Efficient Collision and Self-Collision Detection for Humanoids Based on Sphere Trees Hierarchies

Klaus Steinbach[1,2] James Kuffner[1] Tamim Asfour[2] Ruediger Dillmann[2]

[1]*The Robotics Institute*
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA, 15213, USA
{ksteinba,kuffner}@cs.cmu.edu

[2]*Institute of Computer Science and Engineering*
University of Karlsruhe
Haid-und-Neu-Straße 7, 76131
Karlsruhe, Germany

***Abstract*— We present an algorithm for computing compact sphere tree hierarchies over general 3D polygonal meshes for the purpose of efficient collision detection and minimum distance computation. The sphere tree construction method we use optimizes the location of the sphere centers and radii at each level of the hierarchy to compactly contain the underlying geometry. We enhance the heirarchy construction so that it can be applied to both self-collision and obstacle collision detection for articulated robots. Finally, we introduce distance range bounds that is effective for minimizing the overall number of required distance computations. The result is a faster algorithm that performs particularly well on collections of multiple mesh objects.**

I. Introduction

Safety and reliability are crucial for humanoid robots expected to operate in home or office environments. Specifically, potential collisions between the robot and its environment must be efficiently detected and avoided. We are developing a complete robotic system for autonomous manipulation tasks in a kitchen environment. In this context, the detection and prevention of collisions and self-collisions is an important issue. The prototype ARMAR (Fig. 1) is a humanoid platform with 23 degrees of freedom and consists of five subsystems: the head, left arm, right arm, torso and a mobile base [1].

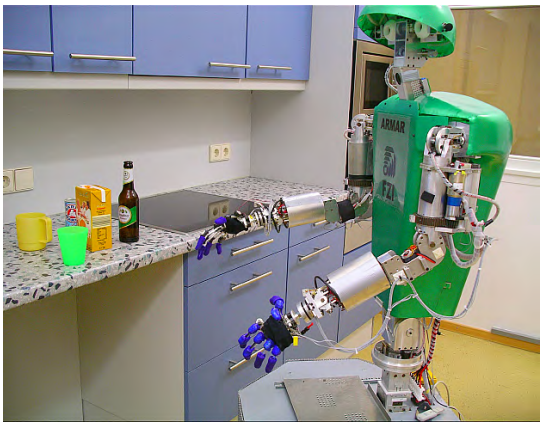


Fig. 1. The Humanoid Robot ARMAR

In the broader sense, collision detection and minimum distance computation between 3D objects is an important problem with applications in robotics, computational geometry, computer graphics, haptics and simulation. Both operations are closely related and can be generalized as two instances of geometric proximity queries. Specifically, *collision detection* typically involves testing for overlap between two or more volumetric or surface models of objects, and returning a binary result. If the objects are disjoint, the *minimum separating distance* between pairs of objects may additionally be returned. In either case, information concerning the location and nature of the proximity event (contact points, contact normals, amount of overlap, pairs of closest points, minimum separating distances, etc.) may also be needed. For a general overview of collision detection and a survey of various algorithms, see [2]–[5]. An important application of collision detection algorithms in robotics is for motion planning, where minimum distance information may be used to maintain clearance between the robot and nearby obstacles as in [6], [7].

As the number of geometric primitives involved in a query become large, hierarchical bounding volumes are often used to spatially partition the objects. Several hierarchical approaches have been proposed with different bounding volumes, each with advantages and disadvantages. Typical examples include axis-aligned bounding boxes (AABBs) [8], object-oriented bounding boxes (OOBBs) [9], [10], spheres [11]–[15], ellipsoids [16], swept-sphere volumes [17], and convex hulls [18]–[20]. The choice of bounding volume results in different tradeoffs between computational efficiency, complexity, and the tightness of the fit to the underlying geometry. Sphere bounding volumes are a popular choice due to their simplicity and computational efficiency due to rotation invariance. Sphere tree hierarchies are simple to implement and suitable for efficient minimum distance computation as in Quinlan's formulation [21], [22]. Relatively few techniques focus on detecting self-collision. For example, convex hulls and pruning techniques to reduce computation complexity by analyzing the kinematic structure of the articulated robots was used in [23]. Other work focused specifically on long kinematic chains [24].

In this paper we present an improved algorithm for collision detection and minimum distance computation between arbitrary triangle meshes in 3-D. Our method works particularly well for articulated kinematic structures represented as collections of rigid links. We expand the hierarchical structure

and introduce an approximate *range of motion hull* for each rigid link and for entire kinematic subchains. The range of motion hull encodes the kinematic *swept volume* that we have found to be very useful for efficiently detecting self-collisions for articulated linkages using the improved hierarchical sphere tree structure.

