Reaching Volumes Generated by Means of Octal Trees and Cartesian Constraints

I. Rodríguez[a], M. Peinado[a], R. Boulic[b], D. Meziat[a]

[a]Computer Engineering Department, University of Alcalá, Madrid, Spain
[b]Swiss Federal Institute of Technology, Lausanne, Switzerland
inma@aut.uah.es, manupg@aut.uah.es, ronan.boulic@epfl.ch, meziat@aut.uah.es

Abstract

This paper presents a system to analyze the reachability of the human body. The inverse kinematics technique is employed to find which regions of space are reachable using a certain reach strategy. This information is stored in a data structure called Volume Approximation Tree (VATree). This tree has proven itself to be an appropriate data structure for two reasons: it provides an efficient representation of the reachable volumes and it reduces the number of inverse kinematics simulations necessary for its construction. Once the VATrees are constructed for the different reach strategies, that information can be used to determine in real time which strategy is most suitable for a given reach task.

1. Introduction

Human movement and, in particular, human arm motions play an important role in studies of the human body. Reaching is a daily life activity. We frecuently reach a glass of water, a door handle, a book in a shelf, etc. Human reachable spaces generated by computers allow the analysis of the human body and its environment. We can generate and compare reachable spaces in different conditions (sitting, standing …)

Computers give us the flexibility to easily change simulation parameters and analyze the results in virtual worlds before applying them to the real world. In a typical industrial workspace many products are manipulated by the worker's arm. That is the reason why reaching tasks have been widely studied in ergonomic research [6]. For example, cars are designed so that front panel elements are easily accessible by the driver; these elements must be reached in a direct or quasi-direct way.

Medical studies also concentrate their efforts in reaching movements. The reachable volume for a physically impaired person should be different than the volume corresponding to a non-handicapped person.

This paper is organized in six sections. Section one is this introductory section. Section 2 describes the state of the art. Section 3 presents an algorithm for volume approximation. Section 4 describes how this algorithm is utilized to approximate reachable volumes. It also outlines the relation between inverse kinematics and the process of reachable volume generation. Section 5 shows some results and compares different reaching strategies. Section 6 presents conclusions and future work.

2. State of the Art

A previous study of reach analysis has utilized an analytically generated reach volume. However, this analytical approach is limited to articulated structures with a small number of degrees of freedom (2 in this case) [10].

Other studies tried to predict the elbow movements from the shoulder movements in order to understand what should be the final position of the wrist (the wrist itself was given no mobility) while reaching. The goal of these studies was the development of a neuroprosthesis for spinal cord injured which no longer have the control of the elbow junction [5].

Interpolation synthesis can produce motions from a mixture of prerecorded data motions. Wiley and Hahn specifically followed this approach. They solved the problem of obtaining natural reach postures by interpolating motion captured data [14].

A study distinguished three reach areas depending on the distance from the hand to the target [12]. The author defined different strategies for each area taking into account factors like the need to control the center of mass or the need for additional supports.

Another research developed a method for delineating surface patches defining the reach envelop of a kinematics chain in closed form. It was designed to be applied to robotic manipulators [1].

A recent work has proposed a method to predict reach motions based on experimental data. A functional regression analysis is made to model how joint angles change over time [8].

3. Approximating Volumes Using Box Trees

We have designed a data structure that let us approximate a certain volume. It is called Volume Approximation Tree (VATree). It leads to a simple and fast algorithm to test whether a given point lies inside the approximated region. In the next section we present the application of VATree to the representation of human reachable regions.

The proposed data structure is an octal tree (nodes have eight children). Octal trees (usually referred to as *octrees*) are commonly used for representing volumes or surfaces [11].

In a VATree each node stores the parameters of its box (origin, width, height, depth) and its type. There are four types of nodes (see Figure 3):

- *Inner node(I):* The eight vertices of its box are inside the volume that is being approximated. We assume that if the eight vertices can be reached, then the entire box is reachable.
- *Outer node(Out):* Its box is totally outside the volume.
- *Parent node(P):* Some of the vertices of its box lay inside the volume and the rest lay outside. To achieve a tighter approximation of the volume, parent nodes are subdivided yielding a set of eight child nodes or *subnodes*.
- *Final nodes(F):* They are related to parent nodes in the sense that they lay on the surface of the volume as well. However, they are treated differently because their depth in the tree is maximum, and thus they can't be subdivided.

