Variable precision distance search for random fractal cluster simulations.

SOSA-HERRERA ANTONIO[1], RODRIGUEZ-ROMO SUEMI[2]
[1]Computational Science Graduate Program, [2]Center of Theoretical Research
Universidad Nacional Autónoma de México, Campus Cuautitlán.
Av. 1 de Mayo s/n. Col.Atlanta. 54740 Cuautitlán Izcalli, Edo. Mex.
MEXICO
jasosah@hotmail.com, suemi@servidor.unam.mx

Abstract: - A simple an effective algorithm for performing distance queries between a large number of points stored in quadtrees and octrees. The algorithm is developed and tested for the construction of diffusion-limited aggregates. To achieve an enhancement on the searching time we accept approximate distance values with low precision at the first levels of the hierarchical structure, and accurate ones at the last level. The structure of the trees is the only feature used for the determination of approximate distances at any stage. These techniques allowed us to build DLA clusters with up to 10^9 particles for the two-dimensional case and up to 10^8 particles for the three-dimensional case. We also worked with the PDLA model obtaining fractal clusters with up to 10^{10} and 10^9 particles for two and three dimension clusters respectively. We worked on a supercomputer to run the PDLA simulations, as well as a high performance server for DLA simulations. We employed POSIX threads to provide parallelization and mutexes as control mechanisms to achieve synchronization between groups of 4 processors, hence simulating PDLA clusters with early convergence to the DLA model.

Key-Words: - DLA , PDLA, distance queries, nearest neighbor, fractal cluster, quadtree, octree

1 Introduction

Distance queries in multidimensional data structures are a very important matter in computer applications like robotics, collision detection, computer graphics, pattern recognition, path planning, and simulation of physical phenomena, among others. Nowadays there is a wide variety of well designed algorithms to handle distance queries in both accurate and approximate fashions [1]. Here we introduce a new variant for searching in quadtrees and octrees that yields accurate and approximate distance values depending on the relative magnitude of such values. This paper is focused in the fast calculation of the distance between a moving query point and a set consisting of a large number of points scattered on a space domain. To improve the performance for the distance queries we take advantage of the precision degree previously known as acceptable in each query for the computation of the distance between a query point and a very large set of points by looking at the nodes of the structure (a quadtree or an octree) on which the data is stored, the area they cover in space, and the distance between the query point and the environment by means of their relation in terms of scale.

A previous idea of having several representations for a diverse set of scales has already been successfully applied for random fractal cluster simulations by the use of bitmaps [2]. We introduce in this paper a different method which has the advantage that it does not make use any kind of map for storing additional information about a variety of scale views of the cluster, instead of that, we take advantage of the internal structure of the quadtree or octree that stores data relative to space in a hierarchical way, this allowed us to save memory and speed up calculations in order to run dynamical cluster simulations having a huge number of elements. In this way, we do not need to store additional data neither to update states of external maps. Here we present our method for Euclidean dimensions $d = 2$ and $d = 3$, this can be straightforwardly extended to higher dimensions, remarking the fact that the number of children for each node in the corresponding octree will be 2^d.

While we present the DLA simulation as a study case, we think this distance search technique might be applied to a variety of problems on which precision requirements can be handled in the same way as it is shown for this particular example.

Creating fractal clusters with a high number of particles has been already treated several times, due to the importance of the exactitude for simulation and establishment of asymptotical parameters for many natural phenomena involving fractal growth [3-6]. The Diffusion Limited Aggregation model,

described by Witten & Sanders [7] was firstly simulated with a few thousands of particles. A significant improvement for simulating the model was given by Ball & Brady [2] by the introduction of large step random leaps for particle moving over empty areas. This was achieved using a hierarchy of grid maps. These techniques allowed Tolman & Meakin [8] to build DLA clusters with up to 10^7 particles. Kaufman et all. [9], introduce the use of quadtrees for the efficient storage of particles obtaining clusters with up to 10^8 particles by using 32 processors working in parallel and based on the PDLA model also described in the same work. This last method was later used by Mandelbrot to study lacunarity [10] over 10^8 particle clusters created using a single processor. We started from the background references cited above and following the methodology described in the next sections we have obtained fractal DLA clusters consisting of $N=10^9$ particles generated by a single in processor in an average time of 14 hours. To improve these figures, we made use of a parallel multiprocessor system to generate clusters following the PDLA model, which converges to the DLA model for a large number of particles [9]. Implementing these techniques with our variable precision distance search algorithm, we constructed 10^{10} particle PDLA clusters with a high concordance in the mass-radius behavior with respect to the DLA model. These simulations required in average 52 hours of execution time using four sharing memory processor per job. For simulation of the three-dimensional DLA, growing 3D-clusters with 10^8 particles took 4.5 hours in an average working in 2D, and 25 hours to obtain a 10^9 PDLA cluster in 3D with four processors

2 Diffusion-Limited Aggregation

The diffusion limited aggregation model introduced by Witten & Sander [7] can be described as the following iterative process: A seed particle is placed at the origin, this is the initial cluster. Another particle makes a random walk starting from one place far from the origin usually called birth radius R_{bh} and ending when the particle visits a location adjacent to the cluster, then the moving particle become part of it. If the particle exceeds some boundaries established by the death radius R_{dh} then it is discarded. The process is repeated until certain number of particles conforming the cluster is reached. The distance from the farthest particle of the cluster to the origin is sometimes called exterior radius R_{ext}.The DLA model describes a large number of natural growth phenomena.

