



Universidade do Minho
Escola de Engenharia

João Gonçalo Botica Ribeiro da Silva

**Real-time Structural Mechanics
Simulation for Tetrahedral Meshes
with Dynamic Topology**



Universidade do Minho

Escola de Engenharia

João Gonçalo Botica Ribeiro da Silva

**Real-time Structural Mechanics
Simulation for Tetrahedral Meshes
with Dynamic Topology**

Tese de Mestrado
Mestrado em Informática

Trabalho efectuado sob a orientação do
Prof. Doutor Adérito Fernandes Marcos

Dezembro de 2009

É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA TESE APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

Universidade do Minho, ___/___/_____

Assinatura: _____

Acknowledgements

I would like to thank Adérito Fernandes Marcos and Sebastian Peña Serna for supervising my work and my parents for their unconditional support. I cannot forget to thank Universidade do Minho and Fraunhofer IGD for providing me this opportunity and for their financial support. Only with these contributions this thesis was possible.

Statement

Real-time Structural Mechanics Simulation for Tetrahedral Meshes with Dynamic Topology.

Abstract

The simulation of meshes with dynamic geometry and topology (also known as dynamic meshes) is a new and unexplored field. Recently, motivated by the challenge of developing multidisciplinary solutions for exploring design variations, such as to simultaneously design and engineer products, the scientific community has started taking interest in dynamic meshes.

The traditional simulation procedure for meshes with constant topology builds a system of equations which then is solved to reveal the solution of the studied problem. The first approaches to dynamic mesh simulation reuse this procedure as many times as the number of topological changes suffered by the simulated object.

In this thesis, a new approach to the simulation of dynamic meshes is presented. This methodology is different from the previous developed approaches as it avoids the systematic rebuilds of the system of equations. Instead, a methodology was developed that quickly and locally revalidates the system so that it always represents the current state of the dynamic mesh.

Tests comparing the performance of the presented methodology with the previous approaches, show a significant reduction in the simulation time, achieving real-time performance for meshes with higher complexity.

Tema

Simulação de estruturas mecânicas em tempo real para malhas com topologia dinâmica.

Resumo

A simulação de malhas com geometria e topologia dinâmicas (também apelidadas de malhas dinâmicas) é um campo novo e pouco explorado. Recentemente, a comunidade científica começou a mostrar interesse na simulação destas malhas. Este interesse é especialmente motivado pelo desafio de desenvolver soluções multidisciplinares para a exploração de variações de design, tais como permitir que o desenho e a análise a nível da engenharia de um produto sejam efectuados em simultâneo.

O processo tradicional para a simulação de malhas com topologia constante constrói um sistema de equações que é depois resolvido para revelar a solução do problema estudado. As primeiras abordagens à simulação de malhas dinâmicas reutilizam este processo tantas vezes quanto o número de alterações topológicas sofridas pelo objecto em estudo.

Esta tese apresenta uma nova abordagem à simulação de malhas dinâmicas. A nova metodologia é diferente das abordagens anteriores na medida em que esta evita a reconstrução sistemática do sistema de equações. Um algoritmo foi desenvolvido para rapidamente e localmente revalida o sistema, de forma a que este represente sempre o estado actual da malha dinâmica.

Os resultados dos testes comparando a performance da metodologia proposta com as abordagens anteriores mostram uma redução significativa no tempo de simulação, alcançando performance em tempo real para malhas com maior complexidade.

Table of Contents

1	Introduction	1
1.1	Motivation	1
1.2	Objective	1
1.3	Thesis Organization	3
2	Review of the State of the Art	4
3	Static Finite Element Analysis	7
3.1	Continuum Mechanics	7
3.1.1	Displacement	8
3.1.2	Strain and Stress Fields	9
3.1.3	Constitutive Laws	10
3.2	Finite Element Method	10
3.2.1	Principle of Virtual Work.....	12
3.3	Stiffness Matrix	14
3.4	Loads and Constraints	15
3.5	Solver.....	15
4	Methodology.....	17
4.1	Computing the element stiffness matrices	19
4.2	Equivalent matrix representation	20
4.3	Build of the equivalent matrix.....	24
4.3.1	Constructing the neighboring information.....	24
4.3.2	Computing the edge matrices.....	25

4.3.3	Computing the diagonal matrices.....	26
4.4	Revalidation of the equivalent system.....	27
4.5	Solution of the equivalent system	29
4.5.1	Multiplication of the equivalent matrix with a vector.....	29
5	Results.....	32
5.1	Build and Multiplication of the matrix of coefficients	34
5.2	Solving with Preconditioned Conjugate Gradient (RealTime)	36
5.3	Topological changes and the corresponding revalidations	36
6	Conclusion.....	40
6.1	Future Work	40
7	Bibliography	42
APPENDIX A.....		45
APPENDIX B.....		46

Index of Figures

Figure 1.1 - Three simulation cycles on the traditional approach (on the left) and on the proposed approach (on the right) to dynamic mesh simulation..... 2

Figure 2.1 – A rectangle moving through fluid (from [KFCO06])..... 5

Figure 3.1 - A 1D problem using a cantilever..... 8

Figure 3.2 – A 2D cantilever beam discretized using triangular elements. 11

Figure 4.1 - Proposed procedure for the simulation of dynamic meshes. (GSM stands for Global Stiffness Matrix) 18

Figure 4.2 - Mesh composed of six elements. 21

Figure 4.3 - Graphical representation of a matrix of coefficients (left). The same matrix of coefficients represented as the equivalent matrix (right)..... 23

Figure 4.4 - A 2D mesh composed of three triangular elements 24

Figure 4.5 - Element contributions to the build of the third edge matrix (E2)..... 26

Figure 4.6 - Element contributions to the build of the second diagonal matrix (D1). 27

Figure 4.7 - Multiplication of the equivalent matrix (represented as a normal matrix for simplification) with a vector. The green and red lined colored boxes indicate the used values when the multiplication iterates on edge E1..... 31

Figure 5.1 - Bunny model..... 33

Figure 5.2 - Gargoyle model..... 33

Figure 5.3 - Hand model..... 34

Figure 5.4 - Dragon model. 34

Figure 5.5 - Decimated bunny..... 37

Figure 5.6 - Mirrored gargoyle..... 38

Figure 5.7 - Scaled dragon..... 39

Index of Tables

Table 4.1 - Characteristics that define the overall linearity of the simulation. (Only one example is shown per option for simplicity)	18
Table 4.2 - Elements needed to calculate the stiffness of each edge.	22
Table 4.3 - Elements needed to calculate the stiffness of each vertex.	22
Table 5.1 - Topological information of the meshes used for the measurements.	33
Table 5.2 - Measurements for the build and multiplication processes (in milliseconds).	34
Table 5.3 - Meshes with real time performance (time in milliseconds).	36
Table 5.4 - Measurements for the decimate operation (in milliseconds).	37
Table 5.5 - Measurements for the mirror operation (in milliseconds).	37
Table 5.6 - Measurements for the scale operation (in milliseconds).	38

1 Introduction

1.1 Motivation

Structural mechanics is the field of continuous mechanics that deals with the computation of deformations and stresses within structures. Structural mechanics simulators are widely used to drive the engineering design process of mechanical structures, such as automobiles, ships and airplanes. In the context of simulation, these structures are digitally represented by means of their geometric shape and material properties. In the simulation process, the shape of these structures is typically handled as a constant input, in other words, the geometry remains constant and only the material properties are modified. Nowadays, the exploration of design variations within the simulation stage has motivated the modification of the shape, leading to changes in the geometry and topology of the mesh, also called dynamic meshes. The dynamic meshes field is therefore new and unexplored, having so far very few approaches to its simulation.

The traditional simulation procedure for meshes with constant topology starts by building a system of equations based on the involved physical laws and properties of the object. This system of equations is then solved to reveal the displacement of the object. The first implementations of simulation for dynamic meshes iterate on this procedure, building and solving a system of equations every time the object suffers a topological change ([KFCO06]). Although this technique works, it is very expensive due to the successive rebuilds of the system of equations.

1.2 Objective

The objective of this thesis work is to devise an efficient procedure for the simulation of dynamic meshes. The matrix of the aforementioned system of equations is known in the

simulation context as the global stiffness matrix. This matrix defines the elasticity of the simulated object and thus when the slightest topology change occurs, this matrix becomes invalid, as it no longer represents the current object. Opposed to the reconstruction of the whole matrix done by other approaches to dynamic mesh simulation, this thesis presents a methodology that revalidates the matrix with lower resource consumption (both in processor usage as in memory access). Using a revalidation method instead of the rebuild, the simulation cycle time decreases (as shown in Figure 1.1) resulting in an increase of the number of frames per second.

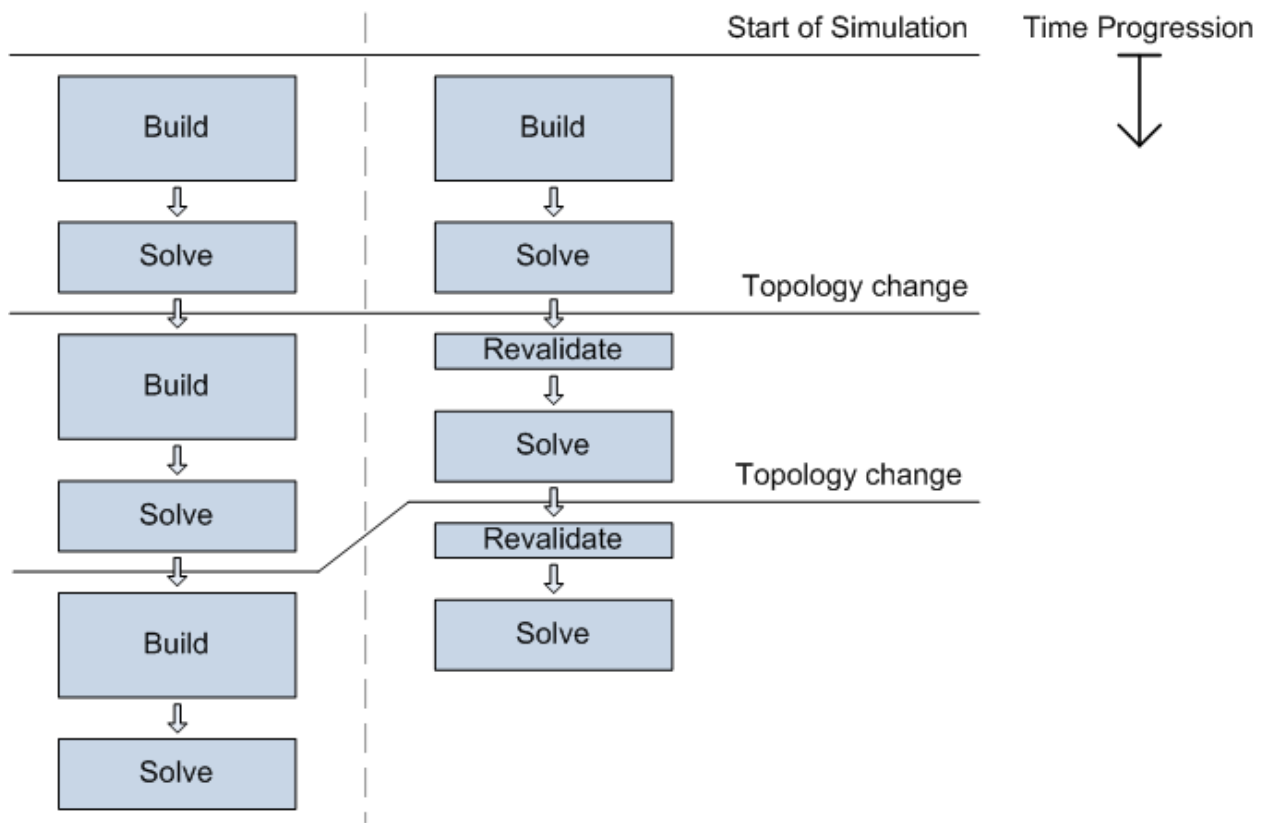


Figure 1.1 - Three simulation cycles on the traditional approach (on the left) and on the proposed approach (on the right) to dynamic mesh simulation.

