Collision Detection for Articulated Deformable Characters

Nadine Abu Rumman [*]
Sapienza University of Rome, Italy

Marco Schaerf [†]
Sapienza University of Rome, Italy

Dominique Bechmann [‡]
University of Strasbourg, France

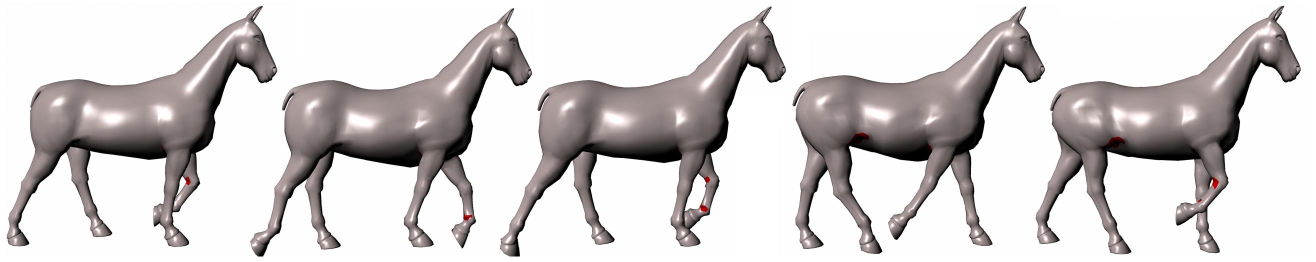

Figure 1: *Real-time self-collision detection for an articulated character deformed by Position Based Skinning: an animated sequence of a walking* HORSE *with a skeleton of 43 bones, 4K vertices and 6K tetrahedrons, all self-collisions are calculated in 2.1 ms per frame, where self-collisions shown in red.*

Abstract

In this paper, we present an efficient method for detecting collisions and self-collisions on articulated models deformed by Position Based Skinning. Position Based Skinning is a real-time skinning method, which produces believable skin deformations, and avoids artifacts such as the well-known "*candy-wrapper*" effect and *joint-bulging*. The proposed method employs spatial hashing with a uniform grid to detect collisions and self collisions. All the mesh primitives are mapped to a hash table, where only primitives mapped to the same hash index indicate a possible collision and need to be tested for intersections. Being based on spatial hashing, our method requires neither expensive set-up nor complex data structures and is hence suitable for articulated characters with deformable soft tissues. We exploit the skeletal nature of the deformation to only update the hash table when required. The resulting algorithm is simple to implement and fast enough for real-time applications. We demonstrate the efficiency of our method on various animation examples. A quantitative experiment is also presented to evaluate our method.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: articulated characters, collision detection, deformable bodies, spatial hashing

Links: DL PDF VIDEO

[*]e-mail:aburumman@dis.uniroma1.it
[†]e-mail:marco.schaerf@uniroma1.it
[‡]bechmann@unistra.fr

1 Introduction

Collision detection is the problem of checking for all possible geometric intersections between models moving in a virtual environment. Solving the collision detection problem has become of major interest in various application areas, ranging from games, animation and virtual reality to surgery simulation and robotics. Moreover, efficient and reliable collision detection is a critical part of almost all graphics applications. A significant amount of research has been done to detect interfering objects moving in space or find exact points of contact. Whereas most of the contributions have been concentrated on collision detection for rigid body simulations, recent approaches started focusing on deformable bodies, i.e. models whose shape changes during the simulation, for example articulated characters, soft tissues, cloth etc. Deformable collision detection is a challenging problem, mainly because of its high computational complexity. Thus, it is difficult to devise an efficient solution to accommodate the rapid interaction speed demanded by users. This is especially true when we are simulating the skin of a virtual character, as self-collisions of the limbs can occur and have to be handled. In this case, collision detection response requires specific information, since it is not sufficient to just detect the collisions. Nevertheless, resolving self-collisions is essential to generate believable skin deformation. Further, by simulating the skin contact in response to collisions, the realism of character animation is highly enhanced. This paper addresses the problem of detecting collisions and self-collisions on articulated deformable characters. In practice, both efficient collision detection and high-quality skinning are needed. Therefore, we focus on collision detection for models deformed by Position Based Skinning (PBS, [Abu Rumman and Fratarcangeli 2015]). PBS is a fast skinning technique that simulates the skin as a soft body, where the deformation approach is based on Position Based Dynamics (PBD, [Müller et al. 2007]). The resulting skinned animations are unaffected by the well-known artifacts of classic interactive skinning techniques, namely the "*candy-wrapper*" effect [1] and *joint-bulging*[2]. Despite offering interesting effects like secondary motions and volume preservation, PBS cannot guarantee the absence of self-intersection deformations.

We propose a fast collision detection method, in which we employ spatial hashing to detect collisions and self-collisions on skeletally

[1]the"*candy-wrapper*" effect is the skin collapsing artifact exhibited by linear blend skinning [Magnenat-Thalmann et al. 1988].

