Visual Comput (2007) 23: 143–154
DOI 10.1007/s00371-006-0089-0

ORIGINAL ARTICLE

Angel Domingo Sappa
Miguel Angel Garcia

Generating compact representations of static scenes by means of 3D object hierarchies*

Published online: 19 December 2006

A.D. Sappa (✉)
Computer Vision Center, Edifici O Campus UAB, 08193 Bellaterra, Barcelona, Spain
angel.sappa@cvc.uab.es

M.A. Garcia
Department of Informatics Engineering, Autonomous University of Madrid, Cra. Colmenar Viejo, Km. 15, 28049 Madrid, Spain
miguelangel.garcia@uam.es

Abstract This paper presents a new heuristic algorithm for computing a compact hierarchical representation of the objects contained in a 3D static scene. The algorithm first builds a fully-connected adjacency graph that keeps the costs of grouping the different pairs of objects of the scene. Afterward, the graph's minimum spanning tree is computed and its edges sorted in ascending order according to their cost. Next, from that sorted list, a cost-based clustering technique is applied, thus generating new objects at a higher level in the hierarchy. A new object can be defined after merging two or more objects according to their corresponding linking costs. The algorithm starts over by generating a new adjacency graph from those new objects, along with the objects that could not be merged before. The iterative process is applied until an adjacency graph with a single object is obtained. The latter is the root of the hierarchical representation. Balance and coherence of the hierarchy, in which spatially close objects are also structurally close, is achieved by defining an appropriate cost function. The proposed technique is evaluated upon several 3D scenes and compared to a previous technique. In addition, the benefits of the proposed technique with respect to techniques based on octrees and kd-trees are analyzed in terms of a practical application.

Keywords World modeling · Object clustering · Hierarchical representation · Minimum spanning tree · Minimum distance computation

1 Introduction

A wide variety of disciplines, such as robotics, virtual reality or computer graphics, share the common need to deal with complex 3D scenes composed of large amounts of objects in order to perform a variety of tasks that involve some type of spatial reasoning, such as collision detection, visibility analysis or path planning, to mention a few (e.g., [5, 8, 9, 12, 13, 15, 17]). If those tasks are to be carried out in real time, it is necessary to organize the scene's objects in some appropriate way to speed up the whole process.

Hierarchical representations are one of the most popular spatial organizations, as they allow to keep complex information at different abstraction levels. In particular, hierarchical models have been proposed as an efficient and compact representation to tackle a multitude of prob-

*This work has been partially supported by the Spanish Ministry of Education and Science under projects TRA2004-06702/AUT and DPI2004-07993-C03-03. The first author was supported by *The Ramón y Cajal Program*.

lems within the 3D modeling community. Basic geometric primitives, including bounding boxes or spheres, have been utilized to decompose either a scene or a single object into a hierarchical structure.

Hence, many techniques have already been proposed in the 3D modeling literature to automatically generate hierarchical structures. These approaches can be broadly classified into *space-oriented* and *object-oriented* schemes. Although they both pursue the generation of a hierarchical representation, their underlying philosophy is significantly different as it is briefly described in Sect. 2.

This paper presents an object-oriented technique for computing a compact hierarchical representation of the objects contained in a 3D scene that retains their spatial distribution. It is based on a heuristic cost function that minimizes the empty bounding space by relating the volume of clustered objects to the volume of the resulting cluster. Graph theory is also used to compute the best set of connections that interrelate all the objects contained in the scene with a minimum total cost.

The algorithm assumes that every object contained in the scene is represented by a single bounding sphere or, alternatively, in case that a tighter fit is required, by a collection of spheres [13]. Initially, a fully-connected graph that keeps the costs of grouping the different pairs of input spheres is determined. Then, the minimum spanning tree of that graph is computed and its edges sorted in ascending order of cost. Afterward, a cost-based clustering technique generates new spheres at a higher level of the hierarchy by merging spheres at the current level. In case the new level contains several spheres, the algorithm starts over at that new level by generating a new fully-connected graph and applying the aforementioned stages again. Otherwise, if the new level contains a single sphere, the iterative process stops with that sphere being the root of the hierarchical representation.

The proposed algorithm is more advantageous than previous proposals since: (1) a more realistic cost measurement function is defined, (2) graph theory is used to find the best set of clusters, (3) efficiency is higher as the costs between all objects of the scene are computed and updated after every new level generation stage, (4) there are not user-tuned parameters and (5) the generated tree is n-ary instead of binary and, thus, it is shallower and more representative of the physical distribution of the objects contained in the scene.

The remainder of the paper is organized as follows. The next section summarizes previous approaches. The proposed algorithm is described in Sect. 3. Section 4 presents experimental results and a comparison with the technique described in [21]. In addition, comparisons with space-oriented hierarchical representations based on octrees and kd-trees are also presented from the point of view of the performance of a particular application (find the nearest sphere to a given segment). Finally, conclusions and further improvements are given in Sect. 5.

2 Previous approaches

Space-oriented schemes, usually based on top-down refinement, consider a given scene as a whole and progressively subdivide its volume at every level of the hierarchy. The most popular representations in this category are: *binary space partitioning trees* [11], *octrees* [16], *extended octrees* [3] and *kd-trees* [1]. However, since the objects contained in the scene are considered as a part of it and not treated as single entities, many basic tasks that require the processing of individual objects (e.g., minimum distance computation) may suffer from a significant downfall in performance, as it will be illustrated in Sect. 4.2.

Space-oriented approaches require the definition of a termination criterion. In general, termination criteria are specified by end-users by means of two thresholds that determine when a node becomes a leaf: (1) when the number of objects contained in that node is lower than or equal to a user-defined threshold, and (2) when the tree's depth reaches a second user defined threshold. Although some work has been done for determining automatic termination criteria (term coined in [20]), it still remains an open problem. The work done in [20] proposes a cost model that can be used to predict the correct termination point, although no particular algorithm is described. A step forward was presented in [10], where an empirical algorithm based on experiments with different scenes and amounts of objects was proposed. But again, the automatic termination criterion is not fully reached due to the fact that two values need to be tuned, with these constants being chosen according to the proposed experiments.

In turn, object-oriented schemes focus on the individual objects contained in the scene rather than on the scene as a whole. Two basic approaches can be distinguished. The first approach consists of decomposing the scene's objects into hierarchies of basic geometric primitives (e.g., [2, 9, 13–15, 17]). This is referred to as *intra-object representations* [7]. The majority of object-oriented schemes that have been proposed so far belong to this category. Alternatively, the second approach does not pretend to decompose objects into simpler parts, but to group them appropriately. Hierarchies generated in this way are referred to as *inter-object representations* [7]. The current paper falls into this second category.

Although both inter-object and intra-object representations have been conceived for different applications, they can complement each other. For example, a single object can be first represented by means of a hierarchy of bounding volumes (intra-object representation). Then, in a second step, all the objects contained in the scene can be interrelated, giving rise to an inter-object representation of that scene. In this case, each leaf of the inter-object hierarchy will be the root of an intra-object representation that describes a single object.

