Noname manuscript No.
(will be inserted by the editor)

A Fast Method for Simulating Destruction and the Generated Dust and Debris

Takashi Imagire · Henry Johan · Tomoyuki Nishita

Received: date / Accepted: date

Abstract Simulating the destruction of objects due to collisions has many applications in computer graphics. Previous methods on the destruction of objects perform physically-based simulation of the fracture of objects into relatively large size fragments. However, if we observe the process of the destruction of objects, we can see that dust and fine debris are also generated. In particular, previous methods do not take into account the dust generation. In this paper, we present a unified framework for simulating destruction and the generated dust and various sizes of debris. Our method simulates destruction on three different scales: coarse fracture, fine debris and dust. We compute the distribution and the amount of fine debris and dust based on the fracture energy which is the energy that causes the object to be fractured. We demonstrate the effectiveness of the proposed method, in terms of the generated effects and the simulation speed, by showing the simulation results of destruction caused by the collision between objects.

Keywords destruction simulation · fine debris · dust generation · voronoi diagram

1 Introduction

Simulating the destruction of objects is important in order to increase the realism in computer graphics applications such as games. Several physically-based simulation methods that generate impressive results have been proposed. These methods simulate the destruction by breaking the objects into fragments. These methods, however, have a high computational cost. Moreover, the scale of destruction that can be handled by these methods ranges from several cm to several mm. On the other hand, in real destruction, fine debris and dust that consists of very fine particles in the range of several μm are also generated. Up to now, there are methods for visualizing dust but there is no method for generating dust due to the destruction of objects.

In this paper, we present a fast method to simulate destruction caused by the collision between objects. Different from previous methods [1,2], our method can efficiently generate dust and debris of various sizes. We propose a simulation on three different scales. First, we compute the destruction at a coarse scale based on the Extended Distinct Element Method (EDEM) [3]. To perform the simulation using the EDEM, each object is represented as a set of elements and nearby elements are connected using springs. Fracture is computed based on the physics-based simulation of the springs between the elements. Second, for each EDEM element, we compute the fracture energy that breaks it. Based on this energy, we determine the maximum size of the debris. If the maximum size is smaller than the EDEM element, we break the EDEM element into several smaller debris. In this case, we compute the location and the velocity of the debris based on those of the EDEM element. We simulate debris smaller than the EDEM elements using a particle simulation method. Third, the amount of dust is also computed based on the fracture energy. We simulate dust by performing fluid simulation. Our main contribution is the generation of dust and debris based on the fracture energy. For the destruction at a coarse

Takashi Imagire
The University of Tokyo and Namco Bandai Games Inc.
E-mail: imagire@nis-lab.is.s.u-tokyo.ac.jp

Henry Johan
Nanyang Technological University
E-mail: henryjohan@ntu.edu.sg

Tomoyuki Nishita
The University of Tokyo
E-mail: nis@nis-lab.is.s.u-tokyo.ac.jp

level, we can use methods other than EDEM as long as the fracture energy can be computed.

For visualization, we also perform the rendering in three different scales. First, we compute the 3D Voronoi diagram where the center of the EDEM elements corresponds to the Voronoi site. Based on the Voronoi faces, we create a mesh for each EDEM element. These meshes are rendered during the simulation. We propose an efficient method to render these meshes using the GPU. Second, when an EDEM element breaks into small debris, we use the scale-down version of its mesh to render the results of the particle simulation. This allows us to render small debris efficiently. Third, we adopt the technique for rendering smoke to visualize the dust.

This paper is organized as follows. In Section 2, we review the previous work. In Section 3, we describe the proposed method. In Section 4, we explain the details for implementing our method. Experimental results are shown in Section 5. We conclude and discuss about the possible future work in Section 6.

2 Related Work

In this section, we describe the previous work in destruction simulation, the relationship between fracture energy and the resulting debris, as well as dust and smoke generation.

2.1 Destruction Simulation

There is a large number of excellent existing research on the subject of destruction simulation. Terzopoulus and Fleischer [1] used a spring model to simulate physically-based fracturing for application in computer graphics. Norton et al. [4] simulated fracturing by calculating the stress inside an object according to a spring model. Smith et al. [5] achieved a high-speed computation of fracturing by precalculating the fracture faces of the object. Müller et al. [6] used a finite element method to compute physically-based fracturing of objects in real-time.