[2]joint-bulging is an unnatural skin bulging effect produced by dual quaternion skinning [Kavan et al. 2007] while bending.

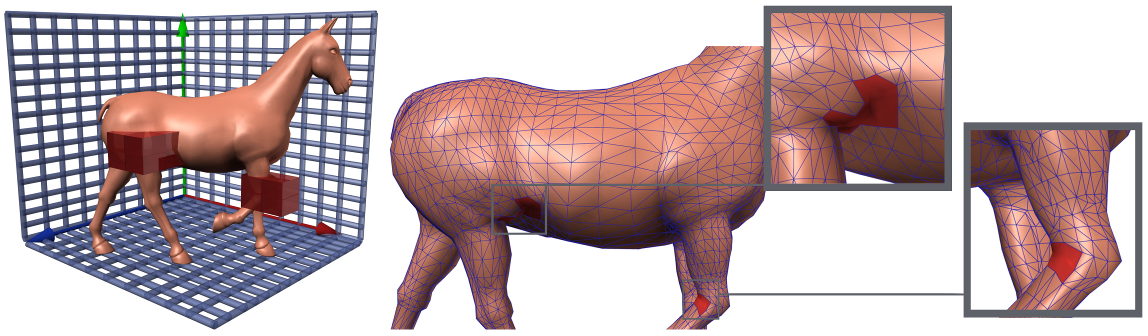
Figure 2: *Self-collision detection. Left. The space is implicitly subdivided into small cells, where red cells contain fully or partially the self-colliding primitives. Middle. The red patches indicate self-collisions, in which intersections are quickly found with an $\mathcal{O}(1)$ cell query. Right. Zoomed view of self-colliding primitives (in red).*

deformable meshes. We use animated skeleton to drive the character's motion, where collisions occur due to the movement. The character's skin is modelled in a two-level representation. First, a fine representation consists of triangles defining the geometry and used for visualization purposes. Secondly, a coarse volumetric representation, which is based on tetrahedral mesh. Tetrahedral meshes are used for collision detection, as well as for calculating deformation. Our method was inspired by the work of [Teschner et al. 2003], but adapted for articulated characters. Unlike highly deformable bodies such as clothes, articulated characters have limited deformation. This deformation is restricted to a function of an underlying skeletal structure, where self-collisions typically occur in very localized regions of meshes. We exploit the skeletal nature of the deformation to obtain real-time self-collision detection for models deformed by Position Based Skinning (Fig. 1). We implicitly subdivide the space into uniform grids of axis-aligned bounding boxes (AABBs), called cells. All mesh primitives (vertices and tetrahedra) are classified with respect to these grids according to their position. Instead of using complex 3D data structures, such as kd-trees, octrees or BSP trees, a hash function is used to project the 3D cells into a finite 1D hash table. Only primitives mapped to the same hash index indicate a possible collision and need to be tested for intersections. The hash index can contain more than one primitive for the same mesh, which allows us to detect self-collisions as well (Fig. 2). Our method focuses on skeletal characters, where we use a lazy procedure that updates the hash table in an on demand way. Therefore, it is not necessary to update the hash table in each time step. Instead, it is only updated when required by collision detection algorithm. This is not only memory efficient, but also leads to a significant improvement in performance. The intersection test is then carried out by computing the barycentric coordinates of a vertex with respect to a penetrated tetrahedron. Thereby, it provides the exact position of a vertex inside a penetrated tetrahedron and this information can be used for collision response.

2 Related Work

Because of its importance, a substantial amount of research in computer animation is related to collision detection. In this section, we mainly focus on those methods, which detect collisions and self-collisions of deformable bodies or models undergoing specific types of deformation, such as skeletal deformation. For a more thorough treatment of the collision detection literature, we refer the reader to the following surveys [Lin and Manocha 2003; Ericson 2004; Teschner et al. 2005].