Note that only nodes of parent type have pointers to its eight child nodes. Children are obtained by subdividing their parent's box.

Construction Procedure

The following parameters are necessary for constructing a VATree:

- Dimensions and position of the initial box B_0. This box must be chosen so that it completely contains the volume being approximated.
- Maximum tree depth, D_{max}.
- Minimum tree depth, D_{min}.
- V, the volume to be approximated.

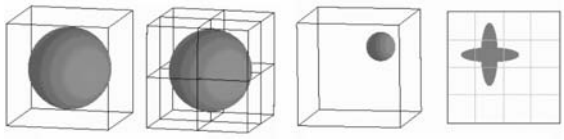
Given a certain node N_i, the following algorithm shows how to construct the branch that hangs from N_i:

1. If the current depth D is less than D_{min}, go to step 7.
2. Compute the position of the eight vertices of N_i's box.
3. Apply the function *isPointInside* to those eight vertices, to find whether each one lies inside or outside V.
4. If every vertex is inside V, label N_i as an INNER node and finish.
5. If every vertex is outside V, label N_i as an OUTER node and finish.
6. It the current depth equals the maximum depth, label N_i as a FINAL node and finish.
7. Label N_i as a PARENT node and allocate memory for its eight child nodes. Repeat from step 1 eight times, taking as root each of those eight child nodes.

Algorithm 1. VATree construction

As many vertices are shared by different subnodes many of the calls to *isPointInside* are redundant. We have provided a mechanism in which, during its subdivision, each parent node calls *isPointInside* for the different vertices of its descendants, storing the result so that this information can be used when building the child nodes. This way, only one call is made for each different vertex.

The success of the algorithm strongly depends on how correctly the initial box B_0 is selected. Extreme situations can arise, such as those shown in Figure 1. The solution to this problem is based on forcing the tree to have a minimum depth D_{min}. During the construction of the tree, all nodes laying at depths greater than D_{min} are unfailingly labeled as parents and subdivided, regardless of their location in relation to the volume.



(a) (b) (c) (d)

Figure 1.

(a) A case in which is needed Dmin=1
(b) A case in which D_{min}=1 is also needed
(c) The box is so big that D_{min}=1 is not enough
(d) A complex volume can make D_{min}=2 insufficient

Several factors must be taken into account when choosing an appropriate value for D_{min}:

- The size ratio of the initial box B_0 and the volume V. In the example of Figure 1.a, one subdivision sufficed because B_0 was chosen with a size similar to V's. In Figure 1.c, however, B_0 is too big when compared to V and thus D_{min} should be greater than 1 for the procedure to work as intended.

- The complexity of the volume V. In Figure 1 d, for instance, the volume requires $D_{min} > 1$ because after the first subdivision all sampling points lay in its exterior.

The construction algorithm works better when V is a simple volume, such as a sphere. When this is not true, it is recommendable to choose a greater D_{min}.

Construction of an Example Tree

An intuitive representation can be achieved if a two-dimensional version of the algorithm is considered. In such a case, each parent node has only four children, and the tree approximates an area instead of a volume.

Note that this simplification implies no loss of generality for symmetric volumes such as a sphere. Figure 2 illustrates the construction of a tree that approximates a circular area, with D_{min}=1 and D_{max}=3.

In Figure 2-a the initial box B_0 can be seen, as well as the circle being approximated. Note that B_0 has been properly selected so that it tightly fits the circle. In the first iteration of the algorithm the root node is labeled as PARENT, despite all of its vertices lying outside the circle. This is correct because the depth D is 0 at this stage, lesser than D_{min}.

Figure 2-b depicts the situation after subdiving the root node. This subdivision yields four subnodes of depth D=1, which are all labeled as *parent* because they have some vertices inside the circle and others outside. It should be remarked that to label these four nodes no calls to *isPointInside* are performed, because all relevant information has been previously collected by their parent.

