Physically accurate chewing simulation for analyzing dental restorations

by

Andra Maria Pascale

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science

Approved Dissertation Committee:

Prof. Dr. Lars Linsen (PhD Advisor)
Jacobs University Bremen

Prof. Dr. Tobias Preusser
Jacobs University Bremen · Fraunhofer MEVIS

Dr. Steffen Hauth
Dentsply Sirona

Prof. Dr. Bernd Kordaß
University of Greifswald

Date of Defense: 17 February 2017

Computer Science and Electrical Engineering
Abstract

CAD/CAM software is the state of the art option for replacing a missing or damaged tooth, by virtually designing the replacement, without the need of taking physical imprints of the teeth. When the restoration’s shape is not optimal it can cause discomfort, requiring further adjustments. In order to prevent this event, several algorithms are used to optimize the restoration shape based on a number of parameters. In addition, a virtual articulator can be used to simulate a chewing motion and identify contact points between the proposed restoration and the teeth it comes in contact with. In an effort to take this idea one step further, we have developed an application which can simulate the chewing process together with a deformable substrate, an element missing until now. By enhancing the application with this additional feature, we are able to observe not only the interaction between the teeth but also the manner in which they interact with the food, therefore identifying a larger range of possible problem areas. Based on the results of the simulation, the optimal restoration can be chosen, therefore further minimizing the necessity of further adjustments.

The movements of the lower jaw are based on data acquired from real life subjects chewing a gummy bear. The deformable substrate is based on a Finite Element method which ensures the realism of the deformations. A collision detection mechanism based on shooting rays through the scene is used to detect active areas and compute the resulting chewing forces. When the stress in the substrate exceeds a certain threshold, the object fractures into multiple deformable pieces which continue to interact with each other until the simulation reaches its end. Throughout the simulation, the collision areas on the teeth as well as the stress areas on the substrate are emphasized with a color map. By running the simulation in the automatic mode, a series of logs collect parameters which can be used to assess the performance of the mastication process, in terms of how much contact the restorations have with the substrate and to what extent they contribute to breaking up the substrate into smaller pieces. The stress pattern experienced by the substrate, which is recorded in these logs, can be used to compare the impact that different restorations have. When a restoration inflicts a higher degree of stress in the substrate, it will cause it to fracture earlier in the simulation, and therefore would be considered a better candidate.
The delicate balance between a high degree of realism and the interactive time constraints has demanded constant optimizing in terms of mesh resolution and number of calculated iterations. It was necessary to include a series of compromises into the current solution, in order to limit the running time. Since this prototype application is the first effort of this kind, our focus was on creating a chewing simulation which proves the concept is feasible and paves the way for further work in this exciting area.
Statutory Declaration
(Declaration on Authorship of a Dissertation)

I, Andra Pascale, hereby declare, under penalty of perjury, that I am aware of the consequences of a deliberately or negligently wrongly submitted affidavit, in particular the punitive provisions of § 156 and § 161 of the Criminal Code (up to 1 year imprisonment or a fine at delivering a negligent or 3 years or a fine at a knowingly false affidavit).

Furthermore I declare that I have written this PhD thesis independently, unless where clearly stated otherwise. I have used only the sources, the data and the support that I have clearly mentioned.

This PhD thesis has not been submitted for the conferral of a degree elsewhere.

_________________________  _______________________
Bremen                        March 6, 2017
Place                          Date

_________________________
Andra Pascale
Acknowledgements

I would like to express my gratitude towards my supervisor, Prof. Dr. Lars Linsen, who has guided me through this long and difficult process, offering along the way, both moral and scientific support. In addition I would to thank the project collaborators from Greifswald, Dr. Sebastian Ruge and Prof. Dr. Bernd Kordaß for contributing to this project with their expertise and for their sustained effort and dedication throughout the project. I would like to thank Dr. Steffen Hauth from Dentsply Sirona for being part of my dissertation committee and for contributing to this project, often taking from his personal time to offer support whenever needed. Last but not least, a special thanks goes to everyone at Sirona who decided in favor of supporting this project financially, attended our regular meetings and offered advice.

I would to thank Dr. Tobias Preusser for being part of my dissertation committee.

In addition, I would like to thank the people from my research group at Jacobs University for their advice, support and friendship.

A special thanks goes to George Mandresi for offering endless emotional support and encouragement. Lastly, I would like to thank Dr. Marina Sahakyan for her friendship and kind words which often offered comfort in times of elevated stress.
Contents

1 Introduction 1
  1.1 Project implementation .............................................. 6
  1.2 Contributions .......................................................... 7
  1.3 Papers ................................................................. 8

2 Related Work 9
  2.1 Chewing application .................................................... 9
  2.2 Deformable models ..................................................... 10
  2.3 Finite Element Method ................................................ 12
    2.3.1 Strain ................................................................. 12
    2.3.2 Stress ................................................................. 13
    2.3.3 Equation of motion ............................................... 13
    2.3.4 Tetrahedral discretization ..................................... 14
    2.3.5 Corotational Linear FEM ........................................ 15
    2.3.6 Invertible FEM ..................................................... 16
  2.4 Mass Spring System ................................................... 17
  2.5 Meshless models ........................................................ 18
    2.5.1 Loosely coupled particle systems ............................ 18
    2.5.2 Meshless Shape Matching ....................................... 20
  2.6 Time integration ........................................................ 23
  2.7 Fracturing deformable models ...................................... 24
  2.8 Stress measures ........................................................ 25
  2.9 OpenTissue ............................................................... 26
  2.10 Vega FEM library ....................................................... 27
  2.11 Chosen methods for the deformable model and fracture ........ 30
  2.12 Collision detection .................................................... 30
    2.12.1 Object partitioning ............................................. 30
    2.12.2 Spatial partitioning ............................................. 32
    2.12.3 Collision detection among deformable models .............. 32
  2.13 Chosen method for collision detection ............................. 33
  2.14 Additional libraries and software ................................... 33
6 Future work 117
7 Conclusion 119

Appendix 120
Bibliography 128
Chapter 1

Introduction

Figure 1.1: A deformable substrate being deformed during chewing

CAD/CAM software is being used to virtually reconstruct missing teeth and aims at suggesting their optimal shape based on a series of algorithms. Ideally the reconstructed tooth is a perfect fit and does not need further adjustment, however this is not always the case. During chewing the patient may discover a discomfort which was not possible to predict by just looking at the results of static and dynamic occlusion.

There are several CAD/CAM software applications which are used in dentistry to prepare a restoration. The CEREC system [1] developed by Dentsply Sirona [70], is dedicated to creating a virtual model of a restoration and subsequently sending the appropriate instructions to a milling machine. The process starts with the dentist scanning the patient’s teeth with an intra-oral camera. CEREC creates a 3D virtual model of the teeth from the scans and displays it on the screen. Afterwards, the dentist can indicate where the missing tooth or preparation is and let the software compute the shape and position of the restoration to be created. Following this step, an adjustment phase begin, where the dentist or technician can inspect the contact points of the new tooth with the adjacent teeth and the opposite teeth in the closed centric position. Based on intuition and visual aids, the virtual tooth can be sculpted or slightly adjusted until it fits well with the surrounding teeth. When the model is completed, it can be immediately milled from a
block of raw material. The whole process from start to end is expected to last no more than 2.5 hours. Other similar systems exist on the market, such as Dentca [15], Dental Solutions Delcam [14] and others.

However, based solely on the contact points in the closed centric position, one cannot accurately predict how the tooth will fit during a normal chewing motion which involves a different set of lower jaw movements which in their turn elicit complex forces on the teeth based on the contact with the food. Our software aims at addressing this missing element, thus further improving the precision of the computations which create the virtual restoration model, by simulating a complete chewing process. With this idea in mind, an interactive chewing simulation has been developed which employs a physically accurate deformable model of the substrate [55]. This thesis presents the complete application which supports the physically accurate chewing process including the resulting deformation and fracture of the substrate. This project was realized in collaboration with Dentsply Sirona [70] and with a team from the University of Greifswald who has provided the chewing motion data. The scope of the projects of our collaborators can be observed in Figure 1.2.

![Figure 1.2: Project scopes and collaborators](image-url)
During simulation, in order to maintain physical accuracy, the fracture event must be implemented in a physically accurate manner that uses the properties of the material and the stress produced by the chewing forces. The fractured pieces must retain the properties of the original material and continue to deform independently and interact with each other dynamically. Furthermore, the entire chewing process must account for other elements which would influence the position of the chewed material, such as the tongue or the physical barrier of the mouth. When including fracture, not only collision detection between the deformable model and the static objects must be performed, but also collision detection between dynamically deformable models representing pieces of the substrate after fracture. The fractured pieces must repel each other continuously while they are at the same time deforming according to their properties. This requires us to deploy a flexible collision detection approach instead of a, for example, static KD tree based approach. A solution based on casting rays has been implemented to determine the intersection points between two deformable models.

The simulation starts with the substrate placed on top of one of the upper jaw molars and the mouth in an open position. As the lower jaw gets iteratively closer to the upper jaw according to the recorded jaw motion, it starts to intersect the deformable model causing it to deform. As the substrate continues to deform under the pressure of the lower jaw, at some point it starts splitting into pieces. The deformable pieces continue to deform and interact with the scene in a realistic manner. They continue to act as independent deformable models and can therefore further split into multiple pieces.

To detect the most suitable tooth reconstruction for the chewing process, the simulation can be run multiple times, each time replacing one or several teeth with different restoration candidates. The overall substrate shape can be observed at the end of the simulation. It is expected that a simulation producing a higher stress in the substrate and therefore splitting it into a higher number of pieces is representative of a more successful mastication process. In order to make this assessment, a number of parameters are recorded during the simulation, which can be compared at the run among different simulation runs. The results are evaluated based on the recorded stress values, the numbers of contact points, the number of fractured pieces and their appearance. In addition, it is also of interest to observe the imprint that a tooth leaves in the deformable model and how this is affected by the properties of the material in correlation with the tooth shape. To our knowledge this is the first approach to solve the overall goal of simulating a chewing process with fractions and analysing the chewing results for restorations of different shape with respect to chewing functionality.

CEREC 1.3 is a 3D CAD/CAM software designed by Dentsply Sirona for the dental industry. It works in collaboration with an intra-oral camera and a milling machine with the purpose of producing a 3D restoration of a crown, veneer or bridge for a human patient. The software takes as input a set of pictures of the teeth adjacent to the damaged
or missing one and creates a 3D triangular mesh of the teeth and gums. There are several cameras produced by Dentsply Sirona which can be used for this purpose. Then, with help from the user and by utilizing a database of tooth shapes, a 3D model for the missing tooth is computed and presented. If no further adjustment is needed, the model is sent to the milling machine which creates the restoration out of a previously selected material.

![CAD/CAM dentistry software CEREC](image)

*Figure 1.3: CAD/CAM dentistry software CEREC [1] developed by Dentsply Sirona [70]*

Currently, the collision of the restoration with the proximal and antagonist teeth is measured for the centric position of the jaws, i.e. the normal position in which the patient would close the jaw when relaxed. Recently, in the latest version of CEREC this measurement also accounts for other positions of the lower jaw. Dentsply Sirona has contributed with 3D upper and lower jaw triangular mesh models and libraries for data structures and collision detection methods.

![Devices](image)

*Figure 1.4: Devices which are part of the envisioned process, developed by Dentsply Sirona [70]*
Jaw motion analyser and Virtual articulator  Our second collaborator was a project carried out in the Centrum für Angewandte Informatik, Flexibles Lernen und Telemedizin at University of Greifswald, Germany [62] which was concerned with measuring the motion of the lower jaw as the patient is chewing a substrate. For the purpose of this project, the subjects chewed a gummy bear like the ones which can be seen in Figure 1.5d. The motion of the lower jaw was recorded with the Jaw Motion Analyser (co. zebris Medical, D-Isny) (see Figure 1.5c) - which is an "ultrasonic measurement system" [63] employing a number of sensors which are placed on the head of a patient who is given a gummy bear (co. Haribo, D-Bonn) to chew. In addition to the mandibular motion, it was also possible to record the muscle activity in the jaw by using electromyography (EMG) [13]. The work by Ruge and Kordass [61] presents this motion analysing device being used in conjunction with the 3D-VAS (Virtual Articulation System) software to perform virtual dynamical occlusion. A virtual articulator is a software that uses the mandibular motion data in conjunction with teeth scans to present an accurate chewing animation with dynamic display of the contact points. The 3D model visible in Figure 1.5b was produced by taking dental impressions from the patient using silicone bite registration materials and subsequently scanning the 3D casts [28].

In the following, we first discuss related work. Then, we present the overall simulation approach and general set-up, before we focus on the mastication process and the respective fracturing including collision detection and computation of resulting forces. Finally, we explain how we can compare different restorations with respect to their chewing functionality using our simulation. We present results and discuss them with respect to a number of simulation parameters.
1.1 Project implementation

The implementation of this chewing simulation project was solely my responsibility, without input from other parties. This implementation was created from scratch and it is meant to be a proof of concept. The purpose of creating this application was to investigate the feasibility of producing a realistic chewing simulation, first and foremost. Once the application was sufficiently feature-rich, its secondary purpose became to serve as a test framework for comparing different restorations. In this manner, the application proves that the idea of simulating chewing is feasible with existing technology and it also provides an example of how it can be used to optimize restoration shape.
1.2 Contributions

The following gives a brief overview of my main contributions contained in this thesis, in addition to implementing the project:

- **Related Work**
  - Literature research for methods used for deformable models and understand their advantages and disadvantages, given our constraints
  - Literature research for methods used to achieve continuous collision detection

- **Setup**
  - The setup and design of the entire system including choosing technologies and libraries was my sole responsibility
  - Creating suitable triangular and tetrahedral meshes from the provided scans
  - Creating a set of test substrate models of different shape and sizes, for example the cube and sphere models

- **Deformable model**
  - Research open source libraries which provide deformable model frameworks
  - Integrate classes from the Vega library and adapt them to the application
  - Design and implement the fracture algorithm

- **Collision detection**
  - Algorithm to allow continuous collision detection among deformable models
  - Algorithm which computes the forces based on the different collision events
  - Visualization of interest points including dynamically adjustable transparency, smoothing of triangular surface mesh, rendering of the tetrahedral mesh with adjustable space between the elements, colour maps for all scene objects, geometrical objects to represent forces, convenience buttons for focusing on the substrate while hiding/showing other elements

- **Results and Discussion**
  - Design of a new application mode which runs independent of user input, in order to run extensive automatic tests
  - Write automatic test scripts which call the application with different parameters and save the results after each call
  - Create a set of smooth restorations from the original restorations
  - Assess of the application with regards to its ability to compare restorations (creating logs, automatic screenshots)
  - Design tests which use the application to identify differences between restorations

*Note: We were not involved in the development of the Vega library which provided the
base classes for the deformable model and the integrator that was used in this application. My contribution was to research available open source libraries which had the ability to fulfill our requirements and from the pool of found libraries to choose the one most suitable. After building and assessing this library, the project was linked against the build dlls. Some of the original classes were extended in order to modify their behaviour, as their original behaviour was not sufficient. In this regard, my main contribution lies in adding the fracture component which required modifications to classes utilized by the deformable model, the volumetric model and the integrator.

