AP157 Output 3 - February 13, 2026

Benedict A. Pangilinan (bapangilinan5@up.edu.ph)

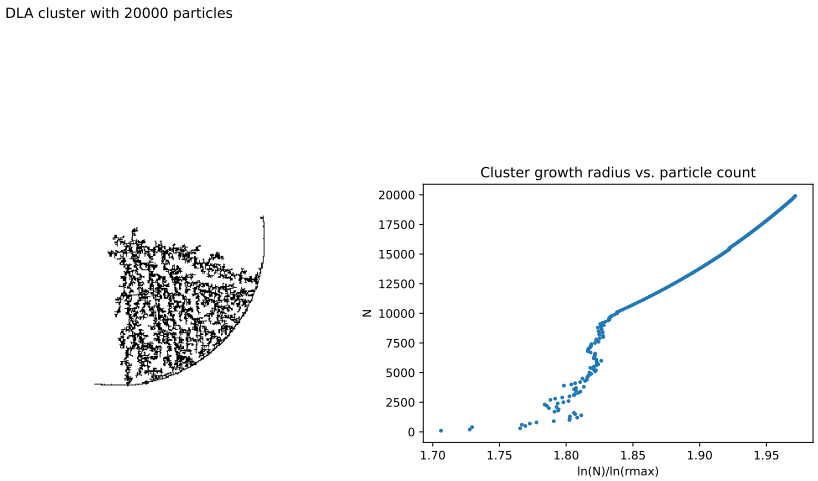

Figure 1: (Left) DLA structure in which the angle is apparently biased, angle randomization likely faulty. (Right) logarithmic normalization of stick count

Simulation Code

This is the code used for Simulation:

```
# Simulation
stick_count = 1
target_particles = 20000
radii_history, counts_history = [], []

while stick_count < target_particles:
    pos_x, pos_y, _, _ = generate_initial_position
        (lattice_dim, rmax_real, rdiff_real,
        rkill_real)
    stuck = False
    while not stuck:
        neighbors = [
            (pos_x-1, pos_y), (pos_x+1, pos_y),
            (pos_x, pos_y-1), (pos_x, pos_y+1)
        ]
        if any(agg_area[nx, ny] == 1 for nx, ny in
            neighbors):
            agg_area[pos_x, pos_y] = 1
            rmax_real = max(rmax_real, R_for_shape[
                pos_x, pos_y])
            if stick_count % 100 == 0:
                radii_history.append(rmax_real)
                counts_history.append(stick_count)
            # keep diffusion modest, kill large
            rdiff_real = 15
            rkill_real = 10*rdiff_real
            stick_count += 1
            stuck = True
        else:
            pos_x, pos_y = next_hop(pos_x, pos_y,
                lattice_dim, rmax_real, rdiff_real,
                 rkill_real)
```

Functions

```
def generate_initial_position(max_index, rmax,
     rdiff, rkill):
angle = rng.uniform(0, 2*np.pi)
pos_x = int(max_index + rkill*np.cos(angle))
pos_y = int(max_index + rkill*np.sin(angle))
return pos_x, pos_y, R_for_shape[pos_x, pos_y
    ], "init"

def next_hop(curr_x, curr_y, max_index, rmax,
    rd, rkill):
dist = R_for_shape[curr_x, curr_y]
if dist < rd:
    # single-step moves
    sequence = [
        (curr_x-1, curr_y), (curr_x+1, curr_y),
        (curr_x, curr_y-1), (curr_x, curr_y+1)
    ]
    pos_x, pos_y = rng.choice(sequence)
    return int(pos_x), int(pos_y)
elif dist < rkill:
    # modest Brownian hops
    k = rng.integers(low=2, high=4)
    th_rand = rng.uniform(0, 2*np.pi)
    pos_x = int(curr_x + k*np.cos(th_rand))
    pos_y = int(curr_y + k*np.sin(th_rand))
    return pos_x, pos_y
else:
    # regenerate walker
    pos_x, pos_y, _, _ =
        generate_initial_position(max_index,
        rmax, rd, rkill)
    return pos_x, pos_y
```