An approach to simulate the aggregation of volcanic ash

BIGOT Romain, BUGNOT Corentin
University of Geneva

Keywords – particles, aggregate, minimization, volcano, algorithm, spheres

The algorithm

In the following sections, we will explain in details how the algorithm works for **one sphere**, but it will repeat these steps **for each sphere provided**.

Idea The idea is to generate a sphere far enough from the core and make it travel towards the center. Once it collides with another sphere on the aggregate, we look for a new position, step by step in the *exploration radius*, which is closer to the core.

1. Generate the sphere

At first, we want to generate a sphere from a given radius at a specific distance from the aggregate. To **optimize** the travel to the center, the sphere will spawn at :

radSpawn = distance between the farthest sphere from the aggregate's core + radius of the generated sphere + dt

where dt is the precision of the simulation.
This is the closest point from which we can **safely** generate the sphere and save computational time.

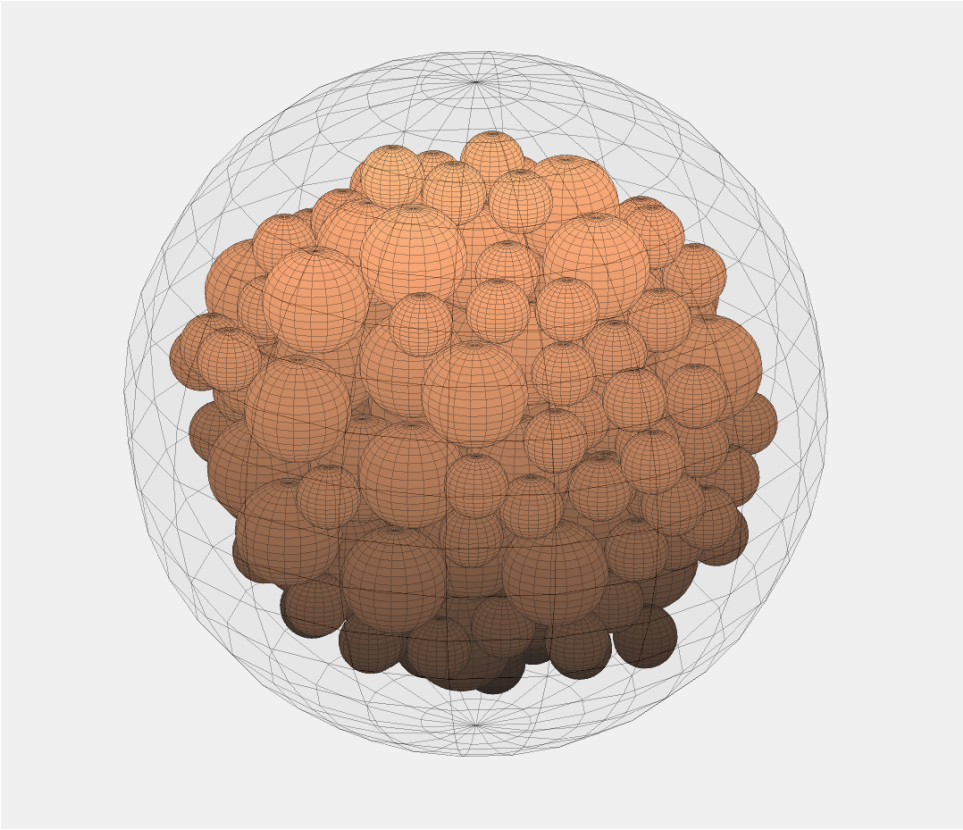

Figure 1: Area of sphere generation

2. Moving the sphere

To make the sphere travel to the core, we simply multiply its position by the vector pointing towards the core, the sphere also travels at a speed of 1. If $\vec{x_0}$ is the position of the sphere, the new position $\vec{x_1}$ becomes :

$$\vec{x_1} = \vec{x_0} - sign(\vec{x_0}) \cdot dt \tag{1}$$

3. Collision detection

Two spheres of positions $\vec{x}$, $\vec{y}$ and radius r_x, r_y overlap eachother if and only if :

$$\|\vec{x} - \vec{y}\|_2 \leq r_x + r_y \tag{2}$$

Instead of checking the distance between all spheres in the aggregate and the current moving sphere, which is too CPU consuming, especially with a large number of spheres, we implemented an **octree data structure** to partition the space into smaller cubes.
With this approach we can easily query all closest neighbours of a given sphere and check collisions with them only, reducing the complexity of collision detection from $O(n)$ to $O(log(n))$ where n is the total number of spheres in the aggregate.

4. Minimize the distance with the core

Once the sphere has collided (*Figure 2*), the surface tension makes it travel towards the core, which acts as the center of gravity. The sphere is constrained to only move around a given *exploration radius* R (*Figure 3*).

