Vniversitat de València

POSITION BASED CONSTRAINT ENFORCEMENT IN GAME PHYSICS

Facultat de Ciències Matemàtiques

Student: **Cristóbal Rodero Gómez**
Tutor: **Ignacio García-Fernández**

"At Pixar, we're all about telling stories, but one story that hasn't been told very much is the huge degree to which math is used in the production of our films."

Tony DeRose, Pixar Research Lead

"¿Café alguien?"

Ignacio García

Abstract

Mechanical systems simulation for video games and other interactive applications impose important restrictions as regards to stability, flexibility in the scenes and computational complexity. In the last few years several resolution strategies for mechanical systems with constraints have appeared. Some of the most popular ones in the development of video games use only the positions of the particles and a projection algorithm over the manifold defined by the constraints, avoiding manipulation of the system's first derivative (velocities). In this way, a great numerical stability is obtained. The main drawback of these methods is its dependence in non-physical parameters, so is hard to simulate a specific material. In this work we explain all the aforementioned methods and focus in the simulation of elastic materials taking as reference another model deeply studied that depends on real, physical parameters. We propose an algorithm to fit the non-physical parameters of the position based algorithm and test this procedure in a elastic cube. An extrapolation to other, more complex objects should not be difficult. As a last contribution we relate these algorithms with some classical numerical methods and point out which are the main hypothesis assumed in the process. This part, although is not very robust since we have not been able to reach a closed result, can be useful as a first step for future works dealing with the convergence topic of this kind of methods.

Acknowledgements

I would first like to thank my project tutor Ignacio García Fernández of the ETSE at Universitat de València. His door office was always open (often literally) whenever I ran into a trouble spot or had a question about a concept or writing. He consistently steered me in the right direction whenever he thought I needed it. I am extremely thankful and indebted to him for his patience, for sharing his expertise and for his sincere and valuable guidance.

Secondly, I must acknowledge all the CoMMLab team, and specially *the engineers* Pedro Zuñeda, Carlos Monteagudo and Miguel Lozano for all the simulations done and all the visual part that has come up with this work.

Obviously, I cannot forget about thanking Pau Real for his support over the year and with this project, our mathematical debates and the healthy competitiveness for learning more and more with each curiosity we bump into (*we know nothing*).

Finally, I would be very hurt if I forgot expressing my very profound gratitude to my mother for providing me with unfailing support and continuous encouragement throughout my years of study.

I also place on record, my sense of gratitude to one and all, who directly or indirectly, have lend me their hand in this venture.

This accomplishment would not have been possible without of all you.

Thank you.

Cristóbal

Contents

List of Figures

Nomenclature

Acronyms

CPU	Central Processing Unit.
FEM	Finite Elements Method.
FE	Finite Element.
GPU	Graphics Processing Unit.
MSM	Mass-Spring Model.
PBD	Position Based Dynamics.

Symbols

$\mathbf{D}$	The matrix which defines the relationship between strain and stress in the Saint Venant-Kirchhoff model. See Equation 2.6.
Δt	Time step of the method.
$\dot{x}$, x'	Derivative of x w.r.t. time.
$\Delta \mathbf{p}$	The correction (an increment) of the position of the particle.
$\mathbf{J}_{\mathbf{p}}$	The jacobian matrix evaluated in the position $\mathbf{p}$. That is, the element (i, j) is $(\partial C_i / \partial \mathbf{p}_j)(\mathbf{p})$ where $\mathbf{p}_j$ is the position of the particle j.
$\mathbf{K}^{\text{FEM}}$	Stiffness matrix of the FEM.

$\mathbf{K}^{\mathrm{PBD}}$	Stiffness matrix of PBD.
λ	A Lagrange multiplier.
$\nabla_{\mathbf{p}} C_j(\cdot)$	The gradient of he function C_j with respect the n coordinates of $\mathbf{p}$.
ν	Poisson's ratio. See definition 6.
$\partial_{\mathbf{p}_j} C_i$	$\frac{\partial C_i}{\partial \mathbf{p}_j}$.
$\rho(A)$	Spectral radius of the matrix A, the greatest modulus of the eigenvalues.
$\boldsymbol{\epsilon}$	Strain, a measure of deformation.
$\boldsymbol{\sigma}$	Stress, force per unit area.
$\mathbf{v}$	A vector. In general, if a symbol is in bold style will represents a vector.
$\mathbf{u}_i$	Nodal displacement of the vertex i.
$C_j(\cdot) : \mathbb{R}^{3n_j} \to \mathbb{R}$	A scalar constraint function.
E	Young's modulus. See definition 5.
k	Stiffness coefficient, characterize the rigidity of a spring between two nodes in a MSM or how strictly is a constraint accomplished in PBD. In the latter case is defined in $[0, 1]$.
k_e, k_f, k_d, k_v	Stiffness parameters of the edges, faces and diagonals distance constraint and the volume preservation constraint of a cube, respectively.
m_i	Mass of the particle/node i.
w_i	The inverse of the mass of the particle i. If we want an immovable particle, it is enough setting this parameter to 0.

Chapter 1

Introduction

The huge growth of computers capacity to process and solve complex problems has provided a new framework for mechanical simulation. Nowadays, ambitious new applications are being developed based on the computational simulation of soft bodies deformation. Medical simulation, for instance, is undergoing a great transformation since modern computers enable real-time workloads both in interactive and haptic environments. Additionally, this performance has been accompanied by an increase in accuracy as simulation methods evolve toward more detailed representation of model geometry and mechanical properties. In the particular case of computer graphics, it is very common to use physically based models to describe the response of some body to mechanical interactions.

The simulation of solid objects such as rigid bodies, soft bodies or cloth has been an important and active research topic for more than 30 years (more in engineering). The main goal of computer simulations in computational physics and chemistry is to replace real-world experiments and thus, to be as accurate as possible. For instance, in disciplines such as surgery to have efficient simulators is getting increasingly needed. Once the different regions captured with the medical imaging devices are identified, the 3D objects have to be translated into virtual mechanical objects so that the computer can simulate their physical behaviour when the user interacts with them in the virtual environment in a realistic way. In particular, it is important to represent realistically how the different soft tissues deform under the virtual forces applied in the interaction.

There are different techniques to create deformable models and each one of them allows specific ways of simulation. Some models are based on geometric approaches while others are physically based. For example, in surgical applications geometrical approaches are hardly seen since they do not provide accurate results and do not allow computing mechanical measures such as forces. Physically based approaches, in turn, are widely adopted in mechanical simulation field as they are much more versatile and accurate.

Physically based deformable models have two important characteristics: mechanical behaviour and topologic configuration. As each region belongs to different types of tissue it can show different behaviour and has to be characterised using different parameters. The 3D bodies are tesselated adopting, usually, cubes or tetrahedra. Mechanical parameters depend on the tissue type, deformation velocity and the extent of the deformation. Now, taking into account user interactivity, the aim of the simulation consists in studying the effects of these interactions over the deformable body. That is, the simulation routine has to evaluate the mechanical effect of the net forces applied on the body or specific parts of the body. To update the state of the model it is necessary to evaluate the deformation making kinematic and dynamics analysis. As these approaches usually require using time integration methods, the most common problems that accompany these procedures are mathematical convergence and stability issues.

Among the aforementioned methods we have two in which we will focus our attention in chapter 2 because they are some of the most used in literature and because will serve as a reference and as a body of comparison: Finite Element Method (FEM) and Mass-Spring Models (MSM). Although the first one is a method to solve partial differential equations in general, when applied to elastic models it creates force based simulation methods where deformations and forces are related. We will see this later in chapter 2.

More recently, these methods have been used in video games and digital animation, but the requirements in these fields are quite different. Besides realism and accuracy, a number of other criteria are also important in computer graphics applications. These criteria are mainly **generality**, **robustness**, **simplicity**, **performance** and **numerical convergence**. By *generality* we mean the ability to simulate a large spectrum of behaviors, such as different types of geometries (solids, shells, rods...), different material properties, or even art-directable extensions to classic physics based simulation. *Robustness* refers to the capability to adequately handle difficult configurations, including large deformations, degenerate geometries, and large time steps. Robustness is especially important in real-time applications where there is no second chance to re-run a simulation, such as in computer games or medical training simulators. The *simplicity* of a solver is often important for its practical relevance. Building on simple, easily understandable concepts and the resulting lightweight codebases eases the maintenance of simulators and makes them adaptable to specific application needs. *Performance* is a critical enabling criterion for real-time applications. However, performance is no less important in offline simulations, where the turnaround time for testing new scenes and simulation parameters should be minimized. And last but not least we must demand *numerical convergence*, since simulation is worthless if divergence occurs. Trying to fulfil these criteria a new method has emerged from the video game world: **Position Based Dynamics**. We explain it in detail in chapter 3, from its basis to some convergence works. Although it born in the video game and digital animation field its applications have spread through more other fields such training medical simulators or physics simulator in which interactivity is added.

All the above explained would be a sort of *state of the art*, i.e., all has been done and studied. This part is presented in chapter 2 and chapter 3. However, Position Based Dynamics is only physically inspired and it has parameter sets that do not correspond to the standard parameters in elasticity. To the best of our knowledge there is not any work done about this topic in Position Based Dynamics although a methodology has been exposed by Lloyd et al. and San Vicente for Mass-Spring Models in [20, 26, 27]. We explain this methodology in section 2.2 and apply it to the aforementioned problem of Position Based Dynamics in chapter 4, the part that would be the nucleus of this document. Our main objective will be to fit a set of Position Based Dynamics parameters which leads to an elastic behaviour consistent with given physical parameters. As the reference for a proper elastic behaviour we will consider an element simulated with the Finite Elements Method. We end obtaining some polynomial functions which relates the Position Based Dynamics parameters with the elasticity theory ones and conclude presenting some error plots in the comparison of FEM with Position Based Dynamics.

In the last section of that chapter, section 4.3, we formalise matricially the Position Based Dynamics (since its formulation can be done in a more general way) in order to try to analyse the convergence of this method by comparing it with some classical numerical methods. We emphasise that this part is not as conclusive as the previous work, but can serve as a first attempt to give Position Based Dynamics a rigorous mathematical base, seen as a root-finding iterative method.

In chapter 5 we recapitulate all the results of this project, gathering all possible future work that has come up along this work and remarking the original and own contributions.

Chapter 2

Simulation methods for elastic materials

As we said in the introduction this first part of the work is going to consist on an overview of some of the most popular methods for simulating elastic materials. We will start in this chapter with those based on forces, i.e., where a relationship is built between deformations and forces. As we will see, even though the first method (the Finite Element Method) is a very general one, suitable for a widely type of problems, we will focus on its performance over deformable bodies. After that, in chapter 3 we will present the main method of this project: Position Based Dynamics. In that chapter, we will see from Position Based Dynamics basic definitions until the work done in the convergence field and we will present some open problems.

The most intuitive approach in order to simulate a system might be if we base on physics' equations and theories. In particular, with the point of view of the dynamics: the classical mechanics branch concerned with the study of forces and their effect on motion. The main advantages of this kind of models is its accuracy, that it is straight-forward translate from real equations to implementation and the fact that they have been widely studied over this past decades. Beginning with the Finite Element Method, we present an overview of the method explained with one simple example and specify the underlying mathematics of this method; the second part will be the Mass-Spring Method where, in expenses of accuracy a more easy implementation and analysis in accomplished.

2.1 Finite Element Method

The Finite Element Method (FEM) can be presented as a numerical technique for finding approximate solutions to boundary value problems for partial differential equations, with very few demands of how this problem must be. FEM subdivides a large problem into smaller, simpler, parts, 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. FEM then uses variational methods from the calculus of variations to approximate a solution by minimizing an associated error function.

Regarding the history, as is often the case with original developments, it is rather difficult to quote an exact date of invention, but [1] traced back the roots of the FEM to three separate research groups: applied mathematicians [9], physicists [28] and engineers [14], although the FEM obtained its real impetus from the development of engineers.

Let's see how this method works in more detail. In order to do this in a more light way, we will use an elasticity example. As told in [6], in a discrete structure, its deformation is defined by a finite number of parameters (deformation modes) assembled in a vector. In the case of a continuous system we can't characterise the deformation with a finite vector, but with a vectorial function. This function is the solution of the differential equation that defines the problem, but a manipulable analytical expression of this can not be assured (maybe it has not even a closed form). In order to solve this problem, the FEM uses the discretisation hypothesis, based on:

- The continuum is divided by imaginary lines or surfaces in adjacent disjointed regions, simple geometrical figures, called **finite elements** (see Figure 2.1).
- Finite elements join each other in a finite number of points, called **nodes**.
- Nodes displacements are the basic variables of the problem, and determine univocally the deformed structure configuration.
- The displacement of an arbitrary point is determined by the displacement of those nodes belonging to the element in which the point is in. In order to do this, for each element are defined the so-called **shape functions**. This shape functions will have to guarantee the compatibility in boundary's elements. Both, shape functions and constitutive equations of the material, define the stress in the element.

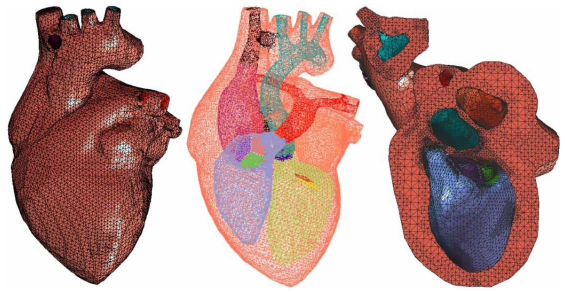

Figure 2.1: Finite element mesh of the human heart ([31])

So, the nucleus of the FEM are these shape functions and the main problem is to choose an adequate function.

2.1.1 A descriptive example

Now, in order to get a better understanding of how this method works we are going to solve with it a generic unidimensional differential equation.

Example 1. We wish to find the unknown function $y(x) : \mathbb{R} \to \mathbb{R}$ on some domain $x \in [x_1, \ldots, x_n]$. All we know is some differential equation

$$\frac{\partial^2 y(x)}{\partial x^2} = c(x)$$

where $c(x) : \mathbb{R} \to \mathbb{R}$ is some known function. For this equation, we must demand $y \in C^2(\Omega)$, that is, be twice differentiable in some open set Ω (where $[x_1, \ldots, x_n]$ is a subset). Furthermore, we are given some boundary condition

$$\begin{aligned} y(x_1) &= a \\ y(x_n) &= b \end{aligned}$$

Now we will do this using the FEM. First we multiply by an arbitrary trial function $v(x)$ defined such that we always have $v(x_1) = v(x_n) = 0$ and then we integrate over our domain

$$\int_{x_1}^{x_n} v(x) \left(\frac{\partial^2 y(x)}{\partial x^2} - c(x) \right) dx = 0$$

Next step would be to do the same to the boundary conditions and add all up into a single equation, but in this case are a bit trivial since they are point based. So we have

$$\int_{x_1}^{x_n} v(x) \left(\frac{\partial^2 y(x)}{\partial x^2} - c(x) \right) dx = \int_{x_1}^{x_n} v(x) \frac{\partial^2 y(x)}{\partial x^2} dx - \int_{x_1}^{x_n} v(x) c(x) dx = 0$$

Integration by parts of the first term give us

$$\left[v(x)\frac{\partial y(x)}{\partial x}\right]_{x_1}^{x_n} - \int_{x_1}^{x_n} \frac{\partial v(x)}{\partial x}\frac{\partial y(x)}{\partial x}dx - \int_{x_1}^{x_n} v(x)c(x)dx = 0$$

Cleaning up

$$\int_{x_1}^{x_n} \frac{\partial v(x)}{\partial x}\frac{\partial y(x)}{\partial x}dx + \int_{x_1}^{x_n} v(x)c(x)dx = 0 \tag{2.1}$$

This is the so-called **weak form** of the problem. So now $y(x)$ only needs to be continuously differentiable (instead of twice like in the beginning). Rather than finding y we wish to find a good approximation $\widetilde{y}$ such that

$$\widetilde{y}(x) = \sum_{i=1}^{n} N_i(x)\hat{y}_i = \overbrace{[N_1(x), \ldots, N_n(x)]}^{\mathbf{N}(x)} \overbrace{\begin{bmatrix}\hat{y}_1(x)\\ \vdots \\ \hat{y}_n(x)\end{bmatrix}}^{\hat{y}(x)} = \mathbf{N}\hat{y}$$

That is, we have broken up our continuous function into a set of n discrete values $(\hat{y}_i, x_i)$ that we somehow combine using some weighting/interpolation scheme given by the global **shape function** N_i. The shape function is subject to

$$N_i(x) = \begin{cases} 1 & x = x_i \\ 0 & x = x_j \wedge j \neq i \end{cases}$$

and

$$\sum_i N_i(x) = 1$$

So we had Equation 2.1, let us insert the approximation into our integrals

$$\int_{x_1}^{x_n} \frac{\partial v(x)}{\partial x}\frac{\partial \mathbf{N}(x)}{\partial x}\hat{y}dx + \int_{x_1}^{x_n} v(x)c(x)dx = 0$$

This is the main trick of the FEM, and here we see why is a method of approximation. So, depending on the strategy (there is not a standard one), we would choose now the shape function, the trial function, the discretization, add the boundary condition and would solve the problem by computing numerically the resulting integrals.

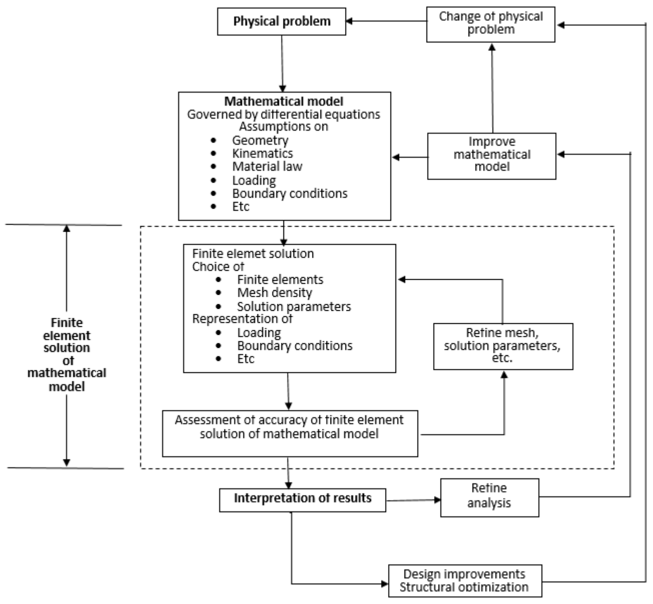

Figure 2.2: The process of finite element analysis

Although we have seen the method in a 1D-example it is no difficult to extrapolate a general form. What is not so trivial are the function's domains.

