Real-time FEM continuum mechanics coupled with articulated rigid body dynamics

Feodor Benevolensky,
Moscow

Andrey Voroshilov,
Moscow

15th October, 2009

Abstract

This paper will show how to derive formulae needed to simulate FEM-based elastic/plastic deformables, using formulations typically used in constrained rigid body dynamics.

Introduction

In this article we'd like to describe a way of simulating deformable bodies dynamics using Finite Element Method (FEM). Although this method is widely spread in the engineering fields, in game development industry - FEM only starts to get utilized and hence there is not much available information in specifics of using FEM in game development, especially there are not much high level overviews of the method. In this article, we'll try to explain all the important definitions and formulae without requiring strong engineering background from reader. This method initially designed in a way that integration with already existing rigid body dynamics engines should be simple and straightforward.

The method we propose also has its negative sides. First of all, it is quite computationally heavy task; and it requires higher accuracy of solving the governing system, otherwise noticeable artifacts will appear.

This article begins with defining basic terminology and deriving formulae, closer to the end solutions to specific problems emerge. Theory in this article is mainly based on books by Landau and Lifshitz [1].

Contents

1. Theory

1.1. Deformation. Strain tensor

We will describe deformation using Theory of Elasticity which formulates mechanics of solid bodies, regarded as continuous medium.

Under the action of applied forces, solid bodies exhibit deformation to some extent, i.e. they change in shape and volume. The deformation of a body is described in the following way. The position of any point in the body is defined by its radius vector $\mathbf{r}$ (with components $\mathbf{x}_1 = x$, $\mathbf{x}_2 = y$, $\mathbf{x}_3 = z$) in some coordinate system. When the boy is deformed, every point in it is in general displaced. Let us consider some particular point; let its radius vector before the deformation be $\mathbf{r}$, and after the deformation have a different value $\mathbf{r}'$ (with components $\mathbf{x}'_i$). The displacement of this point due to the deformation is then given by the vector $\mathbf{r}' - \mathbf{r}$, which we will denote by $\mathbf{u}$:

$$\mathbf{u}_i = \mathbf{x}'_i - \mathbf{x}_i$$

The vector $\mathbf{u}$ is called the *displacement vector*. The coordinates $\mathbf{r}'$ of the displaced point are function of the coordinates $\mathbf{r}$ of the point before displacement, therefore the displacement vector is also a function of the coordinates before displacement. If the vector $\mathbf{u}$ is given as a function of x_i, the deformation of the body is entirely determined.
When a body is deformed, the distances between its points change. Let us consider two points close together, $\mathbf{r}_0$ and $\mathbf{r}_1$. If the radius vector joining them before the deformation is $\mathrm{d}\mathbf{x} = \mathbf{r}_1 - \mathbf{r}_0$, the radius vector joining the same two points in the deformed body is $\mathrm{d}\mathbf{x}' = \mathbf{r}'_1 - \mathbf{r}'_0 = \mathbf{r}_1 + \mathbf{u}_1 - \mathbf{r}'_0 - \mathbf{u}'_0 = \mathrm{d}\mathbf{x} + \mathrm{d}\mathbf{u}$. The distance between the points is $\mathrm{d}l = \sqrt{\mathrm{d}\mathbf{x}_1^2 + \mathrm{d}\mathbf{x}_2^2 + \mathrm{d}\mathbf{x}_3^2}$ before the deformation, and $\mathrm{d}l' = \sqrt{\mathrm{d}\mathbf{x}'^2_1 + \mathrm{d}\mathbf{x}'^2_2 + \mathrm{d}\mathbf{x}'^2_3}$ after it. Using Σ notation, we can rewrite:

$$\mathrm{d}l'^2 = \sum_{i=1}^{3} \mathrm{d}\mathbf{x}'^2_i = \sum_{i=1}^{3} (\mathrm{d}\mathbf{x}_i + \mathrm{d}\mathbf{u}_i)^2$$

Expanding brackets gives us:

$$\mathrm{d}l'^2 = \mathrm{d}l + \sum_{i=1}^{3} (\mathrm{d}\mathbf{u}_i^2 + 2\,\mathrm{d}\mathbf{u}_i\,\mathrm{d}\mathbf{x}_i)$$

Substituting $\mathrm{d}\mathbf{u}_i = (\partial\mathbf{u}_i/\partial\mathbf{x}_k)\,\mathrm{d}\mathbf{x}_k$, we can write:

$$\mathrm{d}l'^2 = \mathrm{d}l^2 + \sum_{i=1}^{3}\sum_{k=1}^{3} \left(2\frac{\partial\mathbf{u}_i}{\partial\mathbf{x}_k}\,\mathrm{d}\mathbf{x}_i\,\mathrm{d}\mathbf{x}_k\right) + \sum_{i=1}^{3}\sum_{k=1}^{3}\sum_{l=1}^{3} \left(\frac{\partial\mathbf{u}_i}{\partial\mathbf{x}_k}\frac{\partial\mathbf{u}_i}{\partial\mathbf{x}_l}\,\mathrm{d}\mathbf{x}_k\,\mathrm{d}\mathbf{x}_l\right)$$

Since the summation is taken over both suffixes i and k in the second term on the right, we can put $\frac{\partial\mathbf{u}_i}{\partial\mathbf{x}_k}\,\mathrm{d}\mathbf{x}_i\,\mathrm{d}\mathbf{x}_k = \frac{\partial\mathbf{u}_k}{\partial\mathbf{x}_i}\,\mathrm{d}\mathbf{x}_i\,\mathrm{d}\mathbf{x}_k$. In the third term, we interchange the suffixes i and l. Then $\mathrm{d}l'^2$ takes the final form:

$$\mathrm{d}l'^2 = \mathrm{d}l^2 + \sum_{i=1}^{3}\sum_{k=1}^{3} (2[u]_{ik}\,\mathrm{d}\mathbf{x}_k\,\mathrm{d}\mathbf{x}_i) \tag{1}$$

where the tensor $[u]$ is defined as:

$$[u]_{ik} = \frac{1}{2}\left(\frac{\partial\mathbf{u}_i}{\partial\mathbf{x}_k} + \frac{\partial\mathbf{u}_k}{\partial\mathbf{x}_i} + \sum_{l=1}^{3}\frac{\partial\mathbf{u}_l}{\partial\mathbf{x}_i}\frac{\partial\mathbf{u}_l}{\partial\mathbf{x}_k}\right) \tag{2}$$

These expressions give the change in an element of length when the body is deformed.

The tensor $[u]$ is called the *strain tensor*. We see from its definition that it is symmetrical, i.e.

$$[u]_{ik} = [u]_{ki}$$

this tensor is also a function of coordinates of the body prior to deformation and is different in each point of a body.

Like any symmetrical tensor, $[u]$ can be *diagonalized* at any given point. This means that, at any given point, we can choose coordinate axes (the *principal axes* of the tensor) in such a way that only the diagonal components $[u]_{11}$, $[u]_{22}$, $[u]_{33}$ (or, in different notation, $[u]_{xx}$, $[u]_{yy}$, $[u]_{zz}$) of the tensor $[u]$ are different from zero.

If the strain tensor is diagonalized at a given point, the element of length (1) near it becomes

$$\mathrm{d}l'^2 = (1 + 2[u]_{xx})\,\mathrm{d}\mathbf{x}_1^2 + (1 + 2[u]_{yy})\,\mathrm{d}\mathbf{x}_2^2 + (1 + 2[u]_{zz})\,\mathrm{d}\mathbf{x}_3^2$$

We see that the expression is the sum of three independent terms. This means that the strain in any volume element may be regarded as composed of independent strains in three mutually perpendicular directions, namely those of the principal axes of the strain tensor. Each of these strains is a simple extension (or compression) in the corresponding direction: the length $\mathrm{d}\mathbf{x}_x$ along the first principal axis becomes $\mathrm{d}\mathbf{x}'_x = \sqrt{(1 + 2[u]_{xx})}\,\mathrm{d}\mathbf{x}_x$, and similarly for the other two axes. The quantity $\sqrt{(1 + 2[u]_{ii})} - 1$ is consequently equal to the relative extension $(\mathrm{d}\mathbf{x}'_i - \mathrm{d}\mathbf{x}_i)/\,\mathrm{d}\mathbf{x}_i$ along the i^{th} principal axis.

In almost all cases occurring in practice, the strains are small. This means that the change in any distance in the body is small compared with the distance itself. In other words, the relative extensions are small compared with unity.

If a body is subjected to a small deformation, all the components of the strain tensor are small, since they give, as we have seen, the relative changes in lengths in the body. The displacement vector $\mathbf{u}$, however, may sometimes be large,even for small strains. For example, let us consider a long thin rod. Even for a large deflection, in which the ends of the rod move a considerable distance, the extensions and compressions in the rod itself will be small. Along with rod deformations, examples of large displacement vectors may be deformations of thin plates to form cylindrical surfaces, or a case when body is rotated around some axis, together with being deformed.

Except in such special cases, the displacement vector for a small deformation is itself small. A three-dimensional body (i.e. one whose dimension in no direction is small) cannot be deformed in such a way that parts of it move a considerable distance without the occurrence of considerable extensions and compressions in the body. In what fallows we will suppose that all strains are small, the problem of rotating bodies will be solved specifically further in the article.

Given that deformations are small, we can neglect the last term in the general strain tensor expression (2), as being of the second order of smallness. Thus, for small deformations, the strain tensor is given by:

$$[u]_{ik} = \frac{1}{2}\left(\frac{\partial \mathbf{u}_i}{\partial \mathbf{x}_k} + \frac{\partial \mathbf{u}_k}{\partial \mathbf{x}_i}\right) \tag{3}$$

In this form, strain tensor is also called *infinitesimal strain tensor*, also called *Cauchy's strain tensor*, *linear strain tensor*, or *small strain tensor*.

The relative extensions of the elements of length along the principal axes of the strain tensor (at a given point) are, to within higher-order quantities, $\sqrt{(1+2[u]_{ii})}-1 \approx [u]_{ii}$, i.e. they are the principal values of the tensor $[u]$.

Let us consider an infinitesimal volume element $\mathrm{d}V$, and find its volume $\mathrm{d}V'$ after the deformation. To do so, we take the principal axes of the strain tensor, at the point considered, as the coordinate axes. Then the elements of length $\mathrm{d}\mathbf{x}_x$, $\mathrm{d}\mathbf{x}_y$, $\mathrm{d}\mathbf{x}_z$ along these axes become, after the deformation, $\mathrm{d}\mathbf{x'}_x = (1+[u]_{xx})\,\mathrm{d}\mathbf{x}_x$, etc. The volume $\mathrm{d}V$ is the product $\mathrm{d}\mathbf{x}_x\,\mathrm{d}\mathbf{x}_y\,\mathrm{d}\mathbf{x}_z$, while $\mathrm{d}V'$ is $\mathrm{d}\mathbf{x'}_x\,\mathrm{d}\mathbf{x'}_y\,\mathrm{d}\mathbf{x'}_z$. This $\mathrm{d}V' = \mathrm{d}V(1+[u]_{xx})(1+[u]_{yy})(1+[u]_{zz})$. Neglecting higher-order terms, we therefore have $\mathrm{d}V' = \mathrm{d}V(1+[u]_{xx}+[u]_{yy}+[u]_{zz})$. The sum $[u]_{xx}+[u]_{yy}+[u]_{zz}$ of the principal values of a tensor is well known to be invariant, and is equal to the sum of the diagonal components

$$\sum_{i=1}^{3}[u]_{ii} = [u]_{xx}+[u]_{yy}+[u]_{zz}$$

In any coordinate system. Thus

$$\mathrm{d}V' = \mathrm{d}V\left(1+\sum_{i=1}^{3}[u]_{ii}\right)$$

We see that the sum of the diagonal components of the strain tensor is the relative volume change $(\mathrm{d}V'-\mathrm{d}V)/\,\mathrm{d}V$.

1.2. Stress tensor

Along with the deformations, another important definition in the theory of elasticity is stress. All forces acting on the considered continuous medium particle can be divided into *volumetric* or *surface*. Volumetric forces - forces exerted by other objects (including other particles of continuous medium), acting on all points of considered particle throughout its volume. Volumetric force can be measured by its intensity. Consider a small continuous medium particle with volume $\mathrm{d}V$ in the neighborhood of point M. Mass of the particle can be calculated as $\rho\,\mathrm{d}V$ where ρ is the density at point M. Let's suppose that some volumetric force $\mathrm{d}\mathbf{F}$ is acting on considered continuous medium particle, then the intensity of such force in M is:

$$\mathbf{F} = \lim_{\mathrm{d}V\to 0}\frac{\mathrm{d}\mathbf{F}}{\rho\,\mathrm{d}V}$$

Examples of volumetric forces are gravitational forces, inertial force, etc.

Volumetric forces can be put into contrast with surface forces, which are supposed to be exerted to the surface of considered continuous medium particle. When the body is subjected to external surface forces (or contact forces) $\mathbf{P}$, internal contact forces and moments are transmitted from point to point in the body, and from one segment to the other through the dividing surface S, due to the mechanical contact of one portion of the continuum onto the other. As $\mathrm{d}S$ becomes very small, resultant vector $\frac{\mathrm{d}\mathbf{P}}{\mathrm{d}S}$ can be defined as the *surface traction*:

$$\mathbf{T} = \lim_{\mathrm{d}S\to 0}\frac{\mathrm{d}\mathbf{P}}{\mathrm{d}S}$$

Surface forces acting on considered particle have counterparts in rigid body dynamics - reaction forces, distributed on a surface of a rigid body. Surfaces of several particles of continuous medium can be found in a particular point of space. And quantities of said forces in that point

of space can be calculated, in order to do that *stress tensor* needs to be known at the point.

Surfaces of several particles at a given point each have its own normal vector $\mathbf{n}$.

Let us consider a small tetrahedral particle $OABC$ of continuous medium. We also introduce coordinate system such that vertex O of this tetrahedron is the origin, and its edges $\mathbf{OA}$, $\mathbf{OB}$ and $\mathbf{OC}$ are orthogonal vectors. Three faces of this tetrahedron lie in the coordinate planes, and face ABC is inclined and its surface is $\mathrm{d}S_n$. Angles between the outward normal to this face and coordinate axes are α, β, γ. External normal to face OBC is negative direction of $\mathbf{x}$-axis, and the surface area is $\mathrm{d}S_x$. Similarly, normal and surface for OAC are negative $\mathbf{y}$-axis and $\mathrm{d}S_y$, and for OAB - negative $\mathbf{z}$-axis and $\mathrm{d}S_z$.

For the considered small tetrahedron, we can consider applying corollary of d'Alembert's principle, which states that summed vectors of all the forces, acting on a considered tetrahedron volumetrically, together with inertial forces give zero-vector sum. Sum of volumetric forces can be calculated as $\mathbf{F}\rho\,\mathrm{d}V$, where $\mathbf{F}$ - average intensity of volumetric force, ρ is the average density, and $\mathrm{d}V$ - is the volume of the tetrahedron. Surface forces, acting on e.g. face OBC, could be calculated as $\mathbf{T}_{-\mathbf{x}}\,\mathrm{d}S_x$, similarly for OAC it would be $\mathbf{T}_{-\mathbf{y}}\,\mathrm{d}S_y$, for OAB calculated as $\mathbf{T}_{-\mathbf{z}}\,\mathrm{d}S_z$, and, finally, for ABC that would be $\mathbf{T}_{\mathbf{n}}\,\mathrm{d}S_n$. Inertial forces for all of the points of the considered tetrahedron: $-\mathbf{a}\rho\,\mathrm{d}V$, where $\mathbf{a}$ is the average acceleration. As stated earlier, vector sum will be zero:

$$\mathbf{F}\rho\,\mathrm{d}V - \mathbf{a}\rho\,\mathrm{d}V + \mathbf{T}_{-\mathbf{x}}\,\mathrm{d}S_x + \mathbf{T}_{-\mathbf{y}}\,\mathrm{d}S_y + \mathbf{T}_{-\mathbf{z}}\,\mathrm{d}S_z = \mathbf{0} \tag{4}$$

Rectilinear parallel projection of an inclined surface $\mathrm{d}S_n$ allows us to calculate derived surface areas: $\mathrm{d}S_x = \mathrm{d}S_n\cos\alpha$, $\mathrm{d}S_y = \mathrm{d}S_n\cos\beta$, $\mathrm{d}S_z = \mathrm{d}S_n\cos\gamma$. Tetrahedron volume can be calculated as $\mathrm{d}V = \frac{1}{3}h\,\mathrm{d}S_n$, where h - is the height of the tetrahedron, considering the plane ABC as the base. Considering that

$$\lim_{h\to 0}\frac{\mathrm{d}V}{\mathrm{d}S_n} = \frac{1}{3}\lim_{h\to 0} h = 0$$

We can divide both sides of equation (4) by $\mathrm{d}S_n$:

$$\mathbf{T}_{\mathbf{n}} + \mathbf{T}_{-\mathbf{x}}\cos\alpha + \mathbf{T}_{-\mathbf{y}}\cos\beta + \mathbf{T}_{-\mathbf{z}}\cos\gamma = \mathbf{0} \tag{5}$$

Equation (5) consists of surface traction at point O and not average tractions. This equation shows that resulting vector of all the surface forces for a infinitesimal tetrahedron is zero. And this is applicable to any shape, since volume to surface ratio in similar limit is also zero.

