14

General Index and Its Application in MD Simulations

Guangcai Zhang*, Aiguo Xu and Guo Lu
Institute of Applied Physics and Computational Mathematics, The People's Republic of China

1. Introduction

In the long-term practice, it is recognized that the properties of materials are not uniquely determined by their average chemical composition but also, to a large extent, influenced by their structures. The impurities and defects in metal will hinder the movement of free electrons and reduce their conduction, therefore, the thermal conductivity of alloy is significantly smaller than that of pure metal. The yield strength, fracture strength, fatigue toughness and other mechanical properties of metal are influenced by defects, such as dislocations, grain boundaries, micro voids and cracks. In weak external magnetic field, due to the existence of spontaneous magnetization within a small area, e.g. magnetic domains, ferromagnet shows strong magnetism. The bonding strength and density of crystalline phases dramatically influence the strength of ceramic. Due to the existence of independent molecules, linear structure (including the branched-chain structure) polymers are flexible, malleable, less hard and brittle, and can be dissolved in a solvent or be heated to melt. However, in three-dimensional polymers, as there are no independent molecules, they are hard and brittle, can swell but cannot be dissolved or melt, and are less flexible. In nematic liquid crystal, the rod-like molecules are arranged parallelly to each other, but their centres of gravity are in disorder. Under external force, molecules can flow easily along the longitudinal direction, and consequently have a considerable mobility. In smectic liquid crystal, the molecules align in a layered structure via lateral interaction of molecule and interaction of functional groups contained by molecules. Two-dimensional layers can slide between each other, but the flow perpendicular to layers is difficult.

With the change of external conditions, the microscopic structures of material may change and consequently alter the material macroscopic properties. For example, the hardness of eutectoid steel with 0.77% carbon content is about HRC15 after annealing, but up to HRC62 after quenching, because the structure of carbon steel differs after different heat treatments. Electric field can change the order of liquid crystal molecules; LCD is made by using this feature, and now has been widely used in everyday life. It is one major goal of material sciences and urgent need of engineering applications to quantitatively clarify the relationship between macroscopic behaviour and microscopic structure. This goal imposes

* Corresponding Author

the task of identifying and describing those lattice defects, including their collective static and dynamic behaviour, which are responsible for specific macroscopic properties.

In a narrow sense, the structure of materials refers to microstructure. Haaseen defined it as the totality of all thermodynamic non-equilibrium lattice defects on a space scale that ranges from angstroms (e.g. non-equilibrium foreign atoms) to meters (e.g. sample surface). Its temporal evolution ranges from picoseconds (dynamics of atoms) to years (corrosion, creep, fatigue) (Dierk, 1998). The generalized structure also includes the electronic structure of atoms and chemical structure of molecules in equilibrium states. According to length scale, the various levels of material structure can be roughly divided into nanoscopic, microscopic, mesoscopic, and macroscopic regimes, where the term nanoscopic refers to the atomic level, microscopic to the level smaller than the grain scale, mesoscopic to the grain scale, and macroscopic to the sample geometry. Of course, this subdivision is to a certain extent arbitrary, various alternative subdivisions are conceivable.

Due to the multilevel nature of material structures and the correlation between them, macroscopic properties of materials are overall performance of structures of each level. Take polymer material for example, small molecules with chemical reactivity and fixed structure are made of different atoms, and can gather into macro molecules by the polymerization reaction (addition polymerization or condensation). The polymer chains consist of several units of the same structure linked in a special sequence and spatial configuration under certain formation mechanism. As the single bond can rotate, polymer conformations with certain potential energy distribution can be formed by chains. The chains with certain conformation gather into polymer body through the second-force or hydrogen bonds. Under certain physical conditions, polymer itself or with other added substances (such as fillers, plasticizers, stabilizers, etc.) can form macroscopic aggregation structure made of a number of microstructures to become a performance polymer material by means of molding.

Currently, all kinds of microscopy techniques are used to observe, trace and analyze material structures. The microscopy techniques with matching resolution are selected according to the characteristic scale of various structures. For instance, optical microscopy is commonly used for micro-scale structures, and electron microscopy for nano-scale structures. Advances in experimental observation methods represented by all kinds of microscopy technologies make possible getting more and more accurate material structure information. With the rapid development of computer technology and materials science, computer simulation is widely used in the study on the relationship between microstructures and macroscopic properties of various materials, and greatly promotes the development of material science. As different simulation methods are only suitable for their own corresponding spatial and temporal scales, we need choose appropriate simulation schemes according to the length scales of structures involved in specific issues.

Molecular Dynamic (MD) simulation is a very powerful tool to solve many-body problems. As being able to provide detailed evolution images of microscopic particles over time, it is particularly suitable for studies on the evolution law and characteristics of material microstructure. Since it was firstly used to study phase transition for hard sphere regimes in 1957, MD simulation technique has been greatly improved. Now it has become an important method for studies on characteristics and evolution law of structure of metallic materials,

inorganic non-metallic materials and polymer materials. In the fields of nano-materials and bio-pharmaceuticals, molecular dynamics simulation is becoming an indispensable basic tool. In the studies on the large deformation of metal, molecular dynamic simulation plays a crucial role in explaining inverse Hall-Petch behaviour, nucleation and annihilation of dislocation involved in grain boundary and the relationship between shear band and dimple crack surface(Kumar et al., 2003). In the studies on protein folding, molecular dynamics is one of the most effective ways to simulate protein folding and unfolding. Via molecular dynamics simulation the transition state and change of energy in folding (or unfolding) process can be clearly analysed.

Structure analysis is the core issue of material simulation. Over the years, many defect identification methods have been proposed. For metal defects distinction, there are the excess energy method, coordination number method, centro-symmetry parameter method (Kelchner et al., 1998), Ackland's bond-angle method (Ackland & Jones, 2006), and bond-pair method (Faken & Jonsson, 1994), etc.

As we know, the complex computations between particles cannot be avoided in any elaborate defect identification methods. Since the computation complexity of traditional methods are of high order, the computation time needed will dramatically increase with growing the system size. Therefore, it is necessary to design new data structure and indexing algorithm to reduce computation complexity. The computation complexity of defects identification methods will be greatly reduced by using background grid and linked list. The background grid index, together with the linked list data structure, is suitable for management of uniform distributed points, and it has been widely applied in computation and analysis of many simulations. The analysis of complex structure in non-uniform system refers not only to points, but also to lines, surfaces and bodies, and their distributions are usually non-uniform. The background grid index cannot meet the needs for managing these objects, but the multi-level division of space is effective way to manage complex data. The SHT (space hierarchy tree), a newly proposed data structure, is a powerful dynamical management framework of any complex object in any dimensional space. Index of objects with complex structure can be created based on SHT, and corresponding fast searching methods could be designed to meet various search needs.

As the elements managed by SHT are abstract objects, such as geometry objects (points, lines, surfaces and bodies) in three-dimensional space and characteristic regions in phase space, the general index based on SHT data structure can be used in various application areas. In this chapter, index methods in MD simulation will be introduced, including the background grid index and general index based on SHT, and then with the emphasis on several applications of SHT general index in computational geometry (shortest path problem and Delaunay division) and characteristic structure analysis (cluster construction, defects identification, and interface construction).

2. General index

In many programs, such as post processing of MD, discrete elements simulation, computational geometry, smart recognition, spatial objects (i.e., points, lines, surfaces, bodies, structures, etc.) are used to construct complex structures in the system. Without index of spatial objects, the quantity of computation for searching object is very high. If a

system contains N objects, the computation complexity related to two objects is N^2 and N^3 for three objects. If the total number of objects is more than 10^4, the computation complexity is not acceptable. Therefore, effective storage and fast search of objects are necessary, and consequently the index of spatial objects should be established. Currently, there are three kinds of spatial indexes: background grid, tree and linked list. These indexing methods mainly focus on points and neighbour search. In this chapter, an SHT general index is presented, which is suitable for management of any objects in any dimensional space. Fast searching algorithms meeting any given requirements are proposed based on SHT. In traditional programming, intuitive idea does not lead to intuitive algorithm. However, if following the idea of SHT to write codes, intuitive idea can directly lead to intuitive algorithm, so that it's beneficial to programming. Taking into account the integrity of this description, background grid index is firstly introduced, and then we introduce the SHT data structure and fast searching methods.

2.1 Background grid index

For points and small objects with uniform sizes, if their spatial distribution is not extremely scatted, the background grid index is suitable for managing. This indexing method is widely used in molecular dynamics, Monte Carlo method, smooth particle hydrodynamics, material point method and discrete element method to search neighbour objects. For this case, small object can be viewed as a point located at the centre of the object or its feature point.

The idea of background grid index (Michael, 2007) is as follows: set a box to contain all objects according to their locations; divide the box into smaller grids with a certain size, then put objects into appropriate grids according to their locations. Hence, an index for spatial objects is created. When searching for objects, several grids are firstly selected according to searching conditions, and then search in selected grids to get the needed objects. In figure 1, small black rectangles stand for objects and thin lines are for background grid. When search for objects in a given circle, what we need to search through is not all the grids but only those covered by the circle.

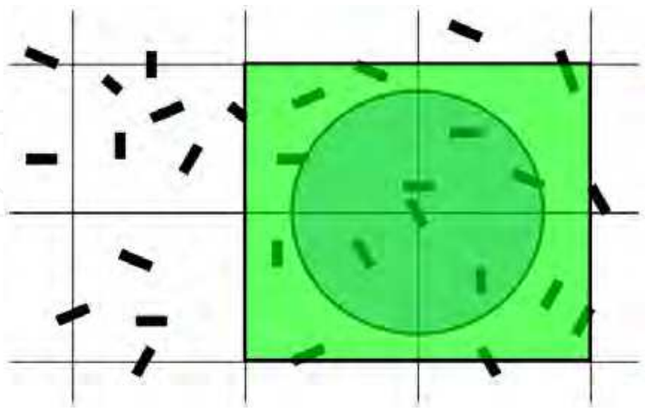


Fig. 1. Scheme for the background grid index of objects.

For creating background grid index, one way to store the objects within each grid is to use array. Thus an array with possible maximum size must be firstly defined. In this way, a large quantity of memory may be wasted. So, the array method is not suitable for the case

with objects nonuniformly distributed. A good alternative is to use the linked list to store such kinds of objects. Figure 2 is the scheme for linked list of objects in background grid (two-dimensional space). The indexing steps are as follows:

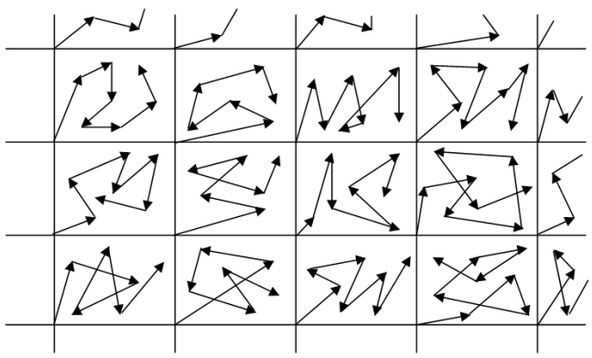


Fig. 2. Scheme for the linked list of objects in background grid.

1. Set the grid size Δ according to the search requirements;
2. Check all objects to get system size $(x_0, x_1)\times(y_0, y_1)$;
3. Calculate the grid array size (I, J), where $I = \left[\frac{x_1 - x_0}{\Delta}\right] + 1$, $J = \left[\frac{y_1 - y_0}{\Delta}\right] + 1$, allocate the grid array dynamically;
4. For each object A, according to its location (x_A, y_A) to get the grid (i, j) containing it, where $i = \left[\frac{x_A - x_0}{\Delta}\right] + 1$ and $j = \left[\frac{y_A - y_0}{\Delta}\right] + 1$. Insert object A into the top of linked list of grid (i, j). In figure 3, the red point stands for grid $g(i, j)$, blue point stands for object A, $g(i,j)\rightarrow obj$ stands for the object pointed to by grid $g(i, j)$. The two steps are as follows:

$$\begin{aligned} A \rightarrow next &= g(i,j) \rightarrow obj \\ g(i,j) \rightarrow obj &= A \end{aligned} \tag{1}$$

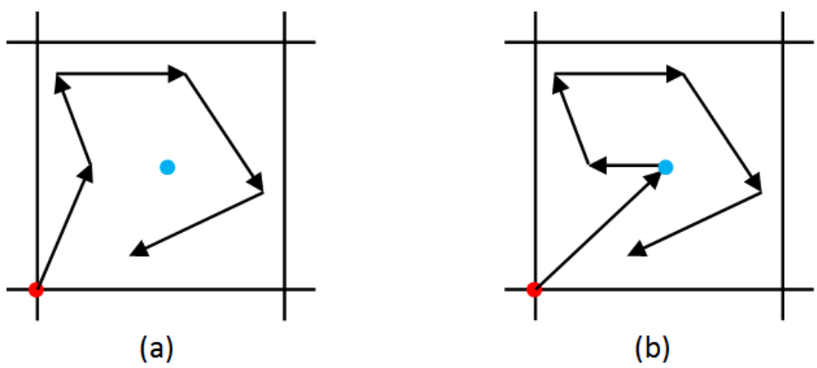


Fig. 3. Scheme for inserting an object into a linked list: (a) before inserting; (b) after inserting.

The algorithm for searching objects in a given region is as follows: (i) search for grids crossing with the given region; (ii) search for objects within those grids.

In background grid index, *correlated grid* refers to the grid crossing with the given region, *number of object included* is the number of objects within each grid. The computational complexity of background grid index is related to the number of correlated grid (N_{gird}) and the average number of object included (N_{obj}), it is of order O($N_{gird}N_{obj}$). The larger the grid size, the less the number of correlated grid and the more the average number of objects included, and vice versa. The memory needed increases with decreasing grid size. A proper grid size has to be chosen to reduce computational complexity. Usually, the selected average number α of objects included ranges from 3 to 5. The grid size can be gained by calculating the average density of object. Such as, in two-dimensional space, the grid size is $\Delta = \left(\frac{\alpha (x_1 - x_0)(y_1 - y_0)}{N} \right)^{1/2}$, and it is $\Delta = \left(\frac{\alpha (x_1 - x_0)(y_1 - y_0)(z_1 - z_0)}{N} \right)^{1/3}$ in three-dimensional space.

The data structure of background grid index is very simple. If the size and spatial distributions of objects are uniform, the calculation based on this index is fast. A few disadvantages of this index scheme should also be pointed out: (i) the background grid cannot be dynamically created, it must be recreated with local dynamical adjustment; (ii) for the size of object is not contained by the index, if the size of objects is not uniform, it is hard to support index related to sizes. (iii) When the dimension is high or the spatial distribution is not uniform, the needed memory is too large.

2.2 SHT (Space Hierarchy Tree) and general fast search

Currently, there is no general scheme for managing different types of objects, there is also no universal way to quickly search for objects under flexible requirements. Different problems require different designs. A large number of complex operations are unavoidable in the corresponding coding procedure. This leads to lots of drawbacks, such as design difficulties, without university, long codes and hard to maintain.

When indexing discrete spatial points, background grid linked list structure can be used. For spatial data, tree structure can provide convenient index. Examples are referred to BSP (Binary space partitioning) tree, K-D tree (short for k-dimensional tree) and octree (de Berg, 2008; Donald, 1998). The BSP and K-D trees have their own specific index methods and are not suitable for general index. For example, the BSP tree is suitable for identification of inner or outer region of polyhedra.

The data structure of object determines its indexing, for instance, the index of discrete points by using background grid is limited to neighbor point index. A better data structure design can implement general index. A data structure meeting any indexing requirements must be appropriate to describe the locations and sizes of objects, adapted to the dynamic changes in the data, and its structure has to be standardized. The octree can manage three-dimensional points. It can also manage other kinds of spatial objects.

In this section, a new data structure, SHT, extended from octree is proposed to unify the management of any objects in any-dimensional space, and two general search methods (conditional search and minimum search) are presented to implement search for object meeting any given requirement. The computation complexities of two search schemes are both logN. By using SHT and corresponding fast searching methods, programming becomes easy.

2.2.1 SHT management structure

In three-dimensional space, SHT data structure is similar to octree. Go a further step, for a system in n-dimensional space, a n-dimensional cube (a line segment in one-dimensional space, a rectangle in two-dimensional space, and a cube in three-dimensional space) is designed to contain the system; then divide this cube in each dimension into two parts to form 2^n sub-cubes; only retain the cubes with objects inside; continue to decompose each cube until the required resolution is reached; put the objects (points, lines, surfaces, bodies) into the appropriate cube according to their locations and sizes; retained cubes are connected together, according to their belonging relationships, to form a 'spatial hierarchy tree'. Up to now, an index for a set of spatial objects has been established, needed objects can be quickly found via the index. Figure 4-a (Figure 4-b) is a scheme for the SHT management structure of two-dimensional (three-dimensional) discrete points.

