, self.points[1].x, self.points[1].y))  
  
 # Reset points for the next line segment  
 self.points = []  
  
 def check\_and\_visualize\_intersection(self):  
 if len(self.lines) == 2:  
 line1\_coords = self.canvas.coords(self.lines[0])  
 line2\_coords = self.canvas.coords(self.lines[1])  
  
 p1 = Point(line1\_coords[0], line1\_coords[1])  
 q1 = Point(line1\_coords[2], line1\_coords[3])  
 p2 = Point(line2\_coords[0], line2\_coords[1])  
 q2 = Point(line2\_coords[2], line2\_coords[3])  
  
 if do\_intersect(p1, q1, p2, q2):  
 print("The line segments intersect.")  
 else:  
 print("The line segments do not intersect.")  
  
def main():  
 root = tk.Tk()  
 app = LineIntersectionApp(root)  
 root.mainloop()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