Inter-object representations aim at describing a scene by interrelating the objects contained in it in a structure

that reflects their spatial distribution and, ideally, unveils the functional organization of the scene. The main objective is to obtain a hierarchical structure that agrees with human intuition and clusters the given objects according to their spatial location.

Hierarchical inter-object representations have been successfully proposed, for example, for solving complex tasks in robotics under kinematic and dynamic constraints (e.g., [4, 5]). In these works, however, a hierarchical representation of the objects contained in the scene is required as an input to the algorithm, which only focuses on the use of those representations for a path search algorithm, without paying attention to the way in which they are generated (no clues about whether they were manually or automatically generated are given).

Approaches based on inter-object representations usually follow a bottom-up strategy. For instance, [21] starts with the bounding volumes (e.g., spheres, convex hulls) of the objects contained in the 3D scene and groups them into bigger bounding volumes by applying heuristic rules that favor the progressive growth of those volumes. The algorithm goes over all the possible pairings of objects in the scene and, for those whose distance is below a certain moving and progressively larger threshold, it computes a grouping cost. The pair of objects with the lowest cost is grouped into a new object and the costs between the new and the old objects are recomputed. The algorithm stops when a single object is left. Thus, a binary tree is generated. At every iteration, the cost function defined in [21] takes into account three heuristics that favor the grouping of the candidate pair of objects whose bounding volume has: (1) the smallest volume, (2) the smallest amount of empty space inside and (3) the most similar ratio between the size of the two objects that constitute the candidate pairing. The cost function is defined by a weighted average of those three criteria. An important drawback of the algorithm is that the topology of the final hierarchy is highly dependent on a large number of user-tuned parameters that intervene in the definition of the aforementioned heuristics.

A different approach was presented in [7]. In this technique, a minimum spanning tree (MST) is extracted from a fully-connected graph that keeps the cost of grouping every pair of objects of the scene. The cost function is based on Newton's law of universal attraction. From the MST, an n-ary tree is finally generated. This approach is very efficient as the cost between the objects of the scene is only computed once. Furthermore, it has no parameters that have to be carefully tuned. However, the lack of a cost update as the grouping proceeds becomes a disadvantage when large, complex scenes are considered. This favors the generation of non-intuitive groupings when both large and small spheres coexist in the scene. A more efficient approach to the problem of inter-object grouping is presented in this paper as it is explained below.

3 3D object hierarchies

This section presents a new algorithm for generating a hierarchical clustering of the objects contained in a complex 3D scene. This algorithm follows a bottom-up strategy that progressively groups the bounding spheres corresponding to the original objects in order to produce a tree. Each node of the tree represents a level of abstraction in the scene and has associated the minimum bounding sphere that contains its children nodes.

The algorithm consists of an iterative process that builds each level of the hierarchy by grouping the bounding spheres contained at the previous level. The process stops when a single bounding sphere is left. Three stages are subsequently applied at every level. In the first stage, a fully-connected adjacency graph that keeps the cost of grouping the different pairs of spheres present at the current level is built. In the second stage, the *minimum spanning tree* (*MST*) of that graph is computed. In the last stage, the edges of the MST are sorted in ascending order according to their costs. Nodes linked by those edges are merged whenever a merging criterion is satisfied. As a result, a new level of the hierarchy is generated, which contains the new spheres and the ones that could not be grouped. The same three stages are then applied to the next level. These stages are illustrated in the flow chart shown in Fig. 1 and described below after introducing some notation and concepts used throughout the rest of the paper.

Let $S = \{S_1, \ldots, S_n\}$ be the set of n spheres present at a certain level λ of the hierarchy. Every sphere S_i has radius r_i and is centered at a 3D position $\boldsymbol{C}_i$. Those spheres will initially be the bounding spheres of the original ob-

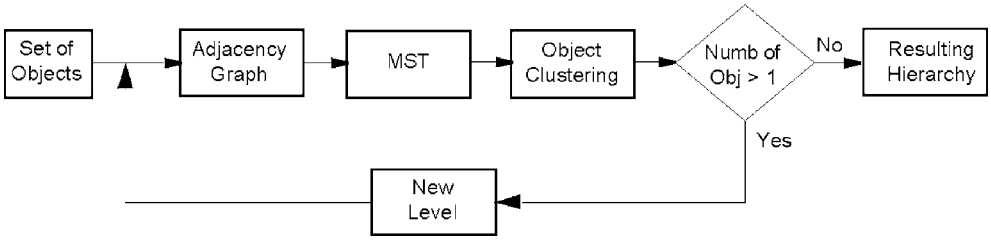

Fig. 1. Illustrations of the algorithm's stages

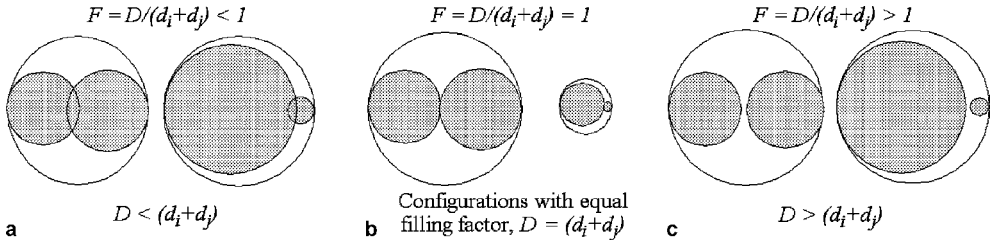

Fig. 2. Illustrations of filling factors computed according to [21]. In the three situations, the left and right configurations have similar filling factors, while the real filling is significantly different

jects contained in the scene. If two spheres S_i and S_j are to be grouped, they are replaced by their smallest bounding sphere S_{ij}, which has radius r_{ij} and center $\boldsymbol{C}_{ij}$:

$$\begin{aligned} &r_{ij} = r_i \quad \boldsymbol{C}_{ij} = \boldsymbol{C}_i \quad \text{if} \quad (\|\boldsymbol{C}_i - \boldsymbol{C}_j\|_2 + r_j) \leq r_i \\ &r_{ij} = r_j \quad \boldsymbol{C}_{ij} = \boldsymbol{C}_j \quad \text{if} \quad (\|\boldsymbol{C}_i - \boldsymbol{C}_j\|_2 + r_i) \leq r_j \\ &r_{ij} = (r_i + r_j + \|\boldsymbol{C}_i - \boldsymbol{C}_j\|_2)/2 \quad \text{and} \\ &\boldsymbol{C}_{i,j} = \frac{\boldsymbol{C}_i - \boldsymbol{C}_j}{\|\boldsymbol{C}_i - \boldsymbol{C}_j\|_2}(r_{ij} - r_j) + \boldsymbol{C}_j \quad \text{otherwise.} \end{aligned} \tag{1}$$

In case several spheres are to be grouped, they are substituted for the smallest bounding sphere S_b that contains them all. The radius and center of the new sphere are respectively computed through a fast technique based on the algorithm proposed in [6]. This algorithm computes the smallest enclosing ball of a given point set. That point set is constituted by the spheres' center points enlarged according to their corresponding radii.

