Abstract

English title

Keywords: k1, k2

Streszczenie

Tytuł polski Tytuł polski Tytuł polski

Słowa kluczowe: slowo1, slowo2

Bolesław Prus
Nr albumu 100000

Warsaw, ...

Declaration

I hereby declare that the thesis entitled „English title", submitted for the magisters degree, supervised by dr inż. Promotor Promotorski, is entirely my original work apart from the recognized reference.

.....

Bolesław Prus

Spis treści

Introduction

What is this thesis about.

Soft Body Dynamics - Overview

Mass Spring

Formulation

The simplest method to simulate soft body deformation is Mass Spring System. Such a system includes a set of N particles with masses m_i, positions x_i and velocites v_i, where $i \in 1...N$. The particles are connected by a set of springs S. A single spring $s \in S$ consists of $s = (i, j, l_0, k_s, k_d)$, where i and j are the indices of connected particles, l_0 is the rest length, k_s is the spring stiffness and k_d is the damping coefficient. To calcuate the forces acting on particles i and j, we use the following formula:

$$f_i^S = f^S(x_i, x_j) = k_s \frac{x_j - x_i}{|x_j - x_i|}(|x_j - x_i| - l_0) \tag{1.1}$$

$$f_j^S = f^S(x_j, x_i) = -f^S(x_i, x_j) \tag{1.2}$$

It is easy to see that the forces conserve momentum, since $(f_i + f_j = 0)$

We apply damping by computing the damping forces:

$$f_i^D = f^D(x_i, v_i, x_j, v_j) = k_d(v_j - v_i)\frac{x_j - x_i}{|x_j - x_i|} \tag{1.3}$$

$$f_j^D = f^D(x_j, v_j, x_i, v_i) = -f_i^D \tag{1.4}$$

Combining two forces we get the final spring force

$$f(x_i, v_i, x_j, v_j) = f^S(x_i, x_j) + f^D(x_i, v_i, x_j, v_j) \tag{1.5}$$

Simulation

In order to simulate the mass spring system, we use the Newton's second law of motion,

$$f = m\ddot{x} \tag{1.6}$$

where f is the force, m is the mass and $\ddot{x}$ is the acceleration or the second derivate of position with respect to time. By transforming the equation to solve for acceleration, we get a second order ordinary differential equation(ODE):

$$\ddot{x} = \frac{f}{m} \tag{1.7}$$

In order to solve it, we can split this equation into two first order ODEs

$$\dot{v} = \frac{f}{m} \tag{1.8}$$

$$\dot{x} = v \tag{1.9}$$

Analytically, these can be solved by definite integrals:

$$v(t) = v_0 \int_{t_0}^{t} \frac{f(t)}{m} dt \tag{1.10}$$

$$x(t) = x_0 \int_{t_0}^{t} v(t) dt \tag{1.11}$$

where $v_0 = v(t_0)$ and $x_0 = x(t_0)$ are the initial conditions.

Numerical Integration

Explicit Euler Integration

One of the most basic numerical integration of ODE is explicit Euler integration scheme. The scheme approximates the derivatives using finite differences:

$$\dot{v} = \frac{v^{t+1} - v^t}{\Delta t} \tag{1.12}$$

$$\dot{x} = \frac{x^{t+1} - x^t}{\Delta t} \tag{1.13}$$

where Δt is a discrete time step and t is the index of the simulation iteration. By substituting these equations into Eq. 1.8 and Eq. 1.9, we get get the explicit Euler integration method:

$$v^{t+1} = v^t + \Delta t \frac{f(x^t, v^t)}{m} \tag{1.14}$$

$$x^{t+1} = x^t + \Delta t v^{t+1} \tag{1.15}$$

The term 'explicit' comes from the fact that information of the next time step can be directly computed using the information at the current time step.

The entire simulation can be summed with the following algorithm:

f^g is the gravity force and f^{coll} collision forces.

Algorithm 1 Mass Spring Simulation

1: **procedure** Simulation
2: **while** true **do**
3: **for** all particles i **do**
4: $f_i = f^g + f_i^{coll} + \sum_{j,(i,j)\in S} f(x_i, v_i, x_j, v_j)$
5: **for** all particles i **do**
6: $v_i = v_i + \Delta t \frac{f_i}{m_i}$
7: $x_i = x_i + \Delta t v_i$

A known drawback for explicit Euler integration is the fact that it requires small time steps to remain stable. This problem accurs becouse explicit Euler does not account for the near future and it assumes that the force is constant during the entire time step. Let us assume a system of two particles connected with a spring. Assume the following configuration: the spring is stretched and the two particles start moving towards each other. If we take a large time step to compute the next configuration, the particles might pass the equilibrium configuration, which in theory means that the force should change its sign during that time step. Sadly, since the force is constant throughtout the entire time step, the sign change of the force is not accounted for. This might lead to particles overshooting and gaining energy which in turn leads to a so called simulation explosion. Other numerical integration methods exist that are more accurate. Among the most popular are the second and fourth order Runge-Kutta integrators. These schemes compute forces multiple times during a single time step, which might reduce the effect of the problem mentioned above.

Runge-Kutta Integration

The second order Runge-Kutta integrator has a different method of numerically solving ODEs. The approximation of explicit Euler Eq. 1.14 and Eq. 1.15 are instead computed by the formulas:

$$\begin{aligned} a_1 &= v^t \\ a_2 &= \frac{f(x^t, v^t)}{m} \\ b_1 &= v^t + \frac{\Delta t}{2} a_2 \\ b_2 &= \frac{f(x^t + \frac{\Delta t}{2} a_1, v^t + \frac{\Delta t}{2} a_2)}{m} \\ x^{t+1} &= x^t + \Delta t b_1 \\ v^{t+1} &= v^t + \Delta t b_2 \end{aligned} \tag{1.16}$$

It is easy to see that the forces are computed twice during one time step. This makes the secord order Runge-Kutta integrator more accurate compared to the simple first order explicit Euler method.