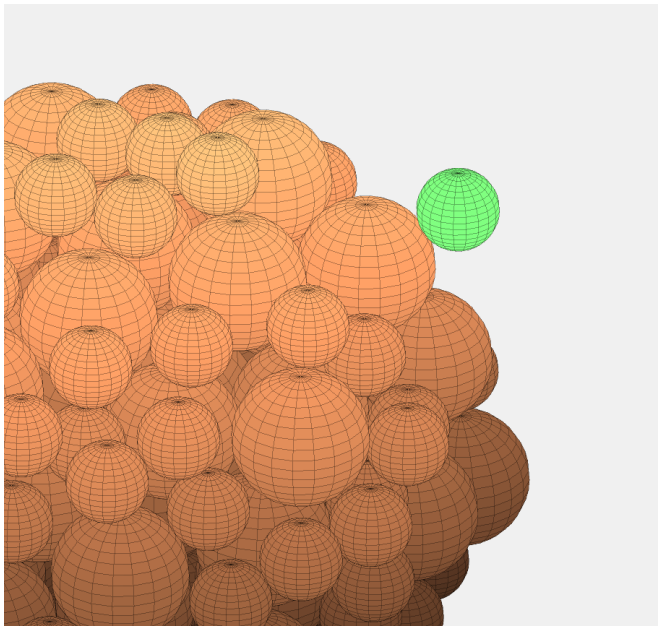

Figure 2: The sphere has collided

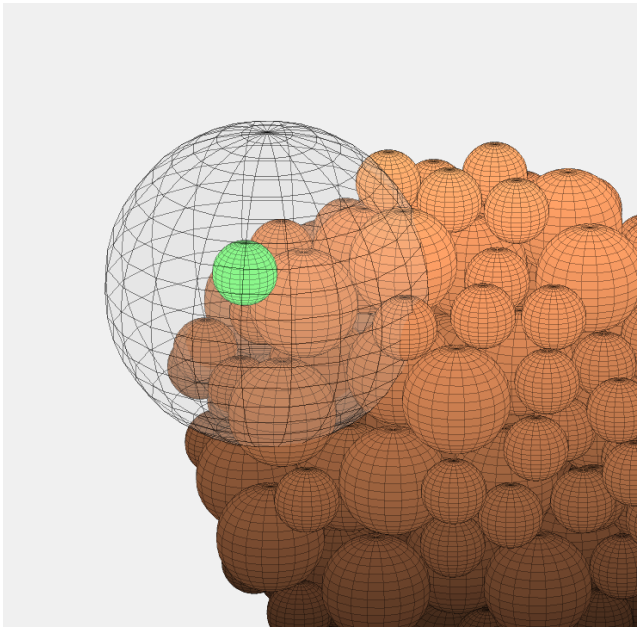

Figure 3: The exploration radius

To simulate this phenomenon, we divide the *exploration radius* into smaller radius $\Delta r < R$ (*Figure 5*).
This sphere of radius Δr is then divided into multiple "layers" of randomly generated points $\vec{x}$ (*Equation* 3).

$$\vec{x} = \frac{\Delta r}{\left\| \vec{X} \right\|_2} \cdot \vec{X} \tag{3}$$

$$\vec{X} \in \mathbb{R}^3,\ \vec{X} \sim Uniform(0,1)$$

The number of points vary according to Δr^2 to keep the same density at each level (*Figure 4*).

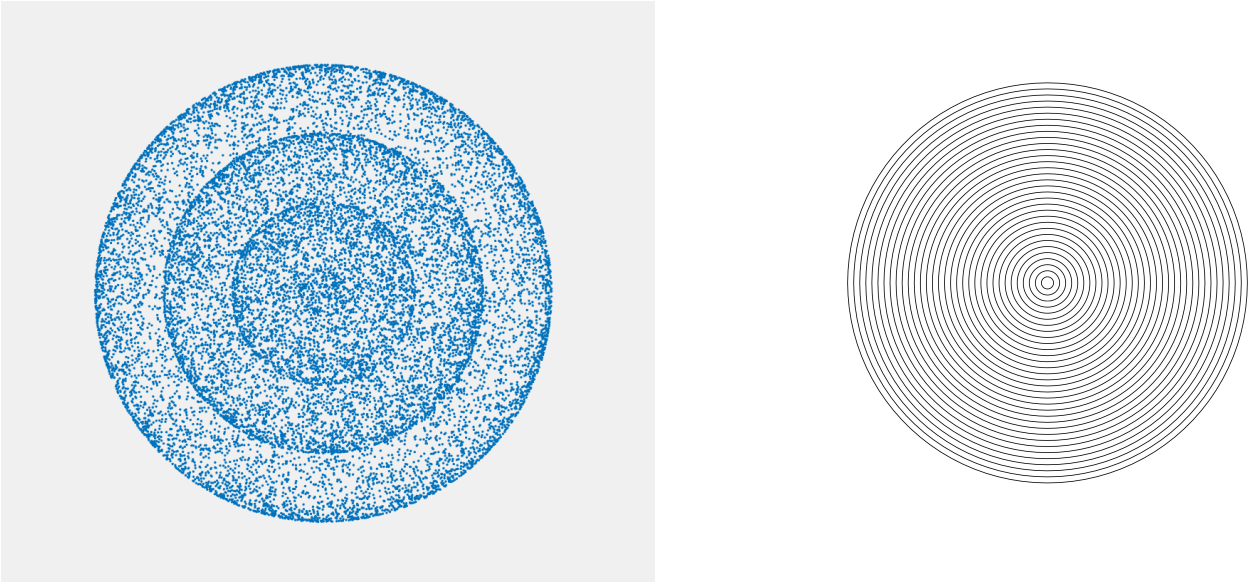

Figure 4: A cut section of the different levels of generated points

From those generated points, we keep only those with a **lower** distance from the core than the actual sphere, and which **do not overlap** in a radius of the current sphere with another sphere from the aggregate (*Figure 5*).

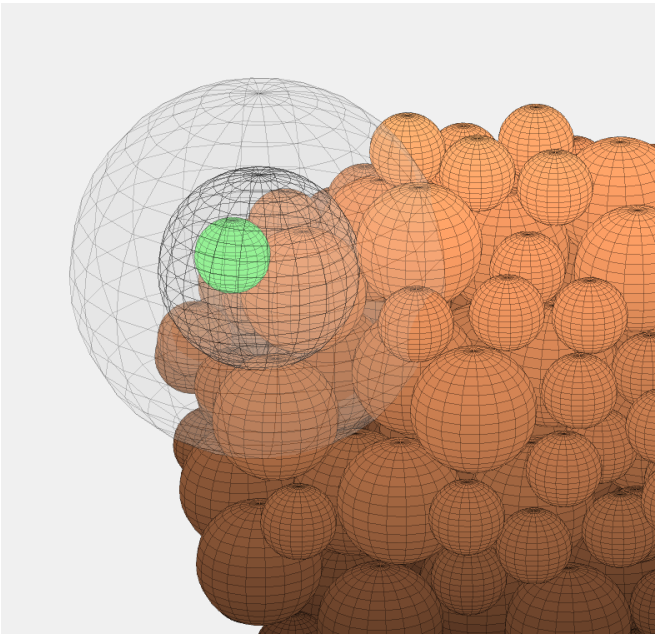

Figure 5: Smaller radius Δr

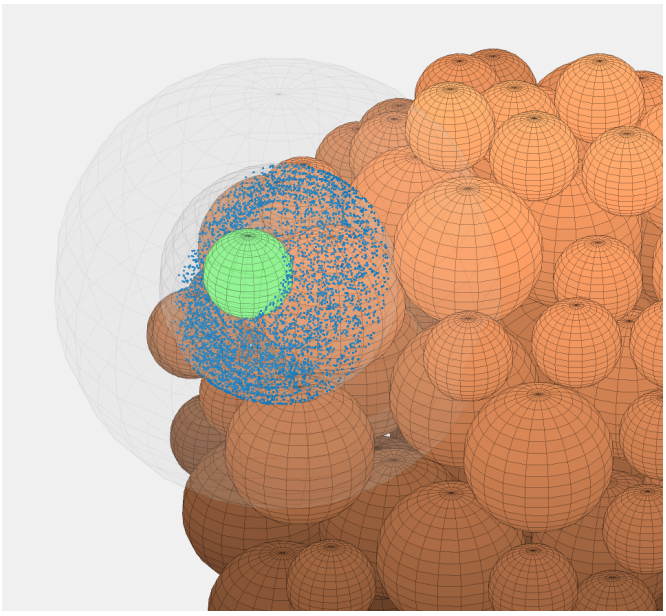

Figure 6: The potential data

The closest generated point (to the core) will then be the **new position** of the current sphere (*Figure 6*). We repeat this process until the total distance covered by the current sphere does not exceed R (*Figure 7*), or until the number of steps has not been reached. To limit the resource usage the number of steps can be adjusted.

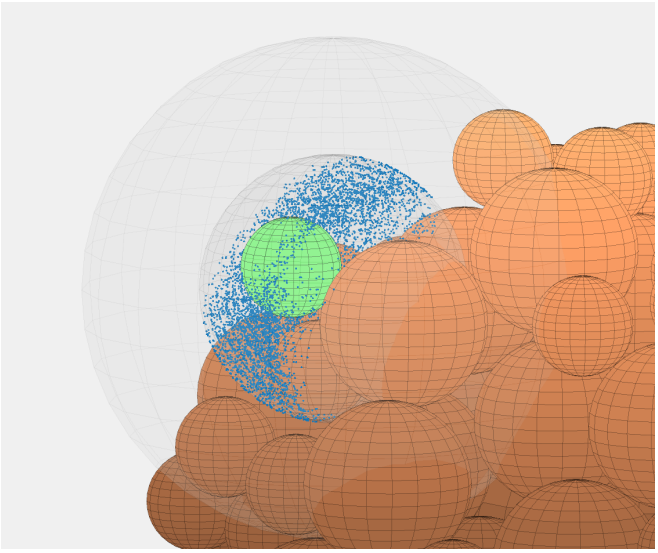

Figure 7: The new minimum position we found

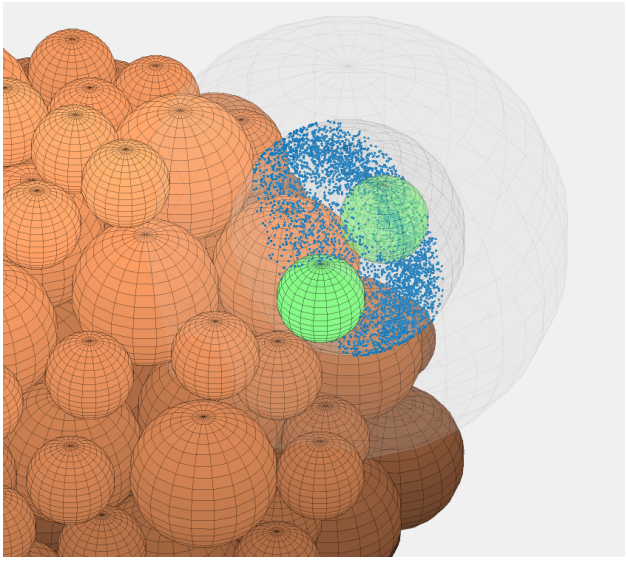

Figure 8: Repeat the process

Once this process is finished, the closest position to the core in the exploration radius R (*Figure 7*) has been found.

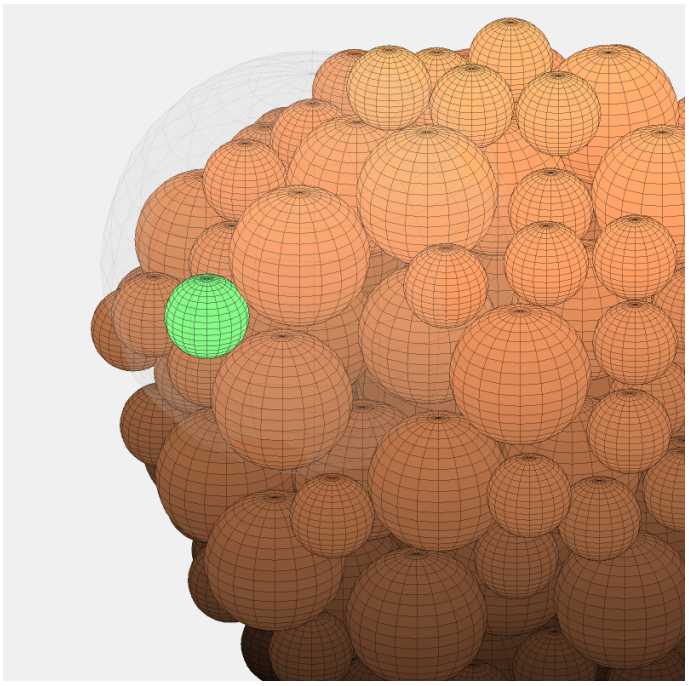

Figure 9: The final minimum

5. Conclusion

With this approach, we were able to compute an aggregate of around 5000 spheres in approximately 20 seconds, this result may vary between computers.
Moreover, we only used **spheres** to describe particles but to get closer to the reality, this can be extended to **ellipsoids**.
Finally, the complete algorithm is described below :

for *all spheres we want to generate* **do**
  spawn a sphere
  while *no collision* **do**
    move the sphere towards the center
  end
  $distTravelled = 0$
  $cpt = 0$
  while $distTravelled \leq$ *exploration radius* $\vee$ $cpt \geq nbStepMin$ **do**
    generate potential new positions
    $distTravelled = distTravelled +$ *distance from old to new position*
    $cpt = cpt + 1$
  end
  append this sphere to the aggregate
end

Algorithm 1: The whole process