3.1 Adjacency graph generation

The first stage of the grouping process generates a fully connected weighted graph in which every node represents a sphere present at level λ. Every pair of nodes in the graph is linked with an edge whose weight expresses the cost of grouping the bounding spheres corresponding to the two nodes linked by the edge.

The key point of this first stage is the formulation of the grouping cost function that defines the weight associated with every graph's edge, since, in the end, it will determine the order in which spheres will be grouped. This function must be properly defined in order to achieve a well balanced and spatially coherent hierarchy. Previous schemes propose cost functions based on: (a) the attraction force between two objects [7]; (b) a heuristics that favors that objects of similar size are merged, beginning with those with the smallest volume [21]; (c) a relationship between the area of the enclosed spheres (children) and the corresponding enclosing one (father) (e.g., [8, 10, 20]). As it was stated in [12], techniques based on surface area are commonly used for ray tracing applications, since the probability that a ray will intersect a bounding volume is proportional to its surface area. For collision detection applications, however, volume needs to be minimized expecting that it is proportional to the probability of intersecting other objects.

Thus, we define a cost function Eq. 5 that depends on both the volume of the smallest sphere that contains the spheres to be grouped, and a filling factor defined as the ratio between both the volume occupied by those spheres and the volume of their smallest bounding sphere. In other words, the proposed filling factor is a measure of empty space, as opposed to [21], where the filling factor is defined as the ratio between the diameter of the bounding sphere (D) and the sum of diameters of the merged spheres ($d_i + d_j$).

Figure 2 illustrates some situations where the filling factor F proposed in [21] is not a good indication of the real filling. For example, the smallest bounding sphere in Fig. 2b [right grouping] should be the one with the best filling factor. However, both bounding spheres have the same factor ($F = 1$). The same occurs in Figs. 2a and 2c: the configurations on the right (i.e., Figs. 2a [right grouping] and 2c [right grouping]) have better filling than the ones on the left (i.e., Figs. 2a [left grouping] and 2c [left grouping]), although their filling factors computed according to [21] are approximately the same.

Another advantage of the proposed technique over the one defined in [21] is that the proposed filling factor is preserved in the hierarchy as an indicator for further grouping. Thus, every sphere S_i is associated with a filling factor F_i. The bounding spheres corresponding to the original objects of the scene have a filling factor equal to one. In turn, when a set of spheres $\zeta = \{S_i | 1 \leq i \leq n\}$ is grouped, the filling factor of their minimum bounding sphere, S_b, is formulated as:

$$F_b = \Big(\Big(\sum_{\zeta} F_i Vol_i\Big) - \Big(\sum_{\zeta} Vol_{\text{overlap}(i,j)}\Big)\Big) \Big/ Vol_b, \tag{2}$$

where Vol_i, F_i correspond to the volume and filling factor of every sphere to be grouped respectively; Vol_b is the volume of S_b; and $Vol_{\text{overlap}(i,j)}$ is the volume defined by the overlap (intersection) between every pair of spheres from ζ. Assuming S_i and S_j are two spheres from ζ, their overlapping volume, $Vol_{\text{overlap}(i,j)}$, is computed as the union of the volumes of two spherical caps (Fig. 3):

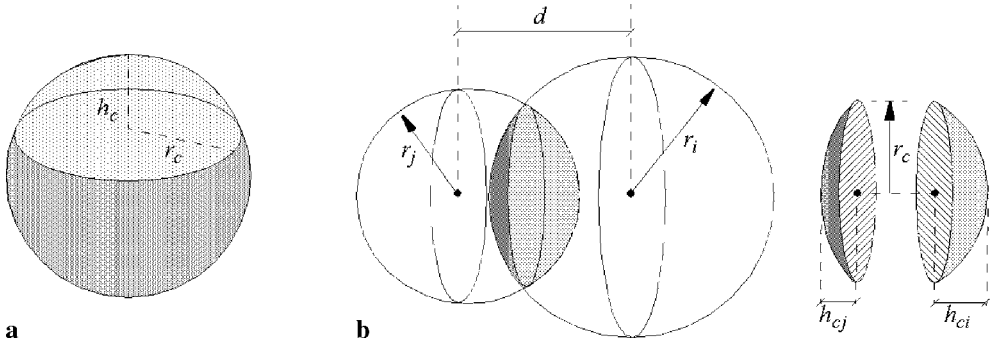

Fig. 3. a Portion of a spherical cap cut off by a plane. **b** Illustration of two spherical caps defining the volume of overlap (intersection) between two spheres

$Vol_{\text{overlap}(i,j)} = F_i V_{ci} + F_j V_{cj}$. The volume of a spherical cap is defined as:

$$V_{ci} = \frac{\pi}{6}\left(3r_c^2 + h_{ci}^2\right)h_{ci}, \tag{3}$$

where r_c is the radius of the cap's base and h_{ci} is the cap's height, Fig. 3a. The radius of the cap (circle of intersection between the two overlapped spheres) and the heights of the two caps depend on the distance d between the centers of the two spheres and their size. If the cap corresponding to the sphere of radius r_i has height h_{ci}, and the cap of the sphere of radius r_j has height h_{cj}, Fig. 3b, the radius of the common circle r_c is computed as follows:

$$r_c = \frac{\sqrt{\left((r_i + r_j)^2 - d^2\right)\left(d^2 - (r_i - r_j)^2\right)}}{2d}, \tag{4}$$

where h_{ci} and h_{cj} are defined as: $h_{ci} = \frac{r_j^2 - (r_i - d)^2}{2d}$, $h_{cj} = \frac{r_i^2 - (r_j - d)^2}{2d}$; V_{ci} and V_{cj} are respectively computed from (h_{ci}, r_c) and (h_{cj}, r_c) by applying Eq. 3.