One of the most popular methods of integrating ODE is a forth order Runge-Kutta integrator. It extends the the second order Runge-Kutta by computing the force four times during a single time step. Making it even more accurate. The accuracy obviously comes with longer computations.

Implicit Euler

Another way to improve stability is to use implicit integrator. Among the most populars is the implicit Euler method. As opposed to explicit integrators, the implicit is more physically correct.

$$v^{t+1} = v^t + \Delta t \frac{f(x^{t+1})}{m} \tag{1.17}$$

$$x^{t+1} = x^t + \Delta t v^{t+1} \tag{1.18}$$

First difference lies in the force function f. Now it only depends on the position. In another words, the force does not include friction. It is said that implicit integration introduces enough numerical damping to accomedate for physical damping(TODO citation). However, if needed, the friction force can be added in the explicit step after the implicit solve. The most important change, is the fact that the force f depends now on the position of next step x^{t+1}. Thus, it is no longer possible to explicitly compute the two equations. Instead we now deal with a algebraic system, with unknowns being x^{t+1} and v^{t+1}.

In order to compute these equation we must first construct the algebraic system. The position, velocites and forces are concatanted into vectors:

$$\begin{aligned} x &= [x_1, x_2, ..., x_n] \\ v &= [v_1, v_2, ..., v_n] \\ f(x) &= [f_1(x_1, ..., x_n), ..., f_n(x_1, ..., x_n)] \end{aligned} \tag{1.19}$$

Further, we construct a mass matrix $M \in \mathbb{R}^{3N \times 3N}$ which is diagonal with values $m_1, m_1, m_1, ..., m_N, m_N, m_N$ on the diagonal.

$$Mv^{t+1} = Mv^t + \Delta t f(x_{t+1}) \tag{1.20}$$

$$x^{t+1} = x + \Delta t v^{t+1} \tag{1.21}$$

Substituing Eq. 1.21 into Eq. 1.20, results in single system of algebraic equations:

$$Mv^{t+1} = Mv^t + \Delta t f(x + \Delta t v^{t+1}) \tag{1.22}$$

We solve this system for v^{t+1}.

Examples

Example implementation of mass spring system can found in open source software created by the author of this thesis in [2]. TODO Show screen TODO Give profesional example

Conclusions

Mass spring systems are easy to implement and for many applications give good enough results(e.g. computer games). However, relatively expensive ODE integrators have to be used in order to keep the simulation stable. Moreover, modeling physically correct materials can be a complicated task, since the parameters of the system hardly reflect reality.

Position Based Dynamics

As the name suggests, Position Based Dynamics(PBD) omits integrating over velocity and works directly on positions. The biggest advantage over mass spring system is avoidance of overshooting problem during integration step.

Formulation

The system of PBD includes a set of N particles and a set of M contraints. Each particle i has three attributes:

1. mass m_i
2. position x_i
3. velocity v_i

Each constraint j has five attributes:

1. Cardinality - n_j
2. Scalar constraint function - $C_j : \mathbb{R}^{3n_j} \to \mathbb{R}$
3. Set of indices - $\{i_1, ..., i_{n_j}\}, i_k \in [1, ..., N]$

4. Stiffness parametr - $k_j \in [0...1]$

5. Type - *unilateral* or *bilateral*

It is said that bilateral contraint j is satisfied if $C_j(x_{i_1}, ..., x_{i_{n_j}}) = 0$ or if the case of unilateral $C_j(x_{i_1}, ..., x_{i_{n_j}}) \geqslant 0$. The strength of the contraint is defined by the stiffness parameter k_j.

Given initial conditions for positions and velocities the simulation proceeds as follows:

Algorithm 2 Particle Based Dynamics

1: **procedure** Simulation
2: **while** true **do**
3: **for** all particles i **do**
4: $v_i = v_i + \Delta t \frac{f_i}{m_i}$
5: $p_i = x_i + \Delta t v_i$
6: $generateCollisionContraints(x_i, p_i)$
7: **while** iteratively **do**
8: $projectContraints(C_1, ..., C_{M+M_{ext}}, p_1, ..., p_N)$
9: **for** all particles i **do**
10: $v_i = \frac{(p_i - x_i)}{\Delta t}$
11: $x_i = p_i$

The lines 4 and 5 compute explicit Euler integration on velocities and positions. However, output positions p_i are only used as predictions. The line 6 generates external constrain such as collisions. The original and predicted positions x_i, p_i can be used in this step in order to perform continuous collision detection. The simulation then computes line 8 which iteratively corrects the predicted positions such that they satisfy the M_{ext} external and M internal constraints. Finally, the corrected positions p_i are used to compute velocites and positions.

Finite Element Method

Solid Mechanics

Introduction

This assumes introduction to linear materials

Stress tensor mass density external forces strain tensor. Chauchy displacement stiffness tensor Stress vector

Formulation

Strong formulation

For linear static solid deformation.

$$\begin{aligned} \sigma_{ij,j} + \hat{f}_i = 0 \quad & x \in \Omega \\ \sigma_{ij} = C_{ijkl}\epsilon_{kl} \quad & x \in \Omega \\ \epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}) \quad & x \in \Omega \end{aligned} \tag{2.1}$$

where $\hat{f}_i = \rho f_i$

$$\begin{aligned} u_i = \hat{u_i} \quad & x \in \partial\Omega_u \\ \sigma_{ij} n_j = \hat{t_i} \quad & x \in \partial\Omega_\sigma \end{aligned} \tag{2.2}$$

Dynamic System

In order to accommodate for the dynamic behavior of the system, we must slightly change the equation of motion.

$$\sigma_{ij,j} + \hat{f}_i = \rho\ddot{u}_i \quad x \in \Omega \tag{2.3}$$

where ρ is a known quantity.

Weak formulation

Set of kinematically acceptable displacement fields or trial functions

$$\mathcal{P} = \{u(x) : u_i = \hat{u}_i \quad for \quad x \in \partial\Omega_u\} \tag{2.4}$$

Set of kinematically acceptable variations of function in $\mathcal{P}$.

$$\mathcal{W} = \{\delta u(x) : \delta u_i = 0 \quad for \quad x \in \partial\Omega_u\} \tag{2.5}$$

δu is called a virtual displacement.