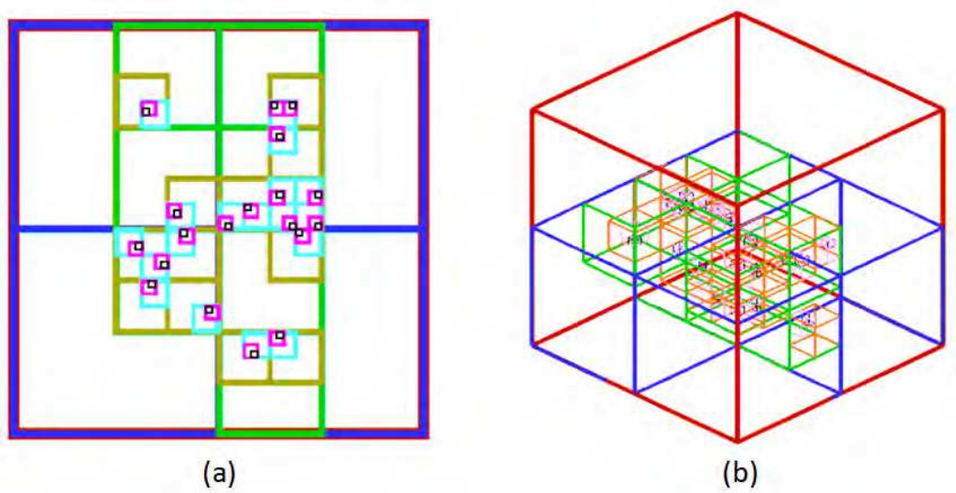


Fig. 4. Scheme for management region of SHT of discrete points. (a) Two-dimensional points; (b) three-dimensional points.

This SHT data structure contains the properties of directed tree. Each cube is named a 'branch'. Its child-cubes are named 'sub-branches' and its parent-cube is called the 'trunk'. The largest or the top-level cube is named the 'root' and the smallest or bottom-level cubes are called 'leaves'. The process for linking an object with a branch is named 'putting into' and the opposite process is called 'cutting down'. The SHT is different from the octree in two aspects. Firstly, it can be used in any-dimensional space and the number of sub-branches is arbitrary. Secondly, objects can be put into not only leaves but also any other branches, and the number of objects is arbitrary. Due to the uncertainty of the number of 'sub-branches' in a 'branch', 'branches' sharing the same 'trunk' are linked as a list; Similarly, 'objects' belonging to the same 'branch' are also linked as a list. Figure 5 is the scheme for object management by SHT. For the convenience of description, from now on, unless specifically pointed out, we do not distinct between 'branch' and its corresponding cube region, such as, the 'center of branch' is also referred to the geometrical center of the cube, the 'size of branch' is also the cube size, and the 'branch corner' is also the corner of the cube, etc. Meanwhile, tree are named according to the managed objects, such as, tree created to manage points is called 'point tree', and to triangle is 'triangle tree', etc.

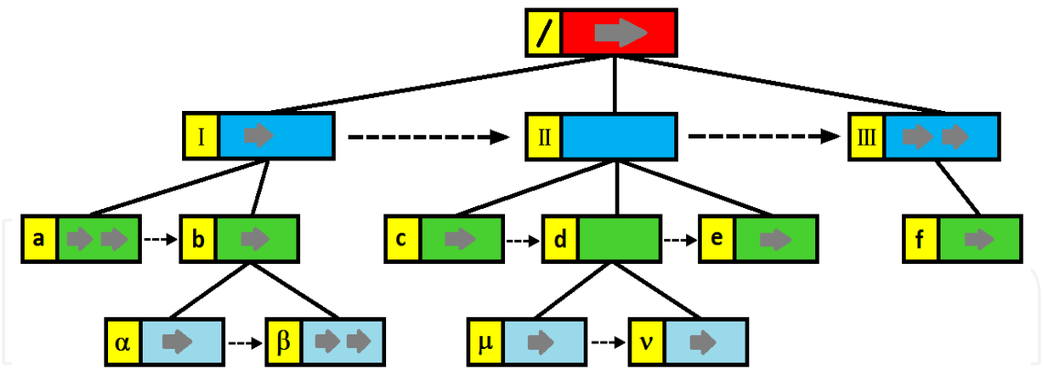


Fig. 5. Scheme for object management by SHT. The rectangle in each row stands for branch, horizontal grey arrow stands for object, and vertical black arrow stands for the list which connects the sub-branches belonging to a same branch.

In practical applications, the number of objects may be variable. Therefore, the SHT is constructed dynamically. The dynamic management procedure of SHT includes three basic operations: (I) establishment of a tree, (II) adding a new object to a tree, (III) removing an object from a tree.

(I) The establishment of a 'tree' from an object is to establish a minimum 'branch' to contain the object. The idea of algorithm is thus: Enlarge the size of the given minimum cube until the object can be put into. The typical procedure consists of two steps: (i) Get the known minimum resolution 'σ', i.e. the smallest edge length of cubes. Use the center of an object as the geometrical center of the cube to create a 'branch'; (ii) Check whether or not the branch can contain the object. If yes, the branch size is proper, and put the object into it; if not, double the branch scale and go back to step (ii) to continue. Up to this step, a 'tree' with only one 'branch' which contains one 'object' has just been established. Figure 6 shows the construction process of the original tree, where a triangular object in two-dimensional space is used as an example.

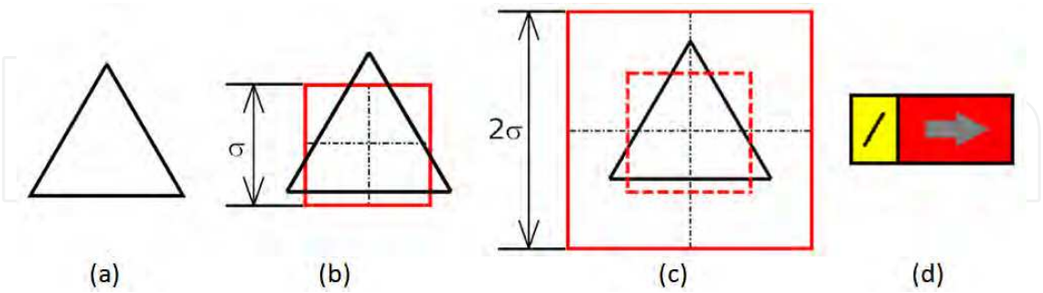


Fig. 6. Scheme for the establishment of the original tree. (a) triangle object; (b) create an original branch with minimum resolution 'σ'; (c) double the scale of original branch until the triangle can be contained; (d) established triangle tree, where '/' is root ID, grey arrow stand for the triangle.

(II) The idea of adding an object to a tree is thus: Enlarge the tree to contain the object and then add the object to an appropriate branch. The algorithm for adding a new object B to the tree is as below: (i) Adjust the root. (a) Check whether or not the object B can be contained

by the current root r. If yes, end this step and go to step (ii). If not, take the center of root r as a reference point to calculate the quadrant where the center of object B is located, and then create trunk s of root r, link root r to trunk s and set trunk s as root. (b) Go back to step (a). (ii) Place object B. (c) Set the root as current branch; (d) Check whether or not the sub-branch of current branch, with respect to quadrant B, contains object B. If not, place the object into current branch and break; if yes, enter the sub-branch (if the sub-branch does not exist, create it) and take it as the current branch. (e) Go back to step (d).

Figure 7 is the scheme for adding a triangle to an existing tree, where A and B stand for triangle objects. O1 is the geometrical center of root of the original tree and O2 is the corresponding center of the enlarged root. The root is '/', the second-level sub-branches are 'I', 'II', 'III' and 'IV', 'a' is the third-level sub-branch. B0 is the geometrical center of object B. In figure 7(c), the rectangle in each row stands for the corresponding branch (symbol in the left part is the branch, and in right part is the object).

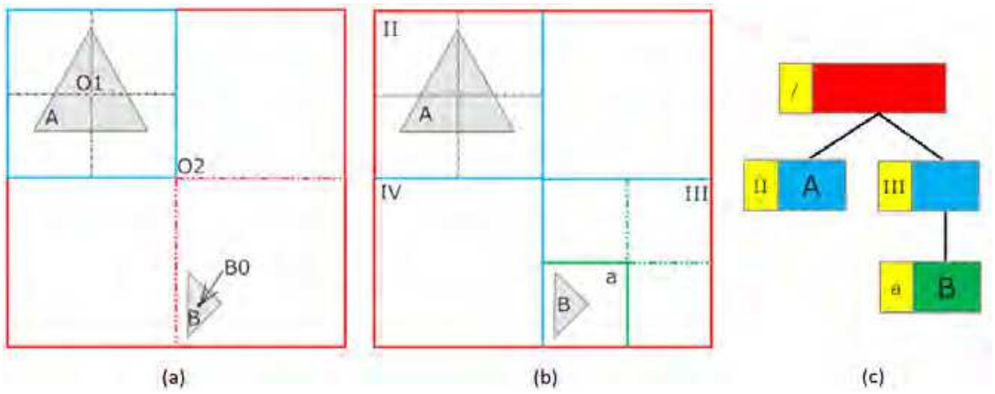


Fig. 7. Scheme for adding objects to the tree. (a) Generation of new root; (b) placing an object; (c) SHT corresponding to (b).

(III) The algorithm for removing an object from the tree is as follows: (i) According to the link, pick up the branch containing the object. Remove the object from the corresponding object-list, and then set this branch as current branch; (ii) Check whether or not current branch contains other objects and sub-branches. If not, remove this branch, enter its trunk and set the trunk as new current branch, and then go back to step (ii) to continue this process until all the useless branches are eliminated. Figure 8 is the scheme for removing an object from a tree.

In the dynamical algorithm of SHT, except for adding a sub-branch or trunk of a branch, other operations have nothing to do with space dimension. We can use four functions to describe the operations related to space dimension and to the location and size of object in this process: (i) Create an appropriate branch to contain the current object, (ii) Check whether or not a branch contains the current object, (iii) Take the current branch as reference point, and then create a trunk of current branch in the quadrant where the object is located, (iv) Take the current branch as reference point, and then create sub-branch of current branch in the quadrant where the object is located. The computer memory required by the SHT is approximately equal to kNlogN, where N is the number of objects. It is independent of the spatial dimension. When the spatial dimension is higher or spatial objects are have a highly

scattered distribution, the SHT can save a large quantity of memory compared with the background grid method. In addition, because SHT is dynamically constructed, the size of the system can dynamically increase or decrease with the addition or deletion of objects. This is a second obvious advantage over the traditional background grid method.

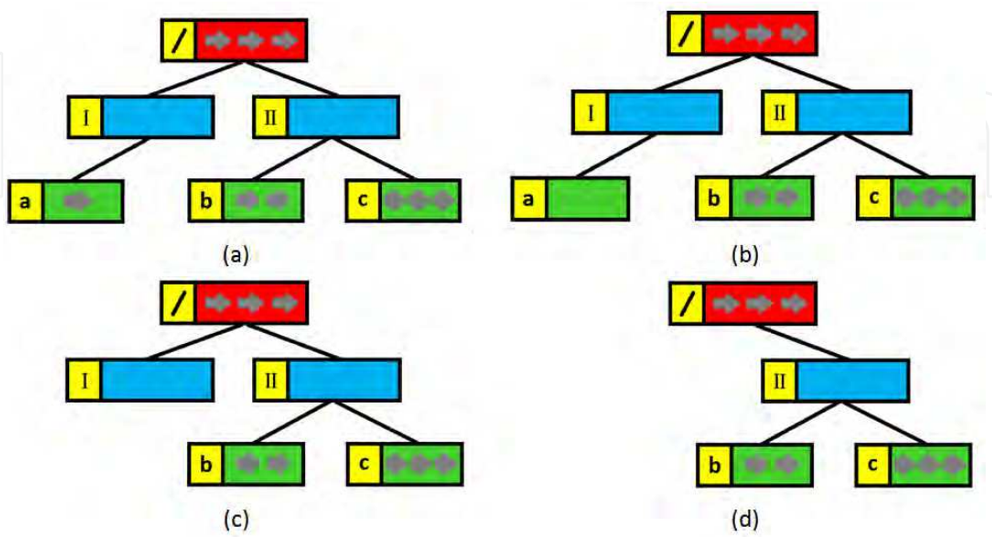


Fig. 8. Scheme for removing an object from a tree. (a) pick up the branch containing the object, in this figure, it is branch 'a'; (b) remove the object from branch 'a'; (c) remove branch 'a'; (d) remove branch 'I', which contains no more objects and sub-branches.

2.2.2 Fast searching algorithms based on SHT

In practical application, we generally need a fast search of objects satisfying certain requirements. The computational complexity of an ergodic search is N. It is clearly not practical when dealing with a huge number of objects. For such cases, we need to develop fast searching algorithms. As the objects managed by SHT can be located, fast searchers with computational complexity logN can be easily created. The basic idea is as below: It is unnecessary to search the objects directly, but rather check branches. Skip those branches without objects under consideration. Thus, searching is limited to a substantially smaller range. Depending on requirements of applications, two fast searching algorithms are presented: conditional search and minimum search. The goal of conditional search is to search for objects meeting certain conditions. For example, find objects in a given area. The goal of minimum search is to search for an object whose function value is minimum. For example, find the nearest object to a fixed point.

2.2.2.1 Conditional search

Conditional search is to search for objects meeting given conditions among the objects hanged up to the SHT. The idea of algorithm is as follows: Check whether or not a branch contains objects meeting given conditions. If not, do not search the branch (including all sub-branches of it and corresponding objects). For example, if searching for all objects in a given circle, check whether or not a branch crosses with the given circle. If not, skip this branch. In this way, the searching is limited to branches crossed with the given circle.

For the convenience of description, we define *candidate branch* as the branch which may contain the objects meeting the given condition during search procedure. Conditional search is implemented using the stack structure. The steps are as follows: (i) Push the root into a stack A; (2) Check whether or not the stack A is empty. If yes, end the search; if not, pop out a branch b from the stack A; (ii) Check whether or not each sub-branch of b is a candidate branch. If yes, push it into stack A; (iv) Check whether or not the object in b satisfies the given conditions; pick out the required objects; (v) Go back to step (ii).

Figure 9 shows the given circle and spatial division for managing planar triangles. Figure 10 is the scheme for the SHT corresponding to Figure 9, where '/' stands for the root. The process to find out objects in the given circle is shown in Figure 11. In the example, apply the procedure given above, we will finally find object A and B is in the given circle.

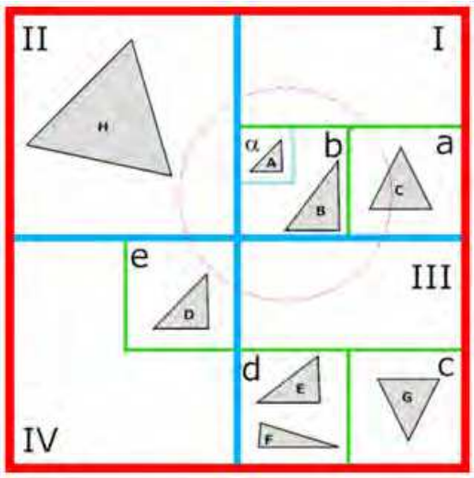


Fig. 9. Distribution of planar triangles and the corresponding spatial division.

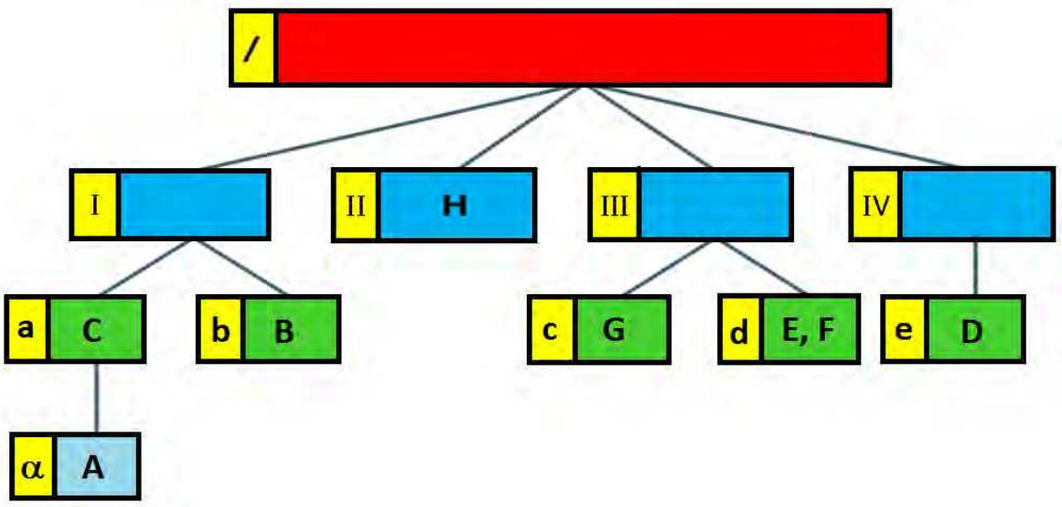


Fig. 10. SHT corresponding to Figure 9.