2.1.2 Functional analysis of the FEM

First of all we recall some definitions from several variables calculus. If there is not additional comment, Ω will be an open set and derivatives will be understood in the classical meaning:

Definition 1.

$$C^m(\Omega) = \big\{u(x)|u, u', \dots, u^{(m)} \text{ are continuous on } \Omega\big\} \quad 0 \le m \le \infty$$

Using the multi-index notation, we represent partial derivatives such as

$$D^\alpha u(x) = \frac{\partial^{|\alpha|}u}{\partial x_1^{\alpha_1}\cdots\partial x_n^{\alpha_n}}, \quad |\alpha| = \alpha_1 + \alpha_2 + \dots + \alpha_n, \quad \alpha_i \ge 0$$

Definition 2. $L^p(\Omega)$ space is defined as

$$L^p(\Omega) = \left\{u(x)\,\middle|\,\int_\Omega |u(x)|^p dx < \infty\right\}$$

Definition 3. The **Sobolev space** of general dimension is defined as

$$H^m(\Omega) = \{v(x) | D^\alpha v \in L^2(\Omega), \forall \alpha : |\alpha| \leq m\}$$

The inner product in $H^m(\Omega)$ is

$$(u, v)_{H^m(\Omega)} = (u, v)_m = \int \int_\Omega \sum_{|\alpha| \leq m} (D^\alpha u(x)) (D^\alpha v(x)) \, dx$$

Therefore, $H^m(\Omega)$ is a Hilbert space.

We need also

Theorem 1 (The Sobolev embedding theorem)**.** *If $2m > n$ then*

$$H^{m+j} \subset C^j, \qquad j = 0, 1, \ldots$$

where n is the dimension of the elements in the Sobolev space.

The proof of this theorem, although not of extreme difficulty, needs a wide background that would exceed the scope of this work. For the interested reader, the proof can be found in [10].

Now, if $u(x) \in C^1(0,1)$, then for any function $\phi \in C^1(0,1)$ such that $\phi(0) = \phi(1) = 0$ we recall

$$\int_0^1 u'(x)\phi(x) = u\phi|_0^1 - \int_0^1 u(x)\phi'(x)dx = -\int_0^1 u(x)\phi'(x)dx$$

where $\phi(x)$ is a test function in $C^1(\Omega)$ with $\phi(0) = \phi(1) = 0$. The first order **weak derivative** of $u(x) \in L^2(\Omega) = H^0(\Omega)$ is defined to be a function $v(x)$ satisfying

$$\int_\Omega v(x)\phi(x)dx = -\int_\Omega u(x)\phi'(x)dx \quad \forall \phi(x) \in C^1(\Omega) \text{ s.t. } \phi(0) = \phi(1) = 0$$

If such function exists, then we write $v(x) = u'(x)$. Similarly, the m-th order weak derivative of $u(x) \in H^0(\Omega)$ is defined as a function $v(x)$ satisfying

$$\int_\Omega v(x)\phi(x)dx = (-1)^m \int_\Omega u(x)\phi^{(m)}(x)dx \quad \forall \phi(x) \in C^m(\Omega)$$

$$\text{s.t. } \phi(x) = \phi'(x) = \ldots = \phi^{(m-1)}(x) = 0 \qquad \forall x \in \partial\Omega$$

If such function exists, then we write $v(x) = u^{(m)}(x)$

Now, returning to FEM we have the simple 1D model problem

$$-u'' = f, \qquad 0 < x < 1, \qquad u(0) = u(1) = 0$$

we know that the weak form is

$$\int_0^1 u'v'dx = \int_0^1 fvdx$$

Intuitively, because v is arbitrary we can take $v = f$ or $v = u$ to get

$$\int_0^1 u'v'dx = \int_0^1 u'^2dx \quad \text{or} \quad \int_0^1 fvdx = \int_0^1 f^2dx$$

so u, u', f, v and v' should belong to $L^2(0,1)$, i.e., we have $u, v \in H^1(0,1)$ so the solution is in the Sobolev space $H^1(0,1)$. From Sobolev embedding theorem, we also know that $H^1 \subset C^0$, so the solution is continuous. But, what happens with the discretization?

Definition 4. If the Finite Element (FE) space is a subspace of the solution space, then the FE space is called a **conforming FE space**, and the FE method is called a **conforming FE method**.

For example, the piecewise linear function over a given triangulation is a conforming FE space for the model problem. As [32] does, we will discuss only conforming FEM's. On including the boundary conditions, the solution space is defined as

$$H_0^1(0,1) = \left\{v \in H^1(0,1) | v(0) = v(1) = 0\right\}$$

When we look for a FE solution in a finite dimensional space V, it should be a subspace of $H_0^1(0,1)$ for a conforming FE method. For example, given a mesh for the 1D model, we can define a finite dimensional space using piecewise continuous linear function over the mesh

$$V = \{v, \, v(0) = v(1) = 0, \, v \text{ is continuous piecewise linear}\}$$

The FE solution would be chosen from the finite dimensional space V, a subspace of $H_0^1(0,1)$. If the solution of the weak form is in $H_0^1(0,1)$ but not in V, then an error is introduced on replacing the solution space with the finite dimensional space. Nevertheless, the FE solution is the best approximation in V in some norm. For seeing this, we need a little background.

We define, for two functions u and v

$$a(u,v) := \int_{x_1}^{x_n} (pu'v' + quv)\,dx$$

where $p(x), q(x) \in C([x_1, x_n])$. This is, indeed a bilinear form, because it is linear for both u and v from the following

$$\begin{aligned} a(\alpha u + \beta w, v) &= \int_{x_1}^{x_n} \left(p\left(\alpha u' + \beta w'\right) v' + q\left(\alpha u + \beta w\right) v\right) dx \\ &= \alpha \int_{x_1}^{x_n} (pu'v' + quv)\,dx + \beta \int_{x_1}^{x_n} (w'v' + qwv)\,dx \\ &= \alpha a(u,v) + \beta a(w,v) \end{aligned}$$

where α and β are scalars; and similarly,

$$a(u, \alpha v + \beta w) = \alpha a(u,v) + \beta a(u,w)$$

Since $a(u,v)$ is an inner product, under the conditions $p(x) \geq p_{\min} > 0$, $q(x) \geq 0$ (for some $p_{\min}$) we can define the *energy norm*

$$\|u\|_a = \sqrt{a(u,u)} = \left\{ \int_{x_1}^{x_n} \left(p\left(u'\right)^2 + qu^2 \right) dx \right\}^{\frac{1}{2}}$$

where the first term may be interpreted as the *kinetic energy* and the second term as the *potential energy*. The conditions for $p(x)$ and $q(x)$ guarantee that the Sturm-Liouville problem

$$\begin{aligned} -(p(x)u'(x))' + q(x)u(x) &= f(x) \quad x_1 < x < x_n && (2.2) \\ u(x_1) = u(x_n) &= 0 && (2.3) \end{aligned}$$

is well-posed, such that the weak form has a unique solution. The bilinear form often simplifies the notation for the weak form, *e.g.*, for the above Sturm-Liouville problem the weak form becomes

$$a(u,v) = f(v) \forall v \in H_0^1(x_1, x_n)$$

Now we are ready for the **error analysis for the FEM**. Error analysis for FEM usually includes two parts:

1. Error estimates for a given FE space.

2. Convergence analysis, a limiting process that shows the FE solution to the true solution of the weak form in some norm, as the mesh size h approaches zero.

We first recall some notation and setting up:

1. Given a weak form $a(u, v) = L(v)$ and a space V the problem is to find a $u \in V$ such that the weak form is satisfied for any $v \in V$. Then u is called the solution of the weak form.
2. A *finite dimensional* subspace of V denoted by V_h (i.e. $V_h \subset V$) is adopted for a conforming FEM and it does not have to depend on h, however.
3. The solution of the weak form in the subspace V_h is denoted by u_h, i.e. we require $a(u_h, v_h) = L(v_h) \qquad \forall v_h \in V_h$.
4. The global error is defined by $e_h = u(x) - u_h(x)$, and we seek a sharp upper bound for $\|e_h\|$ using certain norms.

It was noted that error is introduced when the finite dimensional space replaces the solution space, as the weak form is usually only satisfied in the sub-space V_h and not in the solution space V . However, we can prove that the solution satisfying the weak form in the subspace V_h is the best approximation to the exact solution u in the FE space in the energy norm.

Theorem 2. *1. u_h is the projection of u onto V_h through the energy inner product i.e.,*

$$u - u_h \perp V_h \quad \textit{or} \quad u - u_h \perp \phi_i, \quad i = 1, 2, \ldots, M$$

$$a(u - u_h, v_h) = 0 \forall v_h \in V_h \quad \textit{or} \quad a(u - u_h) = 0, \quad i = 1, 2, \ldots, M$$

where $\{\phi_i\}$ are the basis functions.

2. u_h is the best approximation in the energy norm, i.e.,

$$\|u - u_h\|_a \leq \|u - v_h\|_a, \quad \forall v_h \in V_h$$

Proof.

$$\begin{aligned} a(u, v) &= (f, v), \forall v \in V, \\ \Longrightarrow a(u, v_h) &= (f, v_h), \forall v_h \in V_h \text{ since } V_h \subset V, \\ a(u_h, v_h) &= (f, v_h), \forall v_h \in V_h \text{ since } u_h \text{ is the solution in } V_h, \\ \text{substracting} \Longrightarrow a(u - u_h, v_h) &= 0 \text{ or } a(e_h, v_h) = 0, \forall v_h \in V_h \end{aligned}$$

Now we prove that u_h is the best approximation in V_h

$$\begin{aligned}
\|u - v_h\|_a^2 &= a(u - v_h, u - v_h) \\
&= a(u - u_h + u_h - v_h, u - u_h + u_h - v_h) \\
&= a(u - u_h + w_h, u - u_h + w_h), \text{ on letting } w_h = u_h - v_h \in V_h \\
&= a(u - u_h, u - u_h + w_h) + a(w_h, u - u_h + w_h) \\
&= a(u - u_h, u - u_h) + a(u - u_h, w_h) + a(w_h, u - u_h) + a(w_h, w_h) \\
&= \|u - u_h\|_a^2 + 0 + 0 + \|w_h\|_a^2, \quad \text{since,} \quad a(e_h, u_h) = 0 \\
&\geq \|u - u_h\|_a^2
\end{aligned}$$

i.e., $\|u - u_h\|_a \leq \|u - v_h\|_a$ □

2.1.3 Elasticity in FEM

As we said before, is in our interest to apply the FEM's to the particular case of dealing with elasticity. In order to do this, we need a little of physics knowledge. More concretely, about **elasticity theory**.

Elasticity theory studies the relationship between forces applied on a body and the reversible deformation of the body. Constitutive models are often expressed as a relationship between strain $\boldsymbol{\epsilon}$, as a measure of deformation, and stress $\boldsymbol{\sigma}$, which is force per unit area. In linear elasticity, this relationship is of the form

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\epsilon} \tag{2.4}$$

where D expresses the elastic properties of the material. This is known also as Saint Venant-Kirchhoff model (seen for example in [15]).

The FEM methodology uses a discretisation of this model to turn it into a system of linear equations. In absence of deformation, node i has a position $\mathbf{x}_i$. If the system is deformed, with a nodal deformation $\mathbf{u}_i$ for node i, then, the position of every node will be $\bar{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{u}_i$. By means of the FEM, this relationship can be turn into a relationship between nodal deformations and generalised nodal forces $\mathbf{f}_i$.

We consider the vector $\mathbf{f}$, the applied force, and $\mathbf{u}$, the deformation of the system in static equilibrium . Then, according to [33] the FEM formulation defines a matrix $\mathbf{K}^{\text{FEM}}$ for which

$$\mathbf{f} = \mathbf{K}^{\text{FEM}}\mathbf{u}, \tag{2.5}$$

The FEM stiffness matrix for an element has the form

$$\mathbf{K}^{\text{FEM}} = \int_V \mathbf{B}^T\mathbf{D}\mathbf{B}\, dV$$

where V is the volume of the element, matrix $\mathbf{B}$ includes partial derivatives of the shape functions in the element and $\mathbf{D}$ is the matrix which defines the relationship between strain and stress, given by

$$\mathbf{D} = A \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & 1-2\nu & 0 & 0 \\ 0 & 0 & 0 & 0 & 1-2\nu & 0 \\ 0 & 0 & 0 & 0 & 0 & 1-2\nu \end{bmatrix} \tag{2.6}$$

where

$$A = \frac{E}{(1+\nu)(1-2\nu)} \tag{2.7}$$

Definition 5. Young's modulus (E), which is also known as the elastic modulus, is a mechanical property of linear elastic solid materials. It defines the relationship between stress (force per unit area) and strain (proportional deformation) in a material.

Young's modulus enables the calculation of the change in the dimension of a bar made of an isotropic elastic material under tensile or compressive loads. For instance, it predicts how much a material sample extends under tension or shortens under compression. The Young's modulus directly applies to cases of uniaxial stress, that is tensile or compressive stress in one direction and no stress in the other directions.

Definition 6. Poisson's ratio (ν), also known as the coefficient of expansion on the transverse axial, is the negative ratio of transverse to axial strain.

When a material is compressed in one direction, it usually tends to expand in the other two directions perpendicular to the direction of compression. This phenomenon is called the Poisson effect. Poisson's ratio is a measure of this effect. The Poisson ratio is the fraction (or percent) of expansion divided by the fraction (or percent) of compression, for small values of these changes.

With this two parameters an elastic solid material is completely characterised.

2.1.4 Final remarks

With all this, as [26] remarks, the FEM is a mathematical tool that allows solving a wide range of problem, from deformation studies to heat transfer. The simplest framework corresponds to the linear elasticity assumption (we developed its formulation in the introduction) which gives the advantage of applying well-known techniques to accelerate the computation and work in real time. This assumption is the one interesting for us, since we want to characterise another method (Position Based Dynamics) for this situation. However, realistic surgical simulators, e.g., require handling large displacement and deformations. Nonlinear FEM approaches are more suitable but with the drawback that their formulation is more complex and the techniques adopted to accelerate linear FEM are no longer valid when accurate results are desired. Although a keen effort has been done to develop real-time applications using nonlinear FEM, they still face the problem of handling topological changes in real time because, in some cases, this makes avoiding the advantages of preprocessing and evaluating new integrals.

2.2 Mass-Spring Model

Mass-Spring Models (MSMs) are physically based models with simple structure and relatively small computational cost. Consequently, many operators like large deformations and topology modifications can be simulated easily. These characteristics make them suitable for interactive-time applications and parallel computing. Therefore, is very common to find them in many applications involving facial animation, artificial animal animation, cloth simulation, biomechanical analysis and surgery simulation. Again, we make a rough overview of the method but shorter, since its complexity in an introductory level is much lower than with the FEM.

In this kind of models the object is modelled as a collection of point masses linked by springs in a lattice structure (see Figure 2.3).

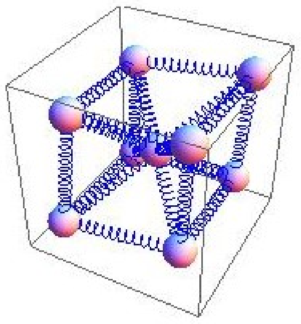

Figure 2.3: Schematical representation of a MSM of adeformed cube with nine vertexes ([7]).

Following the idea of classical mechanics, MSMs represent a body by a single or multiple point masses that have no extension and hold the complete mass of the body. The external forces applied to the body are concentrated in the point masses as well.

These models include a mesh of springs that interconnect those point masses representing the elastic behaviour of the bodies. The spring mesh can have many different configurations depending on the geometry of the object and the topology selected to represent the elasticity properties. When the model interacts with the environment new boundary conditions appear on the surface of the body. This new conditions cause the cange of the length of the springs inducing forces that act over each point mass of the MSM.

Typically, the elastic connection between the point masses is modelled using Hookean or linear springs, that is, the force exerted by the spring that connects two generic nodes i and j is proportional to the elongation of the springs, $k_{(i,j)}$ being the corresponding stiffness coefficient and $l^0_{(i,j)}$ its rest length. Thus, the resulting connection force is:

$$\mathbf{f}_{(i,j)} = k_{(i,j)} \left(\|\mathbf{p}_i - \mathbf{p}_j\| - l^0_{(i,j)} \right) \frac{\mathbf{p}_i - \mathbf{p}_j}{\|\mathbf{p}_i - \mathbf{p}_j\|}$$

The equilibrium shape of the deformable body is always reached when the sum of all forces acting over each node are null. If only the final equilibrium state of the MSM is required then, it is enough to define the stiffness values of the springs. However, if the transition from rest state to the deformed state is also required then, it is necessary to study the dynamic evolution of the body and usually a damping factor is added to the model in order to improve the stability and performance of the system. This is done because in each step of the simulation the system gains energy, so it has to be dumped for a properly and realistic behaviour.

When some point mass is moved, the springs attached to it are strained and forces are exerted on adjacent point masses. In the dynamic simulations these forces induce acceleration to those point masses and change their possition accordingly. The acceleration of a generic point mass i with mass m_i is governed by Newton's Secod Law:

$$m_i \ddot{\mathbf{x}}_i - c_i \dot{\mathbf{x}}_i - \sum \mathbf{f}_{(i,j)} = \sum \mathbf{F}_i^{ext}$$

where c is the damping coefficient, $f_{(i,j)}$ the force of the spring that connects particles i and j and $\mathbf{F}_i^{ext}$ the external forces acting over the i-th particle. The term of the damping effect can be different depending on authors because some of them consider just the velocity of the particle while others the relative velocity with respect to the neighbour particles.