For any variation $\delta u \in \mathcal{W}$ it is true that:

$$\int_\Omega (\sigma_{ij,j} + \hat{f}_i)\delta u_i dV - \int_{\partial\Omega_\sigma} (\sigma_{ij} n_j - \hat{t}_i)\delta u_i dA = 0 \tag{2.6}$$

Let us transform the second integral from Eq. 2.6, having in mind that $\partial\Omega = \partial\Omega_u \cup \partial\Omega_\sigma$. From definition of Eq. 2.5, we see that variation δu_i vanishes on remaining part of the boundary space $\partial\Omega_u$. Thus, we can treat this intergral as a integral over entire boundary space $\partial\Omega$. Now we can apply Gauss-Ostrogradsky theorem and transform it to:

$$\int_{\partial\Omega_\sigma} (\sigma_{ij} n_j - \hat{t}_i)\delta u_i dA = \int_\Omega (\sigma_{ij,j}\delta u_i + \sigma_{ij}\delta u_{i,j}) dV - \int_{\partial\Omega_\sigma} \hat{t}\delta u_i dA \tag{2.7}$$

Substituting the above equation in Eq. 2.6 and naming $\delta\epsilon = \text{sym}\ \delta u_{i,j}$ we get:

$$\int_\Omega (\sigma_{ij}\delta\epsilon_{ij} dV) = \int_\Omega \hat{f}_i \delta u_i dV + \int_{\partial\Omega_\sigma} \hat{t}_i \delta u_i dA \tag{2.8}$$

The Eq. 2.8 is called the principle of virtual work for linear, static solid deformations. It claims that the work made by external forces on the virtual displacements(right hand side) is equal to the work made by interior forces(stress) on certain virtual displacement.

By including the constitutive equation and the symmetry of stiffness tensor, we can further transform the Eq. 2.8:

$$\int_\Omega (C_{ijkl} u_{k,l}\delta u_{i,j} dV) = \int_\Omega \hat{f}_i \delta u_i dV + \int_{\partial\Omega_\sigma} \hat{t}_i \delta u_i dA \tag{2.9}$$

Dynamic System

Once again, we want to add dynamic behavior to the system. We do this by following that same steps but this time using the dynamic equation of motion(E.q 2.3).

$$\int_\Omega (C_{ijkl} u_{k,l}\delta u_{i,j} dV) + \int_\Omega \rho\ddot{u}_i \delta u_i dV = \int_\Omega \hat{f}_i \delta u_i dV + \int_{\partial\Omega_\sigma} \hat{t}_i \delta u_i dA \tag{2.10}$$

Discrete formulation

The displacement fields $u_i(x)$ and their variations $\delta u_i(x)$ must be properly included in classes $\mathcal{P}$ and $\mathcal{W}$. By the defitions of classes $\mathcal{P}$ and $\mathcal{W}$, the following must stay true:

$$\forall_{u_i \in \mathcal{P}, \delta u_i \in \mathcal{W}} \quad u_i + \delta u_i \in \mathcal{P} \tag{2.11}$$

Further, we must include the boundary conditions:

$$\begin{aligned} u_i &= \hat{u_i} \quad x \in \partial\Omega_u \\ \delta u_i &= 0 \quad x \in \partial\Omega_u \end{aligned} \tag{2.12}$$

The displacement fields must also be once-differentiable as can be seen on the left hand side of Eq. 2.9.

Let us consider the following classes of displacement fields and their variations.

$$\begin{aligned} \mathcal{P}^{\mathcal{N}} &= \{\bar{u}(x) = \hat{\Phi}_i(x) + \Phi_{i\alpha}(x) q_\alpha\} \\ \mathcal{W}^{\mathcal{N}} &= \{\bar{\delta u}(x) = \Phi_{i\alpha}(x) \delta q_\alpha\} \end{aligned} \tag{2.13}$$

This is a linear combination of three systems of N linearly independent shape functions $\Phi_{i\alpha}(x)$, $i = 1, 2, 3$, with real coefficients q_α and δq_α. These shape functions satisfy the diffrentiable criteria. We choose such functions $\Phi_{i\alpha}(x)$ so that they satisfy the condition $\Phi_{i\alpha} = 0$ on $\partial\Omega_u$. Furthermore, $\hat{\Phi}_i(x)$ $i = 1, 2, 3$ are any functions satisfying the condition $\hat{\Phi}_i(x) = \hat{u_i}$ on $\partial\Omega_u$.

Let us substitute such displacements fields and their variations into Eq. 2.9.

$$\delta q_\alpha [q_\beta \int_\Omega C_{ijkl} \Phi_{i\alpha,j} \Phi_{k\beta,l} dV + \int_\Omega C_{ijkl} \Phi_{i\alpha,j} \hat{\Phi}_{k,l} dV - \int_\Omega \hat{f}_i \Phi_{i\alpha} dV - \int_{\partial\Omega_\sigma} \hat{t}_i \Phi_{i\alpha} dA] = 0 \tag{2.14}$$

Introducing new notation:

$$\begin{aligned} K_{\alpha\beta} &= \int_\Omega C_{ijkl} \Phi_{i\alpha,j} \Phi_{k\beta,l} dV \\ Q_\alpha &= -\int_\Omega C_{ijkl} \Phi_{i\alpha,j} \hat{\Phi}_{k,l} dV + \int_\Omega \hat{f}_i \Phi_{i\alpha} dV + \int_{\partial\Omega_\sigma} \hat{t}_i \Phi_{i\alpha} dA \end{aligned} \tag{2.15}$$

We can now rewrite Eq. 2.14:

$$\delta q_\alpha (K_{\alpha\beta} q_\beta - Q_\alpha) = 0 \tag{2.16}$$

The above equation must be true for all variation $\delta \bar{u}_i$, thus they are also true for all coefficients δq_α. Finally, the following system of equations must be true for all q_α coefficients:

$$K_{\alpha\beta} q_\beta = Q_\alpha \tag{2.17}$$

We can write this in matrix form:

$$K_{N\times N} q_{N\times 1} = Q_{N\times 1} \tag{2.18}$$

In Eq. 2.24 we are presented with a linear system of algebraic equations. Matrix K is called a stiffness matrix. Vector Q is called a exterior force vector, associated with displacements q. For now it will be only mentioned briefly that matrix K is a symmetric and sparse matrix. Details will follow.

