Finite Element Method Deformables

EGE CIKLABAKKAL, University of Waterloo, Canada

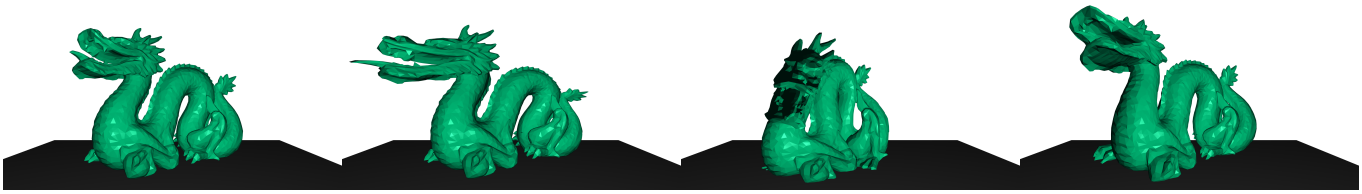

Fig. 1. Dragon is stretched from both ends and before being let go, resulting in elastic deformations.

The goal of physics-based simulation in computer graphics is to generate realistic animations by describing the physics of a system. Elastic objects have naturally complex behavior as they deform volumetrically. Finite element method works by discretizing the volume into smaller elements and solving differential equations on each element. Therefore, it is an logical choice to simulating elastic objects: partitioning the volume into tetrahedral elements and solving the equations of motion to figure out how each point inside the volume deforms in time. We show how to construct a standard 3D Finite element simulations for elastic deformables. Moreover, we compare three different potential energy functions, evaluating their stability and volume preservation properties. Further, we demonstrate basic collision handling with penalty forces and test it in an environment with gravity.

CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: finite element method, deformables, penalty forces

1 INTRODUCTION

The Finite Element Method (FEM) is a popular method to simulate elastic bodies in video games, movies, and visual effects in general [10]. Compared to a simpler simulation with springs between every two vertices of the object, FEM simulates the motion of the whole volume of an object. The volume based simulation relies on *the continuum hypothesis*, which states that the particles that compose the volume are distributed continuously [7]. Regardless of how much we zoom into the volume, the particle distribution will be homogeneous. This assumption does not hold at an atomic scale, however it is reasonable to make at larger scales, where our simulation takes place.

In this project, we go over the components to building an FEM solution to physically-based elastic object simulation. We define the basis functions of our finite elements and take the variational approach to deriving the motion of the object, which requires defining the equations for the kinetic and the potential energies of the object. We show that the motion is described by partial differential equations, which are solved on each finite element by integrating them in time. We also make use of springs to define simple interactive and penalty forces.

The main resource for this project is the course notes and assignments by Levin [8] and we largely follow their notation as well.

2 BACKGROUND

2.1 The Finite Element Method

Finite Element Method is a numerical method for solving partial differential equations. The main idea of FEM is to discretize the space into smaller *finite elements*, describe the equations to solve the PDE for each finite element and then assemble these equations into a large system of equations which approximate the solution for the whole system. It does so, by representing quantities inside a volume as a sum of quantities stored at the nodes of the finite element, weighted by spatially varying *basis functions*. In the case of 3D elastic body simulations, we use tetrahedral elements to represent positions inside a tetrahedron as a linear combination of positions stored at the vertices of the tetrahedron, weighted by basis functions associated with each vertex:

$$\mathbf{X} = \sum_{i=0}^{3} \mathbf{X}_i \phi_i(\mathbf{X}) \tag{1}$$

where $\mathbf{X} \in \mathbb{R}^3$ is an arbitrary position inside the tetrahedron with vertex positions $\mathbf{X}_i \in \mathbb{R}^3$ and basis functions ϕ_i.

Next, we define the basis functions. Let $\mathbf{X} = \begin{pmatrix} X & Y & Z \end{pmatrix}^T$ and $\mathbf{X}_i = \begin{pmatrix} X_i & Y_i & Z_i \end{pmatrix}^T$. Then, we write out Equation (1) as a matrix vector product and obtain the following linear system:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X_0 & X_1 & X_2 & X_3 \\ Y_0 & Y_1 & Y_2 & Y_3 \\ Z_0 & Z_1 & Z_2 & Z_3 \end{pmatrix} \begin{pmatrix} \phi_0(\mathbf{X}) \\ \phi_1(\mathbf{X}) \\ \phi_2(\mathbf{X}) \\ \phi_3(\mathbf{X}) \end{pmatrix}. \tag{2}$$

Equation (2) is underdetermined, therefore we add an additional constraint: $\phi_0(\mathbf{X}) = 1 - \phi_2(\mathbf{X}) - \phi_2(\mathbf{X}) - \phi_3(\mathbf{X})$ to arrive at the system:

$$\begin{pmatrix} X - X_0 \\ Y - Y_0 \\ Z - Z_0 \end{pmatrix} = \underbrace{\begin{pmatrix} \Delta X_1 & \Delta X_2 & \Delta X_3 \\ \Delta Y_1 & \Delta Y_2 & \Delta Y_3 \\ \Delta Z_1 & \Delta Z_2 & \Delta Z_3 \end{pmatrix}}_{\mathrm{T}} \begin{pmatrix} \phi_1(\mathbf{X}) \\ \phi_2(\mathbf{X}) \\ \phi_3(\mathbf{X}) \end{pmatrix} \tag{3}$$
$$\phi_0(\mathbf{X}) = 1 - \phi_2(\mathbf{X}) - \phi_2(\mathbf{X}) - \phi_3(\mathbf{X})$$

where $\Delta X_i = X_i - X_0$ and the vectors $\begin{pmatrix} \Delta X_i & \Delta Y_i & \Delta Z_i \end{pmatrix}^T$ represent the edge vectors of the tetrahedron. Inverting the matrix T in Equation (3), we finally arrive at the system of equations to describe

the basis functions:

$$\begin{aligned}\begin{pmatrix}\phi_1(\mathbf{X})\\ \phi_2(\mathbf{X})\\ \phi_3(\mathbf{X})\end{pmatrix} &= \mathrm{T}^{-1}(\mathbf{X}-\mathbf{X}_0)\\ \phi_0(\mathbf{X}) &= 1-\phi_2(\mathbf{X})-\phi_2(\mathbf{X})-\phi_3(\mathbf{X}).\end{aligned} \tag{4}$$

These basis functions are called linear basis functions as they are linear in $\mathbf{X}$. They are also known as the barycentric coordinates.

To create interesting animations, we need to define an expression for time varying positions of our elements. Using the basis functions defined above, we can express any position inside a deformed tetrahedron at some time t by:

$$\mathbf{x}^t(\mathbf{X}) = \sum_{i=0}^{3} \mathbf{x}_i^t(\mathbf{X})\phi_i(\mathbf{X}) \tag{5}$$

where $\mathbf{x}_i^t(\mathbf{X})$ is the position of the ith vertex at time t. Essentially, we have a mapping function $\mathbf{x}^t(\mathbf{X})$, which takes the positions of the undeformed tetrahedron, to their time varying deformed positions. The undeformed tetrahedra are embedded in the reference space and the deformed tetrahedra are in the world space. By keeping track of the deformed vertex positions of a tetrahedron, we can figure out the deformed position of any arbitrary point inside the tetrahedron thanks to the basis functions.