For representing small debris, the method by Zhang et al. [7] used fine tetrahedral subdivision to create the fragments. However, the computational cost of this method to generate extremely fine debris and dust is very high.

2.2 Fracture Energy and Debris Size

The research on the distribution of debris size after crushing an object has been done for a very long time. Rosin and Rammler [8] performed crushing experiments using a tube mill and sorted the resulting coal fragments through a sieve to determine their size distribution. Gaudin [9] and Schuhmann [10] also proposed distribution functions applicable to the comminution production of minerals.

Objects will break into smaller fragments when they are crushed with more energy. Some famous laws governing the relation between the crushing workload and the debris size have been proposed by Rittinger [11], Kick [12] and Bond [13]. The Bond work index has been used widely in estimating the energy requirements of mineral grinding. In our research, we use these results to estimate the distribution of dust and small debris.

2.3 Dust and Smoke Generation

Chen et al. [17] presented a method to generate dust caused by the pressure of the moving air behind a traveling vehicle. On the subject of smoke generation, there are work on fuel combustion that takes the underlying chemical reactions into account [15,16]. However, these methods cannot be applied to generate dust resulting from the destruction of objects.

3 Proposed Method

We model the destruction of objects by combining three different scales of simulation. Below, we describe the characteristics of each scale of the simulation and the rendering method.

3.1 Destruction at Coarse Scale

We consider the destruction of object due to mechanical impact. For the coarse scale simulation, we use an extension of the Distinct Element Method or the Discrete Element Method (DEM) [18,19] designed to solve the dynamic behavior of objects, called the Extended Distinct Element Method (EDEM) [3]. In DEM, objects are represented by a set of elements and by solving the equation of motion describing the interaction of the elements, the behavior of the entire system is calculated. Interaction in DEM is modeled by contact force due to linear springs that exhibit a repelling force when compressed by the elements, as well as dashpots which provide damping of the relative velocity.

EDEM introduces connecting force due to *pore springs* in addition to the contact force (see Figure 1). The pore springs simulate the effect of a gap-filling material such as mortar. Neighboring elements are connected by pore

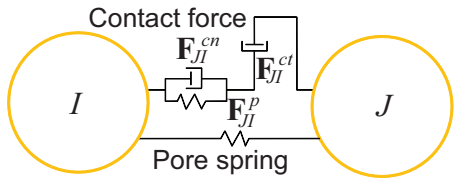


Fig. 1 The force between two EDEM elements.

springs. Due to the force of the pore springs, the object retains its original shape. The pore springs disappear when the connected elements move farther apart than a certain distance or differ in orientation more than a given threshold due to an external force. This represents a situation when a spring is broken and fracture is taken place.

3.1.1 Initialization of EDEM elements

We define the EDEM elements as spheres of radius r. To arrange the EDEM elements inside the object, we use the following method.

1. Represent the original object as a closed surface model.
2. Arbitrarily arrange the EDEM elements inside the object. The elements are allowed to overlap at this point.
3. Move the elements by performing the EDEM simulation (see Section 3.1.2). However, we only consider the contact force in this simulation.
4. Perform collision detection between the object's surface and the elements, making sure that the elements always stay inside the object.
5. Repeat (3)-(4) until the elements are stabilized.
6. Construct a Delaunay diagram from the set of elements and put the pore springs on the Delaunay edges that connect the elements.

3.1.2 EDEM Simulation

The position $\mathbf{x}_I$ and velocity $\mathbf{v}_I$ of element I can be found using Newton's equation of motion as follows:

$$M\frac{d\mathbf{v}_I}{dt} = \sum_{J\in contact} \mathbf{F}_{JI}^{c} + \sum_{K\in pore} \mathbf{F}_{KI}^{p} + M\mathbf{g}, \tag{1}$$

$$\frac{d\mathbf{x}_I}{dt} = \mathbf{v}_I. \tag{2}$$

Here, $\mathbf{g}$ is the gravitational vector, $\mathbf{F}_{JI}^{c}$ is the contact force, $\mathbf{F}_{KI}^{p}$ is the force due to the pore springs and M is the element's mass. *contact* contains elements $\{I, J\}$ if they are closer than $2r$, the diameter of a single element, while *pore* contains a pair of elements $\{I, K\}$ when they are connected by a pore spring.

