Fast Adaptive Computation of Neighboring Atoms

Stephane Redon

Nano-D, INRIA Grenoble - Rhône-Alpes, France, stephane.redon@inria.fr

ABSTRACT

The main cost of a molecular dynamics or Monte Carlo simulation is the computation of the current potential energy or forces resulting from the interaction of atoms composing the molecular system. When a distance cut-off is used to speed up this computation, a fast method is needed to determine pairs of neighboring atoms.

We have recently introduced an adaptive torsion-angle quasi-statics simulation algorithm, which enables users to finely trade between precision and computational cost, while providing some precision guarantees. In that algorithm, proximity queries are adaptively performed using hierarchies of *oriented* bounding boxes.

In this paper, we show that using *axis-aligned* bounding boxes results in faster proximity queries. We thus introduce a semi-adaptive method to determine pairs of neighboring atoms, where all bounding boxes in the hierarchy are updated, but where interaction lists are adaptively updated during the simulation. The new method allows us to perform proximity queries about two orders of magnitude faster than the previous approach.

Keywords: adaptive, simulation, neighbors, lists

1 INTRODUCTION

As is well known, the main cost of a molecular dynamics or Monte Carlo simulation is the computation of the current potential energy or forces resulting from the interaction of atoms composing the molecular system. Since each atom may interact with each other atom, this computation has a quadratic complexity in the number of atoms, which may be too slow for large systems.

One traditional way to speed up this process is to restrict the computation to pairs of *neighboring* atoms, and accept the resulting error. Precisely, a user-defined *distance cut-off* defines how far two atoms can "see" each other. This reduces the number of interacting atoms, and thus the number of interatomic forces that have to be computed. However, a fast method is needed to determine pairs of neighboring atoms.

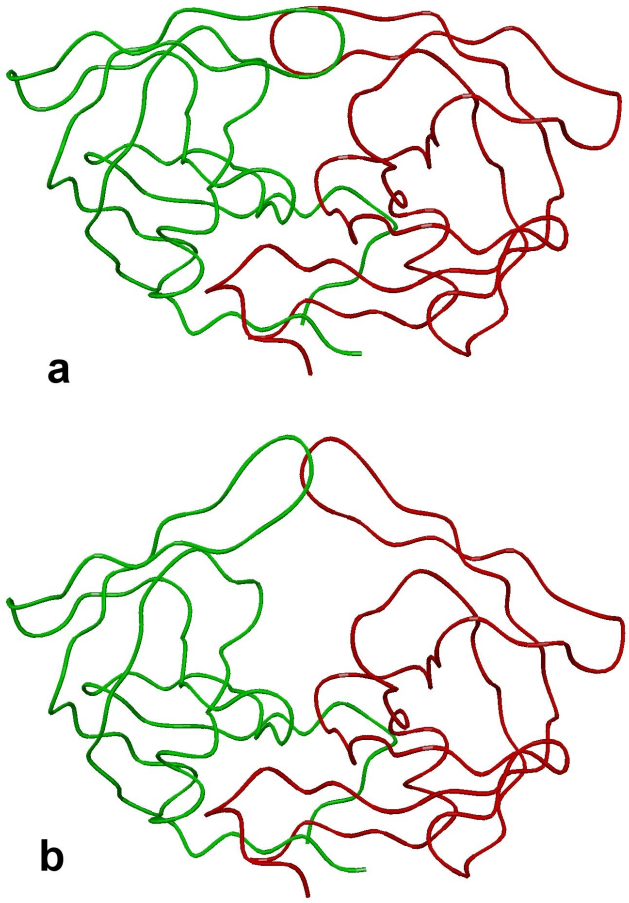

Figure 1: **Creating an open structure for an HIV protease**. In this example, the user opens an HIV protease (**a**, pdb code 2AZ8) with a few mouse clicks (**b**). The model contains two dimers, 1834 atoms, and 816 degrees of freedom.