Bounding Volume Hierarchies (BVHs) have been commonly used to accelerate collision detection algorithms. By using a BVH, the time complexity of a geometric query can be reduced from, i.e. $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$. Many types of bounding volumes (BVs) have been investigated, e.g., spheres [Spillmann et al. 2007], axis-aligned bounding boxes (AABBs) [Bridson et al. 2002], oriented bounding boxes (OBBs) [Gottschalk et al. 1996], discrete oriented polytopes (DOPs) [Klosowski et al. 1998], boxtrees [Zachmann 1995]. Widely used hierarchies are those based on simple BVs such as spheres and AABBs [He et al. 2015]. For a comprehensive discussion on bounding volume hierarchies, we refer the reader to [Zachmann and Langetepe 2002]. In contrast to rigid objects, deformable bodies change their shapes almost at each time step of the simulation. Therefore, their hierarchies must be updated accordingly and the cost of these updates can be high. Traditional approaches require $\mathcal{O}(n)$ operations for each update, where n is the number of object primitives. Several techniques have been proposed to speed up the updates, including refitting algorithms [Larsson and Akenine-Möller 2006; Zachmann and Weller 2006] and dynamic restructuring [Otaduy et al. 2007]. In order to further reduce the computational complexity, on-demand refitting algorithms have been proposed. These algorithms exploit information provided by the deformation model, in which bounding volumes are only recomputed when required by the collision detection algorithm. Such refitting is presented in [James and Pai 2004] for reduce deformation model, in [Larsson and Akenine-Möller 2003] for morphing, in [Kavan and Zara 2005] for linear blend skinning and in [Kavan et al. 2006] for spherical blend skinning. On the other hand, deformable bodies often suffer collisions against themselves and detecting self-collisions is usually a bottleneck for real-time simulations. Self-collision queries with BVHs are executed by starting a recursive query on the root BV against itself. Unfortunately, BVHs lose all their advantages in this case, because tests between triangles adjacent to each other cannot be culled away at high levels in the hierarchy. Thus, detecting the self-collisions is still computationally expensive using BVHs. Moreover, BV approaches typically detect the intersections, but they require an additional processing step to compute the penetration depth for collision response. In order to address these problems, various approaches have been introduced, which are based on surface normals and curvature [Volino and Thalmann 1994; Schvartzman et al. 2009] or based on star-shaped decomposition in [Schvartzman et al. 2010]. An efficient algorithm to handle self-collisions is proposed in [Govindaraju et al. 2005], where they employed both visibility-based culling and chromatic decomposition to radically improve the performance of self-collision detection using BVHs.

Spatial Subdivision Representations are simple and fast approaches to speed up collision and self-collision detection on deformable bodies. These methods have similarities with bounding volume hierarchies, however, the idea here is subdividing the space not the objects. Several spatial subdivision schemes have been proposed for collision detection, such as octrees-like structures [Madera et al. 2006], BSP trees [Luque et al. 2005], kd-trees [Teller and Sequin 1991], uniform grids [Zhang and Yuen 2000] and spatial hashing [Teschner et al. 2003; Alcantara et al. 2009]. An important structure to be highlighted here is spatial hashing, because it has become widely used for collision detection of deformable bodies [Sud et al. 2006; Eitz and Lixu 2007]. Instead of using complex data structure and explicitly performing a spatial subdivision, a hash function is used to map 3D cells into a hash table. In [Maciel et al. 2007] a spherical hash is used to detect collisions on biomechanical models. Despite that this method has shown a fast performance in the main processing stage, there is a drawback: In case the objects deform in a non-radial direction, the preprocessing stage must be repeated and it is slow. Jund et al. [Jund et al. 2009] presented an efficient collision detection method based on spatial subdivision, which supports geometric and topological changes on deformable bodies. As mentioned in the introduction, we decided to employ spatial hashing [Teschner et al. 2003] to detect collisions on models deformed by Position Based Skinning. Despite the simple data structure, the main advantage of using spatial hashing is that it allows to carry out collision and self-collision detection at the same time and in one single pass. Moreover, the collision information provided by the spatial hashing algorithm can directly be used to compute collision response. Our goal is to detect collisions for models deformed by Position Based Skinning in an arbitrary posture. Therefore, we exploit special properties of skeletally animated models to speed up the collision detection.

3 Method Overview

The inputs to our method are a fine surface mesh and an animated skeleton. From the surface mesh, we generate a tetrahedral mesh using [Si 2015], which preserves the original outer surface geometry and consists of tetrahedrons of roughly the same size. Tetrahedral meshes are used for collision detection, as well as for calculating deformation. The method is composed of the following phases: in the *initialization phase*, the vertices of the original skin mesh are mapped to the tetrahedral elements by using barycentric coordinates. Then, at every animation frame, in the *deformation phase*, the skin is deformed with Position Based Skinning (for an exhaustive explanation of the PBS method, please refer to [Rumman and Fratarcangeli 2014; Abu Rumman and Fratarcangeli 2015]). Although PBS is unaffected by the artifacts of classic interactive skinning techniques, it cannot guarantee self-intersection free deformations. To detect the self-intersections, in the *collision detection* phase (Section 4), the algorithm gets the deformed vertices as input and computes the colliding vertices, however, the vertices are moved during the constraints solving step of PBS and may encounter new collisions. Therefore, we also detect the collisions inside of the solver loop. The collision detection algorithm proceeds in two steps: first, we map all mesh primitives (vertices and tetrahedra) into a hash table according to their position. However, the reconstruction of the hash table is done in an on demand way and it depends on the underlying skeleton. Then, only primitives mapped to the same hash index indicate a possible collision and need to be tested for intersections, which is preformed in the second step.

