Published in IET Image Processing
Received on 19th October 2009
Revised on 1st February 2010
doi: 10.1049/iet-ipr.2009.0336

Special Issue on Computer Graphics And Multimedia Security

ISSN 1751-9659

Collision handling for free-form deformation embedded surface

S. Jung[1] *M. Hong*[2] *M.-H. Choi*[1]

[1]*Computer Science and Engineering Department, University of Colorado Denver, Denver CO 80212, USA*
[2]*Division of Computer Science and Engineering, Soonchunhyang University, Asan, Korea*
E-mail: mhong@sch.ac.kr

Abstract: Recently, free-form deformation (FFD)-based simulation has received a lot of attention to achieve real-time animation of complex objects, and many researches have improved the accuracy of modelling complex material property. Previously, a free-form deformation axis aligned bounding box (FFD AABB) was proposed to approximate the FFD-embedded surfaces. Using FFD AABB, an efficient update of bounding box is achieved with balancing between accuracy and computational cost. The authors extended the FFD AABB with a more conservative collision handling method between FFD AABBs to deal with the cases that the nodes on one side of FFD AABB are out-of-plane. In addition, the authors adopted a broad-phase culling algorithm to cope efficiently with both self-collision and inter-collision using parallel spatial hashing.

1 Introduction

Free-form deformation (FFD) is a flexible and intuitive modelling tool to manipulate and deform objects for designing and styling of three-dimensional objects. Primarily, FFD has not been physics-based [1–5] but has been combined with physically based simulation to simplify modelling complex objects for physically based simulation [6, 7]. Many exciting animations and simulations [8–11] became viable by combining the FFD control grid (or an FFD grid) for deformation modelling with the embedded surface for rendering.

Unfortunately, no efficient collision handling techniques tailored to the FFD-embedded surface have been proposed to exercise the full potential of the fast FFD-based simulation. For an FFD-based dynamic simulation, collision between FFD-based objects can be performed over the FFD grids (physical model). Since FFD grids do not approximate the embedded surfaces accurately, the FFD-based collision handling produces severe artefacts such as floating and bouncing-off without making a surface-to-surface contact. Alternatively, embedded surfaces can be used for collision handling of the FFD-based objects directly. Collision handling based on the embedded surface can give accurate interactions between FFD-based objects. However, this is impractical for most applications that require an interactive simulation rate, because deformable objects require frequent updates of their position, which negates the purpose of using the FFD altogether.

This paper proposes a new efficient and comprehensive collision handling technique for FFD-based objects with a plausible accuracy. Our approach is based on an axis-aligned bounding box updated by FFD (FFD AABB) along with the embedded surface at each time step. An FFD AABB accurately approximates the embedded surface for FFD-based objects, and the computational cost of maintaining FFD AABB is very low. The cost for updating the FFD AABBs of FFD-based objects is negligible, because the linear FFD requires only several multiplications and additions. In addition, since our method makes use of a spatial subdivision method that requires no tree structure and no additional update operations, it alleviates the computational cost and memory requirement. Collisions are corrected using a hybrid method combining an impulse-based method with a geometry-based method.

Our method for collision detection consists of two phases: the broad phase and the narrow phase. In the broad phase, two possible FFD AABB pairs are found using a fast spatial hashing approach. In the narrow phase, two FFD AABBs are checked to determine the lower level collision situation. During the collision resolution process, collisions between FFD AABBs are accurately handled by a combination of a geometric method and an impulse-based method. Our examples show the robustness and usefulness of our methods for various applications with large deformations.

2 Related work

Our work is based on FFD-based simulation and collision detection and resolution for deformable objects. An FFD-based simulation has been achieved successfully by mass–spring systems [6], the finite element method (FEM) [9, 10] and shape-matching deformation [11] for skeleton-driven simulations and large deformations. Most approaches used the linear FFD with a regular FFD grid and produced visually pleasing results [12].

The collision detection and resolution for deformable objects has been studied rigorously and several noteworthy

surveys have been proposed. Lin and Gottschalk [13] categorised collision detection methods according to the type of collision surfaces. Jimenez *et al.* [14] provided a comprehensive description according to the type of approach. Teschner *et al.* [15] introduced and analysed three different approaches for deformable objects. Recently, deformation-bounded approaches have been proposed for reduced coordinate deformation [16] and for matrix-based deformation (meshless deformation) [17]. Unlike general bounding volumes such as sphere trees [18], AABBs [19] and *k*-DOPs [20], deformation-bounded approaches can reduce the cost of updating the bounding volumes by applying deformation to the bounding volume directly without any recalculations from the mesh.

However, it is not trivial to directly employ these approaches for FFD-based simulation, because FFD-based deformation cannot be represented by a matrix or in reduced coordinates. In addition, bounding spheres can easily overestimate an FFD-embedded surface when the radius for all the primitives is not updated during the deformation. Previously, we proposed an efficient bounding box [21] updating to balance the computational cost with the accuracy of collision handling without self-collision handling. Our method improved the performance of collision detection using a spatial hashing and the robustness of collision handling with a conservative overlapping test between bounding boxes.