II. Improved Minimum Distance Computation for Nonconvex Objects

Our approach is based on the hierarchical sphere trees of Quinlan [21], [22], which we will assume the reader is familiar with. We selected Quinlan's method because of its simplicity and applicability to arbitrary nonconvex meshes and "polygon soups". We also wish to obtain not just a binary result for collision detection, but rather some approximate measure of the minimum separating distance. Quinlan's algorithm conveniently allows the user to set an error tolerance on the computed minimum distance value that can speed up the computation for cases where the exact minimum distance information is not needed. For our humanoid robot application, we adjust the error tolerance according to the task requirements, and propose several modifications to Quinlan's approach that improves the overall efficiency and running time for moving articulated kinematic structures.

A. Distance Range Interval

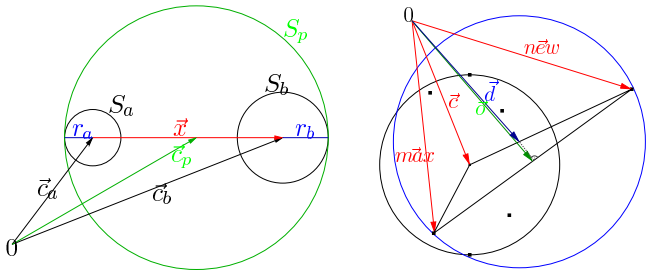
We define the distance range interval $I = [min_d, max_d]$ to be a user-selected range of object distances of interest to the application. For example, when validating a candidate motion trajectory for a serial chain manipulator, the largest possible distance that any point on the robot can move from the start configuration is a good candidate for max_d. In this case, it is unnecessary to compute the exact distance to objects which are further away than max_d. Similarly, due to uncertainty in control, the manipulator should maintain a safety distance margin to all obstacles when moving. This safety margin is a good candidate for the value of min_d, as it is unnecessary to compute a more precise minimum distance value to obstacles which have already been shown to penetrate the safety margin.

The primary advantage of a well-chosen distance range interval is that Quinlan's algorithm can be tuned to easily avoid unnecessary distance computations. When the algorithm is initialized, the variable d which keeps track of the upper bound on the minimum distance between two query objects should not be set to infinity as in [21], [22], but with max_d. Similarly, rather than reporting a collision whenever $d \leq 0$, we can simply terminate whenever $d \leq min_d$.

Another advantage of utilizing the distance range interval appears when considering a sequence of queries between multiple pairs of objects. In this case, max_d can be initialized to the current minimum distance for all objects tested so far. Whenever a subsequent object is shown to be beyond max_d, it can be skipped. Thus, it is beneficial to begin a sequence of queries with the nearest object, which can be heuristically selected if unknown a priori.

B. Sphere-Tree Hierarchy Construction

The key to the efficiency of Quinlan's algorithm is the hierarchical data structure of spheres used to approximate the underlying polygonal geometry of some arbitrarily-shaped object. To construct the hierarchy, Quinlan first creates an initial grid of small "leaf spheres" with fixed size is blanketed over the surface of every polygon [21], [22]. Nearby spheres are grouped together and combined to form a hierarchy from the bottom-up. The radii of larger intermediate spheres are calculated to entirely contain the smaller ones. For binary sphere trees, each intermediate sphere contains at most two child spheres. Finally, the root sphere of the tree will contain the entire collection and forms a bounding volume for the underlying geometry. Clearly, the value of the resulting root sphere radius heavily depends on the ordering and method used to combine child spheres. If we combine two spheres and insist on the fact that the child spheres be entirely contained within the parent sphere as in [21], [22], the minimal parent radius is straightforward to compute as shown in Figure 2(a). We will refer to this strategy as the *greedy* combine method.



(a) *Greedy* Combine Method (b) *Compact* Combine Method

Fig. 2. Illustration of the *greedy* and *compact* combine methods.

The condition that every parent sphere must entirely contain all child spheres is not necessary, since we are interested in the computation of the distance between the underlying object geometry. Thus, it is sufficient for the parent sphere to only completely contain all of the underlying geometry (i.e. polygon vertices) contained by its children. This fact is also realized and exploited in [25] for performing more efficient collision detection for haptics model display, and in [24] for efficient collision detection between long chains. We will refer to this as the *compact* method, which is illustrated in Figure 2(b). Various numerical optimization techniques can be used to search for the sphere center location that results in a minimal radius value. Figure II-B graphically illustrates the minimal radius value field for different candidate center locations (red being worse and green being better). The optimal compact sphere center in this case is coincident with the geometric center of the cube.