4 Collision Detection

In order to detect collisions and self-collisions within the Position Based skinning method, we employ the *spatial hashing* procedure with temporal marks introduced in [Teschner et al. 2003] (Section 4.1). To speed up the self-collision detection, we exploit the skeletal nature of the deformation to only update the hashing table when required. This on-demand hashing operation is the key part of our fast self-collision detection algorithm (Section 4.2).

4.1 Spatial Hashing

The use of a regular partition is suitable for our system, since all the tetrahedrons of the meshes have about the same size. Therefore, we implicitly subdivide the space $\mathbb{R}^3$ into uniform grids composed of small axis-aligned bounding boxes (AABBs), called cells. Each cell maintains a list of the mesh primitives (vertices and tetrahedrons) that are fully or partially contained in the cell. Rather than using complex 3D data structures, a hash function is used to map these cells to a finite number of hash table entries. The algorithm proceeds in two phases:

Hashing Phase: in this phase, all mesh primitives are classified with respect to cells and mapped into hash table entries in uniformly random fashion. Hence, all vertices are mapped into their cell, and all tetrahedrons are also mapped into the cells touched by their bounding box (Fig. 3). This hashing process is dependent on the following parameters:

Table size: the optimal size is related with the number of primitives in the scene, and must be a high prime number in order to minimize the risk of mapping different positions to the same hash index.

Grid cell width: influences the number of mesh primitives that are mapped to the same hash index. Thus, a reasonable choice is to employ the tetrahedron's average edge length.

Hash function: a function that maps a cell to an arbitrary hash table address. A Simple and fast to execute hash function is preferable for spatial hashing. The following function is used:

$$h = hash(i,j,k) = (i\,u \oplus j\,v \oplus k\,w)\ \mathbf{mod}\ n \qquad (1)$$

where $\oplus$ stands for exclusive-or operation, i, j, k are grid coordinates, u, v and w are high prime numbers and n is the hash table size.

For example, a vertex with position $\mathbf{p} = (x, y, z)$ is mapped into a hash table of size n by computing its table index h as follows: $h = \Big[\Big(\lfloor\frac{x}{d}\rfloor \cdot u\Big) \oplus \Big(\lfloor\frac{y}{d}\rfloor \cdot v\Big) \oplus \Big(\lfloor\frac{z}{d}\rfloor \cdot w\Big)\Big]$ **mod** n, where for u, v, w we use the prime numbers 73856093, 19349663, 83492791, respectively. The value d is the cell size.

Intersection Phase: in a second phase, if a tetrahedron interferes with a cell, all associated vertices of that cell are checked for collision with the tetrahedron. To speed up the intersection test: we first test vertex $\mathbf{v}$ against the bounding box of tetrahedron $\mathbf{t}$. If $\mathbf{v}$ is inside the bounding box of $\mathbf{t}$, then an actual vertex/tetrahedron intersection test has to be performed. If an intersection of a vertex $\mathbf{v}$ with a tetrahedron is detected and $\mathbf{v}$ is part of the same mesh, a self-intersection has been detected, but only if $\mathbf{v}$ is not part of the penetrated tetrahedron itself. The actual intersection test computes barycentric coordinates of a vertex with respect to the tetrahedron in order to detect whether a vertex collides with the tetrahedron or not. This step detects all colliding vertices in the scene and provides the exact position of a vertex inside a penetrated tetrahedron. This information can be employed to handle collisions by adding inequality constraints to the system of constraints within the Position Based Skinning method.

4.2 On-demand Hashing

Construction of the hash structure is performed in the hashing phase. This phase maps all primitives into a hash table and it takes

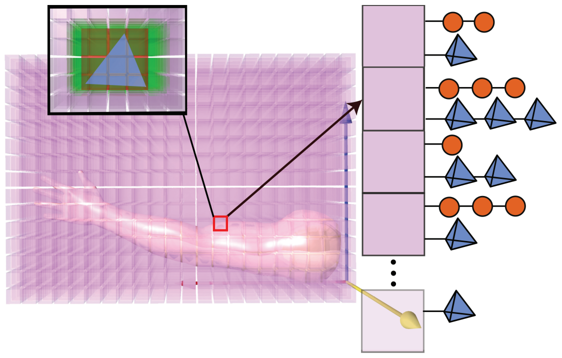

Figure 3: *An example of 3D spatial hashing for an arm . Left. The arm mesh is embedded in a spatial partitioning. The zoomed view shows a tetrahedron (in blue), its bounding box (in red) and all cells affected by the tetrahedron's bounding box (in green). Right. In the hashing phase, all vertices of the arm mesh are mapped into their cell and the hash table indices are computed for all cells covered by the tetrahedron's bounding box. Therefore, in the intersection phase, the tetrahedron is checked for intersection with all vertices found at these hash indices.*