3 FFD-based simulation

In this section, FFD-based simulation is briefly explained. Physical modelling is separated from rendering in FFD-based simulation. The FFD control grid is generated for the geometric model (embedded surface) automatically, and the control nodes in the grid are deformed by physically based simulation techniques such as a mass–spring system, FEM and meshless deformation. The local coordinates (s, t, u) of all nodes among the embedded surface are calculated before applying deformation by (1) (for the case of the regular control grid)

$$s = (x - x_0),\ t = (y - y_0),\ u = (z - z_0) \tag{1}$$

In (1), $C_0 = (x_0,\ y_0,\ z_0)$ is the origin of the FFD local coordinate system and $p = (x,\ y,\ z)$ is the position of the node in a deformable object.

While simulating the FFD grid by a physically based simulation technique in Fig. 1, the deformation of FFD control nodes is applied to the embedded surface by evaluating (2) at the local coordinates (s, t, u). B_j is the linear Bernstein function and C'_j is the jth control node of the cell

$$p' = \sum_{j=0}^{7} B_j(s)B_j(t)B_j(u)C'_j \tag{2}$$

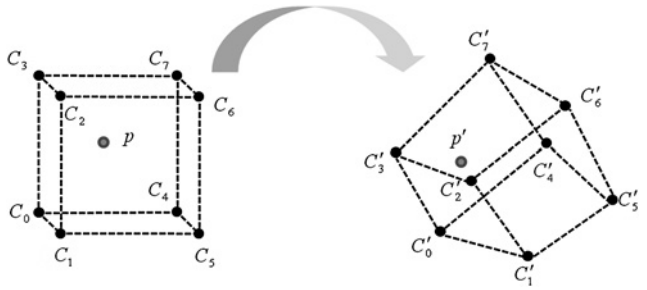

Fig. 1 *FFD in a regular grid cell for a node of the embedded surface: the position of the node p is updated according to FFD and the result position is p′*

4 Building FFD AABB

An AABB consists of six boundary float variables (maximum x, minimum x, maximum y, minimum y, maximum z and minimum z) that are updated at each time step by traversing the primitives in the AABB. In general, the cost of an AABB tree update cost is at least $O(N)$, where N is the number of primitives. Unlike AABB, FFD AABB consists of a set of eight boundary nodes (BNs) that have local coordinates (s, t, u) and global coordinates x, y and z. Fig. 2 illustrates the FFD AABB building process. After the FFD control grid is built for the input mesh, the maximum and minimum for each axis are calculated and the boundary nodes of FFD AABB are located at the eight corners. The local coordinates of the boundary nodes are calculated by (1). In Fig. 2, there are eight BNs of FFD AABB.

The BNs are updated at each time step using the FFD formulation (2) along with the embedded surface nodes. Since the trilinear interpolation is simple and fast, the FFD AABB can be updated quickly and independently from the complexity of the mesh. The FFD AABB contains the embedded surfaces of an FFD cell while the cell undergoes dynamic deformation. Because FFD involves scaling and shearing frequently, bounding spheres do not approximate the FFD-embedded surfaces well, but FFD AABB represents the embedded surfaces efficiently because FFD AABB moves along with embedded meshes. The cost of an FFD AABB update is only evaluating (2) eight times for each FFD cell. Fig. 3 shows a relationship between the FFD grid, FFD AABB and the embedded surfaces. The FFD AABB approximates the embedded surface without overestimation of the surface under large deformation.

5 Collision detection and resolution overview

Our collision detection method consists of two phases. In the first phase, spatial hashing is applied to determine the potential collision pairs (PCPs) of box bounding nodes. In the second phase, the bounding nodes in the same bucket in the hash table undergo collision checking. In general, collision detection is separated from collision response. Collision detection returns the collision result such as a list of the collided primitive pairs (e.g. bounding sphere/bounding sphere). Then the collision response is calculated and applied to the pair to settle the collision situation. In the collision detection process, one of the most time-consuming steps is removing duplicated collision results in order to avoid applying multiple collision corrections.

The key idea of our method is to apply the collision correction and update the boxes of the corrected cells at the collision detection time. Instead of returning a list of PCPs after performing collision detection, the collision response is directly applied and the state of the boxes is updated. Removing redundant pairs from the list of PCPs is expensive when the number of collisions is relatively large. To remove the repetition of collision detection and resolution for the same pair, collision-involved boxes are