We can no longer assume that the current estimate of the maximum possible value of the minimum separation distance will monotonically decrease while descending the sphere hierarchy. Instead, the maximum distance should be updated only if it is strictly smaller than the current estimate. Intuitively, this

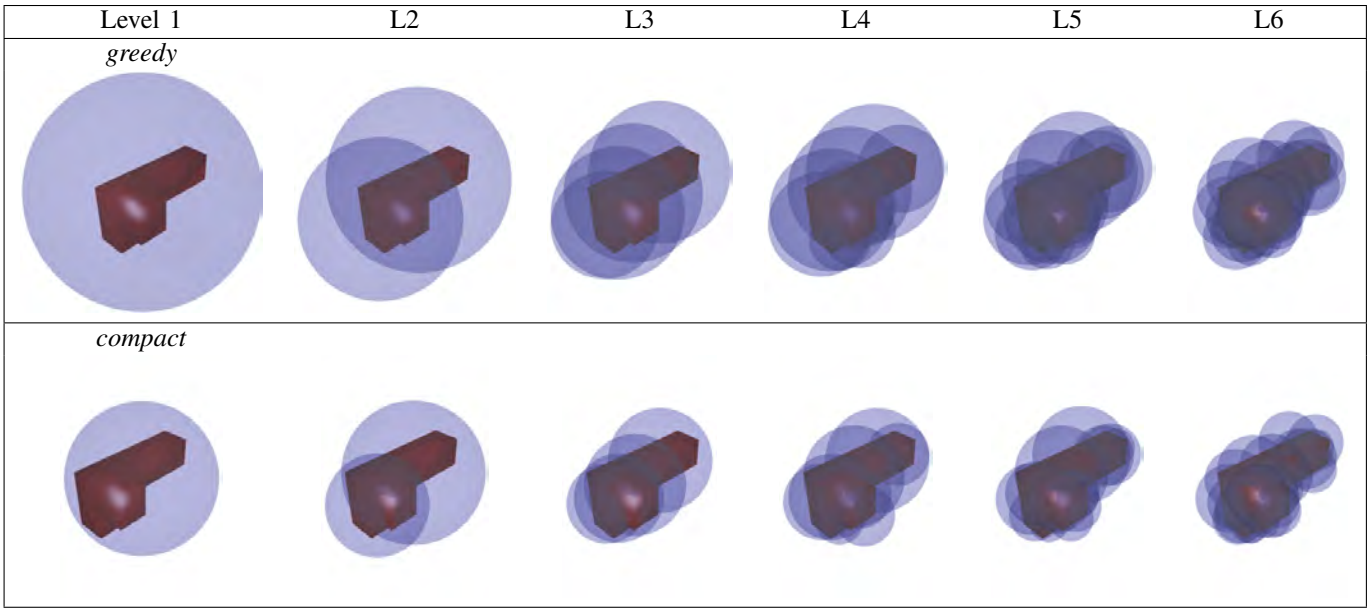


Fig. 4. Comparison of different levels of the sphere tree hierarchy for the *greedy* construction method versus our *compact* method for an L-shaped mesh.

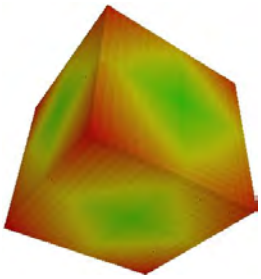


Fig. 3. The candidate *compact* sphere center locations for a cube colored according to the relative size of the smallest possible sphere radius containing the entire cube.

has the effect of maintaining a bound on the maximum value for the minimum separation distance between the intersection of the volumes of the parent and child spheres. Thus, if a child sphere includes any volume which the parent does not include, the algorithm uses only the part of the sphere inside the parent. Figure 4 shows a comparison of the relative sizes of different levels of the sphere tree hierarchy for a 3D L-shaped object for both the greedy and compact tree construction methods. Notice how each level of the compact hierarchy more compactly contains the underlying geometry.

C. Sphere Center Transformation Caching

Finally, we implemented an additional performance optimization to our algorithm by caching relative sphere transformations. Because sphere centers are stored in object local coordinates, for every distance computation between spheres, a transformation of one of the center points is needed. However, under certain circumstances this center point will be the same for some sequence of intermediate steps. For example, when processing subtrees for both a large and small object, the tree of the large object will be subdivided several times prior to that of the small object. Thus, the transformation of the small object sphere into the large object's coordinate system can be cached from the previous step and reused for future subdivisions. In our experiments, caching improved the overall runtime performance of a query by approximately 30% on average.

