A Simple Method for Real-Time Metal Shell Simulation

Zhi Dong, Shiqiu Liu, Yuntao Ou, and Yunxin Zheng

School of Software of Engineering, South China University of Technology
{scut.zhidong,shiqiu1105}@gmail.com

Abstract. Realistic animation and rendering of the metal shell is an important aspect for movies, video games and other simulators. Usually this is handled by FEM method, which requires extremely high computational cost due to solving the stiffness matrix. We now propose a trick that could smartly avoid the large scale computation required by FEM, and could still generate visually convincing images of metal shell. We do so by first partitioning the shell into k rigid sub-shells that evolve independently during impact, before using a simple spring-mass model is applied between each adjacent vertex to retain continuity of the shell. This technique could produce realistically looking deformed rigid shells in real-time and can easily be integrated in any game engine and physics engine.

Keywords: Real-Time Simulation, Game Development, Physics Engine.

1 Introduction

Rendering of the metal shell plays an important role in games. Such as when rendering deformed vehicle shell, deformed bridge, traffic accident or simply metal simulation. The discussion of this phenomenon is mainly in the field of mechanics of materials. Early research focused on metal forming and metal cutting, for example [2] used finite element method (FEM) to simulate metal forming,[4] simulated metal cutting. Later, researches with more practical value started to take place, typically the research of vehicle collision[19].However, all these methods were based on FEM, which makes them impossible to be applied to any interactive system like games, because the required computational expanse of FEM is significantly high. We introduce a simple method to render metal shell, whose computational cost is much less than FEM solution. Our method consists of three steps:

1) Partitioning the shell into a number of sub-shells using K-Means.

2) Treating each sub-shell as a rigid-body and classic rigid-body solver can be applied to solve its movement.

3) Applying an elastic force calculated with spring-mass model to all the adjacent vertices in the shell to maintain continuity.

In the end of this paper we have shown the simulation result and analyzed the performance of our method, to demonstrate that it can produce visually convincing result and can be used in real-time.

2 Related Work

FEM was first initiated in Mechanics of Materials, and it has been applied to computer animation for a long while.

O'Brien et al. [7] and [8] presented a method based on FEM to simulate brittle and ductile fracture in connection with elasto-plastic materials, in which they used tetrahedral meshes combined with linear basis functions and Green's strain tensor. And the resulting equations were solved explicitly and integrated explicitly. The method produced realistic looking results, but it cannot be used for real-time purpose. In order to tackle the non-real-time performance of FEM, the boundary element method (BEM) was designed. BEM is an interesting alternative to FEM because all computations are done on the surface of the elastic body, not in its interior volume. The most common method in BEM is the spring-mass method, which was first used in computer graphics for facial modeling [10] [14] [18]. Not before long, dynamic models were used to simulate skin, fat and muscle[1] [4] [17]. The locomotion of creatures like snakes, worms and fish [5] [13]was also handled with spring-mass model. In these systems, spring rest-lengths vary over time to simulate muscle actuation. In previous researches, [6] mentioned the malleable metal deformation. That paper mainly focuses on fracture simulation; moreover, it proposed a solution to simulate plastic deformation, which had been successfully applied to the simulation of clay. Similar to our research, that solution has real-time performance, but it is not designed to simulate metallic material and its supported number of vertices is fairly small.

Numerous real-time techniques exist for handling deformations and fracture. The most related work to us is [11]. [9] proposed an advancing method for deformation in game environment, but it does not focus on deformation of malleable metal.

Based on the previous works, we have proposed a new method to simulate deformation of metal shell. Our method is not as accurate as FEM, but it is easy to implement and to be integrated in games while can still produce nice images of deformed metal shell.

3 Mesh Segmentation

In order to simulate deformation of metal shell, we first decompose the shell into a number of sub-shells. Here, we choose the method proposed in [12] to handle this task. The reason we chose this method is that it can finely decompose the shell in a way that triangle faces sharing the same plane are considered closer to each other and are more likely to be partitioned into the same sub-shell. Therefore, the simulated shell will be more likely to bend in area where the curvature is high. The methodology is stated clear in [12].Here we briefly summarize the gist below.