The contact force $\mathbf{F}_{JI}^{c}$ is made up of a normal component $\mathbf{F}_{JI}^{cn}$ and a tangential component $\mathbf{F}_{JI}^{ct}$. $\mathbf{F}_{JI}^{c} = \mathbf{F}_{JI}^{cn} + \mathbf{F}_{JI}^{ct}$. The *normal* direction is defined as a direction towards the center of a pair of elements $\{I, J\}$ and the tangential direction points towards a plane perpendicular to the normal. The normal component of the contact force between elements I and J is given by $\mathbf{F}_{JI}^{cn} = -k^{cn}\Delta\mathbf{x}_{JI}^{cn} - \eta^{cn}\mathbf{v}_{JI}^{cn}$. Here, k^{cn}, η^{cn} are the linear spring's spring constant and the dashpot's decay constant, respectively. $\Delta\mathbf{x}_{JI}^{cn}$ is the change in the normal direction for the pair of elements $\{I, J\}$ in a position where they are not in contact. $\mathbf{v}_{JI}^{cn}$ is the normal component of the relative velocity vector. The tangential component of the contact force between elements I and J is given by $\mathbf{F}_{JI}^{ct} = -\eta^{ct}\mathbf{v}_{JI}^{ct}$, where $\mathbf{v}_{JI}^{ct}$ is the change of the relative speed vector subtracted the normal component of the change.

The force due to the pore springs, $\mathbf{F}_{KI}^{p}$, is defined as a restoration force to keep the shape of the object. The restoration force in each element is caused in the direction opposite to the change of a relative position with another element, $\mathbf{F}_{KI}^{p} = -k^{p}\Delta\mathbf{x}_{KI}$. Here, $\Delta\mathbf{x}_{KI}$ represents the changes of the relative position with respect to the initial state and k^{p} is a pore spring's spring constant

Finally, we also need to handle the interaction with other objects. Planes like the ground are represented using a polygon mesh and the contact force between the EDEM elements of the simulated objects and these polygons are used to calculate the behavior of the objects. Other objects are modeled using a single EDEM element or a set of EDEM elements connected by pore springs. The collision response between EDEM objects are based on the contact force, not by the force of the pore springs.

3.2 Fine Debris Generation and Simulation

To generate fine debris using the EDEM simulation, we must set the size of the EDEM elements to be very small. This approach is not practical because it results in an extremely large number of elements, making it difficult to realize fast simulation. To efficiently generate fine debris, we break the EDEM elements into several debris when the energy that causes the fracture is big. We then use the particle simulation method [20] to simulate the debris.

In order to determine whether the EDEM elements break into fine debris, we introduce fracture energy. We define the fracture energy W_I corresponding to the

EDEM element I as the energy of the pore spring.

$$W_I = \sum_{K \in pore} \frac{k^p}{2} \parallel \Delta \mathbf{x}_{KI} \parallel^2, \quad (3)$$

where $\| \cdot \|$ is the Euclidean norm of a vector.

We used the Bond's law [13], well-known as a measurement of the work index representing crushing efficiency, to calculate the fineness of the debris resulting from objects broken by fracture energy. The Bond's law expresses the work W required to fracture an object of size R_f into debris of size R_{max} with the formula $W = C_B \left(1/\sqrt{R_{max}} - 1/\sqrt{R_f}\right)$, where C_B is a constant coefficient. The second term on the right side of the equation can be ignored if the initial size of the object before fracturing is considered to be infinite. Therefore, following the Bond's law, the maximum size R_{max} of the debris produced during the destruction can be determined from a work W_I that causes the destruction:

$$R_{max} = \left(C_B / W_I\right)^2 . \quad (4)$$

In our method, we consider the EDEM element I to be broken when R_{max}, the maximum fragment size due to the fracture energy, is smaller than the EDEM element's size r.

Broken elements are taken out of the EDEM simulation and inserted into the particle simulation as multiple debris. We determine the size of the debris based on the Gaudin−Schuhmann distribution [9,10], which is widely used in mineral crushing. The Gaudin−Schuhmann distribution expresses the distribution of the debris size of a crushed object as the retained weight fraction [%] of debris smaller than R_{max} with the formula $U_{R_{max}} = 100 \left(R_{max}/R_e\right)^m$, where debris size modulus R_e and debris size distribution modulus m depend on the material. m takes on values of 1.0 or less.