The presented methodology revalidates the global stiffness matrix after every topological change. It takes as input the information of the applied topological changes that combined with the mesh information reveals which are the entries of the matrix that were affected and therefore need to be recomputed.

In order to test the performance of the proposed methodology a static linear finite element simulator was developed. Results revealed that the implemented revalidation strategy is a success compared to the reconstruction of the global stiffness matrix. It solves in real-time for meshes of 30,000 elements.

1.3 Thesis Organization

This thesis is structured in the following way:

Chapter 2 - Review of the State of the Art. This chapter describes the literature survey done to assess the state of the art in terms of: what other approaches have been developed so far to simulate meshes with dynamic topology; the simulation characteristics of the studied problem; and the most appropriate solvers to find the solution of the system of equations.

Chapter 3 - Static Finite Element Analysis. In this chapter the key concepts of the static finite element analysis are presented. This is an introductory chapter meant to ease the reader to the simulation process.

Chapter 4 - Methodology. This chapter describes the methods and algorithms developed, in order to implement, test and prove the proposed methodologies. Section 4.1 explains how to build the stiffness matrix from the involved physical laws. Section 4.2 and 4.3 describe the concept and the build of the equivalent matrix representation. Section 4.4 explains how the linear system is revalidated after a geometrical or topological change. Section 4.5 shows the adaptations made to the solving technique to allow the usage of the equivalent matrix.

Chapter 5 - Results. This chapter offers performance comparisons between the different implemented techniques. It is divided into sections where the test being performed is explained, the results are shown and a discussion is provided.

Chapter 6 - Conclusion. In this chapter the conclusions from this thesis work are drawn out. It ends with a couple of suggestions for future work that can increase the performance of the proposed methods.

2 Review of the State of the Art

This thesis work started with the study of different techniques in computer graphics such as deformable models ([NMK+06]), shape modeling ([Ale06]), animation ([MSJT08]) and simulation ([BFMF06]), in order to understand how linear systems are used within this community. The study of these techniques revealed, that linear systems are used in a classical way and that there are no optimization procedures to minimize its build time. Hence, partial differential equations ([Lan03]) and discretization techniques were analyzed, particularly the Finite Element Method ([Hug00, She94]), in order to explore the requirements for building linear systems.

Additionally, aiming at understanding how the solution of a linear system is computed, the literature regarding iterative solvers ([SVDV00]) and specially the conjugate gradient method ([She94]) was revised. Some solvers might be better suited than others depending on the characteristics of the problem to be solved. Therefore, the revision of the literature was done having in mind the solid mechanics problem [Bow09], so that the properties of the generated systems (symmetry, definiteness, among others) would be taken into account when choosing a solver.

Augarde et al. [ARS06] explained that in the linear elasticity problem, the Galerkin method causes the stiffness matrix to be symmetric and positive definite. This fact makes the Conjugate Gradient Method a suitable solver for the linear system of equations yielded in the linear elasticity problem. Saad and Vorst presented, in [SVDV00], an in-depth historical perspective of iterative solutions of linear systems and they attributed their origin to the work of Gauss in the early nineteenth century and show how the main contributions over the years led to the iterative solvers we have nowadays.

The studied related work suggests that the Conjugate Gradient method is currently the most appropriate solver for computing the solution of a linear system of equations in an iterative form and without using a hierarchy of discretizations or adaptivity methods. For this reason, the

Conjugate Gradient Method is the chosen solver for this thesis study. The implementation of this solving technique is based on [She94], where Shewchuk presents a practical explanation on how the Conjugate Gradient works and also shows the building blocks and its interconnections, i.e. the method of Steepest Descent, the method of Conjugate Directions and finally their relations within the Conjugate Gradient method.

As mentioned before, this thesis work was motivated by the aim to find an algorithm, which is able to handle the simulation of dynamic meshes at interactive rates. The computer graphics community has not deeply dealt with the simulation of changing meshes, but there are some interesting approaches. Bro-Nielsen ([BNC96, BN96]) presented the advantages of condensed or Fast Finite Elements for deformable models in surgery simulation, where only the surface of the volumetric model is considered for the simulation. Gissler et al. [GBT07] proposed recently constraint sets for FE models, where topological changes (merging and breaking) of deformable tetrahedral meshes are supported, by means of replacing mass points by mass portions (in a constraint set) according to the number of incident tetrahedra.

Klinger et al. [KFCO06] developed a fluid animation application, which requires the remesh of the whole model after every iteration (or mesh change), leading also to a rebuild of the linear system (see Figure 2.1).

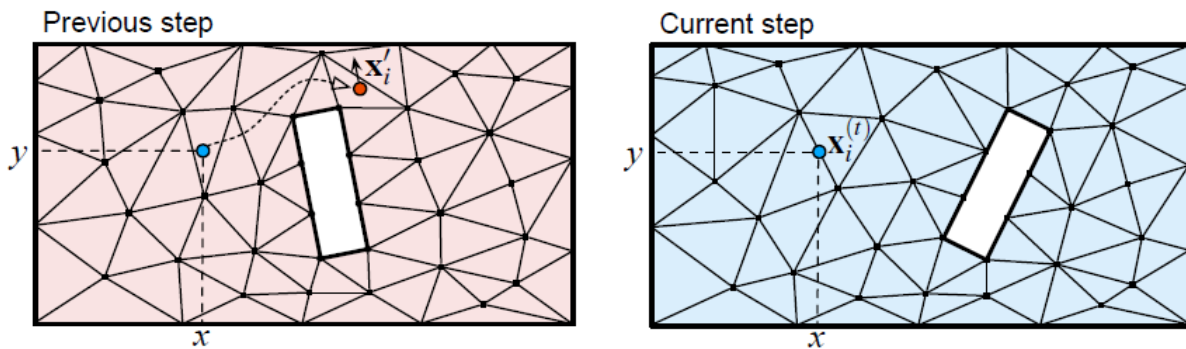


Figure 2.1 – A rectangle moving through fluid (from [KFCO06]).

Von Funck et al. [vFTS06] developed an algorithm for shape deformation based on time-dependent divergence-free vector fields, for which they need to build a linear system, in order

to find the path between steps. They also implemented a remesh step, since the deformation of the mesh between steps led to poor quality triangles, which affected the convergence of the linear system. Hence, they also need to rebuild the linear system after every remesh step.

There are several algorithms dealing with the simulation of physically-based deformable models. For example Mezger et al. [MTPS08] proposed a real time physically-based shape editing algorithm based on the simulation of the mechanical properties of the model with a Finite Element Method discretization. However they used computing meshes with few tetrahedral elements (up to 1,500) and the rebuild of the linear system (for every geometry change, the topology is constant) did not cause any performance problem. These kinds of simulations with bigger meshes (around 10,000 tetrahedral elements) will not achieve real time performance.

Based on the performed studies it is foreseen that several applications in computer graphics can benefit from the developed methodology, by improving their convergence and their performance, reducing the number of iterations and the computation time. The methodology proposed in this thesis, will specially contribute to the rapid simulation of dynamic meshes. To the best of the author's knowledge, there are no investigations in the same direction as the proposed in this thesis. There have been made several efforts regarding the improvement of solvers, either with new methods or with parallelization techniques on the CPU or the GPU, but there is not enough information about the intelligent handling and processing of linear systems.

3 Static Finite Element Analysis

The static finite element simulation can be divided in various steps: The problem under study is defined based on the object, the material properties and the user defined constraints. Then, a linear system of equations is built using the finite element method and the physical laws under study. Finally, this system of equations is solved to reveal the solution of the simulation.

Here follow the simplified methods needed to perform static structural analysis simulation using the finite element method (these are explained in more detail on the following sections):

1. Divide the object under study into elements;
2. Define shape functions;
3. Compute the stiffness matrices for all elements;
4. Assemble the matrices into the global stiffness matrix;
5. Modify the global stiffness matrix to enforce the boundary constraints;
6. Solve the system $Ku = f$ to find the displacements;
7. Calculate the strains;
8. Calculate the stresses.

3.1 Continuum Mechanics

Computer simulation often uses Continuum Mechanics to reproduce the behavior of materials modeled as a continuum. Even though all matter is made of atoms separated by empty space (which leads to heterogeneity on the substance of a body), the continuum concept assumes that matter is evenly distributed throughout a body and that it fills completely the space occupied by that body.

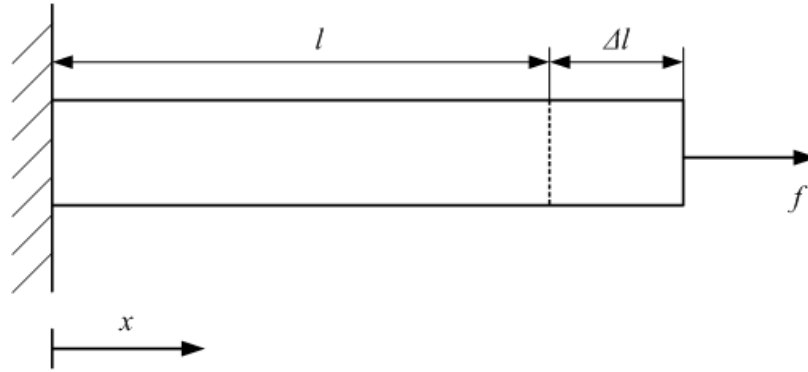


Figure 3.1 - A 1D problem using a cantilever¹.

There are three key quantities that describe the behavior of a continuous object: displacement, stress and strain. Consider the one dimensional problem shown in Figure 3.1. The beam with cross sectional area A and original length l suffers an expansion of Δl when the force f is applied. These quantities along with the material property E can be related via Hooke's law:

$$\frac{f}{A} = E \frac{\Delta l}{l} \quad (3.1)$$

The constant E is the Young's modulus of the beam's material and it represents the elastic stiffness of the material. Analyzing (3.1) it is easy to observe some logic properties such as: the bigger the force applied per area, the bigger will be the beam's relative elongation; and stiffer materials will elongate less, provided they have the same area and length.