As seen, the four nodes that resulted after subdividing the root node are themselves *parent* nodes. Thus they must be subdivided, giving the situation shown in Figure 2-c. This figure shows the sixteen nodes of depth D=2. Four of these nodes are classified as inner and won't be further subdivided. The remaining twelve nodes have vertices inside and outside the circle, and thus are labeled as parent nodes and are subdivided.

After performing the second subdivision, the situation is as shown in Figure 2-d. Note that nodes labeled *inner* in the previous step are now absent, since they weren't subdivided. Also note that from all nodes of depth three, 28 are labeled *outer* because they are completely outside the circle. The remaining nodes are labeled *final* since they belong to the frontier of the circle (i.e. some of their vertices lay inside an others lay outside) but can't be subdivided, being their depth equal to D_{max}.

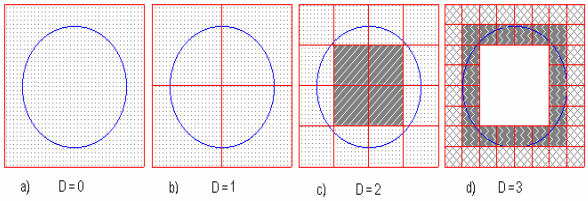


Figure 2. Construction of a 2D version of the tree

The resultant tree is represented in Figure 3. Some branches have been omitted for the sake of clarity. In Figure 3 nodes labeled as *P* are Parent nodes, *Inn* are Inner nodes, *F* are Final nodes and *Out* are outer nodes.

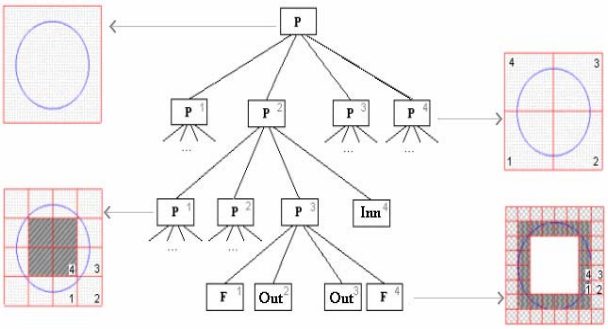


Figure 3. Structure of an example tree

4. Generating a Human Reachable Volume

We aim to know, before the animation takes place, whether a goal is reachable. In other words, to determine in real time if the human model is able to reach a certain point in 3D space. The first solution we attempted was to perform a background IK (Inverse Kinematics) simulation whose outcome was the reachability of the point. However, this approach was soon found unfeasible due to the excessive amount of computation required (IK simulations are inherently costly). We decided to store which regions of 3D space were reachable by the human model. Thus, finding whether a point is reachable is merely a matter of consulting this pre-computed information.

The VATree was tested with a spherical volume using the algorithm described in the previous section. In that case the *IsPointInside* function represented the equation of a sphere. What would be the *IsPointInside* function to approximate the reachable space of a human arm or foot? For humans we ask the inverse kinematics machine if each vertex of the box is reachable after a limited number of iterations. Figure 4 shows the initial box utilized to generate the reachable space of a human arm.

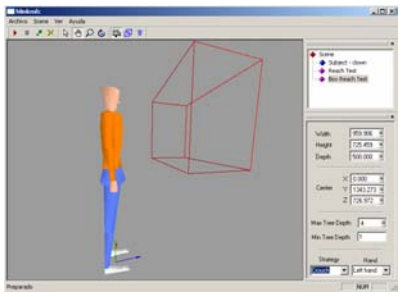


Figure 4. Initial box to calculate reachable space

Inverse Kinematics and VATree Construction

Inverse kinematics is an animation technique that let the user attach Cartesian constraints to some parts of an articulated structure, in our case the human body [2][4]. As our goal is to approximate the volume representing the reachability of a virtual human, the Cartesian constraints are situated in each vertex of the box utilized to construct the reachability tree.

Figure 5 shows the process of construction of the reachability tree as in Algorithm 1 but in a graphical way. The interaction between the tree construction module and the Inverse kinematics module occurs when the algorithm needs to know if the point is or is not reachable; this is shown by red lines in the figure.