Using Newton's third law (for every action, there is an equal and opposite reaction), we can get $\mathbf{T}_{-\mathbf{x}} = -\mathbf{T}_{\mathbf{x}}$, $\mathbf{T}_{-\mathbf{y}} = -\mathbf{T}_{\mathbf{y}}$, $\mathbf{T}_{-\mathbf{z}} = -\mathbf{T}_{\mathbf{z}}$. As an example, consider part of continuous medium (that we previously did not take into consideration), which exerts some traction $\mathbf{T}_{-\mathbf{x}}$ through face OBC; then, $\mathbf{T}_{\mathbf{x}}$ is the traction that tetrahedron exerts through the same face on the continuous medium. And for the medium, outward normal is positive direction of $\mathbf{x}$-axis. Similar logic can be applied to another two faces as well. Given that, we can write:

$$\mathbf{T}_{\mathbf{n}} = \mathbf{T}_{\mathbf{x}}\cos\alpha + \mathbf{T}_{\mathbf{y}}\cos\beta + \mathbf{T}_{\mathbf{z}}\cos\gamma \tag{6}$$

Equation (6) expresses traction to an inclined plane through three mutually perpendicular planes that all have common point O. In general case, $\mathbf{T}_{\mathbf{x}}$, $\mathbf{T}_{\mathbf{y}}$, $\mathbf{T}_{\mathbf{z}}$ are not exactly collinear with the $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ axes, similarly to inclined face that is associated with $\mathbf{T}_n$. Then, coordinates projections will give us:

$$\mathbf{T}_{\mathbf{n}x} = \mathbf{T}_{\mathbf{x}x}\cos\alpha + \mathbf{T}_{\mathbf{y}x}\cos\beta + \mathbf{T}_{\mathbf{z}x}\cos\gamma$$

$$\mathbf{T}_{\mathbf{n}y} = \mathbf{T}_{\mathbf{x}y}\cos\alpha + \mathbf{T}_{\mathbf{y}y}\cos\beta + \mathbf{T}_{\mathbf{z}y}\cos\gamma$$
$$\mathbf{T}_{\mathbf{n}z} = \mathbf{T}_{\mathbf{x}z}\cos\alpha + \mathbf{T}_{\mathbf{y}z}\cos\beta + \mathbf{T}_{\mathbf{z}z}\cos\gamma$$

Here first subscript index shows normal to the face on which traction is exerted, and the second one - coordinate axis which gives rectilinear projection. So in order to calculate $\mathbf{T}_\mathbf{n}$, in addition to three angles α, β, γ we also need to know 9 components $\mathbf{T}_{\mathbf{x}x} \dots \mathbf{T}_{\mathbf{z}z}$ which form stress tensor. We will denote it $[\sigma]$ and will further work with it as with 3×3 matrix. This tensor is symmetric, but the proof will be omitted here due to its size, and it can be looked up in [1], chapter 2.

$$[\sigma] = \begin{bmatrix} \mathbf{T}_{\mathbf{x}x} & \mathbf{T}_{\mathbf{y}x} & \mathbf{T}_{\mathbf{z}x} \\ \mathbf{T}_{\mathbf{x}y} & \mathbf{T}_{\mathbf{y}y} & \mathbf{T}_{\mathbf{z}y} \\ \mathbf{T}_{\mathbf{x}z} & \mathbf{T}_{\mathbf{y}z} & \mathbf{T}_{\mathbf{z}z} \end{bmatrix}$$

Let us consider some volume of continuous medium, V, which is bounded by some imaginary closed surface with area S, and apply the d'Alabert's principle corollary which was already mentioned above around equation (4). Volumetric force acting on selected volume can be calculated as $\mathbf{F}\rho\,\mathrm{d}V$, and according inertial force $-\mathbf{a}\rho\,\mathrm{d}V$, the total vector sum of all the volumetric forces across considered volume will be a volume integral:

$$\iiint\limits_V (\mathbf{F} - \mathbf{a})\rho\,\mathrm{d}V$$

At the same time, small portion of imaginary surface $\mathrm{d}S$ suffers surface force $\mathbf{T}_\mathbf{n}\,\mathrm{d}S$, where $\mathbf{T}_\mathbf{n}$ - traction at a point with outward normal having α, β, γ angles to $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ co-ordintate axes. Thus, vector sum of all the surface forces will be a surface integral:

$$\iint\limits_S \mathbf{T}_\mathbf{n}\,\mathrm{d}S$$

And the vector sum of all the forces is zero-vector:

$$\iiint\limits_V (\mathbf{F} - \mathbf{a})\rho\,\mathrm{d}V + \iint\limits_S \mathbf{T}_\mathbf{n}\,\mathrm{d}S = \mathbf{0} \tag{7}$$

Using divergence theorem, we can transform surface integral into volume integral:

$$\begin{aligned} \iint\limits_S \mathbf{T}_\mathbf{n}\,\mathrm{d}S &= \iint\limits_S (\mathbf{T}_\mathbf{x}\cos\alpha + \mathbf{T}_\mathbf{y}\cos\beta + \mathbf{T}_\mathbf{z}\cos\gamma)\,\mathrm{d}S \\ &= \iiint\limits_V \left(\frac{\partial \mathbf{T}_\mathbf{x}}{\partial \mathbf{x}} + \frac{\partial \mathbf{T}_\mathbf{y}}{\partial \mathbf{y}} + \frac{\partial \mathbf{T}_\mathbf{z}}{\partial \mathbf{z}}\right)\mathrm{d}V \end{aligned} \tag{8}$$

Now substitute (8) into (7), we can get:

$$\iiint\limits_V \left(-\mathbf{a}\rho + \mathbf{F}\rho + \frac{\partial \mathbf{T}_\mathbf{x}}{\partial \mathbf{x}} + \frac{\partial \mathbf{T}_\mathbf{y}}{\partial \mathbf{y}} + \frac{\partial \mathbf{T}_\mathbf{z}}{\partial \mathbf{z}}\right)\mathrm{d}V = \mathbf{0} \tag{9}$$

Since this expression is true for arbitrary volume, the integrand then have to be zero in every point of this volume. And with some rearrangements, we get:

$$\mathbf{a}\rho = \mathbf{F}\rho + \frac{\partial \mathbf{T}_\mathbf{x}}{\partial \mathbf{x}} + \frac{\partial \mathbf{T}_\mathbf{y}}{\partial \mathbf{y}} + \frac{\partial \mathbf{T}_\mathbf{z}}{\partial \mathbf{z}} \tag{10}$$

And reformulating in projections:

$$\begin{aligned}
\mathbf{a}_x\rho &= \mathbf{F}_x\rho + \frac{\partial[\sigma]_{xx}}{\partial\mathbf{x}} + \frac{\partial[\sigma]_{yx}}{\partial\mathbf{y}} + \frac{\partial[\sigma]_{zx}}{\partial\mathbf{z}} \\
\mathbf{a}_y\rho &= \mathbf{F}_y\rho + \frac{\partial[\sigma]_{xy}}{\partial\mathbf{x}} + \frac{\partial[\sigma]_{yy}}{\partial\mathbf{y}} + \frac{\partial[\sigma]_{zy}}{\partial\mathbf{z}} \\
\mathbf{a}_z\rho &= \mathbf{F}_z\rho + \frac{\partial[\sigma]_{xz}}{\partial\mathbf{x}} + \frac{\partial[\sigma]_{yz}}{\partial\mathbf{y}} + \frac{\partial[\sigma]_{zz}}{\partial\mathbf{z}}
\end{aligned} \tag{11}$$

Therefore, equations of motion of continuous medium incorporate derivative of a stress tensor. Such equation is solved in each point of a volume that is being calculated when continuous medium (e.g. fluids) mechanics is simulated. In those cases additional formulae that express stress tensor through the *strain rate* tensor are used. For solids, stress tensor is expressed through the strain tensor, which will be described below. For viscoelastic simulations, stress tensor is expressed through a combination of strain and strain rate tensors.

In this article however, we will apply further simplifications, and will approximate continuous medium with finite elements. But it is important to note, that all of the simplifications are based on the model, described these chapters.

1.3. Hooke's law

To complete the continuum mechanics model for solids, we need to discover a connection between stress and strain. Let us consider some deformed body, and suppose that the deformation is changed in such a way that the displacement vector $\mathbf{u}_i$ (a function of coordinates of the undeformed body) changes by a small amount $\delta\mathbf{u}_i$ (also a function of coordinates), this causes changes in surfaces forces that strive towards keeping body's undeformed state; and let us determine the work done by the internal stresses in this change. From (10) we can deduce integrating over the volume of the body, that internal stress forces can be expressed as $F = \frac{\partial\mathbf{T_x}}{\partial\mathbf{x}} + \frac{\partial\mathbf{T_y}}{\partial\mathbf{y}} + \frac{\partial\mathbf{T_z}}{\partial\mathbf{z}}$. Work can be calculated as volume integral of elementary work done by the internal stress per unit volume δR (which is a scalar product of force and displacement):

$$\begin{aligned}
\iiint\limits_V \delta R\,\mathrm{d}V = \iiint\limits_V \Bigg(& \left(\frac{\partial[\sigma]_{xx}}{\partial\mathbf{x}} + \frac{\partial[\sigma]_{yx}}{\partial\mathbf{y}} + \frac{\partial[\sigma]_{zx}}{\partial\mathbf{z}}\right)\delta\mathbf{u}_x + \\
& \left(\frac{\partial[\sigma]_{xy}}{\partial\mathbf{x}} + \frac{\partial[\sigma]_{yy}}{\partial\mathbf{y}} + \frac{\partial[\sigma]_{zy}}{\partial\mathbf{z}}\right)\delta\mathbf{u}_y + \\
& \left(\frac{\partial[\sigma]_{xz}}{\partial\mathbf{x}} + \frac{\partial[\sigma]_{yz}}{\partial\mathbf{y}} + \frac{\partial[\sigma]_{zz}}{\partial\mathbf{z}}\right)\delta\mathbf{u}_z \Bigg)\,\mathrm{d}V
\end{aligned} \tag{12}$$

We integrate by parts and use divergence theorem, obtaining

$$\begin{aligned}
\iiint\limits_V \delta R\,\mathrm{d}V = \iint\limits_S \Bigg(& ([\sigma]_{xx}\delta\mathbf{u}_x + [\sigma]_{xy}\delta\mathbf{u}_y + [\sigma]_{xz}\delta\mathbf{u}_z)\mathbf{n}_x \; + \\
& ([\sigma]_{yx}\delta\mathbf{u}_x + [\sigma]_{yy}\delta\mathbf{u}_y + [\sigma]_{yz}\delta\mathbf{u}_z)\mathbf{n}_y \; + \\
& ([\sigma]_{zx}\delta\mathbf{u}_x + [\sigma]_{zy}\delta\mathbf{u}_y + [\sigma]_{zz}\delta\mathbf{u}_z)\mathbf{n}_z \Bigg)\,\mathrm{d}S - \\
\iiint\limits_V \Bigg(& [\sigma]_{xx}\frac{\partial\delta\mathbf{u}_x}{\partial\mathbf{x}} + [\sigma]_{xy}\frac{\partial\delta\mathbf{u}_x}{\partial\mathbf{y}} + [\sigma]_{xz}\frac{\partial\delta\mathbf{u}_x}{\partial\mathbf{z}} + \\
& [\sigma]_{yx}\frac{\partial\delta\mathbf{u}_y}{\partial\mathbf{x}} + [\sigma]_{yy}\frac{\partial\delta\mathbf{u}_y}{\partial\mathbf{y}} + [\sigma]_{yz}\frac{\partial\delta\mathbf{u}_y}{\partial\mathbf{z}} + \\
& [\sigma]_{zx}\frac{\partial\delta\mathbf{u}_z}{\partial\mathbf{x}} + [\sigma]_{zy}\frac{\partial\delta\mathbf{u}_z}{\partial\mathbf{y}} + [\sigma]_{zz}\frac{\partial\delta\mathbf{u}_z}{\partial\mathbf{z}} \Bigg)\,\mathrm{d}V
\end{aligned} \tag{13}$$

By considering an infinite medium which is not deformed at infinity, we make the surface of integration in the first integral tend to infinity; then stress tensor components $[\sigma]_{ij} = 0$ on the surface, and the integral is zero. By introducing

$$e = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix}$$

we can simplify some of the formulae below. The second integral can, by virtue of the symmetry of the stress tensor, be written

$$\begin{aligned}
&\iiint_V \delta R \, \mathrm{d}V = -\frac{1}{2} \iiint_V \Bigg(\\
&[\sigma]_{xx} \left(\frac{\partial \delta \mathbf{u}_x}{\partial \mathbf{x}} + \frac{\partial \delta \mathbf{u}_x}{\partial \mathbf{x}} \right) + [\sigma]_{xy} \left(\frac{\partial \delta \mathbf{u}_x}{\partial \mathbf{y}} + \frac{\partial \delta \mathbf{u}_y}{\partial \mathbf{x}} \right) + [\sigma]_{xz} \left(\frac{\partial \delta \mathbf{u}_x}{\partial \mathbf{z}} + \frac{\partial \delta \mathbf{u}_z}{\partial \mathbf{x}} \right) + \\
&[\sigma]_{yx} \left(\frac{\partial \delta \mathbf{u}_y}{\partial \mathbf{x}} + \frac{\partial \delta \mathbf{u}_x}{\partial \mathbf{y}} \right) + [\sigma]_{yy} \left(\frac{\partial \delta \mathbf{u}_y}{\partial \mathbf{y}} + \frac{\partial \delta \mathbf{u}_y}{\partial \mathbf{y}} \right) + [\sigma]_{yz} \left(\frac{\partial \delta \mathbf{u}_y}{\partial \mathbf{z}} + \frac{\partial \delta \mathbf{u}_z}{\partial \mathbf{y}} \right) + \\
&[\sigma]_{zx} \left(\frac{\partial \delta \mathbf{u}_z}{\partial \mathbf{x}} + \frac{\partial \delta \mathbf{u}_x}{\partial \mathbf{z}} \right) + [\sigma]_{zy} \left(\frac{\partial \delta \mathbf{u}_z}{\partial \mathbf{y}} + \frac{\partial \delta \mathbf{u}_y}{\partial \mathbf{z}} \right) + [\sigma]_{zz} \left(\frac{\partial \delta \mathbf{u}_z}{\partial \mathbf{z}} + \frac{\partial \delta \mathbf{u}_z}{\partial \mathbf{z}} \right) \\
&\Bigg) \, \mathrm{d}V
\end{aligned}$$

$$\begin{aligned}
&= -\frac{1}{2} \iiint_V \sum_{i=1}^{3} \sum_{j=1}^{3} \left([\sigma]_{ij} \left(\frac{\partial \delta \mathbf{u}_i}{\partial \mathbf{e}_j} + \frac{\partial \delta \mathbf{u}_j}{\partial \mathbf{e}_i} \right) \right) \mathrm{d}V \\
&= -\frac{1}{2} \iiint_V \sum_{i=1}^{3} \sum_{j=1}^{3} \left([\sigma]_{ij} \delta \left(\frac{\partial \mathbf{u}_i}{\partial \mathbf{e}_j} + \frac{\partial \mathbf{u}_j}{\partial \mathbf{e}_i} \right) \right) \mathrm{d}V \\
&= -\iiint_V \sum_{i=1}^{3} \sum_{j=1}^{3} \left([\sigma]_{ij} \delta [u]_{ij} \right) \mathrm{d}V
\end{aligned} \tag{14}$$

In (14) $[u]$ - strain tensor and $\delta[u]$ is a change of strain tensor. Thus, we can express δR:

$$\delta R = \sum_{i=1}^{3} \sum_{j=1}^{3} [\sigma]_{ij} \delta [u]_{ij} \tag{15}$$

This formula gives the work δR in terms of change in the strain tensor, as a sum of component-wise multiplication of it and stress tensor. For a deformation from undeformed state (where the initial strain tensor is zero), then $\delta[u] = [u]$ and δR is just a sum of component-wise multiplication of stress and strain tensors at a given point.