Although its algorithmic sketch seems quite simple, many implementation techniques are required to achieve the simulation of large clusters. Having large DLA fractal aggregates is important for the analysis of asymptotical properties of this model which covers a wide variety of growth phenomena. The main difficulty that appears working with large clusters is that a moving particle must be aware of all other particles in the cluster, its necessary then to know the distance from the particle to the cluster and this value changes at each step of the Brownian motion, so a massive number of queries has to be performed in order to get large DLA clusters. Brownian motion describes the erratic behavior of microscopic particles suspended in a liquid. Hence it is an important model to explain diffusive aspects on many natural processes. Applications of Brownian motion are extended to many other areas, for example determination of shapes for natural landscapes [11, 12].

3 Variable Precision Search for DLA and PDLA Simulations.

In this section we introduce our variable precision distance search algorithm to make queries about distance between one point and a set of points stored in a bucket quadtree or octree. This kind of search exploits the implicit information about space contained in the data structure by looking for the first area free of particles that encounters and then, depending on the level in which this area was found it sets an approximate distance. We applied this algorithm to DLA and PDLA model simulations and present the obtained results on performance and fractal measurements taken from the fractal clusters generated.

3.1 Variable Precision Distance Queries In Quadtrees And Octrees.

Roughly speaking, the variable precision distance search algorithm works by looking for the distance between a query point and the center of the occupied nodes at the first levels of the quadtree, if the query point is far enough, then an approximate distance is returned, otherwise the algorithm will verify if it is in a leaf containing points and then returning an exact distance. This procedure is carried on recursively through all the levels of the quadtree. The method is exactly described in the pseudo code shown in Listing 1. A brief explanation of the functions used in the pseudo-code is given at Table

1. To examine the potential of our algorithm we analyze the possible cases and values returned by the procedure above described. Suppose the algorithm defined by the function VPDIST_SEARCH given in Listing 1 is applied to find the distance between a query point p and a set of points $A \neq \emptyset$ stored in a quadtree Q, assume that p is in the geometric domain of Q. Let d_g be the length of the diagonal that crosses the area covered by one node at the last level n of Q and let $\mathrm{d}(p,q)$ be the distance between p and its nearest neighbor $q \in A$. Then the algorithm returns $\mathrm{d}(p,q)$ if the distance from p and the center of the node at level n containing q is less than $2d_g$, otherwise it returns $\mathrm{d}(p,q)$ with an error of at least $+0.2\,\mathrm{d}(p,q)$ and at most $+0.4\mathrm{d}(p,q)$. To prove this, we take the cases for which the function returns exact and approximate distances. For the exact case it is clear that the condition of line *13* in listing 1 will be true for a search at level n and line *07* tell us that the exact verification is done when $\mathrm{d}(p,q) < 2d_g$. Moreover, line 07 also tells that if $n \leq 2$ the algorithm will always yield $\mathrm{d}(p,q)$. The calls in lines *24* to *32* ensure that all the potential nodes for nearest neighbors will be searched at any level of Q due to the intersection conditions on lines *25*, *28* and *31*. On the other hand, if an approximate distance d_{app} is returned then this will be done by inspecting one node η at a level k with $2 < k \leq n$. Let q_c be the center of the area covered by η and r be the distance between q_c and one of the corners of such area. We have $d_{app} = \mathrm{d}(p,q_c) - r$ as result of line *10*. We can see that the closest that p can be from q_c is $4r$ due to the condition stated at line *07*, in the same way, the farthest that p can be from q_c is $9r$, otherwise the condition at line *07* would have been true for the preceding node al level $k-1$, since it checks for a distance greater than $8r$ from p to the center q_{cf} for the father of η, which is the length of the diagonal crossing a node at level $k-2$. Note that $\mathrm{d}(q_{cf}, q_c) = r$ so we have $\mathrm{d}(p,q_c) \leq 9r$. To ensure the expression $\mathrm{d}(p,q) \in [d_{app}, d_{app} + 2r]$ consider that q is not closer than d_{app} to p because in that case the circle with center on p and radius d_{app} would intersect with a not-null node at level $k-1$ and recursion calls in lines *24* to *32* would returned a shorter distance than d_{app}. Furthermore, although η is not necessarily the node at level k that contains q, it has at least one point, so the nearest neighbor is not beyond that $d_{app} + 2r$ from p. Then we have that for the approximate cases the relation $\mathrm{d}(p,q) \in [d_{app}, d_{app} + 2r]$ is held, and this is equivalent to the positive error range for d_{app} stated before.

Fig. 1 illustrates some of the ideas given in the argument above by showing one case in which the query point p_a is the closest possible to the center q_c of the node that is inspected for an approximate distance d_{app}. It also shows the case when the query point p_b is the farthest possible from q_c before the approximate distance would have been taken from an upper level on the quadtree inside the recursive process as the search is performed in a top-down way. The shaded area corresponds to all those points for which the algorithm will return an approximate distance based on q_c at the level depicted in Fig. 1, where the node with center q_c is the north-east child of its father.

Note that the precision given by the algorithm described above might be inadequate for many distance search applications. However it can be pretty useful when a big error is unimportant if it allows a process to be conducted to a further step, as in the problem here studied.