It is more convenient to expand Equation (5) as a matrix vector product:

$$\mathbf{x}^t(\mathbf{X}) = \underbrace{\begin{pmatrix}\phi_0\mathrm{I} & \phi_1\mathrm{I} & \phi_2\mathrm{I} & \phi_3\mathrm{I}\end{pmatrix}}_{\mathrm{N}(\mathbf{X})} \underbrace{\begin{pmatrix}\mathbf{x}_0^t\\ \mathbf{x}_1^t\\ \mathbf{x}_2^t\\ \mathbf{x}_3^t\end{pmatrix}}_{\mathbf{q}_j^t} \tag{6}$$

where I is a 3×3 identity matrix and $\mathbf{q}_j^t$ is a 12×1 stacked vector of vertex positions of the jth tetrahedron. $\mathbf{q}_j^t$ is called the generalized coordinates of the tetrahedron.

2.2 Variational principle

A variational principle is one that seeks to find unknown functions that makes an integral take on an extremum for a given problem [4]. The variational principle to describe mechanics is called the principle of least action, which yields the *equations of motion* when applied to a mechanical system. The principle of least action states that the motion of an object in time is the one that minimizes the difference between the object's kinetic energy and the potential energy:

$$\mathbf{q}(t) = \arg\min_{\mathbf{q}} \int_{t_0}^{t_1} T(\mathbf{q},\dot{\mathbf{q}}) - V(\mathbf{q},\dot{\mathbf{q}})\, dt \tag{7}$$

where T and V are the kinetic and potential energies and $\dot{\mathbf{q}}$ is the generalized velocity of the system, that is the derivative of $\mathbf{q}(t)$ with respect to time.

The equations of motion of a system, which tell us how to compute $\mathbf{q}(t)$ are found by satisfying the principle of least action. By calculus of variations, we can solve the following differential equations known as the Euler-Lagrange equations:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right) = -\frac{\partial L}{\partial \mathbf{q}} \tag{8}$$

where $L = T - V$ is called the Lagrangian. The solution to Equation (8) will provide the valid, time varying motion of $\mathbf{q}(t)$. Therefore, to utilize the variational approach, we first need to define the kinetic and the potential energies of a system, and then solve the Euler-Lagrange equations to update our generalized coordinates in time.

2.3 Kinetic energy of an elastic body

First, we derive the kinetic energy of a single tetrahedron. It is given by integrating the kinetic energy of a single point inside the tetrahedron, over the tetrahedron volume Ω. The kinetic energy of a single point is given by $\frac{1}{2}\rho||\mathbf{v}(\mathbf{X})||^2 d\Omega$, where ρ is the density of the material of the tetrahedron and $\mathbf{v}(\mathbf{X})$ is the velocity of the point. Then, considering that $\mathbf{v}(\mathbf{X}) = \dot{\mathbf{x}}(\mathbf{X}) = \mathrm{N}(\mathbf{X})\dot{\mathbf{q}}$ and integrating over Ω, we have:

$$\int_\Omega \frac{1}{2}\rho\left(\dot{\mathbf{q}}^T\mathrm{N}(\mathbf{X})^T\mathrm{N}(\mathbf{X})\dot{\mathbf{q}}\right) d\Omega. \tag{9}$$

Since $\dot{\mathbf{q}}$ is constant over the tetrahedron, we can rearrange the terms to arrive at

$$\frac{1}{2}\dot{\mathbf{q}}^T \underbrace{\left(\int_\Omega \rho\mathrm{N}(\mathbf{X})^T\mathrm{N}(\mathbf{X}) d\Omega\right)}_{\mathrm{M}_j} \dot{\mathbf{q}}. \tag{10}$$

M_j is the 12×12 mass matrix of the jth tetrahedron. To compute this mass matrix, we integrate its terms coming from $\mathrm{N}(\mathbf{X})^T\mathrm{N}(\mathbf{X})$ separately, that is:

$$\rho\int_\Omega \phi_r(\mathbf{X})\phi_s(\mathbf{X}) d\Omega\, \mathrm{I} \quad r = 0\ldots 3,\ s = 0\ldots 3. \tag{11}$$

Since they are tedious to evaluate by hand, a symbolic computation tool is useful.

In summary, the kinetic energy of a single tetrahedron is given by

$$T_j = \frac{1}{2}\dot{\mathrm{q}}_j^T\mathrm{M}_j\dot{\mathrm{q}}_j. \tag{12}$$

The kinetic energy of the whole system is given by summing over the kinetic energies of all tetrahedra. Note that, when computing the motion of the object as a whole, we will be computing the generalized coordinates of the whole system that is the vector $\mathbf{q}$. $\mathbf{q}$ consists of the positions of all the vertices, stacked as a vector. Hence, we need to obtain $\mathbf{q}_j$, given $\mathbf{q}$. To do so, we use the *element selection matrices*.

An element selection matrix E_j simply maps $\mathbf{q}$ to $\mathbf{q}_j$ by $\mathbf{q}_j = \mathrm{E}_j\mathbf{q}$. Similarly, it maps the generalized velocities as $\dot{\mathbf{q}}_j = \mathrm{E}_j\dot{\mathbf{q}}$. To define E_j, first let $\mathrm{S}_k = \begin{pmatrix}0 & 0 & \cdots & \mathrm{I}_{3\times 3} & \cdots & 0\end{pmatrix}$. Then, the product of S_k with $\mathbf{q}$ selects the vertex $\mathbf{x}_k$. E_j is simply the appropriate S_k matrices stacked.

Now, we can write the total kinetic energy of the system with M tetrahedra as a function of $\mathbf{q}$:

$$T = \sum_{j=0}^{M-1} \frac{1}{2}\dot{\mathbf{q}}^T\mathrm{E}_j^T\mathrm{M}_j\mathrm{E}_j\dot{\mathbf{q}}. \tag{13}$$

We can rearrange the terms by taking $\dot{\mathbf{q}}$ terms out of the sum to obtain

$$T = \frac{1}{2}\dot{\mathbf{q}}^T \underbrace{\left(\sum_{j=0}^{M-1} \mathrm{E}_j^T \mathrm{M}_j \mathrm{E}_j\right)}_{\mathrm{M}} \dot{\mathbf{q}} \tag{14}$$

where M is the mass matrix of the whole system.

2.4 Potential energy of an elastic body

To compute the potential energy in a deforming object, we first need to measure its deformation. In particular we need to compute the *strain*, which measures the relative displacement of the deformed and the undeformed states, and then use a *strain energy density function* to describe how deformation converts to potential energy.

