```
movDis (generic function with 1 method)
```

```julia
##########generate movement according to force and particle property###########
function movDis(fTmp2::Vector{Float64},Rs::Float64,ρp::Float64)
fTmp=deepcopy(fTmp2);
Δt=0.01;
aa=0.5*Δt*Δt*3/4/π/Rs/Rs/Rs/ρp; #coefficient in calculating displacement in each step
return aa.*fTmp;
end
```

```
sepCheck (generic function with 1 method)
```

```julia
###########################in case of overlapping#########################
function sepCheck(parPos::Matrix{Float64},Rs::Float64)
parPosTmp=deepcopy(parPos);
RTmp=Rs+0.00015;
pNo=length(parPosTmp[:,1]);
sepDisDataTmp=fill(16*RTmp*RTmp,sum(1:pNo-1));
countTmp=1;
for iTmp in 1:pNo-1
    for jTmp in iTmp+1:pNo
        sepDisDataTmp[countTmp]=sum((parPosTmp[iTmp,:].-parPosTmp[jTmp,:]).*(parPosTmp[iTmp,:].-parPosTmp[jTmp,:]));
        countTmp+=1;
    end
end
if findmin(sepDisDataTmp)[1]<=(2*RTmp)^2
    println("OVERLAPPING!")
    countTmp=1;
    for iTmp in 1:pNo-1
        for jTmp in iTmp+1:pNo
            if sepDisDataTmp[countTmp]<(2*RTmp)^2
                θTmp=acos((parPosTmp[jTmp,3]-parPosTmp[iTmp,3])/max(0.0000001,sqrt(sepDisDataTmp[countTmp])));
                yyTmp=parPosTmp[jTmp,2]-parPosTmp[iTmp,2];
                xxTmp=parPosTmp[jTmp,1]-parPosTmp[iTmp,1];
                if yyTmp==0&&xxTmp>0
                    ϕTmp=0.0;
                elseif yyTmp==0&&xxTmp<0
                    ϕTmp=π;
                else
                    ϕTmp=(1-sign(yyTmp))*π+sign(yyTmp)*acos(xxTmp/max(sqrt(xxTmp*xxTmp+yyTmp*yyTmp),0.00000001));
                end
                parPosTmp[jTmp,1]=parPosTmp[iTmp,1]+3*Rs*sin(θTmp)*cos(ϕTmp);
                parPosTmp[jTmp,2]=parPosTmp[iTmp,2]+3*Rs*sin(θTmp)*sin(ϕTmp);
                #parPosTmp[jTmp,3]=parPosTmp[iTmp,3]+3*Rs*cos(θTmp);
                parPosTmp[jTmp,3]=0.0;
            end
            countTmp+=1;
        end
    end
    sepCheck(parPosTmp,Rs);
else
    return parPosTmp;
end
end
```

```
disUpCheck (generic function with 1 method)
```

```julia
####################in case of too large movement######################
function disUpCheck(disTmp2::Vector{Float64},maxStepTmp::Float64)
disTmp=deepcopy(disTmp2);
if findmax(abs,disTmp)[1]>maxStepTmp
    println("rescale too large displacement")
    disTmp=disTmp./2;
    disUpCheck(disTmp,maxStepTmp);
else
    return disTmp;
end
end
```

```
disLowCheck (generic function with 1 method)
```

```julia
####################in case of too small movement######################
function disLowCheck(disTmp2::Vector{Float64},minStepTmp::Float64)
disTmp=deepcopy(disTmp2);
if findmax(abs,disTmp)[1]<minStepTmp
    println("rescale too small displacement")
    disTmp=disTmp.*2;
    disLowCheck(disTmp,minStepTmp);
else
    return disTmp;
end
end
```

```
reScal (generic function with 1 method)
```

```julia
############in case of too large gap between displacements############
function reScal(disTmp2::Vector{Float64})
disTmp=deepcopy(disTmp2);
dimension=3;     #for 3D
pNo=Int(length(disTmp)/dimension);     #for 3D
disTmp2=Matrix{Float64}(undef,pNo,dimension);
disTmp3=fill(0.0,pNo);
minMaxDis=1000000.0;
minMaxInd=1;
for iTmp in 1:pNo
    for jTmp in 1: dimension
        disTmp2[iTmp,jTmp]=disTmp[(iTmp-1)*dimension+jTmp];
    end
    disTmp3[iTmp]=findmax(abs,disTmp2[iTmp,:])[1];
    if minMaxDis>disTmp3[iTmp] && disTmp3[iTmp]!=0
        minMaxDis=disTmp3[iTmp];
        minMaxInd=iTmp;
    end
end
maxDisTmp=findmax(abs,disTmp)[1];
if maxDisTmp/minMaxDis>100
    for iTmp in 1:pNo
        disTmp2[iTmp,:]=disTmp2[iTmp,:]*sqrt(maxDisTmp/max(disTmp3[iTmp],0.0000001));
    end
    for iTmp in 1:pNo
        for jTmp in 1: dimension
            disTmp[(iTmp-1)*dimension+jTmp]=disTmp2[iTmp,jTmp];
        end
    end
    reScal(disTmp);
else
    for iTmp in 1:pNo
        for jTmp in 1: dimension
            disTmp[(iTmp-1)*dimension+jTmp]=disTmp2[iTmp,jTmp];
        end
    end
    return disTmp
end
end
```

```
ensMov (generic function with 1 method)
```

```julia
################in case of ensemble movement################
function ensMov(disTmp2::Vector{Float64})
disTmp=deepcopy(disTmp2);
countTmp=0;
dimension=3;     #for 3D
pNo=Int(length(disTmp)/dimension);     #for 3D
disTmp2=Matrix{Float64}(undef,pNo,dimension);
for iTmp in 1:pNo
    for jTmp in 1:dimension
        disTmp2[iTmp,jTmp]=disTmp[(iTmp-1)*dimension+jTmp];
    end
    if disTmp2[1,:]==disTmp2[iTmp,:]
        countTmp+=1;
    end
end
if countTmp==pNo
    return 1
else
    return 0
end
end
```

```
md (generic function with 1 method)
```

```julia
###############molecule dynamics for equilibrium###############
function md(parPos::Matrix{Float64},Rs::Float64)
parPosTmp=deepcopy(parPos);
precSet=10^-6;  #set the precision of equilibrium position
rndDigNo=9; #digit No. of round force.