3.2 Simulation of the DLA and PDLA Models.

The technique exposed here is an improvement for the use of the hierarchical data structure called quadtree storing point data for a fractal cluster described by Kaufman *et al* [9], besides simulating the DLA model we worked too with the PDLA model which consists in allowing several particles to move around randomly until they reach the cluster, each particle being handled by a processor so the simulation of the Brownian motion for each particle is done simultaneously. The PDLA model converges to the DLA model for a large number of particles belonging to the cluster [9]. We also employed the hierarchical structure to efficiently store points in main memory, and we used pointers to chunks of particles to reduce the memory space required per particle. The major enhancement we have made with respect to the cited reference is that we use the quadtree not only for efficient storage but also we exploit it to perform approximate distance queries, avoiding the use of hierarchical maps since we take all the information we need from the structure of the

quadtree or octree and working in this way we have a procedure not limited to two-dimensional spaces. We used the algorithm described in the previous section to find out the distance between a moving particle and the cluster. This situation is involved in the simulation of Brownian motion for particles in a complex background defined by all the other points that are already part of the cluster on which we need to know the length of the clearance where a particle can move freely in a random way. High precision in this measure is not a critical issue when the particle is relatively far of the cluster's branches. Besides staying away from hierarchical maps we also avoid the implementation of killing-free techniques [9,13], instead of that, when we have a moving particle at a point p with $|p| > R_{bh}$ we take $|p| - R_{ext}$ as the length of displacement for the next move in a random direction. The idea is illustrated in Fig. 2. In this way we can put R_{dh} as far as $R_{bh} \times 10^2$ without slowing down the simulation in a significant way. This ratio is limited by the use of 64-bit integer arithmetic used for the simulations applied to get our data as described in the section corresponding to the implementation of our program. Using floating point arithmetic the ratio R_{dh} / R_{bh} can be as large as 10^{20} without affecting too much the processing time.

We first applied the variable precision distance search algorithm to work under a Euclidean dimension $d = 2$ .To build 10^{10} particle PDLA clusters we ran our simulations on a *HP Cluster Platform 4000* supercomputer. The jobs were run using groups of 4 *AMD Opteron* processors at 2.4 GHz with 64 GB of shared memory and a 400MHz FSB bandwidth. We used only one processor to grow the clusters up to 10^7 particles under the DLA regime, *i.e.* using only one processor, and then we brought in the other three processors to achieve 10^{10} particles for the PDLA model. This gave us a good convergence from the PDLA to the DLA model as is presented in a later section. The average time required to get a 10^{10} particle PDLA cluster was 42 hours. Memory space required to create a 10^{10} particle PDLA cluster is 42 GB of RAM. A binary file containing raw coordinate data for this cluster occupies about 150 GB of disk space.

In addition, we created 10^9 DLA clusters using a single processor. For these cases the program was ran over on a *PowerEdge 2900* server with 8 *Intel Xeon* processors at 2.66 GHz with 32GB of shared RAM and 1333MHz FSB a bandwidth, as the memory requirements were not so extensive as for the 10^{10} particle clusters. Here the jobs were executed by a single processor as it is established by the DLA model. Here the multiple processors were only providing concurrency to run several jobs at the same time, so we kept reproducing the original DLA model taking an average time of 10 hours per job. This wall-clock time was small as expected because these simulations were performed on the mentioned server with an improved front side bus bandwidth (1333MHz) and a slightly faster CPU speed (2.66 GHz) with respect to the characteristics of the nodes of the *HPCP4000* cluster.

To work with a Euclidean dimension $d = 3$ we modified the data structure changing from a quadtree to an octree, we also modified several functions to be adapted to the new structure but the main steps of the algorithm were still remaining the same. The simulations with the variable precision algorithm were carried out to simulate PDLA clusters with up to 10^9 particles in a space with Euclidean dimension $d = 3$, and once again we had one processor working until it reaches a cluster of 10^7 particles, then we launched other 3 processor running the algorithm for the construction of the same aggregate, forming a group of 4 sharing memory processors to get a cluster with size of 10^9 particles. It took an average of 12 hours to build a 3d-PDLA cluster in this fashion. For the 3d-DLA model we built 10^8 particle clusters in an average time of 4 hours.

Fig. 4 a). shows a picture of one two-dimensional cluster conformed by 10^9 points. In Fig. 4b we can see a PDLA cluster consisting of 10^{10} particles. Fig. 5 also shows a tiny three-dimensional DLA cluster of 3500 particles, this small size cluster is shown in this paper for practical reasons as the structure of large 3d-DLA is completely lost when projected particle by particle to a 2d image.

3.3 Program Implementation.

In this section we describe the details of the code implementation we made to get the DLA and PDLA models working in a computer system. The code was written in *C* language and was compiled with the *GNU Compiler Collection* (*gcc*) utility [14]. To give multiprocessing capabilities to the code we used the the POSIX-threads library [15] running over the *Linux kernel* 2.6.39.3 [16]. We made use of mutexes in order to avoid race conditions that could be generating data inconsistency and particle overlaps. The central function in the implementation performs the simulation of the Brownian motion for one particle. Sequential and parallel procedures were coded inside the same application, giving the option to specify the number of threads that will be executing the Brownian motion function for each different particle moving in the space at a given time. When the parallel model is required a