Dynamic System

Let us consider more general case in which we include dynamic behavior. The displacement fields are not functions of time.

$$\bar{u}(x,t) = \hat{\Phi}_i(x,t) + \Phi_{i\alpha}(x)q_\alpha(t) \tag{2.19}$$

The first and second derivative over time:

$$\begin{aligned} \dot{\bar{u}}(x,t) &= \dot{\hat{\Phi}}_i(x,t) + \Phi_{i\alpha}(x)\dot{q_\alpha}(t) \\ \ddot{\bar{u}}(x,t) &= \ddot{\hat{\Phi}}_i(x,t) + \Phi_{i\alpha}(x)\ddot{q_\alpha}(t) \end{aligned} \tag{2.20}$$

We can now state the discrete formulation for dynamic system using the Eq. 2.10.

$$\delta q_\alpha [q_\beta \int_\Omega C_{ijkl}\Phi_{i\alpha,j}\Phi_{k\beta,l}dV + \int_\Omega C_{ijkl}\Phi_{i\alpha,j}\hat{\Phi}_{k,l}dV + \ddot{q_\beta}\int_\Omega \rho\Phi_{i\alpha}\Phi_{i\beta}dV + \int_\Omega \rho\Phi_{i\alpha}\ddot{\hat{\Phi}}_i dV - \int_\Omega \hat{f}_i\Phi_{i\alpha}dV - \int_{\partial\Omega_\sigma} \hat{t}_i\Phi_{i\alpha}d \tag{2.21}$$

As before, Eq. 2.21 must be satisfied for all coefficients δq_α which means that everything else must vanish for all indices α. Let us denote mass matrix by:

$$M_{\alpha\beta} = \int_\Omega \rho\Phi_{i\alpha}\Phi_{j\beta}dV \tag{2.22}$$

The stiffness matrix K does not change, however we must change force vector Q.

$$Q_\alpha = -\int_\Omega C_{ijkl}\Phi_{i\alpha,j}\hat{\Phi}_{k,l}dV + \int_\Omega (\hat{f}_i - \rho\ddot{\hat{\Phi}}_i)\Phi_{i\alpha}dV + \int_{\partial\Omega_\sigma} \hat{t}_i\Phi_{i\alpha}dA \tag{2.23}$$

We can expand our system of linear equations to:

$$M_{N\times N}\ddot{q}_{N\times 1} + K_{N\times N}q_{N\times 1} = Q_{N\times 1} \tag{2.24}$$

This is a linear system of differential equations of second order. The unknows being the displacement fields $q_\alpha(t)$, which must be satisfied for each time t. Details on solving this system will follow.

The final part of dynamic system is the damping matrix C. Damping is introduced to simulate friction forces internal to the structure. One model of creating damping matrix C for structural systems is called Rayleigh damping matrix [5]:

$$C = \alpha M + \beta K \tag{2.25}$$

In other words, C is the linear combination of mass and stiffness matrices.

Thus the final form of dynamic system is presented here:

$$M_{N\times N}\ddot{q}_{N\times 1} + C_{N\times N}\dot{q}_{N\times 1} + K_{N\times N}q_{N\times 1} = Q_{N\times 1} \tag{2.26}$$

Finite Element Method

We will divide the space Ω using E finite elements, each represented by the set Ω_e, $e = 1, 2, ..., E$, where $\Omega_e \cap \Omega_f = \emptyset$, for $e \neq f$.

Boundary of e-th element will be denoted $\partial\Omega_e$. Common boundary of neighbours e and f will be denoted $\partial\Omega_{ef} = \partial\Omega_e \cap \partial\Omega_f$. The part of elements boundary that also happens to be boundary of the entire body will be denoted $\partial\Omega_{\bar{e}} = \partial\Omega \cap \partial\Omega_e$.

We can use the finite elements to compute the integral over the entire body.

$$\int_\Omega (.)dV = \sum_{e=1}^{E} \int_{\Omega_e} (.)dV \tag{2.27}$$

Similarly for the integrals over the area:

$$\int_{\partial\Omega} (.)dA = \sum_{\bar{e}\in\{E_{\partial\Omega}\}} \int_{\partial\Omega_{\bar{e}}} (.)dA \tag{2.28}$$

where $E_{\partial\Omega}$ is a set of finite elements which have non empty $\partial\Omega_{\bar{e}}$ set.

In FEM models we can apply physical intuition behind the discrete formulation. Recall that the shape functions $\Phi_\alpha(x)$ were associated with certain real coefficients q_α. These coefficients will now be associated with vertices x_α of the finite elements. The coefficient q_α will now be called parameter of vertex x_α. In 3D solid mechanics the parameter q_α denotes the component of displacements vector of the associated vertex x_α. Formally the displacement field will be aproximated by:

$$\bar{u}(x) = \Phi_\alpha(x)q_\alpha \tag{2.29}$$

for $\alpha = 1, 2, ..., N$, where N is the number of vertices $x_\alpha \in \Omega$.