As solving these systems might be extremely difficult, it is typical to use the following numerical integration methods: explicit (e.g. forward Euler and Runge-Kutta), implicit (backward Euler) and combined methods. Generally speaking the explicit methods are simple but need small time steps for stability while the implicit methods are more complex and stable.

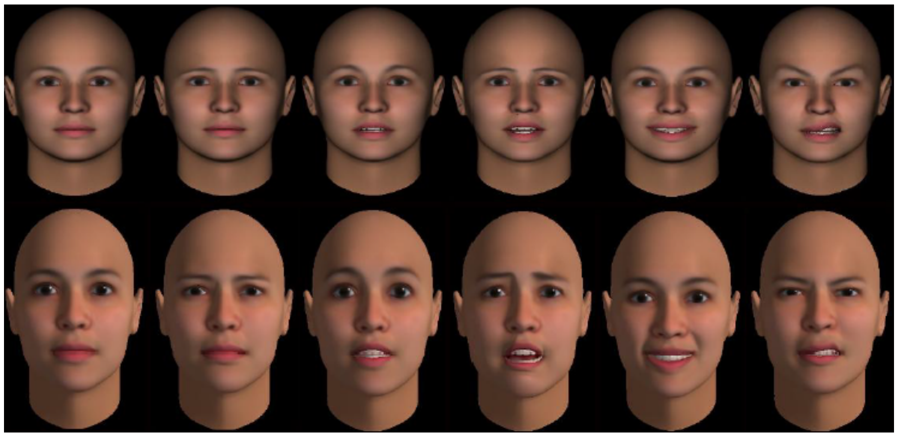

Figure 2.4: A facial animation example using MSMs, probably one of the first applications using MSMs ([30]).

Now, since in chapter 4 we are going to follow mostly the work of [20] we find appropriate explain some details and conclusions of this work.

Four steps are necessary to derive the spring coefficients of a single cubical MSM before the whole body is assembled:

1. The element stiffness matrix must be calculated analytically using the FEM.

2. The equations of the MSM have to be obtained.

3. These equations are linearised and the linear stiffness matrix is calculated.

4. The stiffness matrices of both MSM and FEM are compared.

Normally, a linearised MSM with the same stiffness matrix as the linear FEM of the same geometry cannot be found, making it necessary to find the parameters that minimize the difference between them.

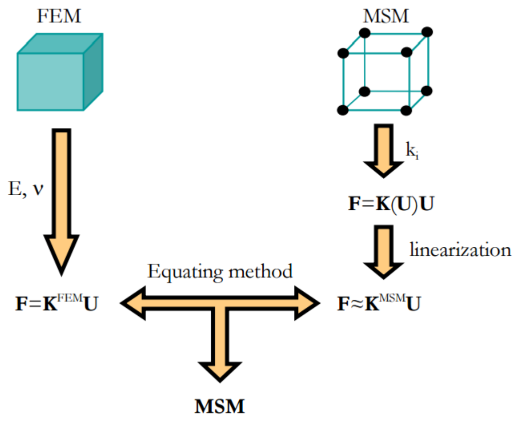

Figure 2.5: Method to obtain a cubical MSM from linear elastic FEM. Figure extracted from [26]

In order to build an hexaedral element in MSM equivalent to one in FEM, we follow the traditional convention. That is, the element is represented by an ordered set of vertices from 1 to 8, where the point masses are placed. These vertices are connected to each other by linear springs. Three types of springs are defined to connect each of the vertices of the cube with the remaining seven vertices: edge springs with k_e stiffness (e.g. between vertices 1 and 2), face diagonal spring with k_f stiffness (e.g. between vertices 1 and 6) and internal diagonal springs with k_d stiffness (e.g. between vertices 1 and 7). The graphic representation of the different springs and the node numbering are shown in Figure 2.6.

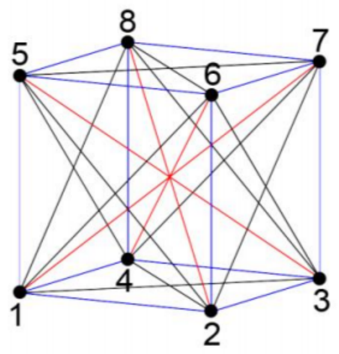

Figure 2.6: Node numbering and spring distribution in the cube. Edge springs in blue, face diagonal springs in black and internal diagonal springs in red.

Now, the first step of the aforementioned algorithm has been explained in the previous section. The second one has been developed some lines above, when Hooke's Law has been mentioned. Although the relation between the force and the elongation is constant, the evaluation of this force leads to a nonlinear formulation. The reason is that the computation of the length of the spring in the deformed state is a Euclidean distance. Thus, the assembly of the 28 springs that compose the cube produces a system of nonlinear equations. The process concerning the assembly of the equations of the 28 springs that compose the single cube is straightforward and will not be detailed. In any case, this linearisation (made with Taylor developments of the forces around the rest state) allows computing approximately the equilibrium of a MSM under the hypothesis of small deformations around the initial rest position with a system of equations of the following form:

$$\mathbf{F} = \mathbf{K}^{\mathrm{MSM}}\mathbf{U}$$

$\mathbf{F}$ is the vector of 24 forces applied at the nodes (3 components for each node), $\mathbf{U}$ is the vector of the displacements of the nodes and $\mathbf{K}^{\mathrm{MSM}}$ is the symmetric 24x24 stiffness matrix obtained from the linearisation. This matrix has a structure of 8×8 blocks, each block being a 3×3 matrix. The blocks located in the diagonal of the matrix correspond to the sum of the spring forces exerted at each node. The blocks not contained in the diagonal, in turn, define the stiffness relation between only two particular nodes.

2.2.1 Matrix distance approach

For the tuning of the stiffness parameters in order to get a realistic elastic behaviour in a material two approaches can be followed:

Definition 7. Data driven approaches use a set of reference deformations, and rely on different optimisation methods to find the parameters for the MSM. Typically the function to optimise is some error measure of the deformation of a simulated MSM, compared to the reference deformation samples.

Definition 8. Analytical approaches try to develop expressions that involve both the Mass Spring parameters and the elastic parameters describing a deformable material.

Van Gelder [12], following the former strategy, linearised the system of equations of the MSM to find that, in general, its stiffness matrix cannot be directly equated to a linear FEM stiffness matrix. More recently, Lloyd et al. [20] derived analytic expressions for triangular (for 2D) and tetrahedral (for 3D) Mass-Spring elements and find that the closed solution can only be found on equilateral triangles. However, as the stiffness parameters of the MSM are not enough to equate both matrices they increased the number of degrees of freedom of the problem by considering also as output variable the Poissons ratio. In the particular case of triangles and rectangles, the Poissons ratios that make the submatrices of $\mathbf{K}^{\text{FEM}}$ symmetric are $\frac{1}{3}$ and $\frac{1}{4}$ respectively. In the case of rectangles they even had to increase the number of variables by considering prestrained springs. In other words, they force $\mathbf{K}^{\text{FEM}}$ to have the same form as the stiffness matrix of the linearised MSM by selecting a particular value for Poissons ratio. However, including ν in the set of design parameters limits the usefulness of the method because it admits as input parameters any value of E but a unique ν. Therefore, this method is able to find MSMs equivalent to only certain linear elastic material models. In the case of regular tetrahedra, apart from the three stiffness parameters and ν, they included volume preserving forces. However, the equating problem still did not yield an exact solution. Therefore, they proposed a minimization approach to obtain the best possible approximations, and divided the method into two steps: first the value of ν that makes both stiffness matrices have similar form is selected and then the minimization is performed. Calling $\mathbf{C}$ to the difference between the linearised MSM and FEM stiffness matrices the optimization strategy consists in minimizing the Frobenius norm of $\mathbf{C}$:

$$\begin{aligned} \mathbf{C} &= \mathbf{K}^{\text{FEM}} - \mathbf{K}^{\text{MSM}} \\ \min \|\mathbf{C}\|_F &= \min \sqrt{\sum_{i=1}^{24}\sum_{j=1}^{24} |c_{ij}|^2} \end{aligned}$$

The choice of the Frobenius norm for the minimisation problem can be understood if we think about eigenvalues and eigenvectors. Taking as reference the stiffness matrix of the FEM, the study of the eigenproblem provides the $\mathbf{V}$ and $\mathbf{D}$ square matrices that satisfy the following equation:

$$\mathbf{K}^{\text{FEM}}\mathbf{V} = \mathbf{V}\mathbf{D}$$

The columns of $\mathbf{V}$ are the eigenvectors ($\mathbf{v}_n$) of $\mathbf{K}^{\text{FEM}}$ and $\mathbf{D}$ is a diagonal matrix whose values correspond to the eigenvalues (λ_n) of $\mathbf{K}^{\text{FEM}}$.

As both the stiffness matrices of the FEM and the linearised MSM are symmetric, their eigenvectors are orthogonal. This means that the 24 eigenvectors of each matrix form an orthogonal basis of the space of $\mathbf{U}$. Analysing the transformations suffered in those directions is equivalent to studying all the possible displacements vectors since the rest of them are linear combinations of this basis.

Consequently, analysing the eigenproblem of the stiffness matrices is equivalent to studying the effects of applying different deformations in the specific directions that define the whole behaviour of the model. The error obtained in the MSM when performing the linear transformation in the principal directions defined by $\mathbf{K}^{\text{FEM}}$ can be computed as follows:

$$\mathbf{e}_n\left(k_e, k_f, k_d\right) = \mathbf{K}^{\text{MSM}}\mathbf{v}_n - \mathbf{K}^{\text{FEM}}\mathbf{v}_n \qquad n = 1, \ldots, 24$$

The strategy proposed (and followed by [26]) to optimize these transformations is the minimization of ϕ using a least squares method, ϕ being:

$$\phi\left(k_e, k_f, k_d\right) = \sum_{n=1}^{24} \left\|\mathbf{e}_n\right\|^2$$

Proposition 3. *This new strategy based on the study of the eigenvectors of* $\mathbf{K}^{FEM}$ *is consistent with the minimization of the Frobenius norm of difference between* $\boldsymbol{K}^{MSM}$ *and* $\mathbf{K}^{FEM}$

Proof. Indeed, taking into account that $\mathbf{C}$ is a real matrix and $\mathbf{V}$ is orthogonal ($\mathbf{V}^T = \mathbf{V}^{-1}$):

$$\begin{aligned}
\phi\left(k_e, k_f, k_d\right) &= \sum_{n=1}^{24} \left\|\mathbf{K}^{\text{MSM}}\mathbf{v}_n - \mathbf{K}^{\text{FEM}}\mathbf{v}_n\right\|^2 = \left\|\mathbf{K}^{\text{MSM}}\mathbf{V} - \mathbf{K}^{\text{FEM}}\mathbf{V}\right\|_F^2 \\
&= \left\|\mathbf{C}\mathbf{V}\right\|_F^2 = tr\left(\left[\mathbf{C}\mathbf{V}\right]^*\left[\mathbf{C}\mathbf{V}\right]\right) = tr\left(\left[\mathbf{C}\mathbf{V}\right]^T\left[\mathbf{C}\mathbf{V}\right]\right) \\
&= tr\left(\mathbf{V}^T\left[\mathbf{C}^T\mathbf{C}\mathbf{V}\right]\right) = tr\left(\mathbf{C}^T\mathbf{C}\mathbf{V}\mathbf{V}^T\right) = tr\left(\mathbf{C}^T\mathbf{C}\right) = \left\|\mathbf{C}\right\|_F^2
\end{aligned}$$

□

2.2.2 Final remarks

Generally speaking, MSMs are easy to construct, physically based models. They allow real-time simulations even when the model has to handle user interactions that involve topology changes. Another well-known advantage is their ability to deal with both large displacements and large deformations. Additionally, their discrete formulation allows adopting easily parallel computing. However, MSMs have also some drawbacks. These models are tuned through their spring parameters and it is difficult to find methods to assign proper values for these constants (recently, [26] achieved this comparing MSM with FEM, concluding that it is not a realistic method in most of situations). Furthermore, although it has been done in particular cases, it is difficult to express certain constraints (like incompressibility and anisotropy) in a natural way. Other problems are that they cannot capture effects including volume conservation or prevention of volume inversion. Also, we have problem with numerical instability, because in the cases we are far from the equilibrium position the solution diverges.

2.3 Strengths and weaknesses of the force based simulation methods in video games

Once these methods have been presented a natural question is if they are appropriate in the context of this work. Thus, we have to specify the field where we are moving on: real-time, with user interaction (most famous examples are simulators and video games). On the one hand, the advantages of the FEM are its accuracy, since it is based on real physics and the transliteration of most equations describing a problem is immediate, but they have the inconvenient of being very expensive computationally and hence not suitable for dynamic situations with unforeseeable behaviours. The latter problem is solved with MSMs as they are really understandable and easily implementable models. But they have not the advantage of the FEM: they are not so accurate or based directly in physics, so some constraints or situation have no direct transliteration on these models. Moreover, if a large deformation is done the model becomes unstable and numerically diverges.

In order to overcome this problems some authors have proposed an easily implementable, unconditionally stable method: Position Based Dynamics. This method was born in the *fabless* semiconductor company *Ageia*, now property of the American technology company *NVIDIA*, and it has become a very popular technique in Computer Graphics literature. We are going to explain this technique deeply in the next chapter.

Chapter 3

Position Based Dynamics

In all the explained cases in the previous pages, forces are computed due to their velocities and the actual deformation of the given mesh (either mass-spring or a finite element) but, if we want to control the positions in real-time (v.g. because user moves a particle) is often more desirable to work directly with positions, avoiding integration and other problems such as energy gain. This is what Position Based Dynamics (PBD) consists on. Some advantages of using PBD are

- Removes the typical instability problems (in FEM sometimes also happens).
- The objects themselves can directly be manipulated during simulation (FEM and MSM also achieve it).
- The formulation of the algorithm allows the handling of general constraints (only FEM achieves it).
- The explicit position based solver is easy to understand and implement (only MSM achieves it).

In this chapter we will explain what PBD consists on, from its original formulation until an acceleration method for convergence going through some alternative points of view of this method.

3.1 Notation and basic definitions of PBD

We are going to see the method and the algorithm developed by Müller et al. in [23]. In order to simulate any situation, we will have a set of n particles or vertexes (we are going to treat them indistinctly) and a set of m constraints. In each case, we have:

For each particle i	
m_i	mass
$\mathbf{p}_i,\ \mathbf{x}_i$	position
$\mathbf{v}_i$	velocity

For each constraint j	
n_j	cardinality
$C_j : \mathbb{R}^{3n_j} \to \mathbb{R}$	scalar constraint function
$\{i_1, \ldots, i_{n_j}\}$, $i_k \in [1, \ldots, n]$	set of indices
$k_j \in [0, 1]$	stiffness parameter
unilateral or bilateral	type of the restriction

Clearly, positions and velocities are vectors with three components each one (because we are in a three-dimensional space). With a cardinality n_j we mean a subset of the n variables that constraint j applies on. Constraint j with type *unilateral* is satisfied if $C_j(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_{n_j}}) \geq 0$ and if it is type *bilateral* if $C_j(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_{n_j}}) = 0$. The stiffness parameter k_j defines the *strength* of the restriction (from 0 to 1). Based on this data and a time step Δt, the dynamic object is simulated as exposed in Algorithm 1.

Line (2) initializes the state variables. We create the w_i variable as the mass inverse because thus we can make an unmovable object just setting $w_i = 0$. It is the same to say that we have an object with infinite mass, no matter how much force you apply, it can't be moved. But instead of treating with infinities is more desirable numerically treating with zeros. We do not have problems with dividing by zero because an object with no mass is not interesting for computer animation.

In line (6) we set the new velocity after a time step. It is a explicit forward Euler integration step on the velocities but adding external forces, mainly gravity (but also the user's interactions). It can be used if we cannot convert some force to positional constraint. Some authors like [18] pointed out that PBD can be interpreted as a heuristic variant of the variational implicit Euler method taking the inertial term out of the solver and into the integration step of the simulation. The main difference with respect to a simple explicit forward Euler integration step is the non-addition of energy. With a explicit step, usually an amount of energy is added to the system creating the sensation of a instability problem. This is avoided with the semi-implicit step.

Line (8) is used to avoid energy gain problems. In real world, energy is reduced due to heat dissipation or due to the contact with other objects but in this algorithm we

Algorithm 1 Algorithm of Position Based Dynamics

```
1: for all vertices $i$ do
2:     $\mathbf{x}_i = \mathbf{x}_i^0, \quad \mathbf{v}_i = \mathbf{v}_i^0, \quad w_i = 1/m_i$
3: end for
4: loop
5:     for all vertices $i$ do
6:         $\mathbf{v}_i = \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
7:     end for
8:     dampVelocities($\mathbf{v}_1, \ldots, \mathbf{v}_n$)
9:     for all vertices $i$ do
10:        $\mathbf{p}_i = \mathbf{x}_i + \Delta t \mathbf{v}_i$
11:    end for
12:    for all vertices $i$ do
13:        GenerateCollisionConstraints($\mathbf{p}_i$)
14:    end for
15:    loop
16:        projectConstraints($C_1, \ldots, C_{m+m_{Coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_n$)
17:    end loop
18:    for all vertices $i$ do
19:        $\mathbf{v}_i = (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
20:        $\mathbf{x}_i = \mathbf{p}_i$
21:    end for
22:    velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_n$)
23: end loop
```

have to simulate these effects by damping the velocities every time we calculate them. It can be done roughly by a step like $\mathbf{v}_{new} = 0.99\,\mathbf{v}_{old}$ (has been proved empirically that it works quite fine).

Line (10) is again a simple semi-implicit forward Euler integration but, this time, on the positions. It is not the final position, only a prediction without taking in account the restrictions.

Non-permanent external constraints (m_{coll} constraints) such as collision ones are generated in line (13). They change from time step to time step.

Line (16) is where the main solver of PBD happens, the constraint projection loop. It establishes which constraints are not satisfied and corrects one by one the predicted positions. The involved parameters are the number of iterations, $solverIteration$, the predicted positions, and both types of constraints: fixed and collision constraints. We will explain these solver deeply in a following section.