reference to this function is passed to a different thread for its simultaneous execution. Note that with this structure there must be at least the same number of processors as threads; otherwise there won't be any improvement on simulation speed. The flow diagram detailing the process carried by this central function implementing an off-lattice DLA model for the Euclidean dimension $d = 2$ is shown in Fig. 6. Here N represents the number of particles. A is the set of points that compose the aggregate, R_{bh} and R_{dh} are the birth and death radius respectively. The label *tol* stands for the tolerance distance that particles have to be from the cluster to get added to it and can be interpreted as the particle diameter. The value $\varepsilon << tol$ is an adjustment needed to deal with discrete numeric representation issues. $d(p, A)$ indicates the distance between the point p and the set A, and it is obtained by a call to the variable precision distance search function VPDIST() described in Listing 1. The main loop for the Brownian motion process is identified by the label *1*. This performs the Brownian motion simulation for the current moving particle. The loop labeled with *2* resets the motion for a particle that has travel beyond R_{dh} from the cluster origin puts it again at a random point situated at a distance R_{bh} from the origin. The loop marked with *3* performs the addition of a new particle to the aggregate A. If the desired number of particles has not been already reached, the process begins again. In this last loop the dashed verification shown in Fig. 6 is only performed for the case of having more than one thread executing the function. This avoids particle overlapping as it checks that no particle has been placed in the space where the new particle is going to be added. If there were a collision, the new particle would be discarded. The shadow processes in the flow diagram indicate critical sections that are protected from simultaneous execution from different threads by mutexes. To avoid an excessive blocking time we used three separate mutexes for this critical sections. This operation of adding a new particle to the cluster may involve the creation of new nodes to the tree structure and the allocation for memory space reserved to the program. We use one mutex for the search and possible creation of the node on which the new particle will be added, and another mutex to modify or create the bucket list structure for storing particles. A third mutex protects the actualization of data employed by calculations involving the radius of gyration.

Random numbers for the value θ are generated with the reentrant version of the standard function *drand48()* [14]. In this way we placed uniformly distributed points on a circle of radius R_{bh} taking the polar coordinates $x = R_{bh}\cos(\theta)$, $y = R_{bh}\sin(\theta)$. When having an Euclidean dimension $d = 3$ the generation of uniformly distributed random points over a sphere is done by generating two independent random variables u and θ, and applying the equations

$x = R_{bh}\sqrt{1-u^2}\cos(\theta)$, $y = R_{bh}\sqrt{1-u^2}\sin(\theta)$, $z = R_{bh}u$; these equations replace those shown in the flow diagram from Fig. 6. The functions INTERSECTS() and CIRCLE() shown in line *25* of Listing 1 are also changed for the 3d case to find the intersection of a sphere defining a neighborhood and the limits of the space enclosed by a node.

The VPDIST() function is coded managing orthogonal distances whenever possible, avoiding in this way unnecessary calls to the standard function *sqrt()*, otherwise the overall time execution will be drastically increased as such function calls are executed by the floating point unit with a relative low instruction latency.

To handle the distance difference: $h = \mathrm{DIST}(p, \mathrm{CENTER}(node)) - \mathrm{DIAGONAL}(node)/2$ required by lines *09* and *10* in Listing 1, we take $h^2 = (|x - x_c| - \delta/2)^2 + (|y - y_c| - \delta/2)^2$ where (x, y) and (x_c, y_c) are respectively the coordinates of the moving point p and the center of the node being analyzed. Here δ is the previously calculated value for the diagonal that crosses the space area defined by a node and is a unique value for all nodes at the same level in the quadtree.

Points in the data structure are stored using relative coordinates based on the center of the leaf at which they are kept, the base coordinates are not stored, instead they are calculated incrementally while traversing the nodes of the trees. At the tree structure we have pointers addressing list structures containing chunks of 10 particles. Each relative coordinate uses 1 byte for storage. At the final node of each list there is no next pointer, so we handle them with void pointers and in this way we are able to address at two kinds of data structures with the same pointer.

Since we used 64-bit architecture we worked with integer arithmetic for all data and calculations involved, obtaining an improvement on performance and yet on precision as each coordinate had evenly distributed values under the space taken for the simulations.

3.4 Results.

In order to evaluate the performance on the simulation of the DLA model for the algorithm with variable precision distance search; we needed a reference so we have run experiments using the algorithm and comparing the results with a classic exact nearest neighbor search [17] adapted to a quadtree, instead of using a kd-tree as it is exposed in the original work. Fig. 3 a) shows the overall performance of the methods by presenting the number of particles N against execution time T measured in seconds for the case with Euclidean dimension $d = 2$, and Fig. 3 b) shows a comparison between the $d = 2$ and $d = 3$.

For $d = 2$, we can see empirically by fitting the obtained data to the curves $T = 9.6\exp(9\times10^{-6}N)$ and $T = 3\times10^{-5}N - 6.9$ with a correlation coefficient $r^2 = 0.9981$ (the dashed lines in Fig. 3 a) how the complexity of the DLA simulation is reduced from an exponential trend to a linear one by the use of our techniques. This result is comparable with the ones obtained by using bitmaps at different levels of resolution [2, 8, 9] but our method has the advantages that needs no additional storage for external maps, neither needs to update several hierarchical representations of the cluster, and can be easily extended to dimensions greater than 2. For $d = 3$ the fitting is made with $T = 1\times10^{-4}N - 25.4$ having a correlation coefficient $r^2 = 0.9976$.