If the deformation of the body is fairly small, it returns to its original undeformed state when the external forces causing the deformation cease to act. Such deformations are said to be *elastic*. For large deformations, the removal of the external forces does not result in the total disappearance of the deformation; a *residual deformation* remains, so that the state of the body is not that which existed before the forces were applied. Such deformations are said to be *plastic*, and this kind of deformation is defined later in the document. In this section, all the deformations will be considered elastic.

We will also suppose that the process of deformations occurs so slowly that thermodynamic equilibrium is established in the body at every instant, in accordance with the external conditions. The process will then be thermodynamically reversible.

In what follows we will take all such thermodynamic quantities as entropy S, the internal energy U, etc., relative to the unit volume of the body, and not relative to unit mass as in fluid mechanics. The following remark should be made. Strictly speaking, the unit volumes before and after the deformation should be distinguished, since they in general contain different amount of matter. We will always relate the thermodynamic quantities to unit volume of the undeformed body, i.e. to the amount of matter therein, which may occupy a different volume after the deformation. Accordingly, the total energy of the body, for example, is obtained by integrating U over the volume of the undeformed body.

An infinitesimal change $\mathrm{d}U$ in the internal energy is equal to the difference between the heat acquired by the unit volume considered and the work $\mathrm{d}R$ done by the internal stresses. The amount of heat is, for a reversible process, $T\,\mathrm{d}S$, where T is the temperature. Thus $\mathrm{d}U = T\,\mathrm{d}S - \mathrm{d}R$; with $\mathrm{d}R$ given by (15), we obtain:

$$\mathrm{d}U = T\,\mathrm{d}S - \sum_{i=1}^{3}\sum_{j=1}^{3}[\sigma]_{ij}\,\mathrm{d}[u]_{ij}$$

This is the thermodynamic identity for deformed bodies.

Introducing the free energy of the body, $F = U - T \cdot S$, we find the form

$$\mathrm{d}F = -S\,\mathrm{d}T + \sum_{i=1}^{3}\sum_{j=1}^{3}[\sigma]_{ij}\,\mathrm{d}[u]_{ij} \tag{16}$$

Of the thermodynamic identity. T and u_{ij} are independent variables, thus the components of the stress tensor can be obtained by differentiating F with respect to the components of the strain tensor, for constant T:

$$[\sigma]_{ij} = \left(\frac{\partial F}{\partial [u]_{ij}}\right)_{T=const} \tag{17}$$

In order to be able to apply the general formulae of thermodynamics to any particular case, we must know the free energy F of the body as a function of the strain tensor. This expression is easily obtained by using the fact that the deformation is small and expanding the free energy in powers of $[u]_{ij}$. We will at present consider only *isotropic bodies*, i.e. bodies that deform equally independent of direction.

In considering a deformed body at some temperature (constant throughout the body), we will take the undeformed state to be the state of the body in the absence of external forces and at the same temperature; this last condition is necessary on account of the thermal expansions. Then for zero strain ($[u]_{ij} = 0$), the internal stresses are zero also ($[\sigma]_{ij} = 0$). From (17) it follows that there is no linear term in the expansion of F in powers of $[u]_{ij}$. Next, since the free energy is a scalar, each term in the expansion of F must be a scalar also. Two independent scalars of the second degree can be formed from the components of the symmetrical tensor $[u]_{ij}$: they can be taken as the squared sum of the diagonal components, and the sum of the squares of all the components. Expanding F in powers of $[u]_{ij}$, we therefore have as far as terms of the second order:

$$F = F_0 + \frac{1}{2}\lambda\left(\sum_{i=1}^{3}[u]_{ii}\right)^2 + \mu\sum_{i=1}^{3}\sum_{j=1}^{3}[u]_{ij}^2 \tag{18}$$

This is the general expression for the free energy of a deformed isotropic body. The quantities λ and μ are called *Lamé coefficients*.

We have seen in §1.1 that the change in volume in the deformation is given by the sum of diagonal components of the strain tensor. If this sum is zero, then the volume of the body is unchanged by the deformation, only its shape being altered. Such a deformation is called a *pure shear*. The opposite case is that of a deformation which causes a change in the volume of the body but no change in its shape. Each volume element of the body retains its shape also. Such a deformation is called *hydrostatic compression*. We will see below that every deformation could be represented as a combination of pure shear and hydrostatic compression:

$$[u] = \left([u] - \frac{1}{3}\operatorname{tr}([u])[I]\right) + \frac{1}{3}\operatorname{tr}([u])[I] \tag{19}$$

Where $[I]$ - is identity matrix and $\operatorname{tr}([u])$ - is a sum of diagonal components of strain tensor, its trace. The first term on the right is evidently a pure shear, since the sum of its diagonal terms is zero. The second term is hydrostatic compression.

As a general expression for the free energy of a deformed isotropic body, it is convenient to replace (18) by another formula, using this decomposition of an arbitrary deformation into a pure shear and hydrostatic compression. For the sake of brevity, the free energy of undeformed body (F_0) will be omitted (taking F to be only the free energy of the deformation, the elastic free energy, as it is called):

$$F = \frac{1}{2}K\operatorname{tr}^2([u]) + \mu\sum_{i=1}^{3}\sum_{j=1}^{3}\left([u]_{ij} - \frac{1}{3}\operatorname{tr}([u])[I]\right)^2 \tag{20}$$

The quantities K and μ are called respectively the *modulus of hydrostatic compression* (or simply *modulus of compression*) and the *modulus of rigidity*, K is related to the Lamé coefficients by $K = \lambda + \frac{2}{3}\mu$. Both K and μ are positive numbers.

We now use the general thermodynamic relation (16) to determine the stress tensor. To calculate the derivatives $\partial F/\partial[u]_{ij}$, we write the total differential $\mathrm{d}F$ (for constant temperature):

$$\mathrm{d}F = K\operatorname{tr}([u]\circ\mathrm{d}[u]) + 2\mu\sum_{i=1}^{3}\sum_{j=1}^{3}\left(\left[[u]_{ij} - \frac{1}{3}\operatorname{tr}([u])[I]\right]\mathrm{d}\left[[u]_{ij} - \frac{1}{3}\operatorname{tr}([u])[I]\right]\right)$$

Where $\circ$ is a Hadamard product (element-wise multiplication of matrices). In the second term, multiplication of the first parenthesis by $[I]$ gives zero, and in combination with $\operatorname{tr}(\mathrm{d}[u]) = \sum_{i=1}^{3}\sum_{j=1}^{3}([I]\circ\mathrm{d}[u])_{ij}$, we get:

$$\mathrm{d}F = \sum_{i=1}^{3}\sum_{j=1}^{3}\left([K\operatorname{tr}([u])[I]_{ij} + 2\mu([u]_{ij} - \frac{1}{3}\operatorname{tr}([u])[I]_{ij})]\,\mathrm{d}[u]_{ij}\right)$$

Hence the stress tensor is

$$[\sigma] = K\operatorname{tr}([u])[I] + 2\mu\left([u] - \frac{1}{3}\operatorname{tr}([u])[I]\right) \tag{21}$$

This expression determines the stress tensor in terms of the strain tensor for an isotropic body. It shows, in particular, that, if the deformation is a pure shear or a pure hydrostatic compression, the relation between $[\sigma]_{ij}$ and $[u]_{ij}$ is determined only by the modulus of rigidity or of hydrostatic compression respectively.

We will hereinafter apply *Voigt notation* (see [2]) - a way to represent a symmetrical tensor by reducing its order. Since $[\sigma]$ and $[u]$ are symmetric, both have only 6 independent components:

$$\widetilde{[\sigma]} = ([\sigma]_{xx}, [\sigma]_{yy}, [\sigma]_{zz}, [\sigma]_{yz}, [\sigma]_{xz}, [\sigma]_{xy})$$
$$\widetilde{[u]} = ([u]_{xx}, [u]_{yy}, [u]_{zz}, [\epsilon]_{yz}, [\epsilon]_{xz}, [\epsilon]_{xy})$$

Where $[\epsilon]_{yz} = 2[u]_{yz}$, $[\epsilon]_{xz} = 2[u]_{xz}$, $[\epsilon]_{xy} = 2[u]_{xy}$ - are *engineering shear strains*. The benefit of using different representations for stress and stain is that scalar product of these tensors in Voigt notation is equal to the matrix product of the original tensors:

$$[\sigma][u] = \widetilde{[\sigma]} \cdot \widetilde{[u]}$$

The tilde will be further omitted, and we will use stress and strain tensors in Voigt notation (unless specifically mentioned otherwise). In this notation, we can express stress tensor via the strain tensor using single matrix multiplication:

$$[\sigma] = [D][u]$$

Where D is a 6×6 matrix:

$$[D] = \begin{pmatrix} 2\mu+\lambda & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & 2\mu+\lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & 2\mu+\lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{pmatrix}$$

Please note that non-diagonal elements of $[u]$ are transformed with additional multiplication by $\frac{1}{2}$ relative to the original equation (21) - this is required to compensate for doubling in the engineering shear strains in Voigt notation.

The *modulus of extension* or *Young's modulus*, E, defines a relationship between the stress along an axis and the strain along that axis in a material. The *Poisson's ratio*, ν, is the ratio of transverse compression to the longitudinal extension. Both of these coefficients can be derived from the Lamé coefficients. Since they are widely used in the engineering field, we will express matrix D using the following:

$$\lambda = \frac{\nu E}{(1-2\nu)(1+\nu)}$$
$$\mu = \frac{E}{2(1+\nu)}$$

For the sake of clarity, temporarily introduce $N = \frac{\nu}{1-\nu}$:

$$[D] = \frac{E}{(1+\nu)(1-N)} \begin{pmatrix} 1 & N & N & 0 & 0 & 0 \\ N & 1 & N & 0 & 0 & 0 \\ N & N & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}(1-N) & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}(1-N) & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}(1-N) \end{pmatrix}$$

Young's modulus determines how stiff the body is, is a positive value, and the unit of measurement coincide with the pressure units, Pascals. Practical units for solids are gigapascals (GPa).

Poisson's ratio shows how thinner the body will become upon extension, and for the most bodies lies within the range of $[0; 0.5]$, but some materials exist with negative Poisson's ratio. Hereinafter, we will consider bodies that have Possion's ratio within $(0; 0.5)$ range (i.e., excluding borders), this way we guarantee that $[D]$ is positive definite matrix.

Various forms of matrix $[D]$ exist (see [3]) which allow to simulate complex materials, but we will consider only isotropic bodies in this work.

Relation between strain and stress tensors, derived in this section, allows to solve equations of continuum mechanics. The model that is presented here is continuous - i.e. the model characterizes motion in each point of the medium. Further in this document, we will describe a numerical method of the model discretization, which will allow to approximate the solution in practical cases.

2. Implementation

2.1. Finite Element Method

The *finite element method* (FEM) is a numerical technique for finding approximate solutions to boundary value problems for partial differential equations. It subdivides a large problem into smaller, simpler parts that are called finite elements. The simple equations that model these finite elements are then assembled into a larger system of equations that models the entire problem. We will use this method to get an approximate solution to the solid body formulated as continuous medium simulation problem. After we apply FEM, we will only need to do the computations at certain points of the continuous medium (we *discretize* the mathematical model). These points are moving along with the medium, thus they are particles, distributed throughout the volume of a body; in FEM they are called nodes. Nodes appear at elements boundaries and serve as connectors that fasten elements together. Behavior of all the other points can be derived through the interpolation of quantities, calculated at the nodes. In order to do the interpolation, the volume of the body is divided into elements - thus the name of the method. In general case of FEM, elements can have different shape and/or size; it is important that each element should stay convex for any configuration of its nodes, should be an affine combination of its nodes, and the volume of the body should be divided into non-intersecting finite elements, that share nodes. I.e. it should be possible to parametrize the space of a finite element, using two coordinates for 2D-problem and three coordinates for 3D-problem. Typically, coordinates sweep range $[-1; 1]$ or $[0; 1]$. A tuple of parametric coordinates uniquely defines a point within the finite element.

If $\mathbf{p_0}, \mathbf{p_1}, \ldots, \mathbf{p_n}$ - coordinates of nodes of a finite element; $q^{(1)}, q^{(2)}, q^{(3)}$ - parametric coordinates, functions $N_0(q^{(1)}, q^{(2)}, q^{(3)}), N_1(q^{(1)}, q^{(2)}, q^{(3)}), \ldots, N_n(q^{(1)}, q^{(2)}, q^{(3)})$ - *interpolation (or shape) functions* of nodes $0, 1, \ldots, n$ respectively, then geometric coordinates $\mathbf{x} = x, y, z$ of a point with parametric coordinates $q^{(1)}, q^{(2)}, q^{(3)}$ can be determined by:

$$\mathbf{x} = N_0(q^{(1)}, q^{(2)}, q^{(3)})\mathbf{p_0} + N_1(q^{(1)}, q^{(2)}, q^{(3)})\mathbf{p_1} + \ldots + N_n(q^{(1)}, q^{(2)}, q^{(3)})\mathbf{p_n} \qquad (22)$$

And it can be seen that this is simply an interpolation, and in general case it can be nonlinear (e.g. quadratic) - which depends on the choice of interpolation functions. Examples of finite elements are triangles/quads (2D-problem) or tetrahedra (3D-problem).

In engineering, one instance of problem solution typically contains several types of finite elements (various shape functions), and have adaptive elements size - where more accuracy is needed, elements are of smaller size; choosing shape functions and sizes of elements - one of

the key aspects of applying the FEM to the engineering tasks. However in this article, we concentrate on applications to game development, hence we favor fast and automatic methods over accuracy.

Quadratic shape functions require solving non-linear systems of equations, thus we will only consider linear shape functions; also, we will only consider tetrahedral elements (for 3D-problems, in 2D-problems it should be triangles) - one of the reasons is that we require automatic finite element representation creation, and in case of using tetrahedra we can use *Delaunay triangulation (tetrahedralization)* of the body. For convex (in undeformed state) bodies one of the simple methods of doing Delaunay triangulation is projecting points of a body onto hyperboloid of a higher dimension (i.e. $z^2 = x^2 + y^2$ for 2D-body and $w^2 = x^2 + y^2 + z^2$ for 3D-body), building a convex hull for the projected body, and then projecting "downward" looking faces (for 2D-body its hull will be three dimensional, and the faces will be triangles; and for 3D-body its hull will be four dimensional and the faces will be tetrahedra) of the hull back to the space of the body's dimensionality. Resulting projected triangles (tetrahedra) will make triangulation (tetrahedralization).

However if a body is non-convex in undeformed state - this approach won't work, but other approaches exist (e.g. [6]), as well as already implemented in tools. It is important that these algorithms allow to manage level of detail, one can force smaller elements near the surface where additional accuracy required; bigger uniform spaces can be divided into bigger elements, since lower accuracy can be tolerated there.