doi: 10.1049/iet-ipr.2009.0336

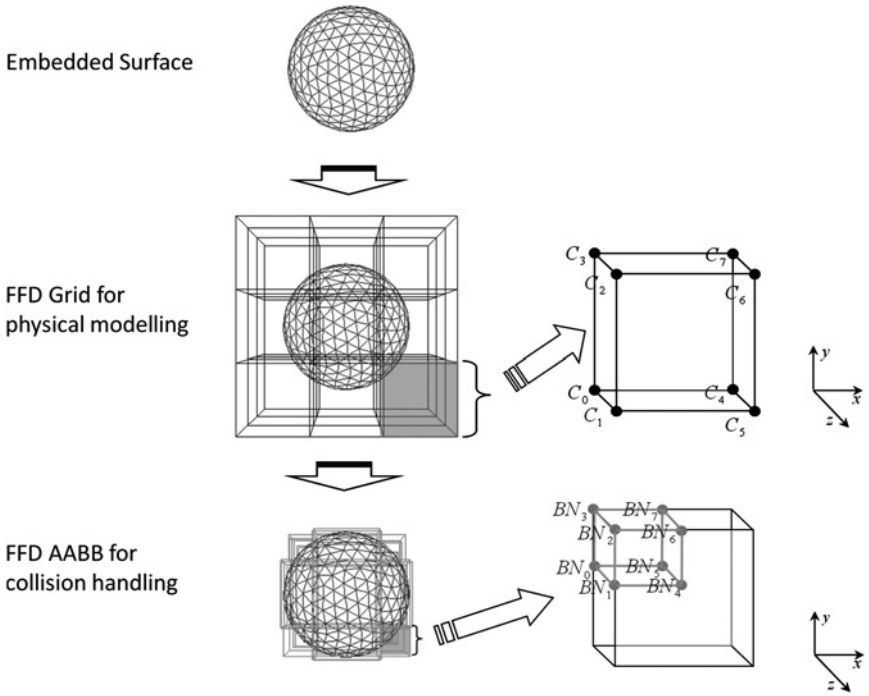

Fig. 2 *FFD AABB building process: after building the FFD grid for the input mesh (embedded surface), the AABB is calculated for each FFD cell, as well as the FFD local coordinates of the boundary nodes*

updated after the collision correction has been applied. This guarantees that the state of the boxes is synchronised throughout the whole collision detection process. The overview of our algorithm is described in Fig. 4.

6 Collision detection

6.1 Spatial hashing of FFD AABB

In the broad phase of collision detection, bounding volume hierarchy or spatial subdivision can be used. However, building and updating the tree of FFD AABB can be costly, since all nodes in the tree must be traversed. Thus we chose spatial subdivision, which is similar to the optimised spatial hashing [22] that requires no modification for self-collision and inter-collision. Unlike traditional spatial subdivision [23], spatial hashing enables unlimited 3D cells to be mapped in a limited data structure efficiently. In our method, the AABBs are used for spatial hashing. After performing voxelisation, the PCPs are determined. For optimal spatial hashing performance, the spatial hashing cell size should be close to the largest length of the edges in optimal hashing. For our case, the FFD cell size is appropriate for the spatial hashing cell size because the FFD cell is the spatial hashing primitive. In our example, we used 4000 for the hash table size because this gave optimal performance. We found spatial hashing time to be constant at 4 ms throughout the alphabet simulation (see the Result section) whereas the number of collisions increases under the same number of objects.

6.2 FFD AABB/node collision detection

During the narrow phase collision detection, the boxes of PCPs are checked, and these boxes undergo various deformations such as shearing and scaling. It becomes non-trivial to determine their collision status. Our method defines the collision regions of the boxes by checking six planes.

If a node is contained within those six planes, the node is inside the current box. The plane equations for the six

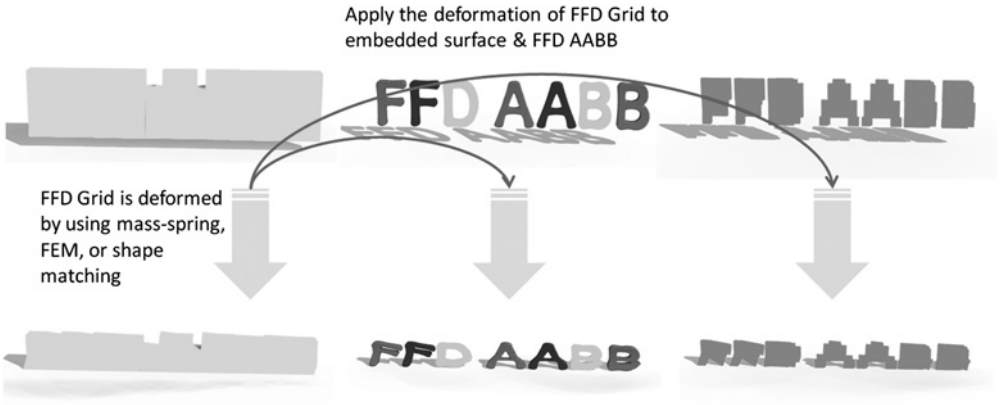

Fig. 3 *FFD AABB update: the deformation of the FFD grid is applied to BNs directly*

```
Algorithm 1 Collision Detection and Resolution
perform spatial hashing
for each bucket B in hash table
  for box b1 in B do
    for box b2 in B and b1≠b2 do
      for node N in b1's boundary nodes (BN) and center of mass do
        if N collided with b2
        then
          apply collision correction
          update b1 and b2
        end if
      end for
      for all  node N in b2's boundary nodes (BN) and center of mass do
        if N collided with b1
        then
          apply collision correction
          update b1 and b2
        end if
      end for
    end if
  end for
 end for
end for
```