The fractal dimension for the aggregates was calculated by the radius of gyration. Table 2 shows the values of fractal dimensionality found in several experiments. Here N is the number of particles for each cluster, P is the number of processors employed, d is the Euclidean dimension of the space that contains the aggregates, and E the number of experiments executed for the computation, the fractal dimension calculated is represented by D, and σ is the standard deviation for this fractal dimension. We were taking 10^3 measurements of the radius of gyration as each cluster was evolving.

To evaluate convergence of the clusters on the PDLA model to the DLA, implemented as described earlier sections, we took data for the radius of gyration obtained from both models. Experimental data is shown in Fig. 7, where we can see no significant variation on the radius of gyration for the PDLA with 4 processors model with respect to the DLA. This is as expected for a low number of parallel processor working with massive clusters [9]. A remarkable fact found here is that we have a higher variance in the measure in the PDLA model and this variance grows with the size of the cluster. For clusters of size $N = 10^{10}$ we have an average value $R_g = 74310.5$ particle diameters with a standard deviation $\sigma = 1119.6$ for the DLA, and $R_g = 75796.5$ with $\sigma = 1980.5$ for the PDLA model. Error bars in Fig. 8 are scaled by a factor of two in order to make them perceptible.

To evaluate multiprocessor efficiency we quantify in PDLA simulations an average speedup of 3.74 when working with 4 processors. Amdahl's law [18] means $speedup = \dfrac{1}{r_s + r_p / n}$, $r_s + r_p = 1$ where r_s represents the ratio of the serial fraction of the program and n the number of processor running the program simultaneously, and tells that the fraction of parallelizable fraction of our executing code is 97.5%. This gives that a 2.5% of execution time on each processor is consumed executing non parallelizable code characteristic of a DLA implementation and being blocked while waiting for the release of a mutex. Given the size of the simulation and the parallelization schemed described we assume that this last percentage is mainly spend by execution threads waiting in a blocked state. These values indicate an acceptable performance taking into account the data dependency of all the entire process.

4 Conclusion

In this paper we have presented the algorithm for distance search with variable precision in quadtrees and octrees and we have also described how can be successfully applied in DLA and PDLA simulations using Euclidean dimensions $d = 2$ and $d = 3$. One of the main benefits of our method is that it does not require external representation of the data so we can handle a huge amount of particles as we do not have to update several states for each representation as the simulation evolves. In this way we have produced faster simulations with less memory space required to manage different precision scales in our models. Another improvement over the traditional way of performing this kind of simulations is that our method can be extended to higher dimensions without difficulty. Dealing with a combination of software and multiprocessor equipment, we attained to build clusters of a size that surpass the ones previously published by others. For this reason we think that these techniques can help to the design of further experiments involving random fractal aggregates with a lot of particles and placed on spaces with different Euclidean dimensions. We're

also planning to adapt this kind of search to other several problems which involve the quadtree and octree data structures to test the advantages and drawbacks that can be raised by applying the algorithm here presented.

5 Acknowledgements.

We want to thank DGSCA-UNAM for providing us access to the HPCP4000 "Kanbalam" supercomputer. This work was in part supported by CONACYT.

References:

[1] Samet Hanan. *Foundations of Multidimensional and Metric Data Structures*. The Morgan Kaufmann Series on Computer Graphics, 2006.

[2] Ball R.C., Brady R.M. Large-Scale Lattice Effect In Diffusion-Limited Aggregation. *Journal Of Physics* A-18 (13): L809, 1985.

[3] Vicsek Tamás. *Fractal Growth Phenomena*. World Scientific Singapore, 1989.

[4] Bunde Armin, Havlin Slomo. Editors. *Fractals and Disordered Systems*. Springer, 1995.

[5] Radu Dobrescu, Loretta Ichim, Stefan Mocanu, Stefan Popa. A Fractal Approach to Pattern Formation in Biological Systems. *WSEAS Transactions on Biology and Biomedicine*. Issue 6, Volume 5, June 2008.

[6] Radu Dobrescu, Catalin Vasilescu, Loretta Ichim. Fractals and Scaling Analysis in Tumor Growth Evaluation. *WSEAS Transactions on Systems*. Issue 1, Volume 6, January 2007

[7] Witten T. A. Sander L. M. Diffusion-Limited Aggregation, A Kinetic Critical Phenomenom. *Physical Review Letters*. 47-19 1400, 1981.

[8] Tolman Susan, Meakin Paul. Off-lattice and hypercubic-lattice models for diffusion-limited aggregation in dimensionalities 2-8. *Physical Review A*. 40-1 428, 1989.

[9] Kaufman Henry, Vespignani Alessandro, Mandelbrot Benoit B. Woog Lionel. Parallel diffusion-limited aggregation. *Physical Review E*. 52-5 5602, 1995.

[10] Maldelbrot Benoit B. Angular Gaps in Radial Diffusion-Limited Aggregation: Two Fractal Dimensions and Nontransient Deviations from Linear Self-Similarity. *Physical Review Letters*. 88-5, 2002.

[11] Peitgen Heinz-Otto, Jürgens Hartnut, Saupe Dietmar. *Chaos and Fractals. New Frontiers of Science. Second Edition*. Springer. 2004.

[12] Deng Fang. The Study of Terrain Simulation Based on Fractal. *WSEAS Transactions on Computers*. Issue 1, Volume 8, January 2009.

[13] Menshutin A.Y. Schur Lev N. Test of multiscaling in a diffusion-limited-aggregation model using an off-lattice killing-free algorithm. *Physical Review E*. 73, 011407, 2006.