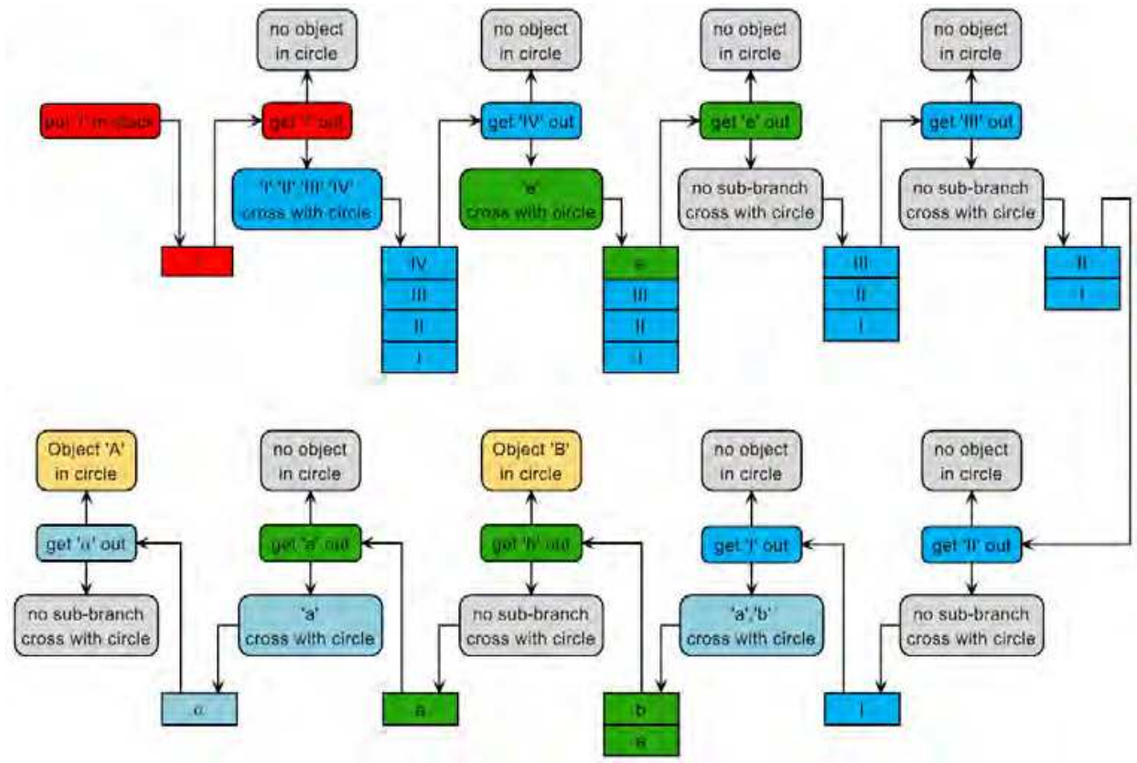


Fig. 11. Flowchart for the fast search of objects in a circular area.

In the searching, other operations have nothing to do with space dimension and type of object except for two operations, (i)checking whether or not an object is the needed one, or (ii) checking whether or not an branch is a candidate branch. So, the algorithm can be built on the abstract level. The conditional search is implemented by providing a *conditional function* and an *identification function*. The c*onditional function* is used to check whether or not an object is needed. Assuming *condition(o)* is the conditional function, the argument *o* is object and the function value is bool number. The *identification function* is used to assess whether or not a branch is a candidate branch. Assuming *maycontain(b)* is identification function, the argument *b* is branch and the function value is bool number. With defining the above two functions, conditional searching meeting any given conditions can be easily implemented.

The conditional search can search for a collection of objects. With the search marks, the actual search process is done step by step. In each step, only one object can be gained. The computation complexity of conditional search is kL, where L~ln N is total level of SHT. The efficiency of the conditional searching algorithm depends on the identification function. The more efficient the identification function, the smaller the k and the faster the searching procedure. If identification function value is always true, k tends to infinity and this searching algorithm goes back to an ergodic browser. However, the computation complexity of identification function influences the k value. Since the computation for sphere regions is more efficient than for cube ones, in complex conditional searching algorithms, circum-spheres of a cube are extensively used to reduce the quantity of computations.

2.2.2.2 Minimum search

In programming related to spatial objects, we often need to find objects meeting some given extreme condition. For example, to find a point with the largest z component from a set of three-dimensional points, or to search a point with the nearest distance to a given point, or to search a sphere closest to a plane, etc. Such searches can be attributed to the minimum searching problem. For spatial objects, each one can be assigned a function value related to its location and size such that the minimum search is to find the object with the minimum function value.

Fast searching can be created based on the SHT. The idea is as follows: Design a function to assess the range of the function value of all objects. Some branches can be excluded by comparing the ranges of the function value of different branches. For example, in a region, $\sum$ is a set of discrete points, we need to search the nearest points to a given point A. The fast searching is not to compute the distance between each point in $\sum$ and point A, but assess the range of distance between 'branches' and point A to excluding unnecessary searching in branches (including the point in it and its sub-branches) with longer distance.

For convenience of description, we define a few concepts.(i) Range of a branch: It means the range covering all the values of the given function for objects in this branch. (ii) B-R-branch: It is a new branch data structure composed of the branch itself and the range of it. (iii) Candidate B-R-branch: It is the B-R-branch which may be checked in the following procedure. It may contain objects whose functional values are minimum. In the minimum searching procedure, we must keep enough candidate B-R-branches. Some of them may be dynamically added or removed according to the need. In order to accelerate the searching speed, the candidate B-R-branches should be linked as a list. According to the above definition, each B-R-branch has a range. So, each B-R-branch has a lower limit to its range. The B-R-branches in the list are arranged in such a way that their lower limits subsequently increase. Obviously, the B-R-branch with the smallest lower limit is placed at the head of the list. (iv) Candidate object: the object with the minimum values during the searching procedure. It is initialized as null, and at the end of searching, it is the object with the minimum value. (v) Candidate value: function value of candidate object.

The minimum searching algorithm is as follows: (i) The root and its range are merged as a B-R-branch; Add the B-R-branch to a candidate list named L; The candidate value V is set as positive infinity; The candidate object is set as null. (ii) Pick out a B-R- branch, for example, B, from the head of candidate list L; Check the values of its objects. If the value of an object O is smaller than V, then replace V with this value; in the meantime, set object O as a candidate object. Remove the B-R-branches whose lower limit values are greater than V from the list L. (iii) Construct a B-R-branch Z for each sub-branch of B. If the minimum value of Z is larger than V, cancel Z; If the maximum value of Z is smaller than the minimum value of the B-R-branch C in L, then all the B-R-branches behind C are removed from L; If the minimum value of Z is larger than the maximum value of a B-R-branch in L, cancel Z; Otherwise, insert Z into list L according to its lower limit. (iv) Repeat steps (ii) and (iii) until the candidate list L is empty. The final candidate object is the required one.

As an application example of the proposed minimum search algorithm, we consider a case to find the nearest point to a fixed point A. Figure 12 shows the distribution of planar points and spatial division. Figure 13 is the corresponding SHT. Suppose point A in Figure 12 is the given fixed point. To seek the nearest point to it, the range of the branch is calculated by a sphere evaluation method. Figure 14 shows the flowchat.

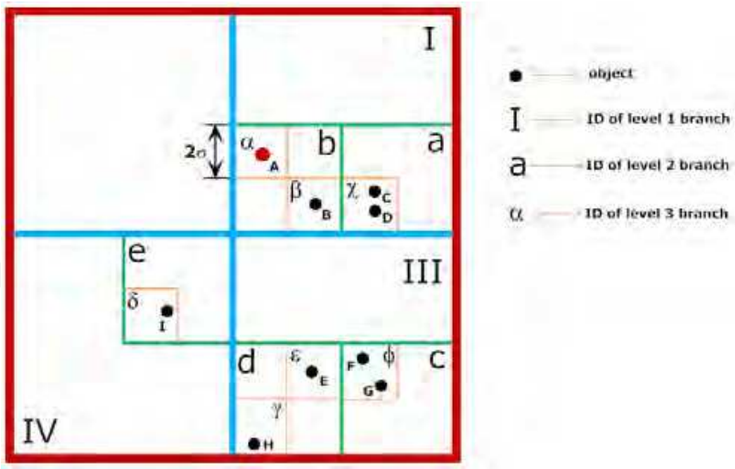


Fig. 12. Planar point distribution and corresponding spatial division.

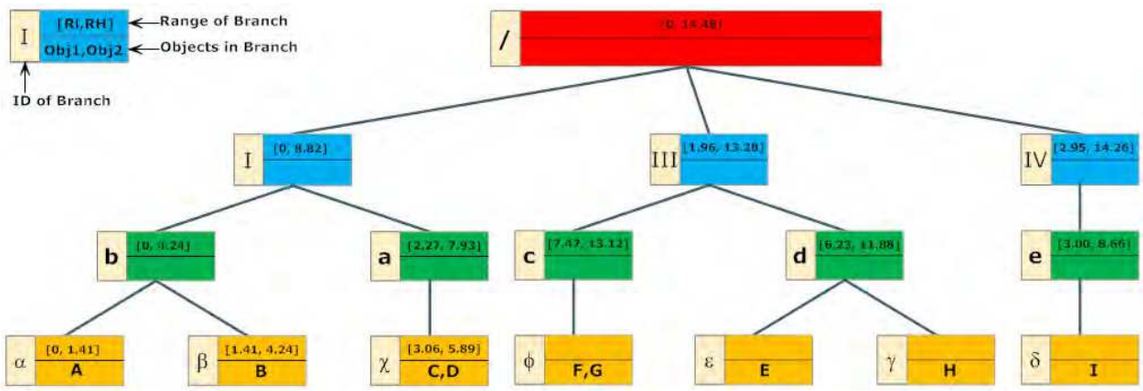


Fig. 13. SHT corresponding to Figure 12.

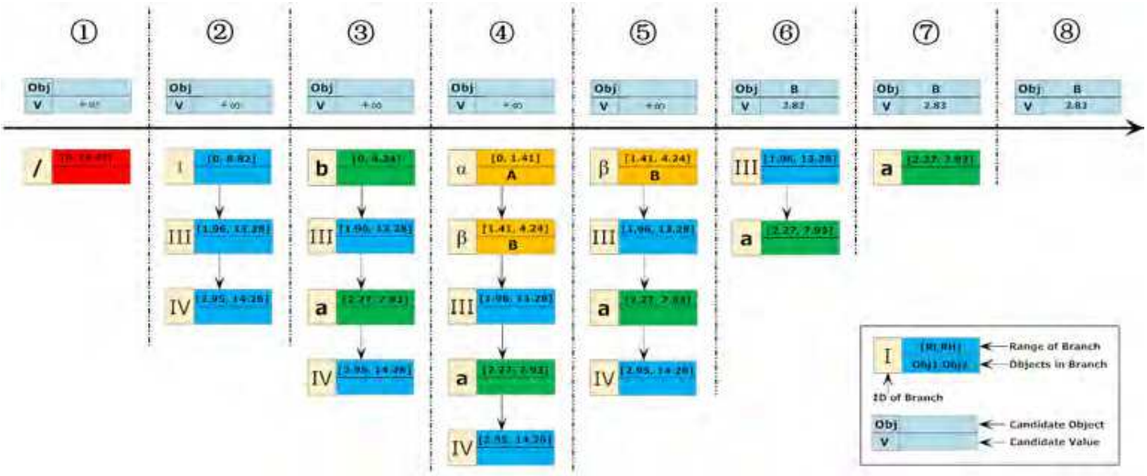


Fig. 14. Scheme for the fast search of the nearest point to a given fixed one.

In minimum search, except for computing the function value of an object and the range of a branch, other operations have nothing to do with space dimension and type of object, so that the algorithm can be built on abstract level. Similar to conditional search, the minimum search is implemented by providing a *value-finding function* and *range-evaluation function*. The *value-finding function* is used to calculate the function value of an object. Assuming *value(o)* is *value-finding function*, the argument *o* is an object and the function value is a real number. The *range-evaluation function* is used to compute the range of a branch. Assuming $M(b)$ is the upper limit and $m(b)$ is the lower limit of the range, the argument of function is branch b and the function value is a real number. With defining the above two functions, various minimum searches can be easily implemented.

The computation complexity of minimum searching is kL, where L~ln N is total level of SHT. The efficiency of the minimum searching algorithm depends on the range-valuation function. The smaller the range given by the range-evaluation function, the smaller the k and the faster the searching procedure. The worst range-evaluation function gives a range from $-\infty$ to $+\infty$. In such a case, the searching algorithm goes back to the ergodic browser. In the case with a large quantity of objects, one should use a good range-evaluation function to reduce the number of objects to be searched. However, a precise range-evaluation generally needs a large quantity of computations, which also decreases the global efficiency. We should find a balance between the two sides. Since the computation for sphere regions is more efficient than for cube ones, in complex minimum searching algorithms, circum-spheres of a cube are extensively used to evaluate the range of a branch.

During conditional searching and minimum searching, we often need to screen some objects. For example, in the searching for the nearest point to a point A, if A is not screened, the searching result is just A and it's not what we need. In the algorithm implementation, a screen function is designed by breaking the corresponding connections in the tree, each call of screen function can screen an object, and a recover function is been design to recover all screened objects.

3. Applications

SHT framework provides a general index of set of spatial objects and effective management of data. Most problems in specific applications can be solved with SHT. In this chapter, object with physical properties is named as 'matter element'. Examples are referred to molecule or molecular clusters with certain energy, mass, momentum and angular momentum, particles with specific density and phase, grains with orientation and finite element with density and energy. For the majority of applications, problem can be summarized as constructing new sets of matter element meeting requirements from the existing one. In this section, the applications of general index in shortest path problem, space division, cluster construction, defect atom identification and interface construction are presented.

3.1 Shortest path problem

The shortest path problem (Moore, 1959) is a classical problem in graph theory. It is to find a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. The shortest path not only refers to the shortest geographical distance, but also can be extended to other metrics, such as time, cost, and line

capacity and so on. The selection and implementation of the shortest path algorithm is the basis of channel route design. It is the research focus of computer science and address information science. Many network-related problems can be incorporated into the category of the shortest path problem. Due to the effective integration of classic graph theory and the continuous development and improvement of computer data structure and algorithm, new shortest path algorithms are emerging with time.

The postman shortest path problem is as follows: Given N cities, one needs to find a shortest loop that the postman can go through all these cities. This is a typical NP problem, because it needs to compare all of the loops to find the optimal path. As a simple application example, the approximate solution is given, it is to find a shorter loop without intersection between its segments to connect all discrete points, and this approximate solution is called a labyrinth. The algorithm of the approximate solution is as follows: add point to the loop one by one in such a way that a previous segment is replaced by two where the newly added point becomes the vertex in between and connecting two previous vertexes, at the same time keep the loop length as shortest as possible.Figure 15 is the schematic for adding a point to loop.

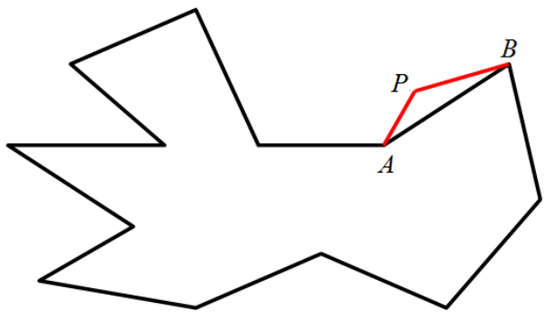


Fig. 15. The change of a loop after adding a point P. The segment AB becomes two segments AP and PB.

The preparatory work for algorithm based on SHT is to design fast searchers. In this section, a design, including three minimum searchers and a conditional searcher, is proposed, where one value-finding function and two range-evaluation functions are defined for each minimum searcher and one conditional function and one identification function are defined for the conditional searcher.