Fig. 4 *Collision detection and resolution*

surfaces of any box are calculated as follows: First, the plane normal ($\boldsymbol{N}$) is calculated by the cross product of two vectors between diagonal boundary nodes (3). Fig. 5 illustrates how to calculate the plane equations for the surface consists of BN_2, BN_3, BN_6 and BN_7. The four boundary nodes generate two planes to calculate the plane normal

$$\boldsymbol{N} = (\mathrm{BN}_6 - \mathrm{BN}_3) \times (\mathrm{BN}_7 - \mathrm{BN}_2) \tag{3}$$

The collision region surrounded by the six planes contains all the boundary nodes. However, there are two possible planes for collision testing and one specific plane should be selected. Fig. 6 illustrates the two different cases of the plane set by one specific plane (green planes). Thus, the plane equation is set by testing (4). If (4) is true, the plane equation is set by BN_2 (5), otherwise the plane equation is set by BN_6 (6), where P is the position vector of a node to

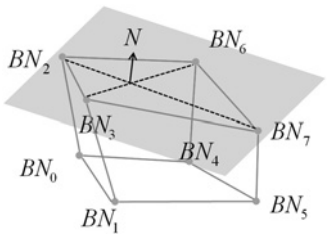

Fig. 5 *Plane for collision check is set to include all involved boundary nodes*

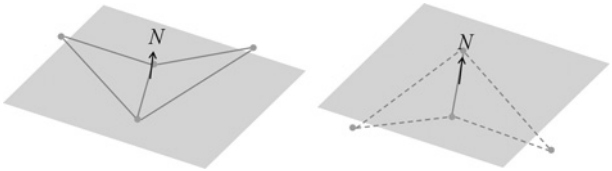

Fig. 6 *Two case of planes where the plane for the four boundary nodes (dots) of a box (FFD AABB) is set by a fixed position (one of two nodes in the middle)*

In the left case, the boundary nodes are above the plane, but all boundary nodes must be below the plane

be collision checked

$$\boldsymbol{N} \cdot (\mathrm{BN}_2 - \mathrm{BN}_6) < 0 \tag{4}$$

$$\boldsymbol{N} \cdot (\boldsymbol{P} - \mathrm{BN}_2) = 0 \tag{5}$$

$$\boldsymbol{N} \cdot (\boldsymbol{P} - \mathrm{BN}_6) = 0 \tag{6}$$

The collision test between a box and a node is performed by inserting the position of the node into the six plane equations. If all values for the plane equation are negative, the node is in the box.

7 Collision resolution

When a collision between a box and a node is found, the collision response is applied to the colliding FFD grids, which have physical property weights and velocities for the FFD AABBs. The collision response can be applied on three levels: force, velocity and position. In order to guarantee a collision-free state, an impulse-based method on velocity and geometric correction are applied to positions.

7.1 Impulse-based response

Let A and B be the FFD simulation grids and assume that a BN of B is inside of A. Firstly the collision direction is calculated by (7)

$$\boldsymbol{R} = r\boldsymbol{N} \tag{7}$$

where $r = \min(d_1, d_2, d_3, d_4, d_5, d_6,)$ and $d_1 \cdots d_6$ are the penetration depths for six surface of the bounding box and $\boldsymbol{N}$ is the normal vector of the plane with the minimum penetration depth

$$\begin{aligned} \boldsymbol{V}'_{i\in A} &= (1-\alpha)(\boldsymbol{V}_{i\in A} \cdot (-\boldsymbol{R}))(-\boldsymbol{R}) \\ &\quad + (1-\beta)(\boldsymbol{V}_{i\in A} - (\boldsymbol{V}_{i\in A} \cdot (-\boldsymbol{R}))(-\boldsymbol{R})) \\ \boldsymbol{V}'_{i\in B} &= (1-\alpha)(\boldsymbol{V}_{i\in B} \cdot \boldsymbol{R})\boldsymbol{R} \\ &\quad + (1-\beta)(\boldsymbol{V}_{i\in B} - (\boldsymbol{V}_{i\in B} \cdot \boldsymbol{R})\boldsymbol{R}) \end{aligned} \tag{8}$$

For collision response, we use an impulse-based collision correction and a geometric correction similar to the methods explained in [24, 25]. The velocity of the cell node i ($\boldsymbol{V}_{i\in A}$, $\boldsymbol{V}_{i\in B}$) of A and B is corrected according to collision response vector ($\boldsymbol{R}$). The response velocity ($\boldsymbol{V}_{i\in A}$, $\boldsymbol{V}_{i\in B}$) is calculated using (7) and applied to the cell nodes when the relative velocity between A and B is negative (getting closer). α is a damping coefficient and β is a friction coefficient to determine the collision characteristics.

doi: 10.1049/iet-ipr.2009.0336

Table 1 Summary of experimental results

Simulation	Number of triangles	Number of FFD grid cells	Collision handling time, ms
teapots	12 640	1204	1.7
alphabets	2 265 744	11 661	279
happy Buddha	293 000	988	0.3
bunnies	347 250	6900	0.5

7.2 Geometric correction for embedded surface

$$\boldsymbol{C}_A = \frac{r}{2}\boldsymbol{R}$$
$$\boldsymbol{C}_B = -\boldsymbol{C}_A \tag{9}$$

Geometric correction ($\boldsymbol{C}_A$ and $\boldsymbol{C}_B$) is half of the minimum penetration depth (8) of six surfaces in the opposite direction because each cell is assumed to have the same mass and stiffness. To correct an embedded surface, the correction is applied to the nodes of the cells (A and B) matching the penetrated box. After applying geometric correction, the box must be updated, and the next collision correction will be calculated correctly.

