Real-Time Simulation of Ductile Fracture with Oriented Particles

Min Gyu Choi
Kwangwoon University
mgchoi@kw.ac.kr

Abstract

This paper proposes a practical approach for real-time simulation of large deformation and ductile fracture. We adopt the oriented particles approach for robust and efficient simulation of large deformation and develop a practical model for plastic flow and fracture. The proposed method finds the optimal rotation and the optimal stretch in shape matching. The newly introduced optimal stretch leads to a material strain that can be employed in the plastic flow and fracture criteria. We also propose a GPU skinning method for real-time rendering of a deformed, fractured visual mesh. Experimental results show that the proposed method can robustly simulate large, elastoplastic deformation and ductile fracture of large visual meshes in real time.

Keywords: deformation, plasticity, ductile fracture, shape matching, oriented particles

1 Introduction

Physics-based simulation is one of the most important techniques in film and game production. In this paper, we focus on physics-based deformation and ductile fracture. These phenomena have been extensively studied in computational mechanics, material science and computer graphics. However, it is still a challenging problem to robustly produce visually convincing, stable deformation and fracture in real time.

Most real-time fracture applications utilize pre-fractured chunks of rigid bodies that shatter when applied with impacts larger than a predefined threshold [1]. Thus, they do not reveal elastic deformation. On the other hand, finite element methods based on continuum mechanics can simulate accurate deformation and fracture [2–6]. However, in real-time applications, robustness and speed are often more essential than accuracy. On that account, we focus on the shape matching techniques [7–9] that can produce visually convincing deformation in a robust and efficient way. In particular, the oriented particles approach [8] can create various types of deformable objects with a small number of simulation particles.

Based on the oriented particles approach [8], we develop a practical model for plastic deformation and fracture that accommodates material failure at excessively stretched or compressed material points. We extend the shape matching to find not only the optimal rotation but also the optimal stretch between the rest and deformed particles. The newly introduced optimal stretch is employed to transfer elastic deformation into plastic deformation with a multiplicative decomposition of deformation [3, 10]. The remaining elastic deformation is supplied to the local fracture scheme formulated with the maximum strain criterion and fracture toughness [11]. In addition, we propose a GPU skinning method for real-time rendering of a deformed, fractured visual mesh composed of pre-fractured fragments corresponding to the oriented particles. This simple and robust method can produce large elastoplastic deformation and fracture of large visual meshes in real time.

2 Related Work

Physics-based simulation of deformation and fracture has a long history in computer graphics since the pioneering work of Terzopoulos et al. [12]. A comprehensive survey in this field can be found in [13]. One of the most popular approaches to simulating deformable solids is to employ a finite element method with linear tetrahedral elements. To reduce the linearization artifacts in large deformation, corotational formulations have been successfully employed [14]. Even flat or inverted elements can be simulated robustly if rectified as suggested in [3].

Excessive deformation leads to fracture involved in crack initiation and propagation. O'Brien and Hodgins [6] introduced a finite element method to simulate brittle fracture. This work was further extended to ductile fracture by incorporating plasticity [5]. Molino et al. [4] proposed a virtual node algorithm that removes ill-conditioned slivers in remeshing. Pauly et al. [15] developed a meshless method for fracturing ductile materials. Basically, these approaches are not intended for real-time applications due to the computational burden in remeshing and time stepping. Nevertheless, Parker and O'Brien [16] successfully applied a corotational finite element method to real-time fracture in computer games by employing relatively coarse tetrahedral meshes together with splinters for geometric details of elements.

Deformable materials can be pre-fractured and subsequently broken apart based on a variety of rules. This approach is handy and fast and thus popular in computer games and feature films [1]. In particular, Voronoi diagrams are widely used to generate fracture patterns at runtime. Su et al. [17] employed a fracture pattern aligned with an impact location at runtime. Bao et al. [2] proposed a fracturing method for rigid materials based on a quasi-static stress analysis and a Voronoi-like fracture pattern. Zheng and James [18] also employed a similar framework, but the fracture pattern was incrementally constructed to maintain a bounded fracture energy. However, Voronoi-like fracturing instantaneously gives rise to shattered fragments without dynamic deformation and crack propagation. Unlike our method, these approaches are designed for rigid or brittle materials.

Meshless deformation based on shape matching is fast and robust enough for real-time applications [7], but shape matching with a single transformation is restricted to modest deformation. Its extension to lattice shape matching can generate large deformation in a more visually convincing way [9]. In this extension, fracture is addressed by simply cutting excessively stretched edges. This work is further extended to adaptive shape matching with octree sampling and interactive cut with topological changes [19]. Plastic deformation is also supported with plasticity transformation matrices introduced in [7]. Combined with position-based dynamics [20], shape matching with oriented particles [8] enables complex dynamic deformation to be simulated in an efficient and robust way, even with a small number of particles. Our method is based on these shape matching approaches. However, none of them addresses ductile fracture.

Least-squares estimation of optimal rotation from two particle sets is at the heart of the shape matching approach. In contrast to others, Umeyama's formulation [21] deals with an additional scalar parameter for uniform scaling under the optimal rotation. We extend this formulation to compute not only the optimal rotation but also the optimal stretch. The optimal stretch is represented as a symmetric matrix so as to estimate anisotropic stretching that leads to a material strain for plastic deformation and fracture. To our knowledge, the optimal stretch matrix has been formulated for the first time in this paper.

3 Ductile Fracture with OP

Our method is based on shape matching with oriented particles. We briefly review shape matching with a single rigid transformation [7] and its extension with oriented particles [8]. Differently from the previous methods, we introduce an additional *optimal stretch* with respect to an *optimal rotation* in the shape matching procedure. The optimal stretch is employed in the plastic flow and fracture criteria. In addition, we enforce angular momentum conservation at each shape matching group to rectify ghost forces that can be exaggerated with a large time step.