We have recently introduced an adaptive torsion-angle quasi-statics simulation algorithm [2]. Our adaptive algorithm enables a user to select the *number* of degrees of freedom that should be simulated, or the *precision* at which the computation of the torsion angle accelerations should be performed, and the adaptive algorithm automatically determines the set of most important degrees of freedom under these constraints. The automatic, rigorous trade-off between precision and computational cost allows a user to perform analysis and design of potentially complex molecular systems on low-end computers such as laptops: Figure 1 presents an HIV protease

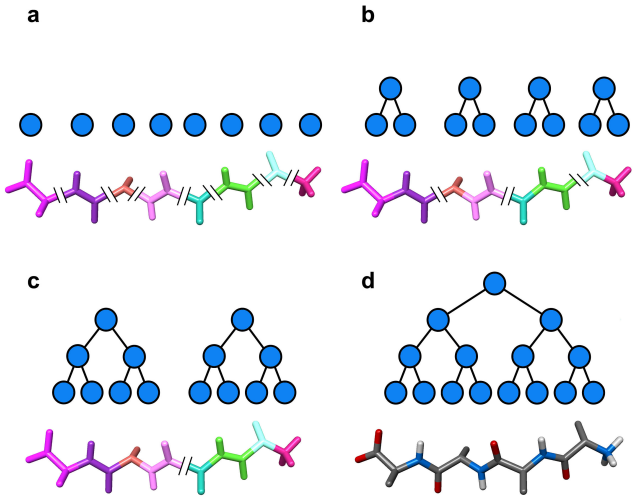

Figure 2: **The assembly tree of a tetra-alanine**. The assembly tree describes a sequence of assembly operations to build the molecular system (see Section 2).

being "opened" with a few mouse clicks by a user using our adaptive tools.

In this paper, we report on how we have managed to speed up the determination of neighboring atoms using an hybrid approach, which combines a linear (non-adaptive) update step of the bounding volumes used to determine neighboring sub-systems, with an adaptive update of the lists of pairs of interacting rigid bodies.

2 ADAPTIVE SIMULATION

Here, we briefly describe our recent work on adaptive molecular simulation for completeness. We refer the reader to Rossi *et al.* [2] for details.

2.1 Adaptive quasi-statics

Our adaptive torsion-angle quasi-statics algorithm is based on a recursive representation of a molecular system: each molecular system is formed by assembling *two* sub-molecular systems, which are in turn each formed of two sub-systems, etc., until we reach single atoms or user-defined groups of atoms that will remain rigid throughout the simulation (the "rigid bodies"). The sequence of assembly operations can thus be described in an *assembly tree*, where leaf nodes represent rigid bodies, internal nodes represent sub-assemblies, and the root node represents the complete molecular system (see Figure 2 for an example). Note that this representation is suitable for both single molecules and groups of molecules.

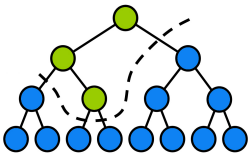

Figure 3: **Active Region.** The active region is the sub-tree which contains the simulated joints.

Each internal node in the assembly tree (including the root node), also represents the relative transformation between the two child molecular systems, so that the goal of the simulation is to compute the accelerations of these relative transformations (in torsion-angle dynamics, the accelerations of the torsion angles).