3.1.1 Displacement

A mechanical object is usually defined by its shape and the properties of the object's material. The undeformed shape, also known as equilibrium configuration or rest shape, describes the object in a state where no forces are being applied and therefore the displacement is null. To study the behavior of an object, the coordinates of a point in an undeformed shape will be henceforward referred to as x . Applying forces to an object causes stress inside it and leads to displaced points. These displaced points have new locations $p(x)$. The displacement suffered by a point will be referred to as $u(x)$ where

¹ A cantilever is a beam supported on only one end.

$$u(x) = p(x) - x \quad (3.2)$$

3.1.2 Strain and Stress Fields

The strain ($\varepsilon = \Delta l/l$) represents the relative elongation of the object. The strain field is usually irregular over the object since the strain is function of the displacements and because some parts of the beam can deform more than others. In a three dimensional problem the strain is no longer a scalar, it is represented by a symmetric 3 by 3 tensor:

$$\varepsilon = \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{xy} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{xz} & \varepsilon_{yz} & \varepsilon_{zz} \end{bmatrix} \quad (3.3)$$

The strain takes this form so it can represent the relative elongation in any direction with respect to any spatial variable (x , y or z). The strain computation can be based on three theories: Infinitesimal strain theory, finite strain theory and large-displacement theory. The infinitesimal strain tensor, most commonly known as Cauchy's strain tensor (equation (3.4)), linearly calculates strain from the gradient of the displacement field. The common non-linear alternative is the finite strain tensor, or Green's strain tensor (equation (3.5)), which is better suited for the simulation of large displacements.

$$\varepsilon_C = \frac{1}{2}(\nabla u + [\nabla u]^T) \quad (3.4)$$

$$\varepsilon_G = \frac{1}{2}(\nabla u + [\nabla u]^T + [\nabla u]^T \nabla u) \quad (3.5)$$

Since in three dimensions the displacement field has three components (u , v and w), the gradient of the displacement field is a 3 by 3 matrix where the three components are derived by the three spatial variables.

$$\nabla u = \begin{bmatrix} u_{,x} & u_{,y} & u_{,z} \\ v_{,x} & v_{,y} & v_{,z} \\ w_{,x} & w_{,y} & w_{,z} \end{bmatrix} \quad (3.6)$$

Stress defines the force applied ($\sigma = f/A$). Following the same logic applied to (3.3), the stress is also represented as a 3 by 3 tensor. Since both strain and stress are representable by

symmetric 3 by 3 matrices, one alternative representation is with vectors containing the 6 independent coefficients:

$$\varepsilon = [\varepsilon_{xx} \quad \varepsilon_{yy} \quad \varepsilon_{zz} \quad \varepsilon_{xy} \quad \varepsilon_{yz} \quad \varepsilon_{zx}] \quad (3.7)$$

$$\sigma = [\sigma_{xx} \quad \sigma_y \quad \sigma_{zz} \quad \sigma_{xy} \quad \sigma_{yz} \quad \sigma_{zx}] \quad (3.8)$$

3.1.3 Constitutive Laws

Constitutive laws relate physical quantities and in structural analysis one is needed to relate stress with strain. Hooke's law (3.1) relates both, so it can be re-written as

$$\sigma = E\varepsilon \quad (3.9)$$

This linear relationship can only be used for isotropic materials which means that the properties of the material are independent of direction in space. By substituting (3.7) and (3.8) in (3.9), the constant of proportionality E can be expressed as a 6 by 6 matrix resulting in the following equation

$$\begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{bmatrix} = \frac{E}{(1 + \nu)(1 - 2\nu)} \begin{bmatrix} 1 - \nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1 - \nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1 - \nu & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 - 2\nu & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 - 2\nu & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 - 2\nu \end{bmatrix} \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{xy} \\ \varepsilon_{yz} \\ \varepsilon_{zx} \end{bmatrix} \quad (3.10)$$

Equation (3.10) is called Hooke's law in stiffness form. The ν represents the Poisson's ratio which defines the change in volume caused by the stretching of the material and E is the aforementioned Young's modulus.

3.2 Finite Element Method

Newton's second law of motion $f = m\ddot{p}$ defines the relation between the applied forces and the kinematics of an object. Dividing the whole equation by the volume yields the equation of motion for infinitesimal volume elements:

$$\rho \ddot{p} = f(x) \quad (3.11)$$

Since $f(x)$ represents the global forces acting on point x , $f(x)$ can be decomposed into external forces (such as prescribed forces, gravity or collision forces) and internal forces (resulting from deformations). Thus, the partial differential equation governing elastic materials is:

$$\rho \ddot{p} = \nabla \cdot \sigma + f_{ext} \quad (3.12)$$

The Finite Element Method is a mathematic procedure used to find approximate solutions to partial differential equations. There are other possible numerical methods such as finite differences and finite volume methods, but the finite element is the proven standard in structural analysis. The finite element approach divides the entire domain into a set of elements. These elements can be triangles, tetrahedrons, bricks or other (there are various choices depending on the dimensionality of the problem) and they are spread through the domain in a manner that covers the whole domain without overlaps. Figure 3.2 shows a possible discretization of a two dimensional beam.

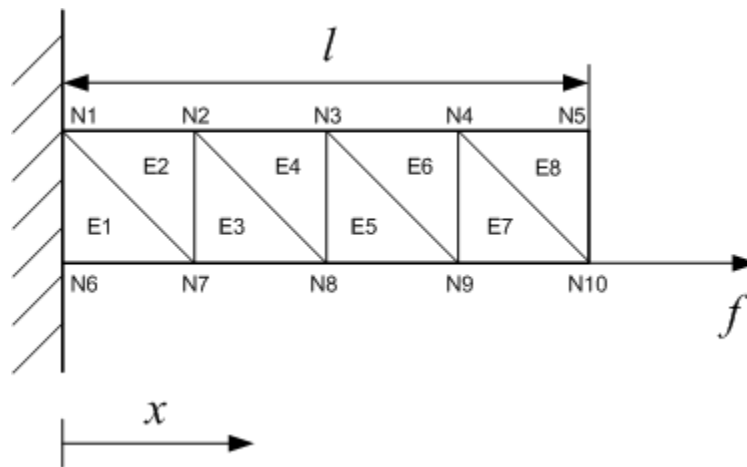


Figure 3.2 – A 2D cantilever beam discretized using triangular elements.

If the object is discretized, the governing equation also has to be discretized so that it can be applied to the points under study. Thus, equation (3.12) gets discretized into $M\ddot{u} + Ku = f$ where M is the diagonal mass matrix, K is the stiffness matrix and \ddot{u} is the acceleration caused by the external forces f . The focus of this thesis is static structural analysis, which means, the

problem is studied at an instant of time where the object is in equilibrium, in other words, the acceleration of every point on the (deformed) object is zero. Thus, the previous equation can be simplified into

$$Ku = f \quad (3.13)$$

3.2.1 Principle of Virtual Work

The Principle of Virtual Work is the basis of solid mechanics analysis when using the Finite Element Method. Going back to Newton's second law of motion, Equation (3.12) can be rewritten as:

$$\rho \frac{dv_i}{dt} = \frac{\partial \sigma_{ji}}{\partial y_i} + \rho b_i \quad (3.14)$$

Through the Principle of Virtual Work it is possible to convert the derivatives of the partial differential equations into an equivalent integral which is easier to handle numerically (also called weak form of the problem). The displacements, strains and stresses for linear elasticity can be found through these formulas respectively:

$$\int_R C_{ijkl} \frac{\partial u_k}{\partial x_i} \frac{\partial \delta v_i}{\partial x_j} dV - \int_R b_i \delta v_i dV - \int_{\partial_2 R} t_i^* \delta v_i dA = 0 \quad (3.15)$$

$$\varepsilon_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (3.16)$$

$$\sigma_{ij} = C_{ijkl} \varepsilon_{kl} \quad (3.17)$$

for all virtual velocity fields δv_i where $\delta v_i = 0$. C_{ijkl} represents the elastic constants of the object. To calculate the resulting displacement for a discrete set of points on the object, the displacement field needs to be discretized. The discretization of the formulas is achieved through the use of shape functions $N^a(x)$. These interpolations are function of position, they vary linearly within the element and each shape function has value one at one node of the

element and value zero at the other nodes. The discretized displacement and virtual velocity can be defined as

$$u_i(x) = \sum_{a=1}^n N^a(x) u_i^a \quad (3.18)$$

$$\delta v_i(x) = \sum_{a=1}^n N^a(x) \delta v_i^a \quad (3.19)$$

respectively. Substituting the interpolated fields (3.18) and (3.19) into the virtual work equation (3.15) results in

$$\int_R C_{ijkl} \frac{\partial N^b(x)}{\partial x_l} u_k^b \frac{\partial N^a(x)}{\partial x_j} \delta v_i^a dV - \int_R b_i N^a(x) \delta v_i^a dV - \int_{\partial 2R} t_i^* N^a(x) \delta v_i^a dA = 0 \quad (3.20)$$

which in matrix form shows a more clear representation of the linear system to be solved:

$$(K_{aibk} u_k^b - F_i^a) \delta v_i^a = 0 \quad (3.21)$$

$$K_{aibk} u_k^b = F_i^a \quad (3.22)$$

where

$$K_{aibk} = \int_R C_{ijkl} \frac{\partial N^a(x)}{\partial x_j} \frac{\partial N^b(x)}{\partial x_l} dV \quad (3.23)$$

$$F_i^a = \int_R b_i N^a(x) dV + \int_{\partial 2R} t_i^* N^a(x) dA \quad (3.24)$$

The shape function derivatives with respect to global coordinates are calculated as follows:

$$\frac{\partial N^a}{\partial x_j} = \frac{\partial N^a}{\partial \xi_i} \frac{\partial \xi_i}{\partial x_j} \quad (3.25)$$

The first parcel of the multiplication ($\partial N^a / \partial \xi_i$) are the shape functions with respect to a local, dimensionless, coordinate system (ξ) within the element. The second parcel is calculated as follows:

$$\frac{\partial \xi_j}{\partial x_i} = \left(\frac{\partial x_i}{\partial \xi_j} \right)^{-1} = \left(\sum_{a=1}^{N_e} \frac{\partial N^a(\xi)}{\partial \xi_j} x_i^a \right)^{-1} \quad (3.26)$$

The stiffness matrix of a tetrahedron can be computed by iterating on the four nodes (a and b) for all coordinates (i and k) on the formula:

$$k_{aibk} = \sum_{l=1}^{N_l} w_l C_{ijkl} \frac{\partial N^a(\xi_l)}{\partial \xi_p} \frac{\partial \xi_p}{\partial x_j} \frac{\partial N^b(\xi_l)}{\partial \xi_q} \frac{\partial \xi_q}{\partial x_i} J(\xi_l) \quad (3.27)$$

where J is the determinant of the Jacobian matrix (Jacobian determinant):

$$J = \det \begin{pmatrix} \frac{\partial x_1}{\partial \xi_1} & \dots & \frac{\partial x_n}{\partial \xi_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_1}{\partial \xi_j} & \dots & \frac{\partial x_n}{\partial \xi_j} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_1}{\partial \xi_n} & \dots & \frac{\partial x_n}{\partial \xi_n} \end{pmatrix} \quad (3.28)$$

and the components of the elastic modulus tensor C_{ijkl} are computed as

$$C_{ijkl} = \frac{E}{2(1+\nu)} (\delta_{il} \delta_{jk} + \delta_{ik} \delta_{jl}) + \frac{E\nu}{(1+\nu)(1-2\nu)} \delta_{ij} \delta_{kl} \quad (3.29)$$

where E represents the Young's modulus, ν the Poisson's ratio and δ the Kronecker delta.