**Note:** Some classes and algorithms used in collision detection were provided by Dentsply Sirona.

1.3 Papers

Parts of this thesis have been directly published as such in or adapted from sections from two papers which have been written about this project, by Pascale et al. [55] and [54]. Some of the figures in this thesis have also been published in the previously mentioned papers.
Chapter 2

Related Work

While the field of deformable objects has been around for more than a decade and was covered extensively in a large number of publications, the exact application of such a model in the context of creating a chewing simulator is, to our knowledge, the first attempt. In order to assemble this application, we focused on investigating existing research in the areas of physically based deformable models, collision detection among rigid and deformable objects, CAD/CAM systems and virtual articulators. In addition to these core elements, we explored open source libraries and software which we needed in order to create tetrahedral meshes and simplify and smooth triangular meshes. A smaller but not insignificant part of the effort has gone into understanding how the chewing process actually works as it was key in understanding of what we were supposed to achieve in the end. However this last part was not based on existing literature, but rather on empirical studies which were conducted for this purpose, using gummy bears.

2.1 Chewing application

The work presented in this thesis describes the interactive chewing application which also is covered by two papers Chewing simulation with a physically accurate deformable model by Pascale et al. [55] and Chewing simulation producing fracture in a physically-based deformable model for analysing restorations by Pascale et al. [54]. In consequence, this section is heavily based on sections included in these papers.
2.2 Deformable models

Researching methods related to deformable models brings into attention the paper by Terzopolous et al. [72] which represents the first work concerning elasticity. Since then deformable models have been widely used in computer graphics to simulate the behaviour of elastic bodies with or without a plastic component. Such simulations are used in the entertainment industry for computer games and movies where the emphasis is usually placed on aesthetic appeal and plausibility. In this case the deformations must behave according to the expectations of the user and for interactive applications especially, they should be fast. For other cases such as scientific simulations, it is necessary to preserve a certain level of physical accuracy while modeling a specific type of material. Due to the inherent complexity, these simulations usually come with an increased computation time which makes the application not interactive. Considering the different requirements and conditions imposed on deformable models, the comprehensive report by Nealen et al. [49] divides the existing approaches into several categories:

- **Finite Element Method** The deformable object is viewed as a continuous volume which needs to be discretized on an irregular mesh. This method is in general physically accurate and although normally slow, many variations exist which improve the speed problem. A comprehensive explanation on the foundations of this model can be found in the work by Müller et al.[45].

- **Mass Spring Systems** The deformable object is viewed as a network of connected massless springs. Although this method is in general fast, it has no basis in physical reality and therefore the behaviour of the object does not converge to the true solution by refining the mesh. In general, this solution cannot be used in cases

![Figure 2.1: The evolution of the chewing application across the two published papers](image-url)
where physical accuracy is important [10]. However there have been attempts to improve this problem [73].

- **Mesh Free Methods** These methods do away completely with the necessity of having a volumetric mesh for the object. They are in general more robust towards large deformations than FEM models and they can also be physically accurate depending on the underlying rules but they usually must be coupled with a surface mesh reconstruction technique. [6]
  
  - *Loosely coupled particle systems* The object is represented by a cloud of particles which in addition from their own properties, also have rules about how they interact with each other based on their position in space. By adjusting these constants and rules, many different types of materials can be modelled while preserving physical accuracy. The main drawbacks of this method come from the need to generate a surface [76]

  - *Smoothed particle hydrodynamics* This method is especially used to simulate liquids and gases. Its main disadvantage comes from the impossibility of maintaining the incompressibility of materials. Although it could be applied also to solid objects, it has been mainly applied to liquids and therefore will not be further discussed in this thesis. It has also been used to simulate melting objects, hair, lava flows and trapped air. [37]

  - *Meshless shape matching* The object is represented by a volumetric mesh but the vertices of each discrete component are treated as points and animated as a simple particle system. After each new configuration the initial mesh is fitted to the new position of the particles/vertices. [42]

  - *Lattice deforming* Lattice deforming are used by Patterson et al. [56] to speed up a classic FEM method by using a Cartesian lattice to represent the deformable object. Complex objects can be simulated and boundary cells are treated with sub-voxel accuracy by using a new second order accurate quadrature scheme for volume integrals. Volume is preserved by using an additional volume change penalization method. This method is physically accurate and fast, however in its current form it has no support for fractures.

- **Eulerian Methods** In contrast to the previously enumerated Lagrangian approaches, the Eulerian methods use an implicit representation and do not regard the object as a set of points that change their position in space. These methods consider a set of stationary points on a grid and then compute how the material properties change at these points over a period of time. Because of their intrinsic nature, Eulerian
methods are used to model fluids [9].

2.3 Finite Element Method

The finite element method regards the deformable object as a continuous volume which is discretized using an irregular mesh and models its behaviour using continuum mechanics [49]. For the current application, the object is being discretized into tetrahedra, since they are better able to capture its irregular shape. Another commonly used shape is a cube. There are 3 quantities which are the most important in computing the deformation of an object [45]:

1. **Displacement** is a measure of the modification of the spatial configuration of the object. As a vector field it is defined as \( u(m) = x(m) - m \) where \( m \) is the set of the material coordinates of object \( M \), i.e. the undeformed rest shape, and \( x(m) \) is the new location of these points after deformation, also called the world coordinates of object \( M \).

2. **Stress** \( \sigma \) is a measure of the amount of force \( f_n \) per unit area \( A \) in the object:
   \[
   \sigma = \frac{f_n}{A}
   \]

3. **Strain** \( \epsilon \) is a measure of the relative deformation \( \delta l \) of the material induced by the applied stress. For the one dimensional case this can be simply expressed as
   \[
   \epsilon = \frac{\delta l}{l}
   \]
   where \( l \) is the initial length of the object before the stress is applied.

The relationship between these quantities can be expressed using Hooke’s Law:

\[
\frac{f_n}{A} = E\frac{\delta l}{l}
\]

where \( E \) is Young’s modulus, \( f_n \) is the force, \( A \) is the area, \( \delta l \) is the relative length and \( l \) is the initial length [45]. This constant expresses the objects stiffness which is a special characteristic of deformable objects derived empirically. In a condensed form this law can be written as \( \sigma = E\epsilon \) which basically says that the relative deformation in the object depends on the object’s stiffness constant and on the applied force.

2.3.1 Strain

For the 3D dimensional case it is more complex to express the strain quantity. In most cases this is not constant throughout an object, for example when twisting a beam, parts of the material are stretched while others are compressed. It becomes clear that strain must be a function of the coordinate point in the object but also on the displacement field since an object which is not deformed should have zero strain. Therefore strain is
CHAPTER 2. RELATED WORK

represented by a 3 dimensional tensor:

$$\epsilon = \begin{bmatrix} \epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{xy} & \epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{xz} & \epsilon_{yz} & \epsilon_{zz} \end{bmatrix}$$

There are two different ways to compute strain from the spatial derivatives of the displacement field:

- **Green’s non-linear strain tensor**
  $$\epsilon_G = \frac{1}{2}(\nabla u + [\nabla u]^T + [\nabla u]^T \nabla u)$$

- **Cauchy’s linear strain tensor**
  $$\epsilon_C = \frac{1}{2}(\nabla u + [\nabla u]^T)$$

where

$$\nabla u = \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix}$$

is the gradient of the displacement field, and the comma is used to denote a spatial derivative. Depending on the specific needs of the simulation, Cauchy’s linear strain tensor might not capture correctly rotations of the model.

### 2.3.2 Stress

Similar to strain, stress is also represented by a 3D tensor:

$$\begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_{zz} \end{bmatrix}$$

because at a single material point the stress also depends on the direction of measurement being directly related to the strain.

### 2.3.3 Equation of motion

With these quantities a dynamically elastic object can be simulated by applying Newton’s second law of motion \(f = m\ddot{p}\) to the volumetric element at location \(x\) in the object. Dividing by the volume of the element, the formula becomes

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

13
where $fb(x)$ is the body force acting on the element with position $x$ and $\rho$ is the density. The force $fb(x)$ is the sum of all the forces acting on the element, this may include external forces such as gravity or collisions. To compute the force acting on one element, one must naturally sum up the forces on each face of the element, for example for a tetrahedron one must sum up four forces, one for each face. Therefore the force vector is composed of:

$$f_{\text{stress}} = \nabla \sigma = \begin{bmatrix}
\sigma_{xx,x} + \sigma_{xy,y} + \sigma_{xz,z} \\
\sigma_{yx,x} + \sigma_{yy,y} + \sigma_{yz,z} \\
\sigma_{zx,x} + \sigma_{zy,y} + \sigma_{zz,z}
\end{bmatrix}$$

In conclusion the equation of motion can be written as:

$$\rho \ddot{p} = \nabla \sigma + f_{\text{ext}}$$

where $\rho$ is the known density and $f_{\text{ext}}$ represents the external forces, $\sigma$ is derived from the strain $\epsilon$ which is derived from the displacement field, and $\ddot{p}$ is computed from $\sigma$ using Newton’s law. Since a partial differential equation must be solved to write down the solution for $p$ it becomes obvious why the finite element method is required. Naturally the solution will be only an approximation of the truth but if the discretization is sufficiently fine the errors can be tolerated.

### 2.3.4 Tetrahedral discretization

Returning to the initial volumetric mesh, we notice that the tetrahedra can be regarded as a good discretization of the continuous object. Therefore, if $p(x)$ is the deformed position of a single tetrahedron, one can write $p(x) = [p_1, p_2, p_3]^T \begin{bmatrix} x_1, x_2, x_3 \end{bmatrix}^{-1} x =Px$ . We can assume $x_0 = 0$ and $p_0 = 0$ because a simple translation would not generate elastic forces. Consequently $\nabla u = P - I$, where $I$ is the identity matrix since we remember that $u = p - x$ and $\nabla p = P$ since $p(x)$ is linear. This implies constant stress and strain inside the tetrahedron. Therefore the computations become a simple process of computing forces based on previously deformed positions and then distributing the forces evenly for the 3 vertices of a face. In the linear case, the relationship for an element $e$ can be expressed as:

$$f_e = Keu_e$$

where $f_e$ contains nodal forces and $u_e$ the nodal displacements of an element, in this case, a tetrahedron. $Ke$ is called the stiffness matrix of the element and by assembling all the stiffness matrices of the elements, a global stiffness matrix for the entire object, $K$, can be computed. The derivation of $K$ can be found in the report put together by Müller et al. [45] for the linear case when $K$ is constant at all times. Peterlik [58] also explores a non linear approach, however this choice is much less common since it involves time.
consuming computations. Therefore the linear equation of motion for the entire mesh can be written as:

\[ M\ddot{u} + C\dot{u} + Ku = f_{ext} \]  
(2.2)

where M is the mass matrix and C the damping matrix [44]. More detailed information about these mass and damping matrices can be found in Cook’s book [12]. In Equation 2.2 \( Ku \) can be replaced with \( f \) according to Equation 2.1 and therefore can be rewritten as:

\[ M\ddot{u} + C\dot{u} + f(x) = f_{ext} \]  
(2.3)

This equation can then be transformed in a system of \( 2 \times 3N \) equations of first derivatives

\[
\dot{x} = v \\
M\dot{v} = -Cv - f(x) + f_{ext} 
\]
(2.4)

Next, the Euler integration method can be used to integrate implicitly which is needed to preserve numerical stability for stiff equations. Using Euler’s implicit method, we can approximate Equation 2.4 to:

\[
x^{n+1} = x^n + \Delta t v^{n+1} \\
Mv^{n+1} = Mv^n + \Delta t (-Cv^{n+1} - F(x^{n+1}) + f_{ext}^{n+1})
\]
(2.5)

In order to solve this system of non-linear equations and find the values of \( v^{n+1} \) and \( x^{n+1} \), Mueller et al. use the Newton-Raphson method to find a solution iteratively [44].

### 2.3.5 Corotational Linear FEM

In the case of linear FEM although implicit integration provides stability over large time steps which is desirable, the linear elastic forces provide correct results only for small deformations and thus corrections must be applied. Rotational deformations are especially problematic. Several methods have been employed which address the issues of linear FEM, most notably the stiffness warping method which was introduced by Müller in the paper Interactive virtual materials [40]. This method, called Corotational Linear FEM, extracts the rotational part of the deformation of each tetrahedral element and then computes the forces separately. In Equation 2.2, \( Ku \) is written as \( K(x - x_0) \) and therefore considering also Equation 2.3, one can write for one tetrahedral element:

\[
f_e = K_e \cdot (x - x_0) = K_e \cdot x + f_{0e} 
\]
(2.6)

where \( x \) is a vector which contains the coordinates of the 4 vertices of one tetrahedron and \( f_{0e} \) contains force offsets. Assuming the rotational part of the deformation of the
tetrahedron is $R_e$ they compute the forces as:

$$f_e = R_e K_e \cdot (R_e^{-1} x - x_0) = R_e K_e R_e^{-1} x - R_e K_e x_0$$

$$f_e = K_e' x + f_0$$

where $R_e$ contains 4 copies of the rotation matrix on its diagonal. More details about the exact corotational linear FEM stiffness matrix can be found in the papers by Barbic[4] and Müller et al. [45].

The main advantage of this method is stability for large deformations while maintaining a reduced time cost similar to traditional linear FEM. Although more computations are required to compute the global stiffness matrix, they are linear in the number of elements. This method produces results with the same accuracy as linear FEM produces for rotations equal to the identity matrix. It also comes coupled with a plasticity model which was initially described in the paper by Hodgins et al. [52] and adapted to this framework. A plastic strain tensor is deducted from the total strain to derive the elastic strain. The plastic strain is initially 0 and then updated at every time step based on 3 scalar values which describe the yield, creep and maximum load. By adjusting these 3 parameters different elastic behaviours can be obtained [33].