Besides taking into account the previous filling factor, the proposed cost function also considers the size of the bounding spheres, S_b, in order to enforce a monotonous growth of the groups of spheres. Let r_b be the radius of the smallest bounding sphere, which contains all the spheres from ζ. The proposed cost function is defined as:

$$C_b = r_b^3 / F_b \tag{5}$$

with F_b computed according to Eq. 2. The smaller and closer the spheres are, the lower their grouping cost will be. Intuitively, spheres with small merging costs should tend to be grouped together as they would denote close objects in the scene. This process is intended to create a hierarchy in which leaves, which represent the original objects, are progressively grouped until the scene's bounding sphere at the root of the tree is obtained. The proposed strategy favors that small clusters are created first, as well as a progressive growth of the radius of the resulting spheres when the hierarchy's level is reduced. By using a more realistic filling factor, objects are grouped in a more intuitive way than in [21]. Finally, merging spheres with lowest cost Eq. 5 involves maximizing the corresponding filling factor; in other words, this strategy mini-

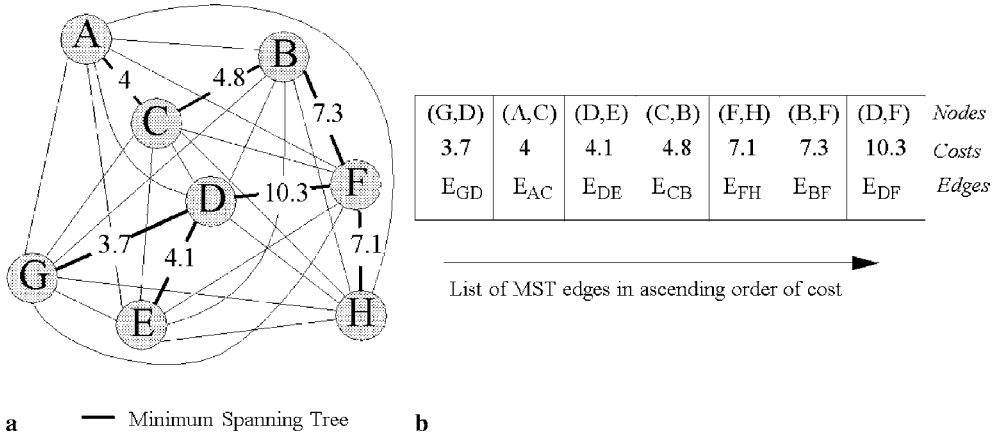

Fig. 4. a Example of adjacency graph and its MST. **b** List of the MST's edges sorted in ascending order of cost

mizes the empty volume and, hence, the total volume. This minimizes the probability of intersecting other spheres as suggested in [9].

3.2 Minimum spanning tree generation

Once an adjacency graph has been created, the next objective is the computation of its minimum spanning tree. The minimum spanning tree (MST) of a graph G is the acyclic subgraph of G that contains all the nodes of G and such that the sum of the grouping costs Eq. 5 associated with its edges is minimum. The MST of a graph with M edges and N nodes can be efficiently computed in $O(M \log N)$ by applying *Kruskal's algorithm* [19]. In our case, the cost is $O(N^2 \log N)$. Figure 4a shows an example of a fully-connected adjacency graph and its MST – notice that cost values associated with the different edges are illustrative but not computed with the proposed cost function. Figure 4b depicts the list of the MST's edges sorted in ascending order of cost.

An advantage of using MSTs for finding the best connectivity among the nodes of the graph (spheres) is that user-defined parameters are avoided. Alternatively, finding the best connectivity with a technique such as the k-nearest neighbors, although similar in philosophy, could produce a different result since disconnected subgraphs could be obtained instead of a single tree connecting every single node. Finally, thresholds such as the number of connections per node or the maximum distance should be defined by the user when the k-nearest neighbors technique is used.

3.3 Node clustering and new level generation

The outcome of the previous stage is a list of edges sorted in ascending order according to their associated cost. The current stage groups the nodes linked by those edges by applying a merging criterion (Fig. 5). A new level of the hierarchy is then created.

In the example given in Fig. 4, the nodes linked by the first edge in the sorted list (G and D) are initially merged, defining thus a new *candidate* object for the new level. This candidate will become a new object in the new level in case that no new nodes are merged to this first cluster. Otherwise, if a new node is merged, that initial candidate object will be removed and a new candidate computed by merging the three nodes: the two initially merged nodes plus the new added node. The clustering algorithm proceeds by testing the merging criterion described in Fig. 5 upon every edge from the sorted list. In the end, all graph nodes will have been eventually grouped into new objects and the clustering stage at the current level concludes.

The next level of the hierarchy is then created with the new objects (defined by their corresponding bounding spheres and filling factors) and the ones that could not be merged at the current level. In case the new level contains two or more objects, the process starts over from the adjacency graph generation stage (Sect. 3.1).

Input Information: Candidate Edge : E_{ij}
Linked nodes : (i, j)
Cost of linking nodes i and j: C_{ij}
Nodes previously clustered with i, if any: Σ_i
Nodes previously clustered with j, if any: Σ_j

Pseudo-code:
if ($\Sigma_i \neq \emptyset$ or $\Sigma_j \neq \emptyset$), then
  define the set of candidate nodes to be merged as: $\Theta = \Sigma_i \cup \Sigma_j$
  compute:
    Bounding Sphere $(\Theta) \Rightarrow S_\Theta$
    Filling Factor $(\Theta) \Rightarrow F_\Theta$
    Cost $(\Theta) \Rightarrow C_\Theta$
  if ($C_\Theta \leq C_{ij}$)
    Create New Object Θ (nodes contained in Θ are grouped)
    Remove Objects generated by previous groupings among nodes contained in Θ
  else
    Discard Edge E_{ij}
else
  Create New Object(nodes i, j are grouped)
    $\Rightarrow$ Bounding Sphere: S_{ij}, Filling Factor: F_{ij}, List of Children: $\{i,j\}$

Fig. 5. Pseudo-code of the merging criterion used to group two nodes at the same level. This is applied to every edge of the current MST

Figure 6 shows an illustration of the final hierarchical representation after applying the proposed algorithm to the example presented in Fig. 4. An object in a new level can be defined by a binary merge, an n-ary merge, or a single object from the previous level. Hence, in the worst case, where every new object in the hierarchy is defined by a binary merging, the depth of the tree generated by the proposed technique is the same as a binary tree.

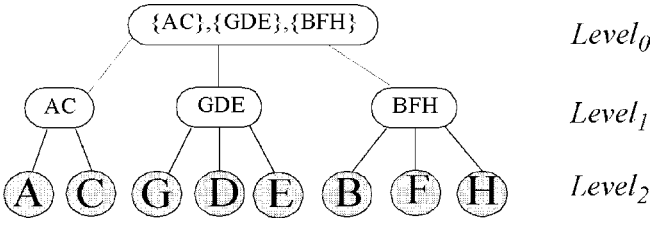

Fig. 6. Final hierarchical representation from the MST and sorted list shown in Fig. 4

4 Experimental results

The proposed technique has been tested upon several 3D scenes and compared to the technique presented in [21]. Figure 7(*top*) shows a scene with 24 objects (the corresponding *vrml* model can be obtained by contacting the authors). Each object is represented by its bounding sphere. The computed hierarchical representation, pre-