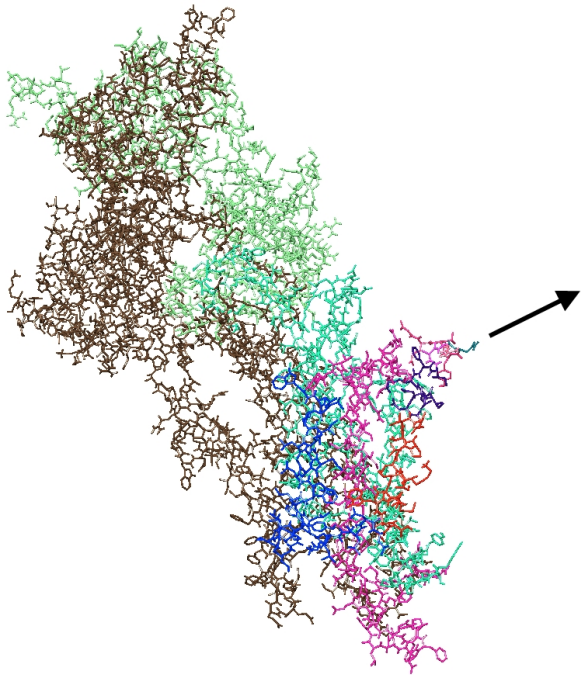

Figure 4: **Adaptive quasi-statics of an ATPase**. The user deforms an ATPase (PDB code 1IW0), while allowing only 10 degrees of freedom out of 4103 present in the system. The adaptive algorithm automatically selects the 10 most relevant degrees of freedom and refines the motion of the ATPase around the point of application of the user force. The resulting rigid bodies are displayed (one color per rigid body).

Our approach to adaptive simulation consists in restricting the computations to a *sub-tree* of the assembly tree. In other words, we simulate a limited subset of degrees of freedom in the molecular system (which form a sub-tree of the assembly tree, see Figure 3). The active degrees of freedom (*i.e.* the most mobile) are determined automatically and rigorously at each time step, which ensures that the user obtains a good approximation of what a complete, non-adaptive simulation would produce, despite potentially limited computational resources. Figure 4 shows an example of an ATPase being interactively deformed by a user allowing 10 degrees of freedom (out of 4103). The adaptive algorithm automatically refines the motion of the ATPase close to the

NSTI-Nanotech 2008, www.nsti.org, ISBN 978-1-4200-8505-1 Vol. 3

point of application of the user force.

2.2 Adaptive proximity queries

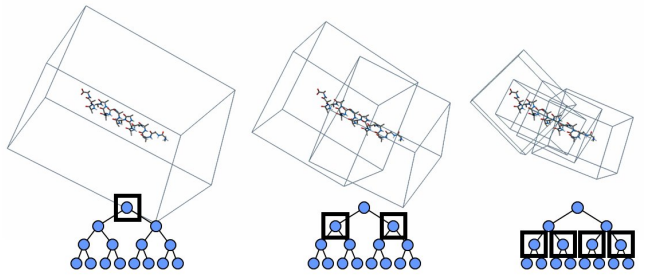

Figure 5: **OBB hierarchy of a poly-alanin**. The figure shows levels 1, 2 and 3 of the OBB hierarchy associated to a poly-alanin.

Freezing and unfreezing relative motions within groups of atoms amounts to cluster the atoms in rigid bodies, whose size depends on the relative amount of local motion. The clusters automatically vary in number and size at each time step, depending on the current configuration of the system, interatomic forces (*e.g.* electrostatic and van der Waals forces), user forces, and user-defined time or precision constraints. One important observation is that all physical quantities which only involve relative positions of atoms in the cluster *do not have to be updated as long as the clusters remain rigid*. This includes the inertia tensor of the cluster, the OBB attached to the cluster, and all interatomic forces which involve atoms in the cluster.

Rossi *et al.* [2] introduces data structures and algorithms which take advantage of this observation to propose a fully adaptive quasi-statics simulation algorithm. In particular, the proximity query algorithm in [2] uses oriented bounding boxes (OBBs) to determine pairs of interacting atoms (or, equivalently, interacting rigid bodies). Specifically, each node of the assembly tree is associated to an OBB which bounds all the atoms in the corresponding sub-assembly (see Figure 5). The OBBs associated to the leaves of the assembly tree are constant, since the leaves correspond to rigid bodies, and they can thus be pre-computed. Each OBB associated to an internal node can be recursively computed from the OBBs of the two children of the node, in constant time, by bounding the two child OBBs. Most importantly, the coordinates of each OBB can be expressed in a reference frame rigidly attached to its corresponding sub-assembly, so that OBBs have to be updated in the active region only. When the atoms move during the simulation, either independently or as part of a rigid group, OBB/OBB overlap tests allow to rapidly detect pairs of interacting atoms, and update *interaction lists* [1], [2]. Interaction lists, stored in each node of the assembly tree, contain the pairs of interactions involving