III. Application to Articulated Linkages

For applications involving serial chains or kinematic trees of rigid links such as humanoid robots, we propose constructing sphere-tree hierarchies that not only approximate the current configuration of the linkage but also their *range of motion*. The *range of motion hull* for a particular kinematic subtree can be intuitively thought of as the union of all possible spatial configurations. Another way to interpret it is the recursive computation and aggregation of the *swept volume* of all child links or subtrees to form a tight bound on the geometric reachability of an entire kinematic hierachy. This calculation needs to be done only once, and can be performed as a precomputation step provided that the valid joint angle ranges are known in advance. A compact sphere center is then computed that encloses the entire range of motion hull geometry.

The benefits of our approach are twofold. First, collision detection or minimum distance computation can be performed between two articulated linkages or a rigid and articulated structure *without* testing every pair of rigid parts against each other. Second, because the range of motion hull bounds the extent of the possible motion of the underlying subtree, validating motion trajectories for potential collisions can be performed very efficiently.

A. Hierarchy Construction for Articulated Linkages

Similar to the case for rigid objects, articulated structures require a "separation" and "combination" method for sphere-

tree construction. The separation method is tailored to the kinematic structure of the articulated object and combines several subtrees at every step considering the possible movements of the affiliated joints. The structure is built from the bottom-up as for rigid objects. Initially, each of the subtrees for each rigid link is constructed as described previously. During the combination method several spheres are merged to form a hierachy. The underlying geometry of the object or the underlying movement of the kinematic sub chain is used instead of the child sphere parameters to compute the parent sphere. Thus, the combination method cannot assume that only two child nodes must to be combined. However, our proposed method for rigid objects calculates sphere radii from point clouds and therefore does not make any assumptions regarding the number of nodes to merge.

To approximate the range of motion hull, the possible joint values for a particular link are sampled at a discrete resolution within the application-specific safety margin. At each subtree level, all rigid parts of the current kinematic subtree must be sampled along their affiliated joint ranges. Figure 5 illustrates the construction for a manipulator arm with three rigid links (L_1, L_2, and L_3). The rigid bounding sphere R_3 for link L_3 is calculated first. Then the joint motion for L_3 is sampled and the corresponding motion sphere M_3 is computed. Next R_2 is calculated and combined with M_3 to form the bounding sphere MR_2 based on the aggregation of their sampled points. In a similar fashion, M_2, R_1, and MR_1 are computed. MR_1 approximates all possible configurations of the kinematic chain relative to the local coordinate system of L_1.

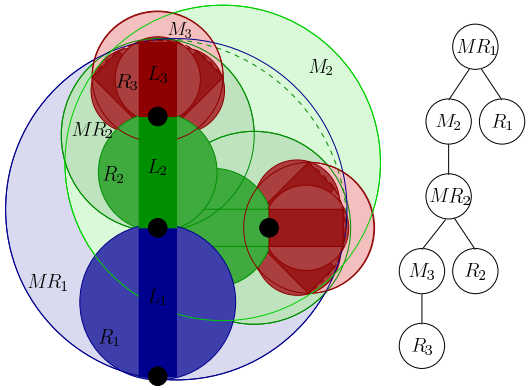


Fig. 5. The *Range of Motion Hull* for an articulated manipulator robot consisting of three rigid parts L_1, L_2, and L_3 (left) and the resulting sphere-tree.

B. Collision Detection

Our enhancement to the hierarchical sphere structure does not claim any conceptual change for the distance computation algorithm. The only thing which has to be considered is the amount of child nodes. Our approach successively tests the child nodes of a motion sphere (i.e. MR_2) with a sphere of another object instead of arranging them based on their minimum distance to the other object. The algorithm starts with the sphere for the rigid object (i.e. R_2) and then use the other motion spheres (i.e. M_3).

IV. Self-Collision Detection for Articulated Linkages

The self-collision detection algorithm makes one assumption, which requires neighboring links to be collision free. This is typically the case if proper joint limits are set for the mechanism. The algorithm works in three steps as illustrated graphically in Figure 6.

1) Collision detection of rigid link against children child nodes
2) Collision detection of child nodes against each other
3) Self-collision detection for child nodes

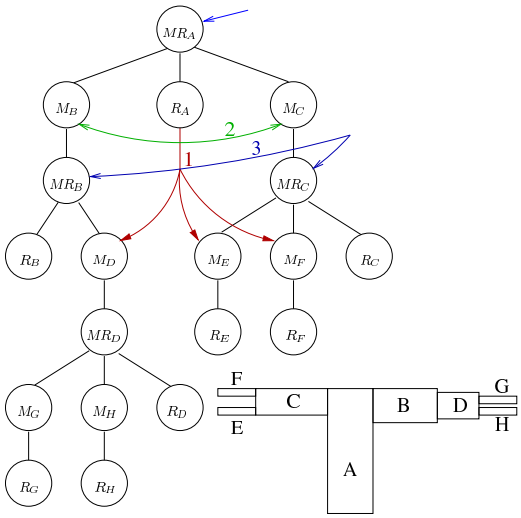


Fig. 6. Self-Collision detection steps for an example manipulator.