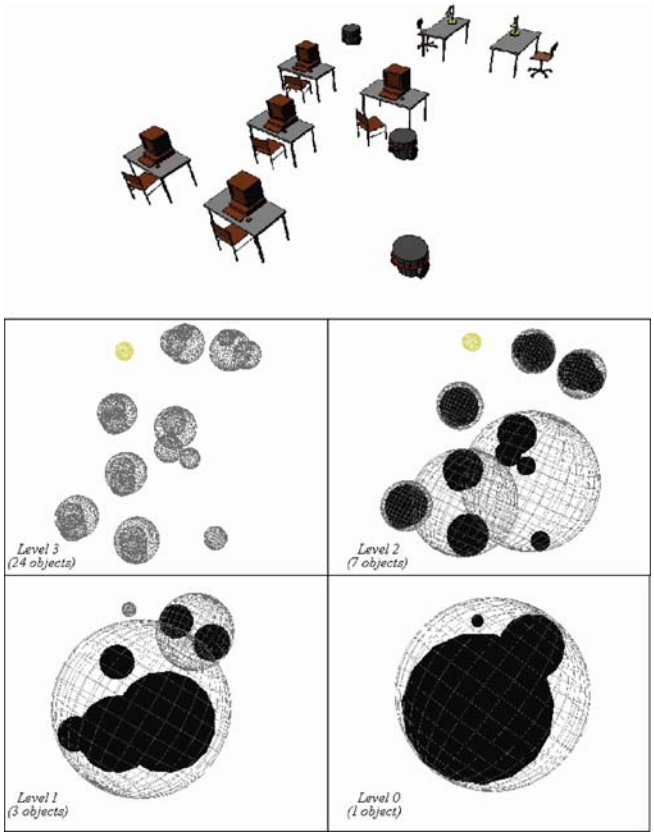

Fig. 7. (*top*) Original scene defined by 24 objects (i.e., spheres). (*bottom*) Levels of the computed hierarchical representation (dark solid spheres correspond to objects merged at the previous level; objects merged at the current level are represented in dark wireframe)

sented in Fig. 7(*bottom*), contains four levels. Dark solid spheres correspond to objects merged at the previous level. Dark wireframe spheres represent objects that will be merged at the current level. Finally, light wireframe spheres represent non-merged objects at that level. The four levels are represented at different scales. They were computed in 3.85 s on a 3.2 GHz Pentium IV PC. Note that more perceptual groupings could be obtained at the different levels by using different node clustering criteria (Fig. 5) (e.g., by adding constraints in the size of the resulting spheres).

In order to better appreciate the clustering capabilities of the proposed technique, three different scenes have been tested. These scenes contain six randomly placed spheres over a plane, each with a random size. The computed hierarchies are presented in Fig. 8. The different levels are represented at different scales. As it was expected, the hierarchies in the three examples are obtained by clustering the objects at the different levels following a kind of perceptual grouping. Note that although these scenes are defined by a few objects, they could be understood as a local grouping of a bigger scene.

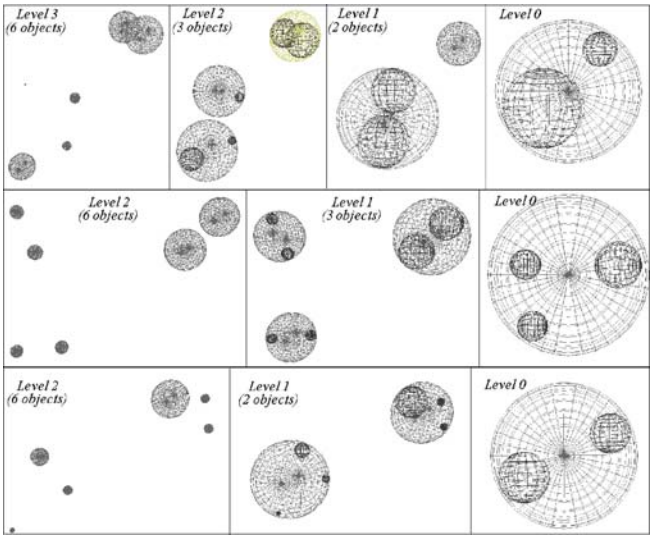

Fig. 8. Hierarchical representations corresponding to three different scenes containing six objects randomly placed with a random size. Notice that although the object size and distribution is different in these scenes, the hierarchies computed with the proposed technique cluster the objects as it could be expected, i.e., a kind of *perceptual grouping* is obtained

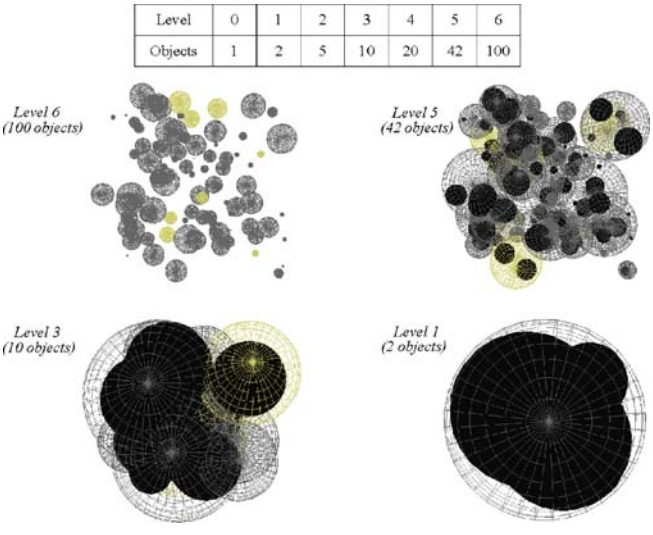

Fig. 9. Hierarchical representation of a scene defined by 100 objects randomly placed with a random size (dark solid spheres correspond to objects merged at the previous level; objects merged at the current level are represented in dark wireframe)

Figure 9(*top*) presents the resulting levels of a scene that contains 100 elements constituted by randomly placed spheres of random size. Figure 9(*bottom*) shows some of the generated levels. The hierarchical representation of this synthetic scene has 7 levels that were computed in 18.96 s.

4.1 Comparison

The proposed technique has been compared with both a version of the algorithm presented by Xavier [21] and the technique presented in [7]; no careful tuning of the user-defined parameters required by [21] has been done and no space partitioning has been considered either in [21] or in the proposed technique. The reason for the former is that no hints were given in [21] for deciding the values of those parameters in order to obtain optimal performance. The motivation for not applying space partitioning is that the performance achieved with it is strongly related to the choice of the aforementioned user-defined parameters. Since no optimal parameter tuning can be granted in general, the space partition may not help improve efficiency and may even lead to performance degradation owing to the overload due to the generation and management of the data structure.

Qualitatively, the algorithm presented in [21] produces results comparable to the ones obtained with the proposed algorithm. However, every time a new cluster is created, Xavier's algorithm recomputes the costs between the bounding volume of that cluster and the other objects of the scene (both individual objects and already existing clusters). In order to speed-up the process, a dynamic threshold that determines a maximum accepted size is kept. All the objects whose size is larger than that threshold are discarded from the computation of the cost function at that step. This dynamic threshold is progressively increased. In the worst case, where all the objects have the same size, this dynamic threshold does not discard any objects. Moreover, every time a pair of objects is grouped, the cost between the new object and the previous ones must be recomputed and the new object must be inserted in the sorted list of candidate object pairs. The cost of this iterative process is $O(N^3)$.