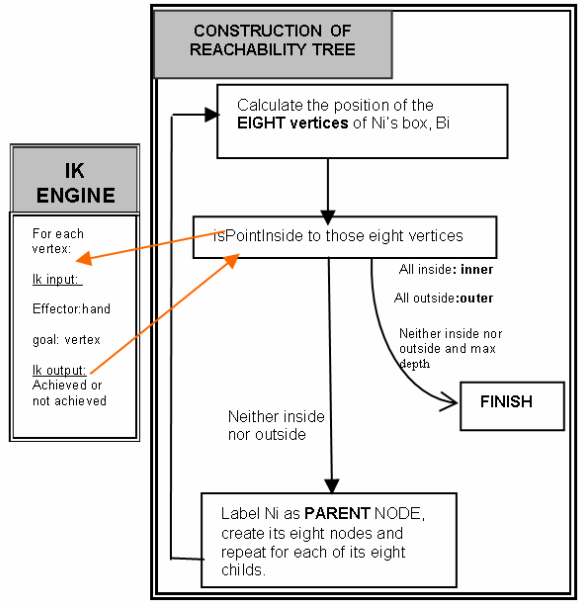


Figure 5. IK & Tree Construction

Depending on the inverse kinematics engine configuration, the tree construction will generate a reachable space with specific characteristics. For example, if the IK engine controls the center of mass this will be reflected in the final reachable space. If several tasks are established to do a reaching (balance, looking at the goal, goal position) tasks configurations with different priorities will generate different reachable spaces.

5. Reaching Strategies

In real life when we have to reach an object we use several strategies. For instance, if the object is close enough we reach it directly. Otherwise, if the object is in a low position we may need to crouch or even to take a step forward. In fact different strategies characterize different type of reaching. Each reaching task is matched with a different strategy depending on where the goal is and where the hand/foot is.

When a person has to reach an object with the hand, depending on the distance between the goal and the hand, the reaching will be direct or it will be necessary to bend the knees (See Figure 6, Figure 7). In the latter case, the goal is not included in the reachable space of the direct one, but inside the reachable space of the crouch strategy. It is worth noting that in the crouch strategy both feet remain on the floor, without heel rising. Another possible strategy would permit a heel-rising movement, thus reaching a wider region.

Our inverse kinematics engine has the possibility of choosing between several strategies if the direct one is not appropriate.

Figure 6. Direct Strategy

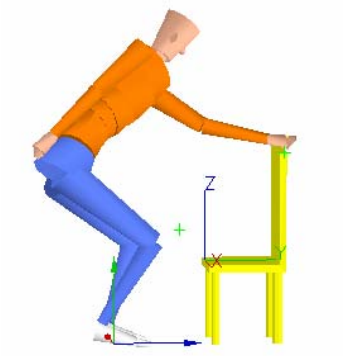


Figure 7. Crouch strategy

We represent reachable spaces by means of the individual boxes whose vertices are inside the reachable volume. In Figure 8 we visualize, using different colors, the reaching volumes corresponding to some of the available strategies. In red, normal reaching with right hand. In green, normal reaching with left hand. And in blue crouch reaching with right hand.

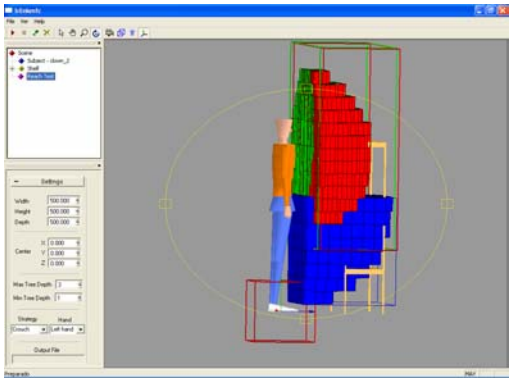


Figure 8. Reachable spaces: direct with left hand(green), direct with right hand(red) and crouch (blue)

Several strategies can share a common intersection space, as shown in Figure 9. If a goal is found to lay inside an intersection region, a high level layer situated on top of the inverse kinematics engine decides which strategy is more suitable. This decision is made based on a priority mechanism.

