```python
from manim import *
import numpy as np
import random

class VoronoiAnimation(Scene):
    def norm(self, vector):
        return np.sqrt(np.sum(np.square(vector)))

    def custom_argsort(self, arr):
        return sorted(range(len(arr)), key=lambda x: arr[x])

    def generate_points(self, n, r_min, r_max, theta_min, theta_max):
        points = []
        for _ in range(n):
            r = random.uniform(r_min, r_max)
            theta = random.uniform(theta_min, theta_max)
            x = r * np.cos(np.radians(theta))
            y = r * np.sin(np.radians(theta))
            points.append(np.array([x, y, 0]))  # Ensure points are 3D
        return points

    def find_closest_point(self, points, P_0, closest_points):
        if len(points) == 0:
            raise ValueError("No points to find the closest point from.")

        distances = np.array([self.norm(point - P_0) for point in points])
        for point in closest_points:
            indices = np.where((points == point).all(axis=1))
            if len(indices[0]) > 0:
                index = indices[0][0]
                distances[index] = float('inf')

        closest_point_index = np.argmin(distances)
        return closest_point_index, points[closest_point_index]

    def find_unit_vector(self, P_from, P_to):
        vector = P_to - P_from
        vector_norm = self.norm(vector)
        if vector_norm == 0:
            return np.array([0, 0, 0])
        return vector / vector_norm

    def filter_points_by_dot_product(self, points, base_point, reference_vector):
        remaining_points = []
        for point in points:
            if np.array_equal(point, base_point):
                continue
            unit_vector = self.find_unit_vector(base_point, point)
            dot_product = np.dot(reference_vector, unit_vector)
            if dot_product >= 0:
                remaining_points.append(point)
        return np.array(remaining_points)

    def find_intersection(self, midpoint1, normal1, midpoint2, normal2):
        A = np.array([normal1, -normal2]).T[:2, :2]
        b = midpoint2[:2] - midpoint1[:2]
        if np.linalg.det(A) == 0:
            return None
        intersection = np.linalg.solve(A, b)
        return midpoint1 + intersection[0] * normal1

    def construct(self):
        P_0 = np.array([0, 0, 0])
        points_first = (
            self.generate_points(n=5, r_min=2.5, r_max=15, theta_min=5, theta_max=85)
```

```
+
            self.generate_points(n=5, r_min=2.5, r_max=15, theta_min=95, theta_max=175
) +
            self.generate_points(n=5, r_min=2.5, r_max=15, theta_min=185, theta_max=26
5) +
            self.generate_points(n=5, r_min=2.5, r_max=15, theta_min=275, theta_max=35
5)
        )
        points = np.array(points_first)
        closest_points = []

        all_points = np.array(points_first + [P_0])
        min_x, min_y, _ = np.min(all_points, axis=0)
        max_x, max_y, _ = np.max(all_points, axis=0)

        scene_width = max_x - min_x
        scene_height = max_y - min_y
        max_dim = max(scene_width, scene_height)

        scale_factor = 6 / max_dim
        all_points = (all_points - np.array([(min_x + max_x) / 2, (min_y + max_y) / 2,
 0])) * scale_factor

        points = all_points[:-1]
        P_0 = all_points[-1]

        new_center = np.array([(min_x + max_x) / 2, (min_y + max_y) / 2, 0]) * scale_f
actor

        axes = Axes().shift(np.append(-new_center[:2], 0))
        origin_dot = Dot(P_0, color=RED)
        self.play(Create(axes), Create(origin_dot))

        initial_dots = [Dot(point, color=GRAY) for point in points]
        self.play(*[Create(dot) for dot in initial_dots])
        self.wait(2)

        while len(points) > 0:
            closest_point_index, closest_point = self.find_closest_point(points, P_0,
closest_points)
            closest_points.append(closest_point)

            reference_vector = self.find_unit_vector(P_from=closest_point, P_to=P_0)
            points = self.filter_points_by_dot_product(points, closest_point, referenc
e_vector)

            self.play(Create(Dot(closest_point, color=GREEN)))
            self.play(Create(Line(P_0, closest_point, color=GRAY, stroke_width=2, stro
ke_opacity=0.5)))

            perp_vector = np.array([-reference_vector[1], reference_vector[0], 0])
            start_point = closest_point - perp_vector * 10
            end_point = closest_point + perp_vector * 10
            perp_line = Line(start_point, end_point, color=YELLOW, stroke_width=2)
            self.play(Create(perp_line))

            remaining_dots = [Dot(point, color=GRAY, fill_opacity=0.3 if point in poin
ts else 0.1) for point in all_points[:-1]]
            animations = [dot.animate.set_fill(opacity=0.1) for dot in remaining_dots
if dot.get_center() not in points]
            if animations:
                self.play(*animations)

            self.wait(1)
```

```
            if points.size == 0:
                break

        midpoints = [(P + P_0) / 2 for P in closest_points]
        normals = []
        for P in closest_points:
            vector = P - P_0
            normal = np.array([-vector[1], vector[0], 0])
            unit_normal = normal / self.norm(normal)
            normals.append(unit_normal)

        midpoints = np.array(midpoints)
        angles = np.arctan2(midpoints[:, 1], midpoints[:, 0])
        sorted_indices = self.custom_argsort(angles)
        midpoints = midpoints[sorted_indices]
        normals = np.array(normals)[sorted_indices]

        for i in range(len(midpoints)):
            start_point = midpoints[i] - normals[i] * 10
            end_point = midpoints[i] + normals[i] * 10
            perp_line = Line(start_point, end_point, color=YELLOW, stroke_width=0.5)
            self.play(Create(perp_line))
            self.wait(0.5)

        intersection_points = []
        for i in range(len(midpoints)):
            next_index = (i + 1) % len(midpoints)
            intersection = self.find_intersection(midpoints[i], normals[i], midpoints[
next_index], normals[next_index])
            if intersection is not None:
                intersection_points.append(intersection)

        self.play(*[Create(Dot(midpoint, color=BLUE)) for midpoint in midpoints])
        self.wait(2)

        intersection_points = np.array(intersection_points)
        if intersection_points.size > 0:
            self.play(*[Create(Dot(intersection, color=PURPLE)) for intersection in in
tersection_points])

            for i in range(len(intersection_points)):
                next_index = (i + 1) % len(intersection_points)
                self.play(Create(Line(intersection_points[i], intersection_points[next
_index], color=PURPLE)))

        self.wait(2)

        if intersection_points.size > 0:
            polygon = Polygon(*intersection_points, color=YELLOW, fill_opacity=0.5)
            self.play(Create(polygon))

        self.wait(2)
```