(I) Minimum searcher MS1: search for the nearest point to a fixed point $\mathbf{r}_0$ in a two-dimensional point tree. The value-finding function is

$$value(p) = \left|\mathbf{r}_p - \mathbf{r}_0\right| \tag{2}$$

The two range-evaluation functions are as follows:

$$\begin{aligned} M(b) &= \left|\mathbf{c}_b - \mathbf{r}_0\right| + \sqrt{2}d_b \\ m(b) &= \max(\left|\mathbf{c}_b - \mathbf{r}_0\right| - \sqrt{2}d_b, 0) \end{aligned} \tag{3}$$

where $\mathbf{r}_p$ is the coordinate of point p, $\mathbf{c}_b$ the coordinate of the center of branch b, $\sqrt{2}d_b$ the circumradius of branch b, $M(b)$ the upper limit and $m(b)$ the lower limit.

(II) Minimum searcher MS2: In a two-dimensional point tree, search for a point with the shortest fold-line connecting the two vertexes of a given segment *AB*. The value-finding function is

$$value(p) = \left|\mathbf{r}_p - \mathbf{r}_A\right| + \left|\mathbf{r}_p - \mathbf{r}_B\right| \tag{4}$$

The circumcircle of branch *b* is used to assess the range of *b*. The range-evaluation functions are as follows:

$$\begin{aligned} M(b) &= \begin{cases} 2\sqrt{2}d_b & A,B \in C \\ 2\sqrt{2}d_b + \left|\mathbf{r}_B - \mathbf{c}_b\right| & A \in C, B \notin C \\ 2\sqrt{2}d_b + \left|\mathbf{r}_A - \mathbf{c}_b\right| & B \in C, A \notin C \\ \left|\mathbf{r}_A - \mathbf{c}_b\right| + \left|\mathbf{r}_B - \mathbf{c}_b\right| & A,B \notin C \end{cases} \\ m(b) &= \begin{cases} 0 & A \in C \text{ or } B \in C \\ \left|\mathbf{r}_A - \mathbf{c}_b\right| + \left|\mathbf{r}_B - \mathbf{c}_b\right| & A,B \notin C \end{cases} \end{aligned} \tag{5}$$

where *C* is the circumcircle of branch *b*.

(III) Minimum searcher MS3: Given a point *P*, in two-dimensional segment tree, search for a segment *AB* with the minimum value of the length of fold-line *APB* minus the length of segment *AB*. The value-finding function is

$$value(p) = \left|\mathbf{r}_p - \mathbf{r}_A\right| + \left|\mathbf{r}_p - \mathbf{r}_B\right| - \left|\mathbf{r}_A - \mathbf{r}_B\right| \tag{6}$$

Ditto, the circumcircle *C* is used for assessment. The range-evaluation functions are as follows:

$$\begin{aligned} M(b) &= 2(2\sqrt{2}d_b + \left|\mathbf{r}_P - \mathbf{c}_b\right|) \\ m(b) &= \begin{cases} 0 & P \in C \\ \max(\left|\mathbf{r}_P - \mathbf{c}_b\right| - 3\sqrt{2}d_b, 0)\left|\mathbf{r}_A - \mathbf{c}_b\right| + \left|\mathbf{r}_B - \mathbf{c}_b\right| & P \notin C \end{cases} \end{aligned} \tag{7}$$

(IV) Conditional searcher CS1: Given a segment *AB*, in two-dimensional segment tree, search for a segment *ab* crossed with segment *AB*. The conditional function is as follows:

$$condition(ab) = \begin{cases} true & 0 \le x \le 1, 0 \le y \le 1 \\ false & else \end{cases} \tag{8}$$

where x and y are obtain from solving the equation, $x\mathbf{r}_A+(1-x)\mathbf{r}_B=y\mathbf{r}_a+(1-y)\mathbf{r}_b$. The identification function is as follows:

$$maycontain(b) = \begin{cases} true & AB \cap b \neq \phi \\ false & else \end{cases} \tag{9}$$

Up to now, the preparatory work has been finished. The labyrinth construction algorithm is as follows:(I) Initialization: Construct point tree *tp* from given planar discrete points, cut

down a point P_0, search for the nearest point P_1 to point P_0 (P_0 is screened) with MS1 searcher in tree *tp*, search for point P_2 (P_0 and P_1 are screened) to make the length of fold-line $P_0P_2P_1$ shortest with MS2 searcher in tree *tp*, generate segment P_0P_1 and construct a segment tree *tl* from it, generate segment P_2P_0 and P_1P_2, put them on the tree *tl*. (II)Connection construction: (i) Cut down a point *P* from tree *tp* and empty stack *s*. (ii) If *P* is null, exit. (iii) Search for a segment *L* (all segments in stack *s* are screened) in tree *tl* with MS3 searcher to make the increased length shortest by adding point *P*. (iv) Search for a segment L_1 crossed with the segment *L* with CS1 searcher. (v) If L_1 is null, cut down *L* from tree *tl*, generate two segments by connecting the two vertexes of *L* with point *P* and put them on the tree *tl*, go back to step (i), and if the stack *s* is not empty, then push *L* into *s* and go back to step (iii). Figure 16 shows the labyrinth constructed by connecting 40000 random points.

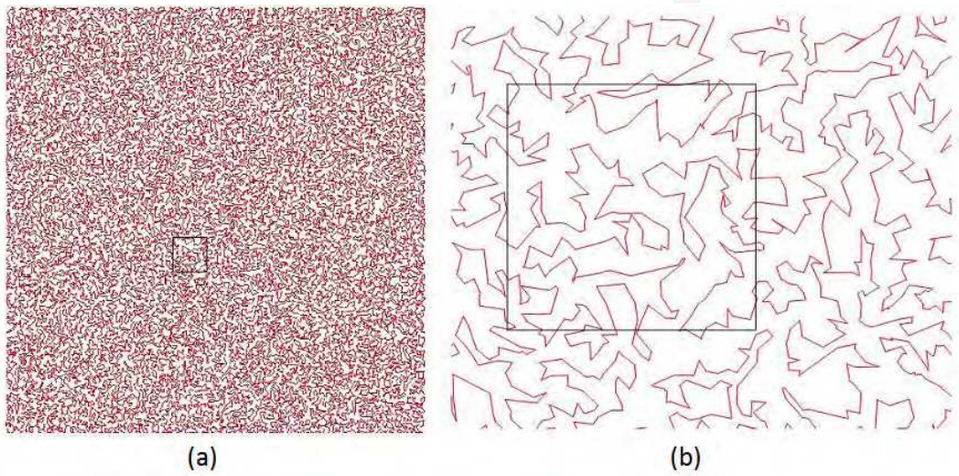


Fig. 16. Labyrinth constructed by connecting 40000 random points. Fig. (b) shows the enlarged portion chosen by the small black rectangle in Fig. (a).

3.2 Fast constructing algorithm for Delaunay division

Since the good mathematical characteristics of Delaunay division (de Berg, 2008), it is widely applied in many areas, such as the pre-processing of three-dimensional finite element method, medical visualization, geographical information systems and surface reconstruction. For designing a subdivision algorithm, an important part is to create a mesh with low complexity and good performance. Today, there are lots of algorithms to the construction of Delaunay tetrahedrons in three-dimensional space or Delaunay triangles in two-dimensional space. The complexities of most algorithms are involved with the searching procedures. Here, we propose an algorithm based on the SHT. The algorithm is simple and intuitive. It is convenient to extend to higher-dimensional space.

It's necessary to use two new data or objects, i.e. 'extended-tetrahedron' and 'extended-triangle' in the two-dimensional and three-dimensional algorithms. The 'extended-tetrahedron' is a combination of tetrahedron and its circumsphere, so its data structure contains that of tetrahedron and its circumsphere, and its center and size are those of the circumsphere. The 'extended-triangle' is a combination of triangle and its circumcircle, so its data structure contains that of triangle and its circumcircle, and its center and size are those of the circumcircle.

The algorithm for constructing Delaunay division from discrete points is as follow: when a new point is added to the formed Delaunay division structure, the division is adjusted to meet the condition for Delaunay division. Only tetrahedra whose circumspheres include the added point need to adjust, and they are named to-be-adjusted-polyhedron. The adjustment is to construct new tetrahedron by connecting each outer surface of the to-be-adjusted-polyhedron with the added point. Figure 17 shows the steps for adding a two-dimensional point and re-dividing the space, the red points stands for the newly added point P, the green triangles in figure 7(a) are to-be-adjusted- triangles, the red segments in figure 7(b) are retained boundary segments, the blue segments in figure 7(c) are segments connecting point *P* with vertexes of boundary. In the three-dimensional case, we need only to replace the triangle with a tetrahedron, replace the line with a triangular face, and replace the extended-triangle with an extended-tetrahedron.

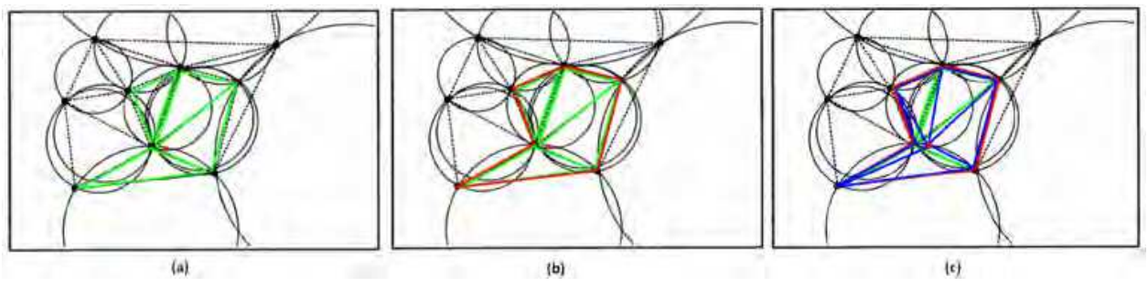


Fig. 17. Three steps to add a new point to a two-dimensional Delaunay division. (a) Finding the triangles whose circumcircle contains the newly added point *P*; (b) Removing the internal lines of these triangles, retaining the external ones; (c) Connecting each left line with point *P* to form new triangles.

The preparatory work for algorithm is to design fast searchers. In this section, a design, including two conditional searchers, is proposed.

(I) Conditional searcher CS1: Given a fixed point $\mathbf{r}_0$, search for an 'extended-tetrahedron' *CT* containing point $\mathbf{r}_0$. The conditional function is as follows:

$$condition(CT) = \begin{cases} true & |\mathbf{c}_{CT} - \mathbf{r}_0| < R_{CT} \\ false & else \end{cases} \tag{10}$$

The circumcircle of branch *b* is used to identify, the center of b is $\mathbf{c}_b$ and the radius is $\sqrt{3}d_b$.
The identification function is as follows:

$$maycontain(CT) = \begin{cases} true & |\mathbf{c}_{CT} - \mathbf{r}_0| < \sqrt{3}d_b \\ false & else \end{cases} \tag{11}$$

(II) Conditional searcher CS2: Given three points P_1, P_2 and P_3, search for an extended-tetrahedron *CT* where P_1, P_2 and P_3 are three of its vertexes. The conditional function is as follow:

$$condition(CT) = \begin{cases} true & \text{three vertexes are } P_1, P_2 \text{ and } P_3 \\ false & else \end{cases} \tag{12}$$

The identification function is as follows:

$$maycontain(CT) = \begin{cases} true & P_1, P_2, P_3 \in b \\ false & else \end{cases} \tag{13}$$

In three-dimensional space, the algorithm for the constructing Delaunay tetrahedron from given discrete points is as follows: (I)Initialization: Generate a sufficiently large tetrahedron to contain all discrete points, record the center and radius of its circumsphere, form an 'extended-tetrahedron' and construct an extended-tetrahedron tree *t*; (II) Construction of Delaunay tetrahedron: (i) Pick out a discrete point *P*, search for the extended-tetrahedra whose circumspheres contain *P* in tree *t* with searcher CS1, remove these extended-tetrahedra from tree *t*; put the removed tetrahedra together to form a set named *Q*; (ii) Add every surface of each tetrahedron in *Q* to SHT *S* which is a tree for the external triangular interfaces, search for the surfaces appearing twice with searcher CS2 and remove them because they are interfaces. Triangles in *S* constitute the external surface of *Q*; (iii) Pick out each face of *S*, together with point *P*, to construct a new tetrahedron; add the newly formed extended-tetrahedra to tree *t*; (iv) Go back to step (i) until all points are used out.

Up to this step, the group of tetrahedra contained by tree *t* is just the Delaunay tetrahedra division. Figure 18 shows the Delaunay triangle division of 20000 randomly distributed two-dimensional points. Figure 19 shows the Delaunay tetrahedron division of 20000 randomly distributed points in a three-dimensional sphere.

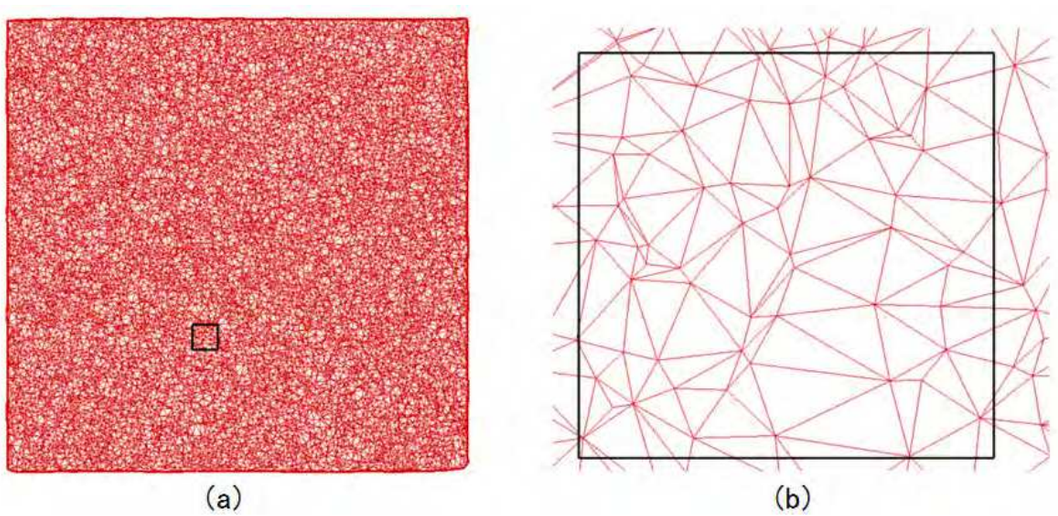


Fig. 18. Delaunay division constructed from randomly distributed discrete points in a two-dimensional square area [0,4]×[0,4]. (b) is the enlarged picture of the portion in the small black rectangle in (a).

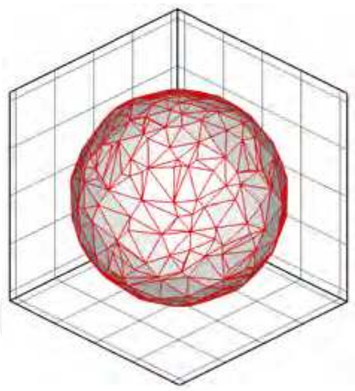


Fig. 19. Delaunay division constructed from randomly distributed discrete points in a three-dimensional spherical region.

The algorithm can be easily extended to n-dimensional space. We need only to replace the tetrahedron with an n-simplex and replace the triangle with an (n-1)-simplex. The circumsphere of the n-simplex constructed from n+1 point, $\{\mathbf{r}_1, \mathbf{r}_2,\cdots,\mathbf{r}_{n+1},\}$ in n-dimensional space is used in the algorithm. The formula to calculate the center of the circumsphere of the n-simplex is $c_i=A_i/B$, where

$$A_i = \begin{vmatrix} 1 & -2\mathbf{r}_1 & \mathbf{r}_1^2 \\ 1 & -2\mathbf{r}_2 & \mathbf{r}_2^2 \\ \cdots & \cdots & \cdots \\ 1 & -2\mathbf{r}_{n+1} & \mathbf{r}_{n+1}^2 \\ 0 & \mathbf{e}_i & 0 \end{vmatrix}, \quad B = \begin{vmatrix} 1 & -2\mathbf{r}_1 & \mathbf{r}_1^2 \\ 1 & -2\mathbf{r}_2 & \mathbf{r}_2^2 \\ \cdots & \cdots & \cdots \\ 1 & -2\mathbf{r}_{n+1} & \mathbf{r}_{n+1}^2 \\ 0 & \mathbf{0} & 1 \end{vmatrix}, \tag{14}$$

and $\mathbf{e}_i$ is the unit vector of the i-th direction. The circumsphere radius is $\sqrt{(\mathbf{c}-\mathbf{r}_i)^2}$.