8 Results

We checked the surface proximity accuracy for the FFD AABB technique as well as the performance of the overall collision detection and resolution through simulations of many different shapes of objects using meshless deformation and a mass–spring system. Table 1 shows the experimental results of our method. The example simulations were run on a PC with a Pentium D 3.0 GHz CPU and 2 GB of RAM.

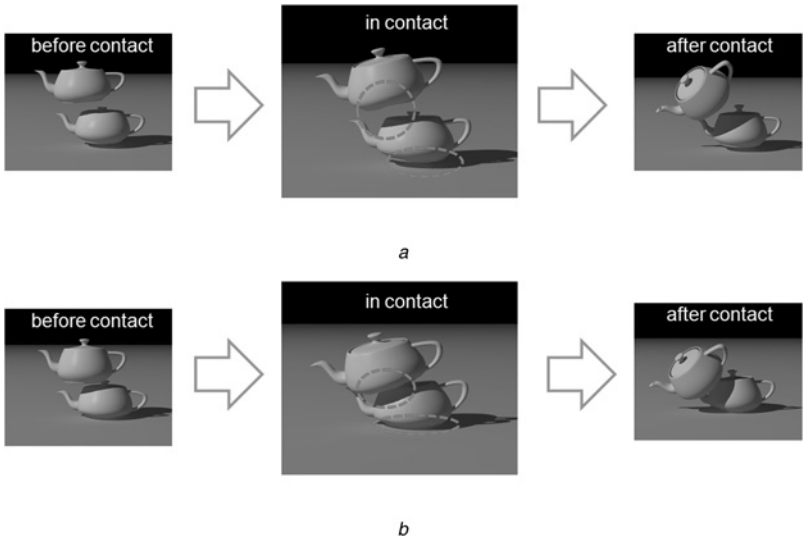

Fig. 7 *Stacking two teapots*

a Snapshots of collision handling with only an FFD grid showing visual artefacts
b Snapshots of collision handling with the proposed FFD AABB showing close proximity
The surface estimation using the FFD AABB is more accurate than the FFD grid estimation. The teapots are staked without any visual artefacts such as floating between two teapots and a gap between floor and the bottom teapot in our method

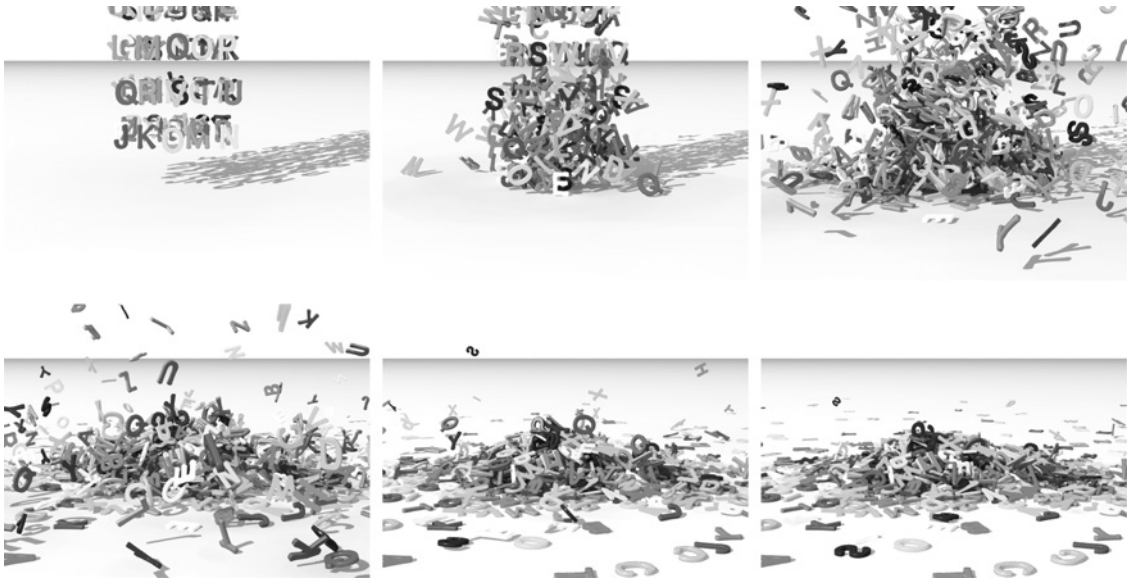

Fig. 8 *Alphabet letters simulation: the snapshots of 1014 alphabet letters simulation at simulation frame number 300 (top left), 810 (top middle), 1680 (top right), 2640 (bottom left), 3640 (bottom middle) and 4800 (bottom right)*

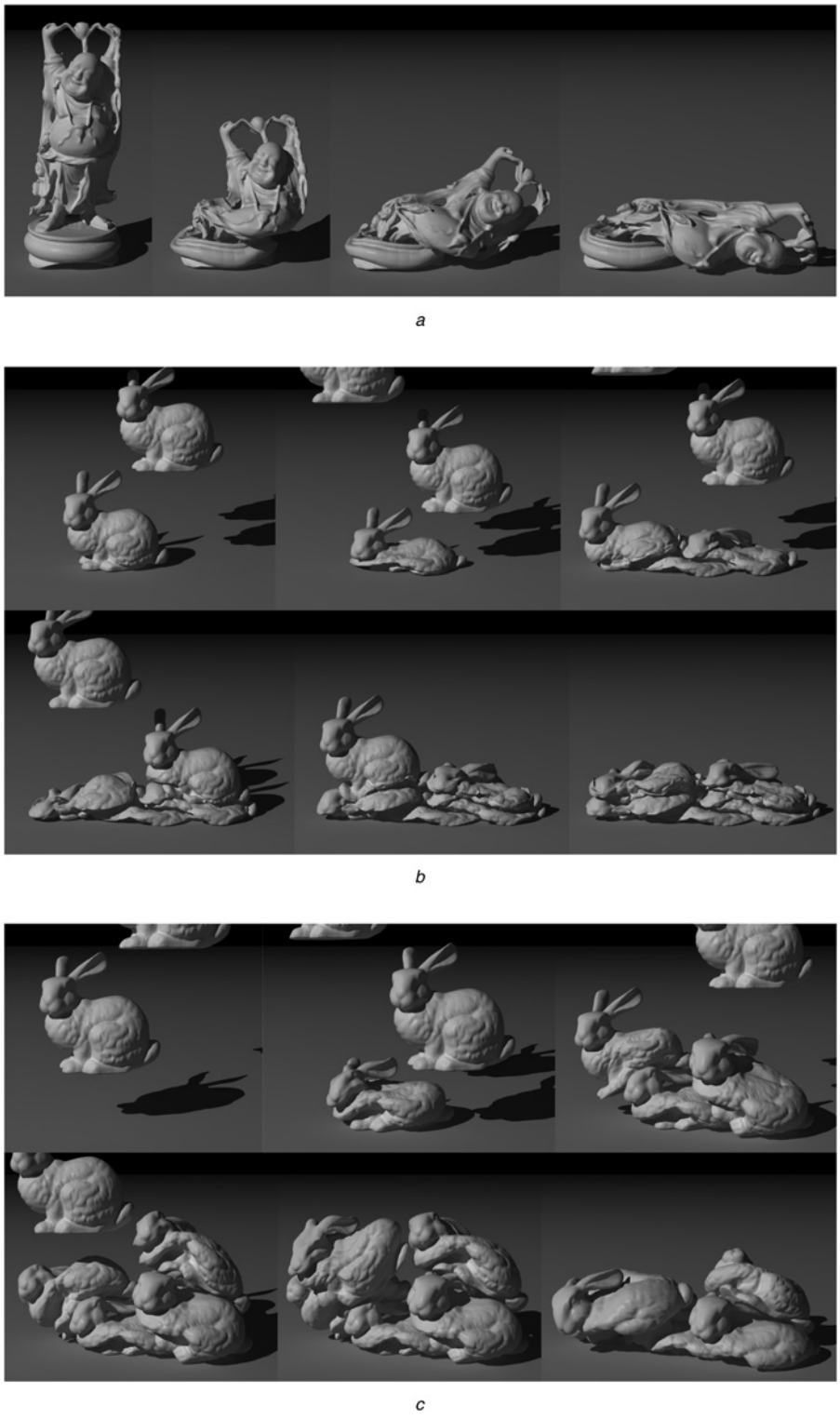

Fig. 9 *Different spring coefficients are used to simulate happy Buddha and five bunny models and the simulations produce large deformations with complex self-collisions*

Soft springs are used for *a* and *b* and hard springs for *c*
a Snapshots to happy Buddha simulation with only self-collisions
b Snapshots of five bunny simulation using soft spring with a complex collision situation including self-collisions and inter-collisions
c Snapshots of five bunny simulation using hard spring with a less complex collision situation than *b*