$\mathcal{O}(n)$. While the intersection phase takes $\mathcal{O}(n \cdot p \cdot q)$ where p is the average number of cells intersected by a tetrahedron and q is the average number of vertices per cell. In order to avoid constructing the hash table in each simulation step, which would reduce the efficiency in case of large tables. We use temporal marks or so called *timestamps* to label each cell, where these marks are associated with the moment each cell was last updated. Thus, an intersection with primitives inside a given cell is considered only if it was updated in the current iteration. In other words, inserted vertices are not removed from the table, instead vertices are relocated whenever they move to another hash index. Our goal is to detect self-collisions on models deformed by Position Based Skinning in an arbitrary posture. Self-collision is the most time consuming part of the collision detection. Hence, to speed up the self-collision detection process, we exploit the skeletal nature of the deformation. In particular, we exploit the fact that the only thing that changes the shape of the deformed skin during the animation are the bone rotations $\mathbf{T}_j$ (all other data are constant). It is therefore possible to base the hashing procedure solely on the actual bone rotations. Moreover, most movement modes of skeletal models require rotation of a body part around an axis that passes through the center of a joint, and such movements are called angular movements. The common angular movements involve either an increase or a decrease in the angle between the articulating bones. The principal angular movements are *flexion*, *extension*, *abduction* and *adduction*. Flexion motion refers to a decrease in joint angle between articulating bones, while extension is the motion of increasing the joint angle between articulating bones (Fig. 4). Abduction is movement of the limbs away from the body, and adduction is movement toward the body (Fig. 5). Self-collisions for skeletal meshes occur in very localized regions, which are often found near joints. In addition to that, we observed that self-collisions usually happen during the flexion and adduction movements. Therefore, the main idea behind our on-demand hashing operation is to update the data structure (incrementally reconstruct the hash table using the *timestamps*) only during the flexion and adduction motions. Accordingly, we consider the angles θ_i between articulating bones during the animation, and we check if such angles would allow self-collisions. Since the number of bones is usually orders of magnitude smaller than the number of primitives, we perform a quick test based on the angles between the articulating bones (angular displacement of the bones during the animation). Thus we are able to discard reconstruction the hash table when such angles would not allow self-collisions. This is of course much more efficient than performing the hashing operation each time step. This quick test, in case of flexion motion, relies on the relative angles between the articulating bones. In case of adduction, it depend on the angles between the bone segment and the midline of the character. The midline is estimated based on the bounding box of the body.

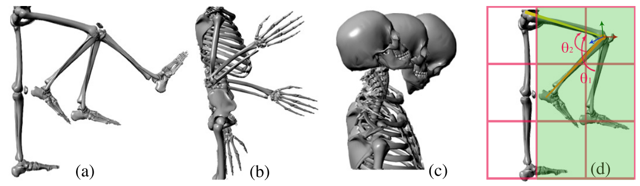

Figure 4: *Examples of flexion and extension motions. Flexion brings two adjoining long bones closer to each other. While extension denotes rotation in the opposite direction of flexion. (a), (b) and (c) Show flexion and extension movements of the knee, elbow and neck joint, (d) shows the angle joint of the knee during flexion, where the angle α_2 is indicating a possible self-collision. The hash table is partially reconstructed by considering only the cells that are affected by the bounding box of the part that indicates collisions (in green).*

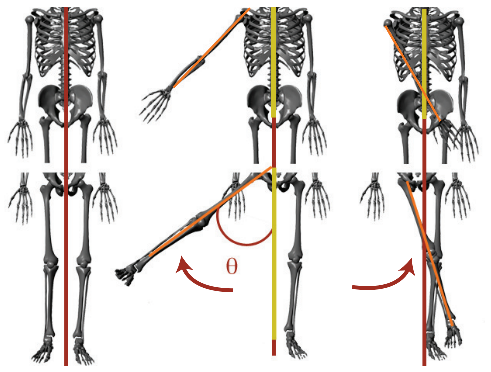

Figure 5: *Examples of abduction and adduction motions. Abduction is the movement of a limb away from the midline. While adduction is the movement toward the midline. We compute the angle between the bone segment and the midline, in order to check whether the angle indicates a possible collision.*

The relative angle is computed automatically as the arc cosine of the dot product of the two vectors: $\theta = arccos\dfrac{a \cdot b}{\|a\|\|b\|}$. If the angle is decreasing (i.e. $\theta_2 < \theta_1$ in Fig. 4 (d)) and less than a prespecified tolerance angle α, then hash table is reconstructed and all primitives in same AABBs are tested for collisions. The tolerance angle α is in the range $\left[\frac{\pi}{3}, \pi\right]$, is defined by the user, and may be dependent on the character. Instead of entirely reconstructing the hash table, we reconstruct the table only in the part where an angle is indicating a possible self-collision. Thus, the hash table is *partially reconstructed* by only considering the primitives, which are in the cells that affected by the bounding box of the potentially colliding part (Fig. 4 (d)).