### 2.3.6 Invertible FEM

This is a finite element based method, introduced in the work by Irving et al. [27] which produced impressive realistic results for scenarios involving deformable objects being completely squeezed between two "rigid interlocking gears" and subsequently reverting back to their original shape. The main contribution of this method is adapting the FEM method for large deformations, especially collision with other objects which would cause a tetrahedron to invert itself inside the mesh. These case are known to be very difficult to solve, one option being to simply delete these inverted elements to avoid propagating further errors. By computing a diagonalization of the deformation tensor, the authors determine the direction along which a tetrahedron becomes inverted. Their method is thus able to handle crushed or flattened tetrahedra, without accumulating numerical errors. In addition to substantial deformations, the paper also presents results involving ductile fracture, which proves that this method is feasible to use in our context, where ductile fracture of the substrate is required. This method can also be applied to anisotropic materials such as muscle, it allows for damping forces to be added and contain a plastic component as well. The main disadvantage of this method comes from the non interactive computation times, a few minutes per frame for a coarse mesh. This work was very interesting for our project since being able to handle severe crush like deformations in a robust manner was crucial for the outcome of our chewing simulation.
CHAPTER 2. RELATED WORK

2.4 Mass Spring System

The Mass Spring System is a deformable model which is generally considered to be the simplest and fastest method to realistically simulate an elastic body [45]. The object is considered to be a network of mass points connected by massless springs. Although plausible results have been produced with this method for muscle tissue deformation [50], [36], with added volume conservation by Hong et al. [26] and for cloth deformation in the papers by Provot [59], Vassilev et al. [78] and hair deformation by Selle et al. [66], it is in general a matter of reproducing an already well known behaviour up to a plausible extent. The mass spring model depends completely on the manner in which the spring network is set up and does not converge to a true solution by refining the mesh. Therefore, one must dedicate special attention to fine tuning the constants that exist in the model in order to obtain the desired behaviour. A somewhat similar method was also used by Müller et al. [73] to simulate plastic deformations with shape and volume preservation. In its most simple form [45], the model is composed from a set of N particles each having a mass, a velocity and a position. A set of springs connects these particles, each spring having a stiffness constant, a damping coefficient and a rest length. There are two force components for each spring:

- **Spring forces**

\[
\begin{align*}
  f_i &= f_s(x_i, x_j) = k_s \frac{x_j - x_i}{|x_j - x_i|}(|x_j - x_i| - l_0) \\
  f_j &= f_s(x_j, x_i) = -f(x_i, x_j) = -f_i
\end{align*}
\]  
\tag{2.8}

where \(i\) and \(j\) are the endpoints of the spring, \(k_s\) the stiffness constant and \(l_0\) the rest length.

- **Damping forces**

\[
\begin{align*}
  f_i &= f_d(x_i, v_i, x_j, v_j) = k_d (v_j - v_i) \cdot \frac{x_j - x_i}{|x_j - x_i|} \\
  f_j &= f_d(x_j, v_j, x_i, v_i) = -f_i
\end{align*}
\]  
\tag{2.9}

where \(v_i\) and \(v_j\) are the velocities of the respective particles and \(k_d\) is the damping constant.

Finally for each spring, these forces are unified into: \(f(x_i, v_i, x_j, v_j) = f_s(x_i, x_j) + f_d(x_i, v_i, x_j, v_j)\). Next using Newton’s second law of motion, the system of coupled first order Equations

\[
\begin{align*}
  \dot{v} &= f(x, v) / m \\
  \dot{x} &= v
\end{align*}
\]  
\tag{2.10}
can be solved using the Euler explicit integration scheme. First the derivatives are approximated with finite differences:

\[ \dot{v} = \frac{v_{t+1} - v_t}{\Delta t} + O(\Delta t^2) \]
\[ \dot{x} = \frac{x_{t+1} - x_t}{\Delta t} + O(\Delta t^2) \]  

and then the following update rules follow from the previous Equations:

\[ v_{t+1} = v_t + \Delta t f(x_t, v_t) / m \]
\[ x_{t+1} = x_t + \Delta t v_t \]  

The simplicity of this method and the speed make it a very good candidate for interactive applications. However, more accurate integration schemes must be used to ensure stability such as Runge-Kutta integrations or the implicit Euler scheme which is unconditionally stable. It is also recommended to implement an accurate physically based model first such as FEM and then to use that behaviour as a reference for fine tuning the constants in the mass spring model.

2.5 Meshless models

Meshless models are based on particle simulations to compute the behaviour of the deforming body and based on these computations employ different techniques to animate a surface mesh. They are usually designed to sustain stable large deformations and can simulate a diverse range of materials and elastic/plastic scenarios. These methods can be grouped in several categories based on the representation of the object such as particle clouds, point clouds etc. or based on the theoretical foundations behind the simulations such as continuum mechanics, Lennard-Jones interaction forces or Smoothed Particle Hydrodynamics theory [6] [30].

2.5.1 Loosely coupled particle systems

represent a family of methods where the object is represented by a particle cloud and no volumetric mesh is constructed. Each particle has its own properties such as mass, position, age, shape, velocity etc. depending on the simulation approach. Tonnesen et al.[71] introduce the concept of spatially coupled particles to describe particles which interact with each other based on their position in space. By using this method physically based deformations have been simulated for both solids and liquids [77] [49]. In this method, there exists a potential energy between any two particles in the cloud. This
potential energy function is borrowed from the field of molecular dynamics and derived based on the Lennard-Jones forces which model the interaction potential between pairs of atoms:

\[ \phi_{LJ}(d) = \frac{B}{d^n} - \frac{A}{d^m} \]  \hspace{1cm} (2.13)

where \( d \) is the distance between the two particles, and the rest are constants. A better control over the meaning of the constants is obtained when rewriting Equation 2.13 as:

\[ \phi(d) = -e_0 m \left( m \left( \frac{d_0}{d} \right)^n \right) - n \left( \frac{d_0}{d} \right)^m \]  \hspace{1cm} (2.14)

where \(-e_0\) is the minimal potential and \( d_0 \) is the equilibrium separation distance meaning that the potential energy for distance \( d_0 \) is exactly \(-e_0\). By adjusting constants \( m \), \( n \) and \( e_0 \), the width of the potential well can be varied. The steeper the potential curve is, forming a narrow well shape, the more stiff and brittle the material is because a very small displacement induces a large amount of force. Therefore to a soft elastic behaviour can be obtained by choosing small exponents \( m \) and \( n \). By adjusting these constants, a wide range of materials can be simulated from stiff and brittle to elastic to fluid \[49\]. Once the potential energy function has been defined, a potential energy value is calculated for each particle by summing up all the potential energies between the current particle \( p_i \) and all the other particles \( p_j \) as it can be seen in Equation 2.15 from which the force acting on \( p_i \) is derived in Equation 2.16:

\[ \phi_i = \sum_{j \neq i} \phi_{ij} \]  \hspace{1cm} (2.15)

\[ f_i = -\nabla x_i \phi_i \]  \hspace{1cm} (2.16)

However, in this exact form, this approach requires substantial parameter tuning (such as the number of particles and the afore mentioned exponents, \( m \) and \( n \)) to achieve the desired behaviour, similar to the mass spring systems method and therefore it is not guaranteed to converge to the true solution by increasing the number of particles. A similar method has been used by He et al. \[24\] to simulate fluid behaviour when coupled with a solid. Efforts have been made to fix this problem by deriving the forces exerted on the particles using continuum mechanics.

In the work by Müller et al. \[43\], a mesh free model based on continuum mechanics is introduced. Instead of particles, the object is represented by a sample of points which each have their own location \( x_i \), density \( \rho_i \), deformation \( u_i \), velocity \( v_i \), strain \( \epsilon_i \), stress \( \sigma_i \) and force \( f_i \) similar to the FEM method. From the displacement gradients, the strain and stress tensors are computed from which the forces are derived. In order to initialize the density of each point, a polynomial kernel \( W(r, h) \) is used which was first proposed by
2.5. MESHLESS MODELS

Müller in 2003 [38]. Then the density at element i is:

\[ \rho_i = \sum_j m_j w_{ij} \]

where \( w_{ij} = W(|x_j - x_i|, h_i) \). When the masses of the points are initialized, a support radius h must be chosen for each element, in this case h was chosen to be 3r where r is the average distance of one element to its 10 nearest neighbours. The spatial derivatives needed to compute the strain and stress, are derived from the displacement vectors of neighbouring pixels. Since first-order accuracy is needed, a moving least squares formulation is used. Once these are computed, the strain energy stored around an element i is computed as:

\[ U_i = \frac{1}{2} (\epsilon_i \cdot \sigma_i) \]

which is then used to derive the force acting at element i:

\[ f_i = -\nabla u_i U_i \]

A method for producing a surface animation is also included. This approach runs at interactive rates, is capable of simulating a wide range of material properties and can support topology changes without becoming unstable or accumulating errors, which can also be observed in Figure 2.3. It can only support Hookean materials and it cannot in its present form support brittle fracture but this does not pose a problem for the current project. Considering its many advantages, it is a good candidate for simulating the behaviour of a gummy bear. A similar meshless method which is physically based in continuum mechanics is used in Meshless animation of fracturing solids [57] to simulate elastic and plastic objects that can sustain brittle fracture.

2.5.2 Meshless Shape Matching

In [42] a new method is presented that is unconditionally stable, interactive and versatile. The deformable object is represented by a set of particles with masses and initial positions. Then a simple particle simulation without particle-particle interactions is performed which results in a new set of positions for the particles. Then the shape of the object is fitted in the least squares sense to the set of new positions, which results in a set of goal positions. The translation vector \( t \) and rotation matrix \( R \) which minimize

\[ \sum_i m_i (R(x_i^0 - t_0) + t - x_i)^2 \]
CHAPTER 2. RELATED WORK

![Figure 2.2: Different types of materials simulated with the point physics simulator](image)

(a) Initial Bar represented by particle elements  
(b) Elastic Bar  
(c) Plastic Bar  
(d) Melting Bar

Figure 2.2: Different types of materials simulated with the point physics simulator [48] © by Matthias Müller-Fischer

where $x_i^0$ is the initial position, $x_i$ the new position and $m_i$ the mass, are:

$$
\begin{align*}
t_0 &= \frac{\sum_i m_i x_i^0}{\sum_i m_i}, \\
t &= \frac{\sum_i m_i x_i}{\sum_i m_i}, \\
R &= \left(\sum_i m_i p_i q_i^T\right) \sqrt{\left(\sum_i m_i p_i q_i^T\right)^T \left(\sum_i m_i p_i q_i^T\right)}, q_i = x_i^0 - t^0, p_i = x_i - t \\
A &= \left(\sum_i m_i p_i q_i^T\right) \left(\sum_i m_i q_i q_i^T\right)^{-1}
\end{align*}
$$

(2.18)

Since $x_i^0$, $x_i$ and $m_i$ are all known, this is a simple calculation following which the goal positions are found as $g_i = R(x_i^0 - t^0) + t$. In order to allow for a more elaborate deviation from the initial shape, the goal position can be calculated also as $g_i = \beta A + (1 - \beta)R$ where $A$ is divided by $\sqrt{\det(A)}$ to preserve the volume of the object. Quadratic deformations are also possible with a slight modification of the calculation of matrix $A$.

The next step is to pull the particles towards their goal positions slowly and gradually in time. This is achieved with a time integration scheme which although explicit is always
stable. The positions are updated with the following rule:

\[ v_i(t + h) = v_i(t) + \alpha \frac{g_i(t) - x_i(t)}{h} + \frac{h f_{ext}}{m_i} \]
\[ x_i(t + h) = x_i(t) + h v_i(t + h) \]  \hspace{1cm} (2.19)

where \( \alpha \) is a stiffness constant in the interval \([0...1]\). Plasticity is also included in the model by introducing a plastic creep and yield constant. This method is promising because it is unconditionally stable, interactive and can support large deformations but unfortunately it is not physically based and does not converge to a true solution when the number of the particles is increased. Müller et al. [41] obtained promising results with a similar geometrically based method that works directly with positions instead of forces or velocities, to simulate complex cloth behaviour interactively. Building on top of these two methods, in the work by Müller et al. [39], oriented particles are used to extend the breadth of objects which can be modelled to complex shape such as a car with spinning wheels and to improve the quality of the mesh being rendered while preserving the same speed.
CHAPTER 2. RELATED WORK

Figure 2.3: 2D circle shape defined by a few points, bounces of the walls in a meshless shape matching demo [47] ©2005 by Matthias Müller-Fischer

2.6 Time integration

Since all the above methods require a time integration step, it was also important to explore what alternatives were possible. The first choice to be made was between implicit and explicit integration. Explicit scheme are conditionally stable and usually faster and easier to compute. However, being conditionally stable means that their stability depends on the size of the time step being used. Depending on the stiffness of the object, this
time step could be required to become unrealistically small. Implicit schemes are more computationally expensive and therefore time consuming but they are unconditionally stable because they step into the future according to physical laws unlike the explicit ones which step into the future blindly [30].

A simple explicit Euler scheme:

\[
\begin{align*}
v^{t+1} &= v^t + \nabla f(x^t, v^t) / m \\
x^{t+1} &= x^t + \nabla t v^t
\end{align*}
\] (2.20)

can be made implicit by a slight modification:

\[
\begin{align*}
\dot{v}^{t+1} &= \dot{v}^t + \nabla t f(x^{t+1}) / m \\
x^{t+1} &= x^t + \nabla t \dot{v}^{t+1}
\end{align*}
\] (2.21)

which is now as the implicit backward Euler method and is also one of the most commonly used with deformable models [35]. Another commonly used implicit integration method is the implicit Newmark integration scheme:

\[
\begin{align*}
\dot{u}_{n+1} &= \dot{u}_n + \frac{\nabla t}{2}(\ddot{u}_n + \ddot{u}_{n+1}) \\
u_{n+1} &= u_n + \nabla t \dot{u}_n + \frac{1 - 2\beta}{2} \nabla t^2 \dot{u}_n + \beta \nabla t^2 \ddot{u}_{n+1}
\end{align*}
\] (2.22)

where \(\beta\) is usually chosen as 0.25 [51].

### 2.7 Fracturing deformable models

Fracture of the model is necessary when internal stresses overcome a certain threshold which can be found empirically. Fracture models that can be used with FEM are described in several papers including the paper by Müller et al. [40] and the one by Teschner et al. and [46] which introduce a simple fracture model that can be computed at interactive rates for a general material model. Terzopoulos et al. [72] present a fracture model fit for simulating tears in a sheet of paper, Pauly et al. [57] describe a physically accurate and versatile algorithm that is time consuming, Tonnesen [81] handles only straight cuts through a model, Dorsey et al. [44] present a technique which is specialized in stiff materials and O’Brien et al.[52] describe a detailed fracture model that produces visually pleasing results but is not capable of achieving interactive rates.