Conversely, the most expensive part of the proposed algorithm is the computation of costs between all the objects of the scene, but this stage is applied over a decreasing number of objects (the decreasing rate is considerably higher than Xavier's algorithm, where the cost needs to be recomputed after every single merging). The remaining stages are very efficient, leading to a better performance of the whole process; in the worst case (only binary clusters) the cost of the proposed technique is $O(N^2 \log N)$. This cost reduction, together with the improvement in the filling factor function and the lack of user-tuned parameters make the proposed technique particularly attractive in front of the one proposed in [21]. Obviously, a careful tuning of parameters and the application of space partitioning would improve the performance of Xavier's technique, but optimal tuning may be difficult to achieve in practical situations.

The proposed algorithm is also advantageous with respect to [7]. Although comparable representations were obtained for scenes containing few objects, the hierarchical representations, as well as the CPU times, are considerably better with the proposed algorithm when complex and large scenes are considered. These advantages are emphasized when a particular scene containing equally-sized objects, regularly distributed over the space, is considered; actually, neither [21] nor [7] can appropriately handle this situation.

The advantages of the proposed algorithm with respect to [7] are due to the combination of two reasons: On the one hand, the new strategy used for clustering objects, which is applied at every iteration over the MST corresponding to the current adjacency graph, helps obtain a more realistic grouping. In other words, the corresponding object distribution at every iteration is taken into account for clustering them at the current level. Recall that in [7], the information related to the object geometry and distribution was only considered in the first stage. Therefore the algorithm proceeds by merging the objects in further levels without updating the information related to the object geometry and distribution. This lack of update leads to wrong results in those scenes containing large amounts of objects. On the other hand, the proposed cost function, as well as the use of the filling factor for every sphere through the whole structure, helps to obtain balanced hierarchical representations. Notice that in [7], no filling information was taken into account, and was not preserved during the hierarchy generation process.

4.2 Object-oriented versus space-oriented hierarchical representations

The proposed object-oriented technique has been compared with two space-oriented hierarchical representations respectively based on octrees and kd-trees (see illustration in Fig. 10) in terms of a practical application: find out whether a segment defined by two points randomly placed in space intersects or not with the computed hierarchy. In case the root's sphere is intersected, an iterative process is applied until either no sphere is intersected by the segment or all the intersected spheres are leaves of the tree. In both cases, the algorithm returns the nearest sphere to that segment and the total number of checked spheres (iterations). In order to compute an average number of iterations, the aforementioned process is repeated 500 times, every time with a different segment.

Although the tested space-oriented hierarchical representations are built in a top-down fashion, respectively following an octree and a kd-tree, both are finally represented by means of a hierarchy of spheres bounding the leaves and internal nodes in order to be able to make a direct comparison with the proposed technique.

Two series of scenes were used for the comparisons; they were defined by sets of spheres ranging from 50 to 500 in the first case and from 50 to 3000 in the second case. The first one consists of scenes defined by randomly distributed spheres, Fig. 11a, while, in the second series,

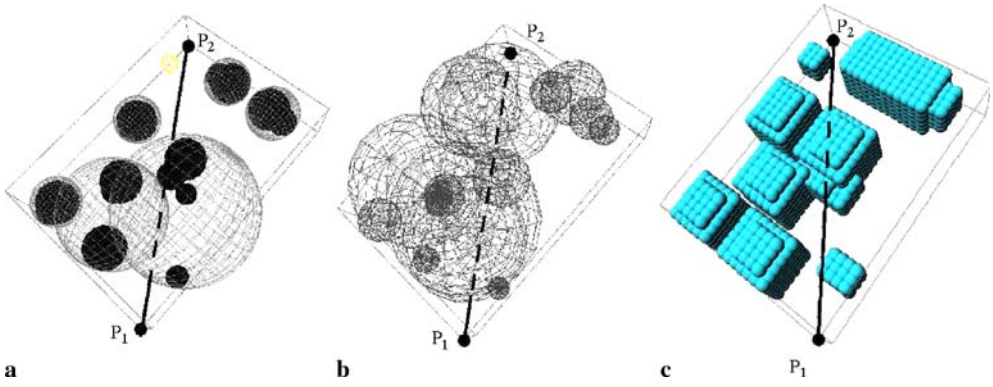

Fig. 10. a Second level of an object-oriented representation of a scene defined by 24 objects (points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ were randomly placed over opposite faces of the bounding box). **b** The fourth level of a kd-tree representation of the same scene. **c** Fourth level of the corresponding octree representation

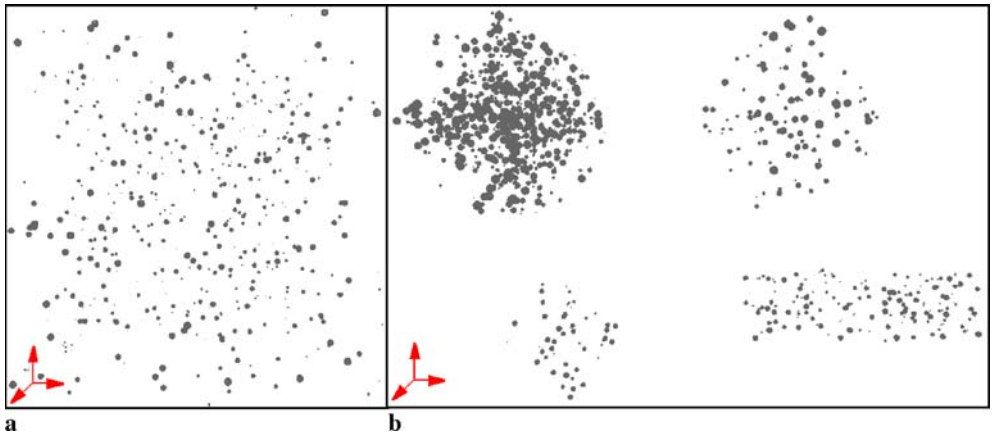

Fig. 11. a Example of a scene defined by a set of 500 randomly distributed spheres. **b** Scene defined by four clusters containing different amounts of randomly distributed spheres, 500 spheres in total

spheres are distributed into four clusters, such that every cluster contains different amounts of randomly distributed spheres: 5% of the total into the first cluster, and 11.5%, 21.5% and 62% into the second, third and fourth clusters, respectively, Fig. 11b. In both cases, for every scene, 500 random segments were used for computing the average number of iterations and CPU time. As indicated before, these random segments are defined by two random points moving over opposite faces of the bounding box. The pair of considered faces are the ones with the smallest area.