scene	# vertices	# tetra	# bones	$CT_{skinning}$ [ms]	$CT_{on-demandHashing}$ [ms]	CT_{total} [ms]	# iterations	$fps_{on-demandHashing}$	$CT_{spatialhashing}$	$fps_{spatialhashing}$
HUMAN	9528	4998	25	6.807	3.93	10.737	12	93.136	5.593	80.64
ARM	4211	4400	3	4.112	1.702	5.814	8	171.998	2.53	150.557
ARM + RIGID BODY	4284	4532	3	3.792	2.52	6.312	8	158.429	3.10	145.095
LEG	3990	4339	3	4.108	1.7	5.808	8	172.176	2.47	152.021
HORSE	3833	6126	43	7.69	2.1	9.79	12	102.145	3.35	90.579

Table 1: *Performance comparison of our on-demand collision detection and optimized spatial hashing. #vertices: number of initial vertices in the render mesh, #tetra: number of elements in the tetrahedral mesh, $CT_{skinning}$: avg. skinning computation time and $CT_{on-demandHashing}$: avg. computation time of our collision detection method during 1 sec simulation, where $(CT_{total} = CT_{skinning} + CT_{on-demandHashing})$, $fps_{on-demandHashing}$: avg. frame rate of our method to detect collisions on models deform by position based skinning. $CT_{spatialhashing}$: avg. computation time of optimized spatial hashing, $fps_{spatialhashing}$: avg. frame rate of using optimized spatial hashing to detect collisions on models deform by position based skinning.*

5 Results

All experiments described in this section have been performed on a mass-market laptop equipped with an Intel i5 2.50 GHz processor and 4GB RAM. We implemented our method in C++, and we tested it on a variety of articulated characters. The animations are adapted from ([Abu Rumman and Fratarcangeli 2015]). The grid width that we employed was slightly higher than the tetrahedron's average edge length. Figs. 1, 2 and 6 show the HORSE and HUMAN models animated with a walking cycle. Fig. 7 shows ARM and LEG models while bending, where the number of iteration was 8. All colliding vertices are detected in real-time and the mean computation times are reported in Table 1. Please refer to the enclosed video for the animated results. The last scenario is a posture of a bending ARM and a RIGID BODY (Fig. 8), where our method successfully detected collisions between the ARM model and the RIGID BODY, as well as self-collision on the ARM model. We have evaluated our on-demand collision detection method on a variety of examples. We also compare the performance of our method with the spatial hashing algorithm [Teschner et al. 2003]. Spatial hashing updates the whole hash table after every frame, which can be inefficient. On the other hand, our method uses a lazy procedure that updates the hash table in an on-demand way, which outperforms spatial hashing (see Table 1).

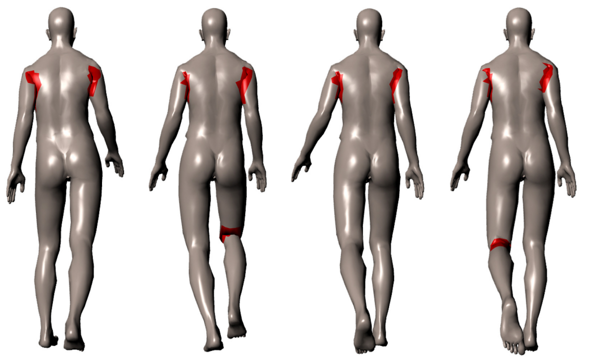

Figure 6: *A back view of a walking HUMAN with a skeleton of 25 bones, 9K vertices and 5K tetrahedrons. All the colliding vertices are computed in 3.93 ms per frame. The red patches indicate self-collisions.*

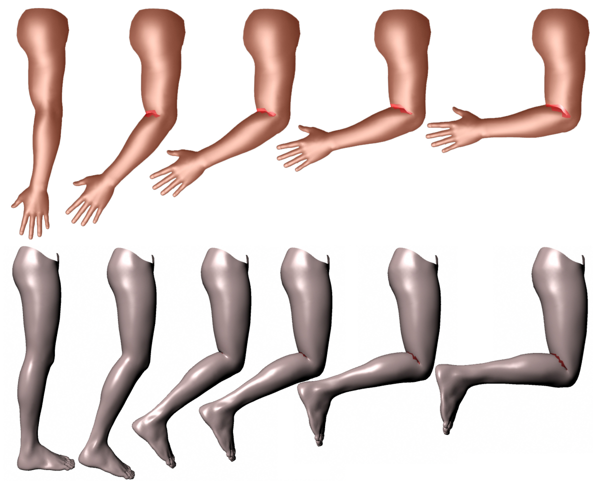

Figure 7: *Top row: an animated sequence of a bending ARM, all self-collisions are calculated in 1.702 ms per frame. Bottom row: an animated sequence of a bending LEG, all colliding vertices are computed in 1.7 ms per frame. (self-collisions shown in red).*