Given S, a polyhedral shell with n vertices, the goal is to decompose S into k disjoint sub-shell $S_1 - S_k$whose union gives S.

It assumed that distant faces, both in terms of physical distance and in terms of angular distance, are less likely to be in the same sub-shells than faces which are close together. We therefore define the distance between two faces F_1 and F_2as follows. If F_1 and F_2 are adjacent, then:

$$Distance(F_1, F_2) = (1 - \delta)(1 - \cos^2(\alpha)) + \delta Phys\_Dist(F_1, F_2) \tag{1}$$

The first term of the distance definition represents the angular distance, where α is the dihedral angle between the faces. Note that the expression. $(1-\cos^2(\alpha))$ reaches its maximum at $\pi/2$ and its minimum at 0 (or π). Thus, the faces share the same plane(or close to the same plane) are considered closer to each other and are more likely to belong to the same sub-shells. The second term represents the physical distance. It is the sum of the distances between the centers of mass of the two faces and the midpoint of their common edge. Here we set δ as 0.5.

The main idea of the method is to iteratively improve the decomposition quality by transferring faces from one sub-shell to another. This is very much similar to the concept of the K-means clustering algorithm. The algorithm can be divided into four steps: preprocessing, selecting the initial representatives of the sub-shells, determining the decomposition, and reselecting the representatives.

1. *Preprocessing*: In the preprocessing step the distances between all the adjacent faces are computed. It is evident that the number of sub-shells in the final decomposition should be at least the number of disconnected components.

2. *Electing the initial representatives of the sub-shells*: Each sub-shell is represented by one of its faces. In theory, the first representatives of the sub-shells could be chosen randomly.In practice, however, there are also several reasons for smartly choosing these representatives. First, this method converges to local minima, which makes the initial decomposition essential for the final quality of the decomposition. Second, good initial representatives mean that the algorithm will only need a small number of iteration to converge. The following is how we choose our initial representatives. Initially, one representative is chosen for each disconnected sub-shell. It is the face having the minimal distance between its center of mass and the center of mass of its component. Selecting is completed if the number of required sub-shells is less or equal to the number of representatives . Otherwise, the model should be partitioned further. It calculates, for each representative, the minimum distances to all the faces (e.g., using Dijkstra's algorithm). A new representative is added so as to maximize the average distance to all the existing representatives on the same connected component. New representatives are added one by one, until the required number of representatives (i.e., sub-shells) is reached. In case the user specifies that the system should automatically determine the number of sub-shells, or the value of K if we think of this method as K-Mean, new representatives are added as long as their distance from any existing representative is larger than a predefined distance, which is pre-defined. The number of required sub-shells can affect the fineness and computational cost of simulation. In practice, the number of sub-shells should be between 100-128, which is enough to achieve realistic-looking result with low computational cost.

3. *Determining the decomposition*: For each face, the distances to all the representatives within its connected component are calculated. Each face is

assigned to the sub-shells that have the closest representatives. This procedure produces a decomposition of the given model.

4. *Re-electing the representatives*: The final goal of the algorithm is to minimize the function

$$F = \sum_p \sum_{f \in sub-shell(p)} Dist_{fp} \tag{2}$$

Where $Dist_{fp}$ is the shortest distance from a sub-shells representative p to a face f belonging to the sub-shells which p represents. Therefore, $\sum_p \sum_{f \in sub-shell(p)} Dist_{fp}$ is the sum on the shortest distances of all the faces to their sub-shells' representatives. In order to converge to a solution, new sub-shells representatives are being elected. This is achieved by minimizing the sum of the shortest distances from each representative to the faces which belong to the relevant sub-shells. In other words, for each sub-shells a new representative p_{new} is elected as the face (belonging to the sub-shells) that optimizes the function $\min_p \sum_f Dist_{fp}$.In practice,an alternative is to choose as a new representative the face whose center of mass is closest to the center of mass of the sub-shells, as was done in the initialization step. Obviously, the complexity and performance of the latter one is much better. Moreover, our experiments have shown that the decompositions produced by this technique are usually better. If any sub-shells had its representative changed in Step 4, the algorithm goes back to Step 3. An example is given in the figure below.

