1 Sphere sampling algorithm

```
Data: Configuration Space
Result: Spheres that cover most of the free space
s0 = sampleSphere( randomly position );
S = {s0};
// Every two boundary points distance equals to δ
boundaryQueue = a queue that contains all boundary points of s0;
while boundaryQueue is not empty do
    point = boundaryQueue.pop();
    if point in an sphere then
        continue;
    else
        si = sampleSphere( point );
        if δ ≤ si.radius then
            S = S ∩ {si};
            boundaryQueue.put( all boundary points of si );
        end
    end
end
```

Algorithm 1: Sphere sampling algorithm

2 Path searhing alogrithm

```
Data: spheres, start configuration, goal configuration
Result: a path that connects start and end configuratioin.(or fail)
openSet = priority queue containing start;
closedSet = empty set;
while openSet is not empty do
    current = remove lowest rank point from openSet;
    add current to closedSet;
    neighbors = getSuccessors( current );
    foreach neighbor in neighbors do
        if neighbor == goal then
            set neighbors parent to current; backtrain( neighbor )
        else
            cost = g(current) + movementcost(current, neighbor);
            if neighbor in openSet and cost less than g(neighbor) then
                remove neighbor from openSet; // because new path is better
            end
            if neighbor in closedSet and cost less than g(neighbor) then
                remove neighbor from closedSet;
            end
            if neighbor not in openSet or closedSet then
                set g(neighbor) to cost;
                add neighbor to openSet;
                set priority queue rank to g(neighbor) + h(neighbor);
                set neighbors' parent to current;
            end
        end
    end
end
```

Algorithm 2: Search an Optimal Path

```
Data: point, goal
Result: successor points
sphere = get the sphere that contains point;
if sphere contains goal then
    return goal.
else
    return boundary points of sphere.
end
```

Algorithm 3: getSuccessors()

3 Finite number of spheres

As is shown in the algorithm, we get boundary points of an existing sphere by limiting the distance between every two points to be equal to δ. Then the algorithm samples new spheres centered at these points. So the distance between two spheres centers is always larger than or equal to δ.

The problem is equivalant to proof there could be finite number of points put in the space such that the distance between every two points is larger than or equal to δ.

Assume we partition the whole configuration space(including free space and obstacle space) into hyper-cubes. The length of every edge of these hyper-cubes is exactly δ. As long as the configuration space has finite volume, the number of hyper-cubes is finite. Assume the number of hyper-cubes is N, the number of hyper-cubes that is totall or partly in free space is less than N. Thus the number of spheres is finite, and less than N.

So the algorithm will always converge.

4 Path quality

Assuming the path maintains δ clearance, the worst path found by our algorithm is this: the optimal path is in a chain of spheres with radius δ. The point our algorithm chooses to use is δ / 2 away from the actual point the path is going through.

Then the chosen path is 2 times of length of the actual optimal path.

????????? SO BAD ???????????

5 Inaccurate matric

Suppose we have an inccurate matric which gives us the distance to obstacle: dist[i] = accurate_dist[i]/c[i], where $1 \leq$ c[i] $\leq$ upper_bound. When sampling at a point near the obstacle, dist[i] = accurate_dist[i]/c[i] = δ the algorithm will stop sampling at that point. So the farthest distance from spheres boundary to obstacles is δ * upper_bound.

if the optimal has δ * upper_bound clearance. The algorithm can still find a path.