In order to apply FEM to our problem, we first need to derive a way of interpolating stress and strain tensors within the body's volume. Important to note, that interpolating displacement vectors is straightforward - using (22), but instead of coordinates, displacement vector at each node should be interpolated:

$$\mathbf{u} = N_0(q^{(1)}, q^{(2)}, q^{(3)})\mathbf{u_0} + N_1(q^{(1)}, q^{(2)}, q^{(3)})\mathbf{u_1} + \ldots + N_n(q^{(1)}, q^{(2)}, q^{(3)})\mathbf{u_n} \qquad (23)$$

We, however, want to calculate strain tensor (2) - and since we use Voigt notation (**??**), we need to calculate it as a 6-vector with doubled no-diagonal elements (engineering shear strain, see section §1.3 for details). We need to express through nodal displacements tensor in the form:

$$\begin{bmatrix} \frac{\partial \mathbf{u}_x}{\partial x} \\ \frac{\partial \mathbf{u}_y}{\partial y} \\ \frac{\partial \mathbf{u}_z}{\partial z} \\ \frac{\partial \mathbf{u}_y}{\partial z} + \frac{\partial \mathbf{u}_z}{\partial y} \\ \frac{\partial \mathbf{u}_x}{\partial z} + \frac{\partial \mathbf{u}_z}{\partial x} \\ \frac{\partial \mathbf{u}_x}{\partial y} + \frac{\partial \mathbf{u}_y}{\partial x} \end{bmatrix}$$

At each point of the finite element volume. This can be done by differentiating (23) - we will get expression $[u] = [B](q^{(1)}, q^{(2)}, q^{(3)})(\mathbf{X} - \mathbf{X_0})$, where $[u]$ - strain tensor at a given point, $[B]$ - some matrix at this point (see below), and $\mathbf{X} - \mathbf{X_0}$ is a displacements in nodal coordinates of the finite element. For example, if finite element has tetrahedral shape function, both $\mathbf{X}$ and $\mathbf{X_0}$ will consist of 4 coordinate triple each. Matrix B is convenient to represent as a set of blocks, one block per each node. For tetrahedral example,

$$[B] = \begin{bmatrix} [B_0] & [B_1] & [B_2] & [B_3] \end{bmatrix}$$

Each block in 3D-problem has a dimension 6×3, as it transforms coordinates of nodes into

elements of strain tensor. Each block will have the following structure:

$$[B_i] = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_i}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_i}{\partial z} \\ 0 & \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} & 0 & \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & 0 \end{bmatrix}$$

This shows that we need to know derivatives with respect to geometric coordinates of shape functions of each of the finite element's nodes, and in general case they can be different at each point of the finite element. We can express these derivatives with respect to geometric coordinates through the derivatives with respect to parametric coordinates, using Jacobi matrix:

$$\begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{bmatrix} = ([J]^T)^{-1} \begin{bmatrix} \frac{\partial N_i}{\partial q^{(1)}} \\ \frac{\partial N_i}{\partial q^{(2)}} \\ \frac{\partial N_i}{\partial q^{(3)}} \end{bmatrix} \tag{24}$$

Where

$$[J] = \begin{bmatrix} \frac{\partial x}{\partial q^{(1)}} & \frac{\partial x}{\partial q^{(2)}} & \frac{\partial x}{\partial q^{(3)}} \\ \frac{\partial y}{\partial q^{(1)}} & \frac{\partial y}{\partial q^{(2)}} & \frac{\partial y}{\partial q^{(3)}} \\ \frac{\partial z}{\partial q^{(1)}} & \frac{\partial z}{\partial q^{(2)}} & \frac{\partial z}{\partial q^{(3)}} \end{bmatrix}$$

Elements of Jacobi matrix can be calculating by differentiating (22), and since node coordinates are independent of parametric coordinates, we get:

$$[J] = \sum_{i=1}^{4} \mathbf{p_i} \cdot \begin{bmatrix} \frac{\partial N_i}{\partial q^{(1)}} \\ \frac{\partial N_i}{\partial q^{(2)}} \\ \frac{\partial N_i}{\partial q^{(3)}} \end{bmatrix}$$

Derivatives of shape function with respect to parametric coordinates can be easily obtained from the shape functions expressions.

We have matrix $[B]$, which allows us to determine strain tensor at a point with given parametric coordinates of some finite element. Using derivations we've made in the previous section, we can calculate strain tenor at this point as $[\sigma](q^{(1)}, q^{(2)}, q^{(3)}) = [D][u](q^{(1)}, q^{(2)}, q^{(3)})$. We now apply principles of virtual displacements, and we take into account that, as seen in (15), work that is required to change strain tensor by $\delta[u]$ can be calculated in Voigt notation as $[\sigma] \cdot \delta[u]$. We consider that all of the volumetric forces (inertial force included) are acting on the nodes of finite element, we'll denote that vector of nodal forces as $\mathbf{F}_e$ - it has three elements for each node. Then, denoting nodal displacement as $\delta\mathbf{X}$, we can calculate work of nodal forces as $\mathbf{F}_e\delta\mathbf{X}$:

$$\mathbf{F}_e\delta\mathbf{X} - \iiint\limits_V \delta[u][\sigma]\,\mathrm{d}V = 0$$

But $\delta[u] = [B]\delta\mathbf{X}$ and $[\sigma] = [D][u] = [D][B](\mathbf{X} - \mathbf{X_0})$. We can isolate $\delta\mathbf{X}$, and since this equality should be true on each virtual displacement, $\delta\mathbf{X}$ can be canceled out. Then we get:

$$\mathbf{F}_e = \left(\iiint\limits_V [B]^T(q^{(1)}, q^{(2)}, q^{(3)})[D][B](q^{(1)}, q^{(2)}, q^{(3)})\,\mathrm{d}V \right) (\mathbf{X} - \mathbf{X}_0)$$

This is a general equality for the FEM applied to the continuous mechanics, it expresses external forces through nodewise displacements.

$$[k] = \iiint_V [B]^T [D][B] \, \mathrm{d}V \tag{25}$$

Is identified as an expression for the *element stiffness matrix* for a given finite element. To calculate this matrix, we need to integrate over the volume. This can be done via parametric coordinates $q^{(1)}, q^{(2)}, q^{(3)}$, taking into account that volumetric element can be expressed through differentials $\mathrm{d}q^{(1)}, \mathrm{d}q^{(2)}, \mathrm{d}q^{(3)}$: $\mathrm{d}V = |J| \, \mathrm{d}q^{(1)} \, \mathrm{d}q^{(2)} \, \mathrm{d}q^{(3)}$, where $|J|$ is a determinant of Jacobi matrix $[J]$ that we derived above. Typically this integration is done numerically. But further in this article we will show how to avoid such integration, hence we don't look into this any further.

Next step after calculating matrix $[k]$ would be to calculate matrix $[K]$ - which is a *global stiffness matrix* for the whole body, which can be used to transform nodal displacements into nodal external forces for the whole body. We can derive matrix $[K]$ in a following way. $[k]$ for an element is a matrix of $3n \times 3n$ elements, where n - is a number of nodes in a finite element, for tetrahedral element, $n = 4$. This matrix can be considered a matrix of $n \times n$ blocks, each block has dimensionality of 3×3. Each such block transforms nodal displacements of a given node into a component of nodal force of other node (or this node itself) of that finite element. Nodal forces of a node, that belongs to several finite elements simultaneously, are affected by respective blocks of stiffness matrices of all these elements, and the forces are trivially summed.

Therefore, in order to get global stiffness matrix $[K]$, we need to blockwise sum matrices $[k]$ of all the finite elements, i.e. if some finite element contains nodes with indices i and j, then block of matrix $[k]$ of this finite element (transforming nodal displacement of i-th node into forces of j-th node) should be summed up with all the blocks of all the matrices $[k]$ which transform nodal displacements of i-th node into forces of j-th node.

Now, having assembled the global stiffness matrix $[K]$, and taking into account that $[F_e]$ now denote all the external forces acting on the body's nodes, including inertial force, we can apply d'Alembert's principle to get:

$$\begin{aligned} \mathbf{F} - [M]\mathbf{a} = \quad & \mathbf{F_e} = [K](\mathbf{X} - \mathbf{X_0}) \\ [M]\mathbf{a} = \quad & \mathbf{F} - [K](\mathbf{X} - \mathbf{X_0}) \end{aligned} \tag{26}$$

This is the equation of motion of body's nodes. Using this formula, we can integrate deformable body (represented by nodal particles) simulations into a computer simulation. Further in the article we suggest a particular way of solving this equation, as well as some additions to the model. For an in-depth look into FEM, please refer to [7].

2.2. Suggested FEM Model for Tetrahedral FE

Our suggestion is to use tetrahedral elements, as in this case a lot of convenient simplifications could be made.

For tetrahedral element, some point within the tetrahedron $OABC$ is given by $\mathbf{x} = \mathbf{O} + (\mathbf{A} - \mathbf{O})q^{(1)} + (\mathbf{B} - \mathbf{O})q^{(2)} + (\mathbf{C} - \mathbf{O})q^{(3)}$. Thus shape functions will look like this:

$$\begin{aligned} N_O &= 1 - q^{(1)} - q^{(2)} - q^{(3)} \\ N_A &= q^{(1)} \\ N_B &= q^{(2)} \\ N_C &= q^{(3)} \end{aligned}$$

Derivatives w.r.t. parametric coordinates of such shape functions are constant:

$$\begin{bmatrix} \frac{\partial N_O}{\partial q^{(1)}} & \frac{\partial N_O}{\partial q^{(2)}} & \frac{\partial N_O}{\partial q^{(3)}} \\ \frac{\partial N_A}{\partial q^{(1)}} & \frac{\partial N_A}{\partial q^{(2)}} & \frac{\partial N_A}{\partial q^{(3)}} \\ \frac{\partial N_B}{\partial q^{(1)}} & \frac{\partial N_B}{\partial q^{(2)}} & \frac{\partial N_B}{\partial q^{(3)}} \\ \frac{\partial N_C}{\partial q^{(1)}} & \frac{\partial N_C}{\partial q^{(2)}} & \frac{\partial N_C}{\partial q^{(3)}} \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{27}$$

And Jacobi matrix will look like:

$$[J] = \begin{bmatrix} \mathbf{A}-\mathbf{O} & \mathbf{B}-\mathbf{O} & \mathbf{C}-\mathbf{O} \end{bmatrix}$$

Matrix $([J]^T)^{-1}$ can be calculated analytically, and then shape function derivative w.r.t geometric coordinates will just be first column of $([J]^T)^{-1}$ for N_A, second column for N_B, and third column for N_C, while being equal to negative of sum of all three columns for N_O. This is a corollary of (24) and (27). Analytical form of those derivative will look like:

$$\begin{aligned} \Delta = \quad & (\mathbf{A}_x - \mathbf{O}_x)((\mathbf{B}_y - \mathbf{O}_y)(\mathbf{C}_z - \mathbf{O}_z) - (\mathbf{C}_y - \mathbf{O}_y)(\mathbf{B}_z - \mathbf{O}_z)) + \\ & (\mathbf{B}_x - \mathbf{O}_x)((\mathbf{C}_y - \mathbf{O}_y)(\mathbf{A}_z - \mathbf{O}_z) - (\mathbf{A}_y - \mathbf{O}_y)(\mathbf{C}_z - \mathbf{O}_z)) + \\ & (\mathbf{C}_x - \mathbf{O}_x)((\mathbf{A}_y - \mathbf{O}_y)(\mathbf{B}_z - \mathbf{O}_z) - (\mathbf{B}_y - \mathbf{O}_y)(\mathbf{A}_z - \mathbf{O}_z)) \end{aligned}$$

For the sake of clarity, temporarily introduce $\mathbf{OA} = \mathbf{A} - \mathbf{O}$, $\mathbf{OB} = \mathbf{B} - \mathbf{O}$, $\mathbf{OC} = \mathbf{C} - \mathbf{O}$:

$$\begin{aligned} \Delta = \quad & \mathbf{OA}_x(\mathbf{OB}_y\mathbf{OC}_z - \mathbf{OC}_y\mathbf{OB}_z) + \\ & \mathbf{OB}_x(\mathbf{OC}_y\mathbf{OA}_z - \mathbf{OA}_y\mathbf{OC}_z) + \\ & \mathbf{OC}_x(\mathbf{OA}_y\mathbf{OB}_z - \mathbf{OB}_y\mathbf{OA}_z) \end{aligned}$$

$$\begin{aligned} \frac{\partial N_A}{\partial x} &= \frac{1}{\Delta}(\mathbf{OB}_y\mathbf{OC}_z - \mathbf{OC}_y\mathbf{OB}_z) \\ \frac{\partial N_A}{\partial y} &= \frac{1}{\Delta}(\mathbf{OC}_x\mathbf{OB}_z - \mathbf{OB}_x\mathbf{OC}_z) \\ \frac{\partial N_A}{\partial z} &= \frac{1}{\Delta}(\mathbf{OB}_x\mathbf{OC}_y - \mathbf{OC}_x\mathbf{OB}_y) \\ \frac{\partial N_B}{\partial x} &= \frac{1}{\Delta}(\mathbf{OC}_y\mathbf{OA}_z - \mathbf{OA}_y\mathbf{OC}_z) \\ \frac{\partial N_B}{\partial y} &= \frac{1}{\Delta}(\mathbf{OA}_x\mathbf{OC}_z - \mathbf{OC}_x\mathbf{OA}_z) \\ \frac{\partial N_B}{\partial z} &= \frac{1}{\Delta}(\mathbf{OC}_x\mathbf{OA}_y - \mathbf{OA}_x\mathbf{OC}_y) \\ \frac{\partial N_C}{\partial x} &= \frac{1}{\Delta}(\mathbf{OA}_y\mathbf{OB}_z - \mathbf{OB}_y\mathbf{OA}_z) \\ \frac{\partial N_C}{\partial y} &= \frac{1}{\Delta}(\mathbf{OB}_x\mathbf{OA}_z - \mathbf{OA}_x\mathbf{OB}_z) \\ \frac{\partial N_C}{\partial z} &= \frac{1}{\Delta}(\mathbf{OA}_x\mathbf{OB}_y - \mathbf{OB}_x\mathbf{OA}_y) \\ \frac{\partial N_O}{\partial x} &= -\left(\frac{\partial N_A}{\partial x} + \frac{\partial N_B}{\partial x} + \frac{\partial N_C}{\partial x}\right) \\ \frac{\partial N_O}{\partial y} &= -\left(\frac{\partial N_A}{\partial y} + \frac{\partial N_B}{\partial y} + \frac{\partial N_C}{\partial y}\right) \\ \frac{\partial N_O}{\partial z} &= -\left(\frac{\partial N_A}{\partial z} + \frac{\partial N_B}{\partial z} + \frac{\partial N_C}{\partial z}\right) \end{aligned} \tag{28}$$

And this allows us to analytically calculate matrix $[B]$ for tetrahedral element:

$$\begin{bmatrix} \frac{\partial N_O}{\partial x} & 0 & 0 & \frac{\partial N_A}{\partial x} & 0 & 0 & \frac{\partial N_B}{\partial x} & 0 & 0 & \frac{\partial N_C}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_O}{\partial y} & 0 & 0 & \frac{\partial N_A}{\partial y} & 0 & 0 & \frac{\partial N_B}{\partial y} & 0 & 0 & \frac{\partial N_C}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_O}{\partial z} & 0 & 0 & \frac{\partial N_A}{\partial z} & 0 & 0 & \frac{\partial N_B}{\partial z} & 0 & 0 & \frac{\partial N_C}{\partial z} \\ 0 & \frac{\partial N_O}{\partial z} & \frac{\partial N_O}{\partial y} & 0 & \frac{\partial N_A}{\partial z} & \frac{\partial N_A}{\partial y} & 0 & \frac{\partial N_B}{\partial z} & \frac{\partial N_B}{\partial y} & 0 & \frac{\partial N_C}{\partial z} & \frac{\partial N_C}{\partial y} \\ \frac{\partial N_O}{\partial z} & 0 & \frac{\partial N_O}{\partial x} & \frac{\partial N_A}{\partial z} & 0 & \frac{\partial N_A}{\partial x} & \frac{\partial N_B}{\partial z} & 0 & \frac{\partial N_B}{\partial x} & \frac{\partial N_C}{\partial z} & 0 & \frac{\partial N_C}{\partial x} \\ \frac{\partial N_O}{\partial y} & \frac{\partial N_O}{\partial x} & 0 & \frac{\partial N_A}{\partial y} & \frac{\partial N_A}{\partial x} & 0 & \frac{\partial N_B}{\partial y} & \frac{\partial N_B}{\partial x} & 0 & \frac{\partial N_C}{\partial y} & \frac{\partial N_C}{\partial x} & 0 \end{bmatrix}$$

Important to note that this matrix is constant, i.e. same for all the points within the finite element! Thus, we can rewrite integral (25) as multiplication:

$$[k] = [B]^T[D][B]V \tag{29}$$

Where V is a volume of the finite element, $V = \frac{1}{6}\Delta$, where Δ - is the same Δ used in (28).