We assume that the element I is broken into N pieces of debris where N is the number of faces on the mesh that corresponds to the element I (see Section 3.4). The size of the n-th debris out of a total of N debris is calculated as follows (see Appendix A):

$$R = \left(\frac{(N-n) R_{dust}^{-(3-m)} + n R_{max}^{-(3-m)}}{N} \right)^{-1/(3-m)}, \quad (5)$$

where R_{dust} is the size of debris that considered as dust particle (in the range of several μm).

In the particle simulation, debris is treated as point entities without size. The initial position of the debris is at the center of the original element and its velocity is given by adding a relative velocity to the original element velocity. The direction of the relative velocity is set such that the particles are scattered in different directions from each other. Specifically, the normal vectors of the faces on the mesh that corresponds to the element (see Section 3.4) are used as the directions of the relative velocities. The relative speed of the debris is calculated as the speed of the original element multiplied by a user-specified value. In this way, we can represent the scattering of debris when an element is broken into small pieces. To visualize the debris, we use the scaled-down mesh of its corresponding EDEM elements.

3.3 Dust Generation and Simulation

When an EDEM element is broken, we determine the density of the dust by estimating the amount of dust particles. We consider debris smaller than R_{dust} to be dust particles. When determining the distribution of debris size using the Gaudin−Schuhmann distribution, for a maximum debris size R_{max}, the ratio of debris smaller than R_{dust} is given by the following formula:

$$P = U_{R_{dust}} / U_{R_{max}} = \left(R_{dust}/R_{max}\right)^m . \quad (6)$$

The ratio of dust P generated by the fracture energy W_I is calculated from Eq. (4) and (6) as follows:

$$P = C\, W_I^{2m}. \quad (7)$$

We determine the values of the coefficient $C = \left(R_{dust}/C_B^2\right)^m$ and the particle size distribution modulus m from experiments.

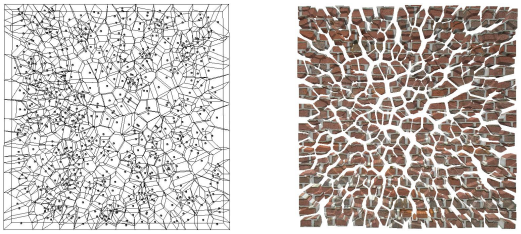
We simulate the generated dust using a grid-based fluid simulation method [14]. We calculate the amount of dust in proportion to Eq.(7) and add that amount to the density field at the grid cell nearest to the center of the corresponding EDEM element. We set the fluid's velocity at the grid cell to be the velocity of the corresponding EDEM element.

3.4 Visualization of the EDEM Simulation

In the EDEM simulation, objects are represented as a set of EDEM elements. After we arranged the EDEM elements inside the object (Section 3.1.1), for each EDEM element, we compute a mesh that represents the region inside the object corresponding to this EDEM element. We compute the mesh as follows.

1. Calculate a 3D Voronoi diagram with the center of the elements as Voronoi sites (see Figure 2(a)).
2. Create a mesh for each element from its Voronoi faces (see Figure 2(b)).

When rendering each mesh based on the position and the orientation of its corresponding EDEM element I, cracks may appear between the meshes even if the elements only move slightly (see Figure 3). To eliminate this problem, when a pore spring is still intact



(a) The Voronoi diagram. (b) The generated meshes.

Fig. 2 The Voronoi diagram and the meshes generated from it. The meshes are slightly scaled down to show their structure more clearly.

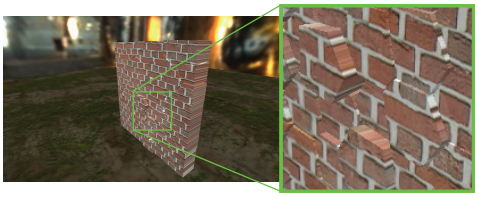


Fig. 3 Meshes rendered using only the position and the orientation of the corresponding elements. Cracks between the meshes appear even when their corresponding elements move only by a small offset.

between two elements, the corresponding meshes are displayed as being smoothly connected and only when a spring is removed we render the meshes as a separate disconnected parts. This way, no cracks occur in the visualization.

Specifically, the following formula is used to obtain $\mathbf{p}_A$, the coordinates of vertex A of the mesh that belongs to element I:

$$\mathbf{p}_A = \sum_{J \in \Omega_A} \omega_A^J \mathbf{x}_J / \sum_{I \in \Omega_A} \omega_A^J. \quad (8)$$

Here, Ω_A is the set of EDEM elements whose meshes share the vertex A, $\mathbf{x}_J$ is the position of element J and ω_A^J is a weight whose value is one or zero depending on whether or not there is a pore spring connecting elements J and I. The value of ω_A^I is one.

4 Implementation

In this section, we describe the implementation details of each scale of the simulation and the visualization method.

4.1 EDEM Simulation

We perform the EDEM simulation on the CPU. In the EDEM simulation, in order to quickly detect which elements are affected by the contact force, we subdivide the space into a uniform grid where the grid size is set to the elements' diameter. We store each element in the grid cell corresponding to its center and perform intersection tests only against other elements in the corresponding grid cell or one of its surrounding cells, checking a total of 27 grid cells. To enable fast lookup of elements connected by a pore spring, each element stores the indices of its connected pairs.

The integration through time is accomplished through the following steps:

1. Calculate the external force of gravity etc.
2. Move the elements using the external force.
3. Move the elements using the contact force.
4. Move the elements using the pore springs force.
5. Perform collision detection and response with other EDEM objects.
6. Perform collision detection and response with polygonal objects forming the floor.

4.2 Particle Simulation

We perform the particle simulations of small debris on the CPU. When an EDEM element is determined to have been broken, a number of particles corresponding to the number of Voronoi faces are created. During the particle simulation, for performance reasons and for saving memory, particles are displayed by rendering the scaled-down version of the mesh of their corresponding EDEM elements. The scaling coefficients are computed by dividing the values of Eq.(5) and the diameter of the EDEM elements.

4.3 Dust Simulation

We use a grid-based fluid simulation method implemented on the GPU [14] to calculate the fluid's velocity and density fields. When an element breaks due to the fracture energy in the EDEM simulation, the amount of dust to be generated is calculated using Eq.(7) and inserted into the fluid simulation by adding density to the corresponding grid cell. Furthermore, by adding an animated noise function [21], we achieved more realistic images.

4.4 Visualization of the EDEM Simulation

As mentioned in Section 3.4, we compute a mesh for each EDEM element. We propose the following GPU-based method to efficiently render these meshes.

During rendering, when processing the vertices of the mesh, the relative coordinates from the center of the adjoining EDEM elements to the vertex position are

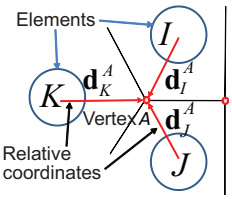


Fig. 4 The relative coordinates for the vertex A.

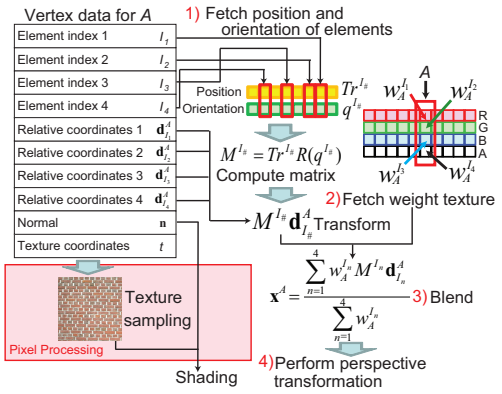


Fig. 5 Vertex data and the flow of computation.

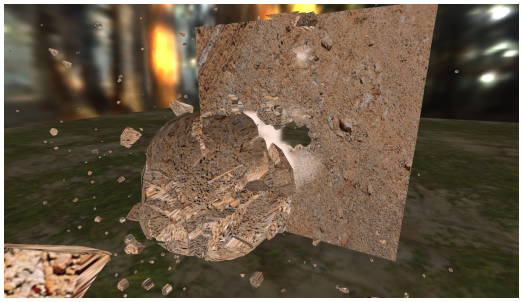


Fig. 7 The destruction of two fragile objects.

Table 1 Performance under various settings.

	320×180	640×360	960×540	1280×720
fps	9.0	8.9	8.9	8.8

(a) Screen resolution

	128	256	512	1024	2048
fps	11.0	10.7	10.7	10.6	8.8

(b) Number of EDEM elements

	8^3	16^3	32^3	64^3
fps	9.1	9.1	9.0	8.8

(c) Resolution of fluid simulation

	128	256	512	1024	2048
fps	320	160	75	30	9.1

(d) Number of EDEM elements (without rendering)

required for each vertex (see Figure 4). Since an EDEM element may be accessed from several vertices, we store the elements' position and orientation in a texture in graphics memory in order to retrieve them efficiently by sampling the texture during vertex processing.

Additionally, we also need to know whether or not the adjoining elements for a given vertex are connected by pore spring. We assume that every tetrahedron in the Voronoi diagram's dual graph, the 3D Delaunay diagram, has a set of points in a general position; that is, no four points are on the same plane and no five points are on the same sphere. Then, the vertex of a mesh corresponds to the center of mass of a tetrahedron in the Delaunay diagram or a vertex on a face at the surface of the object, therefore they are equidistant to at most four EDEM elements. Consequently, we create a texture of the same size as the number of vertices called the *weight texture* and store at each texel the connectivity information of the four elements belonging to the corresponding vertex, one for each of the four available channels (RGBA).

When the pore spring disappears between a pair of elements $\{I, J\}$ in the EDEM simulation, the weight texture is updated at the texels corresponding to the vertices of the polygon created from the Voronoi face between the Voronoi sites represented by elements I and J. This way, only a small amount of data needs to be sent to the GPU, enabling efficient rendering. When there are less than four elements equidistant from a given vertex, the remaining weight values can be set to zero in the weight texture.

Specifically, the vertex processing consists of the following steps (see Figure 5):

1. Fetch the positions and the orientations of the adjoining elements using the indices of the adjoining elements.
2. Sample the weight texture to determine whether adjoining elements are connected.
3. Calculate the vertex position from the relative coordinates to the adjoining elements using Eq.(8).
4. Perform perspective transformation to calculate the screen coordinates.

5 Results

We used a machine with an Intel Core 2 Extreme X6800 2.93GHz and a NVIDIA GeForce 8800 GTX in the experiments. To accelerate the EDEM simulation, we made use of the dual core architecture of the CPU by implementing the EDEM using the OpenMP. As the graphics API, we used Direct3D 10. The screen resolution was 1280×720. The resolution of the simulation domain for dust (fluid) simulation was 64^3.

Figure 6 shows the results of the proposed method. A rigid ball coming from the back of the scene breaks through a wall. The wall has 2048 EDEM elements. The

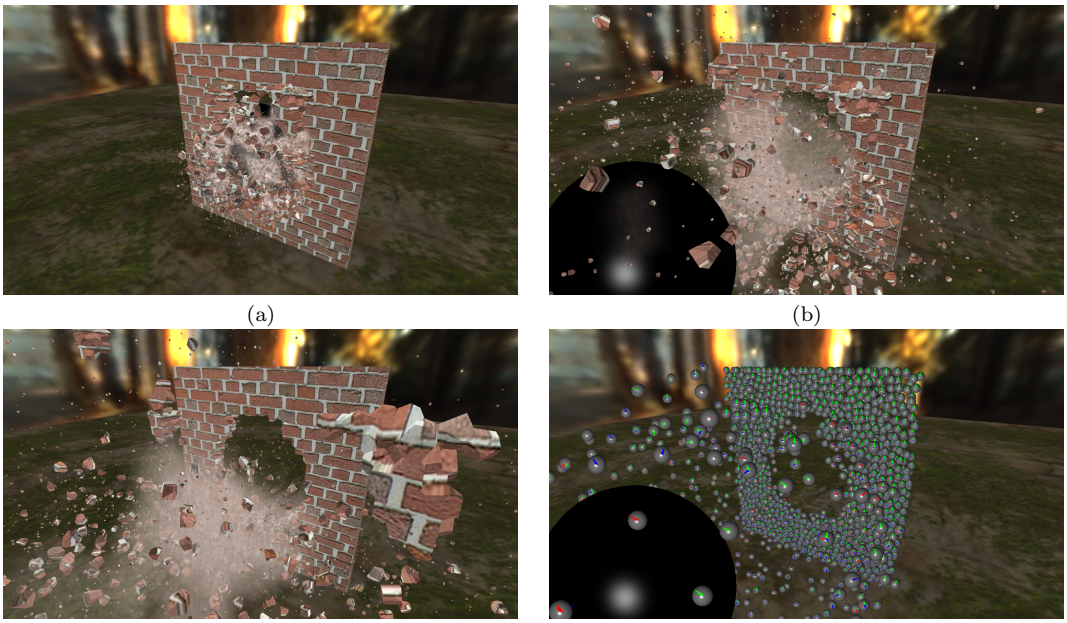


Fig. 6 The destruction of a wall due to a rigid ball can be seen in (a)-(c). We can see the effects of dust and various sizes of debris. In (d) we can see the EDEM elements.

radius r of the EDEM elements is 1.7×10^{-2}. Here, the height, the width and the depth of the wall are 1.0, 1.0 and 8.0×10^{-2}, respectively. The rigid ball is represented using one EDEM element with a large radius. The size R_{dust} of a maximum dust particle is 1.0×10^{-6}. When the rigid ball collides with the wall (see Figure 6(a)-(c)), the interaction between the ball and the EDEM elements of the wall is computed through the contact force. Figure 6(d) shows the EDEM elements of Figure 6(b). In this experiment, the rigid ball destructed a part of the wall and various sizes of debris and dust were generated. In this simulation, we achieved 0.5 fps. If we do not use the advected textures [21] during the dust simulations, the performance is up to 8.8 fps.

Table 1 shows the performance when the resolution of screen (a), the number of elements (b) and the resolution of the fluid simulation (c) are changed. We can see that both the processing loads of the simulation and of the rendering are high since the values hardly change. If we turn rendering off, the performance is proportional to the number of the EDEM elements (see Table 1(d)). This shows that the rendering is the bottleneck when the number of the EDEM elements is low.

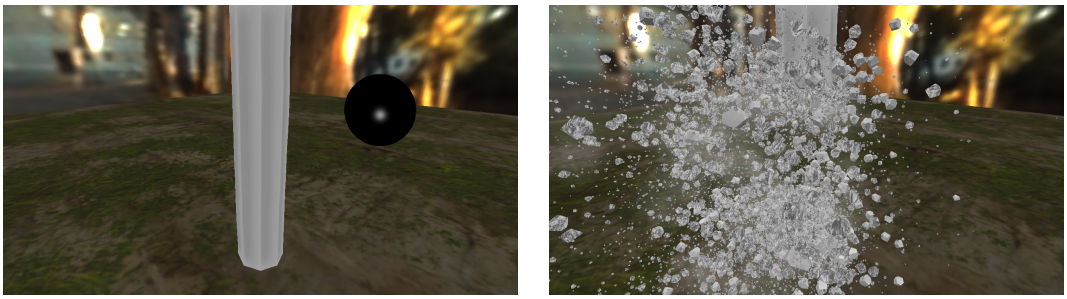
Figure 7 shows other result of destruction simulation of two fragile objects. The wall and the ball are composed of 2048 and 256 EDEM elements, respectively. Different from the results in Figure 6, in this experiment, the ball crumbles after the collision.

Figure 8 shows other result of destruction simulation of a column due to a rigid ball. The column is composed of 2048 EDEM elements.

6 Conclusion

We have presented a fast method for destruction simulation due to the collision between objects. Different from previous approaches, our method is able to generate the effects of dust and various sizes of debris. In order to generate these effects, we perform the simulation on three different scales. Firstly, we compute the coarse scale fracture on the objects based on the EDEM. Secondly, based on the fracture energy, we predict the size distribution of the debris and use particle simulation to animate fine debris. Thirdly, we use the fracture energy to determine the amount of the generated dust and perform fluid simulation to animate the dust. Through experimental results, we have shown that our method can render destruction phenomena with wide range of effects. In addition, our method achieved interactive frame rates for the simulation involving moderate number of EDEM elements. The proposed dust and debris generation can be combined with other destruction simulation methods such as [6].

Our method can be extended in several ways. One would be the fast simulation of destruction involving explosive materials. In our current method, we do not take the temperature of the surrounding into account. How-



(a) (b)

Fig. 8 The destruction of a column due to a rigid ball.

ever, in the presence of explosive materials, we must be able to handle the rise in the temperature and also the shockwave. As mentioned above, for a moderate number of EDEM elements, our current implementation achieves interactive frame rates. To model a larger scene with a large number of EDEM elements or to apply our method in real-time application such as games, we still need to further accelerate the simulation.

References

1. Terzopoulos D., Fleischer K.: Modeling inelastic deformation: viscolelasticity, plasticity, fracture. *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, 269–278, (1988).
2. O'Brien J., Hodgins J.: Graphical modeling and animation of brittle fracture. *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, 137–146, (1999).
3. Meguro K., Hakuno M.: Fracture analyses of concrete structures by the modified distinct element method. *Structural Eng./Earthquake Eng., JSCE 6, 2*, 283–294, (1989).
4. Norton A., Turk G., Bacon B., Gerth J., Sweeney P.: Animation of fracture by physical modeling. *The Visual Computer 7, 4*, 210–219, (1991).
5. Smith J., Witkin A., Baraff D.: Fast and controllable simulation of the shattering of brittle objects. *Computer Graphics Forum 20, 2*, 81–91, (2001).
6. Müller M., McMillan L., Dorsey J., Jagnow R.: Real-time simulation of deformation and fracture of stiff materials. *EUROGRAPHICS 2001 Computer Animation and Simulation Workshop*, 27–34, (2001).
7. Zhang N., Zhou X., Sha D., Yuan X., Tamma K., Chen B.: Integrating mesh and meshfree methods for physics-based fracture and debris cloud simulation. *Symposium on Point-based Graphics*, 145–154, (2006).
8. Rosin P., Rammler E.: The laws governing the fineness of powdered coal. *J. Inst. Fuel 7, 31*, 29–36, (1933).
9. Gaudin A.: An investigation of the crushing phenomena. *Petrolenm Development and Technology 73*, 253–316, (1926).
10. Schuhmann R.: Principles of comminution: I, size distribution and surface calculations. *AIME Technical Publication*, 1189, (1940).
11. von Rittinger P.: Lehrbuch der aufbereitungskude. *Berlin: Ernst and Korn* (1867).
12. Kick F.: Das Gesetz der proporionalem Widerstand und seine Anwending. *Felix, Leipzig*, (1885).
13. Bond F.: The third theory of comminution. *Trans. AIME 193, 2*, 484–494, (1952).
14. Crane K., Llamas I., Tariq S.: Real-time simulation and rendering of 3D fluids. *GPU Gems 3 2007*, 633–675, (2007).
15. Stam J., Fiume E.: Depicting fire and other gaseous phenomena using diffusion processes. *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, 129–136, (1995).
16. Nguyen D., Fedkiw R., Jensen H.: Physically based modeling and animation of fire. *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, 721–728, (2002).
17. Chen J., Fu X., Wegman E.: Real-time simulation of dust behavior generated by a fast traveling vehicle. *ACM Transactions on Modeling and Computer Simulation 9, 2*, 81–104, (1999).
18. Cundall P., Strack O.: A discrete numerical model for granular assemblies. *Geotechnique 29, 1*, 47–65, (1979).
19. Bell N., Yu Y., Mucha P.: Particle-based simulation of granular materials. *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 77–86, (2005).
20. Reeves W.T.: Particle system - a technique for modeling a class of fuzzy objects. *Computer Graphics 17, 3, SIGGRAPH 83*, 359–376, (1983).
21. Neyret F.: Advected textures. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 147–153, (2003).

A Calculation of Debris Size

We calculate the size of the nth debris in a group of N debris based on the Gaudin−Schuhmann distribution. Assuming that the debris are spheres with radius R and constant density ρ, the total mass M of the N debris can be written as follows.

$$M = \sum_{n=1}^{N} \rho \frac{4\pi}{3} \{R(n)\}^3 \rightarrow \int \rho \frac{4\pi}{3} R(n)^3 dn. \tag{9}$$

The right-hand side was obtained by taking the continuous limit assuming that N is very large.

When the debris size follows the Gaudin−Schuhmann distribution, $100\left(\frac{R}{R_e}\right)^m \propto \int \rho \frac{4\pi}{3} R(n)^3 dn$. By differentiating both sides and solving this expression for R, we get $R = (R_0 + R_1 n)^{-1/(3-m)}$, where R_0, R_1 are constant coefficients. The coefficients are calculated, assuming the debris size to be R_{dust} at $n = 0$ and R_{max} at $n = N$. The size of the nth debris can finally be obtained in the form of expression (5).