Because final positions can be different from predicted ones, velocities need to be updated in line (19). Along with them, positions are updated according to the state of the current system (line (20)). Finally, in line (22), the velocities of colliding vertexes are modified according to friction and restitution coefficients.

In this moment we note that, because the integration step is performed by a semi-implicit forward Euler integration, the solver is **unconditionally stable**.

3.2 PBD solver

The goal of the solver step (9)-(11) is to correct the predicted positions $\mathbf{p}_i$ of the particles such that they satisfy all constraints (if they are not contradictory). The problem that needs to be solved comprises of a set of $m + m_{coll}$ equations for the $3n$ unknown position components. If $m + m_{coll} > 3n$ ($m + m_{coll} < 3n$) the system is over-determined (underdetermined). In addition, the equations are in general non-linear. The function of a simple distance constraint $C(\mathbf{p}_1, \mathbf{p}_2) = \|\mathbf{p}_i - \mathbf{p}_j\| - d$ yields a non-linear equation. What complicates things even further is the fact that collisions produce inequalities rather than equalities. Solving a non-symmetric, non-linear system with equalities and inequalities which can be over-determined or underdetermined is a tough problem as seen in [8].

We are going to deal with equalities in all the coming work. In the case of inequalities is enough to check if the inequality is accomplished and if it is, we avoid the step. Thus, we can write the restrictions as

$$C_j(\mathbf{p}) = 0 \qquad j = 1, \ldots, m$$

where $C(\mathbf{p}) := C(\mathbf{p}_1, \ldots, \mathbf{p}_n)$. For now and on we will avoid the cardinality, assuming that the constraint are applied over all the n particles. If it is not the case we can see it as like if we sum the particle multiplied by 0. The process starts with a first guess of a solution. Each constraint function is then linearised in the neighborhood of the current solution using Taylor series. For a concrete point $\mathbf{p}_i$:

$$C(\mathbf{p} + \Delta\mathbf{p}) = C(\mathbf{p}) + \nabla^T_{\mathbf{p}_i} C(\mathbf{p}) \cdot \Delta\mathbf{p}_i + \mathcal{O}\left(\|\Delta\mathbf{p}_i\|^2\right) = 0$$

Here, $\Delta\mathbf{p}_i$ is the correction of the position of the particle, so we set $C(\mathbf{p} + \Delta\mathbf{p})$ to 0 in order to accomplish the restriction. This yields a **linear** system for the global correction vector $\Delta\mathbf{p}_i$

$$\nabla^T_{\mathbf{p}_i} C_j(\mathbf{p}) \cdot \Delta\mathbf{p}_i = -C_j(\mathbf{p}) \qquad j = 1, \ldots, m$$

where $\nabla_{\mathbf{p}_i} C_j(\mathbf{p})$ is the $n \times 1$ dimensional vector containing the derivatives of the function C_j with respect the n coordinates of $\mathbf{p}_i$ and evaluated in $\mathbf{p}$. Both $\nabla^T_{\mathbf{p}_i} C_j(\mathbf{p})$ and the right hand side scalars $-C_j(\mathbf{p})$ are constant because they are *evaluated* at the location $\mathbf{p}$ before the system is solved. When $m = 3n$ (a square matrix) and only equalities are present, the system can be solved by any lineal solver (only if the system is compatible, of course). If we have the case where the problem is under-constrained but with full rank we can use the pseudo-inverse to solve it. In other case is not that easy and can be useful to use Position Based Dynamics.

We make the linearisation

$$C_j(\mathbf{p} + \Delta\mathbf{p}) = C_j(\mathbf{p}) + \left(\nabla_{\mathbf{p}_i} C_j(\mathbf{p})\right)^T \cdot \Delta\mathbf{p}_i + \mathcal{O}\left(\|\Delta\mathbf{p}_i\|^2\right) = 0 \tag{3.1}$$

and, in order to conserve momenta (we will see it later) we restrict to

$$\Delta\mathbf{p}_i = -\lambda \nabla_{\mathbf{p}_i} C_j(\mathbf{p}) \tag{3.2}$$

So, substituting equation (3.2) into (3.1) (without taking in account the term $\mathcal{O}\left(\|\Delta\mathbf{p}_i\|^2\right)$), solving for λ and substituting it back into equation (3.2) yields the final formula for $\Delta\mathbf{p}_i$

$$\Delta\mathbf{p}_i = -\nabla_{\mathbf{p}_i} C_j(\mathbf{p}_1, \ldots, \mathbf{p}_n) \frac{C_j(\mathbf{p}_1, \ldots, \mathbf{p}_n)}{\sum_k \|\nabla_{\mathbf{p}_k} C_j(\mathbf{p}_1, \ldots, \mathbf{p}_n)\|^2} \qquad j = 1, \ldots, M \tag{3.3}$$

In case we want to consider particles with distinct masses or non-unitary masses we weight the corrections $\Delta\mathbf{p}_i$ by the inverse mass $w_i = 1/m_i$, so the equation (3.3) would be

$$\Delta\mathbf{p}_i = -\nabla_{\mathbf{p}_i} w_i C_j(\mathbf{p}_1, \ldots, \mathbf{p}_n) \frac{C_j(\mathbf{p}_1, \ldots, \mathbf{p}_n)}{\sum_k w_k \|\nabla_{\mathbf{p}_k} C_j(\mathbf{p}_1, \ldots, \mathbf{p}_n)\|^2} \qquad j = 1, \ldots, M$$

There are several ways of incorporating the stiffness parameter. The simplest variant is to multiply the corrections $\Delta\mathbf{p}$ by $k \in [0,1]$. However, for multiple iteration loops of the solver, the effect of k is non-linear. The remaining error for a single distance constraint after n_s solver iterations is $\Delta\mathbf{p}(1-k)^{n_s}$. To get a linear relationship we multiply the corrections not by k directly but by $k' = 1-(1-k)^{1/n_s}$. With this transformation the error becomes $\Delta\mathbf{p}(1-k')^{n_s} = \Delta\mathbf{p}(1-k)$ and, thus, becomes linearly dependent on k and independent of n_s as desired. However, the resulting material stiffness is still dependent on the time step of the simulation. Real time environments typically use fixed time steps in which case this dependency is not problematic.

Projecting a set of points according to a constraint means moving the points such that they satisfy the constraint. The most important issue in connection with moving points directly inside a simulation loop is **the conservation of linear and angular momentum**. Let $\Delta\mathbf{p}_i$ be the displacement of vertex i by the projection. Linear momentum is given by

$$\vec{\mathbf{P}} = \sum_i m_i \mathbf{v}_i = \sum_i \Delta p_i$$

where the last equality is given because in every step the time is constant. Now, we want that the derivative of $\vec{\mathbf{P}} = 0$ but the derivative of the linear momentum, due to Newton's second law is the sum of all the forces. So, in order to conserve linear momentum we want

$$\sum_i F_i = 0$$

Here we remember the physical meaning of the Lagrange multipliers (see Appendix A). We said that the $\lambda\frac{\partial C}{\partial t}$ was the force of reaction due to the restriction C. So, if we align

$$\Delta\mathbf{p} = \lambda \nabla_{\mathbf{p}} C(\mathbf{p}) \tag{3.4}$$

with λ a Lagrange multiplier, we will have all the reaction forces which all added (due is a closed system) is 0. So, this way we solve the problem of the linear momentum.

We have the same issue with angular momentum, given by

$$\vec{\mathbf{L}} = \sum_i \mathbf{r}_i \times m_i \Delta\mathbf{p}_i$$

where the $\mathbf{r}_i$ are the distances of the $\mathbf{p}_i$ to an arbitrary common rotation center. Again we want to settle at 0 the derivative so, using the cross product properties we have

$$\frac{d\vec{\mathbf{L}}}{dt} = \frac{d}{dt}\left(\mathbf{r} \times \vec{\mathbf{P}}\right) = \frac{d}{dt}\left(\mathbf{r} \times \vec{\mathbf{P}}\right) + \mathbf{r} \times \frac{d\vec{\mathbf{P}}}{dt} = \mathbf{v} \times (m \cdot \mathbf{v}) + \mathbf{r} \times \frac{d\vec{\mathbf{P}}}{dt} = \mathbf{r} \times \frac{d\vec{\mathbf{P}}}{dt}$$

And now, by the same arrangement we did in (3.4) we see that the angular momentum is also conserved.

If a projection violates one of these conditions, it introduces so called *ghost forces* which act like external forces dragging and rotating the object. However, only internal constraints need to conserve the momenta. Collision or attachment constraints are allowed to have global effect on the object.

3.3 Energy optimization viewpoint

Here we present another perspective on Position Based Dynamics by viewing it as the solution to a constrained optimization problem, seen by Macklin et al. in [21]. Considering only equality constraints, the problem to be solved can be stated as

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\Delta\mathbf{x}^T\mathbf{M}\Delta\mathbf{x} && (3.5)\\ &\text{subject to} && C_i(\mathbf{x}+\Delta\mathbf{x}) = 0, \qquad i = 1,\dots,m && (3.6) \end{aligned}$$

The solution variable $\Delta\mathbf{x}$ is the change in position such that the constraints are satisfied. Thus, the problem is formulated as finding the minimum change in the particle trajectory that minimize the kinetic energy, without violating the constraints (consistent with Gauss' principle of least constraint). If the constraint functions were linear, then Equation 3.5 would be a constrained quadratic minimization problem with a closed form solution. In practice, the constraints are arbitrary non-linear, non-convex functions, for which fast and globally optimal solvers do not exist. As we have said previously, PBD proceeds linearising the constraints and solving a sequence of local constrained quadratic minimizations:

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\Delta\mathbf{x}^T\mathbf{M}\Delta\mathbf{x}\\ &\text{subject to} && \mathbf{J}\Delta\mathbf{x} = \mathbf{b} \end{aligned}$$

where $\mathbf{J}_{(i,j)} = \frac{\partial C_i}{\partial \mathbf{x}_j}$ and $\mathbf{b} = [-C_1,\dots,-C_m]^T$. Now, this problem can be transformed, by theory of Lagrange multipliers, into

$$\begin{aligned} \mathbf{M}\Delta\mathbf{x} &= \mathbf{J}^T\lambda\\ \mathbf{J}\Delta\mathbf{x} &= \mathbf{b} \end{aligned}$$

where we obtain the already known equations of

$$\begin{aligned} \Delta\mathbf{x} &= \mathbf{M}^{-1}\mathbf{J}^T\lambda\\ \left[\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right]\lambda &= \mathbf{b} \end{aligned}$$

The last equation is a matrix equation for the linearised problem. In this way, the solver is equivalent to a problem of energy minimization.

3.4 Chebyshev semi-iterative method applied to PBD

In this section, in order to know what has been done in the convergence topic, we are going to explain the methodology and the conclusions exposed in [29].

PBD has been widely used in many high-end physics engines, such as PhysX, Havok Cloth, and Maya nCloth, thanks to its simplicity. When a vertex is involved in multiple constraints, its position can be updated either sequentially, known as the *Gauss-Seidel way*, or simultaneously through averaging, known as the *Jacobi way*, due to the analogy with the homonymous methods for solving linear systems. To implement PBD on GPU, the Jacobi way is often preferred so that the constraints can be processed in parallel. As we said in other sections previously, PBD uses the number of iterations to control how strictly the constraints are enforced and how stiff an object behaves, so it is free of numerical instability. However, it is difficult to formulate the relationship between the mechanical properties of an object and the number of iterations. In fact, PBD convergence rate drops as the mesh resolution increases.

A new constraint based simulation technique, known as **projective dynamics** ([19, 5]), emerged recently. Different from PBD, projective dynamics tries to bridge the gap between continuum mechanics and PBD. The key idea is to introduce energy potentials with a specific structure. While projective dynamics can be considered as a generalized version of PBD, its converged result is physically plausible and controllable by stiffness variables. Previous research showed that doing this can achieve visually acceptable results even within a few iterations. The catch is that a direct solver cannot be easily accelerated by GPU. So the whole method becomes less efficient, when more iterations are needed to reduce errors.

Since projective dynamics and PBD are similar to linear systems in many ways, a natural question is: *can we borrow ideas from the existing linear system solvers and get these techniques accelerated?* In order to try to answer this question, the method chosen is the Chebyshev semi-iterative method.

3.4.1 Chebyshev semi-interative method

We remember from numerical methods for linear algebra that a linear system can be formulated as: $\mathbf{A}\mathbf{x} = \mathbf{b}$, in which $\mathbf{A} \in \mathbb{R}^{N\times N}$ is a matrix, $\mathbf{b} \in \mathbb{R}^{N}$ is a given vector, and $\mathbf{x} \in \mathbb{R}^{N}$ is the unknown vector that needs to be found. When $\mathbf{A}$ is large and sparse, iterative methods are often favored over direct methods, to avoid matrices from being filled by new nonzeros during the solving process. Based on the splitting idea: $\mathbf{A} = \mathbf{B} - \mathbf{C}$, standard iterative methods, such as Jacobi and Gauss-Seidel, have the form:

$$\mathbf{x}^{(k+1)} = \mathbf{B}^{-1}\left(\mathbf{C}\mathbf{x}^{(k)} + \mathbf{b}\right) \tag{3.7}$$

If we split $\mathbf{B}^{-1}\mathbf{b}$ into $\mathbf{B}^{-1}\mathbf{A}\mathbf{x} = \mathbf{x} - \mathbf{B}^{-1}\mathbf{C}\mathbf{x}$, we can write

$$\mathbf{e}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{x} = \mathbf{B}^{-1}\mathbf{C}\left(\mathbf{x}^{(k)} - \mathbf{x}\right) = \mathbf{B}^{-1}\mathbf{C}\mathbf{e}^{(k)} \tag{3.8}$$

in which $\mathbf{e}^{(k)}$ is the error vector at the k-th iteration adn $\mathbf{x}$ the exact solution. Let the eigenvalue decomposition of $\mathbf{B}^{-1}\mathbf{C}$ be $\mathbf{V}\boldsymbol{\lambda}\mathbf{V}$, where $\boldsymbol{\lambda}$ is the eigenvalue matrix. We can reformulate the error vector at the k-th iteration as:

$$\mathbf{e}^{(k)} = \left(\mathbf{B}^{-1}\mathbf{C}\right)^{k}\mathbf{e}^{(0)} = \mathbf{V}\boldsymbol{\lambda}^{k}\mathbf{V}^{-1}\mathbf{e}^{(0)}$$

This means that these iterative methods converge linearly and their convergence rates depend on the largest eigenvalue magnitude, known as the **spectral radius**: $\rho\left(\mathbf{B}^{-1}\mathbf{C}\right)$. To ensure the convergence, we must have $\rho\left(\mathbf{B}^{-1}\mathbf{C}\right) < 1$.

Given the results produced by the iterative formula in Equation 3.7 $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(k)}$, we would like to obtain a better result from their linear combinations, which has the following form:

$$\mathbf{y}^{(k)} = \sum_{j=0}^{k} v_{jk}\mathbf{x}^{(j)}$$

in which v_{jk} are the blending coefficients to be determined. If the results are good already $\mathbf{x}^{(0)} = \ldots = \mathbf{x}^{(k)} = \mathbf{x}$, we must have $\mathbf{y}^{(k)} = \mathbf{x}$. So we require

$$\sum_{j=0}^{k} v_{jk} = 1 \tag{3.9}$$

The question is how to reduce the error of $\mathbf{y}^{(k)}$. Using Equation 3.9 and Equation 3.8 we can formulate the error $\mathbf{y}^{(k)} - \mathbf{x}$ into:

$$\sum_{j=0}^{k} v_{jk}\left(\mathbf{x}^{(j)} - \mathbf{x}\right) = \sum_{j=0}^{k} v_{jk}\left(\mathbf{B}^{-1}\mathbf{C}\right)^{j}\mathbf{e}^{(0)} = p_k\left(\mathbf{B}^{-1}\mathbf{C}\right)\mathbf{e}^{(0)}$$

in which

$$p_k(x) = \sum_{j=0}^{k} v_{jk} x^j$$

is a polynomial function. So to reduce the error we must reduce

$$\|p_k\left(\mathbf{B}^{-1}\mathbf{C}\right)\|_2 = \max_{\lambda_i} |p_k(\lambda_i)|$$

in which λ_i can be any eigenvalue of $\mathbf{B}^{-1}\mathbf{C}$.