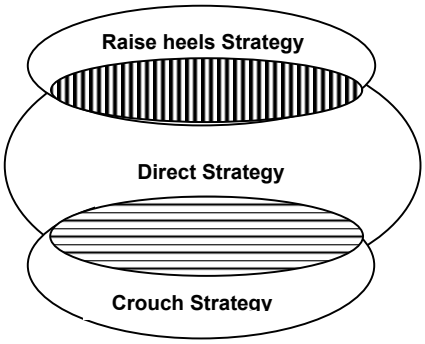


Figure 9. Intersection between strategies

Our VATree structure offers an adequate way to find if a goal is reachable. By maintaining trees for different reaching strategies we can chose the most suitable for a given reach. However, there is an important issue that needs to be addressed: Which posture of the human model (i.e. which combination of joint values) makes the reach possible? Our current approach is to perform an additional inverse kinematics simulation whose output is directly shown to the end user. This approach has the advantage that the resulting posture corresponds to the exact goal, as opposed to other feasible techniques, such as motion interpolation.

Once reachable spaces have been generated, the inverse kinematics engine is used again to simulate a reaching task; it needs a goal in 3D space and an effector and information about the regions that this effector is able to reach.

Comparison of Reachability Trees

Reach space (in red) shown in Figure 10.a was generated by a simulation with a direct strategy (see Figure 6) but using as kinematics chain only the arm of the articulated figure.

Figure 10.b displays the reach space (in green) generated by a simulation with a direct strategy but using the upper body as kinematics chain. Space in (b) is higher than (a) because the clavicle adds extra degrees of freedom. This space is also bigger in front and side views due to the contribution of the spine.

Reach space (in blue) on Figure 10.c is generated by a simulation with a direct strategy using as kinematics chain all the hierarchy and controlling the center of mass. In this case some voxels in the upper and front regions are no reachable anymore, compared to case (b). Also compared to case (b), in case (c) lower positions are reachable due to the contribution of the hip joint that in case (b) was not included.

Figure 11 gathers the three reachable spaces from Figure 10. In this case the blue strategy has been represented by dots instead of boxes and the green one is visualized in wireframe.

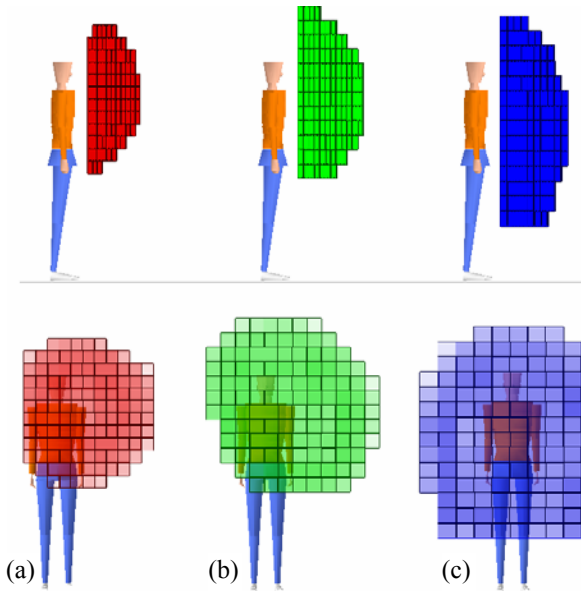


Figure 10. Three reachable spaces:
(a) Direct using only the arm
(b) Direct using the entire hierarchy
(c) Direct using the entire hierarchy and controlling balance

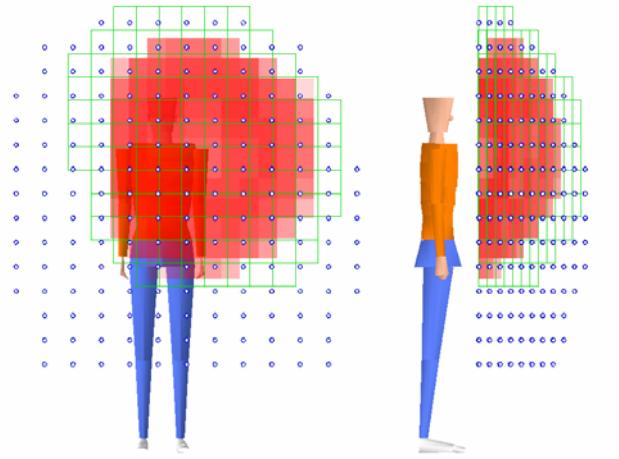


Figure 11. The three reachable spaces together

6. Conclusions and Future work

We have designed a data structure, called VATree, that allows the approximation of volumes and in particular the approximation of human reachable volumes.