In Müller’s work [40], the fracture is physically based but limited to the tetrahedral mesh. First the maximum tensile stress \(\sigma_{\text{max}}\) is found together with its direction \(n_{\text{max}}\) for every tetrahedron. If \(\sigma_{\text{max}}\) exceeds a threshold which is specific to the material being modelled, a crack is initiated or propagated starting from one of the vertices of that tetrahedron. Next, a plane \(\alpha\) is placed perpendicular to \(n_{\text{max}}\) through the selected crack point and from
the initial vertex, two new vertices are created. Then, for all the adjacent tetrahedra, one of the new vertices is assigned based on their position towards $\alpha$ and their vertices are marked as crack tips. A similar method is also used by Teschner et al. [46] to implement a fast and reasonable fracture behaviour, the only significant difference being the vertex assignment for the cubic elements.

### 2.8 Stress measures

The stress measure, mentioned in the previous section, can be defined in multiple manners. We distinguish between the following stress measures:

- **Cauchy stress** - also known as the “true stress” is the classic stress measure used in linear analysis, and the represents the standard concept of stress being the force per unit area in the deformed configuration. In simple terms, the Cauchy tensor $\sigma$, is defined as: $t(n) = \sigma n$, where $t$ is the traction vector corresponding to normal vector $n$. The traction vector $t(n)$ is otherwise defined as $t(n) = \lim_{\Delta a \to 0} f(x) = \Delta p \Delta a$, where for a given element, $\Delta a$ is the its area and $\Delta p$ is the resultant force on this area. Therefore, if we consider $da$ to be the deformed area of an element and $dp$ to be the force acting on this area, we have the following the following relation:

$$dp = tda = \sigma da$$

where $\sigma$ is the Cauchy stress tensor, according to the book by Bonet and Wood [8].

- **First Piola-Kirchhoff stress** - this measure, commonly known as $P$, represents the force acting on the surface element per unit area in the undeformed configuration. This measure is an unsymmetrical “two point stress tensor that relates an area vector in the initial configuration to the corresponding force vector in the current configuration” [8]. We can define $P$, the Piola-Kirchhoff stress, as:

$$P = fJ\sigma F^{-T}$$

where $f$ represents the Jacobian of the deformation gradient $F$.

- **Second Piola-Kirchhoff stress** - this measure, commonly known as $S$, is a symmetric stress tensor which is related to the previously presented stress measures as following:

$$S = JF^{-1}\sigma F^{-T} = F^{-1}P$$

where $\sigma$ is the Cauchy stress and $P$ is the first PK stress. More details about this measure can be found in the book by Bonet and Wood [8].
2.9. OPENTISSUE

Figure 2.4: Representation of the fracture procedure employed by Müller-Fischer et al. [40]

(a) Initial fracture

(b) Complete fracture

Figure 2.5: Fracture of a cow model obtained with a physics simulator demo based on [40] ©by Matthias Müller-Fischer

2.9 OpenTissue

OpenTissue [18] [11] is an open source project, free for commercial use, which is designed to support interactive deformable models. It contains a large number of mini-libraries that implement a variety of methods from deformable models to mesh generation and collision detection. Initially it was meant to be a header-only meta library so that it can be easily included in a C++ project. Unfortunately, it is currently dependent on many other third-party libraries which can make it very difficult to compile and run. Dependency on GLEW and GLUT especially poses difficulties since the project described in this proposal is designed to be written in conjunction with DirectX 11. Nevertheless, it contains implementations of several algorithms that are especially interesting:

- Meshless shape matching
- FEM with stiffness warping which can be seen in Figure 2.6b
- Smoothed particle hydrodynamics which is pictured in Figure 2.6c
- Particle systems which is exemplified in Figure 2.6a
- BV hierarchies
CHAPTER 2. RELATED WORK

(a) Elastic body based on particles
(b) FEM based model
(c) Water simulation step 1
(d) Water simulation step 2

**Figure 2.6: OpenTissue Demos**

- Mesh processing methods

### 2.10 Vega FEM library

In our exploration into existing open source implementations of the FEM model, we came across another prominent candidate, namely the Vega Simulator [5]. The Vega FEM library [5] [69] is an open source C++ physics library which is aimed at simulating deformable 3-dimensional models, including large deformations and anisotropic materials. It has its own file format for tetrahedral meshes called .veg which, besides information about vertices and tetrahedra, also contains information about material density and other material properties. The material can also be divided into subregions which have different densities. In addition to tetrahedral meshes, cubic meshes are also supported. It has its own file format for tetrahedral meshes called .veg which, besides information about vertices and tetrahedra, also contains information about material density and other material properties. The material can also be divided into subregions which have different densities. In addition to tetrahedral meshes, cubic meshes are also supported.
Several widely used methods are supported by Vega including:

- Linear FEM
- Corotational FEM [40]
- Invertible FEM [27]
- Saint-Venant Kirchhoff FEM [3]
- Mass Spring systems

It also supports several types of explicit and implicit integration techniques:

- Explicit Euler
- Central Differences
- Symplectic Euler
- Implicit Backward Euler
- Implicit Newmark [51]

In addition to providing classes for representing deformable models, it also provides numerical integrator classes which are required to compute a deformation given a set of external forces. Furthermore the library comes with a fully working example application which also provides user interaction features, in order to show how their classes can be used to create a working application. However it has no support for collision detection and fracture. It also does not directly support plasticity in the model and it does not support 3D shells such as cloth nor fluids. Since we had our own system for visualizing models based on DirectX rather the Vega employed OpenGL and our user interface based on DXUT, we focused our interest only on their deformable model and integrator classes.
CHAPTER 2. RELATED WORK

Figure 2.7: Deformations of a turtle model using the Vega Simulator application (turtle model was included with the demo application) [5]

(a) Turtle model at rest
(b) Turtle model deformed

Figure 2.8: Deformations of a self provided gummy bear model using different flavors of the FEM model inside the Vega Simulator demo application [5]

(a) Corotational FEM
(b) Invertible FEM
(c) Linear FEM
2.11 Chosen methods for the deformable model and fracture

The Mass Spring System method was interesting due to its speed but we decided against it due to its lack of physical accuracy. The particle based systems were promising due to their flexibility but as the literature research revealed, they are most suited for environments where there is no need for a physical surface, such as modelling a liquid or a cloud. The lack of physical accuracy was the main reason why the meshless shape matching method was not suitable. The lattice deformer based method was both accurate and fast, however keeping in mind that we needed to add fracture, which it does not support, we decided against it too. Therefore, given our strict requirement for physical accuracy, we have decided to use the Finite Element method to create the deformable model which represents the substrate. Despite its complexity and lengthy computations, there are a number of variations from the standard technique which either improve its speed or its stability, which are explored in Section 2.3. Out of these choices, we have opted in favour of the Invertible FE method which is described in Section 2.3.6, due to its proven ability to withstand and recover from severe deformations. This method is also available in the Vega library and the details of its implementation can be found in the paper by Sin et al. [69]. Another strong argument in favour of choosing a method based on FE, was the large amount of already existing algorithms for fracturing objects, as described in Section 2.7. Out of these choices, we decided in favour of the method presented by Müller et al. [40] since it was fast, physically accurate, designed for volumetric objects (as opposed to a sheet of paper) and described in detail.

2.12 Collision detection

Collision detection typically refers to the computational problem of detecting the intersection of two or more objects. This is necessary in the context of physical simulations, which in the present case refers to the collisions between teeth and the collisions between teeth and substrate. In addition to determining whether two objects have collided, collision detection systems may also report the set of intersecting points. A collision detection system is composed of a choice of object representation in the form of a data structure and a choice of a detection algorithm which most commonly reports a collision when the distance between two objects falls under a certain threshold [75].

2.12.1 Object partitioning

A scene is usually composed of multiple objects which might collide with each other. For a basic representation of simple geometry a cubic bounding box can be used but, for more
complex models, it is common to represent the objects with a tree based data structure. For objects that are convex polyhedra a commonly used approach tries to find a plane which has the first object in the right side and the second object in the left side. If no such plane is found then the objects are colliding [31] [34]. For polyhedral objects with no known structural information and possible missing edges or non-manifold geometry, also called polygon soups, Oriented Bounding Boxes Trees are a good choice [64]. They are fit for situations of close proximity where the objects can have multiple contacts. They are also fit for simulations that require very accurate contact determination, as it the case of the current project [21]. A hierarchy of oriented bounding boxes is also used by Redon et al. [60] to compute continuous collision detection for polygon soups at interactive rates.

The bounding boxes hierarchy models save time by avoiding unnecessary intersection computations. If two complex objects are bounded by two cubes and these two cubes do not intersect then there is no need to check for intersections between the many polygons of each complex object. If the cubes do intersect, then they are divided into two smaller cubes that bound, for example, the top half and the bottom half of the object and then they are checked again for intersection. When the remaining objects are smaller than a threshold they are not divided any further and are considered to be leaf nodes [16].

There are several possible shapes for the bounding box:

- **Axis Aligned Bounding Boxes (AABB)** - represented as two points min and max in world coordinate space. Preferred due to simplicity and speed. The main disadvantages include the possibility of having large white spaces, however some studies have shown that in general this type of bounding boxes have the best performance [23].

- **Oriented Bounding Boxes (OBB)** - represented as a position and an orientation in world coordinate space, and an extent vector storing the extents in the x, y, and z directions. The extent vector is given in the coordinate frame of the OBB.

- **Spheres** - represented as a position in world coordinate space and a radius.

- **Capsules** - represented as a position and a direction vector in world coordinate space, and a radius.

- **Convex Hull BV** - each node in the bounding volume hierarchy is represented by a convex hull of its children. This method works well for general shape rigid polyhedra and the results include collision detection between the upper and lower jaw teeth according to the work of Ehmann et al. [17]

- **Zonotopes** – in contrast to the previous explicit bounding volumes, this method is based on implicit bounding volumes [22]
2.12.2 Spatial partitioning

Spatial partitioning is a slightly different approach based on the same fundamental idea. Instead of splitting the object itself into subregions, the 3D space is divided into partitions and used together with a tree hierarchy, most commonly Octrees, BSP trees or KD trees [2]. KD trees are data structures used to represent the spatial partitioning of a k dimensional space. Being a special case of binary trees, every node of the KD tree represents a point in the k dimensional space and is associated with one of the axes of that space. Every node which is not a leaf node, has two children and can be interpreted as an axes-aligned hyperplane splitting the space in two sides also known as half-spaces. The points which lie on one side of this hyperplane are found in the left subtree of this node and the points which lie on the other side are found in the right subtree [32]. An example of such a construction for a 2D space can be seen in Figure 2.9. KD trees are mostly used in nearest neighbour and range searches of multi dimensional data but they have also been shown to be especially efficient in ray tracing algorithms [80].

![Figure 2.9: KD tree associated with a 2D spatial partitioning method](image)

2.12.3 Collision detection among deformable models

In addition to the well explored topics of collision detection between rigid models for which numerous solutions exist, as presented above, we were interested in a solution for collision detection among multiple continuously deforming objects which also aims at interactive speeds. Works such as [7], [74] and [75], [60] (although focused on rigid bodies) present methods for dealing with this complex problem. However, in our application we were confronted with finding a unified solution which could simultaneously solve the collision detection problem for both rigid and non-rigid objects while at the same time retrieving the information needed to compute the forces resulting from the collisions. This last requirement in particular focused our search on ray-based approaches such as the ones described in [25], [80] which is focused specifically on KD trees and [64] - which is focused on OBB trees.
CHAPTER 2. RELATED WORK

2.13 Chosen method for collision detection

After investigating the existing methods for performing collision detection on rigid and deformable objects, it became clear that we cannot directly apply any of the existing methods, but only use them as inspiration. In our case, we needed to find the fastest method which not only identifies the intersecting triangles, but also calculated the forces resulting from these intersections, ideally at the same time. While our approach does not follow exactly any of these described methods, it was created by adapting these ideas to our needs and given framework. In particular, we opted for a solution based on casting rays into a scene represented by KD trees. The details of our solution can be found in Section 4.3.

2.14 Additional libraries and software

I would like to mention here the TetGen library [68] which is a much needed tool when one needs to create a tetrahedral mesh out of an irregular triangle mesh. In addition to creating a tetrahedral mesh, the library also allows the user to refine the mesh by setting a number of parameters and with enough effort it can produce a mesh of a decent quality which can keep the numerical errors accumulated by the Finite Element method to a minimum. Another useful library which achieves the same purpose is NetGen [65]. When it came to simplifying triangular meshes by reducing the number of polygons, the freely available Meshlab software [53] was found to be very useful. It provides numerous filters and methods to reduce triangular meshes with the mesh decimation algorithm by Gahm and Gustav [20], centre the mesh in the coordinate system (which is very helpful when dealing with 3D scenes), close holes in the mesh and smooth the surface of a mesh by employing an improved Laplacian smoothing algorithm [79] which I have also implemented in the project as it was needed there as well.
Chapter 3

Setup

3.1 Overview

This simulation software has been implemented in C++ and DirectX under Visual Studio. Besides the Vega library, an additional private library from Dentsply Sirona has been used to create data structures and to implement a part of the collision detection mechanism. The user interface is based on DXUT (see Figure 3.1a). The application first loads scanned data of teeth and restorations, parses the chewing data file and sets up the start scene which can be seen in Figure 3.1b. In the manual mode, the user can then step through the simulation by triggering jaw movements and subsequent deformation steps. The restorations are separate objects which can be produced with a CAD/CAM software like CEREC. The upper and lower jaws are meant to be 3D models which were created by scanning a real set of teeth. The substrate model is predefined and not meant to be interchangeabe. Originally the idea was to have a gummy bear shaped substrate, which
is still possible, however we have decided to favour a cubic model for our final tests. Theoretically any model shape is possible to load into the application but in practice numerical errors can accumulate for unusual shapes. The graphical user interface of the application allows the user to control the chewing process, enable/disable colour maps and focus on interest points (see Section 4.1).

3.2 Event flow

In Figure 3.2 a rough overview of the chewing simulation event flow can be seen. First the scene setup step is performed, which includes loading the chewing data, scanned teeth, restorations and the substrate model. When the input components are loaded, the simulation is ready to start. The user initiates the simulation by triggering a chewing iteration, which moves the lower jaw to a new position. Following this step, a collision detection mechanism is triggered which retrieves the intersections between the substrate and both the upper and lower jaws, which is then used to compute the forces which are acting on the substrate. Consequently, the substrate is deformed according to these forces and its shape is readjusted, ending the first event cycle. These chewing iteration cycles continue until the end of the chewing data is reached. After every step which modifies an element in the scene, the scene is rendered again. The details regarding
the deformation and collision detection steps, which are the main components in the simulation, are discussed in detail in Chapter 4.