The first step checks for possible self-collision between each rigid link and its descendant nodes. Due to the fact that there cannot be any self-collision between neighboring links, each rigid link is tested against its "grandchildren" subtrees (all descendant nodes of its own children). For example, performing self-collision for the articulated structure in Figure 6 results in collision detection queries between R_A and M_D, M_E, and M_F as step one. The second step checks possible self-collision between the kinematic sub chains denoted by the child nodes. In Figure 6 collision detection between M_B and M_C is performed in step two. Finally, the self-collision detection routine is recursively called on each child node. In Figure 6 the recursive call must be done for MR_B and MR_C. During each step, the distance information is maintained and propagated, thereby lowering the upper bound on the overall minimum self-collision distance.

Pruning Collision Pairs: A precomputation step is performed that computes a lookup table of possible link pairs of the object which can possibly collide based on kinematic reachability. This allows us to avoid unnecessary distance computations during each of the steps of the query phase.

Because the self-collision algorithm not only detects collisions between rigid links against each other, but also against kinematic subtrees, additional lookup tables are computed. The *rigid* look up table denotes if a given rigid link of an articulated object can possibly collide with each kinematic sub chain. In Figure 6 this corresponds to the test in step one between R_A and M_D. For this example, if R_A and M_D are collision-free, then R_A and R_D, R_A and R_G, and R_A and R_H *must* be collision-free. An additional lookup table (*motion*) is calculated in a similar fashion only that its entries denote whether or not two motion spheres collide or not (see Equation 4). In Figure 6 this corresponds to the test between M_B and M_C. Finally, the lookup table (*scfree*) denotes if a motion node is self-collision free or not (see Equation 12). This lookup table is used in the third step of the algorithm and is a conjunction of all tests which the algorithm would perform. Mathematically, this can be described as follows:

$$base_{ij} = \begin{cases} \text{true}, & \text{range of motion hulls of } R_i \\ & \text{and } R_j \text{ are collision free} \\ \text{false}, & \text{otherwise} \end{cases} \quad (1)$$

$$parts_i = \{k \,|R_k \text{ is part of subchain of } M_i\} \quad (2)$$

$$rigid_{ij} = \bigwedge_{\forall k \in parts_j} base_{ik} \quad (3)$$

$$motion_{ij} = \bigwedge_{\forall (l,k) \in parts_i \times parts_j} base_{lk} \quad (4)$$

$$rC_i = k, R_k \text{ is child node of } MR_i \quad (5)$$

$$mC_i = \{k \,|M_k \text{ is child of } MR_i\} \quad (6)$$

$$mCC_i = \left\{ k \;\middle|\; \begin{array}{l} k \in mC_j \wedge l \in mC_i \wedge \\ MR_j \text{ is child of } M_l \end{array} \right\} \quad (7)$$

$$mrC_i = \{k \,|MR_k \text{ is child of } M_l \wedge l \in mC_i\} \quad (8)$$

$$r_i = \bigwedge_{k \in mCC_i} rigid_{rC_i k} \quad (9)$$

$$m_i = \bigwedge_{k,l \in mC_i \wedge k \neq l} motion_{kl} \quad (10)$$

$$s_i = \bigwedge_{k \in mrC_i} scfree_k \quad (11)$$

$$scfree_i = r_i \wedge m_i \wedge s_i \quad (12)$$

If convex hulls are used of discretely sampled swept joint volumes, there will be an underestimation error from the true swept volume. However, an upper bound on this hull underestimation error can be approximated. Sampling all considered joints by α results in a maximum error $e_\alpha = r \cdot \left(1 - \cos\left(\frac{\alpha}{2}\right)\right)$ as illustrated in Figure 7. Thus, two α-sampled convex hulls are collision free if their distance $d > 2 \cdot e_\alpha$.

V. Compact Sphere Construction

In this section, we provide a detailed description of our compact sphere construction algorithm. As described in Section II-B we adopted the condition that every parent sphere must completely contain only the vertices of the underlying link geometry contained by its children, rather than all intermediate sphere volumes. Algorithm 1 shows pseudocode for computing compact sphere radii using methods similar to [26] and [27]. During the Initialization stage, the algorithm computes a compact sphere containing the first two vertices. While executing the loop, the algorithm tests whether a vertex is already contained by the compact sphere or not (line 10). The loop ends if all vertices are contained. If a vertex is not contained within the current sphere, the sphere must be updated (line 12 and 13) to include it.

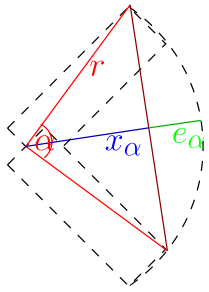


Fig. 7. Estimation of the upper bound error value.