The octree-based representation is recursively expanded until its leaves are one third the size of the smallest input sphere. The kd-tree-based representation is generated by partitioning the input space with planes (see [1] for more details). This set of input spheres is split into two subsets with a vertical plane, YZ, through the mean x coordinate of the aforementioned set of spheres' center points. A recursive procedure is then applied over the resulting subsets of spheres by first partitioning them according to the mean of their y coordinates (XZ plane) and then by the mean of their z coordinates (XY plane). This procedure starts again over the x coordinates and stops when every subset contains a single sphere. Although planes were used for partitioning the given scene, the final representation contains spheres at the leaves as well as at internal nodes. As mentioned above, leaf's spheres are used to represent every single input object whereas spheres associated with internal nodes are defined as the minimum ball bounding all the descendants from that node. This hierarchy of spheres is used for comparing the three techniques.

In order to determine the closest sphere to a segment, a search algorithm explores the given hierarchical structure (octree, kd-tree or proposed technique) starting from its root. At every iteration, those spheres intersected by the given segment are removed and the distances from the segment to their children are computed. This iterative process is applied until either no sphere is intersected by the segment or all the intersected spheres are leaves of the tree. In both cases, after finishing this iterative process, the near-

est sphere and the amount of iterations (number of spheres checked during the process) are returned as a result.

Table 1 presents the average number of iterations necessary to find the nearest sphere to 500 random segments when scenes containing randomly distributed spheres are considered. The difference between the number of iterations required by the proposed technique and the ones required by octrees and kd-trees are respectively given as a percentage. Finally, the average CPU time in milliseconds is provided. On the other hand, Table 2 presents the results when scenes containing several clusters of objects are considered. As mentioned above, four clusters with different amounts of randomly distributed objects are presented.

In almost all cases, the number of iterations required to find the nearest sphere to a segment is smaller with the proposed technique than with octrees and kd-trees. Improvements with respect to octrees range from two to eleven times when structured scenes are processed. Alternatively, when scenes containing randomly distributed objects are considered, improvements range from four to six times. Regarding kd-trees, the proposed technique behaves slightly better when a scene with randomly distributed objects is considered. However, when structured scenes were considered, the proposed technique requires up to 4.8 times fewer iterations than kd-trees. This can be easily explained since kd-trees do not take explicitly into account the spatial distribution of the objects contained in the scene, since kd-tree's partitioning planes are determined according to the mean of the spheres' coordinate corresponding to the current iteration. This drawback of kd-trees can also be appreciated when kd-trees are compared to octrees (for instance scenes containing 1250 and 1750 objects).

Finally, regarding the time to generate the hierarchical structures, octrees and kd-trees are generated considerably faster than the proposed technique as expected. This is due to the fact that both octrees and kd-trees do not take into account the spatial relationship among the objects in the scene as their purpose is to simply partition the input space and not to generate a hierarchical grouping that preserves the topological structure of the objects contained in the scene at different abstraction levels. Thus, in the worst case of a scene defined by 3000 spheres, Fig. 11b, the proposed technique generated the hierarchy in 659 s, while the corresponding kd-tree was generated in 19.84 s and the octree in 1.16 s. This implies that the proposed technique is advantageous over kd-trees and octrees when the hierarchy is generated once and utilized many times, which is the typical situation in practical applications, and also when the time to query the hierarchy must be kept as low as possible, for example, in real time applications. In the

Table 1. Average number of iterations and CPU time in milliseconds necessary to determine the nearest sphere to 500 different random segments, by considering space-oriented hierarchies (octree and kd-tree) and a hierarchical representation computed with the proposed technique. The scenes are defined by randomly distributed spheres, Fig. 11a. In the kd-tree and octree rows, the increase percentage of their average number of iterations with respect to the proposed technique is given

Scene's spheres		50	100	150	200	250	300	350	400	450	500
Prop. tech.	Avg. Iterat.	28	47	49	57	69	83	83	88	94	101
	Avg. CPU time	0.023	0.035	0.038	0.04	0.045	0.055	0.055	0.06	0.065	0.068
Kd-tree	Avg. Iterat.	31	43	54	64	71	84	91	102	107	112
	Difference (%)	+10.7	−9.3	+10.2	+12.2	+2.8	+1.2	+9.6	+15.9	+13.8	+10.8
	Avg. CPU time	0.025	0.032	0.04	0.047	0.05	0.058	0.065	0.07	0.075	0.078
Octree	Avg. Iterat.	196	258	319	354	388	428	448	468	483	494
	Difference (%)	+600	+448	+551	+521	+462	+415	+439	+431	+413	+389
	Avg. CPU time	0.138	0.117	0.215	0.235	0.262	0.29	0.303	0.315	0.303	0.333

Table 2. Average number of iterations and CPU time in milliseconds for structured scenes: four clusters of spheres as illustrated in Fig. 11b. The spheres contained in every cluster are randomly distributed. In the kd-tree and octree rows, the increase percentage of their average number of iterations with respect to the proposed technique is given

Scene's spheres		50	250	500	750	1000	1250	1500	1750	2000	2250	2500	2750	3000
Prop. tech.	Avg. Iterat.	25	53	75	73	98	120	159	110	92	91	95	110	98
	Avg. CPU time	0.013	0.018	0.03	0.03	0.038	0.045	0.06	0.047	0.047	0.04	0.188	0.052	0.05
Kd-tree	Avg. Iterat.	22	64	96	131	201	345	295	359	359	431	490	571	573
	Difference(%)	−13	+20.7	+28	+79.4	+105	+187	+85.5	+226	+290	+373	+415	+419	+484
	Avg. CPU time	0.01	0.03	0.038	0.055	0.075	0.09	0.112	0.132	0.142	0.17	0.192	0.22	0.225
Octree	Avg. Iterat.	143	145	260	385	340	251	351	249	523	1087	1191	554	596
	Difference (%)	+472	+173	+246	+427	+247	+109	+120	+126	+468	+1094	+1153	+403	+508
	Avg. CPU time	0.087	0.13	0.775	1.09	1.45	1.758	2.115	2.415	2.685	3.087	3.48	3.453	3.803

latter case, the hierarchy can be built off-line. Therefore, the construction time in this situation would be unimportant.

5 Conclusions and further improvements

A new technique for generating a hierarchical representation of the objects contained in a 3D scene has been presented. It is assumed that these objects are represented by single bounding spheres or, alternatively, in case that such spheres are not tight enough to the objects, by sets of spheres as suggested in [13, 17]. The proposed technique consists of three stages. The first stage computes the cost of grouping all pairs of input spheres and generates an adjacency graph with those costs. In the second stage, the minimum spanning tree (MST) of that graph is generated. In the last stage, a merging criterion is used to select the set of spheres to be merged at that level. These stages are applied until a level with a single sphere is reached.

The proposed technique has shown better performance than space-oriented approaches (i.e., octree and kd-tree-based representations) for both randomly distributed objects and structured scenes. For example, checking intersections in hierarchies generated with the proposed technique requires up to 4.8 times fewer iterations than when they are generated with kd-trees. This reduction rises to eleven times for hierarchies generated by applying octree-based representations.