Moreover, we can require that the coefficients q_α will be the value of approximated displacement fields at their associated vertices x_α:

$$\bar{u}(x_\alpha) = q_\alpha \tag{2.30}$$

We can easly design a shape function satisfying the above criteria:

$$\Phi_\alpha(x_\beta) = \delta_{\alpha\beta} \tag{2.31}$$

for $\alpha, \beta = 1, 2, ..., N$. In other words, the shape function $\Phi_\alpha(x)$ associated with vertex x_α should have value $\Phi_\alpha(x_\alpha) = 1$ and value 0 in any other vertex. Moreover, we usually require the shape function to sum up to 1:

$$\sum_{\alpha=1}^{N} \Phi_\alpha(x) = 1 \tag{2.32}$$

FEM for Solid Mechanics

For 3D solid mechanics, the approximated displacement field has a form of:

$$\bar{u}_i(x) = \Phi_{i\alpha}(x) q_\alpha \tag{2.33}$$

where $i = 1, 2, 3$ and q_α denotes the values of displacements in associated vertices. However, since in 3D the displacement vector is described by 3 components(x, y, z), we must expand the range of α index. For model with N vertices, $\alpha = 1, 2, ...3N$. We can now represent the Eq. 2.34 in matrix form:

$$\bar{u}_{3\times1}(x) = \Phi^T_{3\times3N}(x) q_{3N\times1} \tag{2.34}$$

That makes our system of linear equations:

$$K_{3N\times3N} q_{3N\times1} = Q_{3N\times1} \tag{2.35}$$

where

$$\begin{aligned} K_{\alpha\beta} &= \int_\Omega C_{ijkl} \Phi_{i\alpha,j} \Phi_{k\beta,l} dV \\ Q_\alpha &= \int_\Omega \hat{f}_i \Phi_{i\alpha} dV + \int_{\partial\Omega_\sigma} \hat{t}_i \Phi_{i\alpha} dA \end{aligned} \tag{2.36}$$

We can use the fact that C_{ijkl} is a symmetric tensor to further simplify the equations. We will represent second order tensor with 6×1 vector and a fourth order tensor with 6×6 matrix.

$$\begin{aligned} K_{3N\times3N} &= \int_\Omega B^T_{6\times3N} C_{6\times6} B_{6\times3N} dV \\ Q_{3N\times1} &= \int_\Omega \Phi_{3N\times3} \hat{f}_{3\times1} dV + \int_{\partial\Omega_\sigma} \Phi_{3N\times3} \hat{t}_{3\times1} dA \end{aligned} \tag{2.37}$$

Where $B_{6\times3N}$ is called a geometric matrix and is defined as follows:

$$\begin{aligned} [B_{6\times3N}] &= [B^{(1)}_{6\times3} B^{(2)}_{6\times3} ... B^{(N)}_{6\times3}] \\ [B^{(\bar{\alpha})}_{6\times3}] &= \begin{bmatrix} \Phi^{\bar{\alpha}}_{,1} & & \\ & \Phi^{\bar{\alpha}}_{,2} & \\ & & \Phi^{\bar{\alpha}}_{,3} \\ \Phi^{\bar{\alpha}}_{,2} & \Phi^{\bar{\alpha}}_{,1} & \\ & \Phi^{\bar{\alpha}}_{,3} & \Phi^{\bar{\alpha}}_{,2} \\ \Phi^{\bar{\alpha}}_{,3} & & \Phi^{\bar{\alpha}}_{,1} \end{bmatrix} \end{aligned} \tag{2.38}$$

If in a given finite element e there are N_e vertices then the local count of vertex parameters is equal to $3N_e$. We will now introduce local stiffness matrix $k^{(e)}_{3N_e \times 3N_e}$ and force vector $p^{(3)}_{3N_e \times 1}$. These are computed using the Eq. 2.37 but using only the local shape functions of each finite element, namely the nonzero values on Ω_e section. However, we must somehow map these local values to our global values, K and Q. We introduce partial global stiffness matrix $K^{(e)}_{3N \times 3N}$ and partial global force vector $Q^{(e)}_{3N \times 1}$. They contain the local values(k^e and $p^{(e)}$) of the finite element e, mapped into their global coordinates using boolean matrix $A^{(e)}_{3N_e \times 3N}$.

$$\begin{aligned} K^{(e)}_{\alpha\beta} &= A^{(e)}_{a\alpha} k^{(e)}_{ab} A^{(e)}_{b\beta} \\ Q^{(e)}_{\alpha} &= p^{(e)}_{a} A^{(e)}_{a\alpha} \end{aligned} \tag{2.39}$$

These partial global matrices are then summed into global matrices K and Q.

$$\begin{aligned} K_{\alpha\beta} &= \sum_{e=1}^{E} K^{(e)}_{\alpha\beta} \\ Q_{\alpha} &= \sum_{e=1}^{E} Q^{(e)}_{\alpha} \end{aligned} \tag{2.40}$$

Dynamic System

For dynamic system we must also include mass matrix M introduced in Eq. 2.22. To complete the matrix form of computing K and Q in Eq. 2.37 we now provide the matrix equation for M:

$$M_{3N \times 3N} = \int_{\Omega} \rho \Phi^{T}_{3N \times 3} \Phi_{3N \times 3} dV \tag{2.41}$$

We compute the local mass matrix m_e using Eq. 2.41 over the Ω_e section. Assembly process is exactly the as in the case of stiffness matrix.

Sparse Matrix Representation

TODO show that Stiffness Mass and Damping matrices are sparse and symmetric.

There exists many methods for storing sparse matrices in efficient way. One of the most popular methods is called Compressed Sparse Row(CSR). This method is chosen becouse of its common usage in numerical libraries in CUDA API. We will learn about its programming usages later on. For now we provide the definition for CSR storage.

For simplicity we assume that a matrix M is square sparse $n \times n$ matrix. CSR format stores matrix M using three one-dimensional vectors A_V, A_I, A_J. Denote the number of nonzero elements in M by n_z. The vectors are constructed as follows:

1. The vector A_V is of length n_z and holds all the nonzero elements of M in left-to-right top-to-bottom(row-major) order.
2. The vector A_J is of length n_z. Contains the column indices in M of each element in A_V.
3. The vector A_I is of length $n+1$ and is defined as follows:
   (a) $A_I[0] = 0$
   (b) $A_I[i] = A_I[i-1]+$ (the number of nonzero elements on the $i-1$th row in M)

Example 2.1. Let M be a matrix:

$$M = \begin{bmatrix} 1 & -1 & 0 & -3 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix} \tag{2.42}$$

Matrix M in CSR format looks like the following:

Let M be a matrix:

$$\begin{aligned} A_V &= \begin{bmatrix} 1 & -1 & -3 & -2 & 5 & 4 & 6 & 4 & -4 & 2 & 7 & 8 & -5 \end{bmatrix} \\ A_J &= \begin{bmatrix} 0 & 1 & 3 & 0 & 1 & 2 & 3 & 4 & 0 & 2 & 3 & 1 & 4 \end{bmatrix} \\ A_I &= \begin{bmatrix} 0 & 3 & 5 & 8 & 11 & 13 \end{bmatrix} \end{aligned} \tag{2.43}$$

Solver

Implicit Euler

CG

CG with precondition

LU

Boundary Conditions

In both static and dynamic systems we end up with the system of linear equations of the form: $Ax = b$. In the case of static system, x is a vector of displacements. However in the implicit dynamic solver x is a vector of the first derivative of displacements, i.e. velicities. In any

case, applying the Dirichlet boundary conditions are done exacly the same, however they have a different meaning. In the former case we apply conditions directly on the values of displacements. In the latter, we apply conditions on their velocities. Method of applying the Dirichlet boundary conditions are presented.

Algorithm 3 Dirichlet BC for $Ax = b$

1: **procedure** Dirichlet BC
2: **for** each bc $\bar{x}_j$ **do**
3: Subtract from each ith member of b, the product of A_{ij} and x_j: $\hat{b_i} = b_i - A_{ij}\bar{x_j}$.
4: Zero the jth row and column of A: $\hat{A}_{ij} = \hat{A}_{ji} = 0$
5: Set $\hat{A_{jj}} = 1$
6: Set $b_j = \bar{x}_j$

Example 2.2. Consider the following system.

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{2.44}$$

After applying boundary conditions with a known value $\bar{x_3}$, the system becomes:

$$\begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 - A_{13}\bar{x_3} \\ b_2 - A_{23}\bar{x_3} \\ \bar{x_3} \end{bmatrix} \tag{2.45}$$

Finite Element - Tetrahedron

One of the most simple finite elements used in FEM models is a linear tetrahedron. Such element consists of four nodes and its shape functions are linear polynomials. This finite element is in particular interested for real time applications since no numerical integration are needed to construct element equations.

The tetrahedron is defined by four vertices with components x_i, y_i, z_i coordinates, $i = 1, 2, 3, 4$, six edges and four faces. For simplicity we can denote the component differences: $x_{ij} = x_i - x_j$, $y_{ij} = y_i - y_j$, $z_{ij} = z_i - z_j$ for $i, j = 1, 2, 3, 4$. The vertices can not be coplanar.

Volume

We can use the Jacobian matrix J to compute the volume V of the tetrahedron.

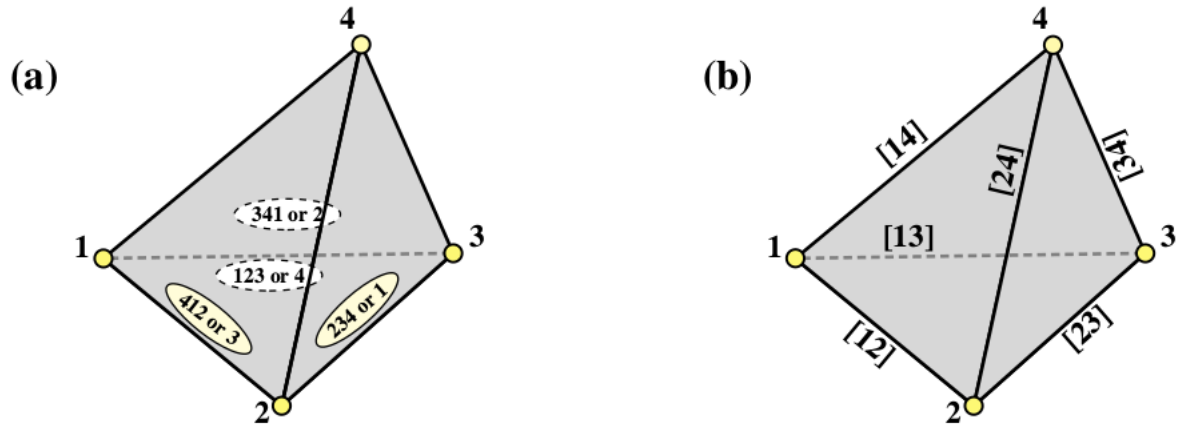
$$V = \int_{\Omega_e} d\Omega_e = \frac{1}{6} det \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} = \frac{1}{6} det(J) \tag{3.1}$$

The vertices are coplanar when V is equal to zero. The vertices have indices: $1, 2, 3, 4$. Edges are denoted by pair of indices e.g. 23 is an edge from vertex 2 to vertex 3. Faces are denoted by their opposite vertex or by triple of vertex indices that make up this face e.g. 1 or 234.

Natural Coordinates

So far we have specified the tetrahedron vertices in Cartesian coordinates x, y, z. An alternative coordinate system is called tetrahedral natural coordiantes and is composed of four functions: $\Phi^{(1)}(x), \Phi^{(2)}(x), \Phi^{(3)}(x), \Phi^{(4)}(x)$. They value of $\Phi^{(i)}$ is equal to i at vertex 1 and 0 in all other vertices. We add a contraint:

$$\Phi^{(1)} + \Phi^{(2)} + \Phi^{(3)} + \Phi^{(4)} = 1 \tag{3.2}$$



Rysunek 3.1: Naming conventions for tetrahedron faces (a) and edges (b). Courtesy of [6]

Transformation

We have defined a different coordinate system. However, all quantities such as displacement fields, strain or stress are expressed in Cartesian coordinate system. Thus we need construct a transformation between this two coordinate systems. We combine the indentity constraint in Eq 3.2 with the linear interpolation of natural coordinates i.e. $x = x_i\Phi^{(i)}$, $y = y_i\Phi^{(i)}$, $z = z_i\Phi^{(i)}$ to get the following matrix relation:

$$\begin{bmatrix} 1 \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} = \begin{bmatrix} \Phi^{(1)} \\ \Phi^{(2)} \\ \Phi^{(3)} \\ \Phi^{(4)} \end{bmatrix} \tag{3.3}$$

The inversion of the above system yeilds:

$$\begin{bmatrix} \Phi^{(1)} \\ \Phi^{(2)} \\ \Phi^{(3)} \\ \Phi^{(4)} \end{bmatrix} = \frac{1}{6V} \begin{bmatrix} 6V_{01} & y_{42}z_{32} - y_{32}z_{42} & x_{32}z_{42} - x_{42}z_{32} & x_{42}y_{32} - x_{32}y_{42} \\ 6V_{02} & y_{31}z_{43} - y_{34}z_{13} & x_{43}z_{31} - x_{13}z_{34} & x_{31}y_{43} - x_{34}y_{13} \\ 6V_{03} & y_{24}z_{14} - y_{14}z_{24} & x_{14}z_{24} - x_{24}z_{14} & x_{24}y_{14} - x_{14}y_{24} \\ 6V_{04} & y_{13}z_{21} - y_{12}z_{31} & x_{21}z_{13} - x_{31}z_{12} & x_{13}y_{21} - x_{12}y_{31} \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \\ z \end{bmatrix} \tag{3.4}$$

The first column is abbreviation for:

$$\begin{aligned} 6V_{01} &= x_2(y_3z_4 - y_4z_3) + x_3(y_4z_2 - y_2z_4) + x_4(y_2z_3 - y_3z_2) \\ 6V_{02} &= x_1(y_4z_3 - y_3z_4) + x_3(y_1z_4 - y_4z_1) + x_4(y_3z_1 - y_1z_3) \\ 6V_{03} &= x_1(y_2z_4 - y_4z_2) + x_2(y_4z_1 - y_1z_4) + x_4(y_1z_2 - y_2z_1) \\ 6V_{04} &= x_1(y_3z_2 - y_2z_3) + x_2(y_1z_3 - y_3z_1) + x_3(y_2z_1 - y_1z_2) \end{aligned} \tag{3.5}$$

It can be shown that $V = V_{01} + V_{02} + V_{03} + V_{04}$.

We can now write the matrix system in more compact way using further abbreviation for the other part of the 4×4 matrix:

$$\begin{bmatrix} \Phi^{(1)} \\ \Phi^{(2)} \\ \Phi^{(3)} \\ \Phi^{(4)} \end{bmatrix} = \frac{1}{6V} \begin{bmatrix} 6V_{01} & a_1 & b_1 & c_1 \\ 6V_{02} & a_2 & b_2 & c_2 \\ 6V_{03} & a_3 & b_3 & c_3 \\ 6V_{04} & a_4 & b_4 & c_4 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \\ z \end{bmatrix} \tag{3.6}$$

The explicit equation for $\Phi^{(i)}$ is given by:

$$\Phi^{(i)} = \frac{6V_{0i} + a_i x + b_i y + c_i z}{6V} \tag{3.7}$$

Derivatives

We can easly compute the following partial derivates

$$\begin{aligned} 6V \frac{\partial \Phi^{(i)}}{\partial x} &= 6V \Phi^{(i)}_{,1} = a_i \\ 6V \frac{\partial \Phi^{(i)}}{\partial y} &= 6V \Phi^{(i)}_{,2} = b_i \\ 6V \frac{\partial \Phi^{(i)}}{\partial z} &= 6V \Phi^{(i)}_{,3} = c_i \end{aligned} \tag{3.8}$$

Analytical Intergration

As mentioned before, intergration over the linear tetrahedron can be done analytically using the general formula:

$$\int_{\Omega_e} \Phi^{i(1)} \Phi^{j(2)} \Phi^{k(3)} \Phi^{l(4)} d\Omega_e = \frac{i!j!k!l!}{(i+j+k+l+3)!} 6V \tag{3.9}$$

Here the indices without brackets i, j, k, l represent the power exponent. Special cases that will be of intereset for us:

$$\begin{aligned} \int_{\Omega_e} d\Omega_e &= V \\ \int_{\Omega_e} \Phi^{(i)} d\Omega_e &= \frac{1}{4} V \\ \int_{\Omega_e} \Phi^{(i)} \Phi^{(j)} d\Omega_e &= \begin{cases} \frac{1}{10} V & i = j \\ \frac{1}{20} V & i \neq j \end{cases} \end{aligned} \tag{3.10}$$

We now have all components needed to compute local stiffness matrix $k_{12\times12}$ and local force vector $p_{12\times1}$. Our natural coordinates $\Phi^{(i)}$ will play a role of shape functions.

Stiffness Matrix

First, we will define geometric matrix B mentioned in Eq. 3.16 explicitely in terms of derivatives computed in Eq. 3.8:

$$B_{6\times12} = \frac{1}{6V}\begin{bmatrix} a_1 & 0 & 0 & a_2 & 0 & 0 & a_3 & 0 & 0 & a_4 & 0 & 0 \\ 0 & b_1 & 0 & 0 & b_2 & 0 & 0 & b_3 & 0 & 0 & b_4 & 0 \\ 0 & 0 & c_1 & 0 & 0 & c_2 & 0 & 0 & c_3 & 0 & 0 & c_4 \\ b_1 & a_1 & 0 & b_2 & a_2 & 0 & b_3 & a_3 & 0 & b_4 & a_4 & 0 \\ 0 & c_1 & b_1 & 0 & c_2 & b_2 & 0 & c_3 & b_3 & 0 & c_4 & b_4 \\ c_1 & 0 & a_1 & c_2 & 0 & a_2 & c_3 & 0 & a_3 & c_4 & 0 & a_4 \end{bmatrix} \tag{3.11}$$

We are going to use the Eq. 2.37 with integrals over Ω_e.