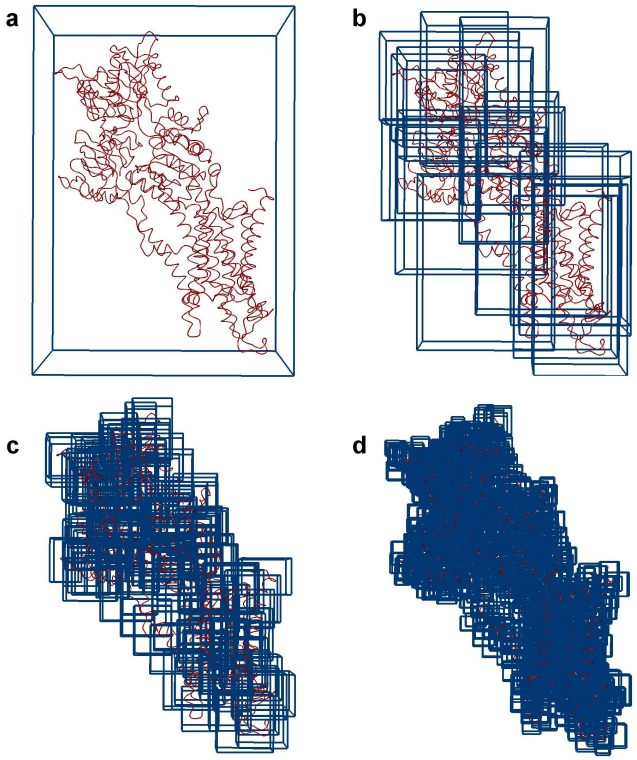

Figure 6: **AABB hierarchy of an ATPase (PDB code 1IW0)**. The figure shows levels 1, 5, 8 and 10 of the AABB hierarchy associated to an ATPase.

one descendent of the left child of the node with one descendent of the right child of the node. An interaction involving two rigid bodies A and B is thus stored in the interaction list of the *deepest common parent* of A and B. Because interaction lists only depend on the relative positions of the atoms in sub-assemblies, they have to be updated in the active region only. With the adaptive OBB hierarchy update, this results in a fully adaptive algorithm to detect pairs of interacting atoms.

3 AXIS-ALIGNED BOXES

In order to make the update of the OBB hierarchy adaptive, each OBB has to be built based on the OBBs of the two child nodes, and not from the set of atoms corresponding to the sub-assembly (else, the cost of the update would be proportional to the number of atoms in that sub-assembly). However, to ensure that each OBB encloses all atoms *without* using the atom positions, the OBB has to *bound* the child OBBs themselves. Unfortunately, this tends to produce OBBs which are larger than necessary. As a result, the recursive proximity query tends to perform too many OBB/OBB overlap tests, which may slow down the determination of interacting atoms significantly. Thus, while the update of the OBB hierarchy is adaptive, the complete algorithm isn't as efficient as it could be.