The proposed technique is more advantageous than [21] since: (1) efficiency is higher as the costs between the objects of the scene do not have to be computed after every new object generation but after every new level generation; (2) a more realistic filling factor considering volumes instead of the difference between diameters is used; (3) there are not user-tuned parameters; and (4) the generated tree is n-ary instead of binary and, thus, it is shallower. Finally, the proposed approach improves [7] by defining a new strategy for generating the hierarchy and by using a new cost function. This cost function uses a volume-based filling factor that helps to build balanced hierarchies; filling information is used for clustering objects at a given hierarchy. At the same time this information is preserved in order to be used in further levels.

Spheres are the most common bounding volume due to the simplicity both to represent them and to detect intersections among them. However, other bounding volumes have also been proposed in the literature, such as *k-dops* [9] and *ellipsoids* [18]. Further work will consist of extending the proposed technique to these alternative geometric primitives, which, at least, will require the definition of new cost functions. In addition, practical applications of the proposed 3D hierarchical representations as a means to speed-up path planning or collision avoidance algorithms will be developed.

References

1. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer, Berlin, Heidelberg, New York (2000)
2. Breen, D., Mauch, S., Whitaker, T.: 3D scan conversion of csg models into distance volumes. In: Proceedings of IEEE International Symposium on Volume Visualization, North Carolina, USA, pp. 7–14 (1998)
3. Brunet, P., Navazo, I.: Solid representation and operation using extended octrees. ACM Trans. Graph. **9**(2), 170–197 (1990)
4. Fernández, J., González, J.: Hierarchical graph search for mobile robot path planning. In: Proceedings of IEEE International Conference on Robotics and Automation, Leuven, Belgium, pp. 656–661 (1998)
5. Fernández, J., González, J.: Multihierarchical graph search. IEEE Trans. Pattern Anal. Mach. Intell. **24**(1), 103–113 (2002)
6. Fischer, K., Gärtner, B., Kutz, M.: Fast smallest-enclosing-ball computation in high dimensions. In: Proceedings of Annual European Symposium on Algorithms No. 11, Budapest, Hungary. LNCS vol. 2832, pp. 630–641 (2003)
7. Garcia, M., Sappa, A., Basañez, L.: Efficient generation of object hierarchies from 3d scenes. In: Proceedings of IEEE International Conference on Robotics and Automation, Detroit, Michigan, USA, pp. 1359–1364 (1999)
8. Goldsmith, J., Salmon, J.: Automatic creation of object hierarchies for ray tracing. IEEE Comput. Graph. Appl. **7**, 14–20 (1987)
9. Gottschalk, S., Lin, M., Manocha, D.: Obb-tree: A hierarchical structure for rapid interface detection. In: Proceedings of SIGGRAPH 96, New Orleans, LA, pp. 171–180 (1996)
10. Havran, V., Bittner, J.: On improving kd-trees for ray shooting. J. WSCG **10**(1), 209–217 (2002)
11. Huerta, J., Chover, M., Quiros, R., Vivo, R., Ribelles, J.: Binary space partitioning trees: a multiresolution approach. In: Proceedings of IEEE International Conference on Information Visualization, London, UK, pp. 148–154 (1997)
12. Klosowski, J., Held, M., Mitchell, J., Sowizral, H., Zikan, K.: Efficient collision detection using bounding volume hierarchies of k-dops. IEEE Trans. Visual. Comput. Graph. **4**(1), 21–36 (1998)
13. Martínez-Salvador, B., del Pobil, A., Pérez-Francisco, M.: Very fast collision detection for practical motion planning Part i: The spatial representation. In: Proceedings of IEEE International Conference on Robotics and Automation, Leuven, Belgium, pp. 624–629 (1998)
14. Ruiz de Miras, J., Feito, F.: Direct and robust voxelization and polygonization of free-form csg solids. In: Proceedings of IEEE International Symposium on 3D Data Processing Visualization and Transmission, Padova, Italy, pp. 352–355 (2002)
15. O'Sullivan, C., Dingliana, J.: Real time collision detection and response using sphere trees. In: Proceedings of 15th Spring Conference in Computer Graphics, Bratislava, Slovakia, pp. 83–92 (1999)
16. Payeur, P., Laurendeau, D., Gosselin, C.: Range data merging for probabilistic octree modeling of 3D workspaces. In: Proceedings of IEEE International Conference on Robotics and Automation, Leuven, Belgium, pp. 3071–3078 (1998)
17. del Pobil, A., Serna, M., Llovet, J.: A new representation for collision avoidance and detection. In: Proceedings of IEEE International Conference on Robotics and Automation, Nice, France, pp. 246–251 (1992)

18. Rimon, E., Boyd, S.: Obstacle collision detection using best ellipsoid fit. Intell. Robot. Syst. **18**(2), 105–126 (1997)
19. Rosen, K.: Discrete Mathematics and its Applications, 2nd edn. McGraw-Hill, New York (1990)
20. Subramanian, K., Fussell, D.: Automatic termination criteria for ray tracing hierarchies. In: Proceedings of Graphics Interface, Calgary, Alberta, Canada, pp. 93–100 (1991)
21. Xavier, P.: A generic algorithm for constructing hierarchical representations of geometric objects. In: Proceedings of IEEE International Conference on Robotics and Automation, Minnesota, USA, pp. 3644–3651 (1996)

ANGEL DOMINGO SAPPA received his degree in Electro-mechanical Engineering in 1995 from the National University of La Pampa, General Pico-La Pampa, Argentina, and doctorate degree in Industrial Engineering in 1999 from the Polytechnic University of Catalonia, Barcelona, Spain. From 1999 to 2002 he undertook a post-doctorate research position at LAAS-CNRS, Toulouse, France and at Z+F UK Ltd., Manchester, UK. From September 2002 to August 2003 he was with the Informatics and Telematics Institute, Thessaloniki, Greece, as a Marie Curie Research Fellow. Since September 2003 he has been with the Computer Vision Center, Barcelona, Spain, as a Ramón y Cajal Research Fellow. His research interests are focused on range image analysis, 3D modeling and model-based segmentation.

MIGUEL ANGEL GARCIA received his B.S., M.S., and Ph.D. degrees in Computer Science from the Polytechnic University of Catalonia, Barcelona, Spain, in 1989, 1991, and 1996, respectively. He joined the Department of Software at Polytechnic University of Catalonia in 1996 as an Assistant Professor. In 1997, he joined the Department of Computer Science and Mathematics, Rovira i Virgili University, Tarragona, Spain as an Associate Professor, being Head of Intelligent Robotics and Computer Vision Group. In 2006, he joined the Department of Informatics Engineering at Autonomous University of Madrid, where he is currently Associate Professor. His research interests include image processing, 3D modeling, and mobile robotics.