3.3 Stiffness Matrix

The Stiffness Matrix is a key concept in the finite element method as it stores the material stiffness of an element with respect to all the degrees of freedom. The discretization step induces in the stiffness matrix two characteristics: symmetry and positive definiteness ([ARS06]). An element stiffness matrix's size depends on the number of nodes per element and on the number of dimensions of the problem. Considering as an example a triangle (3 nodes) in 2D (2 dimensions), an element stiffness matrix would have size 6 by 6 so it could relate the stiffness between any pair of nodes in any dimensions.

The most common finite element implementations combine all element stiffness matrices into one global stiffness matrix (see APPENDIX B

for an explanation of the assembly process). This matrix is therefore very large and sparse since it defines all nodes' stiffness relations.

3.4 Loads and Constraints

To create the conditions in which the objects are being studied loads (also known as boundary conditions) are defined. Loads are applied forces and the most typical loads are:

1. Prescribed forces – These are forces applied directly to nodes (vertices of the elements).
2. Distributed loads – Define the applied force per unit of area. In a finite element point of view these loads are applied to element faces.
3. Body forces - Define the applied force per unit of volume. These are applied to the element, e.g. gravitational loading.

Constraints restrict the motion of nodes by defining their displacements and therefore influence the movement of the whole object. When applied to some or all degrees of freedom of a node, the displacement of this node in the constrained directions will be zero. To illustrate both concepts consider Figure 3.2. Nodes N1 and N6 are constrained in all directions so that they will not move and node N10 has a prescribed force.

3.5 Solver

The solver is an implementation of a mathematic procedure to find the solution of systems of equations and it is a core component of a structural analysis simulator. Consider a problem where N is the number of nodes times the number of degrees of freedom. The solver is used to find the displacements (u) in (3.13) where K is a N by N matrix and u and f are both N size vectors.

There are various methods to solve linear systems of equations and these can be classified as iterative or direct. As seen in [SVDV00], iterative methods are easier to implement and the memory and arithmetic cost does not grow much with the problem size as it does for direct methods. These advantages make iterative methods the typical solvers used in structural analysis.

A technique called preconditioning can be used to improve the condition number of a matrix, hence increasing the convergence and reliability of the solution ([KK09]). Applying preconditioning to a system $Ax = b$ implies multiplying both sides of the system by a symmetric, positive-definite matrix that approximates A , but is easier to invert.

4 Methodology

In order to find what gets affected by the dynamic topology and how to implement an efficient revalidation strategy, all aspects of the simulation process were studied. A closer look at the global stiffness matrix reveals that every value relates to the elasticity between a pair of nodes. Based on this characteristic, the proposed revalidation methodology updates only the cells of the matrix that were affected by the topological change leaving the rest of the matrix intact. However, adding, removing or editing arbitrary cells can prove to be an expensive task depending on how the matrix is stored in memory. Therefore, it is required a storing structure for the global stiffness matrix that enables inexpensive operations regarding its entries. This thesis proposes a matrix representation that provides such features while also taking into account the properties of a global stiffness matrix (sparsity and symmetry) (see Section 4.2).

Having a new matrix representation influences the work of the solution techniques of the system of equations. Solvers perform a series of operations with the matrix in order to find the solution of the system, being the most expensive of all the multiplication with a vector. In this thesis, a multiplication algorithm using the new matrix representation is proposed that proves to be faster than the ones with other matrix representations (see Section 4.5). This representation also provides the flexibility of allowing the matrix to be traversed in any arbitrary order. Such ability can be used by solvers to increase convergence or in any other method (even from other fields) that requires fast traversal of a matrix in a specific order.

This methodology created to enable the simulation of dynamic meshes results in a different procedure (see Figure 4.1) compared to the one used for the simulation of constant meshes. At initialization, the global stiffness matrix is computed based on the rest state of the simulated model. Then, for every occurred topological change, the stiffness values of the affected elements are recomputed and updated on the global stiffness matrix.

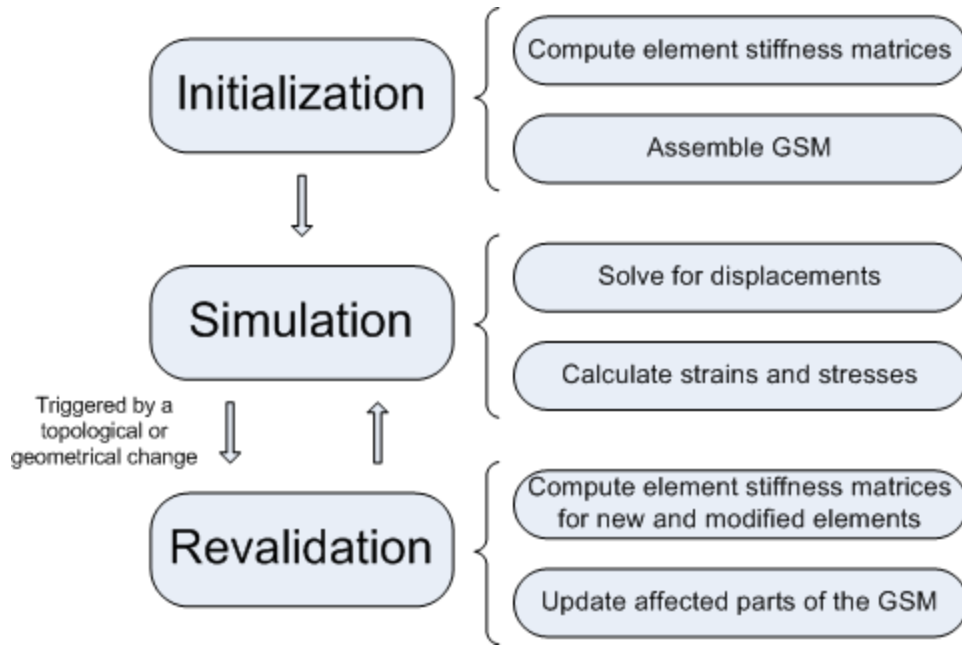
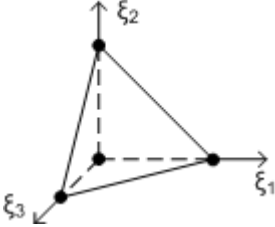
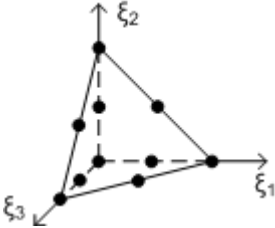


Figure 4.1 - Proposed procedure for the simulation of dynamic meshes. (GSM stands for Global Stiffness Matrix)

The specific simulation type studied in this thesis is Finite Element Method for static linear elasticity. Table 4.1 shows which characteristics influence the linearity of the simulation and one example for each option.

Table 4.1 - Characteristics that define the overall linearity of the simulation. (Only one example is shown per option)

Characteristic/Property	Linearity	Examples
Stress strain relationship (Material linearity)	Linear	Hooke's law of elasticity $\sigma = E\varepsilon$
	Non-linear	Hyperelasticity $\sigma = \rho F \frac{\partial \varphi}{\partial E} F^T$
Measure of strain (Geometric linearity)	Linear	Cauchy's strain tensor $\varepsilon_C = \frac{1}{2}(\nabla u + [\nabla u]^T)$
	Non-linear	Green's strain tensor $\varepsilon_G = \frac{1}{2}(\nabla u + [\nabla u]^T + [\nabla u]^T \nabla u)$
Basis Function	Linear	4 points tetrahedron

		
	Quadratic	10 points tetrahedron 

4.1 Computing the element stiffness matrices

The behavior of the simulated object is defined by its material properties and by its mesh. By combining these two the stiffness matrix can be obtained. Bower shows in [Bow09] a method to implement the computation of the element stiffness matrices through the generalized Finite Element Method.

The construction of an element stiffness matrix can be done linearly thanks to the three restrictions imposed by the studied problem:

1. Material linearity: Its linear due to the use of Hooke's law;
2. Geometric linearity: Cauchy's strain tensor is linear;
3. Basis Function: The used meshes are composed of 4 point tetrahedra.

One important concept that influences the computation of the element's stiffness is the order in which the nodes of an element are defined. The two possibilities are the right-hand rule and the left-hand rule. The proposed implementation works with the right-hand rule, but a workaround was created to enable the use of both: when loading a mesh from a file, a volume check is performed on the first element to determine which is the used hand rule and if a conversion should be done (see APPENDIX B

).

The implemented algorithm for the element stiffness matrices calculation can be seen in Algorithm 4.1.

-
1. **foreach** element **do**
 2. Set integration points and weights
 3. Compute shape functions derivatives wrt local coordinates
 4. Compute Jacobian matrix
 5. Compute Jacobian determinant
 6. Convert shape function derivatives to derivatives wrt global coordinates
 7. **foreach** (a,i,b,k) **do**
 8. Compute Equation (3.27)
 9. Add result to element stiffness matrix
 10. **endfor**
 11. **endfor**
-

Algorithm 4.1 – Element Stiffness Matrices calculation.

4.2 Equivalent matrix representation

The proposed algorithm aims at effectively building, updating and solving linear systems, by means of reducing the processing time and of minimizing the memory consumption. A linear system represented in the matrix form is:

$$A x = b \tag{4.1}$$

where A is the matrix of coefficients, x is the vector of unknowns and b is the vector of solutions. The study of the related work provided an understating of the objective of this system in getting the solution of the problem. The matrix of coefficients aims at collecting the information for the equations regarding the connectivity of the vertices (edge topology) and of the unknowns (vertices without boundary conditions) themselves. The process for building the matrix of coefficients is normally based on traversing the given mesh (element by element), in order to identify the neighboring elements, which need to contribute to an edge (non diagonal positions) or to a vertex (diagonal positions). This process is expensive and when the geometry and the topology of the mesh are changed, the matrix of coefficients also needs to be changed.

Hence, it became clear that if it was possible to have the information concerning the elements around an edge, it would be easy to collect and build the information of the non diagonal positions of the linear system (see Figure 4.2 and Table 4.2). Moreover, if the computed information for every edge (elements around the edge) is stored within a small edge matrix, it will provide enough flexibility to modify or recompute only the incident edges to a vertex, when the vertex changes, without affecting the rest of the linear system. Analogically, the same strategy can be applied for storing the computed information of the diagonal positions of the linear system (see Figure 4.2 and Table 4.3), computing a small diagonal matrix for every unknown. These two sets of matrices are an equivalent representation of the matrix of coefficients, which will be henceforward referred to as the equivalent matrix.

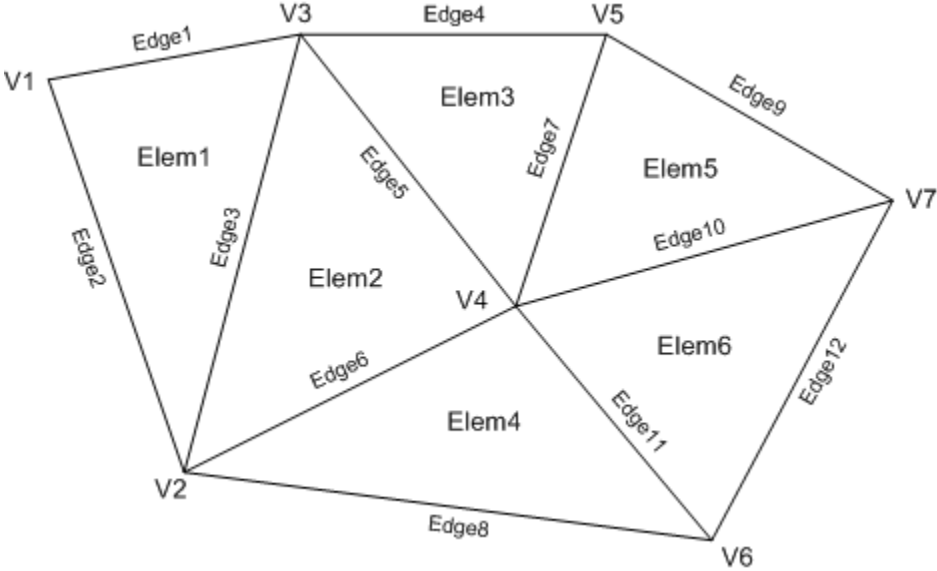


Figure 4.2 - Mesh composed of six elements.