Algorithm 1: CompactSphereConstruction(P)

Data: $P_i = \{V_1, ..., V_{n_i}\}$ convex polygon,
$P = \{P_1, ..., P_N\}$ set of polygons,
$V = \cup_{P_i \in P} P_i$ set of all vertices
Result: compact sphere $S = (c, r)$, $c \in R^3$, $r \in R$
Initialization:
1 $U = \emptyset$
2 choose $v_1, v_2 \in V$
3 $V = V \setminus \{v_1, v_2\}$
4 $U = U \cup \{v_1, v_2\}$
5 $c = \frac{1}{2} \cdot (v_1 + v_2)$
6 $r =$ `EuclDist(`c, v_1`)`
Loop:
7 **while** $V \neq \emptyset$ **do**
8 choose $v \in V$
9 $V = V \setminus \{v\}$
10 **if** `EuclDist(`v, c`)` $> r$ **then**
11 $max = arg\,(\max_{u \in U}(\|v - u\|))$
12 $c =$ `Update(`c, v, max`)`
13 $r =$ `EuclDist(`c, v`)`
end
14 $U = U \cup \{v\}$
end
15 **return** (c, r)

Pseudocode for the update method is shown in Algorithm 2. The purpose of this method is to compute a new location for the compact sphere center. First, we consider the triangle formed by the three points $\vec{c}$, $\vec{v}$ and $\vec{max}$, and compute the projection of the midpoint of the longest edge onto the opposing edge as illustrated in Figure 2(b). This point $\vec{d}$

becomes the new location for the center of the compact sphere.

Algorithm 2: Update(c, v, max)

Data: $c \in R^3$ sphere's old center point,
$v \in R^3$ vertex not contained by the sphere,
$max \in R^3$ *used* vertex with biggest distance to v
Result: $c \in R^3$ sphere's new center point

$offset = \frac{1}{2} \cdot (max + v)$
$temp =$`DotProduct(`$\frac{c-v}{\|c-v\|}, \frac{max-v}{\|max-v\|}$`)`
if $temp \in (0, 1)$ **then**
  $n = \frac{c-v}{\|c-v\|} - temp \cdot \frac{max-v}{\|max-v\|}$
  if $n \cdot (c - offset) < 0$ **then**
    $n = -n$
  end
  $temp =$`Length(`n`)`
  `Normalize(`n`)`
  $temp = \frac{temp}{1-temp^2}$
  $c = offset + \frac{1}{2} \cdot \|max - v\| \cdot temp \cdot n$
else
  $c = \frac{1}{2} \cdot (c + v)$
end
return c

VI. Results

Our algorithms were tested on a wide variety of different models and scenarios. Due to space limitations, only selected results are presented. The kitchen environment (Figure 8) involves more than 95,000 triangles (the humanoid robot ARMAR is modeled with 16,440 triangles).

Query Time*:* Our compact sphere construction method results in smaller overall volumes for the sphere-trees, and hence yields lower overal distance computation times due to the smaller probability of overlapping. Figure 9(a) shows the percentage of additional computation time needed for the greedy method compared to the compact method. For measuring the performance, we used two cubes and computed multiple distance queries across different separating distances. Two different error tolerance (α) values were used: 0% and 30%. The higher error threshhold results in faster distance computation for the compact sphere trees compared to the greedy ones. Thus, as the distance gets smaller, the additional node traversals that must be performed increases, resulting in approximately the same performance for both combination methods.

Distance Range Value Effects*:* Results for self-collision detection with different maximum distance cutoff values for ARMAR are shown in Figure 9(b). Utilizing the maximum value increases the performance of the algorithm not only for values smaller than the real distance, but also for higher values. The same performance improvement can be noticed for collision detection between ARMAR and the kitchen environment obstacles, as the lower cutoff value will quickly prune away distant objects.