3.1 Shape Matching Foundation

Optimal rotation. Given n vertices of a surface mesh with mass m_i, there exists a rotation that optimally matches the rest position $\bar{\mathbf{x}}_i$ with the current position $\mathbf{x}_i$ in a least-squares sense. This can be formulated as seeking the *optimal rotation* $\mathbf{R}$ that minimizes $\sum m_i \|\mathbf{R}\mathbf{q}_i - \mathbf{p}_i\|^2$, where $\mathbf{p}_i = (\mathbf{x}_i - \mathbf{c})$ and $\mathbf{q}_i = (\bar{\mathbf{x}}_i - \bar{\mathbf{c}})$ are the relative positions of the particles with respect to the centers of mass $\mathbf{c}$ and $\bar{\mathbf{c}}$ at the current and rest states, respectively. $\mathbf{R}$ can be obtained with polar decomposition of the covariance matrix [21]:

$$\mathbf{A}_{pq} = \sum m_i \mathbf{p}_i \mathbf{q}_i^{\mathrm{T}} = \mathbf{R}\mathbf{S}_{pq}. \qquad (1)$$

The shape matching procedure given in [7] employs $\mathbf{R}$ to compute the goal position of every particle, $\mathbf{g}_i = \mathbf{R}\mathbf{q}_i + \mathbf{c}$, and then moves the current position $\mathbf{x}_i$ toward the goal position $\mathbf{g}_i$ by the amount proportional to the material stiffness.

Optimal stretch. Plastic deformation and fracture require a stretch measure that can lead to a strain; We find the *optimal stretch* $\mathbf{S}$ that minimizes

$$\sum m_i \|\mathbf{R}\mathbf{S}\mathbf{q}_i - \mathbf{p}_i\|^2 \qquad (2)$$

with the optimal rotation $\mathbf{R}$ given in Equation (1) and the symmetric constraint $\mathbf{S} = \mathbf{S}^{\mathrm{T}}$ that guarantees $\mathbf{S}$ to be a stretch matrix. It can be obtained by minimizing an unconstrained version,

$$f = \sum m_i \|\mathbf{R}\mathbf{S}\mathbf{q}_i - \mathbf{p}_i\|^2 + tr(\mathbf{l} \times (\mathbf{S} - \mathbf{S}^{\mathrm{T}})),$$

where $\mathbf{l}$ is a vector of Lagrange multipliers. The second term of f represents the condition for $\mathbf{S}$ to be symmetric. Differentiating f with respect to $\mathbf{S}$ gives the following equation (*i.e.*, $\frac{\partial f}{\partial \mathbf{S}} = 0$):

$$\sum m_i \big(2\mathbf{S}\mathbf{q}_i\mathbf{q}_i^{\mathrm{T}} - 2\mathbf{R}^{\mathrm{T}}\mathbf{p}_i\mathbf{q}_i^{\mathrm{T}}\big) - 2\mathbf{l}\times = 0, \quad (3)$$

where we exploited $\frac{\partial}{\partial \mathbf{S}}\big(\mathbf{p}_i^{\mathrm{T}}\mathbf{S}\mathbf{q}_i\big) = \mathbf{p}_i\mathbf{q}_i^{\mathrm{T}}$ and $\frac{\partial}{\partial \mathbf{S}}\big(\mathbf{q}_i^{\mathrm{T}}\mathbf{S}^{\mathrm{T}}\mathbf{S}\mathbf{q}_i\big) = 2\mathbf{S}\mathbf{q}_i\mathbf{q}_i^{\mathrm{T}}$. Here, $\mathbf{l}\times$ denotes a skew symmetric matrix of $\mathbf{l}$ such that $(\mathbf{l}\times)\mathbf{a} = \mathbf{l} \times \mathbf{a}$ for an arbitrary vector $\mathbf{a}$. Substituting the covariance matrices $\mathbf{A}_{pq}$ given in Equation (1) and $\mathbf{A}_{qq} = \sum m_i \mathbf{q}_i \mathbf{q}_i^{\mathrm{T}}$ into Equation (3) gives

$$\mathbf{S}\mathbf{A}_{qq} - \mathbf{R}^{\mathrm{T}}\mathbf{A}_{pq} = \mathbf{l}\times. \qquad (4)$$

By transposing both sides of Equation (4), we obtain the following equation:

$$\mathbf{A}_{qq}^{\mathrm{T}}\mathbf{S}^{\mathrm{T}} - \mathbf{A}_{pq}^{\mathrm{T}}\mathbf{R} = -\mathbf{l}\times. \qquad (5)$$

Adding each side of Equations (4) and (5) cancels out the Lagrange multipliers and results in an equation,

$$\mathbf{S}\mathbf{A}_{qq} - \mathbf{R}^{\mathrm{T}}\mathbf{A}_{pq} + \mathbf{A}_{qq}^{\mathrm{T}}\mathbf{S}^{\mathrm{T}} - \mathbf{A}_{pq}^{\mathrm{T}}\mathbf{R} = 0. \quad (6)$$

Substituting $\mathbf{S} = \mathbf{S}^{\mathrm{T}}$ and $\mathbf{R}^{\mathrm{T}}\mathbf{A}_{pq} = \mathbf{A}_{pq}^{\mathrm{T}}\mathbf{R} = \mathbf{S}_{pq}$ given in Equation (1) into Equation (6) yields the *Lyapunov equation* of the form:

$$\mathbf{A}_{qq}^{\mathrm{T}}\mathbf{S} + \mathbf{S}\mathbf{A}_{qq} = 2\mathbf{S}_{pq}, \qquad (7)$$