6 Conclusion

We present a simple and fast collision detection method for articulated deformable meshes. During the animation, the skin is deformed with Position Based Skinning, where the deformation model preserves the volume, allows for passive jiggling behavior and avoids the artifacts of classic interactive skinning techniques. Then, the collision detection algorithm gets the deformed vertices as input and finds all colliding vertices. Detecting the collisions is based on spatial hashing [Teschner et al. 2003], and is performed before and inside the constraint enforcement loop of Position Based Skinning. Thus, our method does not miss collision events during the solver loop. Being based on spatial hashing, our method does not rely on any preprocessing and it does not impose requirements on the characteristics of the meshes. We exploit the skeletal nature of the deformation to incrementally reconstruct the hash table only when required using the on-demand hashing operation. This on-demand hashing operation speeds up the full collision detection considerably when compared to the optimized hashing operation. The intersection test provides the exact position of a vertex inside a penetrated tetrahedron and this information can be employed to compute collision response. The Collision response can be done by generating temporary inequality constraints on-the-fly and including them into the system of constraints inside PBS framework, which is ongoing work. To further improve the performance of our method, you will investigate the use of parallel position based dynamics [Fratarcangeli and Pellacini 2015].

References

ABU RUMMAN, N., AND FRATARCANGELI, M. 2015. Position-based skinning for soft articulated characters. *Computer Graphics Forum*, n/a–n/a.

ALCANTARA, D. A., SHARF, A., ABBASINEJAD, F., SENGUPTA, S., MITZENMACHER, M., OWENS, J. D., AND AMENTA, N. 2009. Real-time parallel hashing on the gpu. In *ACM SIGGRAPH Asia 2009 Papers*, ACM, New York, NY, USA, SIGGRAPH Asia '09, 154:1–154:9.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. 21*, 3 (July), 594–603.

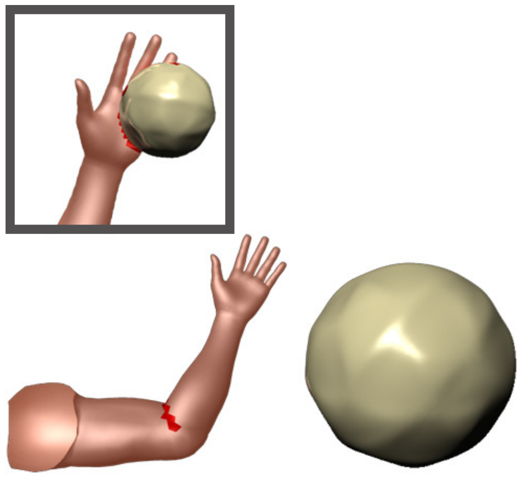

Figure 8: *A bending* ARM *and a* RIGID BODY*, consisting of 4400 tetrahedrons and 132 tetrahedrons, respectively. Both the collisions and the self-collisions of the arm are detected in 2.52 ms.*

EITZ, M., AND LIXU, G. 2007. Hierarchical spatial hashing for real-time collision detection. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2007*, IEEE Computer Society, Washington, DC, USA, SMI '07, 61–70.

ERICSON, C. 2004. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

FRATARCANGELI, M., AND PELLACINI, F. 2015. Scalable partitioning for parallel position based dynamics. *Computer Graphics Forum 34*, 2, 405–413.

GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '96, 171–180.

GOVINDARAJU, N. K., KNOTT, D., JAIN, N., KABUL, I., TAMSTORF, R., GAYLE, R., LIN, M. C., AND MANOCHA, D. 2005. Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. Graph 24*, 991–999.

HE, L., ORTIZ, R., ENQUOBAHRIE, A., AND MANOCHA, D. 2015. Interactive continuous collision detection for topology changing models using dynamic clustering. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games, San Francisco, CA, USA, February 27 - March 01, 2015*, 47–54.

JAMES, D. L., AND PAI, D. K. 2004. Bd-tree: Output-sensitive collision detection for reduced deformable models. In *ACM SIGGRAPH 2004 Papers*, ACM, New York, NY, USA, SIGGRAPH '04, 393–398.

JUND, T., CAZIER, D., AND DUFOURD, J.-F. 2009. Particle-based forecast mechanism for continuous collision detection in deformable environments. In *SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, ACM, New York, NY, USA, 147–158.

KAVAN, L., AND ZARA, J. 2005. Fast collision detection for skeletally deformable models. *Computer Graphics Forum 24*, 3, 363–372.

KAVAN, L., O'SULLIVAN, C., AND ŽÁRA, J. 2006. Efficient collision detection for spherical blend skinning. In *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, ACM, New York, NY, USA, GRAPHITE '06, 147–156.

KAVAN, L., COLLINS, S., ŽÁRA, J., AND O'SULLIVAN, C. 2007. Skinning with dual quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '07, 39–46.

KLOSOWSKI, J. T., HELD, M., MITCHELL, J. S. B., SOWIZRAL, H., AND ZIKAN, K. 1998. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (Jan.), 21–36.