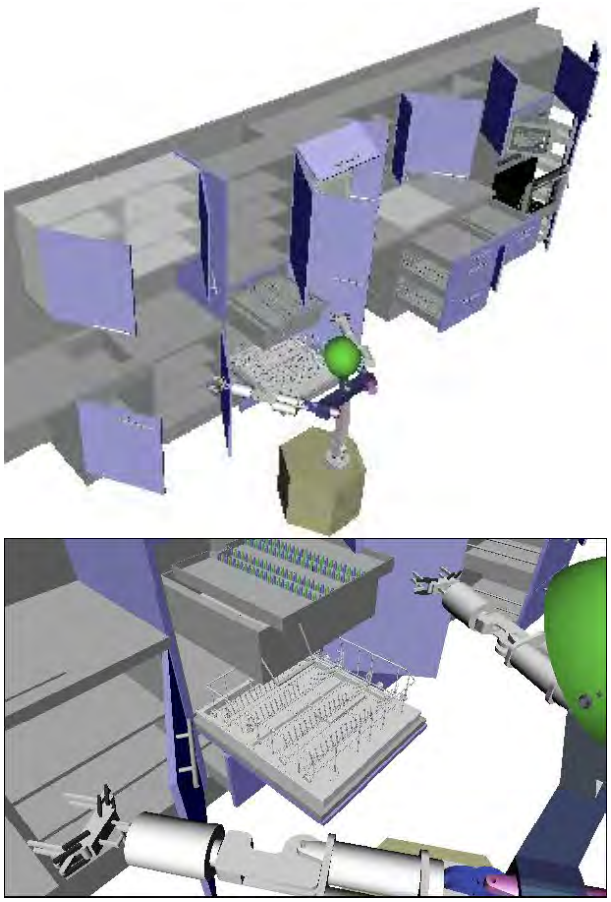


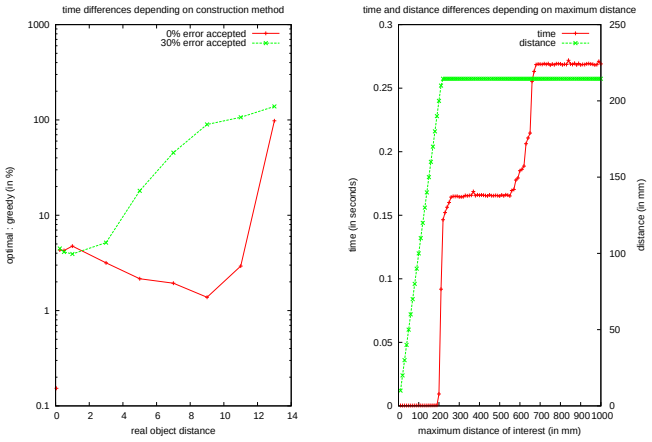
Fig. 8. The kitchen environment with ARMAR.

The effect of changing the minimum distance value is illustrated in Figure 10(a). As soon as the current bound on the distance between ARMAR and the kitchen gets smaller than the minimum safety margin (here: 10mm), our algorithm directly reports a collision situation. Without the minimum distance cutoff, the computation would take much longer, due to the fact that for small inter-object distances, typically many triangle leaf nodes must be evaluated and considered in order to determine the exact true minimum distance.

Error Tolerance*:* Both Figure 10(a) and Figure 10(b) illustrate the effect of error tolerance. Figure 10(a) shows the resulting time and distance for collision detection of ARMAR with the kitchen environment. Figure 10(b) shows the results for self-collision detection of ARMAR. Utilizing a larger error tolerance improves the performance of the collision detection algorithm considerably. Particularly in the case of self-collision detection, a larger error tolerance results in a much faster distance computation.

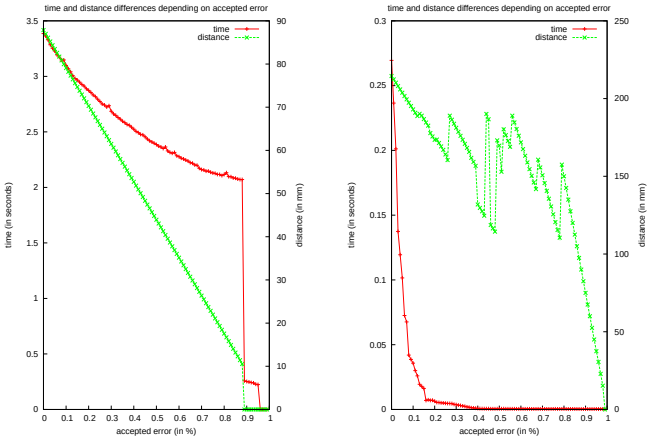
VII. Conclusion

We have presented new techniques for efficient self-collision detection and minimum distance queries for articulated structures of arbitrary 3D meshes. We build compact sphere-tree hierarchies that optimize the locations of the sphere centers and radii to produce bounding volumes that more



(a) (b)

Fig. 9. Elapsed time difference for the two construction methods (left), and the effect of the maximum distance range max_d (right).



(a) (b)

Fig. 10. Performance for various minimum distance range values (left), and for various error tolerance threshholds (right).

tightly approximate the underlying geometry. We improve the distance computation efficiency through the use of a distance range interval, transformation caching, and exploiting our compact sphere tree hierarchies. Finally, we extend and improve the applicability of the algorithm for self-collision detection and obstacle minimum distance computation for articulated structures using range of motion hulls. Exploring additional computational improvements and more detailed analysis form the basis of our future work.