Proposition 4. *If* $\mathbf{A}$ *is a symmetric matrix with positive diagonal entries and* $\mathbf{B}^{-1}\mathbf{C}$ *is created by Jacobi, the eigenvalues are real.*

Proof. Let λ and $\mathbf{v}$ be an eigenvaule and its eigenvector of $\mathbf{B}^{-1}\mathbf{C}$. With this we have

$$\mathbf{B}^{-\frac{1}{2}}\mathbf{C}\mathbf{B}^{-\frac{1}{2}}\left(\mathbf{B}^{\frac{1}{2}}\mathbf{v}\right) = \mathbf{B}^{\frac{1}{2}}\lambda\mathbf{B}^{-\frac{1}{2}}\left(\mathbf{B}^{\frac{1}{2}}\mathbf{v}\right) = \lambda\left(\mathbf{B}^{\frac{1}{2}}\mathbf{v}\right)$$

So λ is also the eigenvalue of $\mathbf{B}^{-\frac{1}{2}}\mathbf{C}\mathbf{B}^{-\frac{1}{2}}$. Since $\mathbf{B}$ is the diagonal matrix of A and its diagonal elements are all positive (hypothesis), $\mathbf{B}^{-\frac{1}{2}}$ must be real. So $\mathbf{B}^{-\frac{1}{2}}\mathbf{C}\mathbf{B}^{-\frac{1}{2}}$ is real symmetric and λ is real. □

With this, we can use the Chebyshev method to accelerate Jacobi iterations. If we know all of the eigenvalues and if k is sufficiently large, we can construct the polynomial function in a way that $p_k(\lambda_i) = 0$ for any λ_i. Unfortunately, it is difficult to know the eigenvalues, when the linear system is large and varying. Instead, if we know the spectral radius ρ such that $-1 \leq -\rho \leq \lambda_n \leq \ldots \leq \lambda_1 \leq \rho < 1$ we can ask $p_k(x)$ to be minimized for all $x \in [-\rho, \rho]$:

$$p_k(x) = \arg\min\left\{\max_{-\rho\leq x\leq\rho} |p_k(x)|\right\} \tag{3.10}$$

The unique solution to Equation 3.10 can be proved to be given by

$$p_k(x) = \frac{C_k(x/\rho)}{C_k(1/\rho)}$$

in which $C_k(x)$ is the Chebyshev polynomial with the recurrence relation

$$\begin{aligned} C_{k+1}(x) &= 2xC_k(x) - C_{k-1}(x) \\ C_1(x) &= x \\ C_0(x) &= 1 \end{aligned}$$

To reduce the computational and memory cost, we can reorganise and using all the identities before we get

$$\mathbf{y}^{(k+1)} = \omega_{k+1}\left(\mathbf{B}^{-1}\left(\mathbf{C}\mathbf{y}^{(k)} + \mathbf{b}\right) - \mathbf{y}^{(k-1)}\right) + \mathbf{y}^{(k-1)}$$

where ω_{k+1} follows the recurrence

$$\begin{aligned} \omega_{k+1} &= \frac{4}{4-\rho^2\omega_k} \\ \omega_1 &= \frac{2}{2-\rho^2} \\ \omega_0 &= 1 \end{aligned}$$

This method has been extensively analysed and it has been pointed out that even it looks similar to weighted Jacobi and successive overrelaxation (SOR), it converges much faster. This is because the Chebyshev method changes the factor ω in each iteration and it uses $\mathbf{y}^{(k-1)}$ instead of $\mathbf{y}^{(k)}$.

The biggest advantage of the Chebyshev semi-iterative method is its simplicity. Compared with Krylov subspace iterative methods, such as generalized minimum residual and conjugate gradient, the Chebyshev method has a short recurrence form and it does not use inner products, so it is ideal for parallel computing. Unfortunately, it is known that the Chebyshev method does not converge as fast as Krylov subspace methods do. Also, we have to know (or stimate numerically) the spectral radius ρ.

Now we are ready to perform this method on PBD. Let i a vertex and $\Delta\mathbf{p}_{i,C}$ be its position movement suggested by constraint C. Wang calculates its actual position update in each iteration by a relaxation coefficient α in [29]:

$$\mathbf{p}_i^{(k+1)} = \mathbf{p}_i^{(k)} + \alpha \sum_C \Delta\mathbf{p}_{i,C} \tag{3.11}$$

It has been preferred using Equation 3.11 since its momentum preserving and slightly more robust against divergence. Given the same α, if no divergence occurs, using the Chebyshev approach always results in faster convergence than not using the Chebyshev approach. The problem is that the Chebyshev approach makes the algorithm more vulnerable to divergence, so smaller α is needed for better stability. When simulating triangular meshes, Wang found the difference in α is not significant: $\alpha = 0.3$ for not using the Chebyshev approach, and $\alpha = 0.25$ for using the Chebyshev approach. But when simulating tetrahedral meshes, α must be much smaller to avoid divergence: $\alpha = 0.045$ for not using the Chebyshev approach and $\alpha = 0.0025$ for using the Chebyshev approach. Overall, the simulation is still accelerated, but the effect is less evident. For all this see figure Figure 3.1

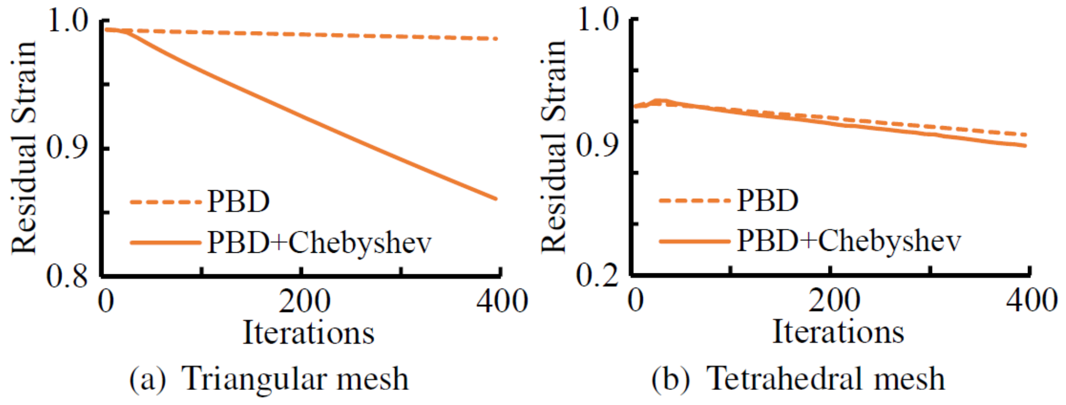

Figure 3.1: The convergence of Position Based Dynamics, using Jacobi iterations. This example shows that the acceleration effect of the Chebyshev approach is more significant on triangular meshes as shown in (a) than on tetrahedral meshes as shown in (b).

3.5 Motivation of the following work

As we have seen, PBD gather some of the advantages of FEM and MSM up: removes the typical instability problems, the objects themselves can directly be manipulated during simulation, the formulation of the algorithm allows the handling of general constraints and the explicit position based solver is easy to understand and implement. In other words, PBD presents robustness, simplicity, generality and, in most of the cases, a good performance. But as the reader has been able to realize, there are some open problems in this area. The main problem we face here is the same one that MSM had: since they are not directly physics based, its parameters have no direct relationship with the elastic theory ones: Young's modulus and Poisson's ratio (explained in chapter 2). Moreover, since its original approach has been from the engineering and video game field, in the best of our knowledge there is not a rigorous analysis of the PBD seen as a numerical method for finding the solution of a system. In order to try to respond these unanswered questions the next chapter is presented. In the first section we characterize PBD for elastic materials tuning the parameters such as the stiffness constraints. This part is an original contribution an has been accepted in CEIG'16 [25]. In the second part we propose some possible first steps in order to study the convergence properties, although this part is not as robust as the previous ones.

Chapter 4

Characterization and convergence of PBD

Computer animation and simulation of deformable materials in interactive applications is commonly addressed using Mass-Spring meshes or shape matching techniques (see chapter 2). More recently, the Position Based Dynamics (PBD) methodology is gaining popularity thanks to its unconditional stability and currently it is a widely used technique in video games (chapter 3). The main inconvenient is that the behaviour of the simulated body and the solution of PBD depends on non-physical parameters as the stiffness parameter of the constraint and the number of iterations. This problem has been addressed for other simulation methods by tuning this non-physical parameter sets in order to approximate the model as far as it is possible to another model with known physical parameter sets (for example, with the Finite Element Method).

Several works have faced the problem of parameter fitting for MSM ([26], e.g). However, the mechanical properties of PBD models have not been studied systematically. A characterisation of the dynamics of the PBD elasticity model would determine to what extent they are capable of reproduce actual elastic materials.

By adjusting k (the stiffness parameter of the constraint), and the number of projection iterations, this approach can be used to simulate elastic materials. Further details on the methodology and different applications can be found in [23, 3].

Recently Bender et al. [2] have introduced a set of energy constraints for the simulation of continuous materials, including elastic deformations, where physically meaningful parameters are used. However, to the best of our knowledge no previous work has been conducted in order to analyse the mechanical properties of PBD elastic materials based on geometric constraints. This problem is analogous to parameter fitting in modelling methodologies such as MSM, as we explained in subsection 2.2.1.

In the following pages, we shall follow the approach by Van Gelder [12] and by Lloyd et al. [20] to fit a linearisation of a PBD stiffness matrix. Our error function is based on that of [20], measuring the difference between the PBD stiffness matrix and a linear FEM stiffness matrix in Frobenius norm. The main difference is that we compute the stiffness matrix numerically by finite differences. We provide a closed formulation in order to, given the two physical parameters, obtain the corresponding stiffness parameters in the case of a hexahedral element with distance and volume preservation constraints. Furthermore, we see numerically "how bad" PBD works compared with FEM and discuss the convergence topic giving the first steps for a rigorous analysis of the Position Based Dynamics.

4.1 Elasticty in Position Based Dynamics

Now, let's consider a portion of material, simulated with PBD. We build it by means of n particles we define one or more constraints that describe the equilibrium state of the cube. Notice that we do not explicit the constraint, so the following method is valid for **any kind of geometrical constraints**. If we linearise the dynamics of the system, we can find an equivalent to the stiffness matrix. We are seeking for a matrix $\mathbf{K}^{\mathrm{PBD}}$ for which, when applying a force $\mathbf{f} = (\mathbf{f}_1, \ldots, \mathbf{f}_n)^T$ to the PBD particles, the resulting deformations $\mathbf{u}$ accomplish

$$\mathbf{f} = \mathbf{K}^{\mathrm{PBD}}\mathbf{u} \tag{4.1}$$

with small $\mathbf{u}$.

In practice, using the dependence of $\mathbf{u}$ as a function of $\mathbf{f}$ is not the most convenient approach, since it leads to the inverse of $\mathbf{K}^{\mathrm{PBD}}$. Instead, we shall consider $\mathbf{f}$ as a function of $\mathbf{u}$. However, this poses a difficulty, since the output of the PBD projection process is not a force or an acceleration, but a position change.

In order to overcome this we shall calculate the force that is equivalent to the PBD projection, $\Delta\mathbf{p}$. We mean a force that, when integrated numerically causes the same change in position than a given $\Delta\mathbf{p}$. Here we see that we depend on the integration scheme we choose for solving the DE derived from the Newton's second law of motion. We have followed the bibliography for settle this (for example, see [17]). When using semi-implicit Euler scheme with a time step Δt, we deduce this force using Newton's second law:

$$\mathbf{f}_i = \frac{m_i}{\Delta t^2}\Delta\mathbf{p}_i, \tag{4.2}$$

where Δt is the time step and m_i is the mass of the particle.

This strategy presents an additional advantage. The dynamics of the PBD method is known to be dependent on the size of the time step. By considering this equivalent force we shall include this dependence in our computation of the PBD parameters, as we shall see.

Thus, we consider $\Delta\mathbf{p}$ as a function that depends on the deformation $\mathbf{u}$ as defined previously. If we linearise this function near $\mathbf{u} = 0$, we have that, for small values of $\mathbf{u}$

$$\Delta\mathbf{p}(\mathbf{u}) \simeq \frac{\partial \Delta\mathbf{p}(0)}{\partial \mathbf{u}}\mathbf{u}, \tag{4.3}$$

and, using Equation 4.2, we can define the stiffness matrix for PBD as

$$\mathbf{K}^{\text{PBD}} = \frac{\mathbf{M}}{\Delta t^2}\frac{\partial \Delta\mathbf{p}(0)}{\partial \mathbf{u}} \tag{4.4}$$

where

$$\mathbf{M} = \begin{pmatrix} m_1 & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & m_n \end{pmatrix}$$

is the mass matrix of the PBD system (a diagonal matrix). This matrix is fully known previously even to the simulation.

In a procedure analogous to FEM matrix assembly, the PBD stiffness matrix can be defined per constraint, and the stiffness matrix of the whole system is assembled by adding all the constraint stiffness matrices. This computation can be done analytically for some constraints, such as the distance constraint proposed in [23], while for others it can be hard to find a closed expression for $\mathbf{K}^{\text{PBD}}$. In this cases we depend on the numerical differentiation strategy followed, such as finite differences.

As we previously mentioned, in PBD there are two main suitable parameters associated to the method; the number of projection iterations, which is a global parameter, and a stiffness-like parameter $0 \leq k \leq 1$, which can be different for every constraint. In this work we shall consider **a single iteration** for our analysis, and consider the stiffness matrix as a function of the different $k_j,\ j = 1, \ldots, m$, for a system with m constraints.

Following the approach by Lloyd et al. [20], we will use an optimisation algorithm to find the values of the k_j which minimise the difference between $\mathbf{K}^{\text{PBD}}$ and $\mathbf{K}^{\text{FEM}}$, where the PBD matrix is considered as a function of the stiffness $\mathbf{K}^{\text{PBD}} = \mathbf{K}^{\text{PBD}}(k_1, \ldots, k_m)$.

This optimisation has an additional difficulty; the FEM stiffness can have an arbitrarily large norm, since E (Young's modulus) is not upper bounded, and the dividing term $1 - 2\nu$ (where ν is the Poisson's ratio) tends to infinity as ν tends to $\frac{1}{2}$. On the contrary, it has been seen in a number of papers (see for example the survey done by [3]) that for a large quantity of constraints PBD converges. This fact makes us think intuitively that the jacobian term in Equation 4.4, $\partial\Delta\mathbf{p}/\partial\mathbf{u}$, has to be bounded somehow. For now and on, although a bound for this term, or a mathematical proof that PBD has contractive iterations, has not been done in the literature we assume this for our analysis. We remark again that this analysis will be for one iteration, so at least initially we are not able to determine how the incorporation of the number of iterations to the following algorithm will change the analysis. Although some parts are not dependent, if more iterations are simulated we recommend to reanalyse the reasoning.

Taking this into account, from Equation (4.4) we see that the only term that is not bound in our definition of $\mathbf{K}^{\text{PBD}}$ is the fraction $1/\Delta t^2$. This implies that, when a particular elastic material is to be reproduced using PBD, the integration time step should not be freely chosen. Taking this into account, we propose to set the value of Δt as

$$\Delta t = \sqrt{\frac{\|\mathbf{M}\|}{\|\mathbf{K}^{\text{FEM}}\|}}, \tag{4.5}$$

since this value, when applied in (4.4), will enable arbitrarily large norms for $\mathbf{K}^{\text{PBD}}$. From this value or Δt, we use the following optimisation problem to fit the PBD parameters:

$$(k_1, \ldots, k_m) = \arg\min_{k_i} \left\{ \left\| \mathbf{K}^{\text{FEM}} - \frac{\mathbf{M}}{\Delta t} \frac{\partial\Delta\mathbf{p}}{\partial\mathbf{u}}(k_1, \ldots, k_m) \right\| \right\} \tag{4.6}$$

where $0 \leq k_j \leq 1$. Here, the norm used is the Frobenius norm, as proposed by [20]. For the optimisation we have used MatLab's instruction `fminsearch`, a command that uses the Nelder-Mead simplex algorithm as described by Lagarias et al. . (see [16]).

The previous methodology to fit the parameters of the PBD system can be summarised as follows:

1. Choose a Young's modulus E and a Poisson's ratio ν.
2. Build a reference PBD system and an equivalent FEM element.
3. Build the FEM matrix, and set Δt.

4. Fit the PBD parameters solving the optimisation problem (4.6).

The first step is absolutely dependent on the material that we want to simulate. So, there is no possible discussion here. In the following section we will do some tests for different values of Young's modulus and Poisson's ratios. The second step is maybe beyond a mathematical reasoning. Obviously, the PBD system and the FEM element have to correspond with the geometry chosen to be simulated, but there are several ways (for example, refining the mesh of points). Our decision, and maybe one of the most intuitive, is to choice the vertexes of the PBD system where the nodes of the elements of the FEM system are. The third step is one of the most expensive computationally speaking, since the construction and assembling of the FEM matrix involve the calculus of integrals and after that we have to obtain the norm of this matrix for the setting of the timestep Δt. The last step, considered the main idea of this algorithm, can be done either analytical or numerically, depending on the complexity of the resulting PBD matrix. For the optimisation we have used MatLab's instruction `fminsearch` and the Python's instruction `minimize`, commands that use the Nelder-Mead simplex algorithm as described by Lagarias et al. [16].

4.2 Numerical experiments

In order to validate our strategy we follow the methodology proposed by San Vicente et al. [27] and use an hexahedral FEM element to fit a cubic PBD system formed by eight masses. In the PBD system, we use distance constraints where [27] use springs and, as in their work, the constraints can be classified in three types: edge constraints, which link masses connected by an edge of the cube; face constraints, which link particles through the diagonal of a face; and internal diagonal constraints, which link vertexes which do not share any edge of the cube. All the constraints of the same type will share a unique stiffness coefficient. This will produce what is known as "isotropic" material, this is, it will have identical values of a property in all directions. This definition is also used in geology and mineralogy. Glass and metals are examples of isotropic materials.

The distance constraint proposed by [23] is given by

$$C(\mathbf{p}_i, \mathbf{p}_j) = \|\mathbf{p}_i - \mathbf{p}_j\| - d$$

where d is the equilibrium distance.

For this constraint, the projection (just substitute in the general predicted position of a particle deduce from PBD) is given by

$$\Delta\mathbf{p}_i = -\frac{w_i}{w_j}\Delta\mathbf{p}_j = \frac{w_i}{w_i + w_j}(\|\mathbf{p}_{ij}\| - d_{ij})\frac{\mathbf{p}_{ij}}{\|\mathbf{p}_{ij}\|} \tag{4.7}$$

where $\mathbf{p}_{ij} = \mathbf{p}_i - \mathbf{p}_j$ and w_i is the inverse of the mass m_i of the particle i.

With a single iteration step, the stiffness matrix for this constraint has the same analytical form as the equivalent stiffness matrix for a Mass-Spring Model developed by Lloyd et al. [20]. But in their work it is shown that the resulting matrix cannot be equated to $\mathbf{K}^{\text{FEM}}$ except for the case of an equilateral triangle of springs. Moreover, although the internal diagonal constraints contribute to volume preservation, they are also coupled with other deformation modes. For these reasons we add to the distance constraints the volume preservation constraint proposed by [23]. This constraint is a discretization as a consequence of Green's and Gauss' theorems (this reasoning can be seen unrigorously in [24]). It is built as an equality involving the location of all eight vertexes. It can be seen as trying to approximate the volume of the triangulation formed by all the nodes to the rest volume:

$$C(\mathbf{p}_1, \ldots, \mathbf{p}_8) = \left(\sum_{i=1}^{N_{\text{triangles}}} \left(\mathbf{p}_{t_1^i} \times \mathbf{p}_{t_2^i} \right) \cdot \mathbf{p}_{t_3^i} \right) - V_0$$

where V_0 is the volume of the undeformed cube. Here t_1^i, t_2^i and t_3^i are the three indexes of the vertexes belonging to triangle i. The sum computes the actual volume of the closed mesh. This constraint function yields the gradients

$$\nabla_{\mathbf{p}_i} C = \sum_{j:t_1^j=i} (\mathbf{p}_{t_2^j} \times \mathbf{p}_{t_3^j}) + \sum_{j:t_2^j=i} (\mathbf{p}_{t_3^j} \times \mathbf{p}_{t_1^j}) + \sum_{j:t_3^j=i} (\mathbf{p}_{t_1^j} \times \mathbf{p}_{t_2^j})$$

The inclusion of this constraint has the drawback that it does not have a tractable expression for its jacobian matrix (or gradient) with respect to displacement, necessary to build the stiffness matrix defined in Equation (4.4). Instead, we shall compute the matrix of the elastic PBD cube numerically using finite differences.

Since all the distance constraints of the same type share the same stiffness coefficient, we have a PBD system with four parameters: the stiffness of edge constraints, k_e; the stiffness of face constraints, k_f; the stiffness of internal diagonal constraints, k_d; and the stiffness of the volume preservation constraint, k_v.

To show the proposed methodology and evaluate the results we have computed the values of the PBD parameters for the range of non-negative values of the Poisson's ratio, $\nu \in \left[0, \frac{1}{2}\right[$. This case corresponds to the so called "non-auxetic" materials. If $\nu < 0$ the material, called "auxetic", expands in one direction as you pull in the transversal direction. For more information about these materials see, for example, [11]. Since the linear FEM stiffness matrix depends linearly on E, as it can be deduced from equations (2.6) and (2.7), the results are independent on its value. For this reason in this section all the results are shown only for $E = 1$.

For each value of ν, the relative error

$$e_r = \frac{\|\mathbf{K}^{\text{PBD}} - \mathbf{K}^{\text{FEM}}\|}{\|\mathbf{K}^{\text{FEM}}\|} \tag{4.8}$$

has been computed (see section B.1). As we mentioned before, we are following a strategy done for the MSM instead of for PBD. If we would be dealing only with distance constraints an element simulated with PBD will behave **exactly** as one modelled by MSM, since the formulation of the Hooke's Law and the distance constraint is the same. But, the main part that conserves volume in a MSM is the set of internal springs, the diagonal springs. When we deform a cube, for example, these springs are modified too, and it cannot behave properly in terms of volume conservation. One of the differences between our PBD elastic cube and the MSM that have been analysed in the context of parameter fitting is the introduction of the volume constraint. Although we have diagonal constraints too, the volume preservation one has not been studied since there is not known a direct transliteration to a MSM. To determine the relevance of this constraint in the results we have repeated the parameter adjustment forcing $k_v = 0$. In Figure 4.1 the values of the fitted stiffness coefficients are presented.

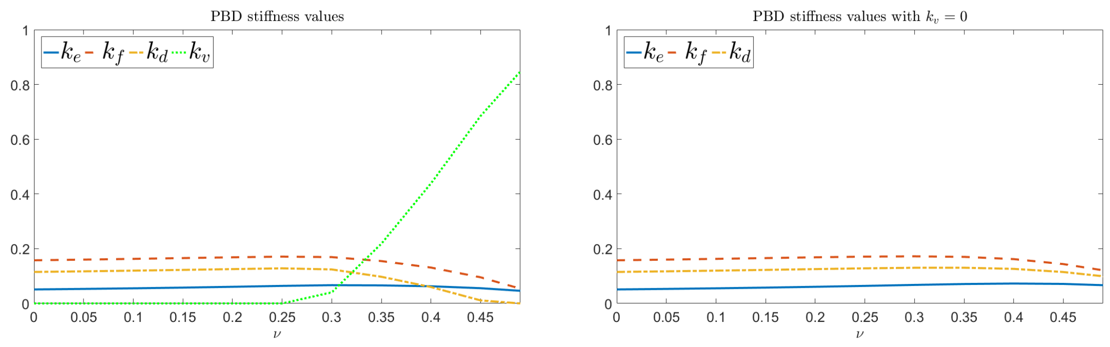

Figure 4.1: Parameter values obtained for the PBD cubic element for different values of Poisson's ratio ν. The values are shown for the problem with all four constraint types (left) and forcing the volume preservation constraint to 0 (right).

The figure on the left shows the values for the fitting considering all four stiffness constants, while the figure on the right shows the results when the volume constraint is not considered, by forcing $k_v = 0$ (see section B.2). From the results it is clear that from $\nu = 0.29$ the stiffness k_v gains relevance, approaching $k_v = 1$ as ν tends to $\frac{1}{2}$. On the contrary, the value of k_d reduces to approach zero in the same range. In the fitting without volume constraint, we also observe this reduction in the value of the constants, but it does not approach $k_d = 0$. We can see this in more detail in Figure 4.2. We have plotted in four different windows the distinct stiffness parameters taking into a account the volume preservation constraint and disregarding it.

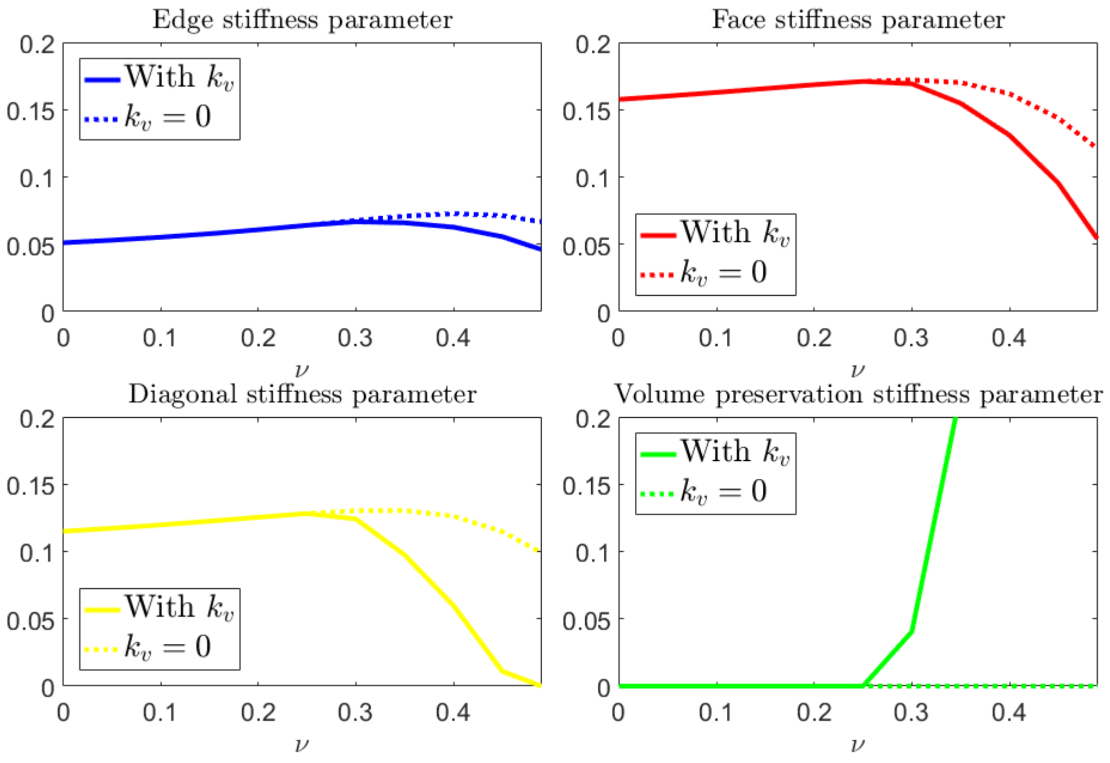

Figure 4.2: An enlargement of differences between the stiffness parameters taking into account the volume conservation constraint and forcing it to 0. It is clear that, as the volume preservation constraint gains importance (right down) all the other ones take lower values.

Looking at the form of the plots in Figure 4.1 we have decided to make a polynomial fitting (see section B.3). We adjust the data by two polynomials of second order (third order for k_v) in each case, when $\nu \leq 0.29$ and when $\nu \geq 0.29$.

$$k_e(\nu) = \begin{cases} 0.0698\nu^2 + 0.0343\nu + 0.0511 & 0 \leq \nu \leq 0.29 \\ -0.7204\nu^2 + 0.4656\nu - 0.0084 & 0.29 \leq \nu < 0.5 \end{cases}$$

$$k_f(\nu) = \begin{cases} -0.0302\nu^2 + 0.0614\nu + 0.1572 & 0 \leq \nu \leq 0.29 \\ -2.3822\nu^2 + 1.2909\nu - 0.0044 & 0.29 \leq \nu < 0.5 \end{cases}$$

$$k_d(\nu) = \begin{cases} 0.0527\nu + 0.1147 & 0 \leq \nu \leq 0.29 \\ -0.5795\nu^2 - 0.2818\nu + 0.2636 & 0.29 \leq \nu < 0.5 \end{cases}$$

$$k_v(\nu) = \begin{cases} 0 & 0 \leq \nu \leq 0.29 \\ -50.6270\nu^3 + 63.1461\nu^2 - 21.4563\nu + 2.1615 & 0.29 \leq \nu < 0.5 \end{cases}$$

Figure 4.3 shows the graph of the polynomial regression compared with the fitted parameters. This part of our work has been accepted for presentation in CEIG'16

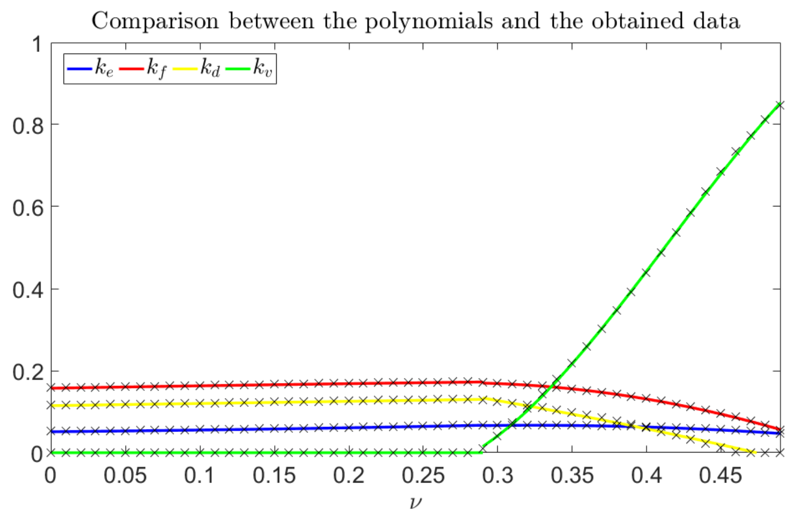

Figure 4.3: Comparison between the previously obtained polynomials and the data from the minimisation strategy. The polynomials are represented with a line, while the fitted data are represented with crosses.

(*Congreso Español de Informática Gráfica*) under the title "Characterisation of Position Based Dynamics for Elastic Materials". See Rodero et al. in [25].

In order to check the quality of the fitting we resort to the mean-squared error. We remember that this quantification measures the average of the squares of the errors or deviations, that is, the difference between the estimator and what is estimated. We have calculated them with the instruction `immse` of Matlab. The results are shown in Table 4.1

	k_e	k_f	k_d	k_v
$\nu \leq 0.29$	$2.146 \cdot 10^{-9}$	$8.9209 \cdot 10^{-8}$	$1.1271 \cdot 10^{-7}$	$3.3103 \cdot 10^{-6}$
$\nu \geq 0.29$	$1.1827 \cdot 10^{-7}$	$9.4752 \cdot 10^{-7}$	$2.6893 \cdot 10^{-5}$	$1.2526 \cdot 10^{-5}$

Table 4.1: Mean-squared errors for the polynomial fitting of the stiffness parameters of PBD.

We observe that the biggest error is in the case of the diagonal stiffness parameter when $\nu \geq 0.29$. Even though, it is a quite small error (smaller than the short precision of Matlab) so we can affirm that the choice of a polynomial fitting has been a good one.

Furthermore, we have calculated also the coefficient of determination of this fitting (R^2). It has been done with the instruction `corr` (also with MatLab). The outputs are shown in Table 4.2.

	k_e	k_f	k_d	k_v
$\nu \leq 0.29$	$9.9990 \cdot 10^{-1}$	$9.9572 \cdot 10^{-1}$	$9.9462 \cdot 10^{-1}$	-
$\nu \geq 0.29$	$9.9701 \cdot 10^{-1}$	$9.9923 \cdot 10^{-1}$	$9.8658 \cdot 10^{-1}$	$9.9982 \cdot 10^{-1}$

Table 4.2: Coefficients of determination (R^2) for the polynomial fitting of the stiffness parameters of PBD.

Consistently with Table 4.1 the "worst" R^2 is in the case of the diagonal stiffness parameter when $\nu \geq 0.29$. But, again, its a quite good value.

The observed behaviour of the stiffness coefficients indicates that the volume constraint plays a relevant role to get a better model fitting. This analysis is confirmed by the relative error value of the approximation, shown in Figure 4.4.

Relative error in both fitting problems is the same until the value $\nu \simeq 0.29$. Figure 4.4 shows that when the volume constraint is not used for the parameter set that excludes the volume constraint raises from ν about 0.29, and increases as the Poisson's ratio approaches its upper limit.

As a conclusion, we have presented a procedure to characterise the dynamics of a PBD elastic cube, by means of the linearisation of the projection process. Our approach is based on building a stiffness matrix and seeking for the parameters that

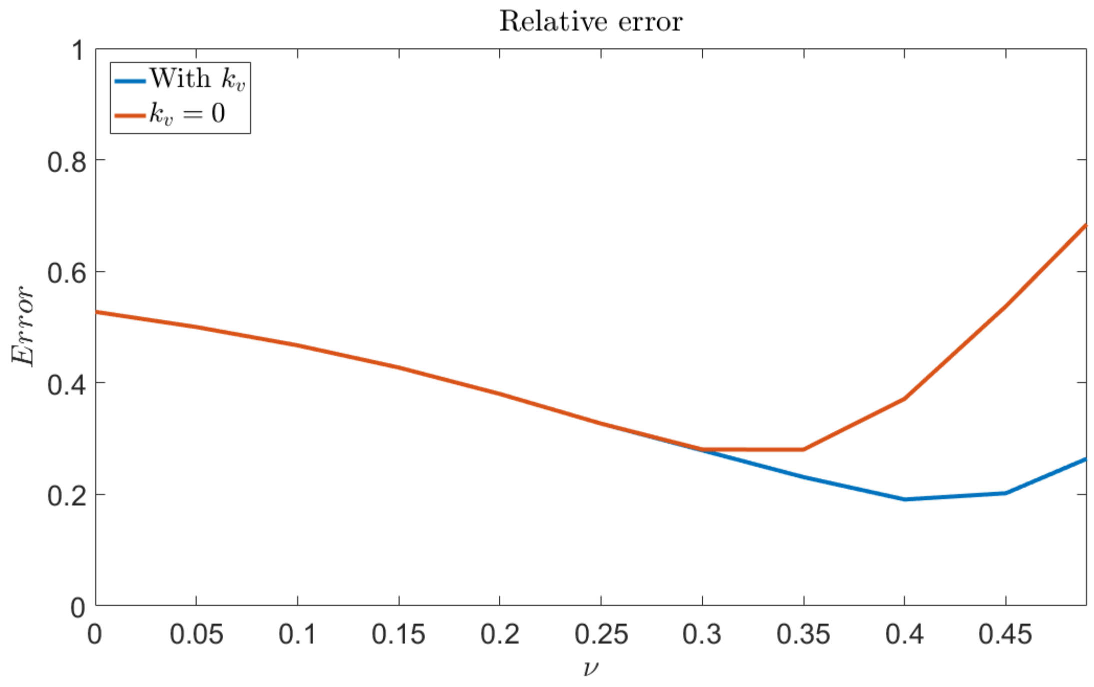

Figure 4.4: The relative error for the optimal parameter values for different values of ν. The errors for the complete set of parameters and for the model without volume constraint are shown.

make it as close as possible, in Frobenius norm, to the equivalent stiffness matrix for linear FEM.

The analysis of the results throws several conclusions. First, the fitting by Frobenius norm has limitations since the best results give relative error around 20% during the fitting. However, the proposed procedure also shows that a systematic parameter fitting can be performed. If a proper cost function is found, PBD based elastic models can be adjusted to reproduce certain scenarios. It is noteworthy that the best results are met for high values of Poisson's ratio, indicating the decision to use volume constraint has been successful.

We conjecture that a data driven approach, in which the parameters are fitted using reference samples, instead of stiffness matrix, must substantially improve the results of this work. We also expect that the introduction of the number of iterations in the fitting process, which can provide stiffer behaviours with the same coefficients, will also help improve the results of the methodology.

4.3 Convergence properties of Position Based Dynamics

As we have been able to see, PBD is not based on physical parameters. Many authors assume the *unconditional stability* of this method but in the best of our knowledge, it has not been proven in any place. Although this fact is quite astonishing in Mathematics, it is very common in Computer Graphics to present results without a convergence analysis. Although we have talked about convergence in section 3.4 the work done by Wang et al. [29] in what this section is based only encompass numerical experiments about convergence comparing several methods and their performance on CPU and GPU. Based on this, we will try to address the convergence issue from a more rigorous point of view.

In the previous sections we have dealt with **one iteration** of Position Based Dynamics, that is, we have not analysed what happens when we see PBD as a **root-finding algorithm** for a system of equations. In this topic, again, there is not available bibliography, so we will try to study this problem with the knowledge acquired in the degree.

First of all, we recall the linearisation done by the PBD algorithm. In this section we will use the same notation of all this work, and the bold characters will denote a vectorial element. Our objective is to find a point $\mathbf{p} = (\mathbf{p}_1, \dots, \mathbf{p}_n)$ for which, for all given constraints $C(\cdot)$

$$C_j(\mathbf{p}) = 0 \qquad j = 1, \dots, m$$

In order to achieve this, PBD made an iterative linearisation, searching for a correction $\Delta\mathbf{p}$ such that, using Taylor series for a concrete point $\mathbf{p}_i$

$$C_j(\mathbf{p} + \Delta\mathbf{p}) = C_j(\mathbf{p}) + \nabla^T_{\mathbf{p}_i} C_j(\mathbf{p}) \cdot \Delta\mathbf{p}_i + \mathcal{O}\left(\|\Delta\mathbf{p}_i\|^2\right) = 0 \qquad j = 1, \dots, m$$

Without taking into account the term $\mathcal{O}(\|\Delta\mathbf{p}_i\|^2)$, and being λ a Lagrange multiplier for conserving momenta we have to accomplish the equations

$$\begin{array}{rcl} \nabla^T_{\mathbf{p}_i} C_j(\mathbf{p}) \cdot \Delta\mathbf{p}_i & = & -C_j(\mathbf{p}) \\ \Delta\mathbf{p}_i & = & -\lambda \nabla_{\mathbf{p}_i} C_j(\mathbf{p}) \end{array}$$

All together gives the closed formula for the predicted position of PBD

$$\Delta\mathbf{p}_i = -\nabla_{\mathbf{p}_i} C_j(\mathbf{p}) \frac{C_j(\mathbf{p}}{\sum_k \|\nabla_{\mathbf{p}_k} C_j(\mathbf{p})\|^2} \qquad j = 1, \dots, m \tag{4.9}$$

where in all the previous recalling we have not took into account the masses of the particles.

If we pay attention to this formula, we can relate it to the **Newton-Raphson root-finding algorithm**. As can be seen in [13], the Newton-Raphson method consists on, given $f : \mathbb{R} \to \mathbb{R}$ follow the iterative process

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \tag{4.10}$$

where the super-index indicates in which iteration we are.

Although they are not the same, there is a clear similarity between Equation 4.9 and Equation 4.10, so we can think of PBD as a sort of Newton-Raphson algorithm for multiple variables with an arbitrary number of functions and variables.

In order to solve such a general set of equations (when a vertex is involved in multiple constraints) the predicted position can be updated either **sequentially**, known as "Gauss-Seidel-type" iteration or **simultaneously**, through averaging, known as "Jacobi-type" iteration. The part they borrow from the original Gauss-Seidel algorithm is the idea of solving each constraint independently one after the other. In contrast to simultaneous iteration, modification to point locations immediately get visible to the process. Although the sequential algorithm is very easy to implement, the main disadvantage is that the order in which the constraints are applied affects to the stiffness of the object and even to the final result (see for example [5]). Some authors like Müller in [22] run the sequential solver two times while processing the constraints in the opposite order in the second iteration to make the process symmetric. On the other hand, the simultaneous solver often converges more rapidly, mostly in GPU (see again [22]). Another pro of this way is that as it performs through averaging it can deal with contradictory constraints as opposed to the sequential solver. We represent this fact schematically in Figure 4.5, extracted from [5].

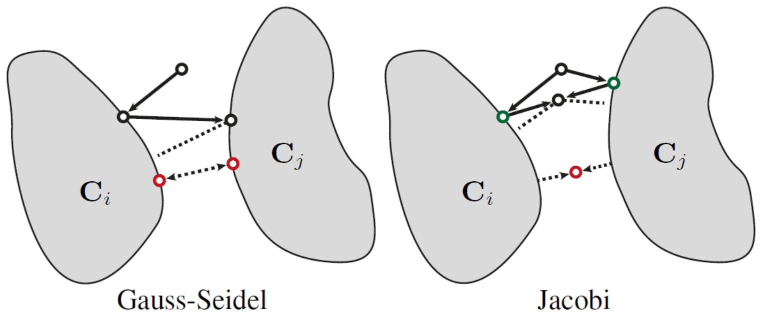

Figure 4.5: "Gauss-Seidel" vs. "Jacobi". If there is no feasible solution, the sequential algorithm will oscillate (between the two red points) while the simultaneous algorithm reaches a consensus. [5].

But, even all of the aforementioned "empyrical evidences" there is not, in our knowledge, a robust and purely theoretical proof of this assumed convergence. As a first step, although we have not arrived to a closed solution of this problem, we conjecture that a good approach might be to consider the root-finding problem presented in PBD in a matricial way. With this we mean follow the linearisation done some lines before but, this time, instead of constraint by constraint, with a vectorial function containing all the constraints. We will denote this new function with bold characters $\mathbf{C}(\mathbf{p}) = (C_1(\mathbf{p}), \ldots, C_m(\mathbf{p}))^T$. So, first of all, by Taylor's developments, we want to find a correction $\Delta\mathbf{p}$ such that

$$\mathbf{C}(\mathbf{p} + \Delta\mathbf{p}) = \mathbf{C}(\mathbf{p}) + \mathbf{J_p} \cdot \Delta\mathbf{p} + \mathcal{O}\left(\|\Delta\mathbf{p}\|^2\right) = \mathbf{0}$$

and, similarly to what has been done in the classic PBD

$$\Delta\mathbf{p} = -\mathbf{J}_\mathbf{p}^T \boldsymbol{\lambda} \tag{4.11}$$

where $\mathbf{J_p}$ is the jacobian matrix evaluated in the point $\mathbf{p}$. Notice that $\boldsymbol{\lambda}$ is not an scalar, but a vector of Lagrange multipliers. Substituting properly we have:

$$\mathbf{C}(\mathbf{p}) - \mathbf{J_p}\mathbf{J}_\mathbf{p}^T \boldsymbol{\lambda} = \mathbf{0}$$

Now here we notice the main *trick* of PBD: instead of using $\mathbf{J_p}\mathbf{J}_\mathbf{p}^T$ the system matrix, they take the matrix formed by its **diagonal**, $\mathbf{D} = diag(\mathbf{J_p}\mathbf{J}_\mathbf{p}^T)$. So, the correction would be

$$\Delta\mathbf{p} = -\mathbf{J}_\mathbf{p}^T \mathbf{D}^{-1} \mathbf{C}(\mathbf{p}) \tag{4.12}$$

Remark 1. If we take into account the masses of the particles we remember that the predicted position in classic PBD was

$$\Delta\mathbf{p}_i = -\nabla_{\mathbf{p}_i} w_i C_j(\mathbf{p}) \frac{C_j(\mathbf{p}}{\sum_k w_k \|\nabla_{\mathbf{p}_k} C_j(\mathbf{p})\|^2} \qquad j = 1, \ldots, m$$

where $w_i = 1/m_i$ were the inverse of the masses.

So, in the matricial form, we need to weight Equation 4.11 by the inverse of the diagonal matrix of masses $\mathbf{M} = diag(m_1, \ldots, m_n)$:

$$\Delta\mathbf{p} \;=\; -\mathbf{M}^{-1}\mathbf{J}_\mathbf{p}^T \lambda \implies C(\mathbf{p}) - \mathbf{J_p}\mathbf{M}^{-1}\mathbf{J}_\mathbf{p}^T \lambda = \mathbf{0}$$

In this case we split the matrix $\mathbf{M}$ into two matrices $\sqrt{\mathbf{M}} := diag(\sqrt{m_1}, \ldots, \sqrt{m_n})$ and create a new weighted jacobian matrix $\widetilde{\mathbf{J_p}} = \mathbf{J_p}\sqrt{\mathbf{M}^{-1}}$ and a new set of restrictions $\widetilde{C(\mathbf{p})} = C(\mathbf{p})\sqrt{\mathbf{M}^{-1}}$ so we will have:

$$\widetilde{\mathbf{J_p}}\widetilde{\mathbf{J_p}}^T \boldsymbol{\lambda} = \widetilde{C(\mathbf{p})}$$

Going back to the case with no masses (for simplicity) we notice that with this arrangement, we have no issues of invertibility, since an entry of the diagonal of this matrix is 0 if and only if a row of the jacobian matrix is a row of zeros, but this makes no sense because this would mean that a constraint does not depend on any particle.

Writing the matricial PBD with the classical form that the iterative processes have:

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \Delta\mathbf{p}^{(k)} = \mathbf{p}^{(k)} - \mathbf{J}^T_{\mathbf{p}^{(k-1)}}\boldsymbol{\lambda}^{(k-1)} = \mathbf{p}^{(k)} - \mathbf{J}^T_{\mathbf{p}^{(k-1)}}\mathbf{D}^{-1}_{\mathbf{p}^{(k-1)}}\mathbf{C}(\mathbf{p}^{(k-1)}) \quad (4.13)$$

where the super-index indicates the iteration where we are. We remark, as can be seen in Equation 4.13 two important facts: Position Based Dynamics is, in essence, a multi-step iterative algorithm, although in a first look we may have omitted this. And the second one, and what makes this process hard to compare to the widely studied iterative methods (such as Gauss-Seidel or Jacobi, for example) is that in each step the vectorial constraint function is evaluated in the new point and as a consequence the jacobian matrix and the $\mathbf{D}$ matrix has to be calculated again. Even though, we can affirm that this process will be convergent if, and only if

$$\Delta\mathbf{p}^{(k)} = -\mathbf{J}^T_{\mathbf{p}^{(k-1)}}\mathbf{D}^{-1}_{\mathbf{p}^{(k-1)}}\mathbf{C}(\mathbf{p}^{(k-1)}) \xrightarrow{k} \mathbf{0}$$

Here we observe that the construction of the points $\mathbf{p}^{(k)}$ has been done in the way that $C(\mathbf{p}^{(k+1)}) \to 0$, so it would be enough to see that the matrix $\mathbf{J}^T_{\mathbf{p}^{(k-1)}}\mathbf{D}^{-1}_{\mathbf{p}^{(k-1)}}$ is bounded as $k \to \infty$.

In order to simplify the notation we will denote

$$\partial_{\mathbf{p}_j} C_i := \frac{\partial C_i}{\partial \mathbf{p}_j}$$

Hence, matrix we want to consider is

$$\mathbf{J}^T_{\mathbf{p}^{(k)}}\mathbf{D}^{-1}_{\mathbf{p}^{(k)}} = \begin{bmatrix} \partial_{\mathbf{p}_1^{(k)}} C_1 & \partial_{\mathbf{p}_1^{(k)}} C_2 & \cdots & \partial_{\mathbf{p}_1^{(k)}} C_m \\ \partial_{\mathbf{p}_2^{(k)}} C_1 & \partial_{\mathbf{p}_2^{(k)}} C_2 & \cdots & \partial_{\mathbf{p}_2^{(k)}} C_m \\ \vdots & \vdots & \ddots & \vdots \\ \partial_{\mathbf{p}_n^{(k)}} C_1 & \partial_{\mathbf{p}_n^{(k)}} C_2 & \cdots & \partial_{\mathbf{p}_n^{(k)}} C_m \end{bmatrix} \begin{bmatrix} \frac{1}{\|\nabla_{\mathbf{p}^{(k)}} C_1\|^2} & 0 & \cdots & 0 \\ \vdots & \frac{1}{\|\nabla_{\mathbf{p}^{(k)}} C_2\|^2} & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & \frac{1}{\|\nabla_{\mathbf{p}^{(k)}} C_m\|^2} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial_{\mathbf{p}_1^{(k)}} C_1}{\|\nabla_{\mathbf{p}^{(k)}} C_1\|^2} & \frac{\partial_{\mathbf{p}_1^{(k)}} C_2}{\|\nabla_{\mathbf{p}^{(k)}} C_2\|^2} & \cdots & \frac{\partial_{\mathbf{p}_1^{(k)}} C_m}{\|\nabla_{\mathbf{p}^{(k)}} C_m\|^2} \\ \frac{\partial_{\mathbf{p}_2^{(k)}} C_1}{\|\nabla_{\mathbf{p}^{(k)}} C_1\|^2} & \frac{\partial_{\mathbf{p}_2^{(k)}} C_2}{\|\nabla_{\mathbf{p}^{(k)}} C_2\|^2} & \cdots & \frac{\partial_{\mathbf{p}_2^{(k)}} C_m}{\|\nabla_{\mathbf{p}^{(k)}} C_m\|^2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial_{\mathbf{p}_n^{(k)}} C_1}{\|\nabla_{\mathbf{p}^{(k)}} C_1\|^2} & \frac{\partial_{\mathbf{p}_n^{(k)}} C_2}{\|\nabla_{\mathbf{p}^{(k)}} C_2\|^2} & \cdots & \frac{\partial_{\mathbf{p}_n^{(k)}} C_m}{\|\nabla_{\mathbf{p}^{(k)}} C_m\|^2} \end{bmatrix}$$

(we have enlarged the last matrix for a better visualization). And here we hypotesise, based on all the satisfactory results obtained in practical cases, that if not for majority of the used constraints, at least for a huge family of constraints this last matrix is bounded (most used constraints can be seen for example in [3]).

We conclude here, since we are not able for the moment with the present knowledge obtain a better closed proof or solution to the convergence issue. Still, in this last section we have done probably the first steps in the direction of a rigorous proof of the validity (mathematically speaking) of Position Based Dynamics. An interesting future line of work may be to contemplate the iterative PBD algorithm from the point of view of other kind of iterative algorithms such as Gauss-Newton algorithm (see [4]) and try to figure out the similarities and discrepancies between the two methods. This path might lead to the main peculiarities of the PBD algorithm and allow to set up a procedure for checking the stability or convergence of the method.

Chapter 5

Conclusion and future work

As it can be seen this work presents two clearly and distinct parts. In chapter 2 and chapter 3 we have explained some of the most used simulation methods for elastic materials emphasising the mathematical part such as the functional analysis of the Finite Element Method or the matrix distance approach for the Mass-Spring Model fitting. We have explained that, although this methods have some good properties, they are not quite adequate for the video game environment. In this moment, we have explained a relatively new (2007) method for these cases, and the algorithm that would be essential in this work: Position-Based Dynamics. After an overview of the algorithm and the justification of its success we focus on the nucleus part of the algorithm, the solver. We have also explained a method for convergence acceleration presenting a semi-iterative method, easy to understand and appropriate for our problem.

The material presented until chapter 3 is a compilation, extraction and summarizing of several scientific papers (mostly from the computer graphics area) and books (some of them purely mathematical). With all this information understood and embraced we have bumped into two main problems which solution could be in our scope: the parameter fitting of PBD for elastic materials and the convergence analysis.

The second part of the document, starting in chapter 4 comprises the core of this work. As we have mentioned throughout that chapter, chapter 4 is an original contribution of this work, that has been accepted for presentation in CEIG'16, the XXVI Spanish Computer Graphics Conference, under the title "Characterisation of Position Based Dynamics for Elastic Materials". Hence, the main contributions of this work have been:

1. We have proposed a general procedure for the parameter fitting in order to reproduce a proper elastic behaviour has been given. As in the reasoning of this algorithm we have dealt with generic constraints it can be used with any kind of geometric constraints, no matter how complex they are. Since the behaviour or PBD is dependant on the time step, we have found a closed expression, made as a justified suggestion, of the choice of the time step Δt and the cost function of the optimisation strategy. We have obtained numerical results with an hexahedral element that show that adding the volume constraint to the simulator has been a good choice.

2. We have presented closed formulae that connect the non-physical parameters of Position Based Dynamics with the elasticity theory parameters: Young's modulus and Poisson's ratio. This fact allows to obtain with barely computational cost how PBD has to be initialise in order to simulate no matter what material.

3. In the convergence field, we have written Position Based Dynamics as a whole, this is, matricially. We have analysed the convergence hypothesis of PBD seeing it as root-finding problem, comparing it with some more known methods such as Newton-Raphson.

4. We have developed a preliminary analysis on what assumption has been done for the convergence of PBD, based on a bound of a matrix of the iterative process. This can serve as initial point to demonstrate that, indeed, Position Based Dynamics is at least in most of the practical cases a convergent iterative method.

Despite this contribution some questions have come up that can be an adequate future work from all the previous investigation.

We have proposed one particular optimisation strategy but maybe there are other more suitable cost functions based so much in the definition of the stiffness matrices distance as the error but in the final position of the simulation, for example. Here, we have noticed that beyond certain value ($\nu = 0.29$) the importance of the constraints begun to change but we do not know the reason of this value, or if there a point of inflexion in any simulated object.

For testing our procedure we have executed it in a single cube. This can be repeated for cubes of different size, for tetrahedra (the other widely used element in digital simulation) and even with a more complex topology objects. Moreover, this is all based on non-auxetic elastic materials but there is not neither work done in the field of auxetic materials (negative Poisson's ratio) neither other type of materials such as viscoelastic or hyper-elastic objects.

We have remarked several times that all the reasoning is for one iteration, we will study how this outcomes change when more iterations are introduced. Possibly, with the adding of a variable number of iterations the material could become more rigid so a relaxation in the integration time step could be done.

In the numerical experiments we have tuned four parameters, gathering all the same kind of distance constraints (edge, face, diagonal...) with one unique parameter for each type. We will analyse the possibility of not making this assumption and taking different parameters for each constraint. This could be a direct way of simulating anisotropy materials, but we do not know with the analysis done how the results could be.

We hypothesise also that a data-driven approach could result in a better behaviour of PBD, since this wiil allow to define an error measure based on the deformations during the simulation, instead of using a lineal spectral analysis.

Finally, in the convergence field has remained one important, and open question: in the sequential algorithm (Gauss-Seidel way) we have pointed out that the stiffness even the result of the PBD simulation depends on the order in which the constraints are applied. We may explore which is the best strategy to avoid, or at least minimise this dependence. Some authors affirm that in over-constrained situations, the process can lead to oscillations if the order is not kept constant, but numerical proof of this is not presented. What most of the authors do is go over all the constraints in a certain order and, in the same iteration, go over them again but in reverse order in order to do the system symmetric. But, the option of randomising the constraint order in each iteration has not been studied. Neither does what would be the best strategy from a probabilistic point of view of doing this or if is possible to relate the stiffness parameters with the order in which the constraints are applied and optimise this.