3.3 Cluster construction and analysis method

Complex configuration and dynamic physical fields are ubiquitous in weapon-physics, astrophysics, plasma-physics, and material-physics. Those structures and their evolutions are characteristic properties of the corresponding physical systems. For example, the interface instability makes significant constraints on the design of an inertial confinement fusion (ICF) device (Ament, 2008), shock waves and jet-flows in high energy physics are common phenomena (de Vriesl et al, 2008), distributions of clouds and nebulae are very concerned issues of astrophysics (Bowle et al., 2009; Hernquist, 1988; Makino, 1990), clusters and filaments occur in the interaction of high-power lasers and plasmas (Hidaka et al., 2008), and structures of dislocation bands determine the material softening in plastic deformation of metals (Nogaret et al., 2008). These structures are also keys to understand the multi-scale physical processes. Laws on small-scales determine the growth, the change and the interactions of stable structures on larger-scales. Description of the evolution of stable structures provides a constitutive relation for larger-scale modeling. Because of the lack of periodicity, symmetry, spatial uniformity or pronounced correlation, the identification and characterization of these structures have been challenging for years.

Existing methods for analyzing complex configurations and dynamic fields include the linear analysis of small perturbations of the background uniform field, characteristic analysis of simple spatial distributions of physical fields, etc. These methods are lacking in a quantitative description of characteristics of the physical domain. For example, the size, the shape, the topology, the circulation and the integral of related physical quantities. Therefore, it is difficult to trace the evolution of the characteristic region or the background. For example, the laws of growth and decline, or the exchange between them.

The difficulties in characteristic analysis are twofold. The first is how to define the characteristic region. The second is how to describe it. The former involves the control equations of the physical system. The latter is related to recovering the geometric structure from discrete points. In recent years, cluster analysis techniques (Kotsiantis & Pintelas, 2004; Fan et al., 2008) in data mining have found extensive applications in identification and the testing of laws of targets. They mainly concern the schemes for data classification. Physicists are concerned more about the nature underlying these structures. Recovering characteristic domains can be attributed to the construction of spatial geometry. The key point is how to connect the related discrete points. The Delaunay grid (Chazelle et al., 2002; Clarkson & Varadarajan 2007) has an excellent spatial neighboring relationship. In this chapter, the Delaunay triangle or tetrahedron is used as the fundamental geometrical element.

For the discrete points in space, there is no strict cluster structure. If the discrete points are considered as objects, such as a molecular ball, lattice or grid, then, the objects can be connected to form clusters. The average size of these assumed objects is the revolution of clusters to be constructed with discrete points. The construction of clusters is very simple. A cluster is formed by connecting all points whose distance in between is less than the revolution length.

After the construction of the Delaunay division for given set of discrete points, remove the lines whose lengths are greater than the revolution length. The remaining spatial structure may have various dimensions. According to connectivity, the structures that are not connected to each other can be decomposed into different clusters. Each cluster may also have structures with various dimensions. For example, a structure consisted of two triangles with a common side, or a structure formed by a triangle and a tetrahedron, etc. In physical problems, the structures with high-dimensional measures play a major role in describing the system. Generally, we need only to analyze clusters with the maximum dimensions.

The cluster construction algorithm consists of three parts. Preparation part: (i) Construct Delaunay tetrahedra from given discrete points. The corresponding SHT is notated as *t*. (ii) Remove the tetrahedrons whose lengths are greater than the given resolution from *t*. Single cluster construction part: (iii) Remove tetrahedron T from *t* if such a T still exists. Create a new cluster named C. Initialize the body tree C->*t* and face tree C->*s* as null. Add T to the body tree C->*t*. Add each of the triangle faces to a triangle tree named *i*. (iv) Pick out a triangle face S from *i*. Search tetrahedron Y containing face S from *t*. If found, add Y to tree C->*t* and add all faces of Y to tree *i*. Two faces with opposite directions will annihilate if they meet each other during the adding procedure. If none are found, add S to the tree C->*s*. (v) Repeat step (iv) until the tree *i* becomes null. Construct all the clusters: (vi) Add the constructed cluster C to a tree for clusters named *c*. Repeat the process of constructing single clusters, and add the new cluster to *c* until *t* becomes null.

The algorithm for adding a face S to the tree *i* is as follows. Search and check if a face with the opposite direction of S exists in the tree *i*. If exists, remove it from the tree *i*. If does not exist, add S to the tree *i*.

Up to now, all the constructed clusters are put to the cluster tree *c*. For each cluster C in the tree *c*, all tetrahedron elements are placed on the body tree C->*t*, all the surface triangles are placed on the tree for faces C->*s*. Figures 20 and 21 show respectively the clusters constructed with random points in two-dimensional and three-dimensional space.

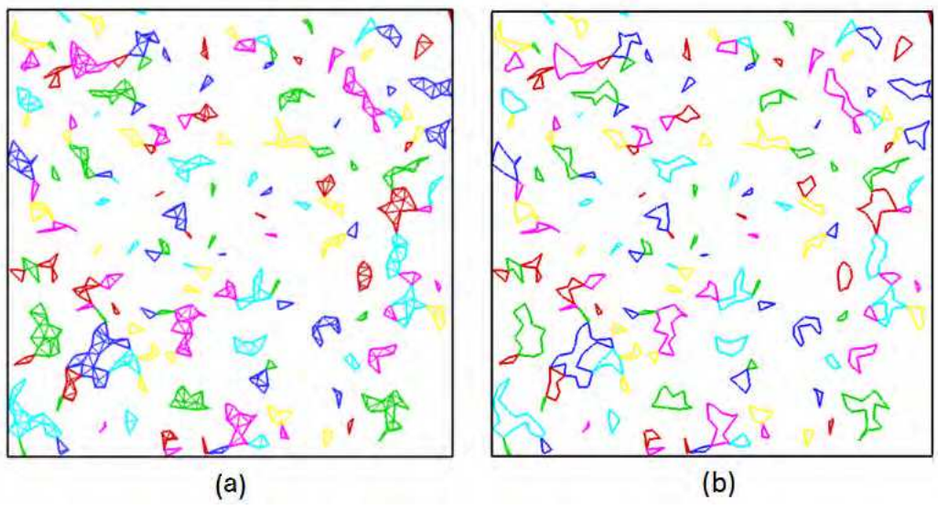


Fig. 20. Cluster structure formed from 1000 random discrete points in a two-dimensional square area [0,1]×[0,1]. (a) A cluster; (b) the corresponding cluster boundary.

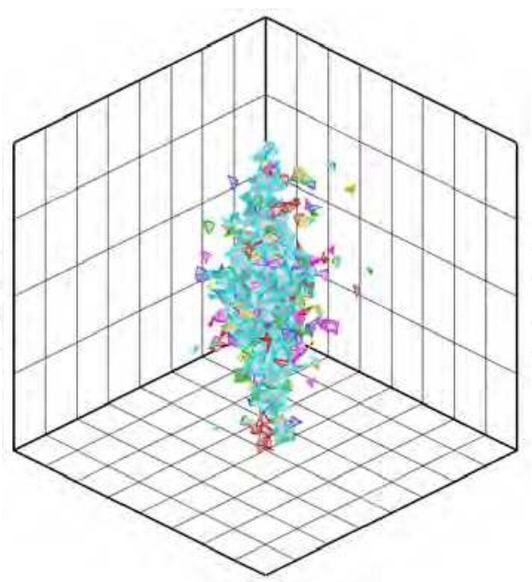


Fig. 21. Cluster structure formed from 5000 three-dimensional random discrete points.

The algorithm is also applicable to n-dimensional discrete points. We need only to replace the tetrahedron with an n-simplex and replace the triangular surface with an (n-1)-simplex.

For space with a dimension higher than three, the number of neighboring points and the connectivity, as well as the number of n-simplex, grow rapidly with the dimension. So, the required memory increases quickly. The Delaunay division can be constructed partition by partition. The main skill in this algorithm is that the space is partitioned according to the main branches of SHT, and points in each partition are added sequentially. After the completion of adding all points in a partition, we need to delete the n-simplex that satisfies two conditions: (i) its external circumsphere is in the completed partition, (ii) at least one side is longer than the given resolution.

3.4 Identification methods of defect atoms

The knowledge about the world is gradually deepened. Before the 19th century, the most prominent theoretical models primarily describe the movement of object. In the 19th century, two important theoretical models are proposed: one is field, another is uniform system. In the 20th century, the outstanding recognition model is about regular structure, represented by energy bond theory and phonon theory on studying crystal structure. Due to the enhanced computing power and increased awareness in the latter half of the 20th century, cognitive models on non-periodic structure and non-equilibrium emerge.

The primary recognition is as follows: During the system evolution, spatial structures of various scales appear, and the evolution of these structures determines the evolution laws of the system, when the system parameter arrives at the critical one, structures of different scales show an overall correlation with each other, and become indistinguishable. The typical structure characteristic is self-similar and renormalization theory can easily handle this singularity.

In the studies on non-periodic structure and non-equilibrium, the phase transition theory and dislocation theory are representatives, and nearly all phenomena can be explained by nonlinear theory. It indicates that the macroscopic properties of non-equilibrium system do not depend on its fast processes, but on the evolution of more stable structure. With different scales of concern, the system structure is not same.

During the technology development with gradually reduced time and space scales, scientific understanding on some processes, which cannot be studies or are unclear before, are gradually obtained, i.e. microscopic mechanism of the fracture process. Meanwhile we've got the multi-scale cognitive model: The overall properties of system are determined by the evolution of large-scale structures, small-scale structures and fast processes determine the large-scale structure and slow processes. In researches, different methods and theories are developed according to structures of different scales. The properties of small-scale structure provide model parameters and constitutive relationships for system with large-scale structures. For the studies on strongly coupled system which is also non-uniform and non-equilibrium, the key issue is to understand the properties and evolution laws of structures.

In particle simulation and 2D or 3D simulations of complex physical systems, how to effectively analyze the spatial and dynamical characteristics the system is the key to understand physical laws. There two problems: how to identify the stable structures existing in the system and how to compute them. For instance, in molecular dynamical simulation studies on the mechanical properties of metal, plastic deformation, phase transitions, and damage processed closely refer to defect structures, such as dislocations, stacking faults,

grain boundaries and interfaces. The emergence of those defect structures indicates the change stages of materials, and the evolution of them determines material properties. As defect structures are collection of arranged atoms, they can be appropriately identified with proper analysis methods. When face a huge number of atom coordinates, the key issue of physical analysis is how to recognize various defect structures.

3.4.1 Excess energy method and centro-symmetry parameter method

According to the physical quantities, structure symmetry or local topological connections, defect atoms can be distinguished by a few corresponding methods, for example, the excess energy method, centro-symmetry parameter method (CSP) (Kelchner et al., 1998) and bond-pair analysis (BPA) method (Faken & Jonsson, 1994).

In excess energy method, the atoms with excess energy are selected as defects atoms. Because the lattice equilibrium positions are stable positions for atoms, the potential energies of atoms deviating from their stable positions are higher. This method depends on the physical quantities of particles, so the output data from MD simulation must be completed. However, in many cases, such as, in phase transition represented by symmetric double-well energy function, energy cannot be used to distinguish different structures.

In CSP method, the geometrical symmetry of the collection of nearest atoms of an atom is used to identify defect atoms. All atoms of perfect crystal are in the geometrical center of its nearest atoms, but the defect atoms are not. Therefore, an order parameter is defined as follows:

$$s = \left| \sum_{i \in neighbour} (\mathbf{r}_i - \mathbf{r}_0) \right| \tag{15}$$

Atoms whose order parameter s is greater than a critical value s_c are defect atoms. In the case of strong temperature perturbation, the result of CSP method is not correct, because random thermal motion reduces the lattice symmetry and the order parameter of perfect lattice becomes greater.

The nearest atoms of a specific atom are the atoms within a given sphere region whose center is the specific atom. The radius of the given sphere must be greater than the distance of nearest atoms in perfect crystal and less than the distance of second nearest atoms. In fcc and bcc crystals, as the second nearest distance is $\sqrt{2}$ times of the nearest distance, the sphere center can be given as $(1+\sqrt{2})/2 \approx 1.2$ times as the nearest distance to resist a certain degree of randomness. In the larger deformation of crystal, as the lattice constant alters, the given sphere radius needs to be adjusted. In more complicate cases, as the lattice constant is not known in advance, a better way is first to compute radial distribution function (RDF) and then set the distance corresponding to the first peak of RDF as lattice constant of perfect crystal.

During the computation procedure of radial distribution function and order parameter algorithm, as atoms in given region need to be searched, an index of atoms must be constructed. Since the distribution of atoms is uniform, the background grid index can be used.

3.4.2 Bond-pair analysis method

The CSP and excess energy methods can distinguish defect atoms, but can not easily identify types of defects atoms. The bond-pair analysis (BPA) based on local topological connections can more accurately identify atom type. The idea of BPA is as follows: a bond type is marked in terms of the connections among atoms bonding with the two atoms composing the bond, and an atom type is marked in terms of all bonds of itself.

The 'bond' is defined as the connection between two atoms whose distance is less than a given value R (bonding distance). For convenience, the name of 'bond' proposed here is same as the one used in chemistry, but their meaning is different. The concept of the proposed bond doesn't contain any meaning of quantum chemistry where it indicates overlap between the electric wave functions. The bonding distance is often set as 1.2 times as the nearest distance in perfect lattice. In the case of unknown lattice constant, the lattice constant is usually set as the distance of the first peak of RDF, so to some extent, the topological analysis can tolerate random disturbance.

3.4.2.1 Bond-type identification

For a given bond L, the bond-type of L is determined by the indirect bonding feature of both atoms of L. Pick out all atoms which bond with both the two atoms of L. The bonding feature of picked atoms is used to identify bond-type of L. Assume the two atoms of bond L are A and B. The collection of atoms bonding with both A and B is c. Specifically, a bond-type is marked by a three-digit number. The first digit number is the atom number in c, the second is the number of bond among atoms in c, and the third is the largest coordination number of atoms in c.

Figure 22 shows an example for bond-type identification. The bond-type of bond is 443, where, from left to right, the first number, 4, denotes the number of common neighbor of atoms A and B (i.e. H, I, C, and D), the second number, 4, means the bond number of the common neighbor atoms (i.e. HI, DI, HD, and CD), the third number, 3, indicates the largest coordination number (the coordination number of atom D, i.e. CD, HD, and DI).

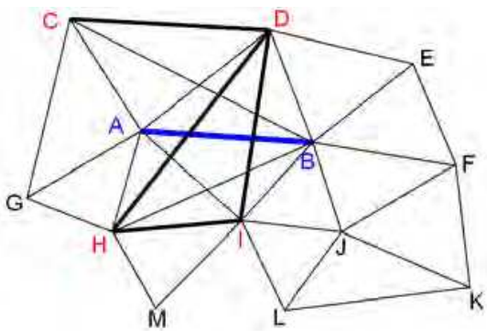


Fig. 22. Scheme for bond-type identification.

In the perfect crystals, the bond-type ID is easy to calculate. For example, all bond types are 421 in fcc crystal. There are two bond-types, 422 and 421, in hcp crystal. There are two bond-types, 441 and 661, in bcc crystal. All bond-types can be gathered as a list which will be referred to bond-type list later. Due to the variety of defect atoms in grain boundaries, there may be a large number of bond-types. For example, {421, 422, 441, 661, 200, 100, 311, 211,

411, 432, 542, 300, 400 …}. The bond-type list is dynamically increasing except for several special bond-types.

3.4.2.2 Atom-type identification

After identifying the bond-types, each atom gets its own bond-types. A distribution vector of bond-type of an atom can be generated by comparing its bond-types with the system bond-type list, and each component of the vector is number of the corresponding bond-type. For a fcc atom, it contains twelve 421 bonds, the corresponding distribution vector is {12}; for a hcp atom, it contains six 421 bonds and six 422 bonds, the corresponding distribution vector is {6, 6}; for a bcc atom, it contains six 441 bonds and eight 661 bonds, the corresponding distribution vector is {0, 0, 6, 8}; and for a kind of boundary atom, a distribution vector {5, 4, 0, 0, 0, 0, 2, 1} indicates that it contains five 421 bonds, four 422 bonds, two 200 bonds and a 100 bond. The distribution vector accurately describes an atom-type. In calculation, all distribution vectors can be gathered as a system atom-type list, the list is dynamically adjusted.