First, we observe how two arbitrary points inside a tetrahedron deform. Let $\mathbf{X}_0$ and $\mathbf{X}_1$ be two points in the reference (undeformed) space and the world (deformed) space positions corresponding to $\mathbf{X}_0$ and $\mathbf{X}_1$ are given by $\mathbf{x}(\mathbf{X}_0)$ and $\mathbf{x}(\mathbf{X}_1)$ respectively. Let $\Delta\mathbf{X} = \mathbf{X}_1 - \mathbf{X}_0$, then

$$\Delta\mathbf{x} = \mathbf{x}(\mathbf{X}_0 + \Delta\mathbf{X}) - \mathbf{x}(\mathbf{X}_0). \tag{15}$$

Assuming that we can pick $\mathbf{X}_0$ and $\mathbf{X}_1$ arbitrarily close and thus Taylor expand the $\mathbf{x}(\mathbf{X}_0 + \Delta\mathbf{X})$ term, we get

$$\Delta\mathbf{x} \approx \mathbf{x}(\mathbf{X}_0) + \frac{\partial \mathbf{x}}{\partial \mathbf{X}}\Delta\mathbf{X} - \mathbf{x}(\mathbf{X}_0) = \frac{\partial \mathbf{x}}{\partial \mathbf{X}}\Delta\mathbf{X}. \tag{16}$$

The term $\frac{\partial \mathbf{x}}{\partial \mathbf{X}}$ is called the *deformation gradient* and is usually represented by F. F is a 3×3 matrix tells us vectors locally deform from the reference to the world space. Due to the linear basis functions, F is constant over a tetrahedron.

A simple strain that is invariant to rigid motion is the Green-Lagrange strain, given by $F^T F - \mathrm{I}$. Being invariant to rigid motion is important as we want the deformation to be independent from the objects translation or rotation from the initial undeformed state.

The strain energy density function $\psi(F(\mathbf{X}))$ is completely determined by the deformation gradient and computes the potential energy due to a deformation per unit undeformed volume. So, by integrating over the tetrahedron, we obtain the potential energy P_j of a single tetrahedron:

$$P_j = \int_\Omega \psi(F_j(\mathbf{X}))d\Omega. \tag{17}$$

In this project, we compare three *Neo-Hookean* energy variants. Firstly, the energy due to Bower [3] is given by:

$$\psi_{\mathrm{B}}(F) = \frac{\mu}{2}(J^{-2/3}I_C - 3) + \frac{\lambda}{2}(J-1)^2 \tag{18}$$

where μ and λ are called the Lamé parameters which relate material properties with strain, $J = \det(F)$, and $I_C = \mathrm{tr}(F^T F)$.

Secondly, the energy due to Bonet and Wood [2] is given by:

$$\psi_{\mathrm{BW}}(F) = \frac{\mu}{2}(I_C - 3) - \mu \log J + \frac{\lambda}{2}(\log J)^2. \tag{19}$$

Lastly, the energy described as the stable Neo-Hookean by Smith et al. [11]:

$$\psi_{\mathrm{S}}(F) = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J-\alpha)^2 - \frac{\mu}{2}(I_C + 1) \tag{20}$$

where $\alpha = 1 + \frac{\mu}{\lambda} - \frac{\mu}{4\lambda}$.

Note that since F is constant over a tetrahedron (property of linear basis functions), so is $\psi(F)$. Therefore, Equation (17) simplifies to

$$V_j = \mathrm{vol}_j \cdot \psi(F_j(\mathbf{X}_i)) \tag{21}$$

where vol_j is the volume of the undeformed tetrahedron and $\mathbf{X}_i$ is any point inside the tetrahedron. We can consider $F_j(\mathbf{q}_j)$ to denote the deformation gradient of the jth tetrahedron in terms of its generalized coordinates $\mathbf{q}_j$ and write the total potential energy of the system as a sum of per tetrahedron potential energies:

$$V = \sum_{j=0}^{M-1} \mathrm{vol}_j \cdot \psi(F_j(\mathbf{q}_j)). \tag{22}$$

Similarly to the kinetic energy case, we can express $\mathbf{q}_j$ in terms of the full generalized coordinates as $\mathbf{q}_j = \mathrm{E}_j\mathbf{q}$. Then, the formula for the total potential energy becomes:

$$V = \sum_{j=0}^{M-1} \mathrm{vol}_j \cdot \psi(F_j(\mathrm{E}_j\mathbf{q})). \tag{23}$$

2.5 The equations of motion

With the kinetic and the potential energies at hand, we can build the Lagrangian $L = T + V$, and substitute it into Equation (8):

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right) &= -\frac{\partial L}{\partial \mathbf{q}} \\ \frac{d}{dt}\left(\mathrm{M}\dot{\mathbf{q}}\right) &= -\frac{\partial V}{\partial \mathbf{q}} \\ \mathrm{M}\ddot{\mathbf{q}} &= -\frac{\partial V}{\partial \mathbf{q}}. \end{aligned} \tag{24}$$

The term $\mathrm{f}(\mathbf{q}) = -\frac{\partial V}{\partial \mathbf{q}}$ is called the generalized forces and the equations $\mathrm{M}\ddot{\mathbf{q}} = -\frac{\partial V}{\partial \mathbf{q}}$ are called the equations of motion. They are the differential equations we need to solve to figure out the positions of our system in time. Another quantity which will be useful later on is the stiffness matrix $\mathrm{K} = -\frac{\partial^2 V}{\partial \mathbf{q}^2}$.

2.6 Backward Euler time integration

Backward Euler method is a numerical method to solve ordinary differential equations which has better stability and allows larger time steps [1] compared to some of the other time integration methods such as the Forward Euler method. In the case of the equations of motion, Backward Euler evaluates the forces at time $t+1$ instead of the current time t like the Forward Euler does.

The equations of motion are second order differential equations. We can however, write a pair of coupled, first order differential equations that solve the second order system. Using the Backward Euler scheme, we have the following update equations:

$$\begin{aligned} \mathrm{M}\dot{\mathbf{q}}^{t+1} &= \mathrm{M}\dot{\mathbf{q}}^t + \Delta t \mathrm{f}(\mathbf{q}^{t+1}) \\ \mathbf{q}^{t+1} &= \mathbf{q}^t + \Delta t \dot{\mathbf{q}}^{t+1} \end{aligned} \tag{25}$$

2.6.1 Linearly-Implicit Backward Euler. To solve Equation (25), we can substitute the second equation into the first one:

$$\mathrm{M}\dot{\mathbf{q}}^{t+1} = \mathrm{M}\dot{\mathbf{q}}^t + \Delta t \mathrm{f}(\mathbf{q}^t + \Delta t \dot{\mathbf{q}}^{t+1}). \tag{26}$$

However, f is not a linear function in $\mathbf{q}$. The idea of the linearly-implicit Backward Euler time integration is to linearize Equation (26) by assuming that Δt is sufficiently small and using a first order Taylor expansion:

$$\mathrm{M}\dot{\mathbf{q}}^{t+1} = \mathrm{M}\dot{\mathbf{q}}^{t} + \Delta t \mathrm{f}(\mathbf{q}^{t}) + \Delta t^2 \frac{\partial \mathrm{f}}{\partial \mathbf{q}} \dot{\mathbf{q}}^{t+1}. \tag{27}$$

Note that $\frac{\partial \mathrm{f}}{\partial \mathbf{q}} = -\frac{\partial^2 V}{\partial \mathbf{q}^2} = \mathrm{K}$ is the stiffness matrix.

Collecting the $\dot{\mathbf{q}}^{t+1}$ terms on the left hand side:

$$\begin{aligned} (\mathrm{M} - \Delta t^2 \mathrm{K})\dot{\mathbf{q}}^{t+1} &= \mathrm{M}\dot{\mathbf{q}}^{t} + \Delta t \mathrm{f}(\mathbf{q}^{t}) \\ \mathbf{q}^{t+1} &= \mathbf{q}^{t} + \Delta t \dot{\mathbf{q}}^{t+1}. \end{aligned} \tag{28}$$

We can solve for the velocity $\dot{\mathbf{q}}^{t+1}$ in the first equation and then update the position $\mathbf{q}^{t+1}$ accordingly.

2.6.2 Fully-Implicit Backward Euler. Linearly-implicit Backward Euler method is a linear approximation. Although it is efficient and works well in general, for stiff systems, it requires very small time steps to remain stable, which goes against the main advantage of the Backward Euler approach. The idea of the fully-implicit Backward Euler is to keep the implicit update as is, however instead of solving the nonlinear system we get, convert the problem into an optimization one.

Similarly to the linearly-implicit case, we start by substituting the position update equation in Equation (25) into the velocity update equation, to obtain Equation (26). Then, we rearrange the terms so that the problem can be viewed as a rootfinding problem:

$$\mathrm{M}\dot{\mathbf{q}}^{t+1} - \mathrm{M}\dot{\mathbf{q}}^{t} + \Delta t \mathrm{f}(\mathbf{q}^{t} + \Delta t \dot{\mathbf{q}}^{t+1}) = 0. \tag{29}$$

This system is nonlinear and that is difficult to solve directly. Instead, consider a minimization problem of some energy function $E(\mathbf{v})$:

$$\mathbf{v}^{*} = \arg\min_{\mathbf{v}} E(\mathbf{v}) \tag{30}$$

where $\mathbf{v}^{*}$ is the optimal value we are solving for. Now, consider the first order optimality condition:

$$\left.\frac{\partial E}{\partial \mathbf{v}}\right|_{\mathbf{v}^{*}} = 0, \tag{31}$$

which is essentially a root finding problem.

Using this idea, we can consider Equation (29) as a gradient of some energy function $E(\dot{\mathbf{q}})$ with respect to $\dot{\mathbf{q}}$ evaluated at $\dot{\mathbf{q}}^{t+1}$. Let $\mathbf{v} = \dot{\mathbf{q}}$, we integrate Equation (29) to obtain $E(\mathbf{v})$:

$$E(\mathbf{v}) = \frac{1}{2}\left(\mathbf{v} - \dot{\mathbf{q}}^{t}\right)^{T} \mathrm{M}\left(\mathbf{v} - \dot{\mathbf{q}}^{t}\right) + V(\mathbf{q}^{t} + \Delta t \mathbf{v}) \tag{32}$$

where V is the potential energy.

We solve the minimization problem Equation (30) by the *Newton's Method*. The main idea is to start from some initial guess $\mathbf{v}^{0}$ of the velocity and iteratively update the velocity while minimizing the energy function until we reach a desired convergence. At each iteration, we minimize a local quadratic approximation of the energy and figure out the direction $\mathbf{d}$ to move our current velocity variable as $\mathbf{v}^{i+1} = \mathbf{v}^{i+1} + \mathbf{d}$:

$$\mathbf{d} = \arg\min_{\tilde{\mathbf{d}}} \frac{1}{2}\tilde{\mathbf{d}}^{T} \left.\frac{\partial^2 E}{\partial \mathbf{v}^2}\right|_{\mathbf{v}^{i}} \tilde{\mathbf{d}} + \tilde{\mathbf{d}}^{T} \left.\frac{\partial E}{\partial \mathbf{v}}\right|_{\mathbf{v}^{i}} + E(\mathbf{v}^{i}). \tag{33}$$

Let $\mathrm{H}^{i} = \left.\frac{\partial^2 E}{\partial \mathbf{v}^2}\right|_{\mathbf{v}^{i}}$ and $\mathrm{g}^{i} = \left.\frac{\partial E}{\partial \mathbf{v}}\right|_{\mathbf{v}^{i}}$ denote the Hessian and the gradient of the energy at the iteration i. More explicitly:

$$\begin{aligned} \mathrm{H}^{i} &= \left.\frac{\partial^2 E}{\partial \mathbf{v}^2}\right|_{\mathbf{v}^{i}} = \mathrm{M} + \Delta t^2 \left.\frac{\partial^2 V}{\partial \mathbf{q}^2}\right|^{i}_{\mathbf{q}^{t} + \Delta t \mathbf{v}} \\ \mathrm{g}^{i} &= \left.\frac{\partial E}{\partial \mathbf{v}}\right|_{\mathbf{v}^{i}} = \mathrm{M}\left(\mathbf{v}^{i} - \dot{\mathbf{q}}^{t}\right) + \Delta t \left.\frac{\partial V}{\partial \mathbf{q}}\right|_{\mathbf{q}^{t} + \Delta t \mathbf{v}^{i}} \end{aligned} \tag{34}$$

Also, since $E(\mathbf{v}^{i})$ does not depend on the direction, we can drop it from the minimization problem:

$$\mathbf{d} = \arg\min_{\tilde{\mathbf{d}}} \frac{1}{2}\tilde{\mathbf{d}}^{T} \mathrm{H}^{i} \tilde{\mathbf{d}} + \tilde{\mathbf{d}}^{T} \mathrm{g}^{i}. \tag{35}$$

To find its minimum, we take the gradient with respect to $\tilde{\mathbf{d}}$ and end up with the following linear system to solve for $\mathbf{d}$:

$$\mathrm{H}^{i}\mathbf{d} + \mathrm{g}^{i} = 0. \tag{36}$$

However, since the direction is computed by an approximation to the energy, it is possible that taking a step in this direction may increase the energy instead. Instead, we will take a smaller step and make sure that the energy never increases. This is done by introducing a scaling parameter α such that the Newton velocity update becomes

$$\mathbf{v}^{i+1} = \mathbf{v}^{i} + \alpha \mathbf{d}. \tag{37}$$

The process of finding an appropriate α is called *line search*.

Given the direction $\mathbf{d}$, choosing *alpha* can also be viewed as an optimization problem:

$$\alpha^{*} = \arg\min_{\alpha} E(\mathbf{v}^{i} + \alpha \mathbf{d}). \tag{38}$$

We solve this minimization problem by starting from some predefined $\alpha_{\max}$ value and backtracking and reducing α if the *sufficient decrease* condition is not satisfied. Sufficient decrease condition is specified by the following inequality:

$$E(\mathbf{v}^{i} + \alpha \mathbf{d}) \leq E(\mathbf{v}^{i}) + c \cdot \mathbf{d}^{T} \mathrm{g}^{i} \tag{39}$$

where c is some user defined parameter. Line search is applied at every Newton step to ensure that the energy is indeed decreasing.

2.7 External spring forces

In order for us or the environment to interact with the deformable system, we will insert some external forces to the system. A simple way to create these forces is by adding artificial springs. If at least one of the ends of a spring is a vertex of our deformable system, we end up introducing energy, and thus forces to the system.

A spring has two endpoints. The generalized coordinates for a single spring $\mathbf{q}_s$ is a 6×1 vector with the two endpoint positions stacked in a vector.

The kinetic energy of a spring is given by:

$$T_s = \frac{1}{2}\dot{\mathbf{q}}^{T} \mathrm{M}_s \dot{\mathbf{q}}, \qquad \mathrm{M}_s = \begin{pmatrix} m_1 \cdot \mathrm{I} & 0 \\ 0 & m_2 \cdot \mathrm{I} \end{pmatrix} \tag{40}$$

where M_s is a diagonal, 6×1 mass matrix of the spring with particle masses m_1 and m_2 at its endpoints.

To describe the potential energy of a spring, we first define a strain that is invariant to rigid motion that is $l - l^{0}$ where l is the distance between the two endpoints of the spring and l^{0} is the rest

length of the spring. Then, the potential energy of a spring can be defined as:

$$V_s = \frac{1}{2}k(l - l^0)^2 \tag{41}$$

where k is the stiffness of the spring. To express V_s in terms of $\mathbf{q}_s$, first define the vector $\mathbf{dx}$ pointing from one endpoint to the other as

$$\mathbf{dx} = \underbrace{\begin{pmatrix}-\mathrm{I} & \mathrm{I}\end{pmatrix}}_{B}\mathbf{q}_s. \tag{42}$$

Note that $l = \sqrt{\mathbf{dx}^T\mathbf{dx}} = \sqrt{\mathbf{q}_s^T B^T B \mathbf{q}_s}$ and therefore

$$V_s = \frac{1}{2}k\left(\sqrt{\mathbf{q}_s^T B^T B\mathbf{q}_s} - l^0\right)^2. \tag{43}$$

The generalized forces due to a spring are then $-\frac{\partial V_s}{\partial \mathbf{q}_s}$.

External forces ($\mathrm{f}_{\mathrm{ext}}$) will appear in the right hand side of the equations of motion:

$$\mathrm{M}\ddot{\mathbf{q}} = -\frac{\partial V}{\partial \mathbf{q}} + \mathrm{f}_{\mathrm{ext}}. \tag{44}$$

3 METHOD

3.1 Assembly

Building the finite element method simulation is a matter of implementing the concepts discussed in Section 2 and scaling them to the whole system (tetrahedral mesh). The process of constructing vectors and matrices for forces and the mass/stiffness corresponding to the whole system is called *assembly*.

Assuming the system consists of N vertices, the generalized coordinates $\mathbf{q}$ is a vector of size $3N$ of stacked world space vertex coordinates, each with flattened x, y, z components. Similarly, the generalized velocities $\dot{\mathbf{q}}$ is a vector of size $3N$.

Since the mass matrix M_j of a single tetrahedron depends only on the basis functions evaluated at the undeformed state, we construct the mass matrix M as a precomputation. However, to get M_j, we need to integrate the products of basis functions over the tetrahedron (Equation (11)). For barycentric coordinates, we have the following expansion of the integral:

$$\rho\int_\Omega \phi_r(\mathbf{X})\phi_s(\mathbf{X})d\Omega = 6\rho\cdot\mathrm{vol}_j\int_0^1\int_0^{1-\phi_1}\int_0^{1-\phi_1-\phi_2}(\phi_r\phi_s)\,d\phi_3 d\phi_2 d\phi_3. \tag{45}$$

We use MATLAB symbolic integration to evaluate these integrals and obtain the 4×4 base matrix for M_j. Each entry of the base matrix is multiplied by an $\mathrm{I}_{3\times3}$ block, which results in the 12×12 tetrahedron mass matrix M_j. Next, each M_j needs to be assembled into a sparse $3N\times3N$ global mass matrix. Note that the effect of multiplying by element selection matrices from both sides as shown in Equation (14) results in each 3×3 block of M_j to be added to the 3×3 block in M starting at the corresponding vertex indices. For example, assume that we are processing a single tetrahedron and we know the mapping $T_j(u)$ which takes in the index $u = 0,\ldots,3$ for jth tetrahedron vertices and returns the index to the generalized coordinates corresponding to that vertex. Then, the block in M_j starting at the indices $(3u, 3v)$, should be added to the block in M starting at the indices $3T(u), 3T(v)$.

To assemble forces, we need to go back to the potential energy of the system as the generalized forces are given by $-\frac{\partial V}{\partial \mathbf{q}}$. Recall that the strain energy density depends on the deformation gradient F, and the deformation gradient is the derivative of the world space coordinates, with respect to the reference coordinates. More explicitly, we can write the mapping from the reference to the world space as follows:

$$\mathbf{x}(\mathbf{X}) = \mathbf{x}_0 + \begin{pmatrix}\mathbf{x}_0 & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3\end{pmatrix}\begin{pmatrix}-1^T\mathrm{T}^{-1}\\ \mathrm{T}^{-1}\end{pmatrix}(\mathbf{X}-\mathbf{X}_0) \tag{46}$$

where T is as described in Equation (3) and $1^T = \begin{pmatrix}1 & 1 & 1\end{pmatrix}$. Taking derivative with respect to $\mathbf{X}$:

$$F = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \begin{pmatrix}\mathbf{x}_0 & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3\end{pmatrix}\underbrace{\begin{pmatrix}-1^T\mathrm{T}^{-1}\\ \mathrm{T}^{-1}\end{pmatrix}}_{\frac{\partial\phi}{\partial\mathbf{X}}}. \tag{47}$$

Therefore, given a tetrahedron and current world space positions of its vertices, we can compute the deformation gradient. We have already provided the formulas for energy in Equations (18) to (20), so V_js can be computed.

To compute the derivative of the potential energy(Equation (23)), we need to take the derivative of the strain energy ψ, with respect to F. This is once again a complicated task to do by hand and we use the MATLAB symbolic toolkit. We write out the formula for the energy, vectorize F in column major order and compute the derivative. Now, we can write out a function which takes the deformation gradient as input and outputs $\frac{\partial\psi}{\partial F}_{9\times1}$ vector. Lastly, we compute $-\mathrm{B}_j^T\frac{\partial\psi}{\partial F}$ to obtain the generalized forces for the tetrahedron where B_j is a 9×12 matrix with the entries of $\frac{\partial\phi}{\partial\mathbf{X}}$ each written as a 3×3 block diagonal matrix. The matrix B_j maps the vectorized world space coordinates to the vectorized deformation gradient. The assembly of forces is simply to add the forces found for each tetrahedron vertex to the corresponding index in the global generalized forces vector.

The assembly of the stiffness matrix requires the per tetrahedron $\frac{\partial^2 V}{\partial\mathbf{q}^2}$. This is obtained similarly to $\frac{\partial V}{\partial\mathbf{q}}$ we computed to figure out the forces. This time, we use the MATLAB symbolic toolkit to compute the Hessian of the energy density ψ with respect to F. Therefore, the function we write using the symbolic computation output takes F and returns $\frac{\partial^2\psi}{\partial F^2}_{9\times9}$. Then, the stiffness matrix of the tetrahedron is given by $-\mathrm{B}_j^T\frac{\partial^2\psi}{\partial F^2}\mathrm{B}_j$. To assemble them into the sparse global stiffness matrix, we repeat the same process as we did for the mass matrix.

3.2 Gravity

We will once again utilize the variational approach to figure out the forces due to gravity. The relative potential energy of a particle inside some tetrahedron is given by

$$\rho\, g\, h(\mathbf{X})d\Omega \tag{48}$$

where g is the gravitational constant and $h(\mathbf{X})$ is the height of the particle. We will take the y component of the particle position as

its height. Then the function $h(\mathbf{X})$ can be expressed in terms of the basis functions as follows:

$$h(\mathbf{X}) = \underbrace{\begin{pmatrix}\phi_0 & \phi_1 & \phi_2 & \phi_3\end{pmatrix}}_{\mathrm{n}(\mathbf{X})} \underbrace{\begin{pmatrix}y_0\\ y_1\\ y_2\\ y_3\end{pmatrix}}_{\overrightarrow{y}} \tag{49}$$

where y_i are the heights of the world space tetrahedron vertices. Now, we integrate Equation (48) over the tetrahedron:

$$\int_\Omega \rho\, g\, \mathrm{n}(\mathbf{X})\overrightarrow{y}\, d\Omega. \tag{50}$$

As the world space coordinates of the vertices are constant over the tetrahedron, taking $\overrightarrow{y}$ out of the integral and integrating the basis functions similarly to Equation (45), we get the following expression for the potential energy of a tetrahedron due to gravity:

$$V_j^g = g \cdot \mathrm{vol}_j \cdot \rho \begin{pmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4}\end{pmatrix} \begin{pmatrix}y_0\\ y_1\\ y_2\\ y_3\end{pmatrix}. \tag{51}$$

Then, to find the forces, we take its derivative with respect to $\overrightarrow{y}$:

$$\frac{\partial V_j^g}{\partial \overrightarrow{y}} = g \cdot \mathrm{vol}_j \cdot \rho \begin{pmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4}\end{pmatrix}, \tag{52}$$

and the forces acting on the jth tetrahedron are given by $-\frac{\partial V_j^g}{\partial \overrightarrow{y}}$. Note that since we found the solution by reducing the coordinates to a single dimension (y), we need to multiply $-\frac{\partial V_j^g}{\partial \overrightarrow{y}}$ by the vector $\begin{pmatrix}0 & 1 & 0\end{pmatrix}^T$ to get the force acting on a 3D vertex. Gravitational forces can then be added as external forces and assembled in the usual way.

3.3 Springs

Springs are introduced to respond to interactions and collisions. The spring forces are also added as external forces. To find the forces due to a spring, we differentiate Equation (43) by MATLAB symbolic toolkit.

The collision detection with the ground is to simply check for the y coordinates of each vertex to see if they are smaller than some $y_{\min}$ with some tolerance.

For the collision response with the ground, we introduce springs with endpoints the collision vertex and some y offset version of it, with a user-specified stiffness k. However, springs require quite a bit of fine tuning of the stiffness to achieve a plausible animation. For instance, if we put the object on the ground with very stiff springs, instead of staying in place, it will get pushed up spuriously, or we will observe some artifacts where a vertex is constantly going up and down. On the other hand, if the springs are not stiff enough, the object will fall through the ground. We experimented with varying the stiffness of the springs depending on the y component of the velocity of the colliding vertex, however this didn't end up being too successful either, it still requires fine tuning and we have made it even more complicated.

3.4 Boundary conditions

To fix some of the vertices at their initial positions and never allow them to move during the simulation is accomplished by introducing them as boundary conditions. To do so, we use a special selection matrix P such that $P\mathbf{q} = \hat{\mathbf{q}}$ where $\mathbf{q}$ is the generalized coordinates of all (fixed + unfixed) vertices, and $\hat{\mathbf{q}}$ is the generalized coordinates of vertices that are not fixed. Our simulation loop only updates $\hat{\mathbf{q}}$ and its derivative the velocities $\dot{\hat{\mathbf{q}}}$.

P is a sparse $(3N - 3b) \times 3N$ matrix with b, the number of fixed point constraints. P has an $\mathrm{I}_{3\times3}$ block if the corresponding index in the column is not a fixed point.

To get the generalized coordinates satisfying the boundary conditions, prepare a vector b that is of the same size as $\mathbf{q}$ and contains the fixed positions at the appropriate indices and has 0 for all nonfixed vertices. Then, $\mathbf{q} = P^T\hat{\mathbf{q}} + \mathrm{b}$ is our final generalized coordinates. Similarly, the velocities can be computed by $\dot{\mathbf{q}} = P^T\dot{\hat{\mathbf{q}}}$.

After their computation, the mass matrix, the forces, and the stiffness matrix need to be projected into the nonfixed vertices. This is because their computation still take into account all the vertices, but we can simply ignore the values corresponding to the fixed vertices by taking:

$$\begin{aligned}\hat{\mathrm{f}} &= P\,\mathrm{f}\\ \hat{\mathrm{M}} &= P\,\mathrm{M}\,P^T\\ \hat{\mathrm{K}} &= P\,\mathrm{K}\,P^T.\end{aligned} \tag{53}$$

3.5 Skinning

Our spatial discretization with finite elements is based on a tetrahedral mesh of an object. We can find fine detailed triangle mesh models [9], for which we can generate a quality tetrahedral mesh [5, 6]. However, the computation time for our simulation will be proportional to the number of tetrahedra. In addition, the visual difference of the simulation with a very large number of tetrahedra and a moderate number of tetrahedra may be too small to justify the excessive computation times. The idea of skinning is to have a coarse tetrahedral mesh that the simulation will run on but display a detailed triangle mesh instead.

The finite element discretization allows us to express any point inside a tetrahedron by a product of basis functions and the generalized coordinates of the tetrahedra (Equation (6)). Therefore, as long as the points of the detailed skinning mesh is fully contained inside the coarse simulation mesh, we can determine how those points deform and render the detailed skinning mesh.

To accomplish this task, we will build a *skinning weight matrix* W such that obtaining the positions for rendering is just a matter of matrix vector multiplication:

$$\mathbf{x}_{\text{surface}} = \mathrm{W}\mathbf{q}. \tag{54}$$

W is a sparse $l \times N$ matrix where l is the number of display vertices. Each row of W, corresponds to an undeformed display position, say $\mathbf{X}_d$. First, we loop over all tetrahedra to find out which one contains $\mathbf{X}_d$. Then, we evaluate the basis functions of the tetrahedron at $\mathbf{X}_d$ and write these four values into the columns corresponding to the indices of vertices of the tetrahedron. Note that finding the tetrahedron and evaluating the basis functions for the display point

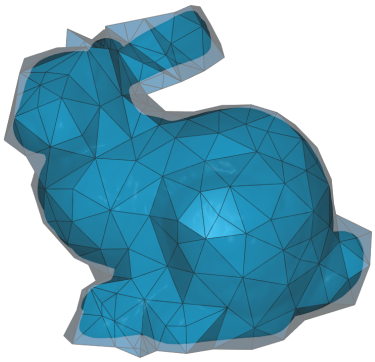

Fig. 2. The coarse tetrahedral mesh and the detailed skinning mesh laid on top of one another. The tetrahedral mesh has to completely enclose the skinning mesh vertices.

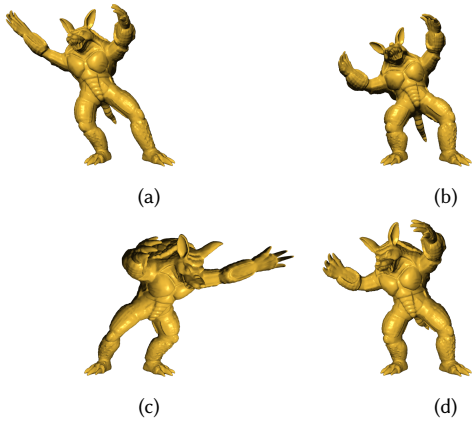

Fig. 3. Testing the stability of strain energies. (a, c) Stiff armadillo is pulled from its hands (b, d) The forces pulling the armadillo are suddenly let go.

can be performed as a precomputation and we only need to build W once.

We are not aware of a straightforward way to generate these simulation meshes that are guaranteed to enclose the skinning mesh (although we are inclined to believe that it is probable a method exists). For the dragon model, we first slightly offset the vertices of the skinning mesh in the direction of the face normals, then remesh it to obtain a coarse triangle mesh. This coarse triangle mesh is manually made to enclose the original triangle mesh. Then, it is given to *fast Tet-Wild* by Hu et al. [5] which generates the coarse tetrahedral mesh and it seems to preserve the boundaries for the most part.

4 RESULTS & EVALUATION

As mentioned in Section 2.4, we have implemented three different strain energy densities. In this section, we compare their stability and volume preservation properties.

Firstly, the setting in Figure 1 is that we have a large dragon object with some boundary conditions at its front feet, essentially gluing it onto the floor. The floor in this experiment does not exert penalty forces and is just for decoration. The material properties are set as *Young's modulus* $\mathrm{E} = 6 \cdot 10^6$ and *Poisson's ratio* $\nu = 0.4$. The Lamé parameters are then given by the following formulas:

$$\begin{aligned} \mu &= \frac{\mathrm{E}}{2(1+\nu)} \\ \lambda &= \frac{\mathrm{E}\nu}{(1+\nu)(1-2\nu)}. \end{aligned} \tag{55}$$

Using linearly-implicit backward Euler integration with timestep $0.33 \cdot 10^{-2}$, we start pulling the dragon from its head and tail in opposite directions by introducing springs. We slowly increase the forces on the springs by increasing the distance between its endpoints (one endpoint on the dragon, the other floating) and then let go (don't add a new spring in the next timestep). The simulations with strain energies ψ_{BW} and psi_{S} allow deformations in a plausible way, with small differences in the animation. However, the strain energy ψ_B explodes after letting go of the springs.

In Figure 6a, we plot the change in the total volume of the dragon. As it stretches, it gains volume. We observe that using ψ_{B}, the volume gain is about 3 times larger than the other two energies, which is probably one of the reasons why the simulation crashes. We also see that ψ_{S} has smaller volume gain during the stretch compared to ψ_{BW}, which is in accordance with the paper by Smith et al. [11]. However, interestingly, when the dragon is compressing, ψ_{S} has a larger volume loss.

In Figure 6b, we try to get an idea of the local change in volume. We plot the maximum change in volume of any tetrahedra. The observations are similar to the total volume case.

As our second experiment, we change the object to an armadillo with similar boundary conditions. The main difference between the armadillo and the dragon is that the armadillo object has a much smaller volume and therefore is more stiff. For stability concerns, we change the integrator to the fully-implicit one and reduce the timestep to $0.4 \cdot 10^3$. We once again apply spring forces to stretch the object for a while and then stop. Figure 3 presents a few frames from this animation. Unfortunately, both of the simulations using ψ_{B} and ψ_{BW} explode. In Figures 6c and 6d, we present plots for total change and the maximum change in volume. The results are similar to the dragon case.

Lastly, we present an example with gravity and penalty forces. Here, we use the linearly-implicit time integrator with $\Delta t = 0.16 \cdot 10^2$. Figure 4 shows the dragon being let go from a certain height and falling with gravitational forces acting on it. Upon contact, the object slightly goes through the floor as penalty forces cannot enforce a hard constraint. Throughout the animation, we observe small jittering on the feet of the dragon with the vertices constantly penetrating the object and being pushed back as a result.

5 CONCLUSIONS & FUTURE WORK

We have discussed how to build a standard finite element method simulation for deformables. We took the variational approach to defining the motion of our objects, which requires expressing quantities such as forces and stiffness in terms of the potential energy derivatives. To define the potential energy, we chose a Neo-Hookean

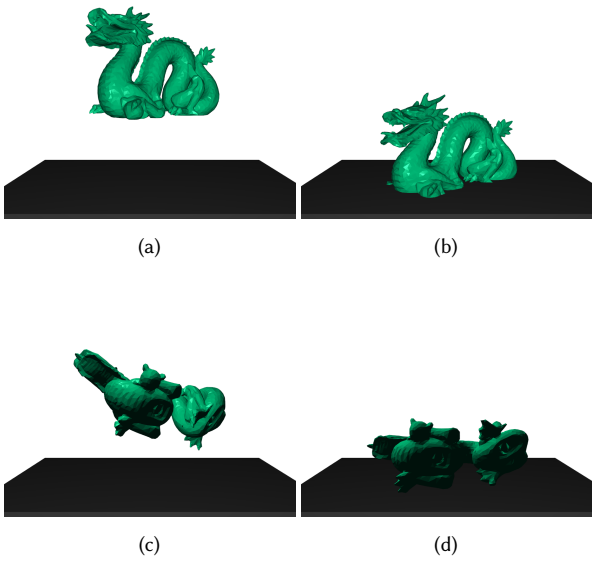

Fig. 4. (a) The dragon falls due to gravity. (b) The penalty forces try to push the colliding vertices upwards but dragon still slightly penetrates the floor. (c) Dragon bounces thanks to its potential energy and the penalty forces. (d) Eventually dragon stabilizes and stays on top of the ground with gravity and penalty forces working against one another.

strain energy density function and compared three different Neo-Hookean energy variants. We verified that the stable Neo-Hookean [11] has better stability and volume preservation qualities.

Currently, the project deals with collisions in a simple manner. We only detect collisions with the ground and introduce spring penalty forces to resolve them. Extending collision detection to multiple objects is not too challenging, we just need to check for all vertices if they are inside some tetrahedron belonging to an object. To do so efficiently will probably require dynamically updated bounding volume hierarchies [12]. On the collision handling end, the penalty springs we introduce are not hard constraints and it is possible for the object to briefly penetrate the object for a few frames. Moreover, the spring parameters require a fair bit of fine tuning to get the desired animation. Therefore, dealing with collisions in a more physically based manner would be a priority as a future work.

Another significant limitation is the fact that self-collisions are not handled. Figure 5 presents an example of a self-collision artifact. Addressing this problem efficiently could make interesting future work.

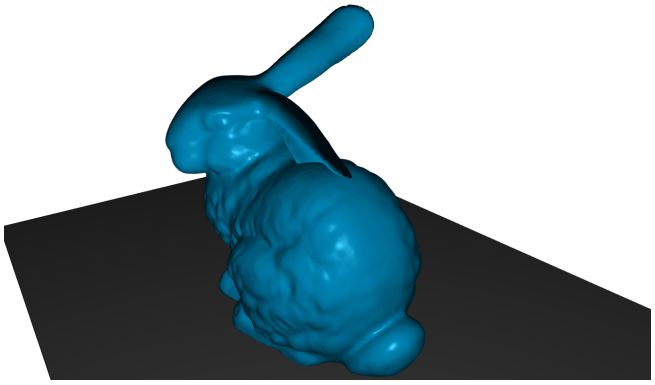

Fig. 5. The bunny is being pulled from its left ear into its body, leading to visible self collision artifacts.

REFERENCES

[1] David Baraff and Andrew Witkin. 1998. Large Steps in Cloth Simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. Association for Computing Machinery, New York, NY, USA, 43–54. https://doi.org/10.1145/280814.280821

[2] Javier Bonet and Richard D. Wood. 2008. *Nonlinear Continuum Mechanics for Finite Element Analysis.* Cambridge University Press.

[3] Allan F. Bower. 2009. *Applied Mechanics of Solids.* CRC Press.

[4] Neil Gershenfeld. 1999. *The Nature of Mathematical Modeling.*

[5] Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. 2020. Fast Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 39, 4, Article 117 (July 2020), 18 pages. https://doi.org/10.1145/3386569.3392385

[6] Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 37, 4, Article 60 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201353

[7] Fridtjov Irgens. 2008. *Continuum Mechanics.* Springer Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-74298-2

[8] David I.W. Levin. Fall 2021. CSC417/CSC2549-Physics-based Animation. https://github.com/dilevin/CSC417-physics-based-animation.

[9] Morgan McGuire. 2017. Computer Graphics Archive. https://casual-effects.com/data `https://casual-effects.com/data`.

[10] Eftychios Sifakis and Jernej Barbic. 2012. FEM Simulation of 3D Deformable Solids: A Practitioner's Guide to Theory, Discretization and Model Reduction. In *ACM SIGGRAPH 2012 Courses* (Los Angeles, California) *(SIGGRAPH '12)*. Association for Computing Machinery, New York, NY, USA, Article 20, 50 pages. https://doi.org/10.1145/2343483.2343501

[11] Breannan Smith, Fernando De Goes, and Theodore Kim. 2018. Stable Neo-Hookean Flesh Simulation. *ACM Trans. Graph.* 37, 2, Article 12 (mar 2018), 15 pages. https://doi.org/10.1145/3180491

[12] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. 2005. Collision Detection for Deformable Objects. *Computer Graphics Forum* 24, 1 (2005), 61–81. https://doi.org/10.1111/j.1467-8659.2005.00829.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2005.00829.x

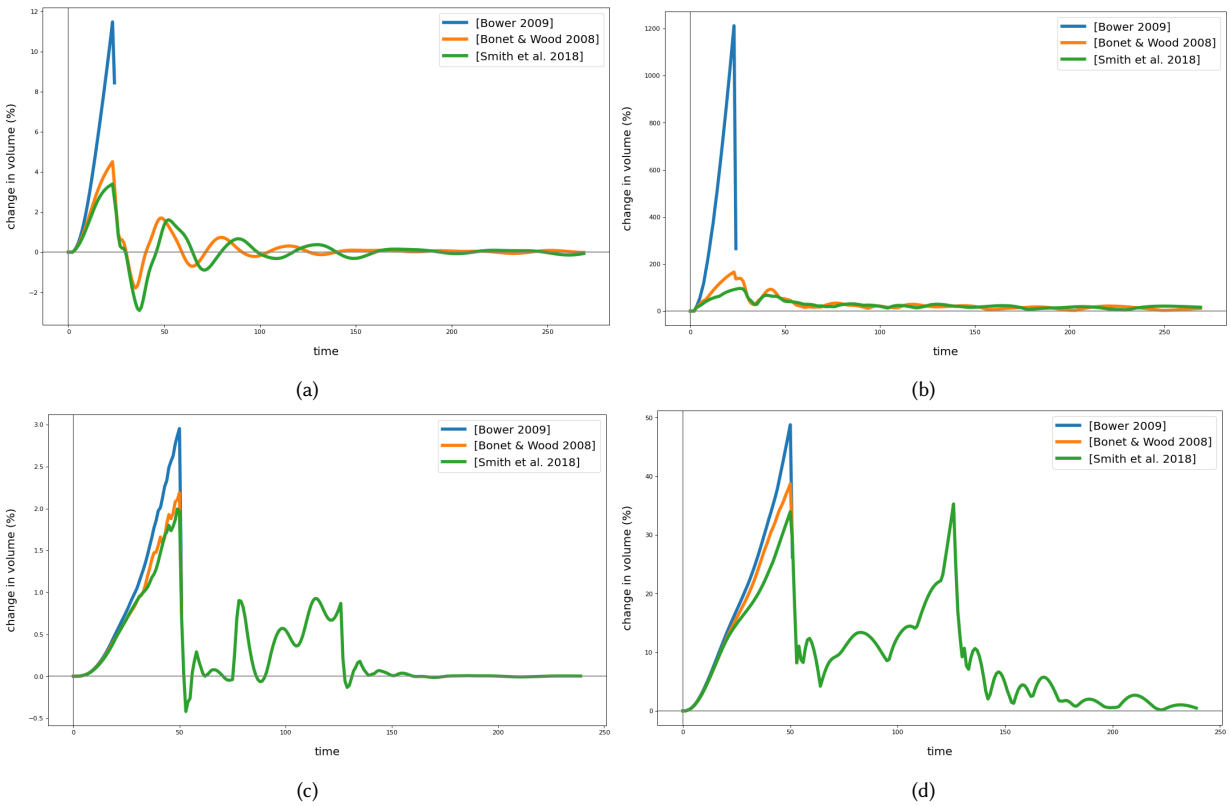

Fig. 6. Plots (a, c) showing the total change in volume over the animation and (b, d) showing the volume change of a single tetrahedron which has the largest volume increase. The first row corresponds to an experiment with the dragon, the second row with the armadillo.