8.1 Approximation for the embedded surface

To verify whether our method provides close proximity of the embedded surface between complex curved and non-convex surfaces, we simulated collision and contact between teapots. Each teapot in Fig. 7 has 3644 nodes, 6320 triangles and 602 FFD grid cells. The FFD AABBs deform with the FFD grid and contain the embedded surface mesh more tightly than the FFD grid. As shown in Fig. 7, the proximity of FFD grid for the embedded surface is too rough. The result of collision detection by FFD grid shows an apparent large gap between the embedded surface meshes and between the embedded surface mesh and the floor. On the other hand, the proximity between contact surfaces is tightly

doi: 10.1049/iet-ipr.2009.0336

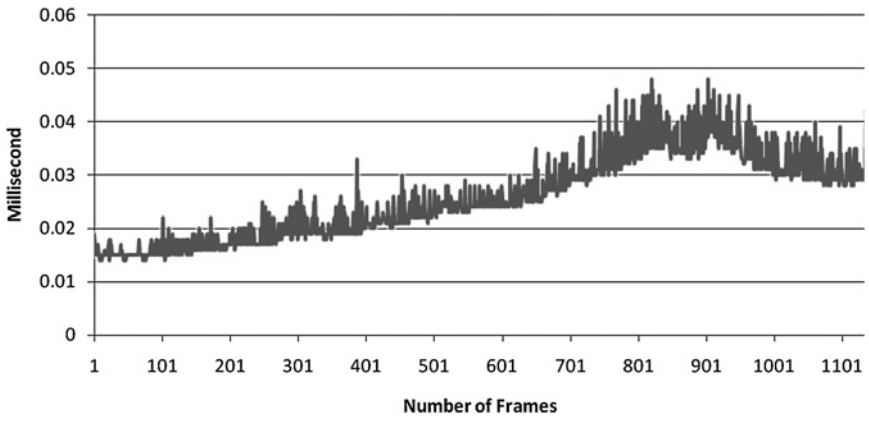

Fig. 10 *Change of collision handling time in millisecond for the happy Buddha simulation*

maintained with our method and does not show any floating artefacts. Both techniques show combined collision detection and response times of 1.7 ms, so there is little or no performance difference. In this example, only one node (centre-of-mass) is added to the FFD AABB because it is sufficient to provide reasonably plausible results. Having more nodes could possibly improve the precise contact at a slight extra computational cost.

8.2 Massive collisions

To verify whether our method can handle massive collisions using spatial hashing, we simulated 1014 alphabet letters. The deforming shapes of alphabets are numerous and the FFD AABB technique efficiently holds the embedded objects under severe deformations and high velocities. Each alphabet model is dropped as a group towards a confined area to create a large number of sustained collisions and contacts. To create various challenging simulation situations, the initial conditions for velocity and orientation or torque for the objects are randomly generated. Fig. 9 shows a series of snapshots of the simulation, time-stamped as it progresses. The number of typical FFD grids for each alphabet model is between 8 and 16 depending on the shape and topology of the model. The number of deformable alphabet objects in Fig. 9 is 1014 and the number of triangles is roughly three millions in the scene. Initially the number of collision is relatively small, but as letters piles up the total number of collision/contact grows accordingly. The processing time per collision/response is kept relatively constant. Our method scales up reasonably as the collision handling instances are increased, and the performance is shown in Fig. 8. The average number of nodes in the embedded surface mesh of alphabets is 1118 ranging from 766 to 1434. The average number of triangles is 2234 ranging from 1528 to 2872. The average collision handling time throughout the whole simulation is 279 ms including the FFD AABB updates, collision detection and resolution.

8.3 Self-collision

To prove the accuracy of our method in handling complex models using conservative approach, several relatively large deformations with self-collisions are simulated.

Happy Buddha (Fig. 9*a*): a Stanford happy Buddha with 293 000 triangles is simulated with 988 FFD grid cells. The model is hollow and has no inner structure to maintain its volumetric shape. The collision handling time is 0.3 ms in average and the detailed computational time is shown in Fig. 10. If all the bounding boxes for 293 000 triangles are updated, the updating process will take more than one second. Self-collision is handled by the same scheme for object/object collisions.

Five bunnies: five bunnies are simulated with different spring coefficients. A Stanford bunny model with 69 450 triangles is modelled by 1380 FFD grid cells and mass–spring underlying structures are used to model a deformable bunny. Five bunnies collide with each other while falling to the floor. Figs. 9*b* and *c* illustrate a close contact between five bunnies under large deformations with self-collisions. The computing time for collision detection and resolution was 0.5 ms per frame for this simulation.

9 Conclusion and analysis

We have extended the concept of FFD AABB to a more robust collision handling for using conservative collision detection and to a more efficient broad-phase culling using spatial hashing. Our technique has few limitations. Our method is well suitable for real-time applications that require simulating complex mesh objects, but it does not guarantee the high accuracy of collision resolution. For accurate collision detection and resolution, FFD AABB can be used as an efficient broad-phase method. Since our spatial hashing method can be benefited by parallel computing using multi-cores or GPUs, we can parallelise our method in the future work.

10 Acknowledgment

The authors would like to acknowledge the Stanford Graphics Laboratory for providing the bunny model and happy Buddha model used in the paper, and the source codes of FastLSM by A.R. Rivers for the 1014 alphabets simulation.