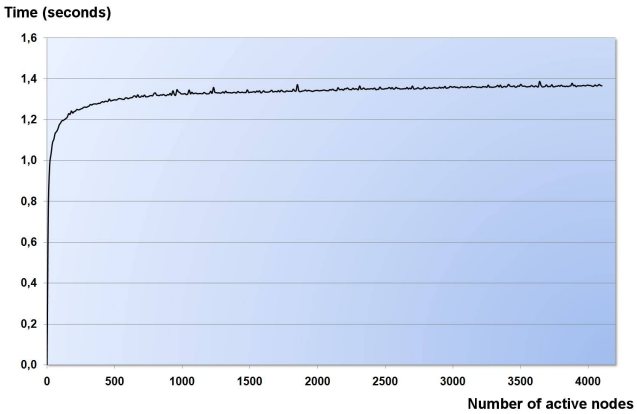

Figure 7: **Cost of proximity queries with OBBs.**

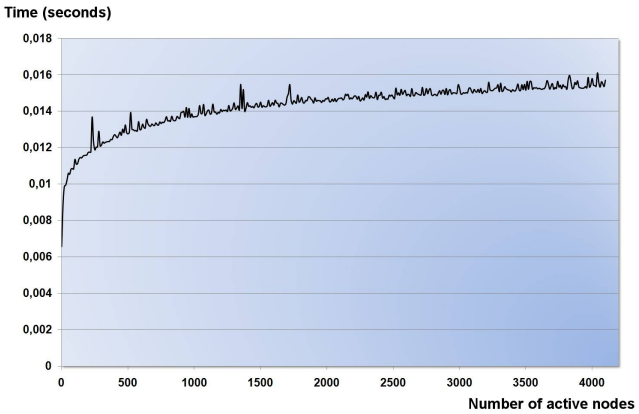

Figure 8: **Cost of proximity queries with AABBs.**

In this work, we thus replace the oriented bounding boxes by axis-aligned bounding boxes (AABBs). As before, one AABB is associated to each node of the assembly tree. However, since by definition all boxes are aligned on the axes of the principal reference frame of the simulation, they all have to be recomputed at each time step. The AABBs of the leaves of the assembly tree (which correspond to the rigid bodies) are computed first, by enclosing all atoms in the rigid body, and enlarging the box by the user-defined distance cutoff. Then, the AABBs of the internal nodes are recursively computed from the bottom up, by bounding the child AABBs. Note that, unlike internal OBBs, internal AABBs are as tight as possible, even though they are directly computed from their child AABBs in constant time. Figure 6 shows levels 1, 5, 8 and 10 of the AABB hierarchy of an ATPase (PDB code 1IW0).

Once all AABBs have been updated, the proximity query algorithm can proceed as before: for each active node, the interaction list of the node is updated by recursively traversing the hierarchy of bounding volumes [2].

4 RESULTS

The method has been implemented in C++ and tested on a 2.8GHz dual core laptop computer with 4GB or RAM (only one core is used, though). The benchmark procedure consists in simulating the motion of an ATPase (PDB code 1IW0, see Figures 4 and 6) and determine the time taken to both (a) update the bounding-volume hierarchy and (b) use the bounding-volume hierarchy to update the interaction lists. The model consists in 4103 internal degrees of freedom (torsion angles) and 9305 atoms. Figure 7 shows the cost of proximity queries depending on the number of active degrees of freedom. As can be seen, this time is zero when no degrees of freedom are allowed (the ATPase behaves as a rigid body), since the method is fully adaptive and nothing has to be updated in this case.

Figure 8 shows that using AABBs results in an about two orders of magnitude speedup over OBBs, despite requiring a complete update of the AABB hierarchy (as can be seen from Figure 8, updating the AABB hierarchy takes about 6.8 milliseconds even when the protease is fully rigid).

Note that in both cases the cost of the proximity queries tends towards its maximum rather quickly when the number of degrees of freedom increases. This is due to the fact that the protein is tightly packed, so that even when few torsion angles are allowed, many interactions may have to be updated. The situation may be quite different when the protein is unfolded (not shown here).

5 CONCLUSION

We have presented a new scheme to compute lists of pairs of neighboring atoms. Our approach uses a hierarchy of axis-aligned bounding boxes — one per assembly node —, which are updated from the bottom up, starting from the atoms. Once the boxes have been updated, we adaptively update the interaction lists for the active assembly nodes only. We have shown that, despite a non-adaptive step required to update all boxes, the overall approach results in faster proximity queries than when using oriented bounding boxes, which were too conservative.

REFERENCES

[1] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: a hierarchical structure for rapid interference detection. *In ACM Transactions on Graphics (SIGGRAPH 1996)*, 1996.

[2] R. Rossi, M. Isorce, S. Morin, J. Flocard, K. Arumugam, S. Crouzy, M. Vivaudou, and S. Redon. Adaptive torsion-angle quasi-statics: a general simulation method with applications to protein structure analysis and design. *Bioinformatics*, 23(13):i408–17, 2007.