[14] gcc.gnu.org.

[15] IEEE. *The Open Group Base Specifications Issue 6*. IEEE Std 1003.1, 2004

[16] The Linux Kernel Archives. www.kernel.org.

[17] Friedman J.H., Bentley J.L., Finkel R.A. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, Vol. 3, No 3, pp 209-226. 1977.

[18] Amdahl Gene. Validity of the single processor approach to achieving large scale computing capabilities. *AFIPS spring joint computer conference*. 1967.

```
00 precondition: min_dist=infinity
01 recursive real function  VPDIST (node,min_dist)
02 /*Search for accurate or  approximate distance */
   /*between one point to a set of points */
03 value pointer node node
04 reference real min_dist
05 pointer node son, real d
06 if not ISNULL(node) then
07    if  DIST(p,CENTER(node)) > DIAGONAL(node)*2
          then
08    /*return an approximate distance */
09       if DIST(p,CENTER(node)) -DIAGONAL(node)/2
           < min_dist then
10          return DIST(p,CENTER(node))-
          DIAGONAL(node)/2
11       endif
12    else
13       if ISLEAF(node) then
14          /*if we have a leaf, do an exact search */
15          d:=min_dist
16          for each point in node do
17             if DIST(p,point)< d then d:=DIST(point)
18          enddo
19          if d< min_dist then return d endif
20       else
21       /*if we do not have a leaf, ask direction */
22          son := DIR_POINT(p, node)
23          /*now search recursively in the structure */
24          min_dist:= VPDIST(son,min_dist)
25          if INTERSECTS(CIRCLE(p,min_dist),
                      LEFT_SIB(son)) then
26             min_dist:=
          VPDIST(LEFT_SIB(son),min_dist)
27          endif
28          if INTERSECTS(CIRCLE(p,min_dist),
          RIGHT_SIB(son)) then
29             min_dist:=
          VPDIST(RIGHT_SIB(son),min_dist)
30          endif
31          if INTERSECTS(CIRCLE(p,min_dist),
                               OPP_SIB(son)) then
32             min_dist:= VPDIST(OPP_SIB(son),min_dist)
33          endif
34       endif
35    endif
36 endif
37 return  min_dist
```

Listing 1. Pseudo-code for the variable precision distance search algorithm.

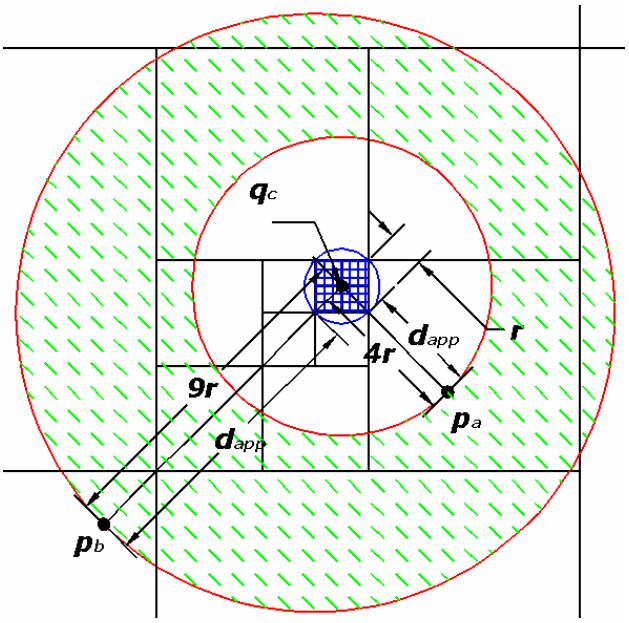


Figure 1. Region assigned to an approximate distance by inspecting the center q_c of a node in a quadtree. p_a and p_b show how close and how far a point can be when its distance to the cluster is approximately evaluated respect to q_c. d_{app} is the returned value.

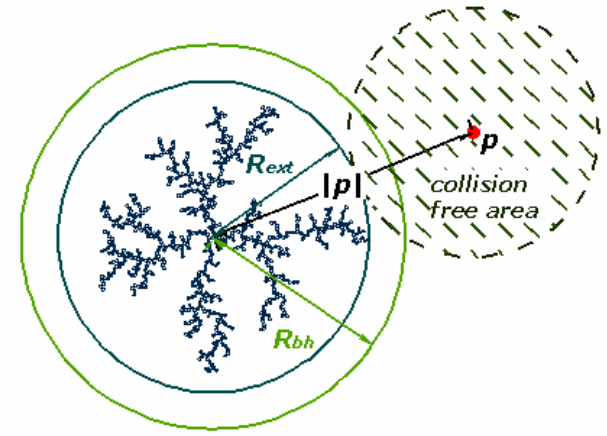
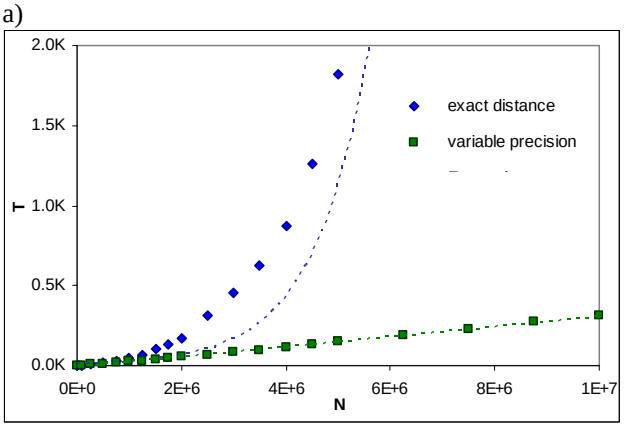