minStepTmp=0.0002;  #minimal step
maxStepTmp=minStepTmp*10;   #maximal step
maxLopNo=500;   #max loop No to find equilibrium position
dimension=3;      #for 3D
ρp=29.0;          #particle density
pNo=length(parPosTmp[:,1]);
parPosData=Array{Float64}(undef,maxLopNo*10,dimension,pNo);
parPosTmp=sepCheck(parPosTmp,Rs);
forceTmp=round.(forcePackLow(Rs,parPosTmp);digits=rndDigNo);
disTmp2=movDis(forceTmp,Rs,ρp);
if findmax(abs,disTmp2)[1]==0
    println("particles are in EQUILIBRIUM positions!")
    return parPosTmp
end
countTmp=1;
countTmp2=1;
while minStepTmp>=precSet
    minStepTmp=minStepTmp/2;
    maxStepTmp=maxStepTmp/2;
    for lopTmp in 1:maxLopNo
        disTmp=movDis(forceTmp,Rs,ρp);
        if findmax(abs,disTmp)[1]==0
            println("particles are in EQUILIBRIUM positions!")
            return parPosTmp
        end
        if ensMov(disTmp)==1
            println("EQUILIBRIUM, ENSEMBLE MOVEMENT")
            println("Rs: "*string(Rs))
            println(parPosTmp)
            return parPosTmp
        end
        disTmp=reScal(disTmp);
        disTmp=disUpCheck(disTmp,maxStepTmp);
        disTmp=disLowCheck(disTmp,minStepTmp);
        if findmax(disTmp.*disTmp2)[1]<=0   #condition of equilibrium
            println("particles are in EQUILIBRIUM positions!")
            println("precision:",maxStepTmp)
            for iTmp in 1:pNo
                for jTmp in 1:dimension
                    parPosTmp[iTmp,jTmp]+=disTmp[dimension*(iTmp-1)+jTmp]/2; #update particles position
                    parPosTmp[iTmp,3]=0.0;
                    parPosData[countTmp,jTmp,iTmp]=parPosTmp[iTmp,jTmp];
                end
            end
            disTmp2=deepcopy(disTmp);
            break
        else
```

```
            for iTmp in 1:pNo
                for jTmp in 1:dimension
                    parPosTmp[iTmp,jTmp]+=disTmp[dimension*(iTmp-1)+jTmp];  #update particles position
                    parPosTmp[iTmp,3]=0.0;
                    parPosData[countTmp,jTmp,iTmp]=parPosTmp[iTmp,jTmp];
                end
            end
            disTmp2=deepcopy(disTmp);
            parPosTmp=sepCheck(parPosTmp,Rs);
            forceTmp=round.(forcePackLow(Rs,parPosTmp);digits=rndDigNo);
        end
        parPosData2=parPosData[1:countTmp,:,:];
        plot3d(plottitles=countTmp);
        println(["Max Displacement: " * string(maxStepTmp), "Total steps: " * string(countTmp),"RND No.: " * string(countTmp2),"Step No.: " * string(lopTmp)])
        for iTmp in 1:pNo
        display(path3d!(parPosData2[:,1,iTmp],parPosData2[:,2,iTmp],parPosData2[:,3,iTmp]))
        end
        println(disTmp)
        println("current particle positions:")
        println(parPosTmp)
        countTmp+=1;
    if lopTmp==maxLopNo
        println("Loop No. reaches Maximum!")
    end
    end
    countTmp2+=1;
end
return parPosTmp
end
```