3.4.2.3 Bond-pair analysis algorithm

The preparatory work for algorithm is to design fast searchers. In this section, a design, including two conditional searchers, is proposed, where design a conditional function and an identification function for each conditional searcher.

(I) Conditional searcher CS1: Given a point p, search for a point whose distance to p is less than r_c. The conditional function is as follows:

$$condition(o) = \begin{cases} true & \left|\mathbf{r}_o - \mathbf{c}_p\right| < r_c \\ false & else \end{cases} \tag{16}$$

The circumcircle of branch b is used to identify. The identification function is as follows:

$$maycontain(b) = \begin{cases} true & \left|\mathbf{r}_P - \mathbf{c}_b\right| < r_c + d_b \\ false & else \end{cases} \tag{17}$$

(II) Conditional searcher CS2: Given two points P_1 and P_2, search for a points whose distances to P_1 and P_2 are less than r_c . The conditional function is as follows:

$$condition(o) = \begin{cases} true & \left|\mathbf{r}_{P_1} - \mathbf{r}_o\right| < r_c \ and \ \left|\mathbf{r}_{P_1} - \mathbf{r}_o\right| < r_c \\ false & else \end{cases} \tag{18}$$

The circumcircle of branch b is used to identify. The identification function is as follows:

$$maycontain(b) = \begin{cases} true & \left|\mathbf{r}_{P_1} - \mathbf{r}_o\right| < r_c + d_b \ and \ \left|\mathbf{r}_{P_1} - \mathbf{r}_o\right| < r_c + d_b \\ false & else \end{cases} \tag{19}$$

The algorithm for bond-pair analysis is as follows: (i) Initialization: set a bond-type array B and an atom-type array A, empty A and B, set a bond-type distribution vector V. Generate a point tree tp from the given discrete points (atoms), calculate the RDF of the system, set

bonding distance as r_c according to the distance corresponding to the first maximum of RDF. (ii) For each atom *a*, empty its bond-type distribution vector V. Search in the tree *tp* for an atom *b* bonding to the atom *a* (the distance between *a* and *b* is less than r_c) with searcher CS1. For each *a*-*b* bond, search in the tree *tp* for all atoms bonding to a and b (whose distance to a and b are less than r_c) with searcher CS2, compute the number of those atoms (denoted as *l*), check the connections between those atoms to get the number of bonds (denoted as *m*) and the largest coordination number (denoted as *n*), compute the bond-type of *a*-*b* as $100l+10m+n$ and then check whether or not it is a new bond-type by comparing with the bond-type list in *B*. If yes, add it to array *B* and then plus one on the corresponding component of *V*. Complete the loop of all bonds of *a*, check whether or not the atom-type of *a* is new by comparing with *A*. If yes, add it to array *A*. (iii) Go back to step (ii) until all atoms are used out.

3.4.2.4 Results show

Figure 23 shows the stacking faults and dislocations formed during the low-temperature evolution of gathered point defects (Frank loop) in fcc copper crystal. The defect atoms belonging to dislocations and stacking faults can be accurately identified, where blue atoms are stacking faults, and dislocation atoms are red. Figure 24 shows the growing of the two sphere voids in fcc copper under tension, at the very beginning, dislocations grows from the void surfaces, and different dislocations cross with each other in the late evolution. The atoms belonging to void surface, dislocations and stacking faults (no shown) can be strictly distinguished, they are marked with different colors.

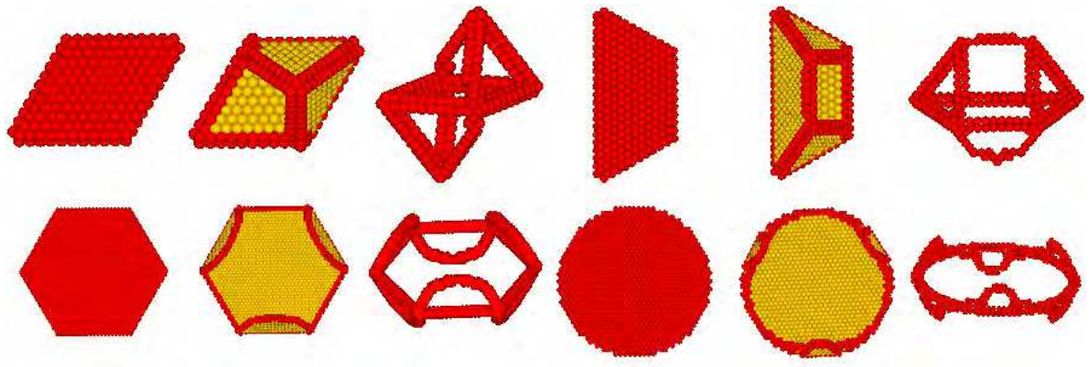


Fig. 23. Stacking faults and dislocations identified by bond-pair analysis.

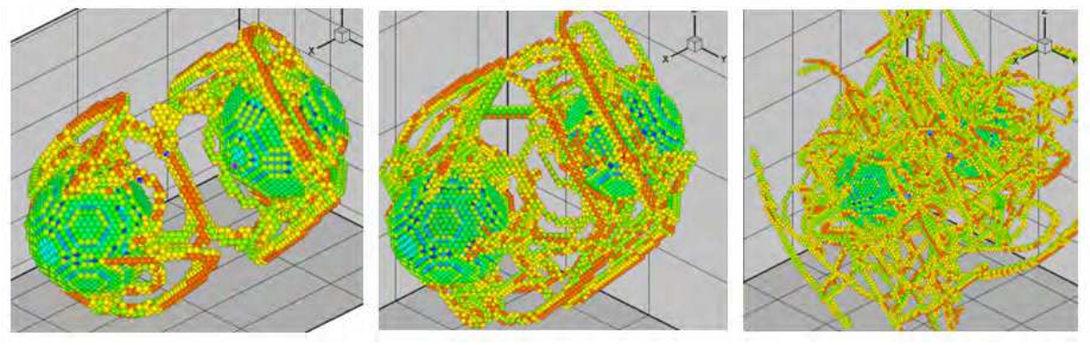


Fig. 24. Voids surfaces and dislocations identified by bond-pair analysis.

3.5 Surface construction algorithm

A key issue in the analysis of complex dynamic system is to form interface of concerned region. For example, in first order phase transition process, particle transportation and structure transition will cause growth and deformation of phase change zone. Interface determination of structural zone is the foundation of analyzing interface movements, cross-interface physical flow and understanding characterization of structure development. Interface construction is similar to surface reconstruction in computational geometry. Surface in computational geometry is formed from sampled points, while physical interface is the results of system evolution.

3.5.1 Packing-sculpting method for constructing object surface from disorder spatial points

It is an important issue in computational geometry to construct object surface from disorder points. The current algorithms can be categorized into four groups (Mencl & Muller, 1997), i.e. space partitioning method (Boissonant, 1984), distance function method (Hoppe et al., 1992), deformation method (Zhao, 2002), and growth method (Bernardini, 1999). Space partitioning is generally based on Delaunay division. The outer surface is generated by removing some Delaunay mesh in the sculpting method. The packing-sculpting method presented below is an intuitive method, the out surface is constructed by directly sculpting the packing convex hull. The basic idea is as follow: First of all, generate the packing convex hull from discrete points, and then sculpt the convex hull to construct the object surface.

3.5.1.1 Packing algorithm

The packing algorithm is to generate a convex hull by packing all given points. In this section, half-plane rotation method is introduced.

For the convenience of description, a new data structure, 'extended-segment', is defined as the combination of one side of triangle and triangle itself, it is an extended side of triangle with the data of the other vertex. The center and length of 'extended-segment' is that of the corresponding side of triangle. In the algorithm, there are various complicate searching for points, lines and surfaces. With the general index based on SHT, fast searching can be designed according to given searching conditions.

The preparatory work for algorithm is to design fast searchers. In this section, a design, including a minimum searcher and a conditional searcher, is proposed.

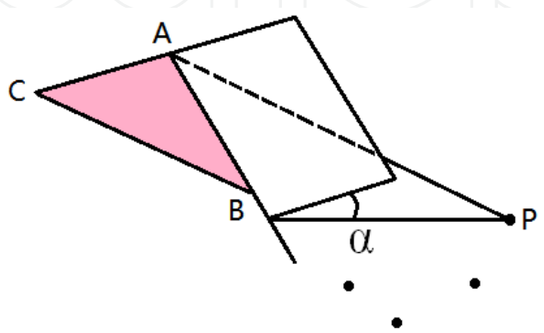


Fig. 25. Scheme for finding the first point during half-plane rotation.

(I) Minimum searcher MS1: Given a triangle *ABC*, pick out a segment *AB*, the extended half-plane of triangle *ABC* rotates around axis *AB*, searcher for the first point *P* meeting the extended half-plane to make smallest the dihedral angle shown in figure 25.

The value-finding function is as follows:

$$value(p) = angle(x, y) \tag{20}$$

Where, $x = \hat{\mathbf{x}} \cdot (\mathbf{r}_P - \mathbf{r}_A)$ and $y = \hat{\mathbf{y}} \cdot (\mathbf{r}_P - \mathbf{r}_A)$ is local coordinate of point *P*,

$$\hat{\mathbf{x}} = \frac{\mathbf{P}_{xy} \cdot (\mathbf{r}_A - \mathbf{r}_B)}{\left|\mathbf{P}_{xy} \cdot (\mathbf{r}_A - \mathbf{r}_B)\right|}, \hat{\mathbf{z}} = \frac{\mathbf{r}_A - \mathbf{r}_B}{|\mathbf{r}_A - \mathbf{r}_B|}, \hat{\mathbf{y}} = \hat{\mathbf{z}} \times \hat{\mathbf{x}} \tag{21}$$

are respectively the unit vector of direction of three reference axes,

$$\mathbf{P}_{xy} = \mathbf{E} - \hat{\mathbf{z}}\hat{\mathbf{z}} \tag{22}$$

is projection operator and **E** is identity operator. The circumsphere *S* of branch *b* is used for assessment. The range-evaluation functions are as follows:

$$\begin{aligned} M(b) &= \begin{cases} 2\pi & A \in S \text{ or } B \in S \\ value(\mathbf{c}_b) + arec\sin(\sqrt{3}d_b / \left|\mathbf{P}_{xy} \cdot (\mathbf{c}_b - \mathbf{r}_A)\right|) & A, B \notin S \end{cases} \\ m(b) &= \begin{cases} 2\pi & A \in S \text{ or } B \in S \\ value(\mathbf{c}_b) - arec\sin(\sqrt{3}d_b / \left|\mathbf{P}_{xy} \cdot (\mathbf{c}_b - \mathbf{r}_A)\right|) & A, B \notin S \end{cases} \end{aligned} \tag{23}$$

where $\mathbf{c}_b$ is the center of branch *b*.

(II) Conditional searcher CS1: Given a directed segment P_1P_2, search in extended-segment for an extended segment *BD* being equal to segment P_1P_2. The conditional function is as follows:

$$condition(BD) = \begin{cases} true & B == P_1 \text{ and } D == P_2 \\ false & else \end{cases} \tag{24}$$

The identification function is as follows:

$$maycontain(b) = \begin{cases} true & P_1, P_2 \in b \\ false & else \end{cases} \tag{25}$$

The packing algorithm is as follows: (I) Initialization: Construct a point tree *tp* from the given discrete points. According to the region size of root *tp*, pick out two vertexes: P_1=(-*a*,-*a*,*a*) and P_2=(-*a*,*a*,*a*), where *a* is half of the edge length of root region. Given a point P_3=(-2*a*,0,*a*). Search in the tree *tp* for the first point Q_1 met by rotating triangle $P_1P_2P_3$ around axis P_1P_2 with searcher MS1, generate a new triangle $P_1Q_1P_2$ and cut Q_1 down from *tp*. Search in the tree *tp* for the first point Q_2 met by rotating triangle $P_1Q_1P_2$ around axis P_1Q_1 with searcher MS1, generate a new triangle $Q_1Q_2P_1$ and cut Q_2 down from *tp*. Search in the tree *tp*

for the first point Q_3 met by rotating triangle Q_1Q_2 P_1 around axis Q_1Q_2 with searcher MS1, generate a new triangle $Q_3Q_1Q_2$ and cut Q_3 down from *tp*. Construct a triangle tree *tt* from triangle $Q_3Q_1Q_2$, generate extended-segment Q_3Q_1, Q_1Q_2 and Q_2Q_3, construct an extended-segment tree *tb* from Q3Q1 and put Q1Q2 and Q2Q3 into *tb*. Figure26 shows the initialization procedure of packing-algorithm.

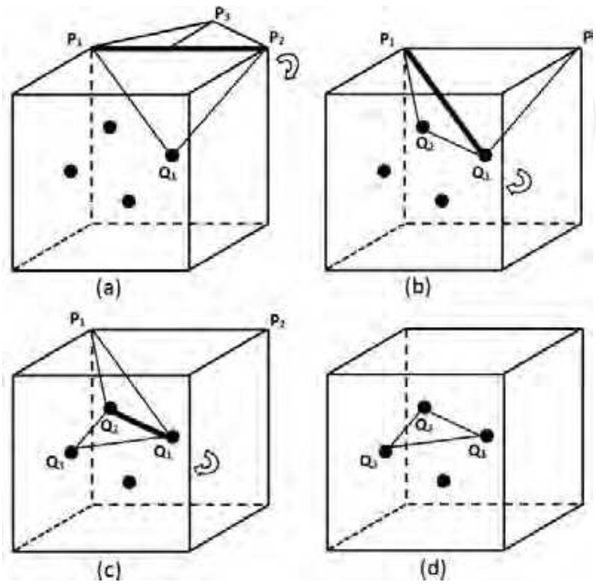


Fig. 26. Scheme for initialization of packing algorithm. (a) the first point Q_1 met by rotating triangle $P_1P_2P_3$ around axis P_1P_2; (b) the first point Q_2 met by rotating triangle $P_1Q_1P_2$ around axis P_1Q_1; (c) the first point Q_3 met by rotating triangle $Q_1Q_2P_1$ around axis Q_1Q_2; (d) initial triangle interface.

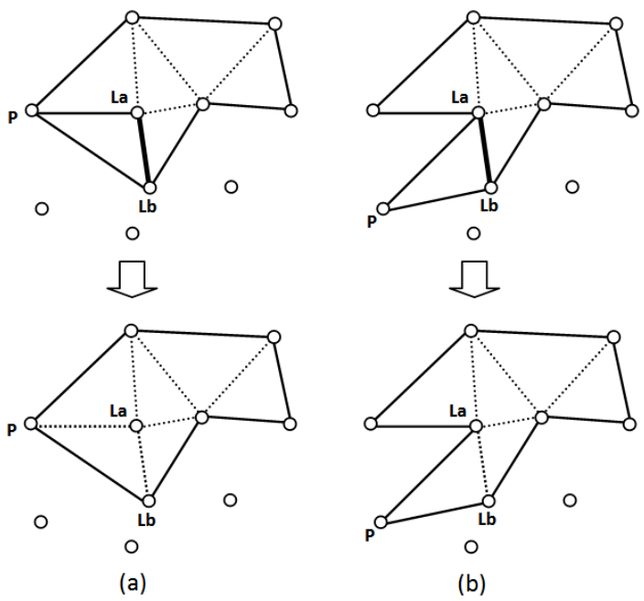


Fig. 27. Two cases. (a) L_aP is in the extended-segment tree; (b) L_aP is not in the extended-segment tree.

(III) Construction of packing convex hull: Cut down an extended-segment L from the tree tb, check whether or not L is empty. If yes, exit. If not, find the corresponding triangle A. Search in tp for the first point P met by rotating A around axis L with searcher MS1, generate a triangle L_bL_aP with point L_a, L_b and P, put the triangle L_bL_aP into the tree tt. Use CS1 to search in tb for an extended-segment G sharing the same segment with L_aP, check whether or not G exist. If yes (see figure 27(a)), cut it down from tb. If not (see figure 27(b)), generate an extended-segment from L_aP and put it into tb. Repeat the same operations to L_bP.

3.5.1.2 Sculpting algorithm

The sculpting algorithm refers to the sculpting method and the sculpting standard. We have to guarantee that no point is carved out during the sculpting procedure and make as smooth as possible the surface after sculpting. The curvature of smooth surface is small, and it indicates that the circumsphere radius of the tetrahedron which is carved is larger. For a triangle ABC under sculpting, the sculpting process means to find a point P in currently packed region to make smallest the height of circumsphere cap $ABCP$. Figure 28 is the scheme for sculpting algorithm.