This expression for element stiffness matrix $[k]$ not only allows to conveniently calculate this matrix, but also decompose it into the product of several matrices. Using this decomposition, we can easily represent the global stiffness matrix, $[K]$. Formula will be similar to (29), but the whole-body matrix $[D_b]$ will be block-diagonal $6n \times 6n$ (where n - number of finite elements) matrix, and each block is a separate finite element matrix $[D]$. Whole-body matrix $[B_b]$ will be a sparse $6n \times 3k$ matrix, where each six rows will transform strain tensor of particular finite element from its nodal displacement, and k - is the number of nodes, i.e. each coordinate of each node within the body will get a column. But since we consider tetrahedral finite elements, each row will only have 12 non-zero values. Practically, each six rows of whole-body matrix $[B_b]$ - is a sparse representation of finite element matrix $[B] = [[B_0][B_1][B_2][B_3]]$, such that each block $[B_i]$ is positioned accordingly to the global index of node that belongs to the finite element.

This decomposition can be used in several ways (e.g. for convenient calculation of full-body matrix $[K]$), but we suggest using it to modify the classic constrained rigid-body dynamics simulation model. We will use (26) and will replace $\mathbf{a} = (\mathbf{v}_t - \mathbf{v}_{t-1})/\,\mathrm{d}t$ to get motion equation for the nodes of deformable body in velocities:

$$[M]\mathbf{v}_t = [M]\mathbf{v}_{t-1} + \mathrm{d}t\mathbf{F} - \mathrm{d}t[K](\mathbf{x}_t - \mathbf{x_0})$$

Approximating $[K](\mathbf{x}_t - \mathbf{x_0}) = [K](\mathrm{d}t\,\mathbf{v}_t + \mathbf{x}_{t-1} - \mathbf{x_0})$, we get:

$$([M] + [K])\mathbf{v}_t = [M]\mathbf{v}_{t-1} + \mathrm{d}t\mathbf{F} - \mathrm{d}t[K](\mathbf{x}_{t-1} - \mathbf{x_0}) \tag{30}$$

It is possible to solve equation in this form, and e.g. [8] does this. However, it is problematic to introduce joints into such system, since typically solution to the constrained rigid body dynamics problem requires expressing $\mathbf{v}_t$, which in this case means finding inverse $([K]+[M])^-1$. If we limit ourselves to particles which do not have rotation, the matrix $[M]$ is constant from frame to frame. Matrix $[K]$ also can be calculated only once for undeformed body, but we only considered very small deformations - and for larger deformations, and, what's even worse, for rotating bodies, using this model will lead to noticeable artifacts; and the way of fixing this problem includes modifying $[K]$ each frame (see detailed explanation in §2.3 further in the article). Thus pre-calculating the inverse of $[K] + [M]$ can work in a very limited case of particles (rigid bodies strictly without rotational component, while typical rigid body dynamics simulation engine of course uses rotational components) and very small deformations, without whole-body rotations.

To avoid the need of calculating the inverse, we can apply decomposition (29):

$$[M]\mathbf{v}_t = [M]\mathbf{v}_{t-1} + \mathrm{d}t\mathbf{F} - \mathrm{d}t[B]^T[D][B]V(\mathrm{d}t\,\mathbf{v}_t + \mathbf{x}_{t-1} - \mathbf{x_0}) \tag{31}$$

Introducing additional vector $\boldsymbol{\lambda}_0$:

$$\begin{cases} [M]\mathbf{v}_t &= [M]\mathbf{v}_{t-1} + \mathrm{d}t\mathbf{F} - \mathrm{d}t[B]^T\boldsymbol{\lambda}_0\,\mathrm{d}t, \\ \boldsymbol{\lambda}_0 &= -V[D][B](\mathrm{d}t\,\mathbf{v}_t + \mathbf{x}_{t-1} - \mathbf{x_0}) \end{cases} \tag{32}$$

The second equation in this system we can rewrite as:

$$[B]\mathbf{v}_t + \frac{1}{\mathrm{d}t}\frac{1}{V}[D]^{-1}\boldsymbol{\lambda}_0 = -\frac{1}{\mathrm{d}t}[B](\mathbf{x}_{t-1} - \mathbf{x_0}) \tag{33}$$

Equation (32) is basically a spring/damper joint equation (the one you typically get why derive constrained rigid body simulation equations with Baumgarte stabilization).. Jacobi matrix here is $[B]$, *regularization term* $[R_r] = \frac{1}{\mathrm{d}t}\frac{1}{V}[D]^{-1}$ and the *right hand side vector* $\mathbf{c} =$

$-\frac{1}{\mathrm{d}t}[B](\mathbf{x}_{t-1} - \mathbf{x_0})$. One peculiarity that should be taken into account, is that $[R_r]$ is not diagonal, but has 3 elements per row (it holds true for both $[D]$ and $[D]^{-1}$). But otherwise - we have a typical joint equation, 6 rows per finite element - which we can easily combine with other general joints. More generalized representation of this system:

$$\begin{cases} [M]\mathbf{v}_t &= [M]\mathbf{v}_{t-1} + \mathrm{d}t\mathbf{F} - \mathrm{d}t[J]^T\boldsymbol{\lambda}\,\mathrm{d}t, \\ [J]\mathbf{v}_t + [R_r]\boldsymbol{\lambda} &= \mathbf{c} \end{cases} \tag{34}$$

$[B]$ simply becomes integrated into the general system $[J]$ matrix, $\boldsymbol{\lambda}_0$ becomes a part of a general vector $\boldsymbol{\lambda}$, and terms $[R_r]$ and $\mathbf{c}$ become embedded into their generalized counterparts. This system then can be transformed using Schur complement:

$$\mathrm{d}t\left([J][M]^{-1}[J]^T + [R_r]\right)\boldsymbol{\lambda} = \mathbf{c} - \mathbf{v}_{t-1} - \mathrm{d}t[M]^{-1}\mathbf{F} \tag{35}$$

Some solve the system after mutual division of both sides by $\mathrm{d}t$ and or solving for $\mathrm{d}t\boldsymbol{\lambda}$, but this doesn't affect the results.

We treated non-diagonal $[R_r]$ by trivially modifying our solver to take that into account, but another approach exists: [10] suggest decomposing $[D]$ using spectral decomposition (since it is positive definite matrix, it is always possible): $[D] = [D'']^T[D'][D'']$, where $[D']$ would be a diagonal matrix. Then, $[K] = [B]^T[D][B]V$ we can rewrite as $[K] = [B']^T[D'][B']V$, where $[B'] = [D''][B]$, and next we can apply the same logic we applied previously, but using $[B']$ instead of $[B]$ and $[D']$ instead of $[D]$. This won't have any important consequences, although regularization term is now a diagonal matrix:

$$[R'_r] = \frac{1}{\mathrm{d}t}\frac{1}{V}[D']^{-1}$$

This allows to use FEM joints and regular constrained rigid body dynamics joints in one system. But this will require the solver to work with both rigid bodies with rotational components and particles without the rotation component (those particles represent nodes of a deformable body). This is not fundamentally a problem: typical rigid body could be represented as six coordinates in the solver (esp velocity, forces, etc.), and its mass is block-diagonal 6×6 matrix, which contains the scalar mass of the body and the inertia tensor for rotational component. Joints typically connect two bodies by constraining hextuples - velocities of bodies. Formulae for linear and rotational parts are very similar, except the mass part: inertia tensor needs to be rotated each timestep, while scalar mass is invariant. Formulae for particles are equivalent to formulae of linear component of constrained rigid body motion.

The actual solution - is to represent rigid body as two triples rather than one hextuple - decouple linear and rotational components. All of the formulae will be defined for a triple, and there will only be differences for the mass parts on the integration stage. Joints, connecting two bodies, will constrain 4 triples using 4 indices, for linear and rotational parts of each body, instead of two indices of hextuples. Thus on the solver level there will be no difference whether join connects two rigid bodies, or four particles.

However, similar joints (e.g. ball-in-socket joint) will be different based on whether they connect two rigid bodies, a rigid body with a finite element, or two finite elements together - Jacobi matrices and constraint equations will be different in those cases. In general case of connecting two finite elements together - three points from each finite element should be picked, which leads to the need of the solver supporting six triples, instead of four triples as in case of basic constrained rigid body simulation. But the basic principles stay the same, one can derive joint equations using $C(\mathbf{x}) = 0$ (where $C(\mathbf{x})$ is a *constraint function*) and the derivations of it. For surface forces acting on finite elements, we can assume that force acts on each of the nodes

proportional to the corresponding barycentric coordinates of an application point.

In this section we derived base FEM model that could be integrated into constrained rigid body simulation. It has several issues, that we plan to attend later in this article.

First of all, the model assumes small deformations, and, as already mentioned, shows incorrect behavior when the body is rotating as a whole. Indeed, looking at how $[B]$ is expressed, we can see that translation will not affect deformation of the body - if we add the same number to O, A, B, C - in expression $[B]\mathbf{q}$ parts, related to that added number will cancel out, for any $\mathbf{q}$. However, rotation of a body as a whole object will affect the strain tensor, despite the fact that distance between the points didn't really change and there shouldn't be any deformation/strain. Correct solution would be to use full strain tensor (2) instead of (3), but the equations the become non-linear. In the next section, we'll describe another approach of solving this issue, suggested by [9].

Second, all of the deformations, described so far, are elastic - i.e. body always returns to its original shape when external force stops acting on this body.

Third, the stability of described method in its naïve implementation is quite low. The model is basically somewhat enhanced spring; and with high stiffness of the spring (with increased Young's modulus), the behavior becomes unstable. This can be addressed by increasing the solution accuracy, and although the system has no damping, symplectic Euler integrator dissipates energy, and when solution is accurate enough, the system will be stable. But this often means incredibly high iteration count for Projected Gauss-Seidel solver, which makes the simulation non-real-time. Another way to increase simulation accuracy is to use solver with increased accuracy, such as Conjugate Gradients, which in its base form can only solve pure linear systems and not *Linear Complementarity Problems* (LCP), which makes it impossible to use e.g. contact constraints in the system. We will later introduce a modified version of Conjugate Gradient solver that can solve LCPs as part of the solution of this problem. We will also introduce damping into the system. But even with damping and more accurate solver, for highly stiff bodies, there is a trade-off between stability and speed.

2.3. Rotation Artifacts - Stiffness Warping

In this section we'll describe a method to eliminate artifacts that appear when the body is rotated as a whole; the method was suggested by [9] and is called *stiffness warping* (which was later improved in [8] by same authors). The model that we suggested in a previous section uses linearized form of a strain tensor (3), which can only describe small deformations. With large deformations, artifacts appear, which manifest themselves as strictly linear nodal displacements (nodes travel from their initial positions in a straight line). It is most noticeable when the body is rotated as a whole object - it will have unnatural deformations.

It is worth noting that translating the body doesn't lead to artifacts. This can be seen in (28). Since all of the derivatives of N_O w.r.t. coordinates have derivatives of all the other shape functions, taken with the negative sign, calculating strain tensor (by multiplying $[B]$ with nodal displacements $\mathbf{x} - \mathbf{x}_0$), it can easily be seen that after rearrangements, common displacement will be canceled out. For example, given that $\frac{\partial N_O}{\partial x} = -\left(\frac{\partial N_A}{\partial x} + \frac{\partial N_B}{\partial x} + \frac{\partial N_C}{\partial x}\right)$, for the first row of the matrix we can write:

$$
\begin{array}{rl}
& \left(\frac{\partial N_O}{\partial x}\ 0\ 0\ \frac{\partial N_A}{\partial x}\ 0\ 0\ \frac{\partial N_B}{\partial x}\ 0\ 0\ \frac{\partial N_C}{\partial x}\ 0\ 0\right) \circ \\
& (\mathrm{d}\mathbf{O}_x\ \mathrm{d}\mathbf{O}_y\ \mathrm{d}\mathbf{O}_z\ \mathrm{d}\mathbf{A}_x\ \mathrm{d}\mathbf{A}_y\ \mathrm{d}\mathbf{A}_z\ \mathrm{d}\mathbf{B}_x\ \mathrm{d}\mathbf{B}_y\ \mathrm{d}\mathbf{B}_z\ \mathrm{d}\mathbf{C}_x\ \mathrm{d}\mathbf{C}_y\ \mathrm{d}\mathbf{C}_z) \\
= & \frac{\partial N_O}{\partial x}\,\mathrm{d}\mathbf{O}_x + \frac{\partial N_A}{\partial x}\,\mathrm{d}\mathbf{A}_x + \frac{\partial N_B}{\partial x}\,\mathrm{d}\mathbf{B}_x + \frac{\partial N_C}{\partial x}\,\mathrm{d}\mathbf{C}_x \\
= & \frac{\partial N_A}{\partial x}\,(\mathrm{d}\mathbf{A}_x - \mathrm{d}\mathbf{O}_x) + \frac{\partial N_B}{\partial x}\,(\mathrm{d}\mathbf{B}_x - \mathrm{d}\mathbf{O}_x) + \frac{\partial N_C}{\partial x}\,(\mathrm{d}\mathbf{C}_x - \mathrm{d}\mathbf{O}_x)
\end{array}
$$

It can be seen, that if all of the nodes get displaced by the same vector, the first component of strain tensor will not change, as this vector will get canceled out. Similarly, other components of the strain tensor will not change too. This, the body can be translated as a whole without any artifacts.

Therefore, the biggest problem is rotation not taken into account in linearized tensor. To eliminate this problem, stiffness warping was suggested in [9], the main idea of this method is to remove the rotational component of the deformation. Since the model doesn't take rotation into account, nodal coordinates can be transformed so that the rotational component is removed; and then the elasticity model is applied to the rectified nodal coordinates; after that, result is being transformed back to the rotated state. Thus, stress only appear when rectified strain is non-zero. The method is applied to each individual finite element, which compensates not only for whole body rotation but also for body parts rotation, which is optimal to eliminate rotational artifacts.