3.3 Input

In this section we present the elements of the simulation scene and how they were created. Being rooted in reality, the geometrical models in this application are the result of scanning and surface reconstructions. The green components in Figure 3.3 represent the Input components from Figure 3.2.

**Figure 3.3:** Input Scene Objects - Yellow components are part of the application while green ones are external
3.3 INPUT

(a) Powdered gummy bear before being scanned
(b) Example of a gypsum teeth mould

Figure 3.4: Input objects were scanned in order to create 3D virtual models for use in our application by our collaborators from the Greifswald University

3.3.1 Scanned Objects

The lower and upper jaws objects were obtained by scanning gypsum molds of real teeth (see Figure 3.4b), with a static scanner (Activity 850, co Smart Optics Sensortechnik, D-Bochum) at Greifswald University. The scan data of the gummy bear was obtained by scanning a powdered gummy bear (co. Haribo, D-Bonn) with the Omnicam (co. Dentsply Sirona, D-Bensheim). All the surface mesh reconstructions were simplified by using a quadric edge collapse decimation algorithm [20]. For this purpose, the free software Meshlab was used which is an open source project designed to manipulate and visualize triangular meshes. The substrate for this chewing application has initially been chosen to be a gummy bear, since it has been previously used successfully with the Jaw Motion Analyzer (co zebris Medical, D-Isny) which collects the chewing motion data [63]. The homogeneous property of the gummy bear is also fortunate since deforming inhomogeneous objects is substantially more complex and not necessary for our purposes. We have used the gummy bear in some of our final tests, alongside with other artificially created substrate models.
3.3.2 Restorations

As one can see in Figure 3.2, in addition to the scanned teeth and substrate, the application also takes as input a number of restoration models. The restorations which we have used for our tests have been created with the CEREC [1] software and each consists of a triangular mesh. The restorations fit on top of the lower jaw mesh and represent the new teeth, crowns or veneers which could potentially be used in order to fix problems that the teeth might have. Restorations could fit on any part of the scanned teeth, be it either upper or lower jaw, or the front teeth, however we need to know approximately where in the scene they will be, in order to properly place the substrate model on top of them. In Figure 3.1b, one can see the restorations which we have used in our tests, resting on the lower jaw and painted in a different colour than the rest of the teeth.

3.3.3 Chewing motion and EMG data

A significant part of our chewing application relies on chewing related data obtained with the Jaw Motion Analyser (co zebris Medical, D-Isny) by our collaborator, a project carried out at the Centrum für Angewandte Informatik, Flexibles Lernen und Telemedizin at University of Greifswald, Germany [62] which is concerned with measuring the motion of the lower jaw as the patient is chewing a substrate.

For the purpose of this project, the subjects have been chewing a gummy bear like the ones which can be seen in Figure 1.5d. The motion of the lower jaw is recorded with the Jaw Motion Analyzer which can be seen in Figure 1.5c produced by zebris Medical. It measures the mandibular movements by applying a frame and a series of sensors of the head and jaw of the subject. This device is also measuring the muscle activity in the jaw with electromyography (EMG for short) and producing a value which can be adjusted to represent the force of bite.

From this project motion data is provided in the form of 3d coordinates of 3 reference points as it can seen in Figure 3.5b on the lower jaw at regular small time intervals which cover the duration of the complete mastication of a gummy bear.

Since the jaw model must be animated, the first step is to find a transformation from the articulator frame which can be seen in Figure 3.5a to the coordinate frame of the lower jaw as it is displayed in Figure 3.5b. Since we know the position of only 3 points on the lower jaw, a 4th point is chosen on the normal of the plane defined by these 3 points in both coordinate frames. Then the transformation matrix is derived from the formula:

\[ T = A^{-1} \times B \] (3.1)

where T is the typical 4x4 transformation matrix which contains a translation and a rotation, A is a 4x4 matrix which contains the homogeneous coordinates for the 4
3.3. INPUT

(a) Articulator coordinate Frame

(b) Lower jaw coordinate frame

Figure 3.5: Chewing Simulator data obtained by our collaborators from Greifswald University [28]

points in the lower jaw space and B is a 4x4 matrix which contains the 4 points in the articulator space. Then the simulator can time step through the chewing by computing a transformation matrix between chewing steps in the same manner and applying it to the model.

The EMG values come in a set of 4, two values representing the activity of the left masseter and temporalis muscles and two values for the left side. Each position of the lower jaw is accompanied by this set of 4 EMG values. The average values between the activity measured in the masseter and temporalis on the side of the active chewing was considered to be most indicative of the amount of force that the jaw is producing during chewing. Chewing motion data contains 3D positions of 3 reference points on the lower jaw at different time steps. In addition, it contains a value representing the muscle activity strength for every time step. This value is being used to compute the force of the bite at every chewing iteration. The position of the upper jaw naturally remains fixed during the chewing activity. The simulation begins with the jaws in maximum open
position and the gummy bear object placed on top of the teeth from the lower jaw. There are a number of predefined positions which can be chosen interactively, but only the default one is matching with the position the gummy bear had on the teeth of the subject from whom the chewing data was collected.

### 3.4 Substrate tetrahedral meshes

Since our application can work with any tetrahedral model, we have created a number of substrate meshes of different shapes in order to observe what effect the shape might have on the deformation process. A summary of all available meshes and their number of components can be found in Table 3.4.

<table>
<thead>
<tr>
<th>Mesh Name</th>
<th>nr vertices</th>
<th>nr elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>GummyM</td>
<td>229</td>
<td>681</td>
</tr>
<tr>
<td>GummyL</td>
<td>1498</td>
<td>5611</td>
</tr>
<tr>
<td>Cube5</td>
<td>343</td>
<td>1296</td>
</tr>
<tr>
<td>Cube10</td>
<td>1728</td>
<td>7986</td>
</tr>
<tr>
<td>Cube20</td>
<td>10648</td>
<td>55566</td>
</tr>
<tr>
<td>Cube30</td>
<td>32768</td>
<td>178746</td>
</tr>
<tr>
<td>Cube40</td>
<td>74088</td>
<td>413526</td>
</tr>
<tr>
<td>SphereS</td>
<td>58</td>
<td>164</td>
</tr>
<tr>
<td>SphereM</td>
<td>323</td>
<td>1312</td>
</tr>
<tr>
<td>SphereL</td>
<td>1070</td>
<td>4687</td>
</tr>
</tbody>
</table>

#### 3.4.1 Gummy Bear

The 3D virtual model of the gummy bear was first simplified. After simplification, remaining mini holes and intersecting edges were corrected also with Meshlab provided filters. The reduced gummy bear surface mesh has been converted into a tetrahedral mesh using the Tetgen library which can generate tetrahedral meshes from STL format triangular meshes. It comes with a diversity of options to refine the mesh and to impose conditions on the size and shape of the tetrahedrons [68]. After fine tuning of these required parameters, a fairly optimal tetrahedral mesh has been obtained as it can be observed in Figure 3.6. The main quality desired in this volumetric mesh was uniformity regarding the shape and volume of the internal tetrahedra, as this is a requirement for minimizing the accumulation of error in the finite element method.
3.4. SUBSTRATE TETRAHEDRAL MESHES

Figure 3.6: Tetrahedral mesh of the gummy bear substrate created with TetGen

Figure 3.7: Tetrahedral and surface meshes of the gummy bear substrate meshes

3.4.2 Cube and Sphere

The gummy bear substrate model has many curves and concave regions on its surface and is therefore difficult to create a truly uniform tetrahedral mesh which can also capture its topological features. Therefore we have decided to also create a number of cubic substrate models which can be split into evenly sized tetrahedra, in an effort to reduce the accumulation of numerical errors. Figure 3.8 displays our finest cubic model, while Figure 3.9 shows the coarser cubic meshes which were used as well in a number of tests.

In addition to the cubic model, we thought it was interesting as well to use a sphere model, which can be seen in Figure 3.10. The sphere model was first created with Meshlab in the
CHAPTER 3. SETUP

(a) The surface mesh of the Cube40 model
(b) The tetrahedral mesh of the Cube40 model

Figure 3.8: The cubic substrate model is composed of equally sized and shaped tetrahedra, thus minimizing the accumulation of numerical errors

(a) Cube5  (b) Cube10  (c) Cube20  (d) Cube30

Figure 3.9: Tetrahedral meshes of the coarser cubic models

form of an STL triangular mesh and then processed with Tetgen to create a tetrahedral mesh. The cube model was constructed directly without using any additional libraries, in a separate project which was written for this specific purpose.

(a) SphereS tetrahedral mesh  (b) SphereM tetrahedral mesh  (c) SphereL tetrahedral mesh  (d) SphereL surface mesh

Figure 3.10: Tetrahedral and surface meshes of the spherical models
Chapter 4

The chewing process

This chapter describes how the chewing process works and goes into more details about its main components. As presented in the previous chapter, the application is designed to expect a set of external inputs, namely the scanned scene objects and the chewing data file. In addition to these, the deformable object component takes as additional input a file which describes the material properties of the substrate. Therefore, the application is rather flexible, allowing us to observe the chewing process for a series of different scenarios. There are two versions of the application:

- **Automatic mode**: neither requires nor allows any interaction with the user. It loads the chewing data and proceeds to automatically move the lower jaw and compute the resulting deformations, for a predefined number of steps. After each chewing iteration, the application computes deformation steps until no more contact points are found between the substrate and the teeth, or until a maximum number of deformation steps has been taken. We have decided to do it this way, since in reality when an object is being chewed, the teeth act with a force on the food until they deform it, until the food piece effectively takes the shape of the teeth imprint or until it breaks apart. However, it is also possible that a food piece is too hard to chew and therefore remains intact. Normally this situation would not occur, but in order to cover all possibilities and avoid that the application gets stuck into an infinite loop computing deformation steps, we set a maximum number of steps allowed. This maximum limit ensures that the simulation will finish computing. As it iterates through the simulation, it saves screenshots from different angles and writes log files to record the number of deformation iterations performed, as well as the stress values produced at each step and the number of contact points between the substrate and the teeth. The importance of these values is explored in the chapter about comparing restorations.

- **Manual mode**: the user controls the chewing process and can interact with the
The user triggers a chewing iteration with a key stroke, which moves the lower jaw to its next position. Then, the user triggers a deformation step with another key stroke and observes the results. The user can trigger as many deformation steps as they desire before triggering the next chewing iteration step. The user can at any time choose scene elements to bring into focus, can adjust the transparency of objects and can activate colour maps which emphasize the contact points between the substrate and the teeth. In addition there is a colour scheme which shows the elevated stress area in the substrate object. The user can end the simulation at any time before reaching the end of the recorded chewing data. The EMG value which represents the force produced in the jaws is also displayed for each chewing iteration. The majority of the screenshots presented in this document were obtained with this mode.

As it can observed in Figure 4.1, after the lower jaw moves to a new position, a collision step is performed which identifies all contact points between the substrate and the teeth. Based on this information, forces are calculated which are then applied to the deformable substrate. The deformations which are resulting from these forces are calculated based on the properties of the deformable model and the strength of the forces. Following this

Figure 4.1: The chewing process including the fracture steps
CHAPTER 4. THE CHEWING PROCESS

step, during the fracture calculation step, we inquire if the object should now split into pieces. If it should split, the application creates two new pieces of substrate and discards the original piece. If we have two or more substrate pieces, we perform an additional collision test to identify contact points between pieces. Since the object or the objects have now suffered deformations, we need a new collision test with the teeth and we are now trapped in the chewing loop. As mentioned above, the automatic application leaves this loop when no more contact points are found or when a maximum number of loop iterations have been carried out. In the manual mode, the user may choose to leave this loop at any time by triggering the next chewing iteration. In the following, we present the details of the deformable object including how it fractures and under what conditions, and last but not least we present the collision test procedures.

4.1 Visualization

We have added a number of features in order to improve usability and to allow the user to understand what is happening at every step, during the chewing simulation. To this end, the application has been made highly interactive, when set to the manual mode and was given a user interface which controls the scene and its elements.

4.1.1 Visibility

![Image](image_url)

(a) The scene including all elements  
(b) The substrate in focus at the centre of the scene

Figure 4.2: The user can bring the substrate in focus in order to inspect its shape

First of all, as the simulation is stepped forward, the user might want to take a closer look at the substrate, in order to better understand how it is deforming. For this purpose the other elements in the scene can be hidden and the substrate is brought into the centre of the scene, as it can be seen in Figure 4.2. In the right hand side of the user interface, we notice a series of toggle buttons which can be activated to show/hide each individual element and to switch between showing the surface mesh of the substrate to showing its
tetrahedral mesh. The restorations are also shown in a different colour than the lower and upper jaw, since they are separate objects.

### 4.1.2 Colour maps

A number of colour maps are implemented, which emphasize interest areas in the scene:

- Red - contact points with the substrate are represented with colour red on the teeth, see Figure 4.3a
- Blue - contact points with the lower jaw are represented on the substrate in blue, see Figure 4.3b
- White - contact points with the upper jaw are shown in white on the substrate, see Figure 4.3c
- Cyan - contact points between substrate pieces are shown in cyan, see Figure 4.3d

In addition to the contact points, there is a separate colour map which can be shown on the tetrahedral mesh of the substrate, namely the stress colour map, see Figure 4.4.
CHAPTER 4. THE CHEWING PROCESS

Figure 4.4: The stress colour map assigns each tetrahedron a colour in the range from green to red, where green represents the minimum amount of current stress and red represents the maximum amount.

When activated, each tetrahedron is displayed in a colour which represents the amount of stress it is currently experiencing. The mapping goes from green which implies the minimum amount of stress in the current configuration, to red which represents the maximum amount of stress which is currently present in the substrate. This colour map shows how the stress is distributed in the object and can be used to assess whether the forces produce the stress where expected.

4.1.3 Forces

In our chewing simulation, we visualize the forces which are currently acting in the scene using a 3D arrow object. There are two kind of arrows:

- Red arrows - represent the chewing forces resulting from intersections of the substrate with the teeth and can be seen in Figure 4.5a
- Blue arrows - represent the repelling forces between substrate pieces and can be seen in Figure 4.5b

Figure 4.5: The arrow objects can be shown or hidden in the scene interactively

- Red arrows - represent the chewing forces resulting from intersections of the substrate with the teeth and can be seen in Figure 4.5a
- Blue arrows - represent the repelling forces between substrate pieces and can be seen in Figure 4.5b
A detailed explanation of how the forces are calculated based on the contact points can be found in Section 4.3.

4.2 Deformable model

The model used to represent the substrate is based on FEM (Finite Element Method, see Section 2.3), in order to preserve physical accuracy, as discussed in Section 2.11. The Vega library [5] has been used to build this object which supports deformations based on forces and physical properties. First we build a volumetric mesh object which contains information about the vertices and tetrahedra which compose the object. In addition it also stores information about the material properties of the object we want to deform: the mass density expressed as $kg/m^3$, Young’s Modulus expressed in $N/m^2$ and Poisson’s ratio. As of now, these three parameters have been fine tuned empirically, following the guidelines of materials similar to silicone. We also define the material of the object as being isotropic, as the originally envisioned gummy bear is a homogeneous object. The volumetric object is used to create the deformable object which can be queried about internal forces and tangent stiffness matrix. There is a choice here from several available deformable models based on FEM, out of which the Corotational Linear FEM and Invertible FEM methods have shown the best results in terms of time cost versus stability. As mentioned in Section 2.3.5, these are both established methods which preserve physical accuracy. The Corotational Linear FEM method, while being faster and less complex, in some cases of extreme deformations produces artefacts and even becomes unstable. The Invertible FEM approach, remains stable and accurate throughout the simulation regardless of the amount of deformation, but comes with a higher time cost.

The simulation is time stepped with the help of an implicit Newmark integrator, provided also by the Vega library. Implicit integration is required in order to have unconditional stability. Due to the artefacts appearing after several deformation time steps, the Corotational Linear FEM was replaced with the Invertible FEM method which does not create these errors in the simulation.

![Figure 4.6: Deformation artifacts using Corotational Linear FEM [40]](image)

50
CHAPTER 4. THE CHEWING PROCESS

Since we are using the Vega library to setup the deformable model, we only needed to extend this class to gain access to the inner stress tensor which is computed for each tetrahedron. In order to compute a deformation we need to add forces to the model and to couple it with an integrator. In our case we choose an implicit Newmark integrator which is provided by Vega as well. A force can only be applied to an existing vertex of the model, and it is represented by a vector which has a direction and a magnitude. The amount of deformation caused by a force depends on the strength of the force and on the properties of the material. The following properties define a material in our application:

- **Mass density**: this value represents the density of the deformable object. A denser object needs a larger amount of force to deform. Therefore we had to closely relate the strength of the forces with this empirically found value. Since it was not possible to calculate an exact value, a number of tests have been carried out in order to test different material densities and find the range which would be fit to describe a gummy bear.

- **Young’s modulus**: this value describes the stiffness of the material. A given material can be very stiff, meaning it will incur a very small amount of deformation before it fractures, or the opposite it can endure a large amount of deformation without breaking. With our gummy bear in mind, we needed to describe a material which can endure significant deformations before it fractures. We have also noticed that a gummy bear is almost completely elastic, meaning that provided it is not fractured, it will return to its original shape when the force acting on it is removed.

- **Poisson’s ratio**: this value which is usually between 0 and 0.5, is related to the amount of volume conservation in the material [19]. A value closer to 0 would imply no volume conservation exits during deformation, while a higher value implies volume conservation exists. In our case, we use the value of 0.45 which implies volume is not completely conserved for our object, but rather largely conserved. The reason for choosing this value is more practical in nature, as the chosen deformable object does not work well with a complete volume conservation scheme and in order to keep the computation stable, it was better to choose the next closest value.

- **Force neighbourhood**: this is an application specific value which defines to how many neighbouring vertices a force will be propagated, with a reduced value. When we apply a force to a vertex, we have the choice to apply a force with a fraction of its magnitude to its neighbours, and so on. The effect that this has on the object is that it simulates how the application of a force affects a real object. When a force is applied to a vertex, the tetrahedra attached to that vertex experience a certain amount of stress. Even though this stress is propagated through the object by the deformable model, the propagation occurs in several deformation steps. When we apply fractions of the force to the neighbours of the vertex, all these
attached tetrahedra will experience stress at the same time, thus influencing the appearance of the deformed object. We have chosen to fix this value to the default value suggested by Vega since it is difficult to relate it directly to a real life process.

- **Fracture threshold**: a value which defines the maximum amount of stress value which an object can experience before it fractures. Since the stress value range depends on the properties of the material, this value has to be defined for each material. We have found this value empirically for each different material that we used.

In addition to these values, each substrate object has an additional property which defines its starting position in the scene. The starting position can make a significant impact on the chewing process, as it decides which teeth will be involved in deforming the substrate.
As we do not have gravity in the scene, we can place the object directly close to the upper jaw, since the first movements of the lower jaw only serve to push the object upwards towards the upper jaw, without creating any deformations in the substrate. By placing the object close to the upper jaw, we speed up these first chewing iterations which only move the lower jaw upwards, by removing a number of collision events from the scene. At each chewing iteration step we use the EMG value ( - see Section 3.3.3 ) to define the magnitude of the forces which are applied to the model. In order to simulate the sharpness of the teeth, we divide the total EMG value to the number of forces currently applied. Therefore applying only one force is similar to the substrate being acted upon by a sharp peak on the surface of a tooth. In consequence, all the stress will be gathered in one tetrahedron, significantly rising the chances that a fracture event will occur.

After being given the forces acting on the object, the integrator performs a predefined number of iterations. During each iteration, the integrator queries the invertible force model, with which it had been previously coupled, about its internal forces and stiffness matrix. It then builds a system of equations which is solved with a Vega provided conjugate gradient solver with Jacobi preconditioning. The method is based on a paper by Shewchuk [67]. The integrator returns a deformation vector which contains offsets for each vertex of the model, which are then applied by our application to the surface and the tetrahedral mesh, for rendering purposes.

4.2.1 Fracturing the substrate

In addition to deforming the surface of the substrate mesh, which implies only a spatial displacement of its vertices, we have also implemented a mechanism which can split the substrate in two separate pieces, when the stress inside the object passes a certain threshold. Following the method presented in [40], we constantly query the deformable object whether any of its tetrahedra has a stress value higher than the predefined fracture threshold. Since this is an approximation of the material properties, we have simply chosen an empirically determined fracture threshold value which is reasonable for the given object. The complexity of correlating the actual fracture threshold as measured in reality for a gummy bear with the relative values that we have available in the simulation, is beyond the scope of this project. A rough overview of the fracturing process can be observed in Figure 4.8.

Internal stress query In order to identify which areas of the substrate are currently under stress, we first collect the stress tensors associated with each element. The first Piola-Kirchhoff stress ( - see Section 2.8) is calculated inside the InvertibleFEMFracture class which we have derived from the InvertibleFEM class provided by the Vega library. The computation of the stress tensors occurs in the base class and it is based on the paper by Irving et al. [27], which presents the equation as:
4.2. DEFORMABLE MODEL

\[ P = P(F) = UP(\hat{F})V^T = U\hat{P}\hat{V}^T \quad (4.1) \]

where \( P \) represents the first Piola-Kirchhoff stress. In this equation, \( U \) and \( V^T \) represent the rotations obtained via the diagonalization of \( F \), in the manner of

\[ F = U\hat{F}V^T \quad (4.2) \]

where \( F \) is a 3x3 matrix which represents the deformation gradient for each element. For each of these \( P \) stress tensors (one for each tetrahedron), we compute the eigenvalues and store the largest eigenvalue together with its corresponding eigenvector. This eigenvalue will represent the stress value in its associated tetrahedron.

After each deformation step, we check whether there exist tetrahedra with stress values exceeding the threshold. If so, we search for the most likely fracture in the list of candidates. We are looking for the vertex from which we should start the fracture. This vertex must belong to the tetrahedron with the highest stress, the stress must exceed the threshold and, in addition, this vertex must be a surface vertex. We have added this requirement since we are only concerned with fractures starting on the surface of the object. Whether it is possible for the fracture to start inside the object and how this should be modelled, is beyond the scope of this project. If we do not find any vertices on the surface of the object fulfilling these conditions, we move on with the next
deformation step. If we do find a suitable vertex, we mark it as the origin of fracture and continue to the next fracture step, the propagation of the fracturing through the object. The propagation steps, which are also shown in Figure 4.9 are:

- **STEP 1: FRACTURE PLANE** The first step defines the plane which splits the object in 2 pieces, one lying on its left side and one its right side. We compute the equation of this plane by using the point \( S \) which defines the origin of fracture and the normal vector to this plane \( N_v \) which is the eigenvector associated with the highest eigenvalue of the stress tensor of the tetrahedron in which the fracture begins. The plane through the point \( S=(x_S, y_S, z_S) \) with normal vector \( N_v=(a, b, c) \) has the standard equation

\[
a(x-x_S) + b(y-y_S) + c(z-z_S) = 0 \tag{4.3}
\]

which can be simplified to

\[
a x + b y + c z = a x_S + b y_S + c z_S = \text{dot}(N_v, S) = d \tag{4.4}
\]

Therefore the fracture plane is stored as a vector with 4 components \((a, b, c, d)\). A visualization of this plane is provided in Figure 4.10.
4.2. DEFORMABLE MODEL

Figure 4.10: Fracture plane according to which the object split in two separate pieces

- **STEP 2: DUPLICATE VERTEX AND REASSIGN NEIGHBORING TETRAHEDRA**

Once we have the fracture plane, we duplicate the current vertex by copying its exact position but none of its connected edges and add it into the model. For all the tetrahedra connected to the original vertex, we check on which side of the fracture plane they lie. We refer to the two sides as left and right side. Each tetrahedron has 4 vertices, but one of them is our current fracture point which lies on the fracture plane. Therefore, we have to decide if the majority of the remaining three vertices lies on the right or on the left hand side. If one of the remaining three vertices happens to also lie on the fracture plane, we assume that it belongs to the left side, which effectively avoids having the same amount of vertices on each side. All tetrahedra on the left side of the plane are connected to the new duplicated vertex and therefore disconnected from the original one, while all on the right side, remain connected to the original vertex. A visualization of this process can be seen.
The vertices found to lie on the right hand side of the plane are coloured in pink while the ones on the left side in purple. This visualization is available only in debugging mode, since we do not expect users to need it.

**STEP 3: COLLECTING CRACK TIPS** After we have split the first vertex, we need to propagate the fracture further into the object. In order to ensure that the fracture follows alongside the fracture plane in a controlled manner, we create a collection of so called “crack tips”, terminology coined in the paper by Müller et al. [40]. From all the tetrahedra that were originally connected to the origin of the fracture vertex, we mark as crack tips all of their vertices which belong to both a tetrahedron on the right and on the left side. We use the data structure of a queue to which we add all crack tips as vertices that are supposed to be split during the fracture process. In Figure 4.12, we see a drawing showing which vertices become crack tips. In this example, we see two tetrahedra sharing a face and a vertex A. Vertex A is duplicated into vertex A*, and the two tetrahedra are split apart, which is visible in the right-hand side of the drawing. The distance between them is augmented in the drawing in order to ease the understanding of what is happening. The fracture plane is represented by the red line, to simplify the drawing, in reality the fracture plane is represented by the face ABC. After the split, vertices B and C belong to both a tetrahedron which lies on the right side of the plane as well as to a tetrahedron which lies on the left side of the plane. Therefore, B and C are chosen to be crack tips.

![Figure 4.12: Crack tip selection](image)

**STEP 4: SPLIT CHECK**

After each vertex split, we must check whether the object has already completely split into 2 pieces. To do so, we try to find a path from the split vertex to its duplicated vertex, see Figure 4.13. In order to search a path, a Depth First Search is applied to a graph structure which represents the mesh. Each node in the graph is a vertex and each edge between two nodes represents an edge in the mesh. If no such path can be found, then the object has in fact become two objects and the
4.2. DEFORMABLE MODEL

Simulation is updated to reflect this (Step 5). If a path has been found then, the next crack tip is popped from the queue and marked as current fracture origin. Then the process is repeated (Steps 1-4).

Figure 4.13: Checking if the substrate mesh has separate components

- **STEP 5: SPLIT AND UPDATE** If the object is split into two parts, two new deformable objects are created. The vertices and material properties for each of them are copied from the original object. Furthermore, the deformation integrator is updated, its internal state is copied into the new instances, so that the new pieces preserve their position, shape, and internal stress values. The vertices are renumbered starting from zero. Each new piece is given a new colour in the simulation in order to distinguish it as a separate object, see Figures 4.14c and 4.14. For practical purposes (keeping computation times low) and application reasons (no splitting into tiny pieces), we limit the total number of pieces which are possible to coexist in a simulation. If this limit is reached, the fracture process is disabled, and the simulation continues to deform the pieces but does not fracture them any more.

### 4.2.2 Partial fracture

The procedure described above will always cause the object to split into two pieces, once fracture has been initiated. It is technically possible to also stop the fracture based on some considerations and thus leave the mesh with a partial fracture. One example of such a fracture can be observed in Figure 4.15. We have chosen to disallow partial fracture for
CHAPTER 4. THE CHEWING PROCESS

(a) Fractured surface mesh  
(b) Fractured tetrahedral mesh  
(c) The cube tetrahedral mesh split into 4 disconnected pieces

Figure 4.14: An example of a common fractures where 2 or more pieces are created - each piece is given a different colour

our final simulation application in order to avoid instabilities in the numerical integrator. Partially fractured meshes experience a significant number of self-intersections, which need to be properly addressed before reasonable results can be expected.

Figure 4.15: The cubic model displaying partial fractures

4.3 Collision detection

In order to compute the deformations of the substrate, the application must first detect the collisions between the substrate and the teeth. These collisions must be frequently detected, after every deformation of the substrate and after every new position of the lower jaw, a new test must be performed. Since finer meshes which contain a significant number of triangles are better for the outcome of our simulation, the collision detection test must be as efficient as possible since we were trying to create an interactive simulation which runs in real time. We distinguish between two types of collisions which can occur in our simulation:
4.3. COLLISION DETECTION

- Collisions of substrate with teeth
- Collisions between substrate pieces

4.3.1 Collisions of substrate with teeth

In order to avoid bottlenecks at the collision detection step, we computed KD trees for the upper and lower jaws. The trees are precomputed before the simulation begins. Since the lower jaw suffers only a rigid motion which is composed from rotations and translations, it is not necessary to recompute the KD tree, a simple multiplication with the inverse of the transformation matrix is sufficient. The substrate surface mesh object is not represented with any bounding volume data-structure since it is dynamically changing its shape and it would require a recomputation of the data structure at every iteration. The collision detection mechanism is then straightforward, for every vertex of the triangular substrate mesh a ray is sent (upwards for the lower jaw and downwards...
for the upper jaw). If the ray intersects the jaw object once then this current vertex of the gummy bear is inside the jaw object. After iterating through all the vertices of the gummy bear we obtain a collection of vertices which lie inside the jaws. For these vertices we compute force vectors which will be later used to deform the object.

Chewing Forces For all the collision points a force vector is calculated as it is explained in Figure 4.17. We distinguish between two possible situations which occur during chewing:

- **Opening motion** The jaws are currently moving away from each other and therefore the substrate is currently experience a relaxation of constraints rather than an active

![Figure 4.17: Calculation of chewing force vectors depending on the status of the chewing](image)

(a) Derivation of forces during an opening motion

(b) Derivation of forces during a closing motion
pressure. In this case, we apply the forces as described in Figure 4.17a. In this example, vertices A and B of the substrate have been found to intersect the tooth which is represented by the yellow shape. We identify the points where the rays sent from A and B have intersected the tooth, namely A* and B*. We then proceed to create the forces as the vectors pointing from A* and B* to A and B respectively. This application of forces will ensure the substrate is constrained spatially by the tooth and adjust its shape so as to not intersect it any more.

- **Closing motion** The jaws are getting progressively closer to each other and they are actively chewing the substrate by applying pressure on it. In this case we compute the forces as explained in Figure 4.17b. In this example we see that vertices D and E have been found to intersect the tooth as it finds itself in the second chewing iteration. We want to apply a force on these two vertices which also captures the direction of motion of the lower jaw. In order to do this, we identify the points D* and E* where the rays sent from D and E intersect the tooth at its current position. Then we compute the positions that these two vertices, D* and E*, had when the tooth was in its previous position, chewing iteration 1. The direction of the forces we want to apply to vertices D and E are found, as shown in the picture, by creating a vector from the previous position of D* to its current position and the same for E*. These two forces are then applied to vertices D and E. This ensures the substrate will capture the movement of the lower jaw and deform accordingly. Naturally, we must keep in mind that the upper jaw does not move and therefore it does not have a previous position. Therefore, the forces derived from intersections with the upper jaw are always pointing downwards, their orientation is not adjusted based on the previous position of the lower jaw.

In both cases the force vectors are normalized and then their magnitude is assigned based on the current EMG value which is derived from the chewing file. More information about the EMG value can be found in Section 3.3.3.

In Figure 4.18, the chewing forces are visualized with red arrows which point at the vertices which intersect the teeth. The orientation of these arrows is in accordance with the orientation of the forces being applied. These geometry objects can be turned off and on at the user's discretion. They are also especially useful to evaluate the effect that the application of these forces has on the local deformation. They are updated after every deformation of the substrate.
CHAPTER 4. THE CHEWING PROCESS

Figure 4.18: Visualization of active chewing area with the help of colours maps and red arrows which represent the chewing forces being applied to the substrate.
4.3. COLLISION DETECTION

4.3.2 Collisions between substrate pieces

When the substrate has been fractured into two or more pieces, those pieces start to intersect each other as they are pushed together by the closing jaws. In order to correct this behaviour, we introduce repelling forces which keep the pieces apart. In order to create these forces, we must first identify all collision events between pieces and record all surface vertices which intersect the mesh of another piece. Since the substrate pieces are deformable objects that consistently changes shape and position, a KD tree approach as used above would require us to recompute the KD tree after each deformation, which is rather costly. Instead, we propose to use an approach based on casting rays, which can be computed efficiently since the pieces are relatively small and consist of a limited number of surface vertices. Intersection tests are performed pairwise between each pair of two substrate pieces. For each surface vertex of one piece we shoot a ray towards its own centroid. A collision is detected, if this ray intersects the second piece. Figure 4.19a illustrates the ray casting for the two pieces represented by the green and orange shapes. The squares in the centre of the shapes represent their centroids. A ray is shot from vertex A and vertex B of the orange piece towards its own centroid. None of the two rays intersect the green shape, i.e., we do not report a collision. A collision event can be observed in Figure 4.19b, where the ray cast from vertex A intersects the green shape, the intersection point being marked with a red cross and vertex A is marked.
CHAPTER 4. THE CHEWING PROCESS

(a) Ray-based collision detection for two deformable pieces

(b) Detection of collision event between two pieces

(c) Computation of repelling force vector resulting from collision

(d) Deformation caused by repelling force onto vertex

Figure 4.19: Collision detection between pieces of substrate used to create repelling forces
4.3. COLLISION DETECTION

Figure 4.20: Fractured pieces shown in context with the chewing scene

Figure 4.21: Repelling forces pushing two substrate pieces apart
Figure 4.22: Four substrate pieces interacting with each other, shown at subsequent deformation steps (the numbers under the images indicate the deformation step)
4.3. COLLISION DETECTION

**Repelling forces**  Once we know which vertices are causing collisions, we compute repelling forces which act on the piece such that the collision is resolved. For each intersecting vertex, a force is applied which has a constant magnitude and a direction pointing from the second piece’s centroid towards the centroid of the first piece. In Figure 4.19c, we can see the vector unifying the two centroids, being applied as the repelling force acting on vertex A. After all the repelling forces are set, a respective deformation step can be, see Figure 4.19d. An example of the interacting substrate pieces is shown in Figure 4.21. An intersection is found between the green and the red piece (highlighted with a different colour). The centroids are displayed by the smaller inner cubes are calculated and the repelling forces are represented by the blue arrows. The two pieces are pushed apart. The same process between four different pieces can be observed in Figures 4.22. The pieces are shown in context together with the whole chewing scene in Figure 4.20.

![Diagram](image)

**Figure 4.23**: Recapitulation of all forces acting in the scene

![Diagram](image)

**Figure 4.24**: Visualization of the point towards which the tongue force attracts the substrate

**Tongue force**  In Figure 4.23 we can see a summary of all forces acting in the scene. Besides the already mentioned chewing and repelling forces, one can notice in the picture,
a third force called the *Tongue force*. Since our simulation does not model the boundaries of the mouth, such as the walls of the mouth or the tongue, the substrate can easily fly away from the scene, which in reality would not happen. In the regular chewing process, the tongue brings back the food and places it on the teeth repeatedly. In order to simulate this process, we have added an attraction force towards the teeth and named it the tongue force. It constantly acts to draw the deformable pieces towards a predefined position on the lower jaw, which can be observed in Figure 4.24. This force is constant and acts on all vertices equally. The application of this force ensure that there will be collisions between the substrate and the teeth throughout the chewing simulation, as it would be expected to happen in a real chewing process.
Chapter 5

Results and Discussion

5.1 Deformable substrate

In this section, we evaluate the behaviour of the deformable model when applied to meshes of different shapes. We believe it is interesting to explore how the Invertible FEM ( - the method we chose for our deformable substrate, see Section 4.2) behaves for a regular geometric shape, like the cube, as well as for the more complex gummy bear and spheric models. We are mostly interested to see how the deformations develop as the chewing simulation progresses. In addition, this evaluation also aims at investigating how stable the method is for the different shapes.

5.1.1 Gummy bear

Figures 5.1 and 5.2 display a series of deformations to the gummy bear model used in first version of the gummy bear chewing application, which did not include fracture. However, the deformable model representing the substrate was the same and one can notice the severe deformations which are caused by the teeth. Figure 5.1 shows a chewing progression overview while Figure 5.2 shows the same simulation while focusing on the gummy bear model in order to display the marks left by the teeth.

In Figure 5.3 our GummyL model is being deformed by subsequent chewing iterations. One can notice that this model is smaller in size than the one in Figure 5.1 and 5.2 because we have tried to create this model to respect a more realistic teeth to gummy bear size ratio. Being also the finest gummy bear model, one can notice the minute details of the deformed surface mesh in Figures 5.3e and 5.3f. In Figure 5.3g the gummy bear is shown in focus together with the restorations in order to visualize the impact that the shape of the restorations had on the deformation. Finally Figure 5.3h shows the gummy bear together with the complete lower jaw, to which the restorations belong to.
We notice that gummy bear model deforms according to expectations and it does not accumulate obvious errors as the deformations become more severe.
Figure 5.2: Deformations of the gummy bear model, contacts points are shown in blue.
Figure 5.3: Deformations of the GummyL model
CHAPTER 5. RESULTS AND DISCUSSION

5.1.2 Cube

Figure 5.4 shows a series of deformations of the Cube40 model, the finest mesh and consequently the one which is able to capture the detailed imprint of the teeth which are pressing on it. In Figure 5.4a, in addition to the deformations, one can also notice the color maps which highlights the area of contact between the substrate and the teeth. The same area is shown in Figure 5.4b, with a focus on the deformation visible in the surface mesh. It is obvious from this picture that the sharp peaks of the restorations have left their imprint into the soft cubic substrate. The tetrahedral mesh of the same cube is shown in Figure 5.4c together with a colour map highlighting the areas experiencing intense stress, which coincides with the areas displaying deformations. In Figure 5.4d we observe the same Cube40 model, but at a later stage in the simulation, when it has become more severely deformed.

![Figure 5.4: Deformations of the Cube40 model](image)

It was also interesting to see how a lower resolution mesh deforms. Figure 5.5 shows the Cube10 mesh deforming over time. The cause of these deformations is the lower jaw
touching the cube model and pushing it upwards. The model, being very coarse, moves upwards without capturing the imprint of the teeth. For this progression the substrate encountered the upper jaw after its initial upwards movement and was therefore pushed back downwards, effectively bouncing up and down between the two jaws.

![Images of deformation steps](image)

**Figure 5.5:** Subsequent deformation steps of the Cube10 model - surface mesh view

### 5.1.3 Sphere

We also tested our deformable model with a spheric model or different resolutions. In Figure 5.6 the SphereL model is being pressed between the two jaws and consequently squeezed into a flat shape. A colour map is also visible which highlights the areas experiencing a high degree of stress. This behaviour, in accordance with our expectations, enforces the validity of the deformable model that we use and confirms that our implementation is flexible and suitable to use for a number of different substrate shapes. Figure 5.7 shows the SphereM model being progressively chewed. Even this coarser model captures the shape of the teeth impringing themselves upon it, as it is visible in Figures 5.7a through 5.7f. Figure 5.7i shows the substrate being pushed upwards towards the upper jaw and intersecting it. A colour map is visible which displays in red the intersection points between the substrate and the tooth belonging to the upper jaw.
Figure 5.6: Subsequent deformation steps of the SphereL model - tetrahedral mesh
Figure 5.7: Subsequent deformation steps of the SphereM model
5.1.4 Discussion

For all the chosen models, we notice that the deformable method is stable and is suitable for the complex deformations that the teeth inflict on the substrate. We did not see any errors being accumulated as the chewing simulation moves forward and we observed the models behaving according to our expectations, reacting appropriately to the forces being applied onto them.

5.2 Chewing Simulation

Usually at the end of an implementation effort it is important to evaluate the success of the software. In our case, it is difficult to evaluate the results of the chewing simulation in a completely objective manner. The truth is, we actually do not know what the result of chewing should be exactly. We do expect a certain behaviour throughout the simulation which we evaluate by visually analysing a set of progression screenshots. For example we expect to see the correct areas of the substrate being identified as intersecting the teeth and the resulting collision colour maps to be correctly functioning as we can see in Figure 5.8. In Figure 5.8d we see the resulting forces being shown, with a closer focus on the arrow elements in Figure 5.8e. The resulting deformation after the application of these forces can be observed in Figure 5.8f. Since what we are observing here looks reasonable and is in accordance with our expectations (and what we have observed in many other scenarios) we can reach the conclusion that this whole process is working as expected.

In Figure 5.9b we notice the stress colour map which shows that the deformed area is also the one which is experiencing the highest stress, which is in tune with our expectations. Figures 5.9a, 5.9c and 5.9d are showing the deformations caused by the teeth at subsequent chewing iteration steps. We can observe that the imprint on the substrate becomes deeper and the affected area progressively enlarges as more tooth surface comes in contact with the substrate.

The other important process to assess is the fracture process and the repelling mechanism which acts to prevent substrate pieces from intersecting each other. What interests us is to see that the fracture occurs and that it successfully breaks the object into separate pieces. We can observe this process in Figures 5.10a and 5.10b where the substrate has broken into multiple pieces and is shown from different viewpoints. In addition we are interested to see that the pieces are separate objects, which can be observed by the different colours they have. In Figure 5.10d we see the substrate breaking apart in 2 pieces which then interact with each other. These pieces are touching in one point and we are interested to see if there is a repelling force acting there. We see this force in Figure 5.10e, where the substrate has been made transparent so that we can observe the
forces which are acting inside of it. The same two pieces are visualized again in Figure 5.10f together with the teeth in order to assess if the scene as a whole makes sense. We see the pieces are positioned between the teeth and their shape looks reasonably fitting to their spatial constraints.

After looking at the individual components of the simulation, we have to focus on the final result of the simulation. In the end, all these mechanisms can be working correctly, but due to other missing elements or due to incorrect assumptions, the final chewing results might not be reasonable. First of all we need to know what we expect to see. In Figure 5.11 we can observe the remains of a gummy bear after it has been chewed by a person. This does not represent the ground truth by any means, it is just presented here as a coarse reference, an example of what a reasonable result could look like. What interests us is that there are multiple pieces, their surface looks rather rough and the shape of the original gummy bear has been completely lost.

In the following Figures 5.12, 5.13, 5.14 and 5.15, we observe the results of a few selected simulations, all ran in automatic mode with 34 total chewing iterations. We present the substrate from two viewpoints, one showing the whole scene and one focused on the substrate pieces. With these figures, which were saved during the run of the automatic simulation, we want to show the state of the substrate pieces at the end of the run and check if the pieces look reasonable, if they are intersecting each other and if they are situated in the active chewing area. We notice that our expectations have been largely met. We do not have as many pieces as we have in the result of the real chewing process, the reason for this being that we artificially limited the amount of pieces which can be created. Due to the complexity of the calculations, we have decided that we can only support a maximum of 5 pieces at the moment. For a complete chewing progression, observed from multiple viewpoints, see the Appendix 7. If we did not use this threshold to limit the maximum number of pieces, the substrate would continue to split, the final result being more similar to that in Figure 5.11.

In Figures 5.14 and 5.15 we notice that there is an extra piece visible in the complete scene, namely the purple one in both cases, which seems to be missing in the focused view. The reason for its absence is that the focused view is zoomed in too much and the purple piece is too far away from the other pieces and is therefore currently out of view. Since these screenshots are taken automatically, it is sometimes difficult to calibrate the
zoom properly. Moreover, in Figure 5.15, the purple piece looks like it almost left the active chewing site. This in particular is a case when in real life, the tongue would bring this piece back. In the context of the simulation, should the simulation have continued, this piece would have been slowly attracted back to the teeth by the tongue force. We have not made this force strong enough to compete with the repelling forces because it is an artificial force and we did not want to influence the simulation too much.
5.2. CHEWING SIMULATION

(a) The cube collides with the teeth
(b) The collision area is highlighted on the teeth
(c) The collision area is highlighted on the substrate
(d) The resulting forces are represented by arrows
(e) Close focus on the arrow elements
(f) The resulting deformation is computed

Figure 5.8: Observing the collision detection, force computation and deformation mechanisms
(a) The restorations are hidden to show the imprint on the substrate on iteration 3

(b) The stress colour map is showing the active areas

(c) Chewing iteration 4

(d) Chewing iteration 7

Figure 5.9: Observing the deformation caused by the teeth and the stress colour map
5.2. CHEWING SIMULATION

(a) The substrate fractured in multiple pieces
(b) The substrate tetrahedral map is brought into focus
(c) The repelling forces are represented as blue arrows
(d) Two substrate pieces touching in one point
(e) The substrate is made transparent in order to see the repelling force
(f) The two substrate pieces resting on the teeth

Figure 5.10: Observing the fracture process together with the repelling forces between pieces
CHAPTER 5. RESULTS AND DISCUSSION

Figure 5.12: Result of the chewing simulation with Cube40, using the original restorations and starting point 1, see Figure 5.32

Figure 5.13: Result of the chewing simulation with Cube40, using the smooth restorations and starting point 1, see Figure 5.32
5.2. CHEWING SIMULATION

Figure 5.14: Result of the chewing simulation with Cube40, using the original restorations and starting point 3, see Figure 5.32

Figure 5.15: Result of the chewing simulation with Cube40, using the smooth restorations and starting point 3, see Figure 5.32
5.3 Interactivity and speed

In this section we first discuss the overall runtime for a complete simulation which uses different substrate meshes, followed by a discussion of the runtime for the individual components of the simulation. All the simulations were run on a DELL Alienware 17 laptop under Windows 8.1 64 bit, Intel Core i7 CPU, 2.8GHz (8 CPUs), 16 GB RAM, GPU Nvidia GeForce GTX 765M.

5.3.1 Overall runtime

For our finest mesh - Cube40 (see Table 3.4), the chewing simulation iterating through 35 positions for the lower jaw, takes between 3:30 and 4 hours, as it can be seen in Table 5.1.

<table>
<thead>
<tr>
<th>Restorations</th>
<th>Start Point</th>
<th>Runtime</th>
<th>Deformation Steps</th>
<th>Seconds per deformation step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Restorations</td>
<td>1</td>
<td>3:44:27</td>
<td>1376</td>
<td>9,7871</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3:42:34</td>
<td>1377</td>
<td>9,6979</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4:03:17</td>
<td>1416</td>
<td>10,3086</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4:07:20</td>
<td>1456</td>
<td>10,1923</td>
</tr>
<tr>
<td>Smooth Restorations</td>
<td>1</td>
<td>3:23:16</td>
<td>1339</td>
<td>9,1083</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3:36:05</td>
<td>1376</td>
<td>9,4222</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3:42:56</td>
<td>1376</td>
<td>9,7209</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3:43:21</td>
<td>1377</td>
<td>9,7320</td>
</tr>
</tbody>
</table>

Table 5.1: Runtimes for simulations iterating through 35 consecutive positions for the lower jaw (fracture disabled)

The amount of time that it takes depends on the other parameters as well, as each new position for the lower jaw triggers multiple deformation iterations which are stopped either when a maximum number is reached or when there are no more contact points with the teeth. The amount of deformation iterations required at each position, therefore depends on all the other parameters, such as the density of the substrate, the starting position, the shape of the teeth, etc. In Table 5.1, we can observe that the simulation requires on average 10 seconds per deformation step, for a setup which uses Mesh 1 and has fracture disabled. In addition, whether fracture is turned on or off, also drastically influences the overall time required to run through a simulation, since the existence of multiple substrate pieces can significantly slow down the simulation. However, as it is shown in Table 5.2, the runtimes for the same setup as above, with fracture enabled are in fact lower than expected. When the mesh breaks into pieces early on in the simulation, as is the case for the runs using the original restorations, some of the pieces may partially
slide outside of the maximum pressure area between the teeth, therefore producing less intersection events, which explains the reduced time required to complete the simulation. Overall, it was always needed to make compromises in order to satisfy the quality / time constraints balance. We have tried using coarser meshes and achieved truly interactive frame rates. However, we have observed that the coarser the mesh, the less reliable the results are. Since the restorations and teeth have subtle hills and creases in their surfaces, we should not use a mesh which is too coarse because the simulation would not be able to capture the necessary amount of details.

<table>
<thead>
<tr>
<th>Restorations</th>
<th>Start Point</th>
<th>Runtime</th>
<th>First fracture at iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>1</td>
<td>2:24:35</td>
<td>18</td>
</tr>
<tr>
<td>Restorations</td>
<td>3</td>
<td>1:33:05</td>
<td>15</td>
</tr>
<tr>
<td>Smooth</td>
<td>1</td>
<td>2:37:43</td>
<td>27</td>
</tr>
<tr>
<td>Restorations</td>
<td>3</td>
<td>2:27:58</td>
<td>23</td>
</tr>
</tbody>
</table>

Table 5.2: Runtimes for simulations progressing through 35 chewing iterations with fracture enabled

In addition to the overall runtime, we are also interested to measure the runtime for the components which make up the simulation. We are mostly interested in measuring the time required by the collision detection algorithms, but it is also interesting to see how much time each chewing iteration and deformation step takes for each mesh.

### 5.3.2 Collision detection between teeth and substrate

In Charts 5.16, 5.17 and 5.18 one can see how the runtime for the step which computes the collision detection between the substrate and the teeth, correlates with the number of contact points. As expected, the collision detection step takes longer for a finer mesh. We notice that for the coarser mesh, Cube20, all collision detection steps take between 0 and 0.25 seconds while for Cube30, the maximum runtime for a step is increased to 0.5 seconds. For Cube40, the runtime for a single collision step can be as high as 4 seconds. The collision step which is timed here includes the collision of the substrate with the restorations and the upper and lower jaw. The difference between the maximum runtimes among the models can be observed in Chart 5.19.

In Table 5.3, we gathered the average runtimes for the whole simulation for the 3 meshes, together with the average number of contact points. We notice that for Cube30 we had a lower number of collision detection steps, this can happen since collision detection steps are performed after each deformation step. The total number of deformation steps is unpredictable, as the simulation performs deformation steps until it detects no more contact points for the current iteration. Given that for Cube40 we had in total approximately 1000 collision steps, each taking an average of 1.45 seconds, we can see that only the collision detection took 25 minutes out of the total simulation time of
approximately 3 hours. In addition, in this table we can also see how much time a deformation step on average for each model. A deformation step includes multiple deformation runs of the object together with collision detection, as measured here. Deformation steps are performed for each chewing iteration until no more collisions with the teeth are found.

<table>
<thead>
<tr>
<th>Mesh Name</th>
<th>Avg coll. detection runtime (s)</th>
<th>Avg def. steps runtime(s)</th>
<th>Avg nr contact points</th>
<th>Nr of coll. detection steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube20</td>
<td>0.078</td>
<td>1.28</td>
<td>123.98</td>
<td>991</td>
</tr>
<tr>
<td>Cube30</td>
<td>0.21</td>
<td>3.49</td>
<td>731.2</td>
<td>527</td>
</tr>
<tr>
<td>Cube40</td>
<td>1.44</td>
<td>10.55</td>
<td>5099.1</td>
<td>1031</td>
</tr>
</tbody>
</table>

*Table 5.3: Average runtimes in seconds for collision detection between substrate and teeth*
Figure 5.17: Collision detection runtimes for Cube30, fracture disabled

Figure 5.18: Collision detection runtimes for Cube40, fracture disabled
CHAPTER 5. RESULTS AND DISCUSSION

Figure 5.19: Comparison among the runtime for the collision detection step between the teeth and substrate, for Cube20, 30 and 40 (see Table 3.4)

<table>
<thead>
<tr>
<th>Mesh Name</th>
<th>Nr of surface triangles</th>
<th>Nr vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube20</td>
<td>5292</td>
<td>10648</td>
</tr>
<tr>
<td>Cube30</td>
<td>11532</td>
<td>32768</td>
</tr>
<tr>
<td>Cube40</td>
<td>20172</td>
<td>74088</td>
</tr>
<tr>
<td>Upper jaw</td>
<td>36774</td>
<td>19400</td>
</tr>
<tr>
<td>Lower jaw</td>
<td>41342</td>
<td>21900</td>
</tr>
<tr>
<td>Restoration1</td>
<td>2516</td>
<td>1284</td>
</tr>
<tr>
<td>Restoration2</td>
<td>2711</td>
<td>1384</td>
</tr>
<tr>
<td>Restoration3</td>
<td>4257</td>
<td>2165</td>
</tr>
<tr>
<td>Restoration4</td>
<td>4022</td>
<td>2045</td>
</tr>
<tr>
<td>All restorations</td>
<td>13506</td>
<td>6878</td>
</tr>
<tr>
<td>Jaws and restorations</td>
<td>91622</td>
<td>48178</td>
</tr>
</tbody>
</table>

Table 5.4: Nr of surface triangles for the meshes involved in the collision detection between teeth and substrate
5.3. INTERACTIVITY AND SPEED

5.3.3 Collision detection among substrate pieces

In order to measure the runtime for the collision detection between substrate pieces, we set a very low fracture threshold to ensure that the simulation will create the maximum number of pieces, which we set to 4. In Charts 5.20, 5.21 and 5.22 we observe the runtime in seconds for the collision detection among pieces, for Cube20, 30 and 40. Although the average runtimes, which are found in Table 5.5, are higher for the higher resolution meshes, we observe that the difference is not so large as for the teeth collision step. These runtimes measure one collision detection run which checks the collision between each pair of pieces. We also notice that the runtimes for this step remain fairly constant throughout the simulation, the reason for this being that the same set of operations is performed for each pair of substrate pieces, regardless of how close or far apart they are, since no KD trees are used in this case. For Cube40 the collision detection between pieces was done 993 times, for an average of 1.07 seconds each which amounts to approximately 17 minutes. This is acceptable given that the runtime for the whole simulation was 4 hours.

Figure 5.20: Collision detection runtimes for Cube20, fracture enabled
Figure 5.21: Collision detection runtimes for Cube30, fracture enabled

Figure 5.22: Collision detection runtimes for Cube40, fracture enabled
5.3. INTERACTIVITY AND SPEED

<table>
<thead>
<tr>
<th>Mesh Name</th>
<th>Avg coll. detection runtime (s)</th>
<th>Nr of coll. det. steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube20</td>
<td>0.588</td>
<td>674</td>
</tr>
<tr>
<td>Cube30</td>
<td>0.78</td>
<td>211</td>
</tr>
<tr>
<td>Cube40</td>
<td>1.07</td>
<td>993</td>
</tr>
</tbody>
</table>

Table 5.5: Average runtimes in seconds for collision detection between 4 substrate pieces

5.3.4 Comparison with the state of the art

It is difficult to make a direct comparison with runtimes for other collision detection algorithms due to the different hardware the algorithms are run on and due to our special circumstances. We did not only need a fast collision detection mechanism, we needed one that also allowed us to compute the forces that were required for the simulation, directly from the results.

This being said, in the paper by Redon et al. [60], a rough estimate is given on how fast collision detections between rigid bodies could be expected to be. They mention a setup in which they have a car door consisting of 16 000 triangles and a car skeleton consisting of 29 000 triangles. The collision detection between the car door and the car skeleton is achieved at interactive speeds, no delay being noticeable to the human eye. Given that our jaws and restorations have together 91622 triangles and our Cube30 has 11532 surface triangles, we could find this comparison to be fair. An average collision detection step between our jaws and restorations and our substrate mesh Cube30 took 0.2 seconds, which can also be considered an interactive speed. In addition, in these 0.2 seconds we also compute the forces from the collision points, and are directly ready to apply them to the deformable substrate afterwards. However, we must keep in mind that the application experimenting with the door and car skeleton was run on a 1 GHz Pentium PC from 2002, while we ran our tests on a significantly more powerful machine, a DELL Alienware laptop with an Intel Core i7 CPU, 2.8GHz (8 CPUs). In addition we must keep in mind that both the car door and the car were rigid objects while we are dealing with a deformable object which is constantly changing both its shape and its position in space.

Therefore we turn to the work by Teschner et al. [75] which discusses collision detection performances especially for deformable objects. In this paper a series of experiments are discussed, which measure the average runtimes for a collision detection step between 2 or more deformable models. The collision detection algorithm used belongs to the spatial partitioning family, like our KD trees, but is focused on spatial hashing and presented originally in a previous paper by Teschner et al. [74]. A setup of consisting of 2 deformable objects having in total 20514 tetrahedra and 5898 vertices measured an average of 70 ms for a collision detection step between them. Our Cube40 has a total of 74088 vertices and 413 526 tetrahedra and an average collision detection step
between the 4 pieces making up the cube took 1.07 seconds. A collision detection step between the 4 pieces of Cube20 which has 55566 tetrahedra and 10648 vertices took, in our simulation, 0.588 seconds. Another setup in the same paper by Teschner et al. [75], used 100 objects having a total of 50 000 tetrahedra and 24 200 vertices, which took an average of 172.5 ms per collision detection step. This scenario resembles more closely our Cube20 simulation, however we had only 4 pieces at a time. Even though the runtime obtained in the paper of 172.5 ms or 0.17 seconds is significantly smaller than our average runtime of 0.5 seconds, we also derived the forces which are produced by the collisions in these 0.5 seconds. A collision detection which simply identifies which triangles are intersecting others is useless for our simulation, as we need to know not only which triangle intersects which object, but which force the object is inflicting on the object it is intersecting. The work by Teschner et al. concludes that a simulation consisting of up to 20k tetrahedra and 6k vertices can be performed interactively on a standard PC, without giving more details about the hardware.

5.4 Comparing Restorations

In addition to the modelling of the chewing of one fixed configuration comprised of teeth and substrate, the purpose of the chewing simulation is to allow comparison between different restorations. In order to achieve this goal, the simulation can be run in automatic mode, which means the user only chooses the restoration to be used and everything else remains fixed. The same chewing file is read, which contains the iterative positions of the lower jaw. The chewing motion is applied to the substrate and its consequent deformation and fracture is recorded. In addition, a number of parameters are recorded in a log file, which could be of interest when assessing the efficiency of the mastication process. It contains the maximum stress, the number of contