```
//-----------------------------------------------------------//
// Javascript code to simulate DLA (Diffusion Limited       //
// Aggregation). Written by Marc Joiret                     //
// Last update : March 17, 2018.                            //
//-----------------------------------------------------------//
var myCanvas;
var startTime;
var lastTime;
var lagTime = 0;
var walkersNumber = 999;// number of intial random walkers.
var walkerRadius = 4;   // radius of random walker.
var fixedNList = [750, 625, 480, 400, 350, 250, 150, 100, 50,
10];
var revfixedNList = [10, 50, 100, 150, 250, 350, 400, 480, 625,
750];
var RandomWalkers = []; // array of objects Random Walkers.
var Aggregates = [];    // array of object Aggregated Walkers =
stuck walkers.
var minX = 9999;
var maxX = -1;
var minY = 9999;
var maxY = -1;
var dataX = [];
var meanSquaredLength = [];
var dataTime = [];
var correctedTime = [];
var dataLogX = []; //array required to infere fractal dimension
and plot
var dataLogY = []; //array required to infere fractal dimension
and plot
var data = [];
var data2 = [];
var beta1;
var sampleSize = 0;
var px = [];
var py = [];
var notYetProcessed = true;
var N;

// Functions :
function reInitVar(){
  fixedNList = [750, 625, 480, 400, 350, 250, 150, 100, 50, 10];
  revfixedNList = [10, 50, 100, 150, 250, 350, 400, 480, 625,
  750];
```

```
    RandomWalkers = [];
    Aggregates = [];
    lastTime = 0;
    minX = 9999;
    maxX = -1;
    minY = 9999;
    maxY = -1;
    dataX = [];
    meanSquaredLength = [];
    dataTime = [];
    correctedTime = [];
    dataLogX = []; //array required to infere fractal dimension
    and plot
    dataLogY = []; //array required to infere fractal dimension
    and plot
    data = [];
    data2 = [];
    sampleSize = 0;
    px = [];
    py = [];
    notYetProcessed = true;

}
function sum(data){
    var result = 0;
    for (var i=0; i< data.length; i++){
        result += data[i];
    }
    return result;
}

function sumOfSquares(data){
    var result = 0;
    for (var i=0; i< data.length; i++){
        result += data[i] * data[i];
    }
    return result;
}

function sumOfProd(data, data2){
    var result = 0;
    for (var i=0; i< data.length; i++){
        result += data[i] * data2[i];
```

```
  }
  return result;
}

function stickingPosition(xSticking, ySticking, xAgg, yAgg){
  // input : the first two arguments are the coordinates of the approaching
  // Random walker, the last two arguments are the coordinates of the
  // closest circle element currently belonging to the aggregates.
  // output : returns a vector whose coordinates are the center of the
  // sticking circle tangent to the aggregate closest circle.
  var dX = xSticking - xAgg;
  var dY = ySticking - yAgg;
  var ds = sqrt(dX * dX + dY * dY);
  if (ds == 0 || ds == undefined){
    var xStuck = xAgg;
    var yStuck = yAgg;
  }
  else{
  var xStuck = xAgg + dX * 2 * walkerRadius/ds;
  var yStuck = yAgg + dY * 2 * walkerRadius/ds;
  }
  var vectorStuck = createVector(xStuck, yStuck);
  return vectorStuck;
}
function setup() {
      myCanvas = createCanvas(1024, 636); //(512, 512)
      myCanvas.parent("myCanvasContainer");
      frameRate(60);
      resetSketch();
      // button to reset and launch the sketch again from htlm...:
      var button = createButton("Click here to start a new aggregation and bring the mouse back in the frame");
      button.parent("myButtonContainer");
      button.mousePressed(resetSketch);
}

function resetSketch(){
  // re-initialize all arrays and varaibles to start from scratch again.
```

```
    reInitVar();

    for (var i = 0; i < walkersNumber; i++){
        var theta = random(0, 2*PI);
        randX = 254 + 250 * cos(theta);
        randY = 254 + 250 * sin(theta);
        RandomWalkers[i] = new RandomWalker(randX, randY, false);
      }
    startTime = millis()/1000; //start time in seconds.

}

function draw() {
    var fr = frameRate();
    if (mouseX >= 0 && mouseX <= 512 && mouseY >= 0 && mouseY <=
    512){
    background(255, 255, 255);
    strokeWeight(4);
    stroke(0, 0, 0);
    line(0, 512, 1024, 512);
    line(512, 0, 512, 636);

    // seed the aggregate with one fixed particle at the center of
    the canvas:
    Aggregates[0] = new RandomWalker(512/2, 512/2, true);
    Aggregates[0].display(0, 0, 0); //0, 77 ,0 would be dark green.

      // display the current array of aggregates :
    for (var i = 0; i < Aggregates.length; i++){
      Aggregates[i].display(0, 0, 0); //0, 77 ,0 would be dark
      green
    }

    // move and display the current array of random walkers :
    for (var i = RandomWalkers.length - 1; i >= 0; i--){
      RandomWalkers[i].move();
      RandomWalkers[i].display(255, 99, 71);
      // If sticking condition is met (and with no overlapping to
      others) :
      if (RandomWalkers[i].isStuck(Aggregates)){
        //  - (1) add the stuck walker to the aggregate :
        Aggregates.push(RandomWalkers[i]);
        //  - (2) remove the stuck walker from the random walkers:
```

```
        RandomWalkers.splice(i, 1);
      }
    }
    // Compute fractal metrics for the current aggregate:
  var currentTime = millis()/1000;
  if (RandomWalkers.length > 0){
    var elapsedTime = currentTime - startTime;
    lastTime = elapsedTime;
  } else{
    var elapsedTime = lastTime;
  };
  N = Aggregates.length;
  if (N < 3){
    lagTime = elapsedTime;
  };
  var WalkerArea = PI * walkerRadius - walkerRadius;
  var W = N * WalkerArea;
  // ... Compute aggregate highest width and highest height:

  for (var i = 0; i < Aggregates.length; i++){
    if (Aggregates[i].x < minX){
      minX = Aggregates[i].x
    };
    if (Aggregates[i].y < minY){
      minY = Aggregates[i].y
    };
    if (Aggregates[i].y > maxY){
      maxY = Aggregates[i].y
    };
    if (Aggregates[i].x > maxX){
      maxX = Aggregates[i].x
    };
  }
  var highestWidth = maxX - minX;
  var highestHeight = maxY - minY;
  // Write aggregate fractal properties on the canvas bottom box:
  var lineOne ="Elapsed time: " + nfs(elapsedTime, 3, 1) + " sec."
  + "   Lag time: " + nfs(lagTime, 2, 1) + " sec.";
  var lineTwo ="Aggregate weight: "+"count= " + nf(N,4) + "
  weight=" + nfs(W,5,1) + " pixels.";
  var lineThree ="Aggregate width: " + nf(highestWidth,3,1) + "
  pixel length units.";
  var lineFour ="Aggregate height: " + nf(highestHeight,3,1) + "
  pixel length units.";
```

```
var lineWidth = "width = " + nf(highestWidth,3,1) + " pixels.";
var lineHeight ="height = " + nf(highestHeight,3,1) + " pixels.";

textFont("Helvetica"); //Courier
textSize(18);
fill(100, 0, 170); //(75, 0, 130)
text(lineOne, 10, 536);
text(lineTwo, 10, 566);
text(lineThree, 10, 596);
text(lineFour, 10, 626);

// Select and store density and mean length of aggregate at
• different fixed
// sizes : when in fixedNList walkers have
// been stuck:

k = fixedNList.length - 1;
if (N >= fixedNList[k]){
  var area = highestWidth*highestHeight;
  var geomLength = sqrt(area);
  var density = fixedNList[k]/area;
  dataTime.push(elapsedTime);
  correctedTime.push(elapsedTime - lagTime);
  dataX.push(geomLength);
  meanSquaredLength.push(area);
  dataLogX.push(log(geomLength)/log(10));
  dataLogY.push(log(density)/log(10));
  fixedNList.pop();
  sampleSize += 1;
  };

// Compute regression statistics for log-log plot when the 10
• points are available
if (k < 0){
  var beta0;
  //var beta1;
  var SSxx;
  var SSyy;
  var SSxy;
  var SSE;
  var Sx = sum(dataLogX);
  var Sy = sum(dataLogY);
  var meanX = Sx / sampleSize;
  var meanY = Sy / sampleSize;
```

```
    var meanY = Sy / sampleSize;
    SSxx = sumOfSquares(dataLogX) - Sx * Sx / sampleSize;
    SSyy = sumOfSquares(dataLogY) - Sy * Sy / sampleSize;
    SSxy = sumOfProd(dataLogX, dataLogY) - Sx * Sy / sampleSize;
    SSE = SSyy - SSxy * SSxy / SSxx;
    var sbeta1 = sqrt(SSE/((sampleSize - 2) * SSxx));
    beta1 = SSxy / SSxx;
    beta0 = meanY - beta1 * meanX;
    var r2 = SSxy * SSxy / (SSxx * SSyy);
    var fractalDim = 2.0 + beta1;
    // Write aggregate fractal statistics on the canvas right
    bottom box:
    var lineFive ="Log-log slope: " + nfs(beta1,1, 3);
    var lineSix ="Goodness of fit r² : " + nfs(r2, 1, 3);
    var lineSeven ="Fractal dimension: " + nfs(fractalDim,1,3);
    //t95_8 = 2.306
    var CLinf = 2 + (beta1 - 2.306 * sbeta1);
    var CLsup = 2 + (beta1 + 2.306 * sbeta1);
    var lineEight ="Fractal dimension 95% CI: [" + nfs(CLinf,1,3)
    + " - " + nfs(CLsup,1,3) +" ]";
    textFont("Helvetica"); //Courier
    textSize(18);
    fill(100, 0, 170); //(75, 0, 130)
    text(lineFive, 522, 536);
    text(lineSix, 522, 566);
    text(lineSeven, 522, 596);
    text(lineEight, 522, 626);

    notYetProcessed = false;
};

// Plot log-log for fractal scale invariance:
// map (x, y) to (px, py) on the canvas :
var lineNine ="log of density as a function of";
var lineTen ="log of length geometric mean.";
var ord ="log density";
var absc = "log L";
var angleArc = atan(-beta1);
var comment = "slope = df - d";

var pxInf = 552;
var pxSup = 1004; // width = 452 pixels
var pyInf = 20;
var pySup = 472; // height = 452 pixels
```

```
279
280  var xInf = 1.2; // log Length
281  var xSup = 2.95;
282  var yInf = -1.05; // Log density
283  var ySup = -2.80;
284  // draw legend:
285  push();
286  textFont("Helvetica"); //Courier
287  textSize(18);
288  fill(120, 125, 120); //(75, 0, 130)
289  text("Power law estimation for aggregate :", 522 +140, pyInf -2);
290  text(lineNine, 522 + 200, pyInf + 35);
291  text(lineTen, 522 + 200, pyInf + 55);
292  pop();
293  // draw axes:
294  push()
295  strokeWeight(2);
296  stroke(120, 125, 120);
297  line(pxInf,pyInf+10, pxInf, pySup); // y-axis
298  line(pxInf, pySup, pxSup, pySup); // x-axis
299  pop();
300  push();
301  textSize(14);
302  fill(120, 125, 120);
303  text(ord, pxInf - 30, pyInf - 2);
304  text(absc, pxSup - 45, pySup + 15);
305  text(comment, pxInf + 230, pyInf + 305);
306  pop();
307  push();
308  strokeWeight(2);
309  stroke(120, 125, 120);
310  // draw the horizontal dashed line arc for slope representation:
311  var angleArc = atan(-beta1);
312  var xcenter = 1.5;
313  var ycenter = beta0 + beta1 * xcenter;
314  var mxcenter = map(xcenter, xInf, xSup, pxInf, pxSup);
315  var mycenter = map(ycenter, yInf, ySup, pyInf, pySup);
316  for (var i=0; i<=16; i++){// dashed line above slope comment
317    line(pxInf + 80 + i*20, mycenter, pxInf + 90 + i*20, mycenter);
318  }
319  pop();
320  // draw arc for slope legend :
321  push();
322  fill(100, 0, 170, 10);
```

```
  strokeWeight(2)
  stroke(120, 125, 120, 255);
  arc(mxcenter, mycenter, 540, 540, 0, angleArc); // test minus
  pop();

  // draw points :
  for (var i = 0; i < sampleSize; i++){
    strokeWeight(6);
    stroke(255, 0, 0);
    px[i] = map(dataLogX[i], xInf, xSup, pxInf, pxSup);
    py[i] = map(dataLogY[i], yInf, ySup, pyInf, pySup); // ???
    point (px[i], py[i]);
  }
  // draw regression line:
  var x1 = 1.3;
  var x2 = 2.7;
  var y1 = beta0 + beta1 * x1;
  var y2 = beta0 + beta1 * x2;
  var mx1 = map(x1, xInf, xSup, pxInf, pxSup);
  var my1 = map(y1,  yInf, ySup, pyInf, pySup);
  var mx2 = map(x2, xInf, xSup, pxInf, pxSup);
  var my2 = map(y2,  yInf, ySup, pyInf, pySup);
  pop();
  push();
  strokeWeight(2)
  stroke(100, 0, 170);
  line(mx1, my1, mx2, my2);
  // draw ticks :
  var ticklength = 7;
  line(pxInf, py[0], pxInf + ticklength, py[0]);
  line(pxInf, py[1], pxInf + ticklength, py[1]);
  line(pxInf, py[6], pxInf + ticklength, py[6]);
  line(px[0], pySup - ticklength, px[0], pySup);
  line(px[1], pySup - ticklength, px[1], pySup);
  line(px[6], pySup - ticklength, px[6], pySup);
  pop();
  // write scales x and y:
  push();
  noStroke();
  fill(120, 125, 120);
  textSize(12);
  text("n                    = "+ nfs(revfixedNList, 3,0), 565,
  pyInf + 110);
```

```
365  if (dataLogY[9] != undefined){
366    text("log density  = "+ nfs(dataLogY, 1, 2), 565, pyInf + 125);
367    text("log L            = "+ nfs(dataLogX, 1, 2), 565, pyInf +
     •  140);
368    text(nfs(dataLogY[0], 1, 2), pxInf - 32, py[0] + 5);
369    text(nfs(dataLogX[0], 1, 2), px[0] - 15, pySup + 20);
370
371    text(nfs(dataLogY[1], 1, 2), pxInf - 32, py[1] + 5);
372    text(nfs(dataLogX[1], 1, 2), px[1] - 15, pySup + 20);
373
374    text(nfs(dataLogY[6], 1, 2), pxInf - 32, py[6] + 5);
375    text(nfs(dataLogX[6], 1, 2), px[6] - 15, pySup + 20);
376  }
377  pop();
378
379  // mouseIsPressed action : draw the width and height of current
     •  aggregate:
380  if(mouseIsPressed){
381    var d = walkerRadius;
382    strokeWeight(1);
383    stroke(100,0,170);
384
385    line(minX-d, minY-d, maxX+d, minY-d);
386    line(maxX+d, minY-d, maxX+d, maxY+d);
387    line(minX-d,maxY+d, maxX+d,maxY+d);
388    line(minX-d,minY-d,minX-d,maxY+d);
389
390    noStroke();
391    textFont("Helvetica");
392    textSize(14);
393    if (minY > 20){
394      text(lineWidth, minX, minY - 15);
395    } else{
396      text(lineWidth, minX + 100, minY + 20);
397    };
398    push();
399    if (minX < 20){
400      var transX = 18;
401    } else {
402      var transX = minX - 10;
403    };
404    translate(transX, (minY + maxY/2));
405    rotate(-HALF_PI);
406    text(lineHeight, 40, 0);
```

```
407     pop();
408   }; // end if mouse is pressed
409
410   }; // end if mouse is in the canvas.
411   } // end of draw -------------
412
413   // Objects are defined hereafter with Constructor function :
414   // -- Object : Random Walker :
415
416   function RandomWalker(tempX, tempY, stuck){
417     this.x = tempX;
418     this.y = tempY;
419     this.stuck = false || stuck;
420     this.stuckTime = 0.0;
421     this.radius = walkerRadius;
422     this.stepLength = 0 + this.radius;
423     var overlapping = false; //?
424     this.isStuck = function(others){
425       //var overlapping = false; //?
426       // What is the sticking condition ?
427       //  stuck = true if any pairwise squared distance between
          current
428       //  RandomWalker and array others[i] is equal or less than 4
          * radius².
429       // var overlapping = false;
430       for (var i = 0; i < others.length; i++){
431         var deltaX = this.x - others[i].x;
432         var deltaY = this.y - others[i].y;
433         var squaredDistance = deltaX * deltaX + deltaY * deltaY;
434         if (squaredDistance <= 4 * walkerRadius * walkerRadius){
435           // compute the correct temporary stuck position
436           // (circles must be tangent):
437           var vectNewStuck = createVector(0, 0);
438           vectNewStuck = stickingPosition(this.x, this.y,
              others[i].x, others[i].y);
439           this.x = vectNewStuck.x;
440           this.y = vectNewStuck.y;
441           // The temporary discovered new stuck position could be
              such
442           // that the candidate stuck walker is overlapping yet
              another
443           // aggregate element.
444           // We should check this before validating the stuck
```

```
        candidate.
445     // If the candidate stuck walker is overlapping some
        other, we will
446     // discard the candidate by breaking the loop. This would
447     // make sure there will be no circles overlapping of any
        kind.
448     // ----**********------overlap check :
449     for (var j = 0; j < others.length; j++){
450       var deltaX = this.x - others[j].x;
451       var deltaY = this.y - others[j].y;
452       var squaredDistance = deltaX * deltaX + deltaY *
        deltaY;
453       if (squaredDistance < 4 * walkerRadius * walkerRadius){
454         // (notice strictly less than ...in the line above)
455         overlapping = true;
456         break;
457       } else {
458         overlapping = false;
459       }
460     }
461     if (!overlapping){
462     // ----**********------end overlap check
463     //  record stuck time:
464       var currentTime = millis()/1000;
465       this.stuckTime = currentTime - startTime;
466       return true;
467       break;
468     }
469     else { // overlapping was found
470       return false;
471     }
472   }
473 }
474 //return false;
475 };
476
477 this.move = function(){
478   var angle = random(0, 2*PI);
479   this.x += cos(angle) * this.stepLength;
480   this.y += sin(angle) * this.stepLength;
481   this.x = constrain(this.x, 0, 512);
482   this.y = constrain(this.y, 0, 512);
483 };
484
```

```
485     this.display = function(r, g, b){
486       this.red = r;
487       this.green = g;
488       this.blue = b;
489       noStroke();
490       fill(this.red, this.green, this.blue);
491       ellipse(this.x, this.y, this.radius * 2, this.radius * 2);
492     };
493   };
494
```