We are using tetrahedral finite elements, and each tetrahedron defines basis in 3D space - one of its vertices we can consider an origin point, and three edges that connect this vertex with all other - basis vectors. We consider same tetrahedron in undeformed state (we'll denote it p) and in deformed state (we'll denote it q, not to be confused with barycentric coordinates). These tetrahedra (undeformed and deformed versions of the same tetrahedron) represent two different bases, and a transformation that transforms one basis into another can be expressed. Let $(q^{(1)}, q^{(2)}, q^{(3)})$ be barycentric coordinates of some point within the tetrahedron, then in undeformed tetrahedron this point will have coordinate expression $\mathbf{x}_p = \mathbf{O}_p + (\mathbf{A}_p - \mathbf{O}_p)q^{(1)} + (\mathbf{B}_p - \mathbf{O}_p)q^{(2)} + (\mathbf{C}_p - \mathbf{O}_p)q^{(3)}$; in deformed tetrahedron, this point will have coordinate expression $\mathbf{x}_q = \mathbf{O}_q + (\mathbf{A}_q - \mathbf{O}_q)q^{(1)} + (\mathbf{B}_q - \mathbf{O}_q)q^{(2)} + (\mathbf{C}_q - \mathbf{O}_q)q^{(3)}$. We'll introduce two matrices:

$$
\begin{aligned}
[P] &= (\mathbf{A}_p - \mathbf{O}_p \quad \mathbf{B}_p - \mathbf{O}_p \quad \mathbf{C}_p - \mathbf{O}_p) \\
[Q] &= (\mathbf{A}_q - \mathbf{O}_q \quad \mathbf{B}_q - \mathbf{O}_q \quad \mathbf{C}_q - \mathbf{O}_q)
\end{aligned}
$$

And we can rewrite

$$
\mathbf{x}_p = \mathbf{O}_p + [P] \begin{pmatrix} q^{(1)} \\ q^{(2)} \\ q^{(3)} \end{pmatrix}
$$

$$
\mathbf{x}_q = \mathbf{O}_q + [Q] \begin{pmatrix} q^{(1)} \\ q^{(2)} \\ q^{(3)} \end{pmatrix}
$$

We can express barycentric coordinates like this:

$$
\begin{pmatrix} q^{(1)} \\ q^{(2)} \\ q^{(3)} \end{pmatrix} = [P]^{-1}(\mathbf{x}_p - \mathbf{O}_p)
$$

And we can substitute this barycentric coordinates expression:

$$
\mathbf{x}_q = \mathbf{O}_q + [Q][P]^{-1}(\mathbf{x}_p - \mathbf{O}_p) = \mathbf{O}_q - [Q][P]^{-1}\mathbf{O}_p + [Q][P]^{-1}\mathbf{x}_p
$$

This is a transformation of some point in undeformed tetrahedron into a point in deformed tetrahedron. We need to extract rotational part of this transformation. The $\mathbf{O}_q - [Q][P]^{-1}\mathbf{O}_p$ part

of the expression we can omit, since it represents translational part, and $A = [Q][P]^{-1}$ in the remaining part represents shear and rotation. Many decompositions exist which produce orthogonal matrix and some other matrix, we will use *polar decomposition* as the resulting orthogonal matrix will be as close to the original matrix as possible, which gives polar decomposition advantage over e.g. QR-decomposition (see [11]). Polar decomposition $[A] = [U][P]$ can be calculated as

$$[P] = \sqrt{[A]^T[A]}$$

$$[U] = [A][P]^{-1}$$

(see [4]). We need $[U]$ - the orthogonal matrix. Matrix square root can be calculated iteratively, e.g. using *Denman–Beavers square root iteration* (see [5]). Important thing to note, that $[U]$ can contain reflection, which can lead to tetrahedra flipping inside out. To avoid this, we can check the rotation matrix determinant - if it will be negative one, matrix contains reflection too (pure rotation matrix will have determinant equal to positive one). Such matrix can be multiplied by -1, giving the pure rotation matrix we need.

Thus we get a rotation matrix, $[R]$ (not to be confused with regularization term $[R_r]$), which describes a rotation component of tetrahedron's transformation. Multiplying nodal coordinates by $[R]^{-1}$, we get deformed tetrahedron, and the deformation lacks rotational part. Its nodal coordinates can then be used to calculate nodal stress, which can be converted back to the original deformed state (the one with rotation) using matrix $[R]$. For a finite element, given that for rotational matrix $[R]^{-1} = [R]^T$, formula (2.1) can be rewritten as:

$$\mathbf{F} = [R][k][R]^T(\mathbf{x} - \mathbf{x_0}) \tag{36}$$

Problem is that matrix $[R]$ is different for each finite element, and different finite elements can share nodes - thus for each tetrahedron nodal coordinates of same node may be required to be transformed with different matrices $[R]$. Explicit computation of global stiffness matrix $[K]$ will require to store element stiffness matrices $[k]$ for each element, then translating it each time step similar to (36), and then assembling the global stiffness matrix.

Thankfully, the (29) decomposition allows us to solve this problem. Using this decomposition, we can rewrite (36) as:

$$\mathbf{F} = [R][B]^T[D][B]V[R]^T(\mathbf{x} - \mathbf{x_0}) \tag{37}$$

Matrix $[B]$ contains matrices $[B]$ for each individual finite element, six rows per finite element, and each six-row submatrix has only four 6×3 non-zero blocks, one block per finite element nodes. Thus, each such block in global $[B]$ affects only on one finite element, and we can transform this block without affecting other finite elements. For each six rows we can take $[R]$ calculated for respective finite element, and multiply each of four non-zero sub-blocks by $[R]^T$. To be more precise, since each such block is 6×3, we multiply each 3×3 sub-blocks by $[R]^T$. The result will be same as if nodal coordinates of a finite element were transformed by $[R]^T$ specifically to calculate strain tensor of this particular finite element, even if exactly same node is used to glue together several finite elements. We can also remember that $([M_A][M_B])^T = [M_B]^T[M_A]^T$, and thus we can introduce a modified matrix $[B_R] = [B][R]^T$, which we can substitute instead of $[B]$ throughout FEM model (35) to incorporate stiffness warping into the model.

Summarizing, to eliminate rotational artifacts, we need to calculate $[A] = [Q][P]^{-1}$ (matrix $[P]^{-1}$ which only depends on undeformed state, can be precomputed once); apply polar decomposition to get rotation matrix $[R]$; check its determinant, and multiply by -1 if needed to remove reflection; multiply $4 \cdot 2 = 8$ blocks of 3×3 elements of six-row $[B]$ section that

corresponds to the given finite element, by $[R]^T$. Polar decomposition is not cheap, but without this step, this method cannot be applied to bodies that are not attached to the world (or, in other words, to infinitely heavy object), or bodies that experience large transformation, leading to relative rotation of parts of this body. However, even with the polar decomposition, the method is still real-time.

2.4. Damping and Stability

In this section we will briefly describe how to introduce damping to the described system. As you might have already noticed (from §1.3), the primary formula that we derived FEM model for, looks like a basic linear spring law. Both are based on Hooke's law, but in linear spring ($F = -k \cdot \mathrm{d}x$) coefficient k is scalar, and in theory of elasticity equations we have matrix relation between displacement (in generalized theory of elasticity the analogue is strain tensor) and reaction (restoring) force (in theory of elasticity that would be the stress tensor). Similarly to full linear spring equation ($F = -k \cdot \mathrm{d}x - d \cdot v$), we can introduce damping into our system, which will allow us to control oscillation reduction within our system. Theoretically, in the model that we described previously, any interaction with the elastic body would lead to infinite oscillations; practically, however, symplectic Euler integration that we use dissipates energy, which subsequently leads to oscillation reduction, but at a rate that we don't necessarily control. Introduction of damping into this system will allow to speed up this process and make it more predictable; and, what's most important - improve the overall stability of the system. As it was already mentioned, with high stiffness, stability of the system drops, and it is required to increase accuracy of the solution in order to maintain plausible behavior. Damping allows to alleviate this problem, since systems with damping are generally more stable, however the problem of instability will not still be solved fundamentally.

To introduce damping into the system, we'll need to briefly repeat some of the steps that we performed earlier, but taking damping into account. First of all, we'll introduce strain rate tensor, which we briefly mentioned in §1.2. Strain rate tensor is a 3×3 tensor, that consists of partial velocity derivatives w.r.t. spatial coordinates:

$$[v]_{ik} = \frac{1}{2}\left(\frac{\partial \mathbf{v}_i}{\partial \mathbf{x}_k} + \frac{\partial \mathbf{v}_k}{\partial \mathbf{x}_i}\right) \tag{38}$$

This tensor looks a lot like strain tensor, but in this case we're interested not in displacements of a point in deformed body from its initial position in undeformed body, but the velocity of this point. Similarly to strain tensor, strain rate tensor can be represented using Voigt notation (described in §1.3). Since we're introducing damping into the system, stress tensor at any point of continuous medium will be expressed through not only strain tensor, but also through the strain rate tensor. We assume that the relation is linear as well, therefore (1.3) will have slightly different form:

$$[\sigma] = [D][u] + [C][v] \tag{39}$$

Where matrix $[C]$ is a 6×6 matrix (just as matrix $[D]$), which translates six components of the strain rate tensor into stress constituents. Matrix $[C]$ sets up a linear damping model for a given material.

This enhanced formula can be discretized using finite elements similarly as the original formula. Just as nodal displacements were interpolated in (23), we can interpolate nodal velocities:

$$\mathbf{v} = N_0(q^{(1)}, q^{(2)}, q^{(3)})\mathbf{v_0} + N_1(q^{(1)}, q^{(2)}, q^{(3)})\mathbf{v_1} + \ldots + N_n(q^{(1)}, q^{(2)}, q^{(3)})\mathbf{v_n} \tag{40}$$

Similarly to strain tensor, we can find interpolation within the volume of a finite element of strain rate tensor in Voigt notation: $[v] = [B](q^{(1)}, q^{(2)}, q^{(3)})\mathbf{V}$, where $\mathbf{V}$ contains four velocity

triples, triple per finite element node. Matrix $[B]$ will be exactly same as in case of strain tensor.

Knowing this, we can easily modify the fundamental equation of the FEM model that we use:

$$[\sigma](q^{(1)}, q^{(2)}, q^{(3)}) = [D][u](q^{(1)}, q^{(2)}, q^{(3)}) + [C][v](q^{(1)}, q^{(2)}, q^{(3)})$$

And we will again use principles of virtual displacements. Work that is required to change strain tensor by $\delta[u]$ can still be calculated in Voigt notation as $[\sigma] \cdot \delta[u]$. Previously we were assuming that volumetric forces act on nodes, we will assume this now too. Then , this equation still holds:

$$\mathbf{F}_e \delta\mathbf{X} - \iiint\limits_V \delta[u][\sigma] \,\mathrm{d}V = 0$$

And still $\delta[u] = [B]\delta\mathbf{X}$ - we can isolate $\delta\mathbf{X}$ and since this equality should hold for each virtual displacement $\delta\mathbf{X}$, it will be canceled out. Also, considering that

$$[\sigma] = [D][u] + [C][v] = [D][B](\mathbf{x} - \mathbf{x_0}) + [C][B]\mathbf{v}$$

We can write:

$$\begin{aligned} \mathbf{F}_e = & \iiint\limits_V [B]^T([D][B](\mathbf{X} - \mathbf{X}_0) + [C][B]\mathbf{V}) \,\mathrm{d}V \\ = & \iiint\limits_V [B]^T[D][B] \,\mathrm{d}V(\mathbf{X} - \mathbf{X}_0) + \iiint\limits_V [B]^T[C][B] \,\mathrm{d}V\mathbf{V} \end{aligned} \tag{41}$$

This is enhanced version of fundamental FEM equation, which introduces damping into the model. It can otherwise be rewritten as $\mathbf{F}_e = [K](\mathbf{x} - \mathbf{x}_0) + [Q]\mathbf{v}$, where $\mathbf{v}$ - nodal velocities of finite elements, $[Q] = \iiint\limits_V [B]^T[C][B] \,\mathrm{d}V$, $[C]$ - damping matrix, and all the other variables stay same. Practically, we just added $[Q]\mathbf{v}$ to the fundamental formula. The process of assembling matrix $[Q]$ is additive, similarly to the one of $[K]$. We can rewrite (26) as:

$$[M]\mathbf{a} = \mathbf{F} - [K](\mathbf{X} - \mathbf{X_0}) - [Q]\mathbf{V} \tag{42}$$

We can also apply simplifications mentioned in §2.2. Since matrix $[B]$ is the same, decomposition steps (29) are same for tetrahedral elements for matrix $[Q]$ too, as well as generalization to the global matrix:

$$[Q] = [B]^T[C][B]V \tag{43}$$

Global matrix $[B]$ is exactly same, global matrix $[C]$ is block-diagonal matrix, which is comprised of element $[C]$ matrices. Thus we can rewrite (31) as:

$$[M]\mathbf{v}_t = [M]\mathbf{v}_{t-1} + \mathrm{d}t\mathbf{F} - \mathrm{d}t[B]^T[D][B]V(\mathrm{d}t\,\mathbf{v}_t + \mathbf{x}_{t-1} - \mathbf{x_0}) - \mathrm{d}t[B]^T[C][B]V\mathbf{v}_t \tag{44}$$

We can also $\boldsymbol{\lambda}_0$ in (32):

$$\begin{cases} [M]\mathbf{v}_t &= [M]\mathbf{v}_{t-1} + \mathrm{d}t\mathbf{F} - \mathrm{d}t[B]^T\boldsymbol{\lambda}_0 \,\mathrm{d}t, \\ \boldsymbol{\lambda}_0 &= -V[D][B](\mathrm{d}t\,\mathbf{v}_t + \mathbf{x}_{t-1} - \mathbf{x_0}) - V[C][B]\mathbf{v}_t \end{cases} \tag{45}$$

And we can rewrite the second equation as:

$$\boldsymbol{\lambda}_0 = -V([D]\,\mathrm{d}t + [C])[B]\mathbf{v}_t - V[D][B](\mathbf{x}_{t-1} - \mathbf{x_0})$$

Or, similarly to (33):

$$[B]\mathbf{v}_t + \frac{1}{V}([D]\,\mathrm{d}t + [C])^{-1}\boldsymbol{\lambda}_0 = -([D]\,\mathrm{d}t + [C])^{-1}[D][B](\mathbf{x}_{t-1} - \mathbf{x_0}) \tag{46}$$

This is enhanced equation of finite element constraints that allows to take damping into account. No other changes required, everything else works just as it worked previously, we can apply all other optimizations, except diagonalization of regularization matrix $[R_r]$. If the timestep is fixed, this matrix can be calculated once.

We deliberately didn't suggest any particular form of $[C]$. Since we use damping to increase stability of the system, it would be sufficient to define $[C]$ as proportional to $[D]$: $[C] = \beta[D]$, where β is the only coefficient that defines damping in the system. Then we can rewrite:

$$([D]\,\mathrm{d}t + [C])^{-1} = ([D](\mathrm{d}t + \beta))^{-1} = \frac{1}{\mathrm{d}t + \beta}[D]^{-1} = \frac{1}{1 + \frac{\beta}{\mathrm{d}t}}\left(\frac{1}{\mathrm{d}t}[D]^{-1}\right)$$

And we can rewrite (46) as:

$$[B]\mathbf{v}_t + \frac{1}{1 + \frac{\beta}{\mathrm{d}t}}\frac{1}{\mathrm{d}t}\frac{1}{V}[D]^{-1}\boldsymbol{\lambda}_0 = -\frac{1}{1 + \frac{\beta}{\mathrm{d}t}}\frac{1}{\mathrm{d}t}[B](\mathbf{x}_{t-1} - \mathbf{x_0}) \tag{47}$$

Which has only slight differences from (33): it introduces multiplication of $[R_r]$ and the right hand side vector $\mathbf{c}$ by scalar $1/(1 + \beta/\,\mathrm{d}t)$. This can be done rather easily, and all the other formulae and principles (this time including $[R_r]$ diagonalization) hold. Damping, introduced in this form, handles stability problems quite efficiently, the only thing that is left to do - is to tweak β parameter; we advise to start from $\beta = 1$.

In this section, we introduced damping into the system. Mostly, this section was dedicated to theoretical formulations, but practically damping model that we used is empiric way to increase stability of the system. We didn't experiment with physically correct damping, but this could be done by starting from the generalized formula (46) and defining physically correct damping matrix $[C]$.

2.5. Plasticity

As it was already previously mentioned, the suggested model features purely elastic deformations, i.e. after external forces stop acting on a body, body will return to its original undeformed shape. But there is the second kind of deformation - plastic deformation, when some residual deformation exists, which internal stress forces do not try to eliminate. Adding this kind of deformations into the suggested model is relatively simple task.

We'll use plasticity model suggested by [12] and [8]. Main equation of this model is decomposition of a total strain tensor (in Voigt notation) into a sum of two tensors, elastic strain tensor and plastic strain tensor: $[u_{total}] = [u_{elastic}] + [u_{plastic}]$. Plastic strain tensor represents that part of the deformation that the body doesn't try to eliminate with internal forces; therefore, the reaction forces only expressed via $[u_{elastic}]$. At the same time, $[u_{total}]$ represents all of the deformations, i.e. has exactly the same value which is calculated within purely elastic deformations model, and this purely elastic model can be considered a special case of a generalized model, when $[u_{plastic}] = [0]$.

Part of the deformations that is plastic deformations calculated each timestep using empiric formulae. The model is governed by three constants, which could be tweaked for each material: C_{yield} - value of an elastic strain tensor, when it is exceeded, residual deformation starts to occur; C_{creep} - rate at which residual deformation is accumulated; C_{max} - maximum amount of residual deformation, when it is exceeded, body starts to act purely elastic. The last constant is needed to avoid large plastic deformations, which will lead to unnatural behavior due to discrete nature of the model. These empiric formulae get evaluated each timestep for each finite element, on constraints update step:

Algorithm 1 Plasticity Update

$[u_{elastic}] = [u_{total}] - [u_{plastic}]$
if $||[u_{elastic}]||_2 > C_{yield}$ **then**
    $[u_{plastic}] = [u_{plastic}] + \Delta t \; C_{creep}[u_{elastic}]$
end if
if $||[u_{plastic}]||_2 > C_{max}$ **then**
    $[u_{plastic}] = [u_{plastic}] \cdot \frac{C_{max}}{||[u_{plastic}]||_2}$
end if

Typical Euclidean norm is used which can be easily calculated as a vector length, since all of the tensors represented as six dimensional vectors using Voigt notation. Unknowns in (**Algorithm 1**) are $[u_{total}]$ and $[u_{plastic}]$. The total strain tensor is calculated similarly as in purely elastic model, $[u_{total}] = [B](\mathbf{x} - \mathbf{x_0})$. Plastic strain tensor is zero initially (if the body is initially in undeformed state), and then changes on a timestep by timestep basis using (**Algorithm 1**).

In order to introduce plasticity into the model, in addition to plasticity update, general formula (33) should be rewritten using elastic strain tensor instead of total strain tensor:

$$[B]\mathbf{v}_t + \frac{1}{\mathrm{d}t}\frac{1}{V}[D]^{-1}\boldsymbol{\lambda}_0 = -\frac{1}{\mathrm{d}t}[u_{elastic}] \tag{48}$$

Where $[u_{elastic}]$ can be calculated using plasticity update (**Algorithm 1**). Model with damping (46) should be modified as well:

$$[B]\mathbf{v}_t + \frac{1}{1+\frac{\beta}{\mathrm{d}t}}\frac{1}{\mathrm{d}t}\frac{1}{V}[D]^{-1}\boldsymbol{\lambda}_0 = -\frac{1}{1+\frac{\beta}{\mathrm{d}t}}\frac{1}{\mathrm{d}t}[u_{elastic}] \tag{49}$$

All of the optimizations described in the article will still be applicable. E.g. in case of using stiffness warping, modified matrix $[B_R]$ will transform nodal displacements into rectified total strain tensor, thus elastic and plastic strain tensors, calculated from the total strain tensor will also appear rectified.

In case of diagonalization of $[R_r]$ (2.2), matrix $[B]$ is multiplied by some rotation matrix $[D'']$, i.e. $[B'](\mathbf{x} - \mathbf{x_0}) = [D''][B](\mathbf{x} - \mathbf{x_0}) = [D''][u_{total}]$. Given that rotation doesn't change length of a vector, we can modify (**Algorithm 1**) to take diagonalization into account.

Algorithm 2 Plasticity Update with Diagonalization

$[D''][u_{elastic}] = [D''][u_{total}] - [D''][u_{plastic}]$
if $||[D''][u_{elastic}]||_2 > C_{yield}$ **then**
    $[D''][u_{plastic}] = [D''][u_{plastic}] + \Delta t \; C_{creep}[D''][u_{elastic}]$
end if
if $||[D''][u_{plastic}]||_2 > C_{max}$ **then**
    $[D''][u_{plastic}] = [D''][u_{plastic}] \cdot \frac{C_{max}}{||[D''][u_{plastic}]||_2}$
end if

Since the initial value of $[D''][u_{plastic}] = [0] = [u_{plastic}]$; $[D''][u_{total}] = [B'](\mathbf{x} - \mathbf{x_0})$ and the right hand side of equations (48) or (49) will contain $[D''][u_{elastic}]$ (since we diagonalized $[R_r]$) – we can just solve for variables that inherit multiplication by $[D'']$ and just use (**Algorithm 1**) with (48) and (49). This shows that introducing plasticity doesn't affect optimizations that we discussed earlier.

3. Constrained Convex Optimization

3.1. Introduction

Initially, the method was developed specifically for coupled deformable-rigid body simulation, as it was discussed earlier in §2.2. Projected Gauss-Seidel, solver, commonly used for real-time dynamics nowadays fails to find solution fast enough for highly-stiff deformables, and we decided to develop a method with better convergence, and one of the possible research directions were Conjugate Gradient based methods. Such a method was developed, and it has proven to be better suitable for deformable bodies simulation than the typical PGS. Further, it showed superior convergence on variety of scenes with typical rigid body joints, which makes it useful for most real-time dynamics engines.

3.2. Algorithm Overview

Algorithm we present in this paper intended to solve mixed linear complementarity problem which could be written as:

$$[A]\mathbf{x} = \mathbf{b}$$
$$\boldsymbol{\ell} \leq \mathbf{x} \leq \boldsymbol{h}$$

As shown in [14], solving linear system is equal to minimization of following quadratic form:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T[A]\mathbf{x} - \mathbf{b}^T\mathbf{x} + \mathbf{c}$$

Solving *Mixed Linear Complementarity Problem* (MLCP) is equal to finding minimum of that quadratic form on a set of feasible solutions (see definitions below).

Our research is based on Zdeněk Dostál's work [15]. He proposed *Modified Proportioning with Reduced Gradient Projections* (MPRGP) method, which has R-linear rate of convergence and is easy parallelizable. The method, however, has quadratic time complexity, and works with lower limits only, which is not suitable for simulation with e. g. friction. **Algorithm 3** shows how the initial formulation of MPRGP looks like.

Definitions used (N is a set of indices $1..n$):

$$\Omega_B = \{\mathbf{x} \in \mathbb{R}^n : \boldsymbol{\ell} \leq \mathbf{x}\}$$
$$A(\mathbf{x}) = \{i \in N : x_i = \ell_i\}$$
$$F(\mathbf{x}) = \{i \in N : x_i \neq \ell_i\}$$
$$\phi_i = \begin{cases} g_i & , i \in F(\mathbf{x}) \\ 0 & , i \in A(\mathbf{x}) \end{cases}$$
$$\beta_i = \begin{cases} 0 & , i \in F(\mathbf{x}) \\ \min\{g_i, 0\} & , i \in A(\mathbf{x}) \end{cases}$$
$$\mathbf{g}^P = \boldsymbol{\phi}(\mathbf{x}) + \boldsymbol{\beta}(\mathbf{x})$$
$$\tilde{\phi}_i = \min\left\{\frac{x_i - \ell_i}{\overline{\alpha}}, \phi_i\right\}$$

where Ω_B is set of feasible solutions, $A(\mathbf{x})$ is active set, $F(\mathbf{x})$ is free set, $\boldsymbol{\beta}$ and $\boldsymbol{\phi}$ is chopped and free gradients respectively, which together produce projected gradient $\mathbf{g}^P$. $\tilde{\boldsymbol{\phi}}$ is reduced free gradient.

3.3. Introducing Box Constraints

As it already said above, basic MPRGP algorithm is not very suitable for dynamics simulation because of single limit usage. We developed an upgraded version of MPRGP which

Algorithm 3 Base MPRGP algorithm

Require: Symmetric positive definite matrix $[A]$ of the order n, n-vectors $\mathbf{b}$, $\boldsymbol{\ell}$, $\Omega_B = \{\mathbf{x} \in \mathbb{R}^n : \boldsymbol{\ell} \leq \mathbf{x}\}$, $\mathbf{x}^0 \in \Omega_B$

Step 0. {*Initialization*}

Choose $\Gamma > 0$, $\overline{\alpha} \in (0,\ 2||[A]||^{-1}]$, set $k = 0$, $\mathbf{g} = [A]\mathbf{x}^0 - \mathbf{b}$, $\mathbf{p} = \boldsymbol{\phi}(\mathbf{x}^0)$

while $||\mathbf{g}^P(\mathbf{x}^k)||$ is not small **do**

if $||\boldsymbol{\beta}(\mathbf{x}^k)||^2 \leq \Gamma^2 \tilde{\boldsymbol{\phi}}(\mathbf{x}^k)^T \boldsymbol{\phi}(\mathbf{x}^k)$ **then**

Step 1. {*Proportional* $\mathbf{x}^k$. *Trial conjugate gradient step*}

$$\alpha_{cg} = \frac{\mathbf{g}^T\mathbf{p}}{\mathbf{p}^T[A]\mathbf{p}},\quad \mathbf{y} = \mathbf{x}^k - \alpha_{cg}\mathbf{p}$$

$$\alpha_f = \max\left\{\alpha :\ \mathbf{x}^k - \alpha\mathbf{p} \in \Omega_B\right\} = \min\left\{\frac{x_i^k - \ell_i}{p_i} :\ p_i > 0\right\}$$

if $\alpha_{cg} < \alpha_f$ **then**

Step 2. {*Conjugate gradient step*}

$$\mathbf{x}^{k+1} = \mathbf{y},\quad \mathbf{g} = \mathbf{g} - \alpha_{cg}[A]\mathbf{p},$$

$$\beta = \frac{\boldsymbol{\phi}(\mathbf{y})^T[A]\mathbf{p}}{\mathbf{p}^T[A]\mathbf{p}},\quad \mathbf{p} = \boldsymbol{\phi}(\mathbf{y}) - \beta\mathbf{p}$$

else

Step 3. {*Expansion step*}

$$\mathbf{x}^{k+\frac{1}{2}} = \mathbf{x}^k - \alpha_f\mathbf{p},\quad \mathbf{g} = \mathbf{g} - \alpha_f[A]\mathbf{p}$$

$$\mathbf{x}^{k+1} = P_{\Omega_B}\left(\mathbf{x}^{k+\frac{1}{2}} - \overline{\alpha}\boldsymbol{\phi}(\mathbf{x}^{k+\frac{1}{2}})\right)$$

$$\mathbf{g} = [A]\mathbf{x}^{k+1} - \mathbf{b},\quad \mathbf{p} = \boldsymbol{\phi}(\mathbf{x}^{k+1})$$

end if

else

Step 4. {*Proportioning step*}

$$\mathbf{d} = \boldsymbol{\beta}(\mathbf{x}^k),\quad \alpha_{cg} = \frac{\mathbf{g}^T\mathbf{d}}{\mathbf{d}^T[A]\mathbf{d}}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_{cg}\mathbf{d},\quad \mathbf{g} = \mathbf{g} - \alpha_{cg}[A]\mathbf{d},\quad \mathbf{p} = \boldsymbol{\phi}(\mathbf{x}^{k+1})$$

end if

$k = k + 1$

end while

Step 5. {*Return (possibly inexact) solution*}

$\tilde{\mathbf{x}} = \mathbf{x}^k$

Algorithm 4 Our version of MPRGP with box constraints algorithm

Require: Symmetric positive definite matrix $[A]$ of the order n, n-vectors $\mathbf{b}$, $\boldsymbol{\ell}$, $\boldsymbol{h}$, $\Omega_B = \{\mathbf{x} \in \mathbb{R}^n : \boldsymbol{\ell} \leq \mathbf{x} \leq \boldsymbol{h}\}$, $\mathbf{x}^0 \in \Omega_B$

Step 0. {*Initialization*}

Choose $\Gamma > 0$, $\overline{\alpha} \in (0,\ 2||[A]||^{-1}]$, set $k = 0$, $\mathbf{g} = [A]\mathbf{x}^0 - \mathbf{b}$, $\mathbf{p} = \boldsymbol{\phi}(\mathbf{x}^0)$

while $||\mathbf{g}^P(\mathbf{x}^k)||$ is not small **do**

if $\tilde{\boldsymbol{\beta}}(\mathbf{x}^k)^T\boldsymbol{\beta}(\mathbf{x}^k) \leq \Gamma^2\tilde{\boldsymbol{\phi}}(\mathbf{x}^k)^T\boldsymbol{\phi}(\mathbf{x}^k)$ **then**

Step 1. {*Proportional* $\mathbf{x}^k$*. Trial conjugate gradient step*}

$$\alpha_{cg} = \frac{\mathbf{g}^T\mathbf{p}}{\mathbf{p}^T[A]\mathbf{p}},\quad \mathbf{y} = \mathbf{x}^k - \alpha_{cg}\mathbf{p}$$

$$\alpha_f = \max\left\{\alpha :\ \mathbf{x}^k - \alpha\mathbf{p} \in \Omega_B\right\} = \min\left\{\begin{array}{ll} \dfrac{x_i^k - \ell_i}{p_i} : & p_i > 0 \\ \dfrac{x_i^k - h_i}{p_i} : & p_i < 0 \end{array}\right\}$$

if $\alpha_{cg} < \alpha_f$ **then**

Step 2. {*Conjugate gradient step*}

$$\mathbf{x}^{k+1} = \mathbf{y},\quad \mathbf{g} = \mathbf{g} - \alpha_{cg}[A]\mathbf{p},$$

$$\beta = \frac{\boldsymbol{\phi}(\mathbf{y})^T[A]\mathbf{p}}{\mathbf{p}^T[A]\mathbf{p}},\quad \mathbf{p} = \boldsymbol{\phi}(\mathbf{y}) - \beta\mathbf{p}$$

else

Step 3. {*Expansion step*}

$$\mathbf{x}^{k+\frac{1}{2}} = \mathbf{x}^k - \alpha_f\mathbf{p},\quad \mathbf{g} = \mathbf{g} - \alpha_f[A]\mathbf{p}$$

$$\mathbf{x}^{k+1} = P_{\Omega_B}\left(\mathbf{x}^{k+\frac{1}{2}} - \overline{\alpha}\boldsymbol{\phi}(\mathbf{x}^{k+\frac{1}{2}})\right)$$

$$\mathbf{g} = [A]\mathbf{x}^{k+1} - \mathbf{b},\quad \mathbf{p} = \boldsymbol{\phi}(\mathbf{x}^{k+1})$$

end if

else

Step 4. {*Proportioning step preparations*}

$$\mathbf{d} = \boldsymbol{\beta}(\mathbf{x}^k),\quad \alpha_{cg} = \frac{\mathbf{g}^T\mathbf{d}}{\mathbf{d}^T[A]\mathbf{d}}$$

$$\alpha_f = \max\left\{\alpha :\ \mathbf{x}^k - \alpha\mathbf{d} \in \Omega_B\right\} = \min\left\{\begin{array}{ll} \dfrac{x_i^k - \ell_i}{d_i} : & d_i > 0 \\ \dfrac{x_i^k - h_i}{d_i} : & d_i < 0 \end{array}\right\}$$

if $\alpha_{cg} < \alpha_f$ **then**

Step 5. {*Proportioning step*}

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_{cg}\mathbf{d},\quad \mathbf{g} = \mathbf{g} - \alpha_{cg}[A]\mathbf{d},$$

$$\mathbf{p} = \boldsymbol{\phi}(\mathbf{x}^{k+1})$$

else

Step 6. {*Proportioning-expansion step*}

$$\mathbf{x}^{k+\frac{1}{2}} = \mathbf{x}^k - \alpha_f\mathbf{d},\quad \mathbf{g} = \mathbf{g} - \alpha_f[A]\mathbf{d}$$

$$\mathbf{x}^{k+1} = P_{\Omega_B}\left(\mathbf{x}^{k+\frac{1}{2}} - \overline{\alpha}\boldsymbol{\beta}(\mathbf{x}^{k+\frac{1}{2}})\right)$$

$$\mathbf{g} = [A]\mathbf{x}^{k+1} - \mathbf{b},\quad \mathbf{p} = \boldsymbol{\phi}(\mathbf{x}^{k+1})$$

end if

end if

$k = k + 1$

end while

Step 5. {*Return (possibly inexact) solution*}

$\tilde{\mathbf{x}} = \mathbf{x}^k$

could handle box constraints by introducing similar concept as in CG-step. Proportioning step now calculates alpha feasible too, and to improve convergence, performs half-step similar to one in the expansion step - please note that chopped gradient norm in proportionality condition replaced by dot product of chopped gradient and reduced chopped gradient (see definitions below). Also, additional check is performed in feasible step size calculation for CG step.