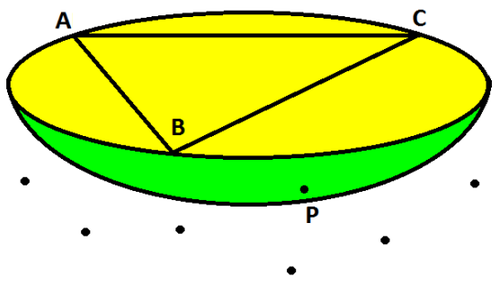


Fig. 28. Scheme for sculpting algorithm.

For a triangular surface, the stopping criterion for sculpting should be that the radius of circumsphere is less than a given value. In the case of uniform sampling of object surface, the criterion meets the requirements. However, in the case of nonuniform sampling (large curvature and dense sample points), the curvature must be related to the distance between local sample points. The ratio of the height of sphere cap and the circumcircle radius of a triangle, i.e. sculpture degree $c=h/r<c_{cri}$, is more suitable for stopping criterion. The criterion can be used in both cases above. Figure 28 shows the sculpting algorithm.

The preparatory work for algorithm is to design fast searchers. In this section, a design, including a minimum searcher, is proposed.

The minimum searcher MS is as follows: Given a triangle ABC, search in point tree for a point P to make smallest the height of circumsphere cap $ABCP$. The construction of value-finding function is as follows: By solving equations,

$$\begin{aligned} \mathbf{r}_o &= \alpha\mathbf{r}_A + \beta\mathbf{r}_B + \gamma\mathbf{r}_C \\ r^2 &= (\mathbf{r}_o - \mathbf{r}_A)^2 \\ r^2 &= (\mathbf{r}_o - \mathbf{r}_B)^2 \quad ,\mathbf{n} = \frac{(\mathbf{r}_B - \mathbf{r}_A)\times(\mathbf{r}_C - \mathbf{r}_A)}{\left|(\mathbf{r}_B - \mathbf{r}_A)\times(\mathbf{r}_C - \mathbf{r}_A)\right|} \\ r^2 &= (\mathbf{r}_o - \mathbf{r}_C)^2 \\ \alpha + \beta + \gamma &= 1 \end{aligned} \tag{26}$$

the circumcircle radius r of triangle ABC, the center position $\boldsymbol{o}$, and the normal direction $\mathbf{n}$ are obtained. By solving equations,

$$\begin{aligned} \mathbf{r}_c &= \mathbf{r}_o + \lambda \mathbf{n} \\ (\mathbf{r}_c - \mathbf{r}_P)^2 &= \lambda^2 + r^2 \end{aligned} \tag{27}$$

the coordinate of circumsphere center $\mathbf{r}_c$ and the height λ of $\mathbf{r}_c$ to circumcircle of triangle ABC are obtained. The height of circumsphere cap is as follows (i.e. the value-finding function):

$$value(P) = \sqrt{\lambda^2 + r^2} - \lambda \tag{28}$$

The range-evaluation function is calculated according to the tangent cases between circumspheres of branch b and sphere cap $ABCP$. It is as follows:

$$\begin{aligned} M(b) &= \begin{cases} \sqrt{\lambda_M^2 + r^2} - \lambda_M & circumsphere\ of\ ABC\ is\ out\ of\ circumsphere\ of\ b \\ \infty & else \end{cases} \\ m(b) &= \begin{cases} \sqrt{\lambda_m^2 + r^2} - \lambda_m & circumsphere\ of\ ABC\ is\ out\ of\ circumsphere\ of\ b \\ \infty & else \end{cases} \end{aligned} \tag{29}$$

where λ_M and λ_m are roots of equation $\left|\mathbf{r}_o + \lambda \mathbf{n} - \mathbf{c}_b\right|^2 = \left(\sqrt{\lambda^2 + r^2} \pm \sqrt{3} d_b\right)^2$, respectively, and they correspond to the two cases shown in figure 29. Set P as the tangent point of sphere cap and the circumsphere of b. When P becomes point Q_1, the height λ_m of sphere cap $ABCP$ is the smallest. When P becomes point Q_2, the height λ_M of $ABCP$ is the largest..

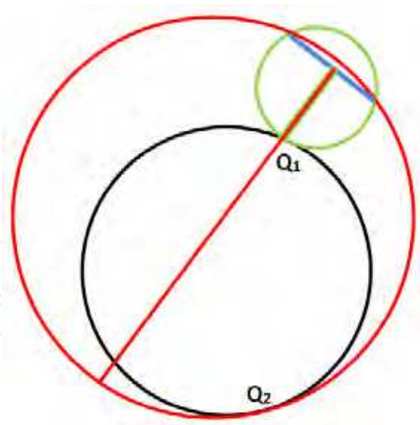


Fig. 29. Two tangent cases between circumspheres of branch b and sphere cap $ABCP$ (cross-section picture), where black circle is for the circumsphere of branch b, blue segment is for the circumcircle of triangle ABC, green and red circle are both for sphere cap $ABCP$, Q_1 and P_2 are for tangent points.

The sculpting algorithm is as follows:(I) Initialization: Take all triangles obtained by packing algorithm as triangle under sculpting, and the corresponding tree as triangle tree tt under sculpting. Set the critical value as 2.0 for stopping sculpting, set surface tree $tt0$ as empty. (II)

Sculpting: cut a triangle *T* from the tree *tt*, check whether or not *T* is empty. If yes, exit. If not, use MS to search for point *P*. Compute sculpture *c*, check whether or not c is less than c_{cri}. If yes, stop sculpting and put *T* into *tt*0. If not, generate three triangles by joining each side of *T* and point p, and then put them into *tt*.(III) Go back to step (II).The object surface consists of all surfaces in surface tree *tt*0.

3.5.1.3 Results show

Figure 30 shows the convex hull constructed from discrete points containing two nano-voids and uniform boundary points of a point defect by packing algorithm and object surfaces generated by sculpting. Figure 31 is for the reformed surface from random points sampled from tori. Figure 32 is for the reformed surface from 9000 random points sampled from spheres and tori.

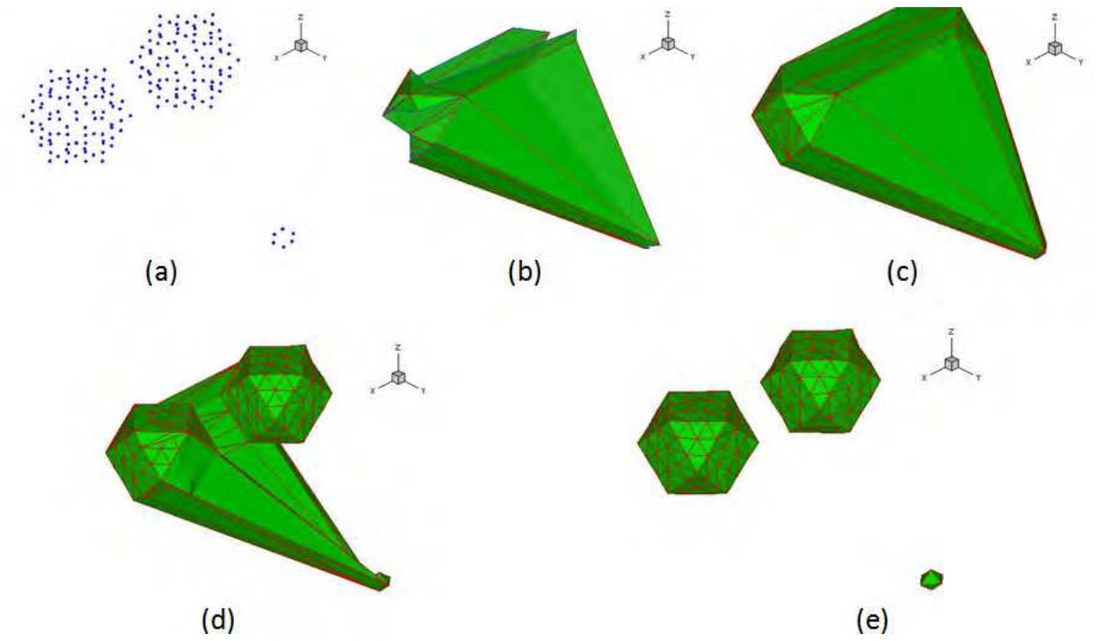


Fig. 30. The procedure of packing-sculpting algorithm. (a) discrete points; (b) the mid of packing procedure; (c) packing convex hull; (d) sculpting procedure; (e) object surface.

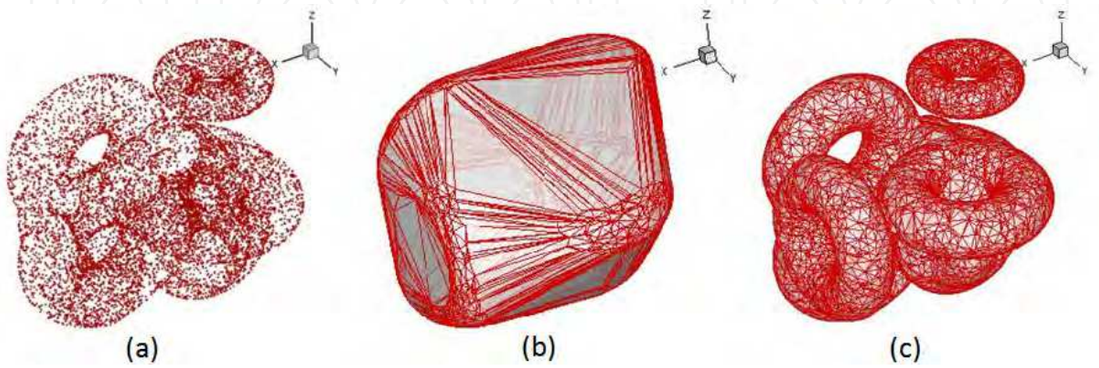


Fig. 31. Reformed surface from random points sampled from tori. (a) discrete points sample; (b) packing convex hull; (c) reconstructed object surfaces.

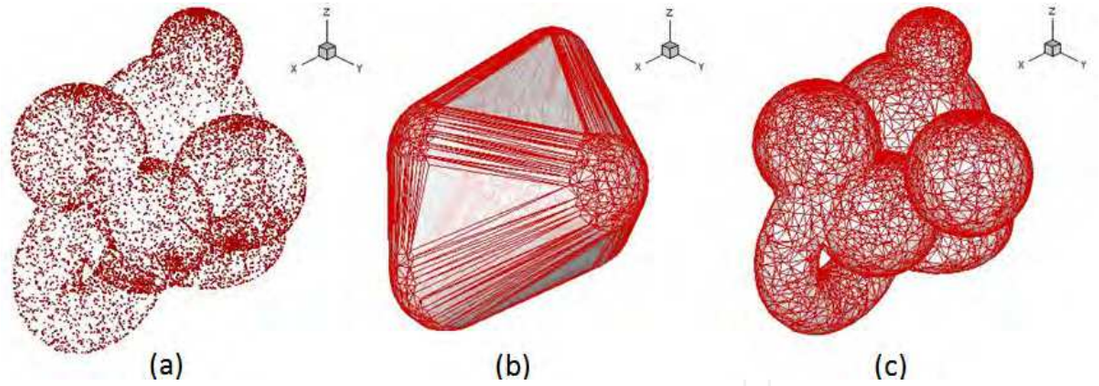


Fig. 32. Reformed surface from 9000 random points sampled from spheres and tori.
(a) discrete points sample; (b) packing convex hull; (c) reconstructed object surfaces.

The value of critical sculpture degree is 2.0 in sculpting algorithm. Application examples show that the algorithm is suitable for the cases of multi-connected or many-body surfaces. In the algorithm, the correctness of topology and geometry are guaranteed and the right surface can be constructed even in regions of greater curvature and at the transition zone of surfaces with different curvature.

3.5.2 Rolling-ball method for finding interfaces of physical regions from disorder spatial point

The difference between searching for interface of physical region from disordered spatial points and constructing surface of object is that the spatial points contain not only the points of physical interface but also points of other structures. The most common approach is to construct physical field on a regular grid, and the contour of physical field is used as the appropriate physical interface. This method is suitable for the case that the distribution of discrete points closed to interface is uniform. In the case of complex distribution of discrete points, it is hard to preserve the smoothness of the constructed interface, the calculated interface is very different from the actual interface. A better way is to use the rolling-ball method without constructing physical fields. The basic idea of rolling-ball method is as follows: roll a ball with fixed size on discrete point group, each rolling goes through three points, and these points constitute a surface element of interface. After the rolling-ball goes through the overall region, the physical interface is constructed. In rolling-ball method, the key parameter is the sphere radius. In the case of sparse sampling, different radiuses define different interfaces. Therefore, the size of the rolling-ball radius is obtained from experience.

3.5.2.1 Rolling-ball algorithm

The preparatory work for algorithm is to design fast searchers. In this section, a design, including three minimum searchers and a conditional searcher, is proposed.

(I) Minimum searcher MS1: Given a directed triangle ABC and pick out a segment AB, search in point tree for the first point met by the rolling-ball above triangle ABC, where the radius of rolling-ball is r and the rotation axis is AB. The construction of value-finding function is as follows: calculate the initial center $\mathbf{r}_o$ of the rolling-ball and directions of local axes $\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}$ .The calculation formula are as follows:

$$\begin{aligned} r^2 &= \left(\mathbf{r}_o - \mathbf{r}_A\right)^2 \\ r^2 &= \left(\mathbf{r}_o - \mathbf{r}_B\right)^2, \hat{\mathbf{x}} = \frac{\mathbf{P}_{xy} \cdot \left(\mathbf{r}_o - \mathbf{r}_A\right)}{\left|\mathbf{P}_{xy} \cdot \left(\mathbf{r}_o - \mathbf{r}_A\right)\right|}, \hat{\mathbf{y}} = \hat{\mathbf{z}} \times \hat{\mathbf{x}}, \hat{\mathbf{z}} = \frac{\mathbf{r}_B - \mathbf{r}_A}{\left|\mathbf{r}_B - \mathbf{r}_A\right|} \\ r^2 &= \left(\mathbf{r}_o - \mathbf{r}_C\right)^2 \end{aligned} \tag{30}$$

Calculate the rolling-ball center $\mathbf{r}_n$ after rotating and local coordinate x, y, z. The calculation formula are as follows:

$$\begin{aligned} r^2 &= \left(\mathbf{r}_n - \mathbf{r}_A\right)^2 \\ r^2 &= \left(\mathbf{r}_n - \mathbf{r}_B\right)^2, x = \hat{\mathbf{x}} \cdot \left(\mathbf{r}_n - \mathbf{r}_A\right), y = \hat{\mathbf{y}} \cdot \left(\mathbf{r}_n - \mathbf{r}_A\right), z = \hat{\mathbf{z}} \cdot \left(\mathbf{r}_n - \mathbf{r}_A\right) \\ r^2 &= \left(\mathbf{r}_n - \mathbf{r}_P\right)^2 \end{aligned} \tag{31}$$

Calculate the rotation angle, i.e. the value of value-finding function. Figure 33 shows the scheme for the rotation of triangle *ABC*.

$$value(P) = angle(x, y) \tag{32}$$

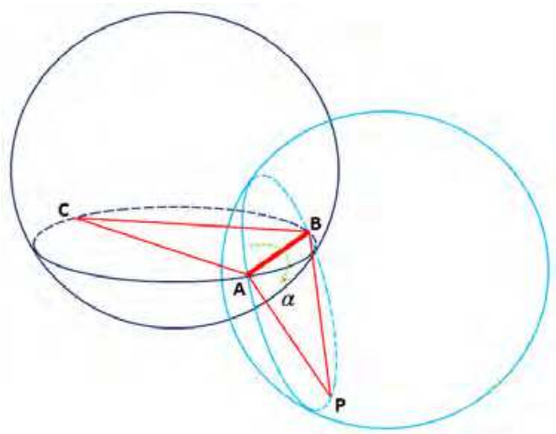


Fig. 33. Scheme for the rotation of triangle *ABC*.

The procedure for constructing range-evaluation function is as follows: calculate the position of the tangent point T of rolling-ball and the circumsphere of a branch. The calculation formula read

$$r^2 = \left|\mathbf{r}_T - \mathbf{r}_A\right|^2 = \left|\mathbf{r}_T - \mathbf{r}_B\right|^2, \left|\mathbf{r}_T - \mathbf{c}_b\right| = r + \sqrt{3} d_b \tag{33}$$

The two roots are $\mathbf{r}_{ML}$ and $\mathbf{r}_{mL}$, and the corresponding points are ML and mL. The range-evaluation functions are as follows:

$$\begin{aligned} M(b) &= \begin{cases} value(ML) & circumsphere\ of\ a\ triangle\ is\ out\ of\ its\ circumsphere \\ \infty & else \end{cases} \\ m(b) &= \begin{cases} value(mL) & circumsphere\ of\ a\ triangle\ is\ out\ of\ its\ circumsphere \\ \infty & else \end{cases} \end{aligned} \tag{34}$$

Figure 34 shows the two tangent cases between the circumspheres of branch b and the rolling-ball (cross section picture), where back circle is for the circumsphere of branch b, blue, green and red circle are for rolling-balls, ML and mL are for corresponding tangent points. If the tangent point is ML, the rotation angle of rolling-ball is the smallest. If it is mL, the rotation angle is the largest.

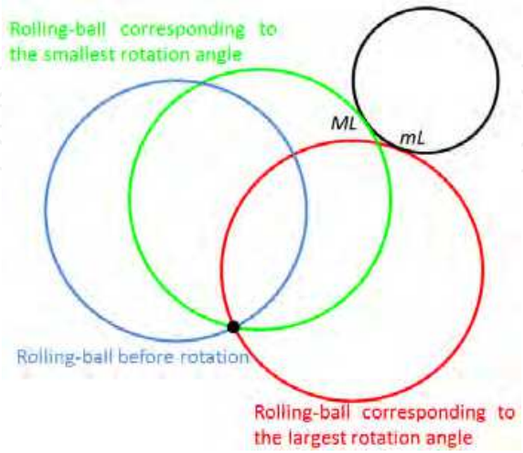


Fig. 34. Scheme for the two tangent cases between the rolling-ball and the circumsphere of branch b.

(II) Minimum searcher MS2: Given a point $\mathbf{r}_0$, search for the nearest point P of the given point in point tree. The value-finding function is

$$value(P) = \left|\mathbf{r}_P - \mathbf{r}_0\right| \tag{35}$$

where $\mathbf{r}_P$ is coordinate of point P. The range-evaluation functions are

$$\begin{aligned} M(b) &= \left|\mathbf{c}_b - \mathbf{r}_0\right| + \sqrt{3}d_b \\ m(b) &= \max(\left|\mathbf{c}_b - \mathbf{r}_0\right| - \sqrt{3}d_b, 0) \end{aligned} \tag{36}$$

(III) Minimum searcher MS3: Given a point and a rotation axis, search in point tree for the first point by the rolling-ball with fixed size. The algorithm is basically the same as MS1. We do not repeat here.

(IV) Conditional searcher CS1: Given two point P_1 and P_2, search in extended-segment tree for segment BD whose vertexes are P_1 and P_2. The conditional function is as follows:

$$condition(BD) = \begin{cases} true & B == P_1 \ and \ D == P_2 \\ false & else \end{cases} \tag{37}$$

The circumsphere S of branch b is used for identification. The identification function is as follows:

$$maycontain(b) = \begin{cases} true & P_1, P_2 \in S \\ false & else \end{cases} \tag{38}$$

The rolling-ball algorithm is as follows: (I) Initialization: Set the radius of rolling-ball and the center as P_0, generate point tree *tp* from given discrete points, search for the nearest point P_1 to P_0 with searcher MS2. Use searcher MS3 to search in tree *tp* for a point P_2 which is the first point met by the rolling-ball rotating along x axis. Use searcher MS3 to search in tree *tp* for a point P_3 which is the first point met by the rolling-ball rotating along the direction of segment P_1P_2. Generate a triangle from P_1, P_2 and P_3, construct a triangle tree *tt* from triangle P_1P_2 P_3 and put it into extended-segment tree *tb*. (II) Interface construction: Check whether or not the tree *tb* is null. If yes, exit. If not, cut down a segment *AB* of triangle *ABC*. Search in tree *tb* for a point *P* with searcher MS1 to make smallest the rotation angle of circumsphere of triangle *ABC*, where *AB* is the rotation axis. Construct triangle *BAP* and put it into the triangle tree *tt*. Use CS1 searcher to search in tree *tb* for an extended-segment *L* whose vertexes are point *B* and *P*. Check whether or not *L* exist. If yes, cut *L* down from *tb* and then delete it. If not, generate an extended-segment *PB* and put it into *tb*. Do the same operations to point *P* and *A*. (III) Go back to step (II). The collection of triangles contained by tree *tt* is just the needed physical interface.

3.5.2.2 Results show

In molecular dynamical simulations on voids coalescence in fcc copper, defects atoms include atoms in void walls and dislocations. Figure 35 shows interface of voids constructed from discrete atom positions. Figure 36 shows the corresponding construction process. The radius of rolling-ball is set as 2/3 times of the lattice constant so as to distinguish residual structures on the wall of voids after dislocation slipping. In figure 35(b), the stairs composed by array of atoms can be clearly seen from the constructed surface. When the voids are growing, the trace of dislocation migration and the irregular shape of the void coalescence zone are truly shown up.

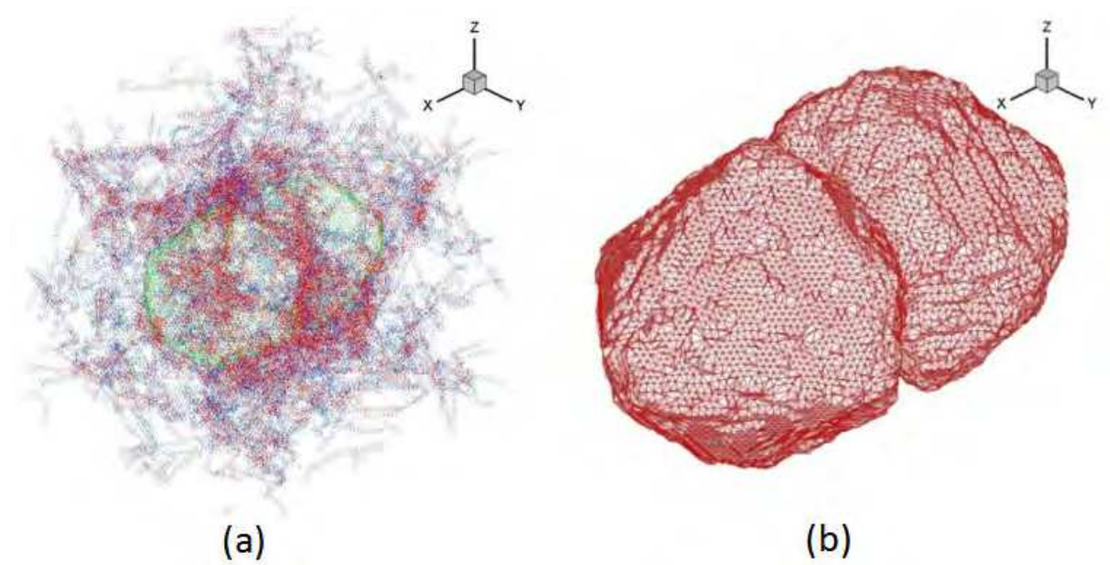


Fig. 35. Interface of voids constructed from discrete points. (a) discrete points; (b)constructed interface.

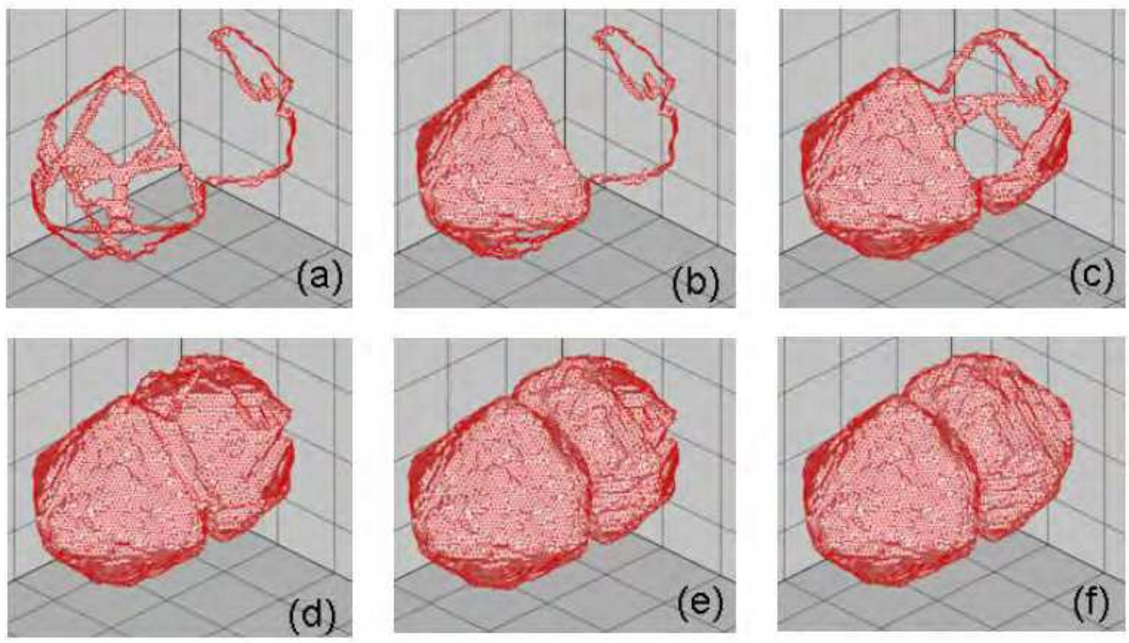


Fig. 36. Process of constructing interface of voids from discrete points.

4. Acknowledgment

Our thanks are due to Profs. Shigang Chen, Yingjun Li, Haifeng Liu, Ping Zhang, Haifeng Song, Guicun Ma, Hui Zheng, and Drs. Feng Chen, Yanbiao Gan, Weiwei Pang, Bo Sun, Yanhong Zhao, Shuaichuang Wang, Gongmu Zhang, Hongzhou Song, Qili Zhang, Mingfeng Tian, for helpful comments and discussions. This work was supported by the National Natural Science Foundation of China (Grant No. 11075021) and the Science Foundations of the Laboratory of Computational Physics and China Academy of Engineering Physics (Grant Nos. 2009A0102005 and 2011A0201002).

5. References

Ackland, G. J. & Jones, A. P. (2006). Physical Review B. *Applications of local crystal structure measures in experiment and simulation*, Vol. 73, No. 5, pp. 054104

Ament, P. (2008). Physical Review Letter, *Effects of ionization gradients on Inertial-Confinement-Fusion capsule hydrodynamics stability*. Vol. 101, No. 11, pp. 11504

Bernardini, F., Mittlelman, J., Rushmeir, H. & Silva, C. (1999). IEEE Trans. Visual. Comput. Graphics, *The ball-pivoting algorithm for surface reconstruction*, Vol. 5, No. 8, pp. 349-359

Boissonant, J. D. (1984). ACM Transactions on Graphics, *Geometric structures for three dimensional shape representation*, Vol. 3, No. 4, pp. 266-286

Bowler, B. P., Waller, W. H., Megeath, S. T., et al. (2009). The Astronomical Journal, *An infrared census of star formation in the horsehead nebula*, Vol. 137, No. 3, pp. 3685—3699

Clarkson, K. L., & Varadarajan, K. (2007). Discrete & Computational Geometry, *Improved approximation algorithms for geometric set cover*. Vol. 37, No. 1, pp.43—58

Chazelle, B., Devillers, O., Hurtado, F., et al. (2002). Algorithmica, *Splitting a Delaunay triangulation in linear time*, Vol. 34, pp.39—46

de Berg, M., Cheong, O., van Kreveld, M. & Overmars M. (2008). *Computational Geometry* (3rd revised ed.), Springer-Verlag, ISBN 978-3-540-77973-5, New York, USA

de Berg, M., Cheong, O., van Kreveld, M. & Overmars M. (2008). *Computational Geometry: Algorithms and Applications*, Springer-Verlag, ISBN 978-3-540-77973-5, New York, USA

de Vriesl, P. C., Hua, M. D., McDonald, D. C., et al. (2008). Nuclear Fusion, *Scaling of rotation and momentum confinement in JET plasmas,*Vol. 48, No. 6, pp. 065006

Dierk, R. (1998). *Computational Materials Science: The Simulation of Materials Microstructures and Properties*, Wiley-VCH , ISBN 978-3527295418, New York, USA

Donald, K. (1998). *The Art of Computer Programming*, Volume 3: Sorting and Searching (2nd Edition), ISBN 978-0201896855, Addison-Wesley

Faken, D. & Jonsson, H. (1994). Computational Materials Science. *Systematic analysis of local atomic structure combined with 3D computer graphics*, Vol.2, No. 2, pp. 279-286

Fan, Y. J., Iyigun, C. & Chaovalitwongse, W. A. (2008). CRM Proc Lecture Notes, *Recent advances in mathematical programming for classification and cluster analysis*, Vol. 45, pp.67—93

Hernquist, L. (1988). Computer Physics Communications, *Hierarchical N-body methods*, Vol. 48, pp.107—115

Hidaka, Y., Choi, E. M., Mastovsky, I., et al. (2008). Physical Review Letters, *Observation of large arrays of plasma filaments in air breakdown by 1.5-MW 110-GHz gyrotron pulses*, Vol. 100, No. 4, pp.035003

Hoppe, H., DeRose, T., Duchanp, T., Mc-Donald, J. & Stuetzle,W. (1992). ACM Computer Graphics, *Surface reconstruction from unorganized points*, Vol. 26, No. 2,pp. 71-78

Kelchner, C. L., Plimpton, S. J. &Hamilton, J. C. (1998). Physical Review B. *Dislocation nucleation and defect structure during surface indentation*, Vol. 58, No. 17, pp.11085

Kotsiantis, S. B. & Pintelas, P. E. (2004). WSEAS Transactions on Information Science and Applications, *Recent advances in clustering: A brief survey*, Vol. 1, No. 1: 73—81

Kumar, K.S., Van Swygenhoven, H. & Suresh, S. (2003). Acta Materialia. *Mechanical behavior of nanocrystalline metals and alloys,* Vol. 51, No. 19, pp. 5743–5774

Makino, J. (1990). Journal of Computational Physics, *Vectorization of a treecode*,Vol. 87, No. 1, pp.148—160

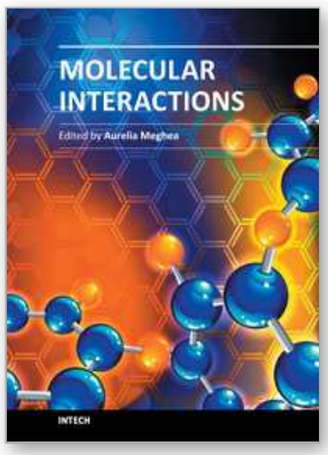
Mencl, E. & Muller, H. (1997). Interpolation and approximation of surfaces from three-dimensional scattered data points, *Scientific Visualization Conference 1997*, Dagstuhl, Germany, June 1997

Michael, G., Stephan, K. & Gerhard Z. (2007). *Numerical Simulation in Molecular Dynamics*, Springer-Verlag, ISBN 978-3-540-68094-9,Berlin, Heidelberg

Moore, E.F. (1959). The shortest path through a maze, *Proceedings of an International Symposium on the Theory of Switching* , Cambridge, Massachusetts, April 1957.

Nogaret, T., Rodney, D., Fivel, M., et al. (2008). Journal of Nuclear Materials, *Clear band formation simulated by dislocation dynamics: Role of helical turns and pile-ups*, Vol. 380, No. 1-3, pp. 22—29

Zhao, H. K. & Osher, S., Merriman, B. & Kang, M. (2002). Computer Vision and Image Processing, *Implicit, non-parametric shape recontruction from unorganized points using variational level set method*, Vol. 80, No. 3, pp. 295-314



Molecular Interactions

Edited by Prof. Aurelia Meghea

ISBN 978-953-51-0079-9
Hard cover, 400 pages
Publisher InTech
Published online 29, February, 2012
Published in print edition February, 2012

In a classical approach materials science is mainly dealing with interatomic interactions within molecules, without paying much interest on weak intermolecular interactions. However, the variety of structures actually is the result of weak ordering because of noncovalent interactions. Indeed, for self-assembly to be possible in soft materials, it is evident that forces between molecules must be much weaker than covalent bonds between the atoms of a molecule. The weak intermolecular interactions responsible for molecular ordering in soft materials include hydrogen bonds, coordination bonds in ligands and complexes, ionic and dipolar interactions, van der Waals forces, and hydrophobic interactions. Recent evolutions in nanosciences and nanotechnologies provide strong arguments to support the opportunity and importance of the topics approached in this book, the fundamental and applicative aspects related to molecular interactions being of large interest in both research and innovative environments. We expect this book to have a strong impact at various education and research training levels, for young and experienced researchers from both academia and industry.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen