Master Thesis

FEM-based elasticity using quasi-conformal deformations

Author(s):
Huber, Christoph

Publication Date:
2009

Permanent Link:
https://doi.org/10.3929/ethz-a-005816934

Rights / License:
In Copyright - Non-Commercial Use Permitted
FEM-Based Elasticity Using Quasi-Conformal Deformations

Huber Christoph

Master Thesis
May 2009

Martin Sebastian

Prof. M. Gross
Abstract

This thesis addresses the development of a novel technique for simulating deformable models (GCFEM). Our method is similar to FEM systems but uses special basis functions for the Ritz-Galerkin discretization. Unlike the usually applied basis functions that are associated to vertices only, the Green coordinates (GC) additionally contain basis functions associated to normals. We use Green coordinates because they lead to space deformations with a shape-preserving property. Green coordinates in 2D induce conformal mappings and quasi-conformal mappings in 3D. Our FEM-like system overtakes these desirable properties and therefore the resulting deformations are by construction able to preserve shapes better than classical FEM systems. The additional normal-based basis functions yield a non-linear problem. By performing a linearization we are able to arrive at a linear problem which can be solved easily, similar to linear elastic problems solved with the classic FEM.

Furthermore, we also show two other possibilities how Green coordinates can be used in elastic deformation systems. One alternative approach is GC fitting that allows us to simulate the dynamics with an arbitrary method and enhances the surface mesh’s representation with Green coordinates. The other method is based on an element-based FEM system and uses Green coordinates only for the representation of the deformation.
Zusammenfassung

Diese Arbeit umfasst die Entwicklung einer neuartigen Technik, mit der Modelle deformiert werden können (GCFEM). Unsere Methode ähnelt FEM, basiert aber auf Green Koordinaten, die nicht nur auf Knoten definiert sind, sondern auch auf Normalen. Unsere Motivation, Green Koordinaten zu verwenden, kommt daher, dass sie die Eigenschaft besitzen, bei Deformationen im Raum die Form eines Objektes gut zu erhalten. Winkel einer 2D Abbildung werden komplett erhalten. In 3D werden die Winkel zumindest so gut wie möglich erhalten, was auch Quasi-Konformalität genannt wird. Unser Deformationssystem übernimmt diese guten Eigenschaften, daher behält unser System die Winkel eines deformierten Objektes besser als dies andere FEM Systeme tun.

Die zusätzlichen Basisfunktionen führen zu einem nicht-linearen System, durch dessen Linearisierung wir auf ein linear Problem kommen, das dem klassischen FEM ähnlich ist.

Zusätzlich haben wir auch andere Wege untersucht, wie Green Koordinaten in elastischen Deformationssystemen angewendet werden können.

Eine Alternative ist GC Fitting das uns erlaubt, die dynamische Simulation mit einer beliebigen Methode durchzuführen und anschliessend die Repräsentierung des Oberflächengitters mit Green Koordinaten aufzubessern. Die andere Methode basiert auf Element-basiertem FEM und benutzt Green Koordinaten ebenfalls nur für die Darstellung des deformierten Oberflächengitters.
## Contents

1. Introduction ......................................................... 1  
   1.1. Motivation .................................................... 1  
   1.2. About This Thesis ........................................... 2  

2. Related Work ....................................................... 3  
   2.1. Finite Element Method ....................................... 3  
   2.2. Coordinates .................................................. 4  
   2.3. Other Physically-based Systems ............................... 5  
   2.4. Shape Matching ............................................... 6  

3. Elastically Deformable Solids and Their Ritz-Galerkin Discretization 7  
   3.1. Representation ............................................... 7  
   3.2. Strain ......................................................... 7  
   3.3. Stress ........................................................ 10  
   3.4. Energy ........................................................ 11  
   3.5. Ritz-Galerkin ............................................... 12  
      3.5.1. Properties of Basis Functions ......................... 14  
      3.5.2. Discretized Energy ....................................... 15  
   3.6. Quadrature Points ............................................ 17  
   3.7. Hexahedron- and Octahedron-based FEM ...................... 18  

4. Green Coordinates .................................................... 21  
   4.1. Idea .......................................................... 21  
   4.2. Definition .................................................... 23  
   4.3. Scaling Factor ............................................... 24  
   4.4. Properties of GC ............................................. 25
A.3. Dynamic System .................................................. 79

B. Gradient Computations .............................................. 81
   B.1. Gradient of Normal and Scaling Factor ..................... 81
       B.1.1. Gradient of Normal ..................................... 81
       B.1.2. Gradient of Scaling Factor ............................ 82
   B.2. Gradient phi and psi ......................................... 82

C. Derivation of The Total Energy's Derivative ..................... 87

Bibliography .......................................................... 89
Introduction

1.1. Motivation

Animating elastically deformable objects is essential for many graphical applications since most objects are not fully rigid. Simulating elastic objects is a difficult problem in computer graphics. There are methods that contain cheap computations and allow quick user interaction as it is desired in video games. But these methods have a lack of physical precision and visual appeal. Applications like surgery simulators (as in Figure 1.1) require an accurate representation of the deformation as well as a realistic physical model. This can only be done at the expense of the simulation’s framerate. A method that is based on correct physics is FEM.

In this thesis, we want to generate a realistic physical model similarly to FEM. Furthermore, we use Green coordinates to get quasi-conformal and shape-preserving deformations.

Figure 1.1: Oregon Health and Science University School of Medicine recently acquired a new high-fidelity simulation device that enables surgeons to practice complex operative tasks before entering the operating room, improving patient safety in Oregon and beyond.
1. Introduction

Figure 1.2.: A 2D illustration of an object that is deformed using Green coordinates (left) and mean value coordinates (right). This picture is taken from [LLCO08].

1.2. About This Thesis

This thesis addresses 3D elastic deformations that are simulated on a FEM-like system using Green coordinates. The use of Green coordinates seems to be very promising since they show desirable properties such as quasi-conformity. A property that other coordinates do not show is the quasi-conformity. Green coordinates represent a deformation such that the distortion is kept at a minimum and the shape of the deformed object therefore stays well preserved. An example of a 2D deformation using Green coordinates and non-quasi-conformal coordinates is depicted in Figure 1.2.

We describe different ways how FEM and Green coordinates can be combined. While other coordinates are based on vertex positions only, Green coordinates respect vertex positions and normal directions. Hence, they bring a normal-based term to our FEM system. This new term is not linear with respect to the vertex positions and therefore leads to a non-linear system. We show several possibilities how we can approximate the normal-based term to get a linear FEM. The motivation of introducing this non-linear term is that we get additional forces that try to preserve an object’s shape by minimizing its distortion.

Before we start with the derivation of the FEM-like system based on Green coordinates, we give an overview of the FEM method and classical coordinates used therein. Also other simulation techniques like shape matching are briefly discussed since they are used in Chapter 7.

Chapter 3 introduces our physical model and the Ritz-Galerkin discretization which is the basis of every FEM system.

Chapter 4 is devoted to the definition and properties of Green coordinates. The Ritz-Galerkin discretization based on Green coordinates is presented in Chapter 5. We discuss the problems of non-linearity as well as a simple approach that uses FEM for simulating and Green coordinates for the representation of the deformed object. The following chapter shows a way to approximate the system. It also discusses warped stiffness, boundary constraints and collision handling specifically for our system.

Chapter 7 presents an alternative technique which constructs a Green coordinates deformation from an arbitrary deformation. We call this GC fitting.

Finally, some results of our proposed methods are presented and a conclusion closes off this report.
Related Work

Before the technical chapters begin, some related work is presented. We start with elastically deformable solids that are discretized by FEM. Then, some work about novel coordinates that can be used in FEM is discussed. We also mention other physically-based methods for the animation of elastic models. Finally, we briefly refer to some papers that describe a geometrical approach to dynamic elastic deformations.

2.1. Finite Element Method

To model deformable objects in a physically realistic manner, the material is regarded as a continuum. The physical laws yield partial differential equations (PDE) that describe the static and dynamic behavior of the material. These equations can be numerically solved by using the finite element method (FEM) presented in [Bat82]. When choosing the physical model, the strain can be chosen to be linear or non-linear. Linear strain only models small deformations accurately, but leads to systems that can be solved efficiently. Even though linear strain leads to fast and stable systems, the object volume increases unnaturally under large rotations, because the linear strain cannot represent rotations correctly.

Non-linear elasticity models large rotations correctly as shown in [ITF04]. On the other hand, the stiffness matrix is no longer constant and the process is slow and numerically instable for certain cases.

In year 2002, Mueller et al. proposed in [MDM+02] a FEM system that is based on linear elasticity but is able to represent rotations correctly. They described that the stiffness matrix can be
warped in such a way that the rotation has no influence on the elasticity. In [MG04], Mueller et al. extended their model to also simulate plasticity and fracturing.

Until now FEM was based on conforming elements. This means, that the elements build a subspace of the approximated object, mostly by subdiving the surface mesh. This can lead to a complex geometric representation and therefore to an overhead of time and memory consumption.

In practical applications, non-conforming elements have two noteworthy advantages to conforming elements. First, non-conforming elements can easily be adaptively redefined in regions where more precision is required. Secondly, non-conforming elements allow two independently analyzed substructures in a large system to be interconnected. It is highly unlikely that two independent substructures with conforming elements coincide.

A simple and fast approach based on non-conforming elements is proposed in [MTG04] where surface meshes are voxelized in order to get a regular cube grid.

Another improvement is presented in [MKB+08]. There, arbitrary polyhedral elements can be used for discretization because the points within the elements are represented with harmonic coordinates. This allows to refine the elements dynamically when the stress has exceeded a threshold.

The GCFEM system presented in this thesis extends FEM to a system that is discretized with Green coordinates and contains only one non-conforming element (see Chapter 5).

2.2. Coordinates

FEM requires coordinates that interpolate function values within the domain. Easy to compute are trilinear or homogeneous coordinates. They are based on linear interpolation and the key idea is to consecutively interpolate linearly in three directions. They can be used on cube elements or other predefined shapes, but not on arbitrary elements.

Popular coordinates that can be used on arbitrary closed triangular meshes are mean value coordinates. They are presented in [Flo03] and then defined for arbitrary 3D-simplex meshes by [JSW05].

Joshi et al. [JMR+07] introduced harmonic coordinates. Harmonic coordinates are non-negative and do not possess a local extrema. These properties lead to more intuitive control in the deformation process compared to mean value coordinates. On the other hand, harmonic coordinates do not possess closed-form formulas as exist for mean value coordinates and are therefore more cumbersome to compute.

Both mean value coordinates and harmonic coordinates are only vertex-based. Green coordinates [LLCO08] also respect the direction of an element’s normal. Our motivation for using Green coordinates for our system are the properties of these coordinates. Deformations based on Green coordinates are shape-preserving and quasi-conformal. With quasi-conformal, we mean that shear component of a transformation is minimized. Furthermore, a closed formula can be derived for Green coordinates.
2.3. Other Physically-based Systems

Figure 2.1 shows a comparison of the three coordinates.

![Figure 2.1](image)

(a) Undeformed  (b) MVC  

(c) HC  (d) GC

**Figure 2.1:** Deformation of a text with an element of eight nodes [LLCO08]. The results of the mean value (MVC), harmonic (HC) and Green coordinates (GC) are displayed.

2.3. Other Physically-based Systems

Terzopoulos et al. [TW88] presented a physically based approach in which deformations are decomposed into a rigid component and a displacement component, allowing deformation away from the rigid shape. The rigid component captures the rigid motion while the displacement component describes the discretized relative displacement to the rigid body. This method resembles shape matching that will be described in the next section but it cannot represent an object that contains regions with significant different rotations.

ArtDefo was presented in [JP99] and is another physically based approach. Unlike FEM, the integrals that are solved are defined on the boundary only. Hence, there is no need for a discretization of the volume and the discretization is only performed on the boundary. This approach is therefore called BEM (boundary element method). ArtDefo uses linear strain and is therefore applicable only for small deformations.

Desbrun et al. [DSB99] proposed a method based on mass-spring systems. The method splits the forces into linear and non-linear parts. The non-linear rotational part is at first neglected and the estimated deformation is correct after the approximation of the implicit integration in order to preserve momentum.
2. Related Work

2.4. Shape Matching

An alternative approach for simulating deformable solids is based on shape matching. Unlike FEM, the underlying model is not physically correct. Shape matching is only geometrically motivated [MHTG05].

The dynamic shape matching simulation integrates explicitly over time and is based on forces that point to goal positions which are chosen such that an object’s shape is preserved. Shape matching is very efficient and allows fast simulations since the model is simpler and does not require as much computation as FEM. Furthermore, the memory consumption is low and the system is unconditionally stable. Even though shape matching deformations are not physically correct, shape matching is very interesting for real-time applications.

Rivers and James [RJ07] extended shape matching to a regular lattice that embeds the object’s geometry (LSM) and allows very efficient computations (Fast LSM). Later, Steinemann et al. [SOG08] further extended the model to use adaptive octrees instead of regular lattices.
Elastically Deformable Solids and Their Ritz-Galerkin Discretization

This chapter presents the physical model used for simulating elastically deformable solids. First the geometric representation is introduced, then the physical model is described, the notion of elastic energy elucidated and the resulting forces derived.

3.1. Representation

We consider an object that consists of a set of points $x^0$. A point is deformed from its original position $x^0$ to $x(x^0)$. The deformation of the point $x^0$ is given by

$$u(x^0) = x(x^0) - x^0. \hspace{1cm} (3.1)$$

Figure 3.1 shows a deformation of an ellipse and a point $x^0$ on the ellipse.

3.2. Strain

Our physical model needs a geometrical measure of deformations. Thus, we introduce strain. Strain is a measure how much a given displacement differs locally from a rigid-body displacement. In 1D, given a spring with length $l$ and a displacement $d$, the strain would be $d/l$. In 3D, the strain is not that intuitive.
3. Elastically Deformable Solids and Their Ritz-Galerkin Discretization

![Figure 3.1: A point $x^0$ undergoes a deformation $u(x^0)$.](image)

We consider again a point $x^0$ and assign an arbitrary direction vector $d^0$ to $x^0$. The deformation’s strain is measured on the point and the corresponding direction vector. After a deformation which includes the point’s displacement $u(x^0)$, we get a new position $x(x^0)$ and a new direction vector $d$. An example of a deformation is shown in Figure 3.2.

![Figure 3.2: The position $x$ and a direction vector $d$ of an ellipse before and after a deformation.](image)

The new direction vector after a deformation $u$ is

$$d = \left( I + \nabla u \right) J d^0.$$  \hspace{1cm} (3.2)

$I$ is the identity matrix and $\nabla u$ is a $3 \times 3$ matrix that contains the derivatives of $u$:

$$\nabla u = \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{pmatrix}.$$ \hspace{1cm} (3.3)

$J = I + \nabla u$ is called Jacobian of the deformation.

Now, we consider the difference in length of the direction vector before and after the deformation. The tensor that is multiplied with the direction vector, is called the strain tensor. By
3.2. Strain

definition the strain is a measure of how much a deformation differs from a rigid-body deformation. The direction vector’s length does not change after a rigid-body deformation. Hence, the difference between non-rigid and rigid deformation is

\[
\|d\|^2 - \|d^0\|^2 = d^0^T J d^0 - d^0^T d^0 = d^0^T (J^T J - I) d^0.
\]

(3.4)

We get the change in length of each direction vector by building a bilinear form. The matrix \( \epsilon_G \) is independent of the chosen direction vector and therefore a good measurement for the strain. \( \epsilon_G \) is called the Green strain and is defined as

\[
\epsilon_G = \frac{1}{2} (\nabla u + \nabla u^T + \nabla u^T \nabla u).
\]

(3.5)

The Green strain is rotation-invariant, i.e. rotations are handled correctly. But it is a quadratic function with respect to the deformation and in order to get a linear FEM, it is preferable to use linear strain.

For a small deformation, the quadratic term \( \nabla u^T \nabla u \) can be neglected. Thus, we get a linear strain tensor

\[
\epsilon_C = \frac{1}{2} (\nabla u + \nabla u^T).
\]

(3.6)

\( \epsilon_C \) is called Cauchy strain and is defined component-wise by

\[
\epsilon_C = \frac{1}{2} \begin{pmatrix}
2u_x & u_y + v_x & u_z + w_x \\
v_x + u_y & 2v_y & v_z + w_y \\
w_x + u_z & w_y + v_z & 2w_z
\end{pmatrix} = \frac{1}{2} \begin{pmatrix}
2\epsilon_x & \gamma_{xy} & \gamma_{xz} \\
\gamma_{xy} & 2\epsilon_y & \gamma_{yz} \\
\gamma_{xz} & \gamma_{yz} & 2\epsilon_z
\end{pmatrix}.
\]

(3.7)

The Cauchy strain is sufficient for small rotations. But it is not rotation-invariant and for larger rotations we get artifacts. For linear FEM there is a method called warped stiffness that prevents artifacts (see Chapter 6.3).

Equation (3.7) shows the strain tensor that we want to use. In the next section, we will introduce a linear dependency between strain and stress, i.e. we can define a single matrix that describes the dependency.

To make use of this matrix, we use for strain and stress vectors instead of \( 3 \times 3 \) tensors. We utilize the symmetry of the strain tensor to turn it into a 6D vector:
3. Elastically Deformable Solids and Their Ritz-Galerkin Discretization

\[
\epsilon_C = \begin{pmatrix}
\epsilon_x \\
\epsilon_y \\
\epsilon_z \\
\gamma_{yz} \\
\gamma_{xz} \\
\gamma_{xy}
\end{pmatrix} = \begin{pmatrix}
\delta/\delta x & 0 & 0 \\
0 & \delta/\delta y & 0 \\
0 & 0 & \delta/\delta z \\
0 & \delta/\delta z & \delta/\delta y \\
\delta/\delta z & 0 & \delta/\delta x \\
\delta/\delta y & \delta/\delta x & 0
\end{pmatrix} \cdot \begin{pmatrix}
u \\
w \end{pmatrix}
\]

The geometric meaning of the displacement’s derivatives are depicted in Figure 3.3. The derivative of a dimension with respect to the same dimension is a scaling. The derivative of a dimension with respect to another dimension represents shearing.

![Figure 3.3: Geometrical meaning of displacement’s derivatives](image)

3.3. Stress

Right now, our physical model includes displacements and strain that describes a deformation geometrically. We also need a measure that describes forces that deform an object. Stress is a measure of force on an object’s surface. More precisely, it indicates force per area acting within a deformable body across a virtual surface.

The Cauchy Stress $\sigma$ is represented by a $3 \times 3$ tensor. On a surface point, the following relation holds:

\[
f = \sigma \cdot n
\]

where $n$ is the normal of a surface and $f$ the force that is acting on the surface. Thus, each row of the stress tensor represents the force in direction of a basis vector (consider also Figure 3.4).
3.4. Energy

Stress and strain are related to each other. A force induces a deformation and vice versa. In a linear model, the relation between strain and stress is stated by Hooke’s law. In order to describe the relation in only one matrix, we turn the stress tensor into a 6D vector $\sigma$. Hooke’s law can then be written as

$$\sigma = E \epsilon.$$  \hspace{1cm} (3.10)

$E$ is a constant $6 \times 6$ matrix, that represents the material properties of an object. It contains the two material parameters $e$ and $\nu$. $e$ is called Young’s modulus and indicates the stiffness. $\nu$ is the Poisson’s ratio and indicates volume preservation under deformation where $\nu < \frac{1}{2}$.

$$E = \begin{pmatrix} (1 - \nu)e & vc & vc \\ vc & (1 - \nu)e & vc \\ vc & vc & (1 - \nu)e \end{pmatrix} \begin{pmatrix} 0 \\ \underbrace{G} \\ \underbrace{G} \end{pmatrix}$$  \hspace{1cm} (3.11)

where $c = \frac{e}{(1+\nu)(1-2\nu)}$ and $G = \frac{e}{2(1+\nu)}$.

Hooke’s law assumes a linear stress-strain relation that is only valid for small deformations.

As an example, some materials’ properties are listed in Table 3.1.

3.4. Energy

In this section, we introduce the total energy $\mathcal{E}$ of the physical system. Energy describes the amount of work that can be performed by a force. In a closed system, the total amount of energy is always the same while there is no dissipation. Different energies are transformed into other
3. Elastically Deformable Solids and Their Ritz-Galerkin Discretization

<table>
<thead>
<tr>
<th>Material</th>
<th>Young’s modulus in GPA</th>
<th>Poisson’s ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rubber</td>
<td>0.01-0.1</td>
<td>~ 0.50</td>
</tr>
<tr>
<td>Magnesium</td>
<td>45</td>
<td>0.35</td>
</tr>
<tr>
<td>Copper</td>
<td>110-130</td>
<td>0.33</td>
</tr>
<tr>
<td>Concrete</td>
<td>30-100</td>
<td>0.2</td>
</tr>
</tbody>
</table>

*Table 3.1.: List of materials and their properties*

...energies, e.g. kinetic energy is transformed into potential energy and vice versa. In real systems however, the total energy always dissipates due to friction or turbulence.

A system is in mechanical equilibrium if the gradient of the energy is zero. Or in other words, when the energy is minimal the system is in a steady state.

First, we consider the static solution of our problem. We will also consider dynamic systems since they are interesting for animations. The difference between dynamic and static system is that in the static system we search for the steady-state only while the dynamic system minimizes the energy step-wise.

The elastic energy stored in a deformed object is given by

\[
E(u) = \frac{1}{2} \int_{\Omega} \epsilon(u)^T \sigma(u) = \frac{1}{2} \int_{\Omega} \epsilon(u)^T E \epsilon(u)
\]  

(3.12)

where \( \Omega \) is the space within the undeformed object. We will always consider the energy of the system as a function of the displacement.

Furthermore, we can add external forces \( f \) to the system. Potential energy is defined as the work against a given force by changing the position of an object. Work is force times distance (or displacement). Hence, our additional energy is \( -\int_{\Omega} u^T f \) and the total energy becomes

\[
E(u) = \frac{1}{2} \int_{\Omega} \epsilon(u)^T E \epsilon(u) - \int_{\Omega} u^T f.
\]  

(3.13)

Figure 3.5 shows the energy function \( E \). In a static system, we try to find the displacement \( u \) which minimizes the energy function \( E \). It follows that the variational derivative \( \delta E / \delta u \) has to be zero. This leads to a PDE that can be solved by discretizing the displacement.

3.5. Ritz-Galerkin

To solve the PDE from Section 3.4 numerically, the displacements have to be discretized. Therefore, we perform a Galerkin discretization on Equation (3.13). We are moving the problem from...
3.5. Ritz-Galerkin

Figure 3.5: The energy function with respect to a 2D displacement $\mathbf{u}$.

an infinite dimensional space into a $N$-dimensional space spanned by some basis functions $\phi_i(x)$. The continuous displacement is now expressed as

$$
\mathbf{u}(x) = \sum_{i}^{N} u_i \phi_i(x^0)
$$

(3.14)

where $u_i$ is a displacement at discrete nodes and a degree of freedom of the discrete system. Equation (3.14) describes an interpolation with basis functions $\phi_i$. This interpolation is depicted in Figure 3.6 for the 1D case. Figure 3.7 shows an example of an object with domain $\Omega$ whose function values can be interpolated by four discrete nodes.

Figure 3.6: A function (blue) is approximated (green) as a linear combination of four discrete values (red).

The new solution vector space is now $N$-dimensional. The displacement that minimizes the
3. Elastically Deformable Solids and Their Ritz-Galerkin Discretization

The total energy is therefore a combination of $N$ displacements $u_i$.

$$\sum_i \phi_i t = \sum_i \phi_i t.$$ (3.15)

**Figure 3.7.:** An object (green) is depicted. The solution domain $\Omega$ lies within the object. Four discrete nodes $x_1, x_2, x_3, x_4$ are placed outside of the object. Function values at $x \in \Omega$ are interpolated by the four discrete nodes

### 3.5.1. Properties of Basis Functions

The basis functions $\phi_i$ must satisfy some requirements to guarantee a correct representation of the solution domain:

**Continuity** First of all, the basis function must be defined for all points in our solution domain $\Omega$. The basis functions have to be $C^1$ smooth within the domain and $C^0$ continuous across the boundary surface of $\Omega$.

**Translation invariance** In order to describe rigid body movements, the basis functions need to reproduce translations:

$$\sum_i \phi_i t = \sum_i \phi_i t.$$ (3.15)

To satisfy Equation (3.15) the basis functions must build a partition of unity, i.e. $\sum_i \phi_i = 1$.

Figure 3.8 visualizes the possible values of three basis functions. The plane depicts all the possible combinations of these three basis functions. Even though we have three basis functions, the space of the possible combinations is only 2D due to the partition of unity.

**Linear transformation invariance** In order to have rigid body movements, the basis functions also need to reproduce rotations and scalings. A linear deformation $A$ is applied to the discrete nodes. It can be shown that the deformed discrete nodes will lead to the same deformation of every point $x$ within the domain $\Omega$: 
3.5. Ritz-Galerkin

\[ \sum_i \phi_i A x_i = A \sum_i \phi_i x_i = A x. \] (3.16)

**Figure 3.8.: The possible combinations of three basis functions**

Only if the basis function fulfills those requirements, the system converges (for more details consider [Hug00]).

The strain measure requires that the gradients of the basis functions are known. We will see that the computation of the Green coordinates’ gradients are not trivial. Their implementation is complicated compared to simpler coordinates like trilinear coordinates for example.

### 3.5.2. Discretized Energy

Before discretizing Equation (3.13), a change of notation is introduced in order to simplify the following derivations:

\[
a(u, v) = \int_\Omega \epsilon(u)^T E \epsilon(v) \, dx
\]

\[
(u, f)_\Omega = \int_\Omega u^T f.
\] (3.17)

\(a(u, v)\) is a bilinear form and \((u, f)_\Omega\) is a standard inner product of a displacement and a force. Both are integrated over the solution domain.

The displacements in \(a(u, v)\) are now discretized as

\[ u = \sum_i \phi_i u_i. \] (3.18)
3. Elastically Deformable Solids and Their Ritz-Galerkin Discretization

Note that the displacements of the discrete nodes are not dependent on a point \( x \in \Omega \). Furthermore, for a bilinear form \( B(\sum_i s_i a_i, b) \) with vectors \( a_i \) and \( b \) and scalars \( s_i \), the following equation is valid:

\[
B(\sum_i s_i a_i, b) = \sum_i a_i^T B(s_i, b). \tag{3.19}
\]

The same is true for the transposed case:

\[
B(\sum_i b_i, s_i a_i) = \sum_i B(b_i, s_i) a_i. \tag{3.20}
\]

From the decomposition of the strain tensor (3.8), it follows that \( \epsilon(u_i \phi_i) = B_i u_i \), where \( B_i \) is a 6 \times 3 matrix. Combining this with Equation (3.19) yields

\[
a(u, v) = a(\sum_i u_i \phi_i, v) = \sum_i u_i^T a(\phi_i, v). \tag{3.21}
\]

Equation (3.20) also yields

\[
a(u, v) = \sum_i a(u, \phi_i) v_i. \tag{3.22}
\]

Furthermore we have

\[
(u, f)_\Omega = (\sum_i u_i^T \phi_i, f)_\Omega = \sum_i u_i^T (\phi_i, f)_\Omega. \tag{3.23}
\]

The derivations above help us to discretize the total energy. The discrete energy is

\[
\mathcal{E} = \frac{1}{2} a(\sum_i u_i \phi_i, \sum_j u_j \phi_j) - (\sum_i u_i \phi_i, f)_\Omega = \sum_i u_i^T \left( \sum_j \frac{1}{2} a(\phi_i, \phi_j) u_j - (\phi_i, f)_\Omega \right) \tag{3.24}
\]

and the gradient with respect to \( u_i \) is

\[
\frac{\delta \mathcal{E}}{\delta u_i} = \sum_j a(\phi_i, \phi_j) u_j - (\phi_i, f)_\Omega. \tag{3.25}
\]

To get the steady-state, for each \( i \), \( \frac{\delta \mathcal{E}}{\delta u_i} \) must be zero. This leads to a linear system of equations

\[
K u = f. \tag{3.26}
\]
3.6 Quadrature Points

\(u\) contains all the displacements \(u_i\) and is a vector of size \(3N\). The vector \(f\) contains all the \(f_i = \int_\Omega \phi_i f\) and is also of size \(3N\). Therefore \(K\) needs to be of size \(3N \times 3N\). \(K\) consists of \(N^2 3 \times 3\) block matrices

\[
K_{i,j} = B_i^T E B_j
\]  

(3.27)

where \(B_i\) is the \(6 \times 3\) matrix known from Equation (3.8) applied on \(\phi_i\):

\[
B_i = \begin{pmatrix}
\frac{\delta \phi_i}{\delta x} & 0 & 0 \\
0 & \frac{\delta \phi_i}{\delta y} & 0 \\
0 & 0 & \frac{\delta \phi_i}{\delta z} \\
0 & \frac{\delta \phi_i}{\delta x} & \frac{\delta \phi_i}{\delta y} \\
\frac{\delta \phi_i}{\delta z} & 0 & \frac{\delta \phi_i}{\delta x} \\
\frac{\delta \phi_i}{\delta y} & \frac{\delta \phi_i}{\delta x} & 0
\end{pmatrix}.
\]  

(3.28)

3.6 Quadrature Points

The entries of the linear system \(Ku = f\) are mostly integrals over the domain \(\Omega\). \(f_i = \int_\Omega \phi_i f\) and \(K_{i,j} = \int_\Omega \epsilon(\phi_i)^T E \epsilon(\phi_j)\).

We discretize the integral over the domain \(\Omega\) by approximating the integral as a finite sum over quadrature points.

In the precomputation phase, we distribute \(Q\) quadrature points over the space \(\Omega\). Each quadrature point has a position \(q_i\) and a weight \(w_i\). During the simulation, an integral over a function \(z(x)\) can be approximated as a sum over the quadrature points:

\[
\int_{x \in \Omega} z(x) \approx \sum_{i=1}^{Q} z(q_i) w_i.
\]  

(3.29)

The weights have to be chosen such that they approximate the domain \(\Omega\). Hence, the sum over all weights has to be equal to the volume of the domain. When no other possibility is given, we measure a weight by setting it to the distance from the quadrature point to its nearest neighbor.

For the quadrature point sampling, we use an algorithm that divides the bounding box of the object into a grid. Each grid point which is inside the object is handled. Given a desired amount of volume points, the algorithm chooses the points such that the distances to each other are maximal. This enlarges the possibility that small features are covered by a quadrature point as well.

As an example, we consider a man’s arm with the attached hand. With uniformly distributed
3. Elastically Deformable Solids and Their Ritz-Galerkin Discretization

volume points we would have most quadrature points in the arm and only few in the hand. If the resolution of the grid is not very high, it would be likely that some fingers do not include a quadrature point. With the furthest distance algorithm, we can choose a higher resolution of the grid and the quadrature points are also placed in the fingers since they maximize the distance.

A good sampling of the object volume is important for GCFEM. A good sampling that calls on more precomputation time is worth much more than a primitive sampling that contains lots of useless quadrature points. A primitive sampling can yield the same results, but far more quadrature points are required. And more quadrature points restrict the performance time.

After the quadrature points are sampled, the Green coordinates are computed at each point. Since the stiffness matrix requires derivatives of GC, we also have to precompute the derivatives.

3.7. Hexahedron- and Octahedron-based FEM

![Figure 3.9: A cactus is overlayed by a grid consisting of hexahedra. Each function value, and therefore each point of the cactus, can be described as a linear combination of the eight cell nodes. This can be used for a FEM deformation system.](image)

While the previous section introduced the general Ritz-Galerkin discretization of our problem, a special kind of Ritz-Galerkin discretization is presented in this section. It is called finite element method (FEM). In a FEM system, the space is subdivided into several elements. The function values within an element are interpolated by the values at the nodes of the element, i.e. the basis functions are defined at the element nodes.

An often used approach for elastic deformations is a FEM based on hexahedra or octahedra. An object is covered by a grid and described by the grid’s nodes. Figure 3.9 shows an example of a grid with cubic elements.

Element-based FEM systems use different basis functions, e.g. harmonic functions in [MKB⁺08]
or mean value coordinates in [WBG07]. Harmonic basis functions show all the required properties and are easier to implement than the Green coordinates. For a grid with cube elements (as in Figure 3.9), the harmonic basis function is equal to the trilinear basis function. They describe the relative distances from three edges of the cube. Thereby three edges that all point into another direction are considered. Assume our element is the unit cube and \( \phi(a,b,c)(x) \) was the basis function at the cube’s node \((a, b, c)\) for a point \( x = (x, y, z) \). Then we would get the following basis functions

\[
\phi(a,b,c) = a \cdot x + (1 - a) \cdot (1 - x) + b \cdot y + (1 - b) \cdot (1 - y) + c \cdot z + (1 - c) \cdot (1 - z).
\]

(3.30)

Figure 3.10: An object surrounded by a grid of squares. Every point in a cell is described by a linear combination of the four nodes.

Figure 3.10 shows an example in 2D. An object and a grid are depicted. We assume that bilinear coordinates are used. The displacement \( u \) of a point is approximated by the four nodes that surround this point. For instance, a point inside the square \((u_1, u_2, u_3, u_4)\) has a displacement \( u(x) = \phi_1(x^0)u_1 + \phi_2(x^0)u_2 + \phi_3(x^0)u_3 + \phi_4(x^0)u_4 \). The \( \phi \)'s are defined similar to trilinear coordinates with the difference that we only consider the distance to two edges.

Each cell can be handled individually. The cell’s stiffness matrix \( K^e \) and the force \( f^e \) are integrated over the cell’s volume. The stiffness matrix \( K \) of the whole system is the sum of all cell’s \( K^e \) and \( f = \sum f^e \). Since a point in a cell is only dependent on the cell’s nodes, most entries in \( K \) are zero and the matrix is sparse.

We consider again the example of Figure 3.10. The cell \((u_1, u_2, u_3, u_4)\)’s \( K^e \) has only non-zero entries for \( K_{i,j}, \{i,j\} \in \{1, 2, 3, 4\} \).

In Section 5.4 we will introduce a method to combine element-based FEM with Green coordinates (GC) representation. This is an easy extension. But the main topic of this work is how to
3. Elastically Deformable Solids and Their Ritz-Galerkin Discretization

use Green coordinates for Galerkin discretization. Thus, we discuss in the next chapter GC and their properties. Then, the Ritz-Galerkin discretization is reformulated with GC. It will add a new term to the linear system $\mathbf{Ku} = \mathbf{f}$ that should help to get quasi-conformal deformations.
Green Coordinates

This chapter introduces Green coordinates (GC), which were proposed lately in [LLCO08]. The chapter starts with elucidating the idea of GC. Then, the definition and the properties of the Green coordinates are declared. Finally, there is a discussion about cage design that is important for a deformation since a cage contains the discrete nodes that interpolate the interior.

4.1. Idea

In the FEM presented in the last chapter, we used basis functions that could interpolate function values as a linear combination of discrete nodes. The basis functions were defined on discrete nodes of elements.

Here, we only have a single element that surrounds the object. All function values in the domain \( \Omega \) can be approximated by a linear combination of this single element’s nodes and normals. We call this single element the cage. Another difference to classical coordinates is that a function value is not only represented by a linear combination of the cage nodes, the representation includes as well a sum over the cage’s normals multiplied by some weights. Thus, we have now two kinds of weights: One kind is multiplied with the cage nodes and the other with the cage normals.

On one side, the sum over the normals adds more complexity to our elastic deformation systems, as we will see. On the other side, however, GC have some properties that cannot be reached without this additional term. An interesting property of GC are the least-distorting and shape-preserving deformations they can reproduce. This is was our motivation to use GC in a
4. Green Coordinates

FEM-like system.

Figure 4.1 shows a dinosaur and a cage. Each vertex of the dinosaur has a position \( x \) which can be expressed as a linear combination of cage nodes and normals. The figure shows the deformation of the cage. The new cage vertices and normals determine the new positions of the dinosaur vertices and we get a nice deformation for the dinosaur object.

![Figure 4.1](image)

(a) Cage and object  
(b) Deformed cage  
(c) Object interpolated by cage nodes

**Figure 4.1.** A dinosaur and a manually modeled cage. First the GC for the dinosaur vertices with respect to the cage nodes are computed. Then, the cage is deformed and the dinosaur updated by applying the new cage positions and normals.

There are also other basis functions that approximate a point in the interior of an arbitrary cage. Popular examples are harmonic coordinates (HC) [JMR+07] or mean value coordinates (MVC) [JSW05]. Their basis functions are only based on cage nodes. Deformations based on these coordinates lead to more distortion and are therefore not interesting for our thesis. Figure 4.2 shows a deformation using GC and MVC. GC preserves the head’s shape clearly better.

![Figure 4.2](image)

(a) Ogre  
(b) GC  
(c) MVC

**Figure 4.2.** This figure is copied from [LLCO08]. It shows an ogre’s head and its deformation. Once Green coordinates are used and once mean value coordinates.
4.2. Definition

For a cage with vertices \( v_i, i = \{1, ..., V\} \) and triangles \( t_j, j = \{1, ..., T\} \), the coordinates \( \phi_i(\eta^0) \) and \( \psi_j(\eta^0) \) are the corresponding basis functions. A cage and the set of all possible \( \eta \in \Omega \) in 2D is depicted in Figure 4.3. In 3D, a face is a triangle instead of an edge.

For \( \eta \in \Omega \):

\[
\eta = \sum_{i=1}^{V} \phi_i(\eta^0)x_i + \sum_{j=1}^{T} \psi_j(\eta^0)n_js_j \tag{4.1}
\]

where \( x_i \) is the position of vertex \( v_i \). \( n_j \) is the normal of \( t_j \) and \( s_j \) a scalar that will be discussed in Section 4.3.

For a point \( \eta \in \Omega \), all the coordinates \( \phi_i \) and \( \psi_j \) can be precomputed. We omit here the complete derivation of the coordinates and refer instead to the GC paper [LLCO08]. The derivation is based on the Green’s third identity. The resulting computation of \( \psi_j(\eta^0) \) in 3D is the following integral over the triangle \( t_j \)'s area:

\[
\psi_j(\eta) = \frac{1}{4\pi} \int_{\epsilon \in t_j} \frac{1}{\|\epsilon - \eta\|} d\sigma_{\epsilon}. \tag{4.2}
\]

The computation of \( \phi_i(\eta^0) \) includes an integral over all the triangles that adjoin the vertex \( v_i \). The points on these triangles are denoted as the set \( N(v_i) \). \( \phi_i(\eta^0) \) is then defined as

\[
\phi_i(\eta) = \frac{1}{4\pi} \int_{\epsilon \in N(v_i)} \Gamma_i(\epsilon) (\epsilon - \eta) \cdot \mathbf{n}(\epsilon) \frac{d\sigma_{\epsilon}}{\|\epsilon - \eta\|^3}. \tag{4.3}
\]

\( \Gamma_i(\epsilon) \) is a piecewise-linear hat function defined on \( N(v_i) \). For a triangle with vertices \( (v_i, v_j, v_k) \), \( \Gamma_i(v_i) = 1 \) and \( \Gamma_i(v_j) = \Gamma_i(v_k) = 0 \).
4. Green Coordinates

![Image of Green coordinates](attachment:image.png)

**Figure 4.4.** An illustration of the Green coordinates from [LLCO08]. It shows the values of $\phi_i$ for the highlighted vertex and $\psi_j$ for the marked edge in 2D.

The domain of influence of both coordinates is visualized in Figure 4.4. $\phi_i$ decreases quickly when moving away from $v_i$. The same figure for bilinear coordinates would show a uniform decrease.

Even though the computation of $\phi_i$ and $\psi_j$ looks complicated, it can be done analytically. This is important to reduce the amount of numerical approximation errors we can get. A pseudocode for the computation can be found in [LLCO08]. Anyhow, it is not as easy to implement as other coordinates. It is necessary to execute the implementation with high precision and to check whether the argument of a cosine or sine function is not greater than one or less than minus one. Even more complicated is the implementation of $\nabla \phi_i$ and $\nabla \psi_j$. These derivatives are required in the following chapters. We implemented the derivatives of GC and present a pseudocode for GC and derivatives of GC in Appendix B.2.

4.3. Scaling Factor

![Image of Stanford bunny](attachment:image.png)

**Figure 4.5.** The Stanford bunny gets transformed using Green coordinates. When the scaling factor is chosen to be least-distorting, the extreme deformation leads to an increase of the volume but the shape is preserved. When the scaling factor is set to one, the shape is heavily distorted but the volume is better preserved. This transformation is actually unrealistic, but clearly illustrates the meaning of the scaling factor.

In this section we discuss the meaning of the scalar value $s_j$ in Equation (4.1). $s_j$ is multiplied with the normal $n_j$. Hence, it scales the normal and is therefore called scaling factor.
4.4. Properties of GC

In the following, we consider a deformation of a triangle from \( t_j \) to \( t'_j \). To guarantee Equation (4.1) for the initial cage (no deformation), we set the initial scaling factor \( s_j = 1 \). We consider a simplex that is defined by \( t_j \) and an additional vertex at \( x_{j1} + n_j \) where \( x_{j1} \) is a triangle node’s position. After the deformation, the simplex is given by \( t'_j \) and \( x'_{j1} + s_j n'_j \). We want to define \( s_j \) such that the mapping from the simplex to the deformed simplex is least-distorting.

With least-distorting we mean that the angles of the simplex are preserved as much as possible. This is reached by setting the scaling factor to the stretch that \( t_j \) undergoes (see [LLCO08]).

In 2D, the stretch of a face is simply \( \|t'_j\| / \|t_j\| \), the fraction of edge lengths. In 3D, we get a least-distorting deformation when

\[
s_j = \frac{\sqrt{\|u'|\|^2 \|v\|^2 - 2(u' \cdot v')(u \cdot v) + \|v'\|^2 \|u\|^2}}{\sqrt{8\text{area}(t_j)}}.
\]  

(4.4)

\( t_j \) denotes the face of the normal \( n_j \), \( u \) and \( v \) two edges of the (triangle) face before the deformation. \( u' \) and \( v' \) are the same edges after the deformation.

When we do not want least distortion, \( s_j \) is simply set to one. In our application, we allow a combination of the two presented scaling factors. A parameter \( \alpha \) describes the fraction of the least-distorting scaling factor:

\[
s_j = \alpha \cdot \frac{\sqrt{\|u'|\|^2 \|v\|^2 - 2(u' \cdot v')(u \cdot v) + \|v'\|^2 \|u\|^2}}{\sqrt{8\text{area}(t_j)}} + (1 - \alpha) \cdot 1.
\]  

(4.5)

Figure 4.5 shows a deformation with \( \alpha = 0 \) and \( \alpha = 1 \).

4.4. Properties of GC

In Section 3.5.1 we have seen that the basis functions of the Ritz-Galerkin discretization have to satisfy some requirements. Since we want to use GC for Galerkin discretization, we have to show that Green coordinates have the required properties.

**Continuity** The basis functions \( \phi \) and \( \psi \) are harmonic. Hence, they are \( C^\infty \) for all points in the interior of \( \Omega \). The basis function for Galerkin discretization has to be also \( C^0 \) continuous across the boundary surface of the domain. This is only satisfied if we define the domain \( \Omega \) to be inside the cage. Points \( \eta \) that lie on the cage boundary are not defined by GC.

**Translation invariance** As seen in Section 3.5.1, basis functions can only represent a translation when the coordinates \( \phi_i(\eta^0) \) build a partition of unity. This is satisfied for GC. The normal-based coordinates \( \psi_j(\eta^0) \) do not need to build a partition of unity since the normal of a triangle does not change for a translation:
4. Green Coordinates

\[
\sum_{i=1}^{V} \phi_i(\eta^0)(x_i + t) + \sum_{j=1}^{T} \psi_j(\eta^0)n_j = \sum_{i=1}^{V} \phi_i(\eta^0)x_i + \sum_{j=1}^{T} \psi_j(\eta^0)n_j + \sum_{i=1}^{V} \phi_i(\eta^0)t = \eta + t.
\]

(4.6)

**Rotation and scale invariance** A linear transformation \( A \) that contains rotation \( R \) and scaling \( S \) has to be represented correctly by Green coordinates. The normals of a transformed object undergo a transformation \((A^{-1})^T \). \( R \) is an orthogonal matrix and therefore \((R^{-1})^T = R \). It can be shown that a rotation of the cage leads to the same rotation of all \( \eta \in \Omega \):

\[
\sum_{i=1}^{V} \phi_i(\eta^0)Rx_i + \sum_{j=1}^{T} \psi_j(\eta^0)s_jRn_j = R \left( \sum_{i=1}^{V} \phi_i(\eta)x_i + \sum_{j=1}^{T} \psi_j(\eta)n_j \right) = R\eta.
\]

(4.7)

Furthermore we defined the scaling factor such that the scaling reproduction is given as well.

Green coordinates also have the property of quasi-conformity, what is our motivation to use them, but is not required for Galerkin discretization. Quasi-conformal means that the shear component of a transformation is minimized as much as possible. Using other coordinates as MVC or HC leads to more distortion.

Due to these properties Green coordinates can be used for Galerkin discretization and result in nice, low-distorting deformations. In the next chapter, we use GC in a FEM-like discretization. The additional normal-based weights \( \psi_j \) give new energies. We expect that those energies improve the deformations by reducing the distortion and shearing of the deformed object.

**4.5. Cage Design**

**4.5.1. Cage Creation**

If we want to use Green coordinates in a FEM-like system, we need a cage for every object that shall be deformed. The only requirement for the cage is, that the object may not touch or intersect the cage. A valid cage for a dinosaur is depicted in Figure 4.1.

For an object, there is an indefinite number of possible cages. But the design of the cage has a great influence on the possible deformations and on our simulation. In Figure 4.6 several cages for a stickman are presented. The simplest case is depicted at the top left. The cage is the bounding box of the object with a small offset (away from the object). It can be constructed automatically from every kind of object, but it cannot deform some special parts of the object. In our example, no leg and no arm can be transformed solely. Only the whole stickman can be
4.5. Cage Design

rotated and stretched. We can add some degrees of freedom by subdividing the bounding box (top right). This gives us more flexibility, we can deform the upper part or the lower part, but not an arm or leg solely.

Cage design is not the main part of our work and we decided not to spend much time on this issue. The cages depicted in Figure 4.6 at the bottom were created manually. The cage at the left hand side is still very primitive. Only the legs can be stretched slightly. A cage that allows deformation of every detail should look like the bottom right one. Note that we have only few details in our example: Arms, legs, torso, neck and head. For very complex objects in 3D we cannot separate all details.

4.5.2. Pros and Cons of Cage-based Coordinates

The importance of cage design can be regarded as a drawback or as an advantage. The drawback is that a simulation based on GC should be independent of the cage, since a cage is only a virtual help. On the other hand, every simulation needs a discretization method. For instance, element-based FEM has an auxiliary grid of elements. This grid can be constructed in several ways and has an influence on the physical simulation as well.

Another drawback of our method is that we need to design cages for every object that should be simulated whereas other methods create their discrete points automatically. We can do this only under some restrictions.

The design of a cage can also be an advantage. We are able to design the parts of an object that are transformed as one unity. And we can determine in which region we need more degrees of freedom than in others.
4. *Green Coordinates*
Ritz-Galerkin Discretization Using Green Coordinates

This chapter introduces the main part of the thesis: A FEM-like simulation based on Green coordinates (GCFEM). First, the energy has to be discretized again. This time we do not discretize a displacement with a linear combination of only vertex-based basis functions. As seen in the last chapter, GC discretize a position with a linear combination of vertex-based and normal-based basis functions. This will give us some additional energy terms. Later, we will discuss the resulting equations we get and why some more simplifications are required. The difference between GCFEM and other FEM systems is also elucidated. Finally, there is another proposition how GC can be used in FEM simulations.

5.1. Discrete Displacement

Using GC, a point’s position can be represented as $x = \sum_i \phi_i(x^0)x_i + \sum_j \psi_j(x^0)ns_j$. The other notations have been introduced in the previous chapter.

The displacement of a point is $u(x^0) = x - x^0$. We discretize it using GC:

$$u(x^0) = \sum_i \phi_i(x^0)x_i + \sum_j \psi_j(x^0)ns_j - \sum_i \phi_i(x^0)x_i^0 - \sum_j \psi_j(x^0)ns_j^0. \quad (5.1)$$

Note that the scaling factor in an undeformed state is always one. Hence $ns_j^0 = n_j^0$.

Rearranging the term and defining $m_j = ns_j - n_j^0$ yields
5. Ritz-Galerkin Discretization Using Green Coordinates

\[ u(x^0) = \sum_i \phi_i u_i + \sum_j \psi_j m_j. \]  

(5.2)

\( u(x^0) \) is a function that depends on the cage vertices only. \( m_j(u) \) is a function of the cage vertices as well. When \((u_1, u_2, u_3)\) are the displacements of the three nodes of the triangle \( t_j \), we get

\[ m_j(u_1, u_2, u_3) = \frac{(u_2 - u_1) \times (u_3 - u_1)}{\| (u_2 - u_1) \times (u_3 - u_1) \|} s_j(u_1, u_2, u_3) - n_j^0. \]  

(5.3)

Equation (5.3) shows that \( m_j \) is non-linear with respect to a displacement \( u_i \). Other FEM systems’ discretizations are normally linear. Thus, GCFEM will not be based on a simple linear system \( Ku = f \) like standard FEM does.

5.2. Energy Minimization

Similar to Section 3.5 we discretize the total energy of our system. We defined the energy \( E(u) \) for a static system in Equation (3.13). Now we discretize the displacement \( u \) using the GC representation (4.1):

\[ E = \frac{1}{2} a \left( \sum_i u_i \phi_i + \sum_i m_i \psi_i, \sum_j u_j \phi_j + \sum_j m_j \psi_j \right) - \left( \sum_i u_i \phi_i + \sum_i m_i \psi_i, f \right)_\Omega. \]  

(5.4)

The discretized energy has to be minimized in order to find the system’s steady point and therefore we consider the derivatives of the energy with respect to the displacements. The derivation of the derivative is quite long, thus we moved it to the Appendix C. Here, we consider only the simplified derivative with respect to \( u_i \):

\[ \frac{\delta E}{\delta u_i} = \sum_j a(\phi_i, \phi_j) u_j + \sum_j a(\phi_i, \psi_j) m_j - (\phi_i, f)_\Omega \]

\[ + \sum_j \left( \frac{\delta m_j^T}{\delta u_i} \right) \left( \sum_k a(\psi_j, \phi_k) u_k + \sum_k a(\psi_j, \psi_k) m_k \right) - (\psi_j, f)_\Omega. \]  

(5.5)

We have not resolved \( \delta m_j / \delta u_i \) because we first discuss the difference to FEM systems that are based on simpler basis functions and the problems we get with the non-linear terms that contain \( m_j \).

5.3. Problems of Nonlinear FEM

For the standard FEM the derivative of the energy results in a linear term. GCFEM suffers from the non-linearity of the terms \( \delta m_j / \delta u_i \) and \( m_j \). Of course we could solve (5.5) for \( u_i \) but we prefer to keep the problem in the linear domain.
Figure 5.1: The normal \( \mathbf{n} \) of the triangle \((0,0,0), (0,0,1), (0,1,0)\) is observed. The vertex at \( \mathbf{v} = (0,0,0) \) is shifted along the x-axis towards \( \mathbf{v} = (1,0,0) \). The graph at right shows the normal’s coordinates (blue) that occur during the shifting. The red curve shows the vertex \( \mathbf{v} \)’s coordinates while it is shifted toward \((1,0,0)\).

Linear reproduction as \( \mathbf{u} = \sum_i \phi_i \mathbf{u}_i \) leads to the discrete energy stated in (3.24). It can be minimized by finding the best displacement \( \mathbf{u}_i \) of every node. Imagine that we try to move \( \mathbf{u}_i \) in every direction in order to get a smaller energy. When moving \( \mathbf{u}_i \) in one direction, a displacement \( \mathbf{u} = \sum_i \phi_i \mathbf{u}_i \) changes only linearly with slope \( \phi_i \). In the GC case, the same movement would not lead to a linear change of \( \mathbf{u} \), since the displacement of a cage node also brings a change of several normals.

Figure 5.1 shows the unit cube and the change of a normal when shifting a node. This change of the normal leads to a non-linear displacement of \( \mathbf{u} \) and finally to a more complex solution space for minimizing the energy.

We decided to drop this non-linear approach. Going on with the discretized derivative in Equation (5.5) would lead to a very complex system. The implementation would be hard to do and we expect that the performance would be considerably slow. These reasons made us try out simplifications such as approximating the normals in \( \mathbf{m}_j \). This approach will be described in the next chapter. We implemented an approach that uses constant \( \mathbf{m}_j \)’s which are updated iteratively. We propose also an approach that linearizes the normals (Appendix A). Before, we will shortly discuss how GC could be used in FEM simulations in another way.

5.4. GC Representation for Standard FEM Simulations

A much simpler method can be reached when we do not rely on GC for the FEM. Given an object and a grid that approximates this object, we simply do an element-based FEM simulation (as in [MKB’08] for example) that is fully independent of GC. The deformed grid is merged
5. Ritz-Galerkin Discretization Using Green Coordinates

![Diagram showing voxelization and deformation process](image)

(a) Voxels  (b) Deformed cage  (c) Deformed object

**Figure 5.2:** Element-based FEM using GC only for representation. First the object is voxelized. Then, the inner nodes are removed to get a cage. The object is represented by GC on the deformed cage.

such that only the boundary nodes stay. This merged grid is taken as the cage for the representation of the object. Algorithm 1 describes the method and Figure 5.2 depicts a deformation done with this technique.

**Algorithm 1** The algorithm of the element-based FEM with GC representation.

The object is voxelized to get the grid with the elements

while running do
  The object and the grid undergo an element-based FEM simulation
  The deformed grid is merged into a cage
  The computed cage determines the object’s deformation
end while

Even though this simulation results in nice animations (consider Figure 5.3), it is not the goal of this thesis. The FEM simulation is fully independent of GC and the physics does not consider GC deformations.
5.4. GC Representation for Standard FEM Simulations

Figure 5.3.: The object is deformed using element-based FEM. The FEM works with trilinear coordinates within the voxels. Replacing them by GC for the representation of the object yields a nicer result.
5. Ritz-Galerkin Discretization Using Green Coordinates
GCFEM With Constant Normal

The chapter before introduced GCFEM, a FEM-like method that is based on Green coordinates (GC). We present an approximated approach that leads to a system of linear equations. The linearity is gained from the constant approximation of the normal and scaling factor. After the static system is derived, warped stiffness is described. Furthermore we also show a possibility to handle plane collisions and boundary constraints. Finally, the static system is turned into a dynamic system in order to compute animations of deformable solids.

In Appendix A we describe how GCFEM could be improved by linearizing the normals instead of assuming them to be constant.

6.1. Taylor Approximation

Equation (5.5) shows that the complexity of the energy is augmented when we use GC instead of only vertex-based basis functions. We want to reduce the complexity and linearize the problem in order to get linear FEM-like equations which are known to give a stable and fully functional system.

$m_j(u)$ is a function of the cage vertex displacements. Consider the Taylor approximation of $m_j(u^t)$ at a time $t$:

$$m_j(u^t) = m_j(u^{t-1}) + \nabla m_j(u^{t-1})^T(u^t - u^{t-1}) + \nabla^2 m_j(u^{t-1})^T(u^t - u^{t-1})^2 + ...$$  

(6.1)
If we assume that the displacement \( u \) changes only slightly, we can approximate \( m_j(u^{t-1}) \) constantly, i.e.

\[
m_j(u^t) = m_j(u^{t-1}).
\]  

(6.2)

Figure 6.1 illustrates the constant approximation of a normal.

![Figure 6.1](image)

**Figure 6.1.** Depicted is the same change of normal as in Figure 5.1, but only the x-coordinate of the normal over time \( t \) (blue). The red graph is the constant approximation of the normal.

Now, we want to compute the energy’s derivative (5.5) with the constant \( m_j \). Since \( m_j \) is a constant, its derivative is zero and we can omit the whole term from (5.5) that is multiplied with \( \delta m_j / \delta u_i \):

\[
\frac{\delta E}{\delta u_i} = \sum_j a(\phi_i, \phi_j) u_j + \sum_j a(\phi_i, \psi_j) m_j - (\phi_i, f)_{\Omega}.
\]  

(6.3)

Equation (6.3) shows that we have a linear system again. The equations have to be solved for each \( u_j \) which is multiplied with the stiffness matrix blocks \( K_{i,j} = a(\phi_i, \phi_j) \). The integral over the external forces on the domain \( \Omega \) times \( \phi_i \) also appears in the Galerkin discretization in Equation (3.25). The only new term is \( \sum_j a(\phi_i, \psi_j) m_j \) where \( m_j \) is updated iteratively with the displacements of the last iteration. Thus, this term is independent of the degrees of freedom \( u_i \) and can be added to the force vector.

We write the new term as \( H_{i,j} m_j \) with new \( 3 \times 3 \) block matrices

\[
H_{i,j} = B_i^T E C_j
\]  

(6.4)

where \( B_i \) and the material matrix \( E \) are known from (3.8) and (3.11). \( C_j = \epsilon(\psi_j) \) is the same \( 6 \times 3 \) strain matrix as \( B_i \) but contains the derivatives of \( \psi_j \) instead of \( \phi_i \);
6.2. Forces on Cage

In Chapter 3.4 the potential energy of an external force is added to the elastic energy. The derivative of the potential energy with respect to a cage node is \((\phi, f)_\Omega\) when \(n_{s_j}\) is assumed to be constant.

The interpretation of this result is that whenever we have a force field \(f(x)\) in the domain \(\Omega\) we can determine a nodal force by integrating the force field times \(\phi_i\) where \(i\) is the index of the node. Consider the visualization of the \(\phi_i\) values in Figure 4.4. The \(\phi_i\) value is very small for most of the points in the domain. Only points that are close to the node \(i\) get a crucial weight. For the expression \((\phi, f)_\Omega\) this means that the nodal force is the interpolated value of the force field in its vicinity.
6. GCFEM With Constant Normal

Figure 6.2: A volume with some quadrature points \( q_i \) and a cage. In the volume is a force field \( f(x) \) (grey). Some discrete forces are sampled at the quadrature points (black). The discrete forces in the domain are integrated for each cage node. The integral’s result is the nodal force (red).

We do not consider the continuous case as we need a discretization of the force field. The volume can be approximated by quadrature points (see Section 3.6). Thus, we need to define the external force on the quadrature points. Figure 6.2 shows a force field with discrete forces at quadrature points and the integrated nodal forces.

For most cases when external forces are added to the system, the forces appear on the object’s surface mesh, i.e. on the domain’s boundary. For instance in our implemented application, the user has the possibility to add a force directly on the surface mesh. A mesh vertex is not a quadrature point and possesses no specified weight. Actually we should also define surface quadrature points with weights that approximate the points’ sizes. But we use meshes with uniformly distributed vertices and therefore we distribute the weights uniformly as well. This means that we can omit the weights and a nodal force becomes

\[
\sum_j \phi_i(m_j)f_j \tag{6.7}
\]

where \( m_j \) is the mesh vertex and \( f_j \) the force that acts on \( m_j \). Even though this approach requires uniform meshes, which are seldomly used, its implementation shows satisfying results for the tested meshes.

6.3. Warped Stiffness

The most important drawback of the linear Cauchy strain is that we lose the rotation-invariance. One approach to prevent artifacts for small time steps is called warped stiffness ([MDM+02],...
6.3. Warped Stiffness

[MG04] and [WBG07]). Figure 6.3 gives an example of the improvement that warped stiffness brings.

**Figure 6.3:** One end of a bar is fixed. The other elements are tied down by gravity forces.

The problem of linearized strain is that the strain is represented wrongly for a deformation that includes rotation. By eliminating the rotation of the deformation, we get a rotation-invariant strain. This is exactly the idea of warped stiffness. We undo the rotation of a deformation and compute the strain afterwards. Then the resulting force is rotated back.

To get a concrete solution of warped stiffness for our system, we first consider a block of the stiffness matrix $K$:

$$K_{i,j} = \int_{\Omega} a(\phi_i, \phi_j) = \sum_q a(\phi_i(q), \phi_j(q))w_q = \sum_q K^q_{i,j}w_q.$$  

(6.8)

According to Section 3.6 the integral is turned into a sum over quadrature points. We call this sum assembly of the stiffness matrix. $K^q_{i,j}$ is the stiffness matrix block for the quadrature point $q$.

We need to consider the deformation of the quadrature points. As we described in Section 3.2, a point’s deformation is given by the deformation’s Jacobian

$$J_q = \nabla u(q) + I.$$  

(6.9)

From the deformation’s Jacobian, we can extract the rotation. A transformation consists always of a scaling and a rotation part (depicted in Figure 6.4). We can split these two parts by polar composition

$$J_q = U^T_q \Sigma_q V_q.$$  

(6.10)

$\Sigma_q$ contains the scaling factors. Thus, the rotation matrix gets
6. GCFEM With Constant Normal

Figure 6.4: Starting from a position \( x^0 \) the deformation to \( x \) can be subdivided into the displacement \( u \) and the rotation \( R \).

\[
R_q = U_q^T V_q. \tag{6.11}
\]

For a displacement \( u(x^0) = x(x^0) - x^0 \), the term that transforms displacements into forces is \( Ku = Kx - Kx^0 \). We rotate \( x \) back to its initial orientation such that our linear model is sufficient. For a quadrature point \( q \) we get a new displacement

\[
u(x^0) = R_q^T x - x^0 \tag{6.12}
\]

that contains no rotation. Thus we can apply the stiffness matrix \( K_q \) and eliminate rotation-based errors. The force we get is just rotated back to \( x \)'s orientation. In other words, instead of the blocks \( K^q_{i,j} \), we assemble the rotated blocks that appear in

\[
R_q K^q_{i,j} (R_q^T x - x^0). \tag{6.13}
\]

The blocks \( R_q K^q_{i,j} R_q^T \) are contributed to the equation’s left-hand side and the forces \( R_q K^q_{i,j} x^0 \) are moved to the right-hand side. The system is no longer solved for the displacement. It is now solved for the positions \( x \).

Unfortunately we cannot assemble \( K \) in the precomputation phase and therefore we have to assemble \( R_q K^q_{i,j} \) and \( R_q K^q_{i,j} R_q^T \) in every iteration.

The algorithm described so far is sufficient for FEM systems that are based on vertex-based basis functions only. GCFEM uses another stiffness matrix \( H \) which transforms the normals into forces.

When a transformation \( D \) is applied to a plane its normal undergoes a transformation \( (D^T)^{-1} \)
in order to remain orthogonal to the surface. Because our transformation is a pure rotation $R$ with $R^T = R^{-1}$, $H$ can be assembled the same way as $K$.
Since $m_j$ is a constant, the assembled force $Hm$ stays on the right-hand side of the linear system.

A summary of the warped stiffness method described in this section is given in Algorithm 3.

**Algorithm 3 GCFEM algorithm with warped stiffness**

{Precomputation}

for all Quad. points $q$ do
  Compute matrix $K^q$
  Compute matrix $H^q$
  Evaluate Green coordinates $\phi_i(q)$ and $\psi_j(q)$
end for

for all Deformations from $x^0$ to $x$ do
  for all Quadrature points $q$ do
    Extract rotation $R_q$
    for all Block matrices $K_{i,j}^q$ do
      Compute $R_qK_{i,j}^qR_q^T$ and $R_qK_{i,j}^q$
    end for
    for all Block matrices $H_{i,j}^q$ do
      Compute $R_qH_{i,j}^qR_q^T$ and $R_qH_{i,j}^q$
    end for
  end for
  Assemble all rotated stiffness matrices
  Solve $Kx = f + Kx^0 - Hm$ for $x$
end for

Note that warped stiffness cannot be included implicitly into our system. The rotation matrix is always derived from the displacement Jacobian of the previous time step. Later we will turn our static system into a dynamic system and see that it is preferable to solve the system implicitly.

In order to improve the runtime performance we can summarize quadrature points to clusters. The weights of the quadratures are summed up to get the cluster’s weight and the cluster’s center point is the mean position of the quadrature points. Then, we only have to assemble as many matrices as we have clusters.

The method for compositing the clusters is listed in Algorithm 4.
This improvement helps to make slow simulations faster, but it introduces a new kind of artifacts and harms the quality of larger deformations. For small deformations it works well as Figure 6.5 shows.
6. **GCFEM With Constant Normal**

![Image of bar approximated with quadrature points grouped to clusters](image)

(a) One cluster  
(b) Two clusters  
(c) Four clusters  
(d) 16 clusters  
(e) 128 cluster  
(f) No clustering

**Figure 6.5:** The same example as Figure 6.3. The bar is approximated with 174 quadrature points which are grouped to clusters. For every clustering, the same gravity force is applied and the static displacement is depicted.

### 6.4. Boundary Constraints

Some elastic deformation applications need the possibility to fix an object in a constrained region. Besides that, it is also necessary to have some constraints in the static simulation. Without any constraints, there is no unique solution of (6.6).

The solution of the following derivation allows us to fix as many parts as required. An example with multiple constraints is depicted in Figure 6.6.

Given a region \( \tilde{\Omega} \subseteq \Omega \) that has a fixed displacement \( u_c \), we want to add a penalty function to our system. Since we try to minimize the energy \( E \), we add a penalty energy to \( E \) which enforces the constraint. We multiply the penalty energy with a parameter \( \beta \) to gain control over the system’s behavior when constraints are not fulfilled. The additional energy penalizes the squared difference of \( u_c \) and the real displacement \( u \):

\[
E_C = \frac{\beta}{2} \int_{u \in \tilde{\Omega}} \|u - u_c\|^2.
\]  

(6.14)

Again we consider just discretized deformations of the form \( u = \sum_i \phi_i u_i + \sum_j \psi_j m_j \). Of course we also have to discretize the integral, but this is explained later. The discretization yields
Algorithm 4 Clustering algorithm

Require: Quadrature points with positions \( q_1, q_2, \ldots, q_n \) and weights \( w_1, w_2, \ldots, w_n \), number of clusters \( m \leq n \)

Ensure: Clusters with centers \( c_1, c_2, \ldots, c_m \) and weights \( d_1, d_2, \ldots, d_m \)

1. Initialization
   for \( i = 1, 2, \ldots, n \) do
      \[ c_i = q_i \]
      \[ d_i = w_i \]
   end for

while Number of clusters greater \( m \) do
   for all Clusters \( x \) do
      Compute nearest neighbor \( N_x \) and distance \( D_x \)
   end for
   for Cluster \( x \) with smallest \( D_x \) do
      \{Melt cluster \( x \) and \( N_x \}\}
      \[ d_x = d_x + d_{N_x} \]
      \[ c_x = (c_x + c_{N_x})/2 \]
      delete \( N_x \)
   end for
end while

Figure 6.6: A bar that has two different fixed parts (green). The nodes in these parts have a constraint displacement. All other nodes are deformed solely by gravity force.

\[
\mathcal{E}_C = \frac{\beta}{2} \int_{u \in \Omega} \left( \sum_j \phi_j u_j + \sum_j \psi_j m_j - u_c \right)^T \left( \sum_j \phi_j u_j + \sum_j \psi_j m_j - u_c \right) \, du. \tag{6.15}
\]

Our energies are minimized to get the steady state. To minimize \( \mathcal{E}_C \) we need to know its derivative with respect to a discrete displacement which is:

\[
\frac{\delta \mathcal{E}_C}{\delta u_i} = \beta \int_{u \in \Omega} (u - u_c)^T \left( \frac{\delta}{\delta u_i} (u - u_c) \right). \tag{6.16}
\]

Since \( u_c \) is a constant, its derivative is zero and only the derivative of \( u \) remains. The discretized displacement contains the cage positions and normals. We defined the normals to be constant as well, so the only non-zero derivative is \( \delta \sum_j \phi_j u_j / \delta u_i = \phi_i \) and we get
6. GCFEM With Constant Normal

\[ \frac{\delta E_C}{\delta u_i} = \beta \int_{\tilde{\Omega}} \phi_i (\sum_j \phi_j u_j + \sum_j \psi_j m_j - u_c). \] (6.17)

The equation above shows that when we add the penalty force, several new terms raise in our system. The new terms can be added to the already existing matrices and vectors \( K, H \) and \( f \). The additional penalty matrices are defined as

\[
K_{i,j}^P = I \cdot \beta \int_{\tilde{\Omega}} \phi_i \phi_j \\
H_{i,j}^P = I \cdot \beta \int_{\tilde{\Omega}} \phi_i \psi_j \\
f_i^P = \beta \left( \int_{\tilde{\Omega}} \phi_i u_c \right)
\] (6.18)

and they allow us to rewrite Equation (6.17) as

\[ \frac{\delta E_C}{\delta u_i} = \sum_j K_{i,j}^P u_j + \sum_j H_{i,j}^P m_j - f_i^P. \] (6.19)

We add the derivative of the penalty energy to the other energies and get a new system of linear equations

\[ (K + K^P) u = f + f^P - (H + H^P) m. \] (6.20)

Therefore the GCFEM system is modified by adding the penalty matrices \( K^P \) and \( H^P \) and the penalty vector \( f^P \) to their corresponding stiffness matrix or force vector. The penalty terms can be precomputed if the fixed region \( \tilde{\Omega} \) does not change.

The last open question of this section is how we compute an integral over \( \tilde{\Omega} \). Section 3.6 describes that an integral can be approximated by a sum over quadrature points that approximate the volume.

Our application should be able to fix the vertices of the surface mesh. Hence, given a set of fixed vertices \( \Omega' \), the integral of the squared errors is

\[ \int_{u \in \Omega} \| u - u_c \|^2 = \sum_{m_i \in \Omega'} \| u(m_i) - u_c(m_i) \|^2 \] (6.21)

where \( m_i \) denotes a vertex on the surface mesh.

The method described in this section was also applied to other FEM systems. In the case of GC, the term \( H^P \) arises which can be ignored for system that are based on simple vertex-based basis functions only.
6.5. Collisions

Collision detection and handling are important aspects of dynamic simulations. Since we will convert our static system into a dynamic system, we want to include collisions. For simplicity we consider only the collision with a plane.

A plane is given by an anchor point \(a\) and a normal \(n\). A point \(x\) has collided with the plane if the penetration depth \(d = (x - a)^T n\) is negative. If so, we add a penalty force proportional to \(d\). The force shall be perpendicular to the plane, i.e. the penalty force has the same direction as \(n\). Furthermore we multiply the force with a penalty parameter \(\alpha\).

Figure 6.7 shows a plane and two points. The point with the negative penetration depth has collided with the plane. A force \(f\) is applied to the collided point.

The described force can be computed as

\[
f = -\alpha n \cdot d = -\alpha n \cdot (x - a)^T n. \tag{6.22}
\]

Note that the minus sign in the front is necessary due to the negativity of \(d\).

Next, we discretize the point’s position. A stable dynamic system should be based on an implicit handle of collisions. Thus, we split up the cage positions into the initial position and the displacement. In other words, we replace \(x_i\) by \(x_i^0 + u_i\) and the penalty force gets

\[
- \alpha n \cdot \left( \sum_j \phi_j u_j + \sum_j \phi_j x_j^0 + \sum_j \psi_j n s_j - a \right)^T n. \tag{6.23}
\]

Remember how an external force is added to the system. The nodal force \(f_i\) is the negative integral over the domain of the force field times \(\phi_i\). Since we only have one vertex with a penalty force, the integral disappears and we get the nodal penalty force.
6. GCFEM With Constant Normal

\[
\alpha \phi_i \mathbf{n} \cdot \left( \sum_j \phi_j \mathbf{u}_j + \sum_j \phi_j x^0_j + \sum_j \psi_j \mathbf{n}_j - \mathbf{a} \right)^T \mathbf{n}
\]  
(6.24)

for the \(i\)th cage node, which can be rewritten as

\[
\sum_j \alpha \phi_i \phi_j \mathbf{n}^T \mathbf{u}_j - \beta \phi_i \mathbf{n} \cdot \left( \mathbf{a} - \sum_j \phi_j x^0_j - \sum_j \psi_j \mathbf{n}_j \right)^T \mathbf{n}
\]  
(6.25)

and we get a matrix \(K^P\) consisting of \(3 \times 3\) block matrices and a penalty vector \(f^P\) consisting of the entries \(f^P_i\). The matrix \(K^P\) is multiplied with the displacements and therefore can be added to the stiffness matrix \(K\). \(f^P\) is moved to the right side of the linear system and added to the force vector \(f\). Our new system of linear equations finally is

\[
(K + K^P) \mathbf{u} = \mathbf{f} + f^P - \mathbf{Hm}.
\]  
(6.26)

with

\[
K^P_{i,j} = \alpha \phi_i \phi_j \mathbf{n}^T
\]  
(6.27)

and

\[
f^P_i = \alpha \phi_i \mathbf{n} \cdot \left( \mathbf{a} - \sum_j \phi_j x^0_j - \sum_j \psi_j \mathbf{n}_j \right)^T \mathbf{n}.
\]  
(6.28)

The collision penalty method described in this chapter is not new. It was already implemented for FEM systems that are based on simple vertex-based basis functions. The algorithm described here is identical except for the penalty force \(f^P\), whose sum \(\sum_j \psi_j \mathbf{n}_j\) has to be omitted.

6.6. Dynamic System

6.6.1. Newton’s law

In order to get a dynamic system, a velocity \(\mathbf{v} = \dot{\mathbf{u}}\) and an acceleration \(\ddot{\mathbf{v}} = \ddot{\mathbf{u}}\) are introduced. Newton’s second law states

\[
f = m \ddot{x}.
\]  
(6.29)

Since our system is based on displacement, we replace \(\ddot{x}\) by \(\ddot{\mathbf{u}}\). To get the acceleration of the whole object, we integrate the point accelerations over the domain \(\Omega\). The weight of a point is
6.6. Dynamic System

Newton’s law shows us the relation between force and acceleration. Our system is based on an energy $E$. When we introduced the energy in Section 3.4, we said that we want to minimize it to find the steady state. For the static case, this means that we find the solution of $\frac{\delta E}{\delta u} = 0$. But for the dynamic case, we follow the force $-\frac{\delta E}{\delta u}$ that points towards the minimum of the energy function. Thus, the force vector in Newton’s law (6.29) can be replaced by the negative derivative of the total energy:

$$- \frac{\delta E}{\delta u} = \int_{\Omega} m \ddot{u}. \quad (6.30)$$

We already know how to discretize $E$. What remains is the discretization of the equation’s right-hand side. Section 3.6 shows that an integral over the domain can be approximated by a sum over quadrature points. Furthermore, the acceleration $\ddot{u}$ can be discretized using GC:

$$\ddot{u} = \sum_j \phi_j \ddot{u}_j. \quad (6.31)$$

The whole force in the domain that is caused by the acceleration is $\int_{\Omega} \rho \sum_j \phi_j \ddot{u}_j$, but we are only interested in forces on the discrete nodes. According to Section 6.2, the acceleration force at the $i$th node gets

$$f_i = (\phi_i, \rho \sum_j \phi_j \ddot{u}_j)_{\Omega} = \sum_j (\phi_i, \rho \phi_j)_{\Omega} \ddot{u}_j. \quad (6.32)$$

The new block matrices $M_{i,j}$ build the mass matrix $M$. In the first section of this chapter we showed that $\frac{\delta E}{\delta u_i} = (Ku + Hm - f)_i$. We add the acceleration force to get the system of ordinary differential equations

$$M \ddot{u} + Ku = f - Hm. \quad (6.33)$$

The introduced matrix $M$ can be precomputed. After a short section about damping, we will show how (6.33) can be solved for each time step.

6.6.2. Damping

For explicit integration over time, there is a need for damping because the system would gain energy for too large time steps and could stay stable.

We add the damping term $Cu = Cv$ with the damping matrix $C$ to our system. In our implementation, the damping matrix is a diagonal matrix with diagonal entries $c$. $c$ is a damping factor that can be set to zero if no damping is needed. The whole linear system now looks as follows:
6. GCFEM With Constant Normal

\[ M \ddot{\mathbf{u}} + C \dot{\mathbf{u}} + \mathbf{Ku} = f - \mathbf{Hm}. \]  

(6.34)

### 6.6.3. Leap Frog Step

In order to get a dynamic system, we have to solve (6.33) every time step. The size of the time step \( dt \) determines the speed of the simulation but can also affect the stability of the system. When we use leap frog in every time iteration, the velocity is integrated explicitly over time. This means that the update is done with the acceleration of the last time step:

\[ \mathbf{v}_{t+1} = \mathbf{v}_t + dt \cdot \dot{\mathbf{v}}_t. \]  

(6.35)

The cage position is integrated implicitly over time.

Explicit velocity integration is easier to implement than implicit integration but requires enough damping or small time steps to be stable ([Poz98]). However, too much damping can make the system too inert and small time steps lead to slow animations.

Given a step size \( dt \), the integration over time for the velocity vector \( \mathbf{v} \) and the position vector \( \mathbf{x} \) can be computed as

\[ \mathbf{v}_{t+1} = \mathbf{v}_t + dt \dot{\mathbf{v}}_t = \mathbf{v}_t + dt \mathbf{M}^{-1} \left( f - \mathbf{Hm}_t - \mathbf{Ku}_t - \mathbf{Cv}_t \right) \]

\[ \mathbf{x}_{t+1} = \mathbf{x}_t + dt \mathbf{v}_{t+1}. \]  

(6.36)

### 6.6.4. Implicit Euler Step

Instead of integrating the velocity explicitly over time, it can also be integrated implicitly. We update the velocity in every time step by adding the acceleration of the new time step:

\[ \mathbf{v}_{t+1} = \mathbf{v}_t + dt \cdot \dot{\mathbf{v}}_{t+1}. \]  

(6.37)

The element-based FEM is stable under implicit integration ([Poz98], [BW98], [MG04]). GCFEM also runs more robust with implicit integration, but the update steps are a little slower since the acceleration of the new time step contains terms that have to be moved to the left-hand side of the equation.

The new update rule for the cage velocity is

\[ \mathbf{v}_{t+1} = \mathbf{v}_t + dt \cdot \dot{\mathbf{v}}_{t+1} = \mathbf{v}_t + dt \mathbf{M}^{-1} \left( f - \mathbf{Hm}_t - \mathbf{Cv}_{t+1} - \mathbf{Ku}^t \right) \]

\[ = \mathbf{x}_{t+1} - \mathbf{x}_0 = \mathbf{x}_t + dt \cdot \mathbf{v}^t - \mathbf{x}_0 \]  

(6.38)
6.6. Dynamic System

The equation above is solved for $v^{t+1}$:

$$(M + dtC + dt^2K) v^{t+1} = Mv^t + dt \left( f - Hm^t - Kx^t + Kx^0 \right). \quad (6.39)$$

The integration step for the cage positions is the same as for the leap frog integration (see Equation (6.36)).

Figure 6.8 shows some frames of an animation that was simulated with GCFEM based on implicit euler steps.

![Dynamic GCFEM](image1)
(a) Dynamic GCFEM

![Dynamic GCFEM](image2)
(b) Dynamic GCFEM

![Dynamic GCFEM](image3)
(c) Dynamic GCFEM

![Static GCFEM](image4)
(d) Static GCFEM

**Figure 6.8:** The goblin model deformed with GCFEM. The first three pictures show frames of a dynamic GCFEM animation. The left foot of the goblin is fixed, the other are parts were deformed by applying some forces manually. The fourth picture shows a static GCFEM example, where several parts of the body are constraint and no other force is applied.
6. GCFEM With Constant Normal
GC Fitting

In this chapter we introduce an alternative deformation technique that is also based on Green coordinates (GC). Compared to GCFEM that is discussed in the previous chapters, it has a complete different approach. The idea of GC fitting is that an arbitrary deformation is transformed to a deformation that can be represented with GC.

This technique is not the main issue of the thesis. We derived GC fitting when we were looking for an other approach since GCFEM did not show the results we hoped for.

GC fitting can be done on any underlying simulation. We use shape matching as in [MHTG05] to simulate deformations, but we could also use FEM or any other technique.

First, the GC fitting algorithm is described. Then, the single steps are elucidated where the focus lies clearly on the fitting step.

### 7.1. Introduction

Before the simulation can start, so-called matching points are placed within the object’s surface mesh. They have to approximate the object’s volume and have to be distributed over all parts of the object. The Green coordinates $\phi_i$ and $\psi_j$ are precomputed for every matching point such that a matching point’s position can be interpolated by a given cage.

Hence a matching point has the position

$$p_i = \sum_j \phi_j(p_i^0)x_j + \sum_j \psi_j(p_i)ns_j.$$  \hfill (7.1)

During the runtime, the algorithm is divided into two parts. First, an arbitrary technique simulates a deformation of a solid. This dynamic system is completely independent of Green
7. **GC Fitting**

coordinates. The only references between GC fitting and the dynamic system are the matching points which are transposed to goal positions by the simulation.

The second step is the actual GC fitting. GC fitting tries to move the cage such that the matching points interpolated by GC lie on their goal positions. By fitting the matching points on the goal positions ensures that the GC deformation has the same shape as the original deformation. The deformed cage can be used to represent the surface mesh’s deformation.

**Algorithm 5 GC fitting**

- **{Precomputation}**
  - Place matching points within the object
  - Compute GC for all matching points and object nodes

- while is running do
  - Simulate a deformation on the matching points and compute the goal positions
  - Cage position estimated by fitting matching points on goal positions
  - Object nodes’ positions are computed with GC

- end while

The pipeline of GC fitting is summarized in Algorithm 5 and illustrated in Figure 7.1.

### 7.2. Matching Point Sampling

The matching points can be chosen in the same manner as the quadrature points in GCFEM (Section 3.6).

A good sampling of the object volume is crucial for GC fitting. Whenever a part of an object is not covered by a matching point, this part will be represented in a random manner even though GC fitting could match the matching points and the goal positions perfectly. Even though additional matching points do not restrict the performance speed as much as for GCFEM, they still restrict it and therefore the number of matching points should not be too large. As discussed in Section 3.6, a more sophisticated sampling method than the uniform method reduces the required number of matching points to cover the object.

Figure 7.2 compares the uniform sampling and the furthest distance sampling.

As an example of furthest distance sample, consider the dinosaur in Figure 7.3 that is approximated with 80 matching points. The 80 points are nodes of a grid with resolution 12 that is placed around the dinosaur. The grid has \((12 + 1)^3 = 2197\) nodes, but only the points with maximal distance to each other are chosen.
7.3. GC Fitting

7.3.1. Error Function

We compute the position of the cage vertices by minimizing the error

\[ E(x) = \sum_i \left\| g_i - \sum_j \phi_j(p_i)x_j - \sum_j \psi_j(p_i)ns_j(x) \right\|^2 \]  \hspace{1cm} (7.2)

where \( g_i \) is the goal position of the \( i \)th matching point \( p_i \) whose initial position equals to the goal position. \( x_j \) is the position of the \( j \)th cage node and \( ns_j \) the normal times scaling factor of the triangle \( t_j \). We are looking for the cage node positions \( x_j \) that minimize the squared distance.
7. GC Fitting

(a) Target object  (b) Uniform sampling  (c) Furthest distance

Figure 7.2: A dinosaur object that is sampled. Once the sampling is uniform and once the furthest distance algorithm with grid resolution 10 is used. Both samplings give 43 matching points. Even though the uniform sampling is much faster than the furthest distance algorithm (1141 ms and 6094 ms), the uniform sampling is not usable for such little number of matching points.

between the matching points and their goal positions. Since \( \text{ns}_j \) is a non-linear function of the cage nodes, the whole minimization problem is non-linear and not analytically solvable.

The Equation (7.2) is solved by searching the solution iteratively. Starting at the cage position of the last time step, the cage nodes are shifted to the direction where \( E \) becomes smaller. There are several iterative optimization methods that fit to our problem. They are discussed in Section 7.3.3.

7.3.2. Minimizing Edge Lengths

For large deformations, the error function is not minimized correctly since there are several local minima with almost the same error. We the iterations stop in a wrong local minimum, the cage can have a strange shape and the minimization method is not capable to find a way out of these undesired local minimum in the next time steps. Thus, we want to reduce the amount of local minima by adding some penalty terms to (7.2). In order to keep the cage’s overall shape, we introduce a term that should penalize the change of edge lengths. Other geometrical penalty terms are possible, but our approach has shown to work well.

We also introduce the penalty variable \( \gamma \). \( \gamma \) is the fraction of how much the edge lengths are penalized compared to the matching point error. \( \gamma \) is zero if no edge length preservation is required. Otherwise \( \gamma \) has to be smaller than 1. If it was 1, there would be an infinitive amount of solutions since the only requirement is the preservation of the edge lengths.

Then, the new error function becomes

\[
E(x) = \frac{1 - \gamma}{P} \sum_{i}^{P} \left\| g_i - \sum_{j} \phi_j(p_i)x_j - \sum_{j} \psi_j(p_i)\text{ns}_j(x) \right\|^2 + \frac{\gamma}{E} \sum_{i}^{E} (e_{i,0} - e_i(x))^2 \quad (7.3)
\]

where \( P \) is the amount of matching points and \( E \) the amount of cage edges. For an edge \( i \), \( e_{i,0} \) denotes the initial edge length and \( e_i(x) \) the new edge length that is dependent on the solution.

54
7.3. GC Fitting

Figure 7.3: This is an example of GC fitting. A dinosaur object (green) with a cage (black) and goal positions (red) is depicted. First, the goal positions are transformed. Then, the cage is fitted to the goal positions. Finally, the dinosaur undertakes the same deformation as the cage did.

Figure 7.4 shows GC fitting with and without edge length preservation ($\gamma = 0.4$). The result with edge length preservation is clearly better. The cage has still a good shape. When the dynamic simulation continues, the cage’s shape is crucial and decides whether the next GC fitting step fails or not since it will be the initial solution for the new search. Taking a strange cage as starting point will always lead to a bad local minimum.

Even though length preservation is an important factor for our GC fitting, it should not have a big influence on the solution. If $\gamma$ is chosen too large, GC fitting restricts a deformation in an unpleasant manner. The cage is not able to deform as much as the goal positions do. Thus, the object shows a deformation that is less significant as it should be. An example of this behavior is shown in Figure 7.5.

7.3.3. Minimization Methods

In order the get a least squares problem, we rewrite the minimization function as
7. GC Fitting

(a) Deformed goal positions
(b) Only matching point fitting
(c) With edge length preservation

Figure 7.4: Here, a large deformation is made. GC fitting with only minimizing the matching point errors results in a strange cage that will lead to a minimization failure in the next time step. Edge length preservation helps to get a better cage that is nearer to the surface mesh’s shape.

\[
E = \sum_i^3 P_i^2 + \sum_i^E s_i^2
\]
\[
r_{3i+j} = \sqrt{\frac{1-\gamma}{3P}} (g_i - \sum_k \phi_k(p_i)x_j - \sum_k \psi_k(p_i)\text{ns}_k(x))_j
\]
\[
s_i = \sqrt{E}(e_{i,0} - e_i(x)).
\]

Notice that we split \(\|g_i - p_i(x)\|^2\) into three squared differences. This way we can rewrite the problem in the usual form \(\sum_i (c_i - f_i(x))^2\).

Since this is a non-linear least squares problem, we have to search for the minimum iteratively.

Steepest Descent

A non-linear least squares problem is normally solved iteratively. The steepest descent method simply moves the solution vector \(x\) into the direction of \(-\nabla E(x)\). We implemented this method. \(\nabla E(x)\) contains the derivatives of the error function with respect to the cage positions. The computation of \(\nabla \text{ns}_j(x_i)\) can be reviewed in Appendix B.1. The gradient of the error function with respect to a cage position \(x_i\) is
7.3. GC Fitting

(a) Undeformed cage  
(b) $\gamma = 0.2$  
(c) $\gamma = 0.8$

Figure 7.5: The dinosaur object’s cage tries to fit the matching points to the goal positions. Once, GC fitting is done with an edge preservation value $\gamma = 0.2$ and once it is done with $\gamma = 0.8$.

\[
\nabla E(x) = 2\frac{1 - \gamma}{P} \sum_{j=1}^{P} \left[ \phi_i(p_j)(\phi'(p_j) - g_j + \psi'(p_j)) + \sum_k \nabla n_{sk}(x_i)\psi_k(p_j)^T(\phi'(p_j) - g_j + \psi'(p_j)) \right] + 2\frac{\gamma}{E} \sum_{j=1}^{E} \nabla e_j(x)(e_j(x) - e_{j,0}) \tag{7.5}
\]

where $\phi'(p_j) = \sum_k \phi_k(p_j)x_k$ and $\psi'(p_j) = \sum_k \psi_k(p_j)n_{sk}$.

The cage nodes are updated after every computation of $\nabla E(x)$ with

\[
x_i^{t+1} = x_i^t - s \cdot \nabla E(x_i^t) \tag{7.6}
\]

where $s$ is the step size.

We define $s$ each time step by adaptive step size control. We shortly describe this adaptive algorithm.

We start with $s_{\text{min}} = 0$ and $s_{\text{max}} = 1$. The new cage position is computed with both the step sizes. If $s_{\text{min}}$ minimizes $E$ better, $s_{\text{max}}$ is set to $(s_{\text{min}} + s_{\text{max}})/2$. If $s_{\text{max}}$ minimizes $E$ better, $s_{\text{min}}$ is replaced. This step can be repeated until $s_{\text{max}} - s_{\text{min}}$ falls below a predefined threshold. Then, the step size that minimizes $E$ more is applied to (7.6). The threshold should not be chosen too small since too many iterations cost too much. Even though this algorithm needs some time to find the best step size, it is still much faster than recomputing the gradient several times instead.
7. GC Fitting

The cage nodes are updated until $\|x^{t+1} - x^t\|$ falls below a given threshold.

The steepest descent method is a first-order optimization algorithm. It converges slowly and needs more steps to find a local minimum than more sophisticated algorithms. Especially our error function is of different curvature in different directions. Thus, steepest descent is not an appropriate method.

**Gauss-Newton and Levenberg-Marquardt**

A more sophisticated method than the steepest descent method is the Gauss-Newton method [Bjo97]. For starting cages that lie not too far away from the solution, it works quite well for our error function.

The derivation of the Gauss-Newton method is based on the first Taylor approximation of the residuals. Gauss-Newton can be seen as a modification of the Newton’s method. But unlike Newton’s method, the second derivatives are not necessary (which would be quite challenging in our case).

But we have to compute the Jacobian of the residuals $J$. $J_{i,j}$ contains the derivative of $r_i$ or $s_{i-3P}$ with respect to $x_j$.

The update rule gets

$$x_{i+1} = x_i + s\delta x \quad (7.7)$$

where $\delta x$ is the solution of the normal equation

$$(J^TJ)\delta x = -Jr. \quad (7.8)$$

$r$ contains the residuals $r_i$ and $s_i$. Thus, it is a vector of size $3P + E$. $s$ is the step size and can be optimized as described in the last section.

The Gauss-Newton method requires less steps than the steepest descent method to find the minimum. Ideally, its rate of convergence is quadratic. Anyhow, a good start cage is necessary to find the right minimum. This implies that the time step of our dynamic simulation may not be too large.

Gauss-Newton seems to be sufficient for our problem since we have sufficient small steps such that the initial cage positions are not far away from the minimum error. For larger time steps, a more robust method should be implemented. An optimization algorithm that is widely used in computer graphics is the Levenberg-Marquardt method [PEG81]. It interpolates between the Gauss-Newton method and the steepest descent method. It is also iterative and each iteration, a step is done in the direction of $\delta x$ which is defined as the solution of
(7.3) GC Fitting

\[
(J^T J + \lambda \text{diag}(J^T J)) \delta \mathbf{x} = J^T \mathbf{r}.
\]

Here, \( J_{ij} \) contains the derivatives of \( p_i(x) \) or \( e_{i-3P}(x) \) with respect to \( x_j \) instead of the residuals’ derivatives. \( \lambda \) is the damping factor. The larger it is, the more behavior like the steepest descent algorithm we get. The damping factor is chosen similarly to the step size for the other two methods.

### 7.3.4. Improvement

Normally we cannot find a solution where the error function \( E \) is zero. The goal positions are always deformed in a manner that cannot be reconstructed exactly by a GC deformation. Hence, the positions of the matching points that are computed by GC are normally not exactly on the goal position. In every time step, the goal positions move further away from their corresponding matching points and the cage is not able to correct the matching points’ positions. These errors are accumulated with every new time step until GC fitting fails.

This accumulation of the error can be prevented by moving the goal positions towards their matching points after every time step. So, the points adapt their positions to the representation ability of the cage. This adaption method can be regarded as a kind of artificial assurance for stability.

We introduce a new variable \( \delta \) that is the adaption ratio:

\[
 g_{i}^{t+1} = (1 - \delta) g_{i}^{t} + \delta p_{i}^{t}.
\]

\( \delta \) has to be between zero and one. Zero means no adaption at all, one is total adaption.

Since a good solution for the GC fitting problem takes some time, it makes sense to do the GC fitting in a post processing step. First, the point cloud is deformed in real time. Then, for every recorded time step, GC fitting can be applied automatically to get the same transformations, but projected into a quasi-conformal deformation. For this, the adaption improvement described above is not applicable.
7. GC Fitting
Results

In this chapter we summarize the results of the introduced methods. We focus on the functionality and stability of the different approaches but also on the runtime and other measurements.

We subdivide the results into three sections. The first deals with GCFEM and is subdivided into static GCFEM and dynamic GCFEM. The static and the dynamic case are considered separately due to the different simulation loop. Furthermore, stability is more important for the dynamic simulation than for the static case. On the other hand, a static simulation has to converge what can be regarded as the simulation’s stability. The second section will contain the results of GC fitting. The last section is kept very short and contains the results of the independent voxel-based FEM that uses GC only for the surface mesh’s representation.

Our test simulations run on a machine with a Quad Core CPU (2.40 GHz) and 3 GB of RAM. The rendering was done on a NVIDIA GeForce 8800 GTX.

8.1. GCFEM

8.1.1. Precomputation

The precomputation step is almost the same for the static and the dynamic case. The computation of the mass matrix is not required for a static deformation, but the remaining matrices and values have to be computed for both cases. The computation of the mass matrix is free
8. Results

![Figure 8.1: Five objects and their cages that are used for testing GCFEM.](image)

<table>
<thead>
<tr>
<th>Object</th>
<th>Surface vtx/tr</th>
<th>Cage vtx/tr</th>
<th>Quad. points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube</td>
<td>110/216</td>
<td>8/12</td>
<td>22</td>
</tr>
<tr>
<td>Short bar</td>
<td>308/612</td>
<td>16/28</td>
<td>60</td>
</tr>
<tr>
<td>Long bar</td>
<td>704/1404</td>
<td>16/28</td>
<td>137</td>
</tr>
<tr>
<td>Goblin</td>
<td>5411/10818</td>
<td>56/108</td>
<td>80</td>
</tr>
<tr>
<td>Dinosaur</td>
<td>2550/5096</td>
<td>59/114</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 8.1: Some properties of the test objects (see Figure 8.1). The second and third column indicate the number of vertices and triangles of the surface and cage mesh respectively.

compared to the setup of the stiffness matrices and therefore the precomputation time for the dynamic and the static case is almost identic.

In the following section we will consider some objects that we used for test deformations. They are depicted in Figure 8.1 and their properties are listed in Table 8.1. The corresponding precomputation costs are given in Table 8.2. Note that the quadrature point sampling costs much more for the goblin than for the dinosaur even though they have both 80 quadrature points. This is due to the greater number of surface mesh vertices and therefore leads to more expensive tests to see whether a point lies inside or outside of a surface mesh.

The most time-consuming precomputation step is clearly the setup of the stiffness matrices which is dependent on the amount of quadrature points and on the complexity of the cage. The quadrature point sampling and the GC computation are primarily expensive when the surface mesh contains many triangles.

For FEM systems that use vertex-based basis functions only, the stiffness matrix $K$ shows some properties that are required to get a stable system. $K$ is symmetric and positive semi-definite. Hence, the eigenvalues of the stiffness matrix are not negative. Six eigenvalues are zero due to
8.1. GCFEM

<table>
<thead>
<tr>
<th>Object</th>
<th>Precomputation (in ms)</th>
<th>Runtime (in ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sampling</td>
<td>GC</td>
</tr>
<tr>
<td>Cube</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Short bar</td>
<td>32</td>
<td>140</td>
</tr>
<tr>
<td>Long bar</td>
<td>157</td>
<td>328</td>
</tr>
<tr>
<td>Goblin</td>
<td>25732</td>
<td>4765</td>
</tr>
<tr>
<td>Dinosaur</td>
<td>6123</td>
<td>2704</td>
</tr>
</tbody>
</table>

Table 8.2: The performance speed of the tests described in Table 8.1. The precomputation time is subdivided into the volume sampling, the GC computation, the stiffness matrix setup and the boundary constraint matrix setup. The runtime performance is subdivided into the framerate and the time required for stiffness matrix assembly and for the surface mesh’s GC representation. The time integration and the linear solver’s costs is dependent on the deformation but are always near to 0ms for all our examples.

The properties of the basis functions. The first three eigenvalues are zero because pure translation results in no force. The next three eigenvalues are zero because a rotation of \( u \) should result in no force as well. Or with other words, three eigenvalues are zero due to partition of unity and three eigenvalues are zero due to linear reproduction of the basis functions.

The GCFEM stiffness matrix \( K \) shows the same properties but contains only three zero eigenvalues because only \( \phi_i \)'s are contained in \( K \), but linear reproduction is valid only for the \( \phi_i \)'s and \( \psi_j \)'s together. Hence, there are only three eigenvalues zero due to \( \phi_i \)'s partition of unity. Anyhow, a rotation can still be reproduced because we update the normals and \( Hm \) each time step.

The stiffness matrix \( K \) of the GCFEM system is positive semi-definite as well. This is necessary because an energy function always has to be convex. Negative eigenvalues would induce saddle points and therefore no single minimum. In a physical system, a steady point must exist, otherwise the system would be unstable and the energy would decrease towards negative infinity.

8.1.2. Static GCFEM

Convergence

We described the static loop in Chapter 6 (Algorithm 2). In every iteration, the system \( Ku = f - Hm \) is solved for \( u \) and afterwards the normals and scaling factors are updated. The loop ends when the solution vector \( u \) converges. When a system converges in the static case, it is normally stable in the dynamic case as well.

The test simulations we have done always converged when we chose the parameters and the quadrature points in a reasonable way. The results of a static deformation look nice and we also get the shape-preserving and quasi-conformal representation we hoped for. Furthermore, stiffness warping and boundary constraints can be applied to our system without harming the
8. Results

Figure 8.2: The dinosaur object with quadrature points and a cage. The tail gets deformed with static GCFEM.

result.

Figure 8.3 and 8.4 show an example of a convergence and how the deformation on the bar changes with every iteration. The deformation changes from iteration to iteration due to the update of $H_m$, which leads to better preservation of the angles as the measurement in Figure 8.4 shows.

Nevertheless, we cannot show that $H_m$ always leads to a better preservation of the angles. But $H_m$ is necessary since it contains the $\psi_j$ coordinates which are required to satisfy the GC properties except for the partition of unity. Hence, this correction force is needed even though it is often small compared to $K_u$ and its meaning is not intuitive.

Quadrature Points

For more complex objects and cages as the goblin and the dinosaur example, the amount of required quadrature points increases when we want to perform a large deformation. We get a better result when we also sample the volume between the cage and the surface mesh. We assume that this extended sampling improves the simulation because a cage that lies far away from the domain leads to small $\phi_i$’s and $\psi_j$’s and therefore to numerical problems. For complex objects, it is very difficult to design a cage that lies near to the surface mesh since we want to keep the number of cage vertices low.

Figure 8.2 shows a small static deformation on the dinosaur object. The dinosaur’s cage is not well designed because there is too much space between the cage and the surface mesh. We still can get convergence for every deformation when we sample the cage’s volume instead of the dinosaur’s volume.

An important factor is the number of quadrature points and how they are placed. We sample the goblin object with 80 quadrature points. If we cut the number of quadrature points in half,
Figure 8.3: The two ends of the long bar are fixed and transformed. 36 static loops are done where \( m \) is updated. The two pictures show the bar after the first and the last loop and therefore the effect of the correction force \( H_m \) can be seen. The red lines help to visualize the variance of the thickness. The correction force compresses the part in the middle of the bar and minimizes the squared angle differences of the mesh’s triangles by 14%. The convergence is plotted in Figure 8.4.

a small displacement cannot achieve convergence. Figure 8.5 shows an example of a small deformation with different samplings.

Timing Information

The computation speed of one iteration is similar to the framerate of the dynamic simulation listed in 8.2. The most expensive step of a loop is the assembly of the stiffness matrices. Normally, the assembly takes around 90% of the whole computation time. Since \( H \) is not symmetric and the amount of triangles is almost twice the amount of nodes, the assembly of \( H \) takes about four times as long as the assembly of \( K \). Furthermore, the stiffness matrices are not sparse since all basis functions interact with each other. Without stiffness warping, no assembly would be required and the simulation would be much faster. Since each quadrature point has to be rotated and added to the assembly solely, the amount of quadrature points has a big influence on the simulation speed.

Comparison With Element-based FEM

Figure 8.6 shows a comparison of our GCFEM system and a FEM system based on hexahedral elements. Our FEM-like system actually requires only one element, namely the cage. The cage can be designed such that an object’s detail can be separated from the others and can therefore be deformed independently. Our system represents a deformation in a nicer manner due to the distortion minimization. Furthermore, GCFEM requires less degrees of freedom to get deformations whose discretization is not visually obvious. Figure 8.6 shows clearly that even though the other FEM system contains more degrees of freedom, it cannot represent the deformation as well as GCFEM does. With a higher resolution, the other FEM system could represent the deformation better, but it cannot guarantee shape preservation.

On the other hand, without the second stiffness matrix \( H \), GCFEM is slower than FEM systems.
8. Results

Another comparison is illustrated in Figure 8.7. The figure shows two disadvantages of classical FEM. First, the legs do not deform independently even though the grid resolution is quite high. Secondly the representation of the leg is clearly better for GCFEM since the distortion is distributed uniformly while the FEM leg is buckled at one point.

8.1.3. Dynamic GCFEM

Stability

Similar to the convergence of the static loop, the dynamic simulation is stable even for large deformations. Simpler test objects like the bars and the cube can be run with only a few quadrature points. Furthermore, they are stable under large deformations, even when constraints and collision handling are introduced. Figure 8.8 shows an example of a dynamic GCFEM animation.

Larger systems can get unstable when the amount of quadrature points is too small or when the sampling is too far away from the cage. Then, the parts of the object that are sampled the worst begin to degenerate for large deformations.

For instance, when there are little quadrature points in the goblin model, already small deformations can lead to strange degenerations of the goblin’s hand. These degenerations can end up in a complete unstable system since the energy starts to grow continuously. The same is true for the dinosaur object. There, the dinosaur’s hand begins to degenerate quite quickly. The tail, on the other hand, can undertake quite big deformations and the system stays stable. Enhancing the sampling over the object does not always help. Anyhow, the extension from the object sampling to the sampling within the whole cage makes the system much more stable. In the tested cases, we were always able to get a stable system.

We didn’t have time to investigate why the extended sampling works better but we expect numerical problems with the coordinates when the cage is too far away.

As for the static model, warped stiffness works well for the dynamic case too. Boundary constraints as defined in this documentation also show good results. For the more complex objects, the penalty parameter should not be too high in order to avoid a degeneration of the system. On the other hand, too small parameters lead to small errors, i.e. the constraints are not always held exact. Anyhow, a tradeoff always leads to a well-working and stable boundary constraint. The same is valid for the collision handling where the parameter has to be kept small but not too small in order to prevent a touch of the object and the plane.

Performance Speed

The framerates of our test simulations are listed in Table 8.2. The presented framerates include the whole simulation, rendering included.

The most time-consuming computation is the assembly of the stiffness matrices whose costs depend on the amount of cage triangles. The representation of the surface mesh using GC costs much less and its costs depends on the surface mesh’s complexity.
8.2. GC Fitting

Unfortunately, the framerate is lower than for other FEM systems due to the expensive assembly of the stiffness matrix $H$. Although simple objects like the cube and the bars run easily in real-time, real-time simulation for more complex models like the dinosaur and golbin example is impossible. The performance of GCFEM can be accelerated by clustering the quadrature points or by reducing their number. Anyhow, the performance speed is still far away from real-time for complex simulations and is probably the greatest drawback of GCFEM.

Loss of Rotation

Even though GCFEM has been shown to be stable and to be able to perform shape-preserving and nice large deformations, the approximation of the normal involves a problem. Remember that linear transformations can be reproduced only by the $\phi_i$ and $\psi_j$ coordinates together. When we approximate the normal constantly, we loose this important property. But by updating the normals in every time step, linear transformations are approximated well. But for large time steps, the rotational component of a dynamic deformation is lost after a certain time. Of course we could just choose very small time steps but nevertheless, a very small time step is not desirable for some applications.

First, user interaction is cumbersome for simulations with small time steps. Then, the implicit time integration damps the system only for not too small time steps. Hence, a tiny time step is not desirable and a tradeoff is the best choice.

As an example of the time step’s influence on the simulation, consider Figure 8.9.

8.2. GC Fitting

The dinosaur depicted in Figure 8.10 is deformed by GC fitting on manually moved goal positions. Afterwards, the dinosaur is also deformed dynamically using shape matching to deform the goal positions. Unlike GCFEM, the simulation runs real-time. The framerate depends heavily on the deformation since a large deformation needs more minimization steps. But as long as the system stays stable, the framerate is above 20 FPS. The system stays stable even for large deformations. But exceptions always appear, mainly when the goal positions represent a shape that is not reachable by the cage.

Edge length preservation is essential for a stable deformation. The length preservation parameter is normally set to 0.2 or 0.3. An unstable system can normally be brought to a stable system by enhancing the adaption parameter.

The precomputation time for GC fitting is dominated by the volume sampling. A complex object needs a good sampling. We approximated the dinosaur’s volume with 80 nodes that were chosen from a grid of resolution 10. This procedure takes about ten seconds. The following
8. Results

computation of Green coordinates is done in two seconds.

The stability and the quality of GC fitting depends strongly on the matching points. The small
details of an object has to be covered by matching points as well. We also tried to distribute
weights to the matching points. The weights are applied to the error function such that matching
points that lie nearer to the boundary are more important than matching points nearer to the
center. We hoped that the quality of the deformed object would improve since the object is
represented by a surface mesh and is more influenced by the matching points in the vicinity
of the surface. Figure 8.11 shows that uniform weights and position-based weights results in
similar deformations.

8.3. GC Representation for Standard FEM Simulations

We handle here very shortly the FEM with GC representation presented in Section 5.4. Since
the dynamics of the system is based on an independent FEM system, the stability depends only
on this FEM system. The grid merging and the GC representation are both stable methods.

We have already shown in Section 5.4 that the cage-based GC representation shows better re-
sults than the voxel-based trilinear coordinates representation. Figure 5.3 compares the two
representations. Figure 8.12 depicts another scene of the same example. GC preserve the shape
much better.

The decimation from the voxel grid to the cage is almost free. The most time-consuming com-
putation is the update of the mesh nodes with Green coordinates.
The object that we tested in Figure 8.12 contains 10k triangles. The voxel grid’s resolution is
six. The resulting cage contained 72 vertices and 140 triangles. The precomputation of the
Green coordinates took 2572ms, but only 125ms for the FEM system we use. The simulation
runs with 6 FPS that is quite slow due the GC representation of the complex surface mesh. An
example with less triangles on the object and on the cage would run much faster and the limiting
step would be the FEM system instead of the GC update.
Even though the GC representation restricts the performance speed, the framerate is still better
than for GCFEM simulations. The same is valid for the precomputation time which is enormous
for GCFEM.

In this method, Green coordinates can only be used for representation. The one-way coupling
from the FEM simulation to the GC representation makes applying a force on the surface mesh
impossible. This is the same drawback as we have in GC fitting.
8.3. GC Representation for Standard FEM Simulations

Figure 8.4: These graphs are based on the convergence example depicted in Figure 8.3. The squared norm of the difference between the new and the old solution vector $\mathbf{u}$ is plotted for each iteration. The second measurement are the squared differences of the angles between the surface mesh’s triangles. The third graph shows the squared stretches of the surface mesh’s triangles. With stretch we mean here the ratio of the new to the old triangle area. The zig-zag curves indicate that the aimed deformation does not minimizes the angle differences and the stretches, but it converges to the desired deformation. Minimized stretch and angle difference would mean that the deformation is completely undone.
8. Results

Figure 8.5.: A static deformation of the goblin object. The same deformation is done for different amounts of quadrature points. 50 quadrature points led to no convergence. Note that deformation artifacts in the hand can only be avoided with a high number of quadrature points.

Figure 8.6.: The first image shows two bars. The yellow bar is simulated with GCFEM and 16 degrees of freedom. The green bar is simulated with element-based FEM and 32 degrees of freedom. One end of the bars is fixed, the remaining regions underlie gravity. The second picture shows the deformed bars. The simulation’s framerate is about 10 FPS where GCFEM uses 82% of the computation time.

Figure 8.7.: Two illustrations of a deformed goblin. Once the deformation is simulated dynamically with GCFEM and once with FEM. The FEM simulation cannot separate the two legs even though the grid resolution is high such that its framerate is extremely lower than the GCFEM’s framerate. For both simulations, the goblin’s right foot is fixed. A magnification of the legs are depicted at the top left corners.
8.3. GC Representation for Standard FEM Simulations

**Figure 8.8.** Dynamic GCFEM on the dinosaur object. A force is continuously applied on the head and pulls toward the red point.

**Figure 8.9.** A cube is accelerated by gravity and collides with a plane. The maximal rebound is depicted for the same simulation with two different time steps. The rebound height with the smaller time step is clearly greater because the system undergoes less damping.
8. Results

**Figure 8.10.** GC fitting on the dinosaur object

(a) Uniform weights  
(b) Position-based weights

**Figure 8.11.** The same deformation with or without position-based weights.

(a) Trilinear coordinates  
(b) Green coordinates

**Figure 8.12.** The same dynamic simulation represented with Green coordinates and trilinear coordinates.
Conclusion and Outlook

We have shown how to use Green coordinates in elastic deformation systems. We mainly focused on a FEM-like system that is derived by a Ritz-Galerkin discretization based on Green coordinates. The discretization with normal-based basis functions results in a non-linear system on which we applied further simplifications. Furthermore, we proposed two techniques that use Green coordinates only for the representation of the surface mesh. The dynamic simulation is independent of Green coordinates and therefore we do not encounter problems due to non-linearity as we have for the Ritz-Galerkin discretization.

Our motivation to use Green coordinates is based on the properties that they show. The additional normal-based basis functions allow us to get a least-distorting representation of our deformable object. Furthermore Green coordinates can be computed analytically. They are also able to reproduce translation, rotation and scaling.

9.1. GCFEM

The non-linearity of the normal-term makes GCFEM a non-linear and very complex system. Approximating the normals constantly or linearly yields a new stiffness matrix that transforms the change of normals into a force. This force does not appear in other FEM systems which contain only one stiffness matrix that transforms the displacement into a force (which our system also contains). This correction force is not easy to interpret but is required since it contains the other basis functions that are not contained in the other stiffness matrix. And both kinds of basis functions are needed in order to get the properties that motivated us to use GC.
9. Conclusion and Outlook

To get better accuracy and no loss of rotation, we suppose to take a linear approximation of the normal instead of a constant approximation (see Appendix A). GCFEM with linearized normal was not tested, but could be a good way to improve GCFEM’s quality. An other open question is how the non-linear GCFEM system would improve deformations. Until non-linear GCFEM is tested, we cannot say for sure how much the approximation of the normals affects the system.

Anyway, we are satisfied with the results we got with GCFEM. We could develop a fully functional deformation system that includes warped stiffness, boundary constraints and collisions. The whole implementation is close to other FEM systems, but extends them by adding some new terms. Unlike other FEM systems, GCFEM works with only a few degrees of freedom. Cage design allows us to decide which part of an object can be deformed independently of the other parts, and where more degrees of freedom are required. GCFEM exploits also the good properties of Green coordinates and we get reasonable deformations that exhibit little distortion.

An important open problem is the creation of an object’s cage. In our implementation, we added manually constructed cages. This contains several drawbacks. First, it is time-consuming to do this by hand. Then, a manually constructed cage is not easy to build, specially if we consider the requirement that cage and surface mesh should lie close together due to mentioned stability problems. To sample the cage volume instead of the surface mesh’s volume is actually a wrong approach. Even though, the cages are of similar shape as the surface meshes’ shapes, the cage is simulated instead of the object itself. A better solution would be a cage that lies much nearer to the surface mesh and has an almost exact reproduced shape. Unfortunately, this is not possible for many cases when we want to keep the number of degrees of freedom low. An interesting approach would be to simulate the object’s volume by taking the surface mesh as the GC cage. Anyhow, this will arise new problems since GC are not defined on the cage’s boundary and the surface meshes normally contain much more vertices as required for a cage.

The deformations of GCFEM are quasi-conformal and least-distorting and therefore look more desirable than other FEM deformations. On the other side, our GCFEM implementation runs slower than other FEM systems due to the assembly of both the stiffness matrices. Single objects that contain many non-convex regions cannot be simulated in real-time mainly because we have full matrices. GCFEM could be accelerated by rotating the 3 × 3 block matrices of the stiffness matrices in a parallel manner what would be easy to implement. One possibility would be to use the GPU for this parallel computations. Cages that lie nearer to the surface mesh would enhance the framerate as well since less quadrature points would be necessary. Another idea we have already implemented is the clustering of quadrature points. Even though clustering accelerates the simulation noticeable, too large cluster radiuses introduce other artifacts. The cluster method could be improved by defining clusters adaptively or choosing them in a more sophisticated way than we do. Since the second stiffness matrix often has not a big influence on the system, it could be neglected or approximated sometimes. An interesting question is as well how much the system would suffer when we would set the smallest values of the stiffness matrices to zero. This would give us more sparse stiffness matrices and therefore less computations, but it also could damage the coordinates’ partition of unity which is absolutely required.
9.2. Dynamic Simulation With GC Representation

Our two alternative approaches use Green coordinates only for the representation of the deformed surface mesh.
One idea is to apply an element-based FEM system for the dynamic simulation and use the boundary nodes of the element grid as a cage. The cage can be used to represent the deformed surface mesh. This method is an easy extension of a stable FEM system and enhances the deformation's representation significantly. In order to enhance the framerate, the GC representation could also be applied in a postprocessing step.

9.2.1. GC Fitting

The other idea is GC fitting which fits the cage such that some interpolated matching points reach their goal positions. The cage is then used to represent the surface mesh. The goal positions can be deformed using an arbitrary dynamic simulation since the dynamics is totally independent of Green coordinates. Thus, there is a variety of possible applications. We can simulate the goal positions with a FEM system or shape matching. But we can also move them manually. The only requirement for a GC fitting application is that the matching points and goal positions are sampled within the surface mesh.
We did not spend much time in elaborating GC fitting and there are several open problems. The main problem is that GC fitting and the dynamic simulation are one-way coupled and therefore the simulation cannot be influenced by forces that appear on the surface mesh. To add collisions or boundary constraints to the system, forces need to be applied directly on the mesh. Unfortunately, the surface mesh is deformed using GC while the goal positions may have a significantly different displacement. When we use full adaption, the goal positions are set to the shape matching’s positions and therefore forces on the surface mesh could be applied to the dynamic simulation. On the other hand, full adaption restricts the dynamic simulation.

The behavior of GC fitting could be altered arbitrarily by adding new penalty terms to the error function such as angle preserving or least-distorting cage triangles. We did not try those possibilities since edge preservation showed the desired results.

GC fitting cannot guarantee stability when the goal positions are deformed such that the matching points cannot be brought into their vicinity. Anyhow, reasonable parameter choices will lead to a stable simulation, but the time step should not be too large. When projecting the original deformation into the GC space, it can also occur that the solution is oscillating around a point that is not in the GC space. Thus, either smoothing or adaption is required for some applications.

Compared to GCFEM, the runtime performance of this method is better. And for complex meshes, the runtime can even be enhanced by applying the fitting steps after the dynamic simulation. Since the cage shape is restricted by the edge length penalty, the deformations look nice and low-distorting because the mapping between the cage and the mesh is quasi-conformal.
9. Conclusion and Outlook
A

GCFEM With Linearized Normal

In Chapter 6, we showed a way how to get a linear FEM-like system with Ritz-Galerkin discretization based on Green coordinates. In this chapter, we introduce another possibility to get a linear system that approximates the system more accurately. We hope that this method leads to a more stable and accurate system than the method discussed in Chapter 6.

This chapter starts with considering again the Taylor approximation of normals. We show how a normal is linearized and what the linearization affects GCFEM. Then, the total energy is regarded and discretized using linearized normals. Finally, we show how a static system is turned into a dynamic system.

A.1. Taylor Approximation

In Chapter 6, we regarded $m_j(x) = n_j(x)s_j(x) - n_j^0$ as a constant such that we can use the normal from the last time step. A constant normal leads to a linear system, but also can introduce approximation errors. We try to reduce approximation errors in GCFEM by using the first Taylor approximation instead of the constant approximation:

$$m_j^t = m_j^{t-1} + \sum_{k \in \ell_j} (\nabla m_j^{t-1})^T(u_k^t - u_k^{t-1}). \quad (A.1)$$

$\nabla m_j^{t-1}$ is the derivative of $ns_j$ with respect to $u_k$. Its computation can be reviewed in Appendix B.1.

Unlike the constant approximation, the first Taylor approximation contains degrees of freedom $u_k^t$ and therefore the derivative of $m_j$ with respect to a degree of freedom $u_i$ is not zero. This
A. GCFEM With Linearized Normal

will for sure give us new terms in the system, but the system will stay linear since our approximation contains only linear terms. 

\( u_{i}^{t-1} \) is regarded to be constant since it is the displacement of the last iteration and therefore given.

Figure A.1 shows that the linear approximation is much more accurate than the constant approximation.

![Figure A.1](image)

**Figure A.1:** Depicted is the same change of normal as in Figure 5.1, but only the x-coordinate of the normal over time \( t \) (blue). The red graph is the constant approximation of the normal. The green graph is the linear approximation of the normal.

A.2. Energy Equation

We consider again the derivative of the total energy in Equation (5.5). For the case of constant \( m_j \), the derivative \( \delta m_j / \delta u_i \) was equal to zero and therefore we could omit the whole term that is multiplied by the derivative. Now, the derivative is not zero since the linear approximation contains degrees of freedom and gets

\[
\frac{\delta m_j}{\delta u_i} = (\nabla m_j(u_i))^T. \tag{A.2}
\]

Hence, our linear system gets much complexity and therefore we rearrange the linear approximation in order to get an easier derivation of the resulting linear system. We rewrite the discretization \( u = \sum_i \phi_i u_i + \sum_j \psi_j m_j \) by assemble the degrees of freedom:
Now we can discretize the displacement in the same form as we did in Chapter 6:

\[ u^t = \sum_i \tilde{\phi}_i u_i^t + \sum_j \psi_j \left( m_j \left( u^{t-1} - u_i^{t-1} \right) u_i^t + \sum_k \psi_k \tilde{m}_j - \sum_k \nabla m_j \left( u_k^{t-1} - u_i^{t-1} \right) u_k^t \right). \]  

(A.3)

Now we can discretize the displacement in the same form as we did in Chapter 6:

\[ u = \sum_i \tilde{\phi}_i u_i + \sum_j \psi_j \tilde{m}_j. \]  

(A.4)

The change of notion leads to the same derivation of the linear system as we had for the case of constant \( m_j \). The derivative of the displacement with respect to a discrete displacement \( u_i \) becomes again \( \tilde{\phi}_i \). Therefore the resulting equation gets

\[ \tilde{K}u = \tilde{H}\tilde{m} - \tilde{f}. \]  

(A.5)

Even though Equation A.5 looks exactly like the GCFEM system with constant \( m_j \), GCFEM with linear \( m_j \) contains significant differences. Both the stiffness matrices \( \tilde{K} \) and \( \tilde{H} \) are composed of \( \tilde{\phi}_i \)'s which change in every iteration and therefore they have to be recomputed iteratively.

### A.3. Dynamic System

Now, we want to turn the static system (A.5) into a dynamic system. As already discussed in Chapter 5, this can be done by moving towards the steady state iteratively, i.e. we follow the negative gradient of the energy.

We need also the acceleration of the discretized displacement. While the second derivative of \( m_j \) was zero for the case of constant \( m_j \), the second derivative in time becomes now

\[ \tilde{m}_j \left( u^t \right) = \sum_{u_k^t} \nabla m_j \left( u_k^{t-1} \right) u_k^t. \]  

(A.6)

We rewrite the discretized energy again with the new coordinates \( \tilde{\phi}_i \) and get the second derivative of the displacement

\[ \ddot{u} = \sum_i \ddot{\phi}_i u_i. \]  

(A.7)
A. GCFEM With Linearized Normal

According to Chapter 6.6, we introduce an acceleration force $\int_{\Omega} \rho \ddot{u} = \sum_i \int_{\Omega} \rho \ddot{\phi}_i \tilde{u}_i$. We also know that an external force on a node $i$ is the integral over the domain of the force field times $\tilde{\phi}_i$. Applying this for our acceleration force yields a nodal force

$$\sum_j (\ddot{\phi}_i, \rho \ddot{\phi}_j)_{\Omega} u_j.$$  \hspace{1cm} (A.8)

Hence, the mass matrix will be the same as for the case of constant $m_j$ but is composed of the modified coordinates $\tilde{\phi}_i$. A $3 \times 3$ block matrix of the mass matrix $\tilde{M}$ looks as follows:

$$\tilde{M}_{i,j} = (\ddot{\phi}_i, \rho \ddot{\phi}_j)_{\Omega}.$$  \hspace{1cm} (A.9)

The resulting linear system we get is of the same form as for the case of constant $m_j$:

$$\tilde{M}\ddot{u} + C\dot{u} + K\dot{u} + \tilde{H}\tilde{m} = \tilde{f}.$$  \hspace{1cm} (A.10)

Like all the other matrices that contain the transformed coordinates $\tilde{\phi}_i$, $\tilde{M}$ has to be update in every time step. The integration over time is also done implicitly.

Table A.1 summarizes the GCFEM system with linear normal and compares the system with the GCFEM system that is based on constant normals.

<table>
<thead>
<tr>
<th>Step</th>
<th>Constant normal</th>
<th>Linearized normal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PRECOMPUTATION</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quad. points</td>
<td>Quadrature points and weights</td>
<td></td>
</tr>
<tr>
<td>Green coordinates</td>
<td>For quad. points and mesh points</td>
<td></td>
</tr>
<tr>
<td>Matrices</td>
<td>$K, H, M$</td>
<td>$K, H, M$</td>
</tr>
<tr>
<td><strong>EACH TIME STEP</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td>$m, f$</td>
<td>$m, \nabla m, \tilde{m}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\tilde{f}, \tilde{K}, \tilde{H}, \tilde{M}$</td>
</tr>
<tr>
<td>Assembly</td>
<td>$K, H$</td>
<td>$\tilde{K}, \tilde{H}$</td>
</tr>
<tr>
<td>Lin. system</td>
<td>Solve for $v$</td>
<td></td>
</tr>
<tr>
<td>Update $u$</td>
<td>$u^{t+1} = u^t + dt \cdot v^t$</td>
<td></td>
</tr>
</tbody>
</table>

*Table A.1.* Comparison of the two GCFEM algorithms with constant and linear normal.
Gradient Computations

B.1. Gradient of Normal and Scaling Factor

For a triangle $t_j$ with vertices $v_1, v_2, v_3$ we present the derivatives of $m_j$ with respect to $v_1$. The derivatives of $v_2$ and $v_3$ are the same after changing the indices such that $v_1$ becomes the desired vertex.

\[
\nabla m_j = \nabla n_j \cdot s_j + n_j (\nabla s_j)^T \tag{B.1}
\]

B.1.1. Gradient of Normal

First some new constants and variables are introduced. $\text{diff} = v_2 - v_3$ and $\text{cross} = v_2 \times v_3$ are constants. With help of them we define the unnormalized normal $x = (v_2 - v_1) \times (v_3 - v_1) = \text{cross} + v_1 \times \text{diff}$. The derivative of this vector is

\[
\nabla x = \begin{pmatrix}
0 & -\text{diff}_z & \text{diff}_y \\
\text{diff}_z & 0 & -\text{diff}_x \\
-\text{diff}_y & \text{diff}_x & 0
\end{pmatrix} \tag{B.2}
\]

and the derivative of $x^T x$ is $2 \text{diff} \times (\text{cross} + v_1 \times \text{diff}) = 2 \text{diff} \times x$. The resulting gradient of the normal is
B. Gradient Computations

\[ \nabla u = \nabla x \frac{x}{\|x\|} - \frac{x (\text{diff} \times x)^T}{\|x\|^3}. \]  

(B.3)

B.1.2. Gradient of Scaling Factor

We define \( u = v_2^0 - v_1^0 \) and \( v = v_3^0 - v_1^0 \), \( u' = v_2 - v_1 \) and \( v' = v_3 - v_1 \).

Remember the definition of

\[ s_j = \alpha \frac{\sqrt{x}}{\text{area}(t_j) \sqrt{8}} + (1 - \alpha) \]

\[ x = u^T u \cdot v'^T v' + v^T v \cdot u'^T u' - 2u^T v \cdot u'^T v'. \]  

(B.4)

The gradient of \( x \) is

\[ 2 \left( u^T v \cdot (u' + v') - v^T v \cdot u' - u^T u \cdot v' \right) \]

and the gradient of the scaling factor is

\[ \nabla s_j = \frac{\nabla x}{2 \sqrt{x} \sqrt{8 \text{area}(t_j)}}. \]  

(B.5)

B.2. Gradient phi and psi

The computation of the gradient of \( \phi \) and \( \psi \) is complex and we do not show the derivation. Only the pseudo-code of the computation is given in Algorithm 6.
Algorithm 6 function gradientGC

Require: Faces \( t_j \) with vertex positions \( v_1, v_2, v_3 \) and a point \( \eta \)

Ensure: \( \phi, \psi, \nabla \phi, \nabla \psi \)

\[ \nabla \phi = \nabla \psi = \phi = \psi = 0 \]

for all each \( t_j \) do

\[ p = ((v_1 - \eta)^T \mathbf{n}(t_j)) \mathbf{n}(t_j) + \eta \]

for \( k \in \{1, 2, 3\} \) do

\[ s = \text{sign}\left((v_k - p) \times (v_{k+1} - p) \cdot \mathbf{n}(t_j)\right) \]

\[ (I, \nabla I) = \text{GCTriIntGrad}(p, v_k, v_{k+1}, \eta, \mathbf{n}(t_j)) \]

\[ \nabla \psi[t_j] = \nabla \psi[t_j] + s \cdot \nabla I \]

\[ \psi[t_j] = \psi[t_j] + s \cdot I \]

\[ q[k] = \frac{\text{sign}(\mathbf{v}_k - \eta) \times (\mathbf{v}_k - \eta)}{\|((\mathbf{v}_{k+1} - \eta) \times (\mathbf{v}_k - \eta))\|} \]

\[ \nabla q[k] = \text{gradq}(v_k, v_{k+1}, \eta) \]

\[ (II[k], \nabla II[k]) = \text{GCTriIntGrad}(\eta, v_{k+1}, v_k, \eta, q[k]) \]

\[ w = w + q[k] II[k] \]

end for

\[ \nabla \psi[t_j] = \nabla \psi[t_j] \cdot \text{sign}(\psi) \]

\[ \psi[t_j] = |\psi[t_j]| \]

\[ w = w - \mathbf{n}(t_j) \psi[t_j] \]

if \( ||w|| > \epsilon \) then

for \( k \in \{1, 2, 3\} \) do

\[ \nabla w = -\mathbf{n}(t_j)(\nabla \psi[t_j])^T + \sum_{l=1}^{3} \nabla q[l] II[l] + q[k+1] (\nabla II[l])^T \]

\[ \nabla qw = (\nabla q[k+1])^T w + q[k+1]^T \nabla w \]

\[ \eta v_k = v_k - \eta \]

\[ qw = q[k+1] \cdot w \]

\[ qv = q[k+1] \cdot \eta v_k \]

\[ \phi[v_k] = \phi[v_k] + qw/qv \]

\[ \nabla \phi[v_k] = \nabla \phi[v_k] + \nabla qw/qv - qw/qv^2 (\nabla q[k+1] \eta v_k - q[k+1]) \]

end for

end if

end for

Algorithm 7 function gradq

Require: \( v_i, v_{i+1}, \eta \)

Ensure: \( \nabla q[i] \)

\[ \text{diff} = v_{i+1} - v_i \]

\[ \text{cross} = v_{i+1} \times v_i \]

\[ R = \begin{pmatrix} 0 & \text{diff}_z & -\text{diff}_y \\ -\text{diff}_z & 0 & \text{diff}_x \\ \text{diff}_y & -\text{diff}_x & 0 \end{pmatrix} \]

\[ x = \text{cross} + \eta \times \text{diff} \]

return \( R \frac{x}{||x||} - x (\text{diff} \times x)^T \frac{x}{||x||} \)
B. Gradient Computations

Algorithm 8 function GCTriIntGrad (1/2)

Require: p, v₁, v₂, η, n
Ensure: I, ∇I

\[ u = v₂ - v₁ \]
\[ v = p - v₁ \]
\[ w = v₂ - p \]
\[ ∇p = -nnᵀ \]
\[ x = \frac{u·v}{∥u∥∥v∥} \]

if \( x ≤ -1 \) or \( x ≥ 1 \) then
  return \((0, 0)\)
end if

\[ α = \cos^{-1}(x) \]
\[ ∇α = \frac{-1}{\sqrt{1-x²}} (\frac{∇pu}{∥u∥∥v∥} - x\frac{∇pv}{∥v∥²}) \]
\[ y = \frac{v·w}{∥v∥∥w∥} \]

if \( y ≤ -1 \) or \( y ≥ 1 \) then
  return \((0, 0)\)
end if

\[ β = \cos^{-1}(y) \]
\[ ∇β = \frac{1}{\sqrt{1-y²}} (\frac{∇p(w-v)}{∥v∥∥w∥} - y∇p \left( \frac{v}{∥v∥²} - \frac{w}{∥w∥²} \right)) \]
\[ λ = ∥v∥² sin²α \]
\[ ∇λ = 2(∇pv + sin(α) · cos(α) · ∇α) \]
\[ c = ∥p - η∥² \]
\[ ∇c = 2∇pp + 2η - ∇pη - p \]
\[ θ = Π - α \]
\[ ∇θ = -∇α \]

...
\textbf{Algorithm 9} function GCTriIntGrad (2/2)

\begin{algorithmic}
  \algfor $i = 1..2$
  \State $S = \sin(\theta)$
  \State $C = \cos(\theta)$
  \State $\nabla S = \cos(\theta) \cdot \nabla \theta$
  \State $\nabla C = -\sin(\theta) \cdot \nabla \theta$
  \State $a_1 = 2cC$
  \State $b_1 = c(1+C) + \lambda + \sqrt{\lambda^2 + \lambda cS^2}$
  \State $t_1 = 1 - \frac{a_1}{b_1}$
  \State $\nabla t_1 = \frac{-2c\nabla C + 2\nabla cC}{b_1} - \frac{1}{\nabla c(1+C) + c\nabla C} \nabla \lambda + \frac{a_1}{2\sqrt{\lambda^2 + \lambda cS^2}} \nabla \lambda S^2 + \frac{2\lambda c}{2\sqrt{\lambda^2 + \lambda cS^2}} \nabla c$
  \State $a_2 = 2\sqrt{\lambda}S^2$
  \State $b_2 = (1-C)^2$
  \State $t_2 = \frac{a_2}{b_2}$
  \State $\nabla t_2 = \frac{\lambda S^2 \nabla \lambda + 4\sqrt{\lambda}S \nabla S}{b_2} + \frac{2a_2}{(1-C)^2} \nabla C$
  \State $a_3 = \sqrt{c}C$
  \State $b_3 = \sqrt{\lambda} + S^2c$
  \State $t_3 = \frac{a_3}{b_3}$
  \State $\nabla t_3 = \frac{\sqrt{c}\nabla C + cs}{b_3} - \frac{a_3}{2(\lambda+S^2c)\frac{1}{2}} (\nabla \lambda + 2Sc \nabla S + S^2 \nabla c)$
  \State $I_i = \frac{-\text{sign}(S)}{2} \left( 2\sqrt{c} \tan^{-1}(t_3) + \sqrt{\lambda} \log(t_2)t_1 \right)$
  \State $\nabla I_i = \frac{-1}{\sqrt{c}} \left( \frac{1}{\sqrt{c}} \tan^{-1}(t_3) \nabla c + \frac{2\sqrt{c}}{1+t_3^2} \nabla t_3 + \frac{1}{2} \sqrt{\lambda} \log(t_2t_1) \nabla \lambda + \frac{1}{2t_2t_1} (t_1 \nabla t_2 + \nabla t_1 t_2) \right)$
  \State $\theta = \Pi - \alpha - \beta$
  \State $\nabla \theta = -\nabla \alpha - \nabla \beta$
  \EndFor

\State $I = I_1 - I_2 - \sqrt{c} \beta$
\State $\nabla I = \frac{-1}{\Pi} \text{sign}(I) (\nabla I_1 - \nabla I_2 - \frac{3\nabla c}{2\sqrt{c}} - \sqrt{c} \nabla \beta)$
\State $I = \frac{-1}{\Pi} |I|$
\Return $(I, \nabla I)$
\end{algorithmic}
B. Gradient Computations
Derivation of The Total Energy’s Derivative

In this chapter we simplify the derivative of the energy function

\[
E = \frac{1}{2} a(\sum_i u_i \phi_i + \sum_i m_i \psi_i, \sum_j u_j \phi_j + \sum_j m_j \psi_j) - \left(\sum_i u_i \phi_i + \sum_i m_i \psi_i, \mathbf{f}\right)_\Omega. \tag{C.1}
\]

First we rewrite \(\frac{1}{2} a(\sum_j u_j \phi_j + \sum_j m_j \psi_j, \mathbf{v})\) as

\[
\frac{1}{2} \sum_j \mathbf{u}_j^T a(\phi_j, \mathbf{v}) + \frac{1}{2} \sum_j \mathbf{m}_j^T a(\psi_j, \mathbf{v}). \tag{C.2}
\]

We separate the sums again and get

\[
\frac{1}{2} \sum_j \mathbf{u}_j^T \sum_k a(\phi_j, \phi_k) \mathbf{u}_k + \frac{1}{2} \sum_j \mathbf{u}_j^T \sum_k a(\phi_j, \psi_k) \mathbf{m}_k + \frac{1}{2} \sum_j \mathbf{m}_j^T \sum_k a(\psi_j, \phi_k) \mathbf{u}_k + \frac{1}{2} \sum_j \mathbf{m}_j^T \sum_k a(\psi_j, \psi_k) \mathbf{m}_k. \tag{C.3}
\]

We get four sums and we denote them \(s(\phi, \phi)\), \(s(\phi, \psi)\), \(s(\psi, \phi)\) and \(s(\psi, \psi)\).

Now we can compute the gradient with respect to a displacement \(u_i\) for each sum individually. The first sum is known from Section 3.5 where we computed the gradient of the same energy but discretized with only vertex-based basis functions. Its gradient is
C. Derivation of The Total Energy’s Derivative

\[
\frac{\delta s(\phi, \phi)}{\delta u_i} = \sum_j a(\phi_i, \phi_j)u_j. \tag{C.4}
\]

The second sum’s derivation is

\[
\frac{\delta s(\phi, \psi)}{\delta u_i} = \frac{1}{2} \sum_j a(\phi_i, \psi_j)m_j + \frac{1}{2} \sum_j u_j^T \sum_k a(\phi_j, \psi_k) \frac{\delta m_k}{\delta u_i}. \tag{C.5}
\]

The third sum equals the second sum and therefore we get the gradient described above twice (and the factor 1/2 disappears).

The fourth sum’s derivation gets

\[
\frac{\delta s(\psi, \psi)}{\delta u_i} = \frac{1}{2} \sum_j m_j^T \sum_k a(\psi_j, \psi_k) \frac{\delta m_k}{\delta u_i} + \frac{1}{2} \sum_j m_j^T \sum_k a(\psi_j, \psi_k) \frac{\delta m_k}{\delta u_i} \tag{C.6}
\]

what can be simplified as

\[
\frac{\delta s(\psi, \psi)}{\delta u_i} = \sum_j \frac{\delta m_j}{\delta u_i}^T \sum_k a(\psi_j, \psi_k)m_k. \tag{C.7}
\]

The derivative of the external force’s potential energy \((u, f)_\Omega\) gets

\[
\frac{\delta (\sum_j \phi_j u_j + \sum_j \psi_j m_j, f)_\Omega}{\delta u_i} = (\phi_i, f)_\Omega + \sum_j \frac{\delta m_j}{\delta u_i}^T (\psi_j, f)_\Omega. \tag{C.8}
\]

We sum up all the derivatives computed above to get the final result

\[
\frac{\delta \mathcal{E}}{\delta u_i} = \sum_j a(\phi_i, \phi_j)u_j + \sum_j a(\phi_i, \psi_j)m_j - (\phi_i, f)_\Omega
\]

\[
+ \sum_j \left( \frac{\delta m_j}{\delta u_i} \right)^T \left( \sum_k a(\psi_j, \phi_k)u_k + \sum_k a(\psi_j, \psi_k)m_k - (\psi_j, f)_\Omega \right). \tag{C.9}
\]
Bibliography