of which solution [22] can be written as $\mathbf{S} = 2\int_0^\infty \big(e^{-t\mathbf{A}_{qq}^{\mathrm{T}}}\big)\mathbf{S}_{pq}\big(e^{-t\mathbf{A}_{qq}}\big)dt$. Here, the shape matching matrix $\mathbf{A}_{qq}$ is symmetric; thus it can be decomposed into $\mathbf{A}_{qq} = \mathbf{V}_{qq}\mathrm{diag}(\lambda_1^{qq}, \lambda_2^{qq}, \lambda_3^{qq})\mathbf{V}_{qq}^{\mathrm{T}}$. Then, its exponential can be simply obtained by $e^{-t\mathbf{A}_{qq}} = \mathbf{V}_{qq}\mathrm{diag}(e^{-t\lambda_1^{qq}}, e^{-t\lambda_2^{qq}}, e^{-t\lambda_3^{qq}})\mathbf{V}_{qq}^{\mathrm{T}}$. Finally, a closed-form solution of the optimal stretch is obtained as follows:

$$\mathbf{S} = \mathbf{V}_{qq}\big[\big(\mathbf{V}_{qq}^{\mathrm{T}}\mathbf{S}_{pq}\mathbf{V}_{qq}\big) \circ \bar{\mathbf{\Lambda}}^{qq}\big]\mathbf{V}_{qq}^{\mathrm{T}}, \qquad (8)$$

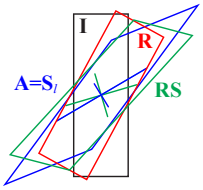
where $\bar{\mathbf{\Lambda}}_{ij}^{qq} = 2/(\lambda_i^{qq} + \lambda_j^{qq})$ and the binary operator $\circ$ represents the Hadamard product that produces the element-wise multiplication of its operand matrices.

Discussion. The derivation of the optimal stretch seems to be rather complicated, but its computation (8) is simple and efficient; Only a single matrix diagonalization of $\mathbf{A}_{qq}$ is required because $\mathbf{S}_{pq}$ is already available in Equation (1). We also note that the optimal stretch matrix has been formulated for the first time in this paper. In [21], only a single scalar parameter is considered for isotropic stretching, whereas the optimal stretch matrix introduced in this paper copes with anisotropic stretching and thus it can be successfully employed for plastic deformation and ductile fracture.

The linear transformation $\mathbf{A} = \mathbf{A}_{pq}\mathbf{A}_{qq}^{-1}$ minimizes $\sum m_i \|\mathbf{A}\mathbf{q}_i - \mathbf{p}_i\|^2$ [7]. Instead of the optimal stretch introduced in this paper, one may employ polar decomposition of the optimal linear transformation: $\mathbf{A} = \mathbf{R}_l\mathbf{S}_l$. However, this

is equivalent to seeking a pair of $\mathbf{R}$ and $\mathbf{S}$ in Equation (2) without fixing $\mathbf{R}$ with the optimal one given in Equation (1). Thus, $\mathbf{R}_l$ differs from the optimal rotation $\mathbf{R}$, which is indeed required in the shape matching deformation. This can also be verified using polar decomposition; $\mathbf{A} = \mathbf{A}_{pq}\mathbf{A}_{qq}^{-1} = (\mathbf{R}\mathbf{S}_{pq})\mathbf{A}_{qq}^{-1} = \mathbf{R}\mathbf{R}_d\mathbf{S}_l = \mathbf{R}_l\mathbf{S}_l$, where $\mathbf{S}_{pq}\mathbf{A}_{qq}^{-1} = \mathbf{R}_d\mathbf{S}_l$. Thus, $\mathbf{R}_l$ differs from $\mathbf{R}$ unless $\mathbf{R}_d = \mathbf{I}$. $\mathbf{R}_d = \mathbf{I}$ *iff.* the principal axes of $\mathbf{S}_{pq}$ and $\mathbf{A}_{qq}$ are the same (*i.e.*, stretching occurs along the principal axes of the rest shape).

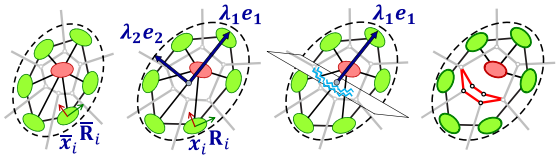
The inset illustrates an example of the optimal stretch. The blue quad is purely stretched with $\mathbf{S}_l$ from the black quad. The axes inside a quad depict stretching directions and magnitudes. Shape matching with a rigid transformation calls for the red quad transformed with the optimal rotation $\mathbf{R}$. Then, the optimal stretch $\mathbf{S}$ measures how much stretching is required to match the red to the blue. In contrast, one might employ the polar decomposition $\mathbf{A} = \mathbf{R}_l\mathbf{S}_l$. Unfortunately, this makes the blue quad be pulled toward the black (not the red) because $\mathbf{R}_l = \mathbf{I}$ in this example.



3.2 Shape Matching with OP

Shape matching with a single rigid transformation is restricted to modest deformation [7]. However, its extended version, shape matching with oriented particles allows an efficient and robust way of simulating visually convincing deformation in a position-based dynamics framework [8]; A surface/volume mesh is approximated with ellipsoidal particles, and shape matching is applied to each *shape matching group* consisting of a particle and its one-ring neighbor particles. Positions and orientations of the particles are used for linear blend skinning of the surface vertices. We adopt this approach for real-time simulation of large elastoplastic deformation and ductile fracture; thus, we briefly reformulate it in this section and then introduce additional *velocity correction* for angular momentum conservation in the next section.

Suppose that ellipsoidal particles of a shape matching group are transformed from the rest position $\bar{\mathbf{x}}_j$ and rotation $\bar{\mathbf{R}}_j$ to the current position $\mathbf{x}_j$ and rotation $\mathbf{R}_j$ (See Figure 1(a) and (b)) [1]. Then, the shape matching matrices $\mathbf{A}_{pq}$ and $\mathbf{A}_{qq}$ for the shape matching group $\mathcal{N}$ can be computed as follows:



(a) Rest (b) Deformed (c) Fractured (d) Remeshed

Figure 1: Deformation and fracture with OP.

$$\mathbf{A}_{pq} = \sum_{j\in\mathcal{N}} \left(\mathbf{R}_j\mathbf{A}_j\bar{\mathbf{R}}_j^{\mathrm{T}} + m_j\mathbf{x}_j\bar{\mathbf{x}}_j^{\mathrm{T}}\right) - M\mathbf{c}\bar{\mathbf{c}}^{\mathrm{T}},$$

$$\mathbf{A}_{qq} = \sum_{j\in\mathcal{N}} \left(\bar{\mathbf{R}}_j\mathbf{A}_j\bar{\mathbf{R}}_j^{\mathrm{T}} + m_i\bar{\mathbf{x}}_j\bar{\mathbf{x}}_j^{\mathrm{T}}\right) - M\bar{\mathbf{c}}\bar{\mathbf{c}}^{\mathrm{T}},$$

where $\mathbf{A}_j = \frac{1}{5}m_j\mathrm{diag}(a_j^2, b_j^2, c_j^2)$ is the moment matrix of an ellipsoid with mass m_j and principal radii a_j, b_j and c_j, M is the total mass of the shape matching group, and $\mathbf{c}$ and $\bar{\mathbf{c}}$ are the current and rest centers of mass, respectively.

Position-based dynamics for oriented particles evolves the states of the particles in three steps. (1) *Prediction*: The linear velocity $\mathbf{v}$ and the angular velocity $\mathbf{w}$ of a particle are updated with the external force and torque, and then its predicted position $\mathbf{x}_p$ and orientation $\mathbf{q}_p$ are computed using an explicit Euler integration scheme,

$$\mathbf{x}_p \leftarrow \mathbf{x} + h\mathbf{v} \quad \text{and} \quad \mathbf{q}_p \leftarrow \exp(h\mathbf{w}/2)\mathbf{q},$$

where h is the size of the simulation time step. (2) *Correction*: The predicted positions and orientations are modified to satisfy a set of constraints such as shape matching and collision. (3) *Velocity update*: The linear and angular velocities of a particle are updated backward from its position and orientation,

$$\mathbf{v} \leftarrow (\mathbf{x}_p - \mathbf{x})/h \quad \text{and} \quad \mathbf{w} \leftarrow 2\log(\mathbf{q}_p\mathbf{q}^{-1})/h.$$

For more details, refer to [8]. In the correction step, shape matching constraints are enforced by Gauss-Seidel iterations where the position and orientation of a particle participating in a shape matching group is modified without considering its participations in other groups.

[1] $\mathbf{R}_j = [\mathbf{r}_j^a|\mathbf{r}_j^b|\mathbf{r}_j^c]$ when $\mathbf{r}_j^a$, $\mathbf{r}_j^b$, $\mathbf{r}_j^c$ are unit axes of an ellipsoid.

3.3 Momentum Conservation

The linear momentum of the oriented particles is conserved perfectly before and after shape matching, regardless of the time step size. However, the angular momentum was not conserved perfectly in its basic form. We observed that, without angular momentum conservation, deformable bodies fallen down onto the floor were slowly but constantly rotating. This problem could be suppressed to some degree by setting the cutoff velocity that freezes the angular motions. However, it was not easy to determine how large cutoff value should be set. The amount of difference in angular momentum is proportional to the time step size. We prefer a large time step for real-time applications and so devise an additional mechanism for perfect conservation of angular momentum.

The angular momentum $\mathbf{L}$ of a shape matching group $\mathcal{N}$ can be computed by

$$\mathbf{L} = \sum_{j \in \mathcal{N}} \Big(\mathbf{H}_j \mathbf{w}_j + m_j \mathbf{r}_j \times \mathbf{v}_j \Big),$$

where $\mathbf{H}_j = \mathbf{R}_j \big[tr(\mathbf{A}_j)\mathbf{I} - \mathbf{A}_j \big] \mathbf{R}_j^{\mathrm{T}}$ is the inertia tensor of an ellipsoid with a moment matrix $\mathbf{A}_j$ and $\mathbf{r}_j = (\mathbf{x}_j - \mathbf{c})$ is the moment arm of the ellipsoid from the center of mass of $\mathcal{N}$. Let $\mathbf{L}^-$ and $\mathbf{L}^+$ be the angular momentums before and after shape matching, respectively. Then, we need to compensate for the difference of the angular momentum, $\Delta\mathbf{L} = (\mathbf{L}^- - \mathbf{L}^+)$. We instantaneously interpret the shape matching group $\mathcal{N}$ as a composite rigid body with a composite inertia tensor $\mathbf{H} = \sum_{j \in \mathcal{N}} (\mathbf{H}_j - m_j \mathbf{r}_j \times \mathbf{r}_j \times)$. Here, the parallel axis theorem is used to shift the moment of inertia of an ellipsoid. Then, $\Delta\mathbf{L}$ can be compensated for by an instantaneous change of the composite angular velocity, $\Delta\mathbf{w} = \mathbf{H}^{-1}\Delta\mathbf{L}$. This compensation is accomplished by *velocity correction* of the ellipsoids participating in the shape matching group $\mathcal{N}$:

$$\forall j \in \mathcal{N} \ \big(\Delta\mathbf{v}_j = \Delta\mathbf{w} \times \mathbf{r}_j \text{ and } \Delta\mathbf{w}_j = \Delta\mathbf{w} \big).$$

This velocity correction is computed for each shape matching group at the time of shape matching *but* applied after the velocity update step because velocities of the oriented particles are recomputed backward from their positions and orientations in position-based dynamics. Instantaneous interpretation of a shape matching group as rigid is similar to the *rigid impact zone* in [23].

3.4 Plastic Deformation

The stretch matrix given in Equation (8) is symmetric, and thus it can be further decomposed into $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}}$, where $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \lambda_3)$ with eigenvalues λ_i and $\mathbf{V} = [\mathbf{e}_1 | \mathbf{e}_2 | \mathbf{e}_3]$ with orthonormal eigenvectors $\mathbf{e}_i$. The stretch matrix has diverse relationships with the strain: $\epsilon_c = \mathbf{V}(\mathbf{\Lambda} - \mathbf{I})\mathbf{V}^{\mathrm{T}}, \epsilon_g = \frac{1}{2}\mathbf{V}(\mathbf{\Lambda}^2 - \mathbf{I})\mathbf{V}^{\mathrm{T}}, \epsilon_l = \mathbf{V}\log(\mathbf{\Lambda})\mathbf{V}^{\mathrm{T}}$, where ϵ_c, ϵ_g, and ϵ_l denote the Cauchy, Green, and logarithmic strain tensors, respectively. We chose the logarithmic strain for convenience of strain decomposition as in [3].

We represent plastic deformation with a multiplicative decomposition of deformation as in [3]. At each shape matching group, the total deformation is written as $\mathbf{F} = \mathbf{F}_{\mathcal{E}}\mathbf{F}_{\mathcal{P}}$, where $\mathbf{F}_{\mathcal{P}}$ is a matrix representing the plastic deformation stored at the shape matching group and initialized with the identity matrix. The elastic deformation is computed by $\mathbf{F}_{\mathcal{E}} = \mathbf{R}\mathbf{S} = \mathbf{R}(\mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}})$. Then, the plasticity model [10] equipped with creep and work hardening can be applied as follows:

$$\mathbf{F}_{\mathcal{P}} \leftarrow \mathbf{V}\big(\mathbf{\Lambda} \cdot |\mathbf{\Lambda}|^{-\frac{1}{3}}\big)^{\gamma}\mathbf{V}^{\mathrm{T}}\mathbf{F}_{\mathcal{P}},$$

where $\gamma = \min\left(\frac{\nu(\|\epsilon_l\| - \epsilon_Y)}{\|\epsilon_l\|}, 1\right)$ determines how much of the symmetric part of the elastic deformation is transferred to the plastic deformation in terms of the plastic flow rate ν whenever a plastic yield criterion exceeds the plastic yield threshold ϵ_Y. We adopt the plastic yield criterion on the logarithmic strain $\|\epsilon_l\| = \|\log(\mathbf{F}_{\mathcal{E}})\|_2 = \max_i(|\log(\lambda_i)|)$. The plastic yield ϵ_Y is increased by $\kappa\|\epsilon_l\|\Delta t$ after the plastic update, where κ controls the amount of work hardening and Δt is the time step.

Once a portion of the elastic deformation is absorbed by the plastic deformation, the remaining deformation becomes the elastic deformation:

$$\mathbf{F}_{\mathcal{E}} = \mathbf{R}\mathbf{S} \leftarrow \mathbf{R}\mathbf{V}\big(\mathbf{\Lambda}^{1-\gamma}|\mathbf{\Lambda}|^{\frac{\gamma}{3}}\big)\mathbf{V}^{\mathrm{T}}. \quad (9)$$

The stretch matrix $\mathbf{S}$ is used to determine material failure in Section 3.5. The plastic deformation $\mathbf{F}_{\mathcal{P}}$ is incorporated into the computation of

the shape matching matrices and the goal positions by replacing the definition $\mathbf{q}_i = (\bar{\mathbf{x}}_i - \bar{\mathbf{c}})$ with $\mathbf{q}_i = \mathbf{F}_{\mathcal{P}}(\bar{\mathbf{x}}_i - \bar{\mathbf{c}})$, as in [7]. The shape matching matrices for plastic deformation are obtained as follows:

$$\mathcal{A}_{pq} = \mathbf{A}_{pq}\mathbf{F}_{\mathcal{P}}^{\mathrm{T}} \quad \text{and} \quad \mathcal{A}_{qq} = \mathbf{F}_{\mathcal{P}}\mathbf{A}_{qq}\mathbf{F}_{\mathcal{P}}^{\mathrm{T}}.$$

The goal positions are computed by $\mathbf{g}_i = \mathbf{R}\mathbf{F}_{\mathcal{P}}(\bar{\mathbf{x}}_i - \bar{\mathbf{c}}) + \mathbf{c}$. Figure 4 shows examples of three rest shapes after plastic deformation.

3.5 Ductile Fracture

A material fails when applied with an excessive stretch or compression. We are interested in the material failure leading to ductile fracture. The key difference between brittle and ductile fracture can be attributed to the amount of plastic deformation that the material undergoes before fracture [5]. Ductile materials experience large amounts of plastic deformation while brittle materials experience little or no plastic deformation. We already paved a way to incorporating plastic deformation into the simulation framework. Thus, ductile fracture can be implemented just by employing a fracture criterion not on the total deformation but on the elastic deformation, given in Equation (9).

In the graphics literature, the maximum stress criterion has been widely adopted for fracture, which assumes that a material fails when the maximum principal stress is larger than the uniaxial tensile strength of the material [2, 4, 14]. To employ this criterion, a stress tensor must be obtained from a strain tensor using a constitutive relation. However, it is cumbersome to introduce additional material constants for a stress tensor. Thus, we employ another phenomenological failure criterion, the maximum principal strain criterion [11], which directly deals with principal strains.

Suppose that we are examining a shape matching group $\mathcal{N}$ consisting of an ellipsoid $\mathcal{E}$ and its one-ring neighbors. Fracture is initiated with a local fracture plane when the magnitude of the principal elastic strain of the shape matching group $\mathcal{N}$ exceeds a user-specified fracture toughness τ. Assume that we are employing the logarithmic strain tensor and $|\log(\lambda_i)| > \tau$. Then, we set the fracture plane to be perpendicular to the eigenvector $\mathbf{e}_i$ corresponding to the principal elastic stretch λ_i and pass through the center of mass of the shape matching group $\mathcal{N}$. If the plane intersects with an edge between the ellipsoid $\mathcal{E}$ and one of its neighbors, then we cut off the edge (See Figure 1(c)). A shape matching group involved with fracture calls for recomputation of its shape matching matrix $\mathbf{A}_{qq}$. In addition, fracture entails remeshing of the visual mesh (See Figure 1(d)). Remeshing will be described in Section 4.

4 Visualization of Fracture

Up to now, we have considered only a *simulation mesh* that consists of a set of oriented particles and a set of edges among the particles. This section addresses how to maintain a deformed, fractured *visual mesh* for rendering along with a simulation mesh. A visual mesh of a deformable body is paired with a simulation mesh of n particles in such a way that, when shattered completely, it is partitioned into n fragments, each of which solely depends on a unique particle (See Figures 1 and 2). We use a *fragment mesh* to represent the geometry of a fragment. Then, the visual mesh becomes, in its basic form, a composite mesh of the fragment meshes. Fragment meshes must be watertight at each interface until fractured. Thus, an *interface* surface between a pair of fragments corresponds to an edge in the simulation mesh, of which both ends are the particles associated with the fragments.

Visualization of fracture is formulated as maintaining the topology of a visual mesh in accordance with a simulation mesh, while deforming vertices of the visual mesh. We employ linear blend skinning for vertex deformation as in [8]. Each vertex depends on the nearest oriented particle and its one-ring neighbors. Regarding fracture, we propose two methods that maintain a watertight, visual mesh from given fragment meshes. One is designed for offline applications where the initial, compact visual mesh is progressively deformed and split at the interfaces, while fractured at runtime. The other is for real-time applications where GPU skinning makes a pair of polygonal fragment surfaces watertight until fractured. For fragment mesh generation of a deformable body, a set of pre-fracturing techniques can be employed, such

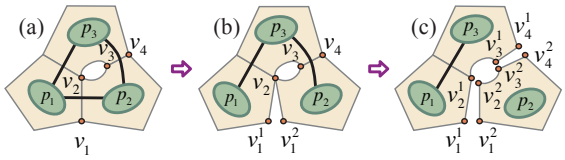


Figure 2: Fracture and fragments.

as Voronoi cell decomposition [1], approximate convex decomposition [24], and volumetric cell decomposition with fracture patterns [25].

4.1 Runtime Splitting of a Visual Mesh

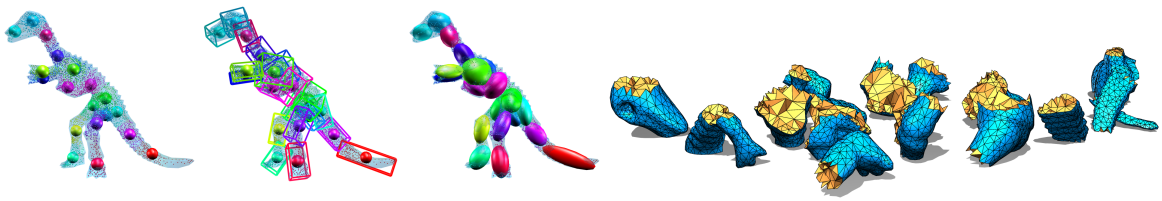
A polygonal surface representing a fragment interface is invisible and useless until fracture occurs between the fragments. We maintain a watertight, compact visual mesh where duplicate or invisible faces do not exist at the intact interfaces. At first, an initial watertight visual mesh is built up by composing all the faces of the fragment meshes except those at the fragment interfaces. Whenever an interface between a pair of fragments is fractured (*i.e.*, the edge between the corresponding particles is disconnected), all the faces of the interface become visible and split for each fragment. The vertices of the interface require special care to make the topology of the visual mesh synchronized with the simulation mesh, which is in contrast to rigid fracturing where all the vertices of the interface are split unconditionally [18, 25]. Each vertex is checked to see if it can be split or not (See Figure 2). A vertex should be split when the fragments sharing the vertex is not connected through them (*e.g.*, v_1^1 and v_1^2 in Figure 2(b)). Otherwise, it cannot be split (*e.g.*, p_1, p_2, p_3 sharing v_2 are connected; thus, it cannot be split in Figure 2(b)). As edges are disconnected, the connected graph becomes separated into many connected components. A connected component with a single particle corresponds to a fragment (See Figure 2(c)).

4.2 GPU Skinning with Fragment Meshes

It is no surprise that, for large visual meshes, updating and transferring the vertex positions and the connectivities into the GPU side results in a performance bottleneck rather than simulating ductile fracture in the CPU side. For interactive applications, we propose a GPU skinning method that supports runtime fracture as well as deformation. At the beginning, all the fragment meshes of the particles are transferred into the GPU side. At runtime, each pair of polygonal surfaces representing a fragment interface is kept watertight until the corresponding edge is disconnected in the simulation mesh. Optionally, a geometry shader can be employed to skip rasterization of an invisible interface surface not fractured yet.

Pre-fracturing makes a vertex of an interface instanced at multiple fragments. To render a smooth surface as well as a sharp interface, each instance of the vertex is assigned with a normal for each adjacent face. Then, multiple instances of the vertex need to have the same position with consistent normals until the interface is fractured (*e.g.*, the vertex v_1^1 of the particle p_1 and v_1^2 of p_2 are instanced from the same vertex v_1; thus they have the same position until the edge $\overline{p_1p_2}$ is disconnected in Figure 2). We employ linear blend skinning that computes the position and normal of a vertex using the nearest particle and its one-ring neighbors. Thus, the nearest particles at the multiple instances should be the same for watertight skinning. As fracture occurs, fragments connected locally at a vertex become disconnected. Accordingly, the nearest particle at an instance of the vertex should be updated as the nearest one among the particles still connected locally to the instance (*e.g.*, the nearest particle changes from p_3 to p_2 at v_2^2 after the edge $\overline{p_2p_3}$ is disconnected, whereas the nearest particle is still p_3 at v_2^1 in Figure 2).

When an edge is disconnected in the simulation mesh, the nearest particle needs to be updated only at the vertices of the corresponding interface. In addition, particles undergoing fracture at a frame exhibit spatial coherence. Based on these observations, we apply spatial clustering to the fragments so as to efficiently transfer the nearest particle information into the GPU side as per-vertex data on a per-cluster basis. Accordingly, pre-fracturing is modified to occur on a per-cluster basis. A watertight, compact visual mesh composited from a cluster of fragments is replaced with the individual fragment meshes, when the cluster of fragments is actually involved with fracture. This excludes rasterization of invisible faces and linear blend skinning for invisible vertices. The vertex shader



(a) Clusters (b) OBB (c) OP (d) Fracturing a dinosaur model with 20 particles

Figure 3: Simulation mesh obtained with k-means clustering.

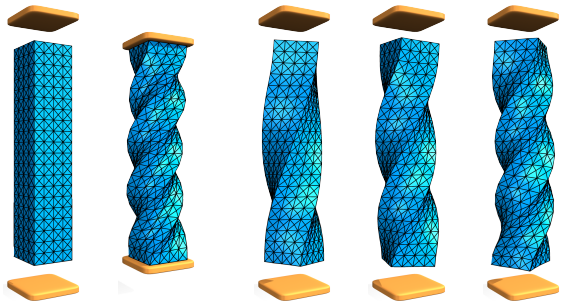
needs the simulation mesh for linear blend skinning. Because the size of the simulation mesh is larger than the maximum size of uniform parameters, it is fetched into the GPU side at each frame as a single texture.

5 Experimental Results

The proposed technique was implemented as an Autodesk MAYA plug-in for contents creation and offline rendering. It was also implemented as a Unity3D plug-in with Cg shaders for interactive application and real-time rendering of large scenes. All experiments were performed with an Intel I7-3770K 3.5GHz CPU, 8GB memory, and an NVIDIA GeForce GTX680.

The fragments of a deformable body come up with a simulation mesh. Thus, we need not only the geometry of a deformable body but also its fragment meshes. One of the most intuitive ways of obtaining fragment meshes is to employ Voronoi-like fracture patterns in a tetrahedral mesh of a given deformable body. For convenience of implementation, we apply k-means clustering to the centers of the tetrahedra while taking into account the connectivities in the mesh. Then, each cluster of tetrahedra becomes a fragment corresponding to an oriented particle. The covariance of the tetrahedra gives the initial orientation of the particle and the oriented bounding box gives the radii of the particle as in [8]. This process is illustrated in Figure 3. Fragment meshes can also be obtained in many different ways [1, 25].

Figure 4 compares plastic deformation of three vertical elastic bars with different plasticity. The top end of each bar is twisted for one full rotation while its bottom end is fixed as illustrated in Figure 4(a) and (b). The bars shown in Figure 4(c), (d), and (e) with low, medium, and high plasticity undergo large amounts of



(a) Rest (b) Twist (c) Low (d) Mid (e) High

Figure 4: Twisted bars with different plasticity.

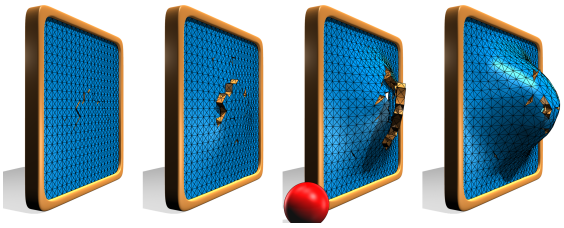


Figure 5: Four plates with increasing plasticity.

plastic flow during the twisting, and thus do not return to their original shapes after being released. The simulation mesh of each bar consists of 320 particles and 1,888 edges.

The animation of fracturing ductile plates in Figure 5 demonstrate a range of visual effects with different plasticity parameters. The leftmost inset shows the behavior of a nearly brittle material. From left to right, the materials become more ductile. Each scene consists of 300 particles including a single particle for the projectile. Figure 6 shows two bunnies struck by a heavy weight. They undergo large deformation and ductile fracture due to collisions with the weight and the floor and self-collisions. The simulation mesh of a bunny model consists of 300 particles and 1,288 edges. The left inset shows the behavior of a purely brittle material, and the right shows the effects of plasticity.

For real-time simulation and rendering of

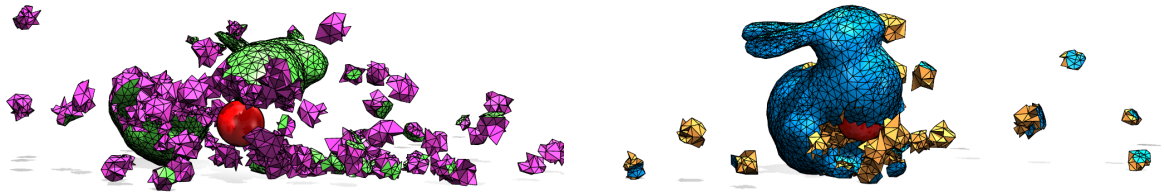


Figure 6: Fracturing two bunnies with a heavy projectile. The left one is brittle and the right is ductile.

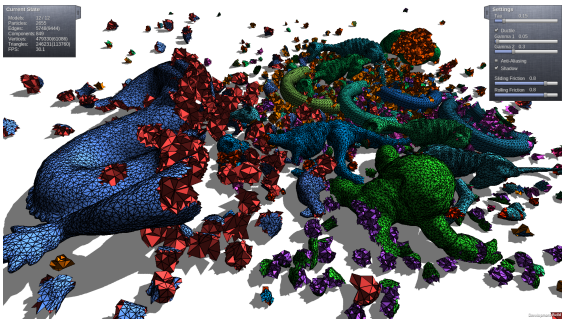


Figure 7: Real-time simulation of a large scene.

large scenes, we exploit not only a multi-core CPU but also a many-core GPU. Figure 7 shows a screenshot of a simple FPS-like game developed with the Unity 3D game engine. The simulation is performed with a simple parallelization using OpenMP and GPU skinning described in Section 4.2. Currently, collision detection and response are handled in a single core. The scene consists of 12 deformable models with 2.7K particles and 9.4K edges for the simulation meshes. As the simulation progresses, the deformable models are falling down to the floor. At the end of the simulation, the visual meshes become 849 components with 479K vertices and 246K triangles. The simulation runs about 30 FPS with 2 sub-steps for collision detection and response. On the average, shape matching, collision handling, and fracturing take 54, 45, and 1 percent of the computation time, respectively.

6 Conclusion

We have presented a practical approach to real-time simulation of ductile fracture. The oriented particles approach [8] is employed for robust and efficient simulation of large deformation, and the shape matching procedure is extended to find not only the optimal rotation but also the optimal stretch. This optimal stretch is used to formulate the plastic flow and fracture criteria for material failure at excessively stretched or compressed material points. We have demonstrated that the proposed method runs in real time even for large visual meshes and it can simulate large deformation and ductile fracture robustly.

Our method is based on shape matching with oriented particles; the physical behavior is dependent on the simulation mesh so that it is not easy to support adaptive refinement of the simulation mesh. Thus, a pre-fractured fragment corresponding to a single particle cannot be fractured into smaller ones. In the same vein, the pre-fractured fragments cannot be aligned with a point of impact given at runtime. These shortcomings could be alleviated to some degree if dynamic fracture with a fracture pattern [25] is employed to produce detailed sub-fragments at runtime. In addition, one can create fine debris and dust distributed over a newly-exposed fragment interface as in [16, 25]. Currently, we detect collisions using the ellipsoidal particles themselves, as in [8], not their corresponding fragment meshes. We plan to incorporate fragment meshes for collision handling.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (NRF-2011-0012878, NRF-2014R1A1A1008486) and the Research Grant of Kwangwoon University in 2013.

References

[1] M. Baker, N. Bin Zafar, M. Carlson, E. Coumans, B. Criswell, T. Harada, and P. Knight. *Destruction and Dynamics for Film and Game Production*. ACM SIGGRAPH 2011 Course Notes, 2011.

[2] Z. Bao, J.-M. Hong, J. Teran, and R. Fedkiw. Fracturing rigid materials. *IEEE TVCG*, 13(2):370–378, 2007.

[3] G. Irving, J. Teran, and R. Fedkiw. Invertible finite elements for robust simulation of large deformation. In *Proc. SCA*, pages 131–140, 2004.

[4] N. Molino, Z. Bao, and R. Fedkiw. A virtual node algorithm for changing mesh topology during simulation. *ACM TOG*, 23(3):385–392, 2004.

[5] J. F. O'Brien, A. W. Bargteil, and J. K. Hodgins. Graphical modeling and animation of ductile fracture. *ACM TOG*, 21(3):291–294, 2002.

[6] J. F. O'Brien and J. K. Hodgins. Graphical modeling and animation of brittle fracture. *Computer Graphics*, 33(4):137–146, 1999.

[7] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. *ACM TOG*, 24(3):471–478, 2005.

[8] M. Müller and N. Chentanez. Solid simulation with oriented particles. *ACM TOG*, 30(4):92:1–92:9, 2011.

[9] A. R. Rivers and D. L. James. FastLSM: Fast lattice shape matching for robust real-time deformation. *ACM TOG*, 26(3):82:1–82:6, 2007.

[10] A. W. Bargteil, C. Wojtan, J. K. Hodgins, and G. Turk. A finite element method for animating large viscoplastic flow. *ACM TOG*, 26(3):16:1–16:8, 2007.

[11] D. Gross and T. Seelig. *Fracture Mechanics: With an Introduction to Micromechanics*. Mechanical Engineering Series. Springer, second edition, 2011.

[12] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *Computer Graphics*, 22(4):269–278, 1988.

[13] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson. Physically based deformable models in computer graphics. *CGF*, 25(4):809–836, 2006.

[14] M. Müller and M. Gross. Interactive virtual materials. In *Proc. Graphics Interface*, pages 239–246, 2004.

[15] M. Pauly, R. Keyser, B. Adams, P. Dutré, M. Gross, and L. Guibas. Meshless animation of fracturing solids. *ACM TOG*, 24(3):957–964, 2005.

[16] E. G. Parker and J. F. O'Brien. Real-time deformation and fracture in a game environment. In *Proc. SCA*, pages 165–175, 2009.

[17] J. Su, C. Shroeder, and R. Fedkiw. Energy stability and fracture for frame rate rigid body simulations. In *Proc. SCA*, pages 1–10, 2009.

[18] C. Zheng and D. L. James. Rigid-body fracture sound with precomputed soundbanks. *ACM TOG*, 29(4):69:1–69:13, 2010.

[19] D. Steinemann, M. A. Otaduy, and M. Gross. Fast adaptive shape matching deformations. In *Proc. SCA*, pages 87–94, 2008.

[20] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. Position based dynamics. In *Proc. VRIPHYS*, pages 71–80, 2006.

[21] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. PAMI*, 13(4):376–380, 1991.

[22] P. J. Antsaklis and A. N. Michel. *A Linear Systems Primer*. Birkhäuser, 2007.

[23] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM TOG*, 21(3):594–603, 2002.

[24] J.-M. Lien and N. M. Amato. Approximate convex decomposition of polyhedra and its applications. *CAGD*, 25(7):503–522, 2008.

[25] M. Müller, N. Chentanez, and T.-Y. Kim. Real Time Dynamic Fracture with Volumetric Approximate Convex Decompositions. *ACM TOG*, 32(4):115:1–115:10, 2013.