11 References

1 Barr, A.H.: 'Global and local deformations of solid primitives', *SIGGRAPH Comput. Graph.*, 1984, **18**, (3), pp. 21–30
2 Sederberg, T.W., Parry, S.R.: 'Free-form deformation of solid geometric models', *SIGGRAPH Comput. Graph.*, 1986, **20**, (4), pp. 151–160
3 Coquillart, S.: 'Extended free-form deformation: a sculpturing tool for 3d geometric modeling'. SIGGRAPH '90: Proc. 17th Annual Conf. Computer Graphics and Interactive Techniques, 1990, pp. 187–196
4 Coquillart, S., Jancène, P.: 'Animated free-form deformation: an interactive animation technique', *Proc. ACM SIGGRAPH: Comput. Graph.*, July 1991, **25**, (4), pp. 23–26

5 Yoshizawa, S., Belyaev, A.G., Seidel, H.: 'Free-form skeleton-driven mesh deformations'. Proc. Eighth ACM Symp. on Solid Modeling and Applications, SM'03, Seattle, Washington, USA, 16–20 June 2003, pp. 247–253

6 Faloutsos, P., Panne, M., van de, D.: 'Dynamic free-form deformations for animation synthesis', *IEEE Trans. Vis. Comput. Graph.*, 1997, **3**, (3), pp. 201–214

7 Frisch, N., Ertl, T.: 'Deformation of finite element meshes using directly manipulated free-form deformation'. Proc. Seventh ACM Symp. on Solid Modeling and Applications (SMA'02), 2002, pp. 249–256

8 Sela, G., Subag, J., Lindblad, A., Albocher, D., Schein, S., Elber, G.: 'Real-time haptic incision simulation using FEM-based discontinuous free form deformation'. Proc. 2006 ACM Symp. on Solid and Physical Modeling (SPM'06), 2006, pp. 75–84

9 Capell, S., Green, S., Curless, B., Duchamp, T., Popović, Z.: 'A multiresolution framework for dynamic deformations'. Proc. 2002 ACM SIGGRAPH/Eurographics Symp. on Computer Animation (SCA'02), 2002, pp. 41–47

10 Capell, S., Green, S., Curless, B., Duchamp, T., Popović, Z.: 'Interactive skeleton-driven dynamic deformations'. Proc. ACM SIGGRAPH'02, 2002, pp. 586–593

11 Rivers, A.R., James, D.L.: 'FastLSM: fast lattice shape matching for robust real-time deformation'. Proc. ACM SIGGRAPH'07, 2007, p. 82

12 Nesme, M., Kry, P., Jeřábková, L., Faure, F.: 'Preserving topology and elasticity for embedded deformable models'. Proc. ACM SIGGRAPH'09, 2002, pp. 586–593

13 Lin, M., Gottschalk, S.: 'Collision detection between geometric models: a survey'. Proc. IMA Conf. on Mathematics of Surfaces, 1998

14 Jimenez, P., Thomas, F., Torras, C.: '3D collision detection: a survey', *Comput. Graph.*, 2001, **25**, (2), pp. 269–285

15 Teschner, M., Kimmerle, S., Zachmann, G., *et al.*: 'Collision detection for deformable objects', *Comput. Graph. Forum*, 2005, **24**, (1), pp. 61–81

16 James, D.L., Pai, D.K.: 'BD-tree: output-sensitive collision detection for reduced deformable models', *ACM Trans. Graph. (SIGGRAPH'04)*, 2004, **vol. 23**, (no. 3)

17 Spillmann, J., Becker, M., Teschner, M.: 'Efficient updates of bounding sphere hierarchies for geometrically deformable models', *J. Vis. Comun. Image Represent.*, 2007, **18**, (2), pp. 101–108

18 Hubbard, P.M.: 'Collision detection for interactive graphics applications', *IEEE Trans. Vis. Comput. Graphics*, 1995, **1**, (3), pp. 218–230

19 Bergen, G., van den, : 'Efficient collision detection of complex deformable models using AABB trees', *Graphics Tools*, 1997, **2**, (4), pp. 1–13

20 Klosowski, J.T., Held, M., Mitchell, J.S.B., Sowizral, H., Zikan, K.: 'Efficient collision detection using bounding volume hierarchies of k-DOPs', *IEEE Trans. Vis. Comput. Graphics*, 1998, **4**, (1), pp. 21–36

21 Jung, S., Hong, M., Choi, M.: 'Free-form deformation axis aligned bounding box', ISA 2009, pp. 804–813

22 Teschner, M., Heidelberger, B., Mueller, M., Pomeranets, D., Gross, M.: 'Optimized spatial hashing for collision detection of deformable objects'. Proc. Eighth Int. Fall Workshop Vision, Modeling, and Visualization (VMV'03), 2003, pp. 47–54

23 Zhang, D., Yuen, M.M.F.: 'Collision detection for clothed human animation'. Proc. Eighth Pacific Conf. Computer Graphics and Applications (PG'00), 2000, pp. 328–337

24 Mirtich, B., Canny, J.: 'Impulse-based simulation of rigid bodies'. Proc. Symp. on Interactive 3D Graphics (SI3D'95), 1995

25 Bridson, R., Fredkiw, R., Anderson, J.: 'Robust treatment for collisions, contact and friction for cloth animation'. Proc. ACM SIGGRAPH, 2002, pp. 594–603

doi: 10.1049/iet-ipr.2009.0336