Figure 2. Length of the random jump for a particle *p* taken when is beyond R_{bh} from the center of the cluster. The shaded area does not contain any particle belonging to the cluster so it is safe for a particle to jump across it.

a)



b)

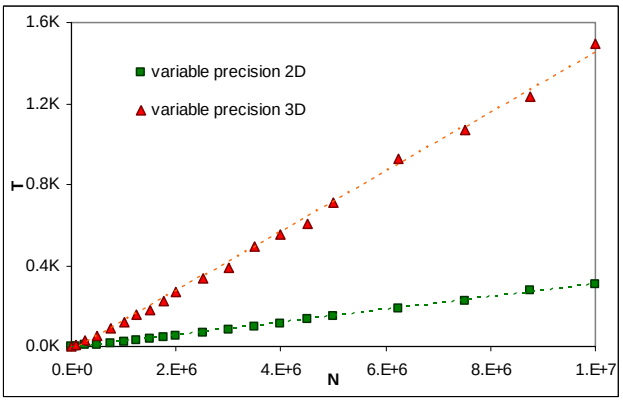
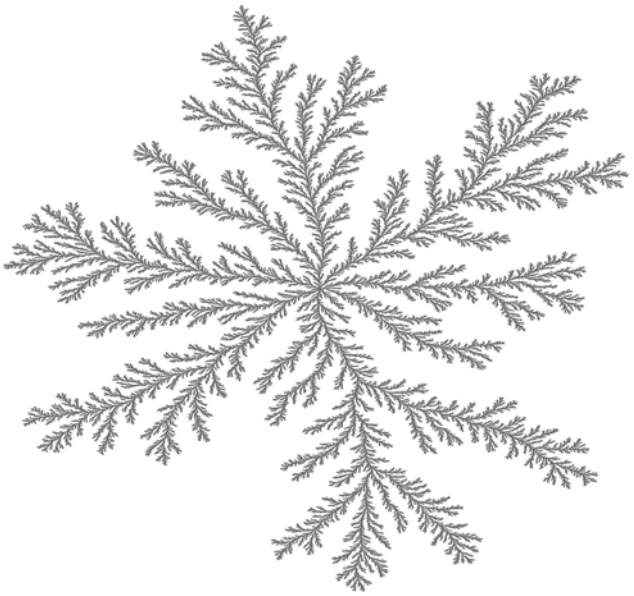


Figure 3. Processing time (in seconds) T required for the simulation of 2D and 3D DLA models of size N. The complexity reduction from an exponential to a linear trend can be observed in a) .In b) we have time required for 2D and 3D simulations using variable precision and exact distance search. Correlation coefficients are 0.9981 and 0.9976 respectively.

a)



b)

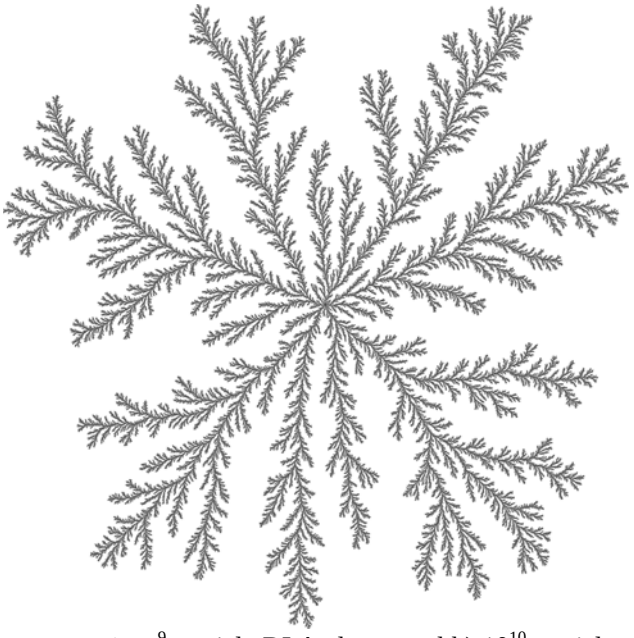


Figure 4. a) 10^9 particle DLA cluster and b) 10^{10} particle PDLA cluster. Here exterior radius are approximately $1.5x10^5$ and $7.5x10^5$ particle diameters respectively.

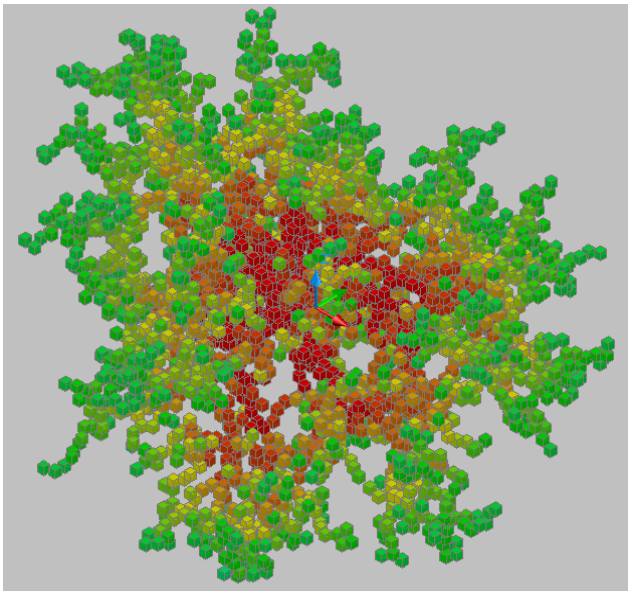


Figure 5. A tiny 3D-DLA cluster consisting of 3500 particles. Different shadings corresponds to the arrival sequence of the particles at the cluster.

