A graupel generation algorithm

Torbjörn Rathsman

November 7, 2016

This article describes an algorithm that can be used to generate graupel-like particles.

Contents

1 Data representation **1**

2 Generating a collision event **1**

3 Accepting collision events, and merging **2**

4 Formal description of the algorithm **2**

5 Algorithmic complexity **3**

6 Improvements **3**

1 Data representation

A graupel can be modelled as a spherical object, build of other spheres. While it is possible to store spheres as polygon meshes, it requires quite a large number of vertices to get a good approximation. Also, a graupel requires a large number of sub-volumes, since the diameter only grows as $V^{1/3}$.

Instead of using polygon meshes, the spheres are represented as a midpoint and a radius. The spheres are stored in a dynamic array.

2 Generating a collision event

A collision event is generated by choosing a random point on a sphere surrounding the aggregate. From that point, a ray is cast towards the aggregate. The idea is to attach a new sphere at the surface of the sphere closest to the ray origin.

Not all collisions need to result in a merge event. It may depend on the energy of the incoming particle, as will as the incidence angle. It is assumed that an incoming particle with a kinetic energy less than 1 sticks to the aggregate, and that an incidence angle of zero makes the incoming particle looses all of its kinetic energy, thus sticking to the aggregate. The relative loss of kinetic energy is taken as the cosine between the reversed direction of the ray, and the normal of the sub-volumes at the incidence point, that is the dot product between the two vectors. Moreover, if this dot product is negative, the intersection point is not considered. Thus, the algorithm implements backface culling.

The true nature of the process, is that incoming particles, that do not stick to the aggregate, must bounce of, loosing some of their kinetic energy, and acquire some heat. Implementing this is possible, but slow, so instead the kinetic energy is just modelled as an exponential of the travelled distance: The longer path, the smaller kinetic energy. The initial kinetic energy is taken as a uniformly distributed random number between 0 and a parameter E_0.

3 Accepting collision events, and merging

After a collision event has been generated, the incoming particle is pulled by a given amount towards the aggregate. This is necessary due to the fact that true spheres only touch each other in a point, and there has to be some surface area at the merge point. In addition to that, an overlap between other sub-volumes may also be accepted. If there are too many overlaps, the event is rejected.

4 Formal description of the algorithm

Let S be the graupel. Formally, the algorithm can be described as follows.

1. Place a sphere S_0 with random radius at the origin O
2. Append S_0 to S
3. While stop condition is false
   a) Choose a point P on a sphere surrounding S
   b) Define r as the ray from P through O. Let $\vec{v} = \frac{O-P}{|O-P|}$ be the direction vector of that ray.
   c) Draw a uniformly distributed random number ϵ between 0 and E_0
   d) Find the intersection point P_0 between r, and the sub-volume S_1 in S closest to P. Also, call the normal of S_1 at P_0 $\vec{n}_1$.
   e) If overlap conditions are fulfilled
      i. Append a new S_0 to S so that S_1 $\vec{n}_1 = -\vec{n}_2$
      ii. Move S_0 a certain amount in the normal direction.

In step 3c, the following test is used to determine whether or not a sub-volume should count.

- If $\vec{n}_1 \cdot (-\vec{v}) < 0$, the ray has hit the backface of the sub-volume, and the search continues.
- If $\vec{n}_1 \cdot (-\vec{v})\, \epsilon \exp\left(-\frac{t}{\tau}\right) > 1$, the incoming particle is assumed to have passed, and the search continues. Here t is the distance between P and P_0, and τ is the decay distance.
- Otherwise, the the current sub-volume is a candidate

5 Algorithmic complexity

Since there is no certain geometrical order among the sub-volumes N in S, the ray-caster needs $\mathcal{O}(N)$ iterations to find the nearest surface point. Furthermore, if the stop condition is in terms of D_{max}—the extent of the aggregate, the complexity becomes $\mathcal{O}(N \cdot N^3) = \mathcal{O}(N^4)$. This is because the volume grows linearly in N, and therefore the algorithm is quadratic if the volume is given as a stop condition. At the same time $V \propto D_{\max}^3 \iff D_{\max} \propto V^{1/3}$. Thus, the algorithmic complexity is $\mathcal{O}(N^4)$.

6 Improvements

Through the use of spatial acceleration structures, the complexity of the ray-caster can be reduced from $\mathcal{O}(N)$ to $\mathcal{O}\log(N)$, due to the introduction of geometrical order. This will lower the total complexity to $\mathcal{O}(N^3 \log(N)$ for a given $D_{\max}$.