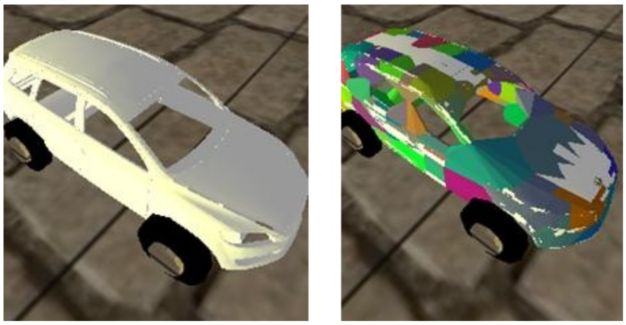
Fig. 1. The left picture is the original mesh of a car, and the right is the mesh model which is segmented with different colors denoting different sub-shells.

Now our shell is partitioned into sub-shells, we can now solve the movement of each sub-shell as a rigid-body.

4 Collision Detection and Response

The first phase of interacting with rigid body is collision detection. Here we use AABB bounding volume tree [15], whose leaf nodes denote sub-shells. When collision detection is performed from the root down to leaves, the GJK algorithm [16] can be applied to solve the narrow phase detection between rigid-bodies and sub-shells.

After collision detection, we get a series of contact points. To handle these contacts and compute impulses, we adopt penalty function which was initially proposed by [3].The most intuitive method to handle collisions is with springs, which means when a collision is detected, a spring is temporarily inserted between the points of closest gap (or deepest interpenetration) of the two colliding objects. The spring law is usually K/d, or some other functional form that goes to infinity as the separation d of the two objects approaches 0 (or the interpenetration depth approaches some small value). K is a spring constant controlling the stiffness of the spring. The spring force is applied equally and in opposite directions to the two colliding objects. The direction of the force is such as to push the two objects apart (or to reduce their depth of interpenetration). Our particular implementation handles variable elasticity by making a distinction between collisions where the objects are approaching each other and collisions where the objects are moving apart from each other. The two spring constants will be related by $K_{recede} = \varepsilon K_{approach}$ where ε is between 0 and 1.

Here we narrow our scope to metal shell. In previous researches, the metal shell's coefficient of restitution is known as a constant (as is often done, especially in games). The relationship between the coefficient of restitution (COR) and ε is written below:

$$COR = \sqrt{\varepsilon} \tag{3}$$

The coefficient of restitution of different materials is listed in the chart below.

Table 1. Coefficient of restitution of different metal materials

Material	COR
Steel	0.2
Aluminum	0.15
Copper	0.1

The spring method is easily understandable and easy to implement. It is one of the most efficient ways to handle collision response in games.

Since we have already partitioned the shell into sub-shells, we now compute the force each sub-shell is suffering using the method described above.

5 Deformation Simulation

In order to achieve realistic looking result, we have analyzed the feature of deformed metal shell. There are primarily two features. The first one is that the dent direction when hit by an impulse is the same with the impulse direction. The second one is that creases are likely to appear in area close to the contact point. One of the most important defects of FEM is that it requires calculating the inverse of high order matrix when updating the stiffness matrix, which makes it unpractical for real-time usage. Our approach of doing this is to treat each sub-shell, which we get from step one, as a rigid-body. And since it's cheap to solve the rigid-body movement, the method could be applied to real-time simulation. Moreover, since each sub-shell evolve independently during impact, adjacent sub-shells are possible to suffer from different movements (translation and rotation), which naturally gives us illusion of crease. Therefore this method can effectively approximate second feature of deformed metal shell.

Each sub-shell has its own angular displacement $\mathbf{d_a}$which is the rotation caused by collision, and linear displacement $\mathbf{d_l}$representing translation(Shown in Fig. 2). The displacement of a sub-shell could be calculated as

$$\mathbf{d_l} = \frac{\sum \mathbf{f}}{S} \tag{4}$$

$$\mathbf{d_a} = \mathbf{K_{xyz}} \frac{\sum \mathbf{r} \times \mathbf{p}}{S} \tag{5}$$

In the formulations above $\mathbf{K_{xyz}}$is a 3x3 matrix similar to inertia matrix, representing the stiffness distribution in x, y, z direction. S denotes the stiffness of the material. The symbol r represents the vector difference between contact point and mass center of sub-shells.

Here, we treat the stiffness as proportional to the mass of vertices. In accordance with linear translation, stiffness distribution $\mathbf{K_{xyz}}$ is equal to the inverse of inertia tensor of each sub-shell. It uses a simple approximation that the higher mass yields higher stiffness. Technical correctness could be more easily

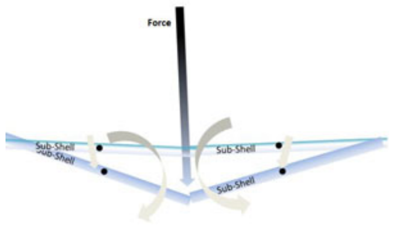

Fig. 2. The transparent sub-shells have not been displaced. The grey arrows denote the translation and rotation of sub-shells.

evaluated by comparing the proposed method directly to real picture. From the Fig. 3 we can tell the visual effect of our simulation is convincing already, but two major artifacts still remain. The first is that creases are way too steep, almost like fractures (evident in the front window in Fig. 3.), while in the real world the crease is much smoother. And secondly, the deformation is a local effect, which means distant areas from the colliding point do not suffer from any deformation. Those artifacts jeopardize the realism of the result of our method. The reason for those artifacts is we neglected the interaction between sub-shells, treating them as rigid-bodies, in previous steps. In order to eliminate this artifact, we are introducing the spring-mass model in the next section.

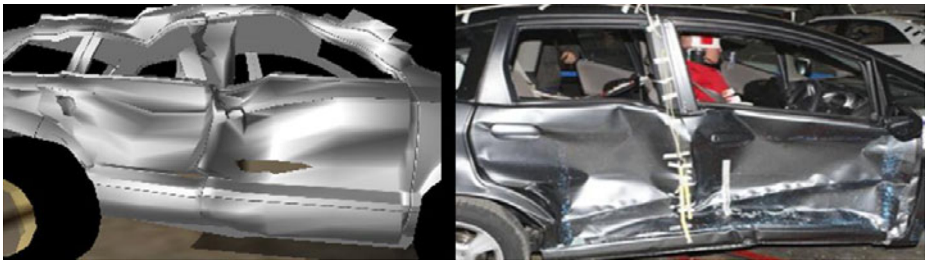

Fig. 3. Left: metal deformed without applying the spring-mass model. Right: Car collision in real world.

6 Globalize the Deformation with Spring-Mass Model

During the previous step, all the deformations are local effect, which do not take the interaction among sub-shells into account. We take into account all vertices on the metal shell using spring-mass model. Please note that this model is different from what we have proposed in Section 4.

We connect each pair of the adjacent vertices in the mesh with a spring. Springs are commonly modeled as being linearly elastic; the force acting on mass i, generated by a spring connecting i and j together is

$$\mathbf{f}_i=\mathbf{k}_s(|\mathbf{x}_{ij}|-\mathbf{l}_{ij})\frac{\mathbf{x}_{ij}}{|\mathbf{x}_{ij}|} \tag{6}$$

Where $\mathbf{x}_{ij}$is the vector difference between the two masses' position of two adjacent vertices. $\mathbf{k}_s$is the spring's stiffness and $\mathbf{l}_{ij}$is its rest length.$\mathbf{l}_{ij}$will be changed in the process of deformation, usually is set as the distance between vertex i and vertex j in the previous frame. Therefore when there is no collision, the distance between all vertices remain still, and the spring-mass model does not have any effect on the simulation result.

When applying spring-mass model to simulating metal shell deformation, some modifications need to be done to prevent deformation being too severe. To all the metal material there is a deformation threshold, higher than which fracture will occur. Since our method does not simulate fracture, we add a constraint to keep the stretching between any two vertices under that threshold. The movement of $\mathbf{x}_i$in t+1 time step can be expressed by the following formulations when huge deformation occurs.

$$\mathbf{x}_i^{t+1}=\mathbf{x}_i^t+\lambda(\mathbf{x}_i^t-\mathbf{x}_j^t) \tag{7}$$

Where $\mathbf{x}_i$is linked with $\mathbf{x}_j$by one spring, and λ is the threshold representing the range of spring stretching. When the stretching length is higher than λ we use Eq. 7 to calculate the displacement. Otherwise, we use Eq. 6 to calculate the displacement.

The figures below demonstrate the difference between simulation with and without the constraint of spring-mass model. From Fig. 4 we can clearly tell

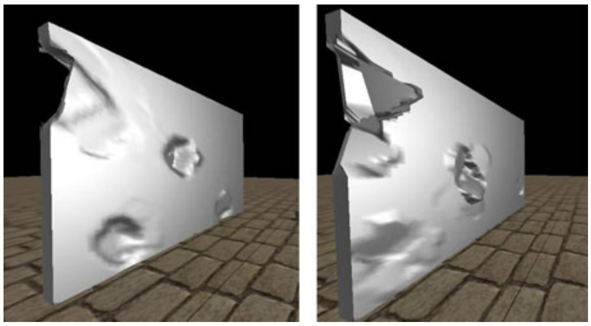

Fig. 4. Left: a metal shell is deformed with spring-mass model applied. Right: deformed without applying the spring-mass model.

that after applying spring-mass model, collision not only yield deformation in local area, but also affect distant area, and the crease is much smoother.

7 Result

In this section we provide two examples to show the result of our method, and to prove it can produce realistic looking images of the metal shell in real-time.

The first example is a car deformed by hitting walls. The rigid body motion is solved by Bullet 2.75.The deformation is not physically accurate, but is already visually convincing enough. This shows that our method could be applied to rendering vehicles in games. The number of vertices of the car is 14658, coefficients of restitution is 0.2 numbers of triangles are 26232, mass being 1K kg. The result is shown in Fig. 5 and in our accompanying video[1].

We can evaluate the technical correctness of our method by comparing with FEM solution. Two identical cars crash the wall, one is computed with our method and other is computed with FEM. We have plotted the one dimension curve of two bonnet's midline computed by our method and FEM. The result is shown in Fig. 6. The x axis indicates the distance to the leftmost of bonnet and the y axis indicates the height of vertices to the ground. The curve shows that height of bonnets caused by wrap is in nearly 80% close. The second example is balls hitting on a metal shell. The metal shell consists of 11144 vertices, 19200 triangles, with friction coefficient being 0.4. 128 sub-shells were generated. We shoot balls to the metal shell to deform the metal shell. The mass of each ball is 100kg, the result is shown in Fig. 7.

In this example, we use the number of colliding points during collision to measure the efficiency. In the diagram (Fig. 8) the x-axis is the number of colliding points during one collision, y-axis means the time cost per frame. The extra computational cost is rotation and translation of colliding sub-shells, which is linearly increased and fairly small, when compared with the cost of collision detection and spring mass model. Run-time evaluation can be done with colliding

[1] Accompanying video at `www.youtube.com/watch?v=_1aoyEAXpwA`

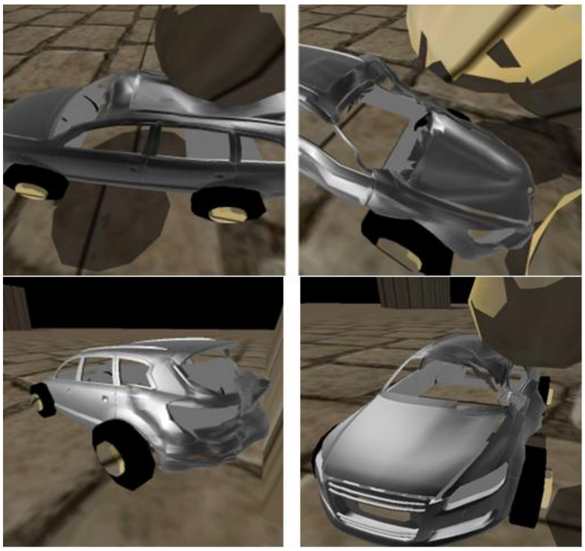

Fig. 5. A car is deformed by impact from walls and balls

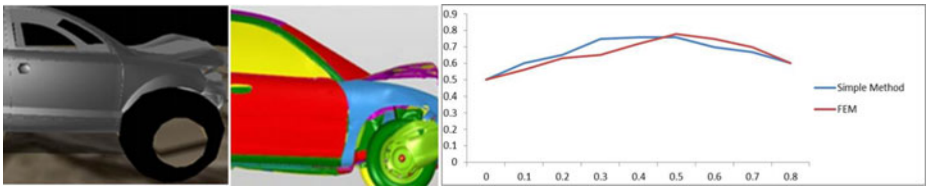

Fig. 6. The left picture is front part of car collision simulated by our method, the middle picture is the result computed by FEM, and the right picture is 1D curve comparison of two method

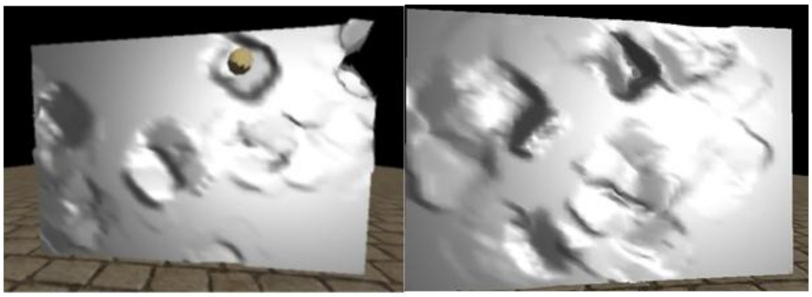

Fig. 7. Shooting balls to the metal shell

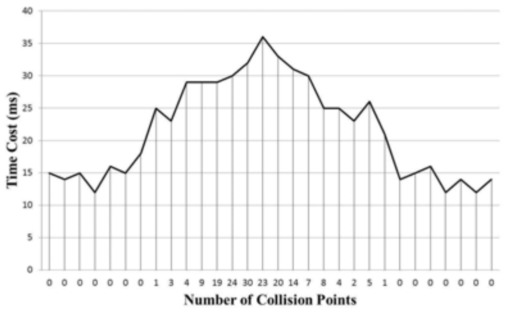

Fig. 8. Run time analysis

points taken into consideration. From Fig. 8, the time cost grows steadily when colliding points increased steeply. Our method can be used in real-time (24 frame per second) when the number of vertices does not exceeded 26500.

8 Conclusion

In this paper we have described a simple method to simulate metal deformation. The most important feature of this method is it's easy to understand and implement and it avoids solving the higher order matrixes smartly. In the meantime, the approximated result is still visually convincing enough to be integrated in games.

This method first uses K-Mean algorithm partition the entire metal shell into a number of sub-shells, during which vertices on different planes are considered further away from each other than vertices sharing the same plane.

In the second step, we compute the movement of each sub-shell as rigid-body. We adopt the classic penalty method to handle collision response. In this step, each sub-shell is displaced without any interaction with other sub-shell, therefore deformation we get in this step is local and discontinuity might occur.

In the final step, a spring-mass model is applied between all of the adjacent vertices to globalize the deformation, and to retain the continuity of the metal shell.

Acknowledgement. This work was supported by China University Student Scientific and Technological Innovation Projects (091056156) sponsored by Ministry of Education of the People's Republic of China. The authors would like to thank Professor Jie Yang for the support of mathematic theory.

References

1. Chadwick, J.E., Haumann, D.R., Parent, R.E.: Layered construction for deformable animated characters. In: Proc. of SIGGRAPH 1989, New York, NY, USA, pp. 243–252 (1989)
2. Fernandes, J.L.M., Martins, P.A.F.: All-hexahedral remeshing for the finite element analysis of metal forming processes. J. Finite Elements in Analysis and Design 43(8), 666–679 (2007)

3. Moore., M., Wilhelms, J.: Collision detection and response for computer animation. In: Proc. of SIGGRAPH 1988, New York, NY, USA, pp. 289–298 (1988)
4. Mamalis, A.G., Horvah, M., Branis, A.S., Manolakos, D.E.: Finite element simulation of chip formation in orthogonal metal cutting. J. Materials Processing Technology 110(1), 19–27 (2001)
5. Miller, G.S.P.: The motion dynamics of snakes and worms. In: Proc. of SIGGRAPH 1988, New York, NY, USA, pp. 169–173 (1988)
6. Mller, M., McMillan, L., Dorsey, J., Jagnow, R.: Real-time simulation of deformation and fracture of stiff materials. In: Proceedings of the Eurographic Workshop on Computer Animation and Simulation, New York, NY, USA, pp. 113–124 (2011)
7. O'Brien, J.F., Bargteil, A.W., Hodgins, J.K.: Graphical modeling and animation of ductile fracture. In: Proc. of SIGGRAPH 2002, New York, NY, USA, pp. 291–294 (2002)
8. O'Brien, J.F., Hodgins, J.K.: Graphical modeling and animation of brittle fracture. In: Proc. of SIGGRAPH 1999, New York, NY, USA, pp. 137–146 (1999)
9. Parker, R.G., O'Brien, J.F.: Real-time deformation and fracture in a game environment. In: Proc. of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2009), New York, NY, USA, pp. 165–175 (2009)
10. Platt, S.M., Badler, N.I.: Animating facial expressions. In: Proc. of SIGGRAPH 1981, New York, NY, USA, pp. 245–252 (1981)
11. Rodrigues, T., Pires, R., Dias, J.M.S.: D4MD: deformation system for a vehicle simulation game. In: Advances in Computer Entertainment Technology, pp. 330–333 (2005)
12. Shlafman, S., Tal, A., Katz, S.: Metamorphosis of polyhedral surfaces using decomposition. Eurographics Proceedings (CGF) 22(3), 219–228 (2002)
13. Tu, X., Terzopoulos, D.: Artificial fishes: physics, locomotion, perception, behavior. In: Proc. of SIGGRAPH 1994, New York, NY, USA, pp. 43–50 (1994)
14. Terzopoulos, D., Waters, K.: Physically-based facial modeling, analysis, and animation. J. Journal of Visualization and Computer Animation 12, 73–80 (1990)
15. Van Den Bergen, G.: Efficient collision detection of complex deformable models using AABB trees. J. Graph. Tools 2(4), 1–13 (1998)
16. Van Den Bergen, G.: A fast and robust GJK implementation for collision detection of convex objects. J. Graph. Tools 4(2), 7–25 (1999)
17. Waters, K.: A muscle model for animation three dimensional facial expression. In: Proc. of SIGGRAPH 1987, New York, NY, USA, pp. 17–24 (1987)
18. Waters, K., Terzopoulos, D.: Modelling and animating faces using scanned data. J. Visualization and Computer Animation 2(4), 123–128 (1990)
19. Yun, Z.X., Long, J.X., Guo, Q.W., Zhi, G.Y.: Vehicle crash accident reconstruction based on the analysis 3D deformation of the auto-body. J. Advances in Engineering Software 39(6), 459–465 (2008)