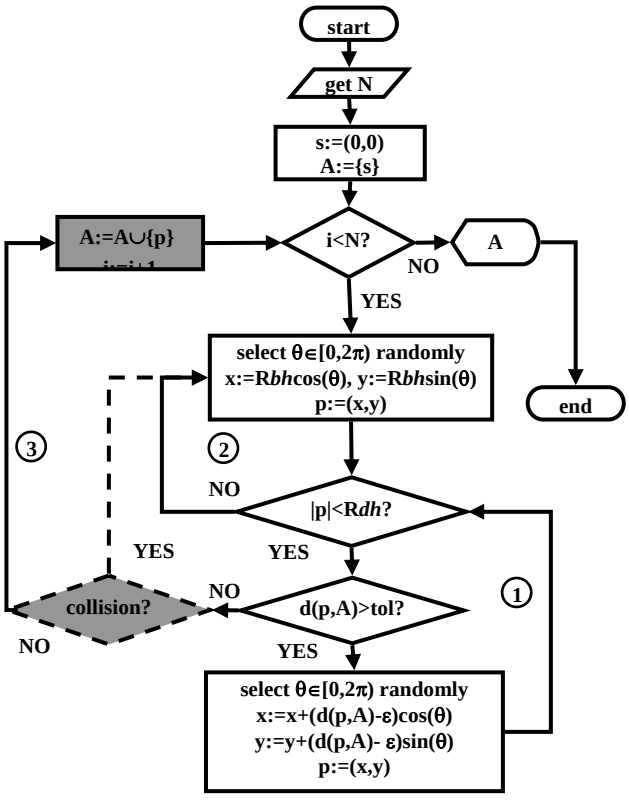


Figure 6. Flow diagram for off-lattice DLA and PDLA codification. Dashed lines indicates a test needed only for the PDLA model. Procedures in gray boxes show critical sections which need to be protected when using multiple processor.

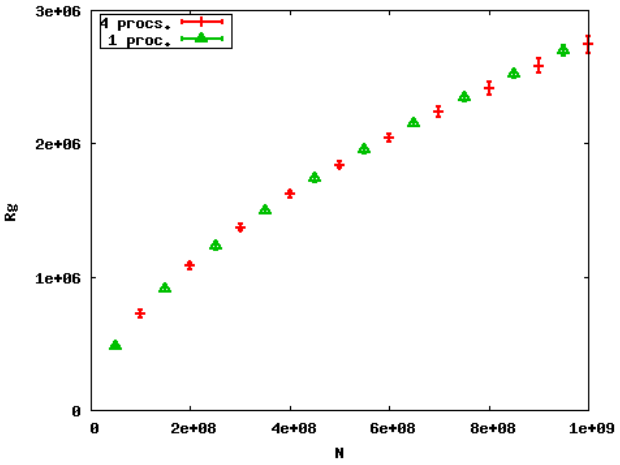


Figure 7. Radius of gyration R_g for clusters of size N generated with 1 and 4 processor. Error bars are scaled by a factor of 2 with the intention of making them visible.

DIR_POINT(*point*, *node*) Returns a pointer to the son of *node* that is in the same quadrant than *point* respect to the center of *node*.
CENTER(*node*) Returns the center point of the space covered by *node* in the quadtree.
DIAGONAL(*node*) Returns the length of the diagonal that goes through the area covered by *node* in the quadtree. Returns infinity if node is a null pointer.
DIST(*point1*,*point2*) Returns the Euclidean distance between *point1* and *point2*.
ISLEAF(*node*) Returns *true* if *node* is at the last level of the quadtree
CIRCLE(*p*, *r*) Returns a reference to a circle object with radius *r* and center at *p*. This is replaced by SPHERE (*p*, *r*) in three dimensions.
INTERSECTS(*circ*, *node*) Boolean function that returns *true* if the circle (or sphere) referenced by circ intersects with the squared area (or cubic volume) belonging to *node* in the quadtree
LEFT_SIB(*node*) RIGHT_SIB(*node*) These functions return a reference to left and right siblings of *node* in the same level of the quadtree hierarchy.
OPP_SIB (*node*) Returns a reference to the diagonally opposite sibling of *node* at the same the quadtree hierarchy.

Table 1. Description of functions used in the pseudo-code given in Listing 1.

N	***P***	***d***	***E***	***D***	σ
10^{10}	4	2	5	1.716542	0.05793
10^{9}	4	2	80	1.712472	0.01292
10^{9}	1	2	10	1.710590	0.00466
10^{9}	4	3	10	2.479468	0.01644
10^{8}	4	3	80	2.503168	0.01715
10^{8}	1	3	50	2.531181	0.00820

Table 2. Fractal dimension measurements on DLA and PDLA clusters of different size N. Here P indicates the number of processors. D and d represent the fractal and Euclidean dimensions respectively. σ indicates standard deviations for D.