Table 4.2 - Elements needed to calculate the stiffness of each edge.

Edge	Elements needed
Edge1	Elem1
Edge2	Elem1
Edge3	Elem1, Elem2
Edge4	Elem3
Edge5	Elem2, Elem3
Edge6	Elem2, Elem4
Edge7	Elem3, Elem5
Edge8	Elem4
Edge9	Elem5
Edge10	Elem5, Elem6
Edge11	Elem4, Elem6
Edge12	Elem6

Table 4.3 - Elements needed to calculate the stiffness of each vertex.

Vertex	Elements needed
V1	Elem1
V2	Elem1, Elem2, Elem4
V3	Elem1, Elem2, Elem3
V4	Elem2, Elem3, Elem4, Elem5, Elem6
V5	Elem3, Elem5
V6	Elem4, Elem6
V7	Elem5, Elem6

Structure wise, the matrix of coefficients is no longer stored in a traditional sparse matrix ([SGV05]). Instead, it is divided into two vectors (see Figure 4.3 for a visual representation). In one vector is stored the edge matrices and in the other is stored the diagonal matrices. Every element of both vectors is a $n \times n$ matrix where n is the dimension (1D, 2D or 3D) of the problem.

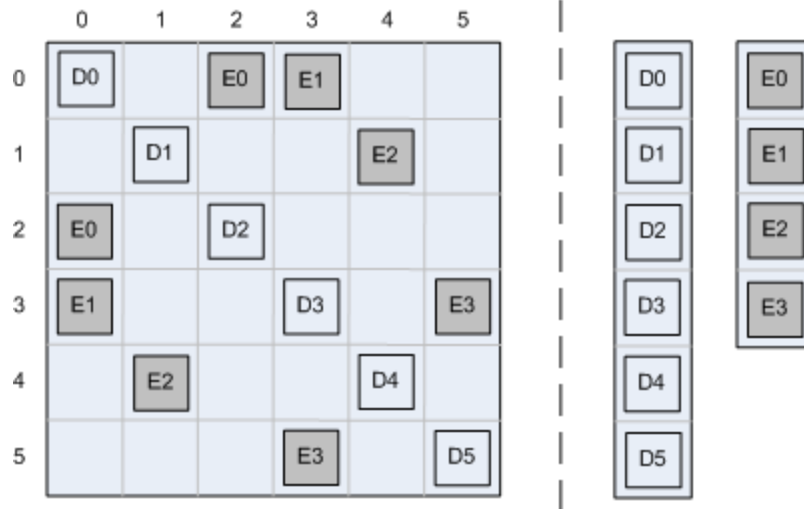


Figure 4.3 - Graphical representation of a matrix of coefficients (left). The same matrix of coefficients represented as the equivalent matrix (right).

The two main characteristics of a matrix of coefficients, sparsity and symmetry, are used by the equivalent matrix to minimize memory consumption: only the nonzero values are stored and only one instance of every edge is stored for every pair of connected vertices. In order to explain how the proposed algorithm uses these characteristics for building the equivalent matrix and for the sake of clarity, the process is subdivided into three steps:

1. Constructing the needed neighboring information;
2. Computing the set of edge matrices;
3. Computing the set of diagonal matrices.

These three simple steps enable the minimization of the space in memory and the reduction of the processing time. In addition, the new representation of the matrix of coefficients allows the flexible handling and modification of individual vertices or edges, without affecting the rest of the matrix. The example mesh shown in Figure 4.4 will be used in association with Figure 4.5 and Figure 4.6 to help follow the explanations given on how to compute the edge matrices and diagonal matrices respectively.

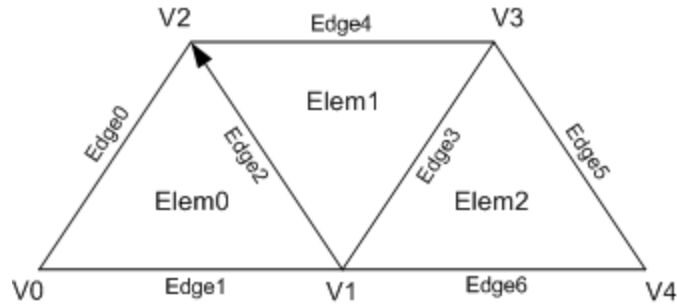


Figure 4.4 - A 2D mesh composed of three triangular elements

4.3 Build of the equivalent matrix

The equivalent matrix replaces the matrix of coefficients by a set of small matrices, which can be computed faster and requires less space in memory. In order to avoid traversing the whole mesh, when computing the equivalent matrix, the neighboring information is pre-computed. The element matrices are also pre-computed and they are the basis for computing the edge and diagonal matrices. Normally, the element matrices are not computed, since the contribution of the elements is directly considered during the solution of the system. However, this thesis aim at effectively handling geometrical or topological changes, therefore it is more efficient to use the element matrices for updating the edge matrices or the diagonal matrices according to the changes, than re-computing the needed element matrices every time.

4.3.1 Constructing the neighboring information

In order to build the equivalent matrix efficiently these three kinds of neighboring information need to be pre-computed: i) the elements around an edge, ii) the elements around an unknown and iii) the vertices of an edge. This information is computed during the initialization process and it is revalidated, if some changes to the topology of the mesh are made. The neighboring information allows the independent computation of the non diagonal and the diagonal positions of the matrix of coefficients and it also provides the information regarding the relationship between vertices. The proposed methodology uses a mesh data structure that automatically constructs the neighboring information, however this process will be explained for the sake of completeness.

During the loading process of the mesh, three double arrays (db) are initialized, where the needed information for the edges (dbEdg) is stored, for the unknowns (dbUkn) and for the vertices of the edges (dbVtsEdg). For every read element, its index is appended to its six corresponding edges in dbEdg and to its four corresponding vertices (unknowns) in dbUkn, and the corresponding pair of vertices is added to every one of the six edges within dbVtsEdg. By the end of the loading process, the neighboring information is also ready.

4.3.2 Computing the edge matrices

Given the neighboring information of the elements around an edge (dbEdg) and the element matrices, the non diagonal positions can easily be computed, by means of traversing the elements around the edge and adding the contribution of the corresponding element. On Figure 4.5 it is shown the computations involved in the build of Edge2. Since both elements Elem0 and Elem1 share Edge2, the components of both elements regarding this edge (colored in grey) are added to the Edge2's matrix. Note that each element has two contributions to every used edge and since one is the transposed of the other, only one is added to the edge matrix. The used contribution is chosen based on the direction of the edge.

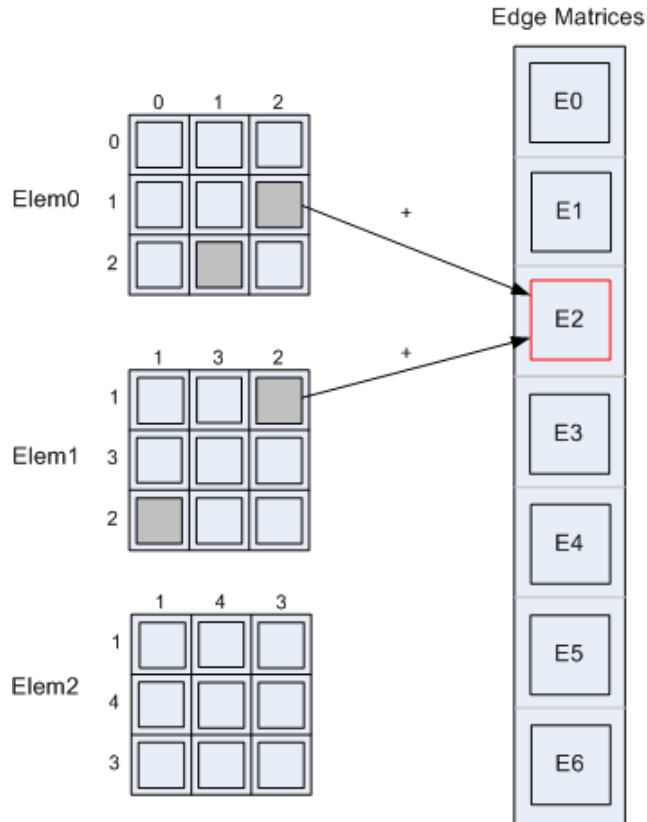


Figure 4.5 - Element contributions to the build of the third edge matrix (E2).

4.3.3 Computing the diagonal matrices

The diagonal positions of the linear system are computed in a similar way to the edge positions. In this case, the neighboring information regarding the elements around an unknown (dbUkn) is used and the contribution of every involved element is considered to the diagonal. Consider Figure 4.6, where the build of the diagonal for vertex V1 is being performed. From Figure 4.4 it is clear that vertex V1 is shared by the three elements. Therefore, D1 is calculated by adding up the three contributions to V1 of the three elements.

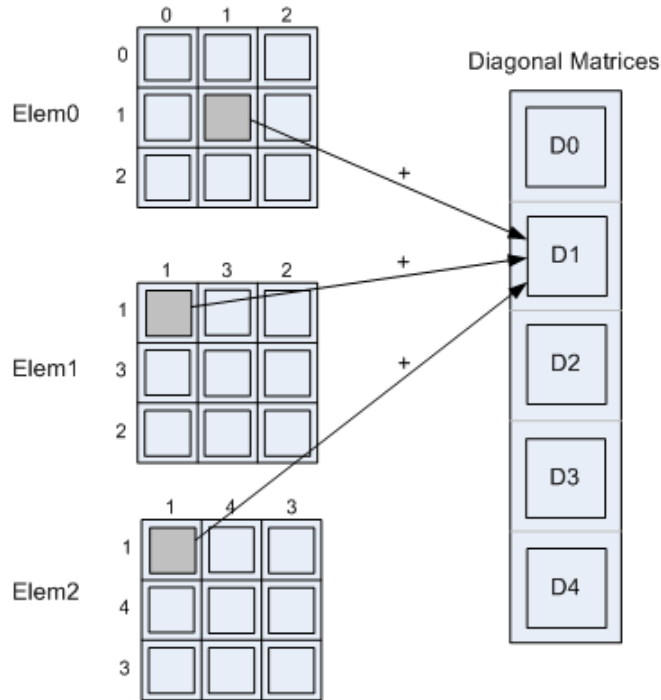


Figure 4.6 - Element contributions to the build of the second diagonal matrix (D1).