Having in mind the fact that additional limit introduced - higher limit, we need to reformulate some of the basic definitions:

$$
\begin{aligned}
&\Omega_B = \{\mathbf{x} \in \mathbb{R}^n : \boldsymbol{\ell} \leq \mathbf{x} \leq \boldsymbol{h}\} \\
&A(\mathbf{x}) = \{i \in N : x_i = \ell_i \vee x_i = h_i\} \\
&F(\mathbf{x}) = \{i \in N : x_i \neq \ell_i \wedge x_i \neq h_i\} \\
&\phi_i = \begin{cases} \mathbf{g}_i & , i \in F(\mathbf{x}) \\ 0 & , i \in A(\mathbf{x}) \end{cases} \\
&\beta_i = \begin{cases} 0 & , i \in F(\mathbf{x}) \\ \min\{g_i, 0\} & , x_i = \ell_i \\ \max\{g_i, 0\} & , x_i = h_i \end{cases} \\
&\tilde{\beta}_i = \begin{cases} \max\left\{\dfrac{x_i - h_i}{\overline{\alpha}}, g_i\right\} & , x_i = \ell_i \wedge g_i < 0 \\ \min\left\{\dfrac{x_i - \ell_i}{\overline{\alpha}}, g_i\right\} & , x_i = h_i \wedge g_i > 0 \\ 0 & , otherwise \end{cases} \\
&\tilde{\phi}_i = \begin{cases} \max\left\{\dfrac{x_i - \ell_i}{\overline{\alpha}}, \phi_i\right\} & , \phi_i > 0 \\ \min\left\{\dfrac{x_i - h_i}{\overline{\alpha}}, \phi_i\right\} & , \phi_i < 0 \end{cases}
\end{aligned}
$$

Algorithm 4 illustrates how modified algorithm will look like after implementing changes needed for box constraints.

3.4. Increasing Performance

The original MPRGP algorithm has $\mathrm{O}(N^2)$ complexity, which is not appropriate for real-time applications. We used common simplification (which is also used together with typical PGS solver, and was invented by Tonge et al., see [17]), which is simply decompose system matrix $[A]$ according to its representation in dynamics simulation engine:

$$[A] = [J][M]^{-1}[J]^T + [C]$$

where $[J]$ is Jacobi matrix, $[M]$ is mass matrix and $[C]$ is regularization matrix.

With that decomposition, $[A]\mathbf{x}$ multiplication which has $\mathrm{O}(N^2)$ complexity in general form, could be easily replaced with two $\mathrm{O}(N)$ operations:

$$
\begin{aligned}
\mathbf{a} &= [M]^{-1}[J]^T\mathbf{x} \\
\mathbf{x} &= [J]\mathbf{a} + [C]\mathbf{x}
\end{aligned}
$$

Matrices $[J]$ and $[M]$ stored as non-square matrices with fixed number of columns, and $[C]$ stored as a vector. Applied to conjugate gradient step, this decomposition will turn it into the following:

$$
\begin{aligned}
\mathbf{x}^{k+1} &= \mathbf{y} \\
\mathbf{a} &= [M]^{-1}[J]^T\mathbf{p} \\
\underline{[A]\mathbf{p}} &= [J]\mathbf{a} + [C]\mathbf{p} \\
\mathbf{g} &= \mathbf{g} - \alpha_{cg}\underline{[A]\mathbf{p}} \\
\beta &= \frac{\boldsymbol{\phi}(\mathbf{y})^T\underline{[A]\mathbf{p}}}{\mathbf{p}^T\underline{[A]\mathbf{p}}} \\
\mathbf{p} &= \boldsymbol{\phi}(\mathbf{y}) - \beta\mathbf{p}
\end{aligned}
\tag{50}
$$

where $\underline{[A]\mathbf{p}}$ is calculated and stored as a vector.

3.5. Preconditioner

Convergence of CG algorithm is heavily dependant on condition number ([16], ϵ is desired accuracy, κ is condition number and k is number of solver iterations):

$$
k \approx \ln\left(\frac{\epsilon}{||\mathbf{x} - \mathbf{x}^0||}\right)\frac{\sqrt{\kappa([A])}}{2}
$$

During development, several testcases showed ill-conditioned matrices, so we decided to use a preconditioner to lower condition number of a system's matrix. The most suitable preconditioner for real-time applications is Jacobi. However, preconditioning with Jacobi could ruin positive definiteness of a matrix, and our approach requires $[A]$ to be symmetric PD. The solution is to decompose preconditioning matrix $[P]$ = $[P_1]$ $[P_2]$, and apply $[P_1]$ as left preconditioner and $[P_2]$ as right preconditioner.

$$
[P_1]^{-1}[A][P_2]^{-1}\left([P_2]\mathbf{x}\right) = [P_1]^{-1}\mathbf{b}
$$

In case of Jacobi preconditioner, the simplest way to decompose preconditioning matrix is $[P] = [P]^{\frac{1}{2}}[P]^{\frac{1}{2}}$, and the fact that preconditioning matrix is diagonal simplifies it even further.

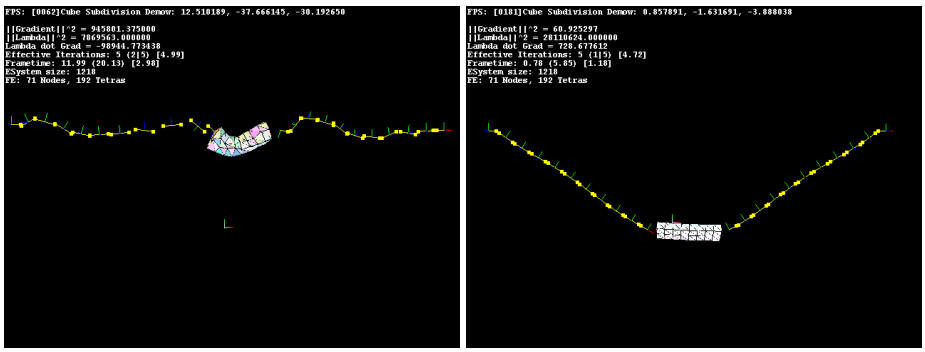
But we need to remember, that lower and higher limits should be transformed by inverse right preconditioner (hence multiplied by $[P]^{\frac{1}{2}}$), and the system is solved for $[P_2]\mathbf{x}$ now (thus after solve it should be transformed back to initial form with $[P]^{-\frac{1}{2}}$). In our implementation, we store inversed square root of the diagonal, so lo and hi limits should be divided by that vector component-wise:

$$
\boldsymbol{\ell} = \frac{\boldsymbol{\ell}}{[D]^{-\frac{1}{2}}}
$$

$$
\boldsymbol{h} = \frac{\boldsymbol{h}}{[D]^{-\frac{1}{2}}}
$$

Below is simple example, how calculation of gradient vector will change according to latest changes (matrix decomposition and double Jacobi preconditioning):

$$
\begin{aligned}
\mathbf{g} &= [A]\mathbf{x}^{k+1} - \mathbf{b} \\
&\Rightarrow \\
\mathbf{a} &= [M]^{-1}[J]^T[D]^{-\frac{1}{2}}\mathbf{x}^{k+1} \\
\mathbf{g} &= \left([J]\mathbf{a} + [C][D]^{-\frac{1}{2}}\mathbf{x}^{k+1} - \mathbf{b}\right)[D]^{-\frac{1}{2}}
\end{aligned}
\tag{51}
$$



(a) CG solver, "noisy" behavior (b) PGS solver, ball joints too stretched

Figure 1: Comparison of low iteration count behavior of two solvers

3.6. Drawbacks

Presented algorithm has several drawbacks which you need to take into account. First one is that due to CG-nature, you cannot use its results too from too low iterations count. While low iterations of PGS will result in visually softer joints, low iterations of CG-based algorithm will result in "noisy" behavior. For FEM-based deformables though, low number of iterations results in increasing oscillation with PGS solver. See Figure 1 for examples.

One iteration of this algorithm is still more complex than one iteration of PGS algorithm, and as it could be seen from pseudo-code, algorithm requires spectral radius calculation (used in the upper bound of $\overline{\alpha}$ calculation). We implemented it via Power Iteration algorithm, one iteration of which is slightly simpler than one iteration of PGS, and overall computational cost of presented algorithm is still quite high. Fortunately, it is parallelizable and hence we could take benefits from its GPU implementation (see next chapter for timings).

3.7. Results

In all tests we used 15 power iterations to calculate $|||[A]|||$ and $\overline{\alpha} = 1.6|||[A]|||^{-1}$ to ensure that we will not cross the upper bound. We didn't test other configurations though, so probably less iterations could be performed in Power Iteration step. Γ was set to 1.0 in all our tests.

We present here results of a test, shown in Figure 2: massive 3D grid of ball-in-socket joints configured as 2D net [20×20 nodes, 4 corner nodes fixed], and to each node, chain of ball-sockets attached [another 20 nodes]. Each node has mass 1.0 and inertia tensor as if it was sphere with radius 0.5.

Resulting system has approx. 21.5k rows (more than 7k ball-in-socket joints). In this configuration, system is simulated in max. 45ms with new solver (maximum amount of iterations performed: 140) on NVIDIA GeForce GTX260. Our PGS implementation was able to solve same system with same accuracy only with max. 3110ms and 840 maximum iterations on a single core of Intel Core i7 920 processor. Setting higher iteration limit of PGS to 140 leads to max. 520ms dynamics time. However, we should mention, that both of our implementations are not pretend to be very optimal, in GPU implementation of there much of accompanying code that still performed on CPU.

Some additional statistics on the new solver only. FEM stress test, brick $96 \times 2 \times 1$ segments (2.3k tetrahedra), 1 GPa Young modulus, 0.35 Poisson's ratio, 2200 mass requires max. 220 iterations to converge to desired accuracy 0.0001(physics update timing is 50ms). Due to some constraint regularization, block is a bit softer than it should be, resulting in image of maximum bend, shown in Figure 3.

With less regularization, following stiffness could be achieved with 150ms per physics update (all parameters same, avg. number of iterations 420), see Figure 4.

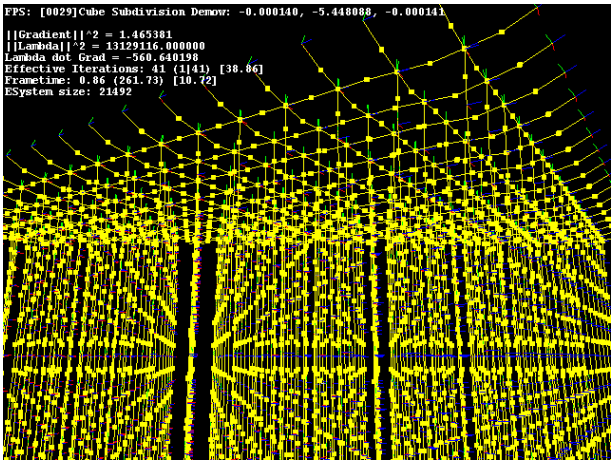


Figure 2: Ball joint grid stress test

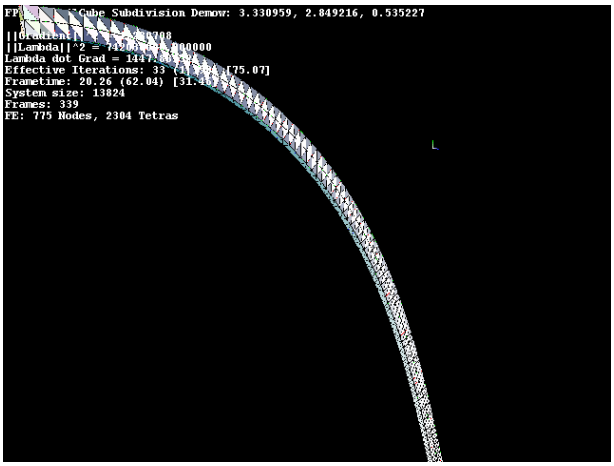


Figure 3: FEM stress test, high regularization

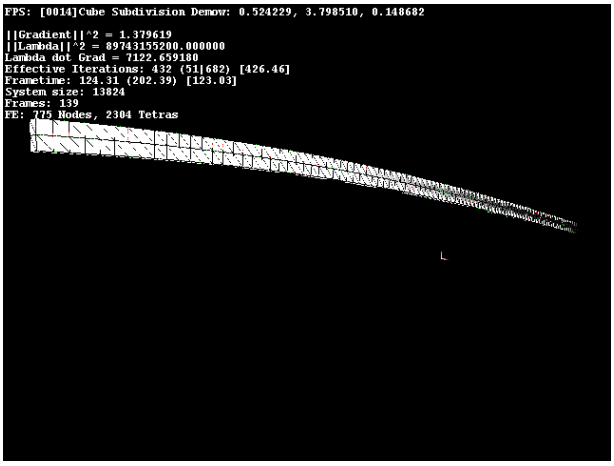


Figure 4: FEM stress test, low regularization

4. Conclusion and future work

The paper provide a detailed description of how to derive and implement coupled Finite Element Method based continuum mechanics simulation with the conventional articulated rigid body dynamics simulation. The coupling is done by means of formulating FEM as a typical rigid body constraint, i.e. via Jacobi matrix, regularization and right hand side vector calculations, which allows seamless integration within single solver iteration. Suggested model incorporates both elastic and plastic deformations, and supports friction.

There are several ways to follow up on this research:

- Expanding the model to support more complex materials, and non-linear FE characteristics.
- Introducing additional dynamic systems into simulation, for example fluid simulation.
- Reformulate the system in terms of smoothed particles, rather than conventional tertrahedral finite elements, this will open up possibilities to do proper melting and mixing of the substances.

5. References

[1] L. Landau, E. Lifshitz, *Theory of Elasticity*, 1959

[2] Wikipedia, *Voigt notation*, https://en.wikipedia.org/wiki/Voigt_notation

[3] Wikipedia, *Hooke's law*, https://en.wikipedia.org/wiki/Hooke's_law

[4] Wikipedia, *Polar decomposition*, https://en.wikipedia.org/wiki/Polar_decomposition

[5] Wikipedia, *Square root of a matrix*, https://en.wikipedia.org/wiki/Square_root_of_a_matrix

[6] R. Bridson, J. Teran, N. Molino, R. Fedkiw, *Adaptive Physics Based Tetrahedral Mesh Generation Using Level Sets*, Engineering with Computers 21, 2-18 (2005)

[7] R.D. Cook, *Finite Element Modeling for Stress Analysis*, John Wiley & Sons, 1995

[8] M. Mueller, M. Gross, *Interactive Virtual Materials*, ETH Zurich, 2004

[9] M. Muller, J. Dorsey, L. McMillan, R. Jagnow, B. Cutler, *Stable real-time deformations*, Proceedings of ACM SIGGRAPH Symposium on Computer Animation, pages 4954, 2002

[10] M. Servin, C. Lacoursiére, N. Melin, *Interactive Simulation of Elastic Deformable Materials*, SIGRAD, 2006

[11] K. Shoemake, T. Duff, *Matrix Animation and Polar Decomposition*, In Proceedings of the conference on Graphics interface(1992): 258-264

[12] J. F. O'Brien, A. W. Bargteil, J. K. Hodgins, *Graphical Modeling and Animation of Ductile Fracture*, Computer Graphics Proceedings, 2002

[13] J. F. O'Brien, J. K. Hodgins, *Graphical Modeling and Animation of Brittle Fracture*, Georgia Institute of Technology, July 1999

[14] J. R. Shewchuk, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*, School of Computer Science Carnegie Mellon University, 1994

[15] Z. Dostál *Optimal Quadratic Programming Algorithms With Applications to Variational Inequalities*, Technical University of Ostrava, Czech Republic, 2009

[16] M. Kreutzer, *Convergence and performance comparison between Gauss-Seidel and Conjugate Gradient method*, 2009

[17] R.Tonge, L.Zhang, D.Sequeira, *Method and program solving LCPs for rigid body dynamics*, U.S. Patent 7079145, 2004