First let us compute $k^{(e)}$

$$k^{(e)}_{12\times12} = \int_{\Omega_e} B^T_{6\times12} C_{6\times6} B_{6\times12} dV \tag{3.12}$$

Since both B and C are constant i.e. do not depend on x we simply get:

$$k^{(e)} = VB^TCB \tag{3.13}$$

Force Vector

We split the force into two independent parts: body forces and traction forces.

$$p_{12\times1} = \int_{\Omega_e} \Phi^T_{3\times12} \hat{f}_{3\times1} dV + \int_{\partial\Omega_{\sigma_e}} \Phi^T_{3\times12} \hat{t}_{3\times1} dA \tag{3.14}$$

Body Forces

Body forces such as gravity, are defined are a single force vector $\hat{f} = \rho[f_1, f_2, f_3]$. This force is applied to all vertices of body and is weighted by the volume of the tetrahedron that the vertex belongs to. The body force is computed using the fist integral:

$$f^{(e)}_{12\times1} = \int_{\Omega_e} \Phi^T_{3\times12} \hat{f}_{3\times1} dV \tag{3.15}$$

Even if we assume that the body force $\hat{f}$ is constant, Φ is not constant. It depends on the shape functions $\Phi^{(i)}$ which in turn depends on x. Let us define Φ explicitely:

$$\Phi_{3\times 12} = \begin{bmatrix} \Phi^{(1)} & 0 & 0 & \Phi^{(2)} & 0 & 0 & \Phi^{(3)} & 0 & 0 & \Phi^{(4)} & 0 & 0 \\ 0 & \Phi^{(1)} & 0 & 0 & \Phi^{(2)} & 0 & 0 & \Phi^{(3)} & 0 & 0 & \Phi^{(4)} & 0 \\ 0 & 0 & \Phi^{(1)} & 0 & 0 & \Phi^{(2)} & 0 & 0 & \Phi^{(3)} & 0 & 0 & \Phi^{(4)} \end{bmatrix} \tag{3.16}$$

We can use the second analatical integral presented in Eq. 3.10. Assuming that the force $\hat{f} = \rho[f_1, f_2, f_3]$ is constant we recieve:

$$f^{(e)}_{12\times 1} = \frac{1}{4}\rho V[f_1, f_2, f_3, f_1, f_2, f_3, f_1, f_2, f_3, f_1, f_2, f_3]^T \tag{3.17}$$

Traction Forces

Traction forces simulate the effect of pressure load, e.g. load applied after collision. As opposed to body forces, the traction forces are applied to element faces along their unit normal vector. Thus, each tetrahedron element should have an input of four scalar traction force magnitudes p_i, $i = 1, 2, 3, 4$, for each face. Traction force for entire tetrahedron is result of a sum of four traction forces computed for each face. To compute traction force for face 1 or 234 with input magnitute p, we use the formula:

$$t^{(e)}_{12\times 1} = \frac{1}{3}pA_1[0, 0, 0, \bar{a_1}, \bar{b_1}, \bar{c_1}, \bar{a_1}, \bar{b_1}, \bar{c_1}, \bar{a_1}, \bar{b_1}, \bar{c_1}]^T \tag{3.18}$$

First we see that the vertex 1, opposite of face 234, has recieved total force equal to $[0, 0, 0]$. Then, all the other vertices recieve the load equaly spread among them $\frac{1}{3}pA_1$, where A_1 is the area of face 234. Further, the entire vector is directed along the dirction cosines $\bar{a_1} = \frac{a_1}{S_1}, \bar{b_1} = \frac{b_1}{S_1}, \bar{c_1} = \frac{c_1}{S_1}$, where $S_1 = \sqrt{a_1^2 + b_1^2 + c_1^2}$. To compute the area A_1 we can use the corss product property. We choose two directed vectors from face 234 coming from any of its corners e.g. $u_{32} = [x_{32}, y_{32}, z_{32}]$ and $u_{42} = [x_{42}, y_{42}, z_{42}]$. Then:

$$A_1 = \frac{1}{2}||u_{32} \times u_{42}||_2 \tag{3.19}$$

The procces is generalized for all face indices $i = 1, 2, 3, 4$.

Mass Matrix

Recalling the equation 2.41 for mass matrix M, we can compute the local mass matrix $m^{(e)}$ following the same steps as in the case of stiffness matrix.

$$m^{(e)}_{12\times 12} = \int_{\Omega_e} \rho \Phi^T_{3\times 12}\Phi_{3\times 12} dV \tag{3.20}$$

Using the equation 3.10 we receive:

$$m^{(e)}_{12\times12} = \frac{\rho V}{20}\begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 \end{bmatrix} \tag{3.21}$$

Similarly to stiffness, mass matrix is also symmetric.

Tetrahedralization

Creating Compulational Mesh

Implementation

CUDA

GPU Architecture

Pascal Architecture

Matrix Vector Multipliation

Linear System of Equations

CG

CG with precondition

LU

RTFEM

RTFEM Integration into Game Engine

Rendering

Collision

User Interface

Manual

Tests

Benchmarks

Float vs Double

Materials

Structures

Speed

Conclusions

Bibliografia

[1] Matthias Muller, Jos Stam, Doug James, Nils Thurey, *Real Time Physics Class Notes.*

[2] Jakub Ciecierski, *https://github.com/Jakub-Ciecierski/SoftBodySimulation*

[3] Michał Kleiber, Piotr Kowalczyk, *Wprowadzenie do Nieliniowej Termomechaniki Ciał Odkształcalnych.*

[4] Zienkiewicz *The Finite Element Method. Volume 1: The Basis.*

[5] Zienkiewicz *The Finite Element Method. Volume 2: Solid Mechanics.*

[5] Altair University *Practical Aspects of Finite Element Simulation.*

[5] Department of Civil and Environmental Engineering Duke University, Henri P. Gavin *Structural Element Stiffness, Mass, and Damping Matrices CEE 541. Structural Dynamics.*

[6] Department of Aerospace Engineering Sciences University of Colorado at Boulder *Advanced Finite Element Methods (ASEN 6367) Lectures https://www.colorado.edu/engineering/CAS/courses.d/AFEM.d/.*

Wykaz symboli i skrótów

nzw.	nadzwyczajny
*	operator gwiazdka
~	tylda

Spis rysunków

Spis tabel

Spis załączników

1. Załącznik 1
2. Załącznik 2

Załącznik 1, załącznik 2 – mają się znajdować na końcu pracy (to jest notka przypominająca)