Appendix A

The variational principles of mechanics

A.1 The principal viewpoint of analytical mechanics

It frequently happens that certain kinematical conditions exist between the particles of a moving system which can be started *a priori*. For example, the particles of a solid body may move as if the body were "rigid", which means that the distance between any two points cannot change. Such kinematical conditions do not actually exist on *a priori* grounds. They are maintained by strong forces. It is of great advantage, however, that the analytical treatment (which considers the whole system, instead of the vectorial treatment which treats particles individually) does not require the knowledge of these forces, but can take the given kinematical conditions for granted. We can develop the dynamical equations of a rigid body without knowing what forces produce the rigidity of the body. Similarly we need not know in detail what forces act between the particles of a fluid. It is enough to know the empirical fact that a fluid opposes by very strong forces any change in its volume, while the forces which oppose a change in shape of the fluid without changing the volume are slight. Hence, we can discard the unknown inner forces of a fluid and replace them by the kinematical conditions that during the motion of a fluid the volume of any portion must be preserved.

A.2 Auxiliary conditions. The Lagrange multipliers method

The configuration space in which the point P can move may be restricted to less than n dimensions by certain kinematical relations which exist between the coordinates. Such kinematical conditions are called "auxiliary conditions" or "constraints" of the given variation problem. Let's see that we can reduce an optimization (variation) problem with auxiliary conditions to a usual system of equations:

Let $\mathbf{P} = (u_1, \ldots, u_n)$ a set of coordinates, the function we want to optimize $F(\cdot)$ subject to a set of constraints $f_1(\cdot) = 0, \ldots, f_m(\cdot) = 0 \qquad 1 \leq m < n$.

Definition 9 (Virtual work)**.** The work of a force acting on a particle as it moves along a displacement will be different for different displacements. Among all the possible displacements that a particle may follow, called **virtual displacements**, one will minimize the action, and, therefore, is the one followed by the particle by the principle of least action. The work of a force on a particle along a virtual displacement is known as the **virtual work**.

Example 2. Let us consider for example a marble which is at rest at the lowest point of a bowl. The actual displacement of the marble is zero. It is our desire, however, to bring the marble to a neighbouring position in order to see how the potential energy changes. A displacement of this nature is called a "virtual displacement". The term "virtual" indicates that the displacement was intentionally made in any kinematically admissible manner. Such a virtual and infinitesimal change of position is called briefly a **variation** of the position.

It was Lagrange's ingenious idea to introduce a special symbol for the process of variation, in order to emphasize its virtual character. This symbol is δ. The analogy to d brings to mind that both symbols refer to *infinitesimal changes*. However, d refers to an *actual*, δ to a *virtual change*. In a plenty of problems of variational physics this distinction is of vital importance.

In accordance with this notation we write the infinitesimal virtual changes of a set $u_1, \ldots, u_n$ of coordinates in the form

$$\delta u_1, \delta u_2, \ldots, \delta u_n$$

The corresponding change of the function F becomes by the rules of elementary calculus (a sort of the chain rule)

$$\delta F = \frac{\partial F}{\partial u_1}\delta u_1 + \frac{\partial F}{\partial u_2}\delta u_2 + \ldots + \frac{\partial F}{\partial u_n}\delta u_n \tag{A.1}$$

This expression is called the **first variation** of the function F.

With this background let's proof the Lagrange multipliers method:

Theorem 5. *Given a set of coordinates $\boldsymbol{P} = (u_1, \ldots, u_n)$, a function $F : \mathbb{R}^n \to \mathbb{R}$ subject to a set of restriction $f_i : \mathbb{R}^n \to \mathbb{R} \quad i = 1, \ldots, m < n$ is equivalent asking for the stationary value of the function*

$$\overline{F(\boldsymbol{P})} = F(\boldsymbol{P}) + \lambda_1 f_1(\boldsymbol{P}) + \ldots + \lambda_m f_m(\boldsymbol{P}) \tag{A.2}$$

Proof. Because the points in which the constraints are satisfied is a set of general constraints, then, if we calculate the variation of the functions:

$$\delta f_i = \frac{\partial f_i}{\partial u_1}\delta u_1 + \ldots + \frac{\partial f_i}{\partial u_n}\delta u_n = 0 \qquad i = 1, \ldots, m \tag{A.3}$$

while the fact that the variation of F has to vanish at a stationary value, gives

$$\delta F = \frac{\partial F}{\partial u_1}\delta u_1 + \ldots + \frac{\partial F}{\partial u_n}\delta u_n = 0 \tag{A.4}$$

If the δu_k were independent of eacht other, (A.4) would lead to the vanishing of each $\frac{\partial F}{\partial u_k}$ but due to (A.3) this is not true. Let us modify the expression (A.4) by adding the left-hand sides of the equations (A.3) after multiplying each one by some undetermined $\lambda-$factor. We thus get

$$\sum_{k=1}^{n} \left(\frac{\partial F}{\partial u_k} + \lambda_1 \frac{\partial f_1}{\partial u_k} + \ldots + \lambda_m \frac{\partial f_m}{\partial u_k} \right) \delta u_k = 0$$

We wish to express the last m δu_k in terms of the independent δu_k. This can be accomplished by the proper choice of the $\lambda-$factors, so that

$$\frac{\partial F}{\partial u_k} + \lambda_1 \frac{\partial f_1}{\partial u_k} + \ldots + \lambda_m \frac{\partial f_m}{\partial u_k} = 0 \qquad k = n-m+1, \ldots, n$$

This leaves

$$\sum_{k=1}^{n-m} \left(\frac{\partial F}{\partial u_k} + \lambda_1 \frac{\partial f_1}{\partial u_k} + \ldots + \lambda_m \frac{\partial f_m}{\partial u_k} \right) \delta u_k = 0 \tag{A.5}$$

Now since only those δu_k remain which can be chosen arbitrarly, the only way in which (A.5) is accomplished is that each factor vanish. In the final analysis we have the equations

$$\frac{\partial F}{\partial u_k} + \lambda_1 \frac{\partial f_1}{\partial u_k} + \ldots + \lambda_m \frac{\partial f_m}{\partial u_k} \qquad k = 1, \ldots, n$$

which can be considered as obtained from the variational principle

$$\delta F + \lambda_1 \delta f_1 + \ldots + \lambda_m \delta f_m = 0$$

considering *all* the u_k as independent variables. Thus we can express it in the form of (A.2). □

This method yields n equations. In addition to those equations we have to satisfy the m auxiliary conditions. This gives $n+m$ equations for the $n+m$ unknowns

$$u_1, \ldots, u_n; \lambda_1, \ldots, \lambda_m$$

Remark 2 (Physical meaning of the Lagrange multipliers)**.** Let us assume that we have a mechanical system of n degrees of freedom, characterised by the generalized coordinates $q_1, \ldots, q_n$ and that there is a kinematical condition given in the form

$$f(q_1, \ldots, q_n) = 0 \tag{A.6}$$

Then, being V the potential energy (usually what we want to minimize in mechanical problems) the Lagrange multiplier method requires that

$$\delta V + \lambda \delta f = 0$$

This equation, however can be expressed in the form $\delta\overline{V} = 0$ where $\overline{V} = V + \lambda f$. This modified potential energy $\overline{V}$ is, however, physically very plausible. If we do not restrict the variation of the configuration of the system by the condition (A.6) but permit *arbitrary* variations of the q_i, then not only the impressed forces will act but also the forces which maintain the given kinematical condition. Thus, the modification of the potential energy on account of the Lagrangian $\lambda-$method represents the potential energy of the forces which are responsible for the maintenance of the given auxiliary conditions in the "direction" of the variable we are deriving it. This is, the term we add with the Lagrange multipliers can be interpreted as **the force of reaction** that the constraint produces.

Appendix B

Code used

B.1 Fitting of the PBD stiffness matrix to the FEM stiffness matrix

We do not attach the PBD and FEM code since it is well known (even can be found in some forums) and this chapter would be too extensive.

The following code has been done with Python:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
Sample generation for the linearisation of a PBD cube with springs in
faces and volume constraint.

"""
from sys import path
path.append('FEM')

import numpy as np
import pbd
import elastic_prism as fem


from pbd_elastic_prism import build_prism

np.set_printoptions(precision=8,linewidth=175)

dens = 1.0
```

```
x = [ 1, 1,-1,-1, 1, 1,-1,-1]
y = [-1, 1, 1,-1,-1, 1, 1,-1]
z = [-1,-1,-1,-1, 1, 1, 1, 1]


pbds = pbd.PBDSystem()

pbds.build_cube(2,2,8*dens)

pbds.set_niters(1)




def get_delta(u):
    p = np.array(pbds.particles.p)
    pbds.particles.p = p + u

    pbds.delta = np.zeros((24,1))
    for c in pbds.constraints:
        dp = c.project_positions()
        pbds.store_delta(dp)

    pbds.particles.p = np.array(p)

    return pbds.delta

def compute_jacobian(dx):
    J = np.zeros((24,24))
    for i in range(24):
        u = np.zeros((24,1))
        u[i] = dx
        dp = get_delta(u)

        J[i] = -1.0/dx*dp.transpose()[0]

    return J


def jacobian_function(k1,k2,k3,k4):
    for c in pbds.l1_constr:
        c.stiffness = k1
    for c in pbds.l2_constr:
        c.stiffness = k2
    for c in pbds.l3_constr:
        c.stiffness = k3
    for c in pbds.vol_constr:
        c.stiffness = 0.5*k4
    pbds.set_niters(1)

    dx = 1e-6
    return compute_jacobian(dx)


def error_value(k):
    #[k1,k2] = k
    [k1,k2,k3,k4] = k
    hz2 = np.linalg.norm(K) #E/((1+nu)*(1.0-2*nu))
    Jac0 = jacobian_function(k1,k2,k3,k4)
    Jac=hz2*Jac0
    errmat = Jac - K
    err = np.linalg.norm(errmat)
    print('in err',E,nu,hz2,np.linalg.norm(Jac0),err,k)

    return err




for i in range(8):
    pbds.particles.set_pos(i,(x[i],y[i],z[i]))

build_prism(pbds,[0,1,2,3,4,5,6,7,8],1,1,1,1)


from scipy.optimize import minimize


for E in [1.0]:
    for pseudonu in range(0,50):
        nu=pseudonu/100.0

        # FEM

        K = fem.build_k(nu,E)

        print("---\nAjuste para E="+str(E)+" y nu="+str(nu)+":\n")
        bnds = ((0,1),(0,1),(0,1),(0,1))
        res = minimize(error_value,[0.5,0.5,0.5,0.5],bounds=bnds)

        print(res)
        rel_err = res.get('fun')/np.linalg.norm(K)
        print('err. '+str(res.get('fun')))
        print('norm(K) = '+str(np.linalg.norm(K)))
        print('rel err. '+str(rel_err))
        dt_inv= np.sqrt(np.linalg.norm(K))
        # np.sqrt(E/((1+nu)*(1.0-2*nu)))
        print('nsteps='+str(dt_inv))
        print("--------")

        [k1,k2,k3,k4]=res.get('x')
        Jac = (dt_inv**2)*jacobian_function(k1,k2,k3,k4)
        print(np.linalg.norm(Jac),np.linalg.norm(Jac-K))

        save_file = True
        if save_file:
            file_ks = open("data/fit_ks_buenas.dat",'a')
            file_ks.write(str(E) + ' '
                        + str(nu) + ' '
                        + str(k1) + ' '
                        + str(k2) + ' '
                        + str(k3) + ' '
                        + str(k4) + ' '
                        + str(dt_inv) + ' '
                        + str(rel_err) + '\n')
            file_ks.close()
    #
#
```

B.2 Fitting of the PBD stiffness matrix to the FEM stiffness matrix disregarding the volume constraint

The following code has been done with Matlab:

```
function [res] = param_adjust(pos,E,nu,fileID)
    %   INPUT:
    %       p: Positions (p1,...,pN)
    %       E: Young's modulus
    %      nu: Poisson ratio
    %
    %  OUTPUT:
    %     res: file with the results

    aux = mat_fem(pos, E, nu);
    dt2=norm(aux,'fro');
    err = @(x) double(norm(dt2*mat_pbd(pos, (sin(x(1))+1)/2,...
    (sin(x(2))+1)/2, (sin(x(3))+1)/2) - aux,'fro'));
    [k,err_abs] = fminsearch(err,[1 1 1]);


    res = [E nu (sin(k(1:3))+1)/2 sqrt(dt2) 0 err_abs/E] ;

    fprintf(fileID,'\n %d %f %f %f %f %f %f %f %f \r\n',res) ;
end
```

B.3 Polynomial fitting

Once the data has been obtained with the code in section B.1 we make a polynomial regression. The input has been a matrix with all the parameters needed, and the output the plots.

The following code has been done with Matlab

```
coefk11=polyfit(nu(1:30),k1(1:30),2);
coefk12=polyfit(nu(30:end),k1(30:end),2);

coefk21=polyfit(nu(1:30),k2(1:30),2);
coefk22=polyfit(nu(30:end),k2(30:end),2);

coefk31=polyfit(nu(1:30),k3(1:30),1);
coefk32=polyfit(nu(30:end),k3(30:end),2);

coefkvol2=polyfit(nu(30:end),kvol(30:end),3)
```

Bibliography

[1] K. Bathe. *Finite Element Procedures.* Prentice Hall, Pearson Educatio, Inc., 2006.

[2] J. Bender, D. Koschier, P. Charrier, and D. Weber. Position-based simulation of continuous materials. *Computers and Graphics*, 44:1–10, 2014.

[3] J. Bender, M. Müller, M. Otaduy, M. Teschner, and M. M. A survey on position-based simulation methods in computer graphics. *Computer Graphics forum*, 33(6):228–251, May 2014.

[4] A. Björck. *Numerical methods for least squares problems.* SIAM, Philadelphia, 1996.

[5] S. Bouaziz, S. Martin, T. Liu, L. Kavan, , and M. Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Transactions on Graphics (SIGGRAPH)*, 33(4):154:1–154:11, 2014.

[6] J. Celigüeta. *Método de los Elementos Finitos para el Análisis Estructural.* UNICOPIA C.B., 2000.

[7] T. Chyou. Dynamics of a cube-shaped mass-spring network from the wolfram demonstrations project. `http://demonstrations.wolfram.com/DynamicsOfACubeShapedMassSpringNetwork/`.

[8] S. Conte and C. De Boor. *Elementary Numerical Analysis.* McGraw-Hill Book Company, 1980.

[9] R. Courant. Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society*, 49:1–23, 1943.

[10] F. Demengel and G. Demengel. *Functional Spaces for the Theory of Elliptic Partial Differential Equations.* Springer, 2012.

[11] K. Evans, M. Nkansah, I. Hutchison, and S. Rogers. Molecular network design. *Nature*, 124(353), 1991.

[12] A. Gelder. Approximate simulation of elastic membranes by triangulated spring meshes. *Journal of Graphics Tools*, 3(2):21–41, 1998.

[13] M. Hazewinkel. *Newton method*. Encyclopedia of Mathematics, Springer, 2001.

[14] E. Hinton and B. Irons. Least squares smoothing of experimental data using finite elements. *Strain*, 4:24–27, July 1968.

[15] R. Kikuuwe, H. Tabuchi, and M. Yamamoto. An edge-based computationally efficient formulation of saint venant-kirchhoff tetrahedral finite elements. *ACM New York, NY, USA*, 28(1), 2009. doi:`10.1145/1477926.1477934`.

[16] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the nelder-mead simplex method in low dimensions. *SIAM Journal of Optimization*, 9(1):112–147, 1998. doi:`10.1137/S1052623496303470`.

[17] E. Lengyel. *Mathematics for 3D Game Programming and Computer Graphics*. Cengage Learning PTR, 3rd edition, 2011.

[18] T. Liu, A. Bargteil, J. O'Brien, and L. Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, 32(6), 2013.

[19] T. Liu, A. W. Bargteil, J. F. OBrien, and L. Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 32(6):214:1–214:7, 2013.

[20] B. Lloyd, G. Szekely, and M. Harders. Identification of spring parameters for deformable object simulation. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):1081–1094, 2007.

[21] M. Macklin, M. Müller, N. Chentanez, and T. Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2014*, 33(4), July 2014.

[22] M. Müller. Hierarchical position based dynamics. In F. Faure and M. Teschner, editors, *Workshop in Virtual Reality Interactions and Physical Simulation "VRI-PHYS" (2008)*. The Eurographics Association, 2008. doi:`10.2312/PE/vriphys/vriphys08/001-010`.

[23] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. Position based dynamics. In 3^{rd} *Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS"*, 2006.

[24] K. Owen. Area and Volume Calculations. `http://archive.gamedev.net/archive/reference/articles/article2247.html`, 2005. (Online; accessed 07-July-2016).

[25] C. Rodero, P. Real, P. Zuñeda, C. Monteagudo-Mañas, M. Lozano, and I. García-Fernández. Characterisation of position based dynamics for elastic materials. In *XXVI Spanish Computer Graphics Conference "CEIG"*, 2016.

[26] G. San Vicente. *Designing deformable models of soft tissue for virtual surgery planning and simulation using the Mass-Spring Model.* PhD thesis, School of Engineering, University of Navarra, 2011.

[27] G. San-Vicente, I. Aguinaga, and J. Celigueta. Cubical mass-spring model design based on a tensile deformation test and nonlinear material model. *IEEE Transactions on Visualization and Computer Graphics*, 18(2):228–241, 2012.

[28] J. Synge. *The hypercircle in mathematical physics.* Cambridge at the University Press, 1957.

[29] H. Wang. A chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 34(6), 2015.

[30] S. Wang and J. Yang. An improved finite element model for craniofacial surgery simulation. *International Journal of Computer Assisted Radiology and Surgery*, 2009.

[31] L. Zhang, A. Gerstenberger, X. Wang, and W. Liu. Immersed finite element method. *Computer Methods in Applied Mechanics and Engineering*, 193(22):2051–2067, 2004.

[32] L. Zhilin, Q. Zhonghua, and T. Tao. *Numerical Solutions of Partial Differential Equations An Introduction to Finite Difference and Finite Element Methods.* Hong Kong Baptist University, 2012. `http://www4.ncsu.edu/~zhilin/TEACHING/MA587/`.

[33] O. Zienkiewicz. *The finite element method.* London : McGraw-Hill, 1977.

Index