Our approach is an important step towards an integrated reach behavior exploiting various strategies associated to their reach volumes. Different reachable spaces for different strategies have been generated.

We have utilized the Inverse Kinematics technique to construct the reachable trees and to create the reaching task.

Besides the normal and crouched strategy, the system can be extended to contemplate other strategies such as *tiptoeing, squat* (buttocks resting on or near the heels), *upper body torsion,* etc..

The high-level library could exploit low level information (i.e: joint level) that would drive the human to do the reaching with another strategy. As fatigue is the exhaustion that limits human activity, we plan to compare reachable space when a person is fatigued [13] with that of a person not fatigued.

We also plan to address the timing of motion in order to get more smooth and realistic reaching animations [3] [7]. For this purpose an interpolation scheme can be useful. This would require storing postural data in each reachable node of the tree. An advantage of this scheme is that it would let us apply available data about the trajectory followed by the hand during reaching motions. More specifically, Faraway has stated that such trajectory is not a straight line [9], contrary to what previous studies had suggested.

7. References

[1] Abdel-Malek, K. and Yeh, H.J., *Analytic Boundary of the Workspace for General Three Degree of Freedom Mechanisms.* International Journal of Robotic Research. Vol. 16, No 2 , 198-213, 1997.

[2] Baerlocher P., Boulic R., Task-priority Formulations for the Kinematic Control of Highly Redundant Articulated Structures, Proc. of IROS'98, Victoria, Canada, Oct. 1998 .

[3] J.E. Bobrow, S. Dubowsky, J.S Gibson. *Time–optimal Control of Robotic Manipulators along Specific Paths*. International Journal of Robotic Research.Vol. 4. No.3, 1985.

[4] R. Boulic, R. Mas, D. Thalmann, *A robust Approach for the Center of Mass Position Control with Inverse Kinetics*. Journal of Computers and Graphics, Vol. 20, No. 5, 693-701, 1996.

[5] L. Cenciotti. Design and Implementation of a System for the Prediction of Upper Arm Articular Synergies Based on Soft-computing Algorithms. PhD, Scuola Superiore Sant'Anna, Pisa, Italy, 2001.

[6] Eui S. Jung, Dohyung Kee and Min K. Chung, *Reach Posture Prediction of Upper Limb for Ergonomic Workspace Evaluation*, Proceedings of the Human Factors Society.-36th Annual .Meeting, 702-706, 1992.

[7] A.H Fagg, L. Zelevinsky, A.G Barto, J.C Houk. *A Pulse-step Model of Control for Arm Reaching Movements*. Proceedings of the 1998 Meeting of the Society for the Neural Control of Movement.

[8] J.J. Faraway, *Modeling Reach Motions Using Functional Regression Analysis*. Digital Human Modeling for Design and Engineering Conference and Exposition, Michigan, June. 2000.

[9] J.J. Faraway, *Modeling Hand Trajectories During Reaching Motion.* TR-383. Department of Statistics. University of Michigan. October, 2001.

[10] Kee, D.H., Shin, Y.T., Kang, D.S., and Chung, E.S. *Reach Volume*. Proceedings of the Fall Conferences of Korean Institute of Industrial Engineers, 232-237, 1996.

[11] D. Libes. *Modeling Dynamic Surfaces with Octrees*, Computers & Graphics Magazine, Vol. 15, No 3, 1991

[12] R. Mas, R. Boulic, D. Thalmann. *Extended Grasping Behavior for Autonomous Human Agents*, First ACM Conference on Autonomous Agents'97, Los Angeles, 1997

[13] I. Rodríguez, R. Boulic., D. Meziat. *A Joint Level Model of Fatigue for the Postural Control of Virtual Humans.* Proceedings of Human and Computer 2002, 220-225, Tokyo, 2002.

[14] D.J. Wiley, J.K. Hahn. *Interpolation Synthesis of Articulated Figure Motion*. IEEE Computer Graphics & Applications, Vol. 17, No 6, Nov./Dec. 1997.