ACKNOWLEDGMENTS

This work has been partly performed in the framework of the interACT exchange program between Carnegie Mellon University and the University of Karlsruhe and the German Humanoid Robotics Program SFB 588 funded by the German Research Foundation (DFG: Deutsche Forschungsgemeinschaft). We thank the anonymous reviewers for their helpful suggestions.

REFERENCES

[1] T. Asfour, K. Berns, and R. Dillmann, "The humanoid robot ARMAR: Design and control." in *The 1st IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS 2000)*, MIT, Boston, USA, 7-8 September, 2000.

[2] G. van den Bergen, *Collision Detection in Interactive 3 D Environments*. Morgan Kaufmann, 2003.

[3] P. Jimenez, F. Thomas, and C. Torras, "3d collision detection : A survey," *Computers and Graphics*, pp. 269 – 285, 2001.

[4] M. Lin and D. Manocha, "Collision and proximity queries," *Handbook of Discrete and Computational Geometry*, vol. 2, 2003.

[5] V. R. de Angulo, J. Cortés, and T. Siméon, "BioCD: An efficient algorithm for self-collision and distance computation between highly articulated molecular models." *Proceedings of Robotics: Science and Systems*, 2005.

[6] O. Brock, "Generation of robot motion: Integrating planning and execution," Ph.D. dissertation, Stanford University, November 1999.

[7] D. Bertram, J. Kuffner, T. Asfour, and R. Dillmann, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'06)*, May 2006.

[8] G. Zachmann, "Minimal hierarchical collision detection," *Virtual Reality Software and Technology*, pp. 121 – 128, 2002.

[9] S. Gottschalk, "Collision Queries using Oriented Bounding Boxes," Ph.D. dissertation, The University of North Carolina, 2000.

[10] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," *International Conference on Computer Graphics and Interactive Techniques*, vol. 23, pp. 171 – 180, 1996.

[11] P. M. Hubbard, "Approximating polyhedra with spheres for time-critical collision detection," *ACM Transactions on Graphics (TOG)*, vol. 15, no. 3, pp. 179 – 210, 1996.

[12] G. Bradshaw, "Bounding Volume Hierarchies for Level-of-Detail Collision Handling," Ph.D. dissertation, PhD thesis, Trinity College Dublin, 2002.

[13] G. Bradshaw and C. O'Sullivan, "Sphere-tree construction using medial-axis approximation," *ACM SIGGRAPH Symposium on Computer Animation SCA*, 2002.

[14] ——, "Adaptive medial-axis approximation for sphere-tree construction," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 1, pp. 1–26, 2004.

[15] ——, "Adaptive medial-axis approximation for sphere-tree construction," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 1, pp. 1 – 26, 2004.

[16] M. J. et al, "Fast and accurate collision detection based on enclosed ellipsoid," *Robotica*, vol. 19, no. 4, pp. 381 – 394, 2001.

[17] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," Department of Computer Science, University of N. Carolina, Chapel Hill, Tech. Rep., 1999.

[18] S. A. Ehmann and M. C. Lin, "Accurate and fast proximity queries between polyhedra using surface decomposition," in *Computer Graphics Forum (Proc. Eurographics 2001)*, 2001.

[19] S. Cameron, "Enhancing GJK: Computing minimum and penetration distances between convex polyhedra," in *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'97)*, Apr. 1997, pp. 3112 – 3117.

[20] B. Mirtich, "VClip: Fast and robust polyhedral collision detection," *ACM Transactions on Graphics*, vol. 17, no. 3, pp. 177 – 208, July 1998.

[21] S. Quinlan, "Efficient distance computation between non-convex objects," *Proc. IEEE Int'l Conf. on Robotics and Automation*, pp. 3324 – 3329, 1994.

[22] ——, "Real-time modification of collision-free paths," Ph.D. dissertation, Stanford University, december 1994.

[23] J. Kuffner, K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue, "Self-collision detection and prevention for humanoid robots." in *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'02)*, May 2002, pp. 2265 – 2270.

[24] P. K. Agarwal, L. Guibas, A. Nguyen, D. Russel, and L. Zhang, "Collision detection for deforming necklaces," *Computational Geometry: Theory and Applications*, pp. 137–163, 2004.

[25] D. Ruspini, K. Kolarov, and O. Khatib, "The haptic display of complex graphical environments," in *Proc. ACM SIGGRAPH*, 1997, pp. 345 – 352.

[26] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," *New Results and New Trends in Computer Science*, pp. 359 – 370, 1991.

[27] K. Fischer and B. Gärtner, "The smallest enclosing ball of balls: Combinatoreal structure and algorithms," *International Journal of Computational Geometry and Applications*, august 2004.