LARSSON, T., AND AKENINE-MÖLLER, T. 2003. Efficient collision detection for models deformed by morphing. *The Visual Computer 19*, 2-3, 164–174.

LARSSON, T., AND AKENINE-MÖLLER, T. 2006. A dynamic bounding volume hierarchy for generalized collision detection. *Comput. Graph. 30*, 3 (June), 450–459.

LIN, M. C., AND MANOCHA, D. 2003. *Collision and Proximity Queries*. CRC Press LLC, Boca Raton, FL, ch. 35.

LUQUE, R. G., COMBA, J. A. L. D., AND FREITAS, C. M. D. S. 2005. Broadphase collision detection using semi-adjusting bsp-trees. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '05, 179–186.

MACIEL, A., BOULIC, R., AND THALMANN, D. 2007. Efficient collision detection within deforming spherical sliding contact. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (May), 518–529.

MADERA, F. A., DAY, A. M., AND LAYCOCK, S. D. 2006. Collision Detection for Deformable Objects using Octrees. In *Theory and Practice of Computer Graphics 2006*, The Eurographics Association, L. M. Lever and M. McDerby, Eds.

MAGNENAT-THALMANN, N., LAPERRIÈRE, R., AND THALMANN, D. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics Interface '88*, Canadian Information Processing Society, Toronto, Ont., Canada, 26–33.

MÜLLER, M., HEIDELBERGER, B., HENNIX, M., AND RATCLIFF, J. 2007. Position based dynamics. *J. Vis. Comun. Image Represent. 18*, 2 (Apr.), 109–118.

OTADUY, M. A., CHASSOT, O., STEINEMANN, D., AND GROSS, M. 2007. Balanced hierarchies for collision detection between fracturing objects. *Proc. of the IEEE Virtual Reality Conference*, 83–90.

RUMMAN, N. A., AND FRATARCANGELI, M. 2014. Position based skinning of skeleton-driven deformable characters. In *Proceedings of the 30th Spring Conference on Computer Graphics*, ACM, New York, NY, USA, SCCG '14, 83–90.

SCHVARTZMAN, S. C., GASCÓN, J., AND OTADUY, M. A. 2009. Bounded normal trees for reduced deformations of triangulated surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '09, 75–82.

SCHVARTZMAN, S. C., PÉREZ, A. G., AND OTADUY, M. A. 2010. Star-contours for efficient hierarchical self-collision detection. *ACM Trans. Graph. 29*, 4 (July), 80:1–80:8.

SI, H. 2015. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw. 41*, 2 (Feb.), 11:1–11:36.

SPILLMANN, J., BECKER, M., AND TESCHNER, M. 2007. Efficient updates of bounding sphere hierarchies for geometrically deformable models. *Journal of Visual Communication and Image Representation 18*, 2, 101–108.

SUD, A., GOVINDARAJU, N., GAYLE, R., KABUL, I., AND MANOCHA, D. 2006. Fast proximity computation among deformable models using discrete voronoi diagrams. *ACM Trans. Graph. (Proc ACM SIGGRAPH 25*, 1144–1153.

TELLER, S. J., AND SEQUIN, C. H. 1991. Visibility preprocessing for interactive walkthroughs. In *IN: COMPUTER GRAPHICS (SIGGRAPH 91 PROCEEDINGS*, 61–69.

TESCHNER, M., HEIDELBERGER, B., MÜLLER, M., POMERANTES, D., AND GROSS, M. H. 2003. Optimized spatial hashing for collision detection of deformable objects. In *Proceedings of the Vision, Modeling, and Visualization Conference 2003 (VMV 2003), München, Germany, November 19-21, 2003*, 47–54.

TESCHNER, M., KIMMERLE, S., HEIDELBERGER, B., ZACHMANN, G., RAGHUPATHI, L., FUHRMANN, A., CANI, M.-P., FAURE, F., MAGNENAT-THALMANN, N., STRASSER, W., AND VOLINO, P. 2005. Collision detection for deformable objects. *Computer Graphics Forum 24*, 1, 61–81.

VOLINO, P., AND THALMANN, N. M. 1994. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum 13*, 3, 155–166.

ZACHMANN, G., AND LANGETEPE, E. 2002. Geometric data structures for computer graphics. In *Proceedings of Eurographics 2002 Tutorials*, The Eurographics Association. Tutorial.

ZACHMANN, G., AND WELLER, R. 2006. Kinetic bounding volume hierarchies for deformable objects. In *ACM International Conference on Virtual Reality Continuum and Its Applications (VRCIA)*.

ZACHMANN, G. 1995. The boxtree: Exact and fast collision detection of arbitrary polyhedra. In *In SIVE Workshop*, 104–112.

ZHANG, D., AND YUEN, M. 2000. Collision detection for clothed human animation. In *Computer Graphics and Applications, 2000. Proceedings. The Eighth Pacific Conference on*, 328–337.