The union of the edge matrices and the diagonal matrices is equivalent to a matrix of coefficients. Although, these two sets of matrices need to be considered in order to solve the system, the order in which the equations will be solved can be arbitrary decided. This flexibility could be advantageous for a rapid convergence, since it is equivalent to having a linear system with an optimized ordering of the unknowns, leading to an improvement of the performance of the solver ([OLHB02, Bar96, BW98]). Moreover, since the matrices within the two sets are independent, they can be changed or revalidated without a major effort, because that would only require the recomputation of a small set of matrices, avoiding the computation of the whole system of equations.

4.4 Revalidation of the equivalent system

Every change that is done to the topology of the mesh implies a revalidation of the matrix of coefficients so that the latter continues to represent the changed mesh. For the sake of functionality, the implemented revalidation method allows the processing of more than one

change to the mesh per call. The proposed method's procedure is based on adding or removing the contribution of the elements that were affected by the change. The arrays of edge matrices and diagonal matrices effectively support the addition and removal of vertices or elements. The arrays are implemented with a buffer according to the size of the mesh, enabling addition of new vertices or elements. In case of removal, the corresponding matrices are flagged as “removed”, but there are no memory reallocations, in order to avoid this expensive task. The double arrays for the neighboring information have the same capabilities. Algorithm 4.2 shows the steps taken to perform the revalidation:

1. **foreach** removed element **do**
2. remove element contribution
3. **endfor**
4. reserve space for new elements
5. **foreach** added element **do**
6. compute element contribution
7. add element contribution
8. **endfor**
9. **foreach** moved node **do**
10. **foreach** element shared by this node **do**
11. add element to recomputeVector
12. **endfor**
13. **endfor**
14. **foreach** element in recomputeVector **do**
15. remove element contribution
16. recompute element contribution
17. add element contribution
18. **endfor**

Algorithm 4.2 - Method to revalidate the equivalent matrix.

The algorithm can be subdivided into three phases, representing the three possible kinds of changes, which can be applied to the topology and geometry of the mesh:

1. Remove an element (line 1 to 3) - The contribution of each removed element is subtracted from the corresponding entries in the equivalent matrix, according to the neighboring information (no memory reallocation is performed);
2. Add an element (line 4 to 8) - Additional space is acquired (the additional space is obtained from the available buffer) to incorporate the new elements. The contribution

of each added element is computed and added to the equivalent matrix (to the corresponding entries according to the neighboring information);

3. Move a vertex (line 9 to 18) - To avoid re-computing the contribution of an affected element more than once, an initial loop is done to find out which elements are affected by the movement of the vertices. For each affected element, its contribution is removed from the equivalent matrix. Then, its contribution is recomputed using the new vertices positions and finally it is added back to the equivalent matrix.

Note that adding or removing an element contribution to the equivalent matrix is done by iterating over the edges and diagonals that are shared by that element. If a vertex is removed, no special operation needs to be considered, since it implies the elimination (flagged as “removed”) of the corresponding diagonal matrix and the edges matrices incident to that vertex. The neighboring information is also revalidated, therefore the following operations will not involve the “removed” vertices.

4.5 Solution of the equivalent system

The implemented algorithm to solve the linear system of equations is the Conjugate Gradient with Jacobi preconditioning as it was proposed in [She94]. The only two noticeable changes done so far are: the way the multiplication of the equivalent matrix with a vector is done; and the addition of a filter system to apply the constraints ([BW98]). Although, the Preconditioned Conjugate Gradient method was chosen for solving the linear system, the presented methodology is completely independent of the kind of solver. It is suitable for other iterative methods as well as for direct methods.

4.5.1 Multiplication of the equivalent matrix with a vector

Having the matrix of coefficients stored in the equivalent matrix form requires a special method for its multiplication with a vector.

```

1. foreach rst in resultVector do
2.   resultVector[rst]  $\leftarrow$  0
3. endfor
4. foreach edge in edgeVector do
5.   edgeStartVertex  $\leftarrow$  start vertex of this edge
6.   edgeEndVertex  $\leftarrow$  end vertex of this edge
7.   resultVector  $\leftarrow$  resultVector + (edgeVector[edge] * multiVector)
8.   resultVector  $\leftarrow$  resultVector + (edgeVector[edge]T * multiVector)
9. endfor
10. foreach diag in diagVector do
11.  resultVector  $\leftarrow$  resultVector + (diagVector[diag] * multiVector)
12. endfor

```

Algorithm 4.3 – Multiplication of the equivalent matrix with a vector (multiVector) and its result (resultVector).

Algorithm 4.3 shows the main steps that are performed to multiply the equivalent matrix with a vector. This method starts by setting the resultVector to zero so that the results of the multiplications performed over the matrix can be added to it. For every edge, the vertices that form that edge (edgeStartVertex and edgeEndVertex) are found by consulting the neighborhood information (dbVtsEdg). This is done to know which is this edge's position in the matrix. Doing so it is known with which position of multiVector this edge should be multiplied and in what position of resultVector it should be stored.

Also notice that for every edge two multiplications are done. This is due to the symmetric characteristic of the matrix. To provide a better understanding of this procedure consider the example shown in Figure 4.7. Assuming that the algorithm is iterating on edge E1 and that this edge has vertex 0 as a starting vertex and vertex 3 as an ending vertex: In line 7 of the algorithm E1 would be multiplied by V3 and its result would be stored in R0 (red boxes). And in line 8, since E1 also connects vertex 3 to vertex 0 with the same value, in R3 would be stored the multiplication of the transposed E1 with V0 (green boxes).

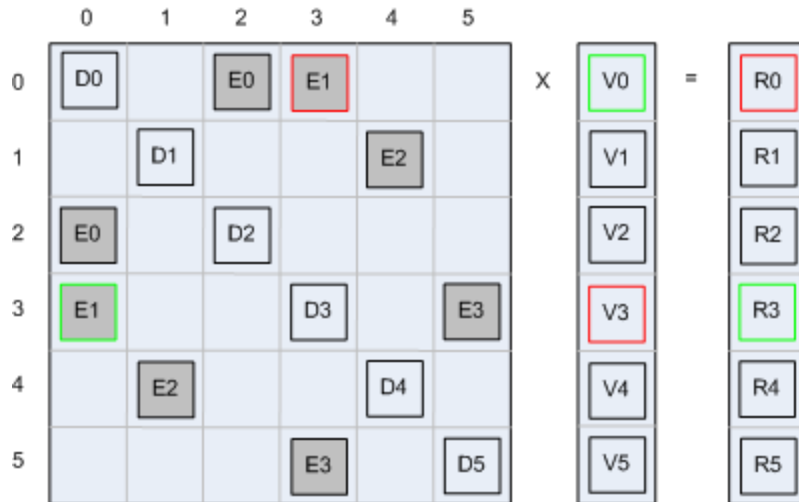


Figure 4.7 - Multiplication of the equivalent matrix (represented as a normal matrix for simplification) with a vector. The green and red lined colored boxes indicate the used values when the multiplication iterates on edge E1.

After processing all the edges, the algorithm iterates on the diagonals (line 10 to 12). The process is similar but simpler for each diagonal relates a vertex to itself.

5 Results

Both the equivalent matrix (DEM²) and its multiplication with a vector have been implemented as proposed in the previous chapter. This implementation is not complex and it can easily be reproduced following the given indications. The implementation is single threading and a desktop PC Intel Core 2 Quad Q6600 with 3.25 GB RAM was used to make the measurements. In order to compare the performance of the algorithms, the classical form to represent the matrix of coefficients (the sparse matrix) has also been implemented. Four implementations of the sparse matrix were programmed:

1. Linked lists without the symmetric characteristic (CSM),
2. Linked lists with the symmetric characteristic (SSM),
3. Compressed row storage (CRS) and
4. Block compressed row storage (BCRS).

The first two implementations (CSM and SSM) were only made for the sake of completeness, hence a simple array of linked lists was employed. The last two implementations (CRS and BCRS) were done without considering the symmetric characteristic, since the aim was to have faster access during the multiplication of the matrix with a vector (SPMV). The implementation of the multiplication with a vector was made for the four representations as well. The build process of CSM, SSM, CRS, and BCRS uses the neighboring information, in other words, the needed information is stored in the representation without traversing the whole mesh. For the multiplication process, the neighboring information was not used for CSM, SSM, CRS and BCRS, since each one has fields for identifying the corresponding position in the mesh.

In order to measure the performance of the five implementations, two different tetrahedral meshes were used: i) a gargoyle with almost 50,000 elements and ii) a hand with almost 100,000 elements. Table 5.1 presents the relevant topological information of the meshes

² DEM stands for Diagonal-Edge Matrix.

(shown in Figure 5.1, Figure 5.2, Figure 5.3 and Figure 5.4). In order to present the capabilities of the revalidation process, three operations representing the basic changes in the mesh (remove an element, add an element and move a vertex) were used: i) decimate, ii) mirror and iii) scale. The decimate operation takes a mesh and removes 50% of the elements of the model. The mirror operation doubles the number of elements of the mesh. The scale operation moves every vertex of the mesh. For the test of these operations, the dragon, the gargoyle and the bunny were used (Figure 5.5, Figure 5.6 and Figure 5.7 show the three aforementioned operations, one on each model).

Table 5.1 - Topological information of the meshes used for the measurements.

Mesh	Vertices	Edges	Elements
Bunny	2,087	12,796	9,997
Gargoyle	13,044	71,873	49,996
Hand	26,649	144,669	99,995
Dragon	26,436	144,285	100,000

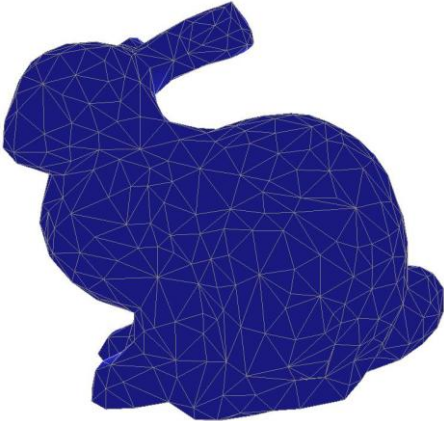


Figure 5.1 - Bunny model.

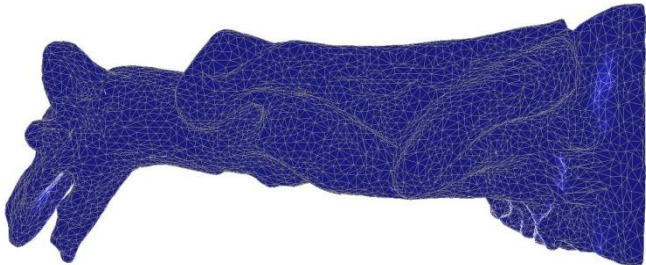


Figure 5.2 - Gargoyle model.

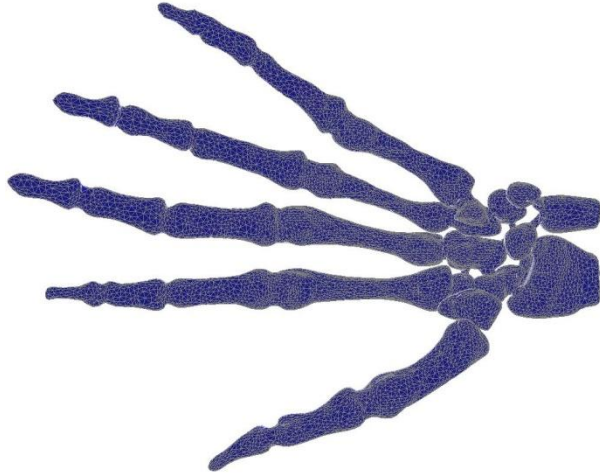


Figure 5.3 - Hand model.

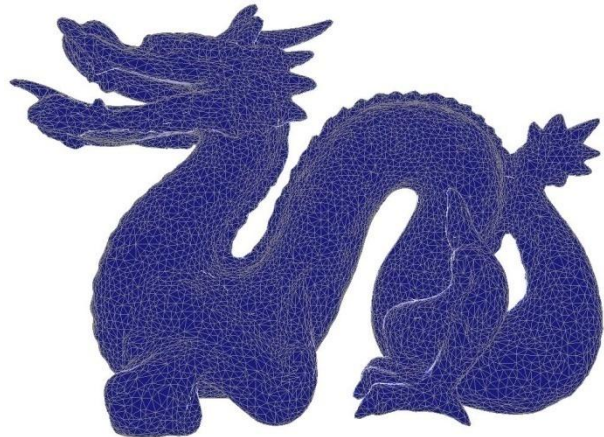


Figure 5.4 - Dragon model.

5.1 Build and Multiplication of the matrix of coefficients

Two kinds of measurements were performed, one for the build process and one for the multiplication with a vector process. For the times regarding the multiplication with a vector 20 multiplications were considered, in order to be able to measure the time, since the measurement for a single multiplication is not very accurate. Table 5.2 shows the results in milliseconds for the mesh models and the two processes. These measurements were made for the five representations (CSM, SSM, CRS, BCRS and DEM) of the linear system.

Table 5.2 - Measurements for the build and multiplication processes (in milliseconds).

Process	Build		Multiplication	
	Gargoyle	Hand	Gargoyle	Hand
CSM	203	456	213	598
SSM	115	256	140	347
CRS	459	901	313	691
BCRS	203	420	83	225
DEM	94	215	78	179

These results show that the proposed algorithm is faster than the other four implementations. The algorithm is, for the build process, between 49% and 80% faster than the CSM, CRS and BCRC implementations, since these implementations require almost two times the space in memory than the SSM and ours. However, for the multiplication process, the CRS and BCRC implementations will have an advantage, because of the direct access, but this is not the case for the CSM, where the access is more expensive (because of the linked lists). Although the build process for the SSM implementation is similar to the CSM, the latter is still 18% faster.

In the multiplication process our algorithm performs between 44% and 75% faster than the CSM, SSM and CRS implementations. The reason for these results is the expensive access of the linked lists (CSM and SSM). The CRS implementation is slower than the BCRC implementation, because it also considers the nonzero entries of the $n \times n$ element matrices (where n is the dimension of the problem). On the other hand, the BCRC implementation considers the $n \times n$ element matrices as a block, improving the performance during the multiplication process. DEM is in this case only 6% faster for the Gargoyle and 20% faster for the Hand. These results also show that the BCRC algorithm does not present a proportional behavior to the number of elements, but DEM does.

Although the BCRC algorithm could be an interesting alternative for the multiplication process, it will be useless for topological changes, since it will require the addition and removal of block entries and therefore memory reallocation in the arrays (the memory is continuous), a special characteristic that DEM can handle very well. In terms of memory consumption, the CSM, CRS and BCRC implementations require more memory than the SSM and the DEM implementations, since these do not profit from the symmetric characteristic of the matrix of coefficients. Only the upper or the lower part of the matrix of coefficients is stored in the SSM and the DEM implementations, hence they have similar memory consumption.

5.2 Solving with Preconditioned Conjugate Gradient (RealTime)

The primary aim of the proposed implementation is to effectively support geometrical and topological modification of mesh-based techniques, requiring the utilization of linear systems. Nevertheless, the limits of this implementation in terms of real time performance were also explored. Table 5.3 presents the measurements for 20 iterations (not only the multiplication) of the conjugate gradient method with a Jacobi preconditioner for three meshes with different sizes.

Table 5.3 - Meshes with real time performance (time in milliseconds).

Mesh	Vertices	Elements	Time	FPS
Bunny_10	2,087	9,997	16	63
Gargoyle_30	7,944	29,998	47	21
Gargoyle_50	13,044	49,996	99	10

These results reveal that this algorithm can perform in real time with meshes up to 30,000 elements and at interactive rates with meshes up to 50,000 elements.

5.3 Topological changes and the corresponding revalidations

As mentioned above, the developed algorithms aim at effectively handling geometrical and topological changes. Because of that, the pre-computation of the element matrices is stored, in order to easily revalidate the equivalent system, whenever a change in the topology (remove or add vertices and elements) or in the geometry (move vertices) happens. The three implemented operations (decimate, mirror and scale) are extreme examples, since they employ at least the 50% of the elements or the vertices and this is not a common activity in computer graphics, where fast performance is expected.

The decimate operation removes half of the elements of the model, hence only the elements, which are on the boundary of the removal are recomputed (see Table 5.4). Since the elements

on the boundary of the removal are much less than half of the model, the revalidation process is much faster than the reconstruction.

Table 5.4 - Measurements for the decimate operation (in milliseconds).

Mesh	Initialization	Revalidation	Reconstruction
Bunny	140	<1	67
Gargoyle	702	71	359
Dragon	1,427	47	719

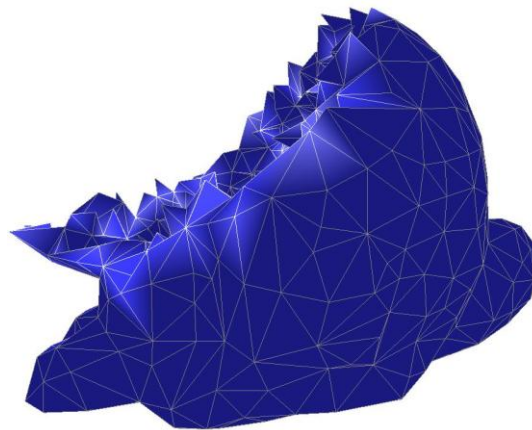


Figure 5.5 - Decimated bunny.

The revalidation process for the mirror operation takes approximately 50% of the time of the reconstruction (see Table 5.5), because it only processes the mirrored elements in comparison with the reconstruction, which processes two times the number of elements (the original and the mirrored ones).

Table 5.5 - Measurements for the mirror operation (in milliseconds).

Mesh	Initialization	Revalidation	Reconstruction
Bunny	140	125	276
Gargoyle	702	783	1,416
Dragon	1,427	1,380	2,857

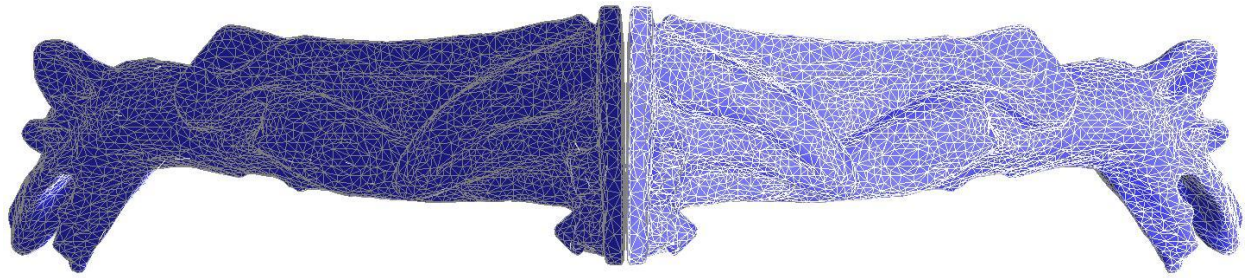


Figure 5.6 - Mirrored gargoyle.

The revalidation process for the scale operation takes slightly more time than the reconstruction (see Table 5.6), however moving the complete set of vertices in a single step is the worst case scenario and thus any other operation involving moving vertices will perform much faster than the reconstruction.

Table 5.6 - Measurements for the scale operation (in milliseconds).

Mesh	Initialization	Revalidation	Reconstruction
Bunny	140	140	141
Gargoyle	702	736	703
Dragon	1,427	1,483	1,427

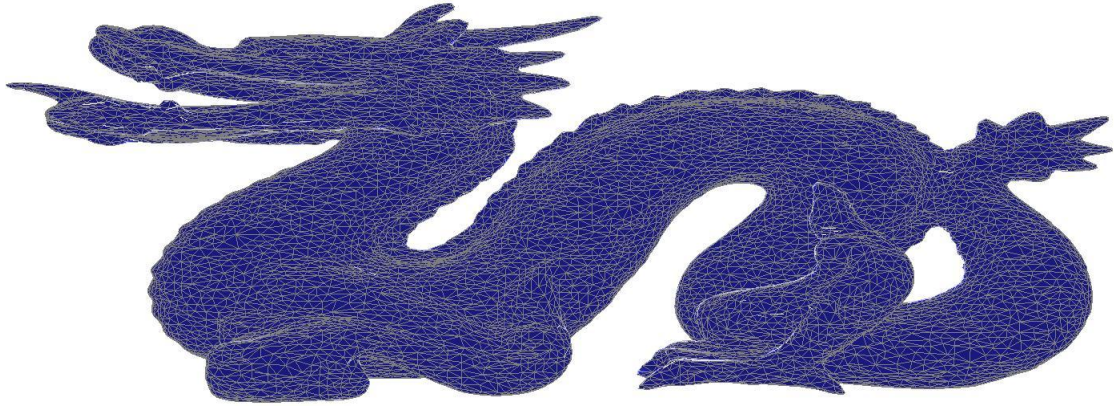


Figure 5.7 - Scaled dragon.

The presented three operations show the performance of the revalidation algorithm for geometrical and topological changes. These tests demonstrate that this algorithm is in normal conditions (no extreme examples) much faster than a complete reconstruction of the linear system. This is an improvement towards mesh-based applications such as simulation, shape deformation, virtual surgery, and fluid/smoke animation, among others, where geometrical (and sometimes topological) changes affect the linear system, which needs to be solved. These kinds of applications will not only benefit from faster revalidations, but also from faster solutions, as demonstrated with the multiplication process.

Hence, this methodology proves to be a step forward in dynamic mesh simulation as it enables interactive rates for dynamic meshes with up to 50,000 elements in a single thread. The previous results also reveal that the proposed algorithms behave proportionally to the size of the meshes. In other words, the complexity of the algorithms is $O(n)$.

6 Conclusion

The objective of this thesis involved the development of a methodology that would surpass the previous approaches to the simulation of dynamic meshes. This methodology should have a faster simulation cycle, which consequently would allow real-time performance for meshes with higher complexity.

In order to achieve this goal, a revalidation strategy was created to avoid the systematic rebuild of the matrix of coefficients. This involved the development of a special storing structure for the matrix that allowed fast operations regarding its entries with little memory manipulation. Also, the build and revalidation of the matrix were aided by the use of the precomputed neighboring information. This proved to be a crucial improvement as it avoided the expensive task of querying the mesh for its nodes' connectivity.

The previous chapter confirms that the methodology developed within this thesis work represents a step forward towards the simulation of dynamic meshes. The results show that the revalidation strategy is faster than the rebuild of linear system of equations. Hence, higher frame rates can be achieved and bigger meshes can be simulated in real-time.

An implementation of this methodology will be used in a product development application to simultaneously design and analyze mechanical structures.

6.1 Future Work

However, there is still room for improvement, as the proposed methodology opens doors to other possible optimizations. Although the implemented solver on this case was the preconditioned conjugate gradient, the equivalent matrix is solver independent. Plus, the matrix can be traversed in any given order, which means that solvers that can take advantage of this ordering flexibility will perform better with this matrix. Strategies like Cuthill-McKee

(RCM), self-avoiding walk (SAW), out-in ordering or multi layer solving can be used to re-order the equations of the matrix to meet the solver requirements.

Another possible improvement could be done through the parallelization of the presented methodology. The results shown in the previous chapter were achieved with a single threaded implementation, but a parallelization in either the CPU or GPU would certainly yield smaller simulation times.

7 Bibliography

- [Ale06] Marc Alexa, editor. *Interactive shape editing*. ACM SIGGRAPH 2006 Courses, 2006.
- [ARS06] CE Augarde, A. Ramage, and J. Staudacher. An element-based displacement preconditioner for linear elasticity problems. *Computers and structures*, 84(31-32):2306–2315, 2006.
- [Bar96] David Baraff. Linear-time dynamics using lagrange multipliers. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 137–146, New York, NY, USA, 1996. ACM.
- [BFMF06] Robert Bridson, Ronald Fedkiw, and Matthias Muller-Fischer. Fluid simulation: Siggraph 2006 course notes. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 1–87, New York, NY, USA, 2006. ACM.
- [BN96] Morten Bro-Nielsen. Surgery simulation using fast finite elements. In *VBC '96: Proceedings of the 4th International Conference on Visualization in Biomedical Computing*, pages 529–534, London, UK, 1996. Springer-Verlag.
- [BNC96] Morten Bro-Nielsen and Stephane Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. In *Computer Graphics Forum*, pages 57–66, 1996.
- [Bow09] A.F. Bower. *Applied Mechanics of Solids*. Taylor and Francis, 1st edition, August 2009.
- [BW98] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM New York, NY, USA, 1998.

- [GBT07] M. Gissler, M. Becker, and M. Teschner. Constraint sets for topology-changing finite element models. In *Virtual Reality Interactions and Physical Simulations VRIPHYS*, pages 21–26, November 9th 2007.
- [Hug00] T.J.R. Hughes. *The finite element method: linear static and dynamic finite element analysis*. Dover Publications, 2000. ISBN: 0486411818.
- [KFCO06] B. M. Klingner, B. E. Feldman, N. Chentanez, and J. F. O’Brien. Fluid Animation with Dynamic Meshes. In *International Conference on Computer Graphics and Interactive Techniques*. ACM New York, NY, USA, 2006.
- [KK09] Autar Kaw and Egwu Kalu. *Numerical Methods with Applications: Abridged*. <http://www.autarkaw.com>, last visited on November 29th, 2009.
- [Lan03] H.P. Langtangen. *Computational partial differential equations: numerical methods and diffpack programming*. Springer Berlin, 2nd edition, 2003. ISBN: 3540408878.
- [MSJT08] M. Müller, J. Stam, D. James, and N. Thürey. Real time physics: class notes. In *International Conference on Computer Graphics and Interactive Techniques*, pages 1–90, New York, NY, USA, 2008. ACM.
- [MTPS08] J. Mezger, B. Thomaszewski, S. Pabst, and W. Strasser. Interactive Physically-Based Shape Editing. In *ACM Solid and Physical Modeling Symposium (SPM)*, 2008.
- [NMK⁺06] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlon. Physically Based Deformable Models in Computer Graphics. *Computer Graphics Forum*, 25(4):809–836, 2006.
- [OLHB02] L. Oliker, X. Li, P. Husbands, and R. Biswas. Effects of ordering strategies and programming paradigms on sparse matrix computations. *SIAM Review*, 44(3):373–393, 2002.

- [SGV05] F.S. Smailbegovic, G. N. Gaydadjiev, and S. Vassiliadis. Sparse matrix storage format. In *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc 2005*, pages 445–448, November 2005.
- [She94] Jonathan R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. *Computer Science Tech. Report*, pages 94–125, 1994.
- [SSSM09a] Sebastian Serna, João Silva, Andre Stork, and Adérito Marcos. Efficient algorithm for building and solving linear systems. In *17th Portuguese Conference on Computer Graphics*, pages 143–148, October 2009.
- [SSSM09b] Sebastian Serna, João Silva, Andre Stork, and Adérito Marcos. Neighboring-based linear system for dynamic meshes. In *Workshop on Virtual Reality Interaction and Physical Simulation*, November 2009.
- [SVDV00] Y. Saad and H.A. Van Der Vorst. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1-2):1–33, 2000.
- [vFST06] W. von Funck, H. Theisel, and H. P. Seidel. Vector Field Based Shape Deformations. In *International Conference on Computer Graphics and Interactive Techniques*. ACM New York, NY, USA, 2006.

APPENDIX A

Assembly of the global stiffness matrix

Consider the cantilever beam shown in Figure 3.2. This beam is discretized into 8 elements. In this 2D triangular element example, each stiffness matrix is a 6 by 6 matrix. After computing the element stiffness, these can be assembled into a global stiffness matrix. Each cell of an element matrix is added to the global stiffness matrix at the cell where the pair of nodes and spatial coordinates match.

In this particular case, the global stiffness matrix is a band matrix. A different ordering of the nodes in the object discretization will result in a different distribution of the non-zero values on the matrix.

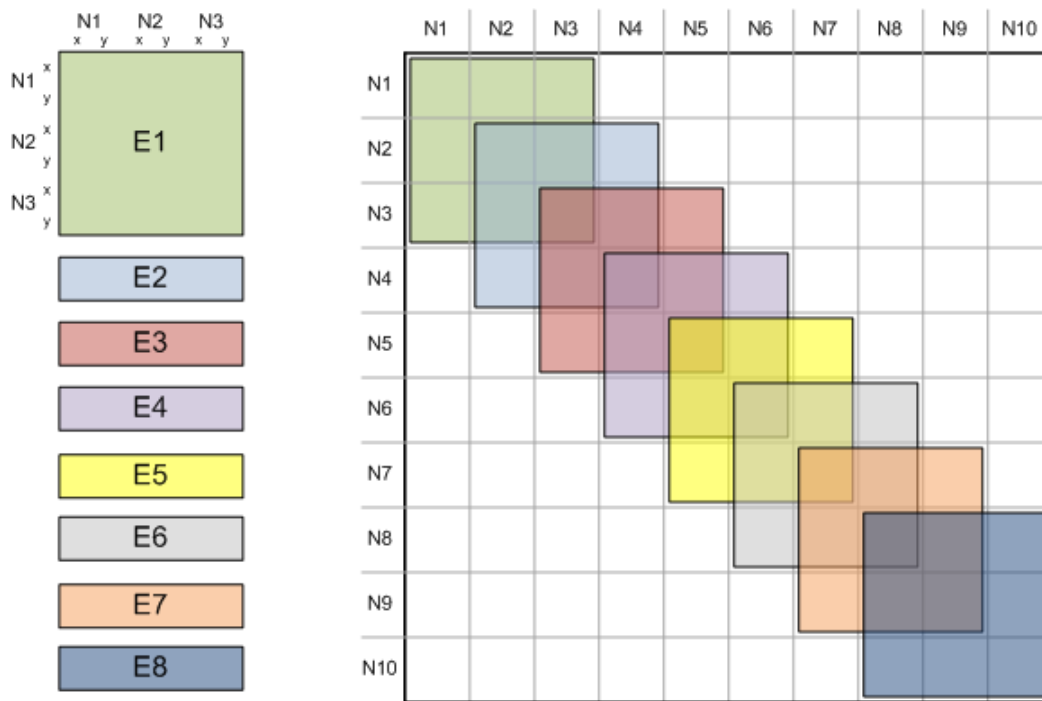


Figure A.1 - Element stiffness matrices (on the left side) and the same matrices assembled into a global stiffness matrix (on the right side).

APPENDIX B

Workaround to load both right and left hand rule tetrahedra

The implemented element stiffness matrix construction algorithm takes as input right hand rule tetrahedron (as the one shown in Figure B.1). In order to be able to load both right and left hand rule tetrahedra, a test is performed to the first tetrahedron of the mesh to identify the hand rule of the mesh. This check is performed by calculating the volume of the tetrahedron through the formula:

$$V = \frac{(N0 - N3) \times ((N1 - N3) \cdot (N2 - N3))}{6}$$

In the cases were the volume is negative, the tetrahedron is right hand rule and therefore no conversion needs to be done (like the tetrahedron in Figure B.1).

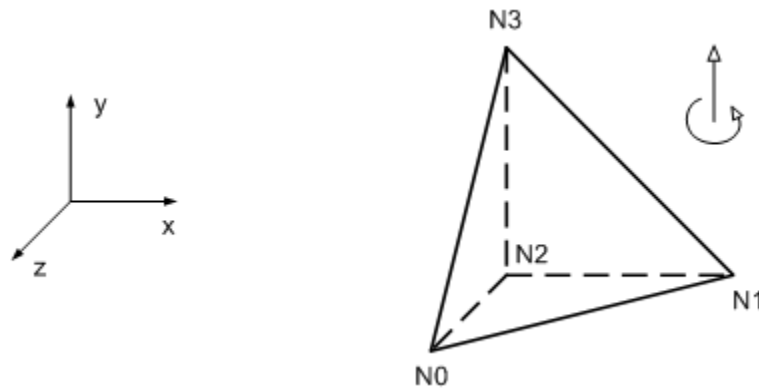


Figure B.1 – Right hand rule tetrahedron.

In the cases were the volume is positive, the tetrahedron is left hand rule). In such cases, a reordering of the nodes is performed to every tetrahedron of the mesh by exchanging the two last nodes. This operation does not change the geometry or topology of the tetrahedra and changes them to right hand rule.

On the left side of Figure B.2 a left hand tetrahedron is shown. The volume test is positive, so the last two nodes are switched. The only consequence of this reordering was turning the tetrahedron to right hand rule (right hand rule of Figure B.2).

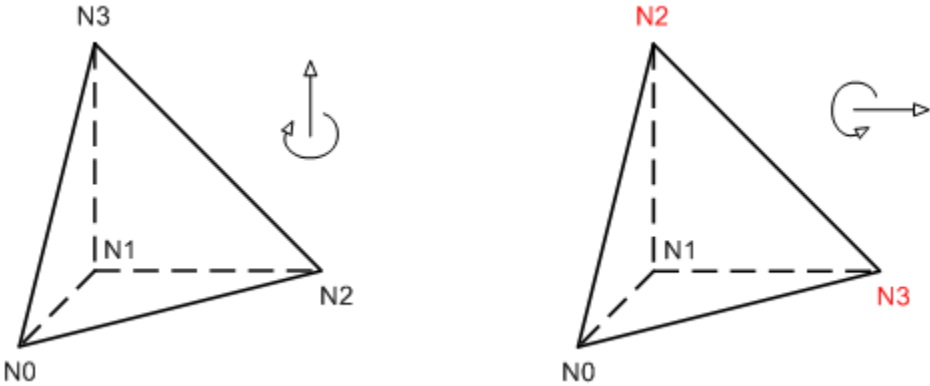


Figure B.2 – On the left, a left hand rule tetrahedron. On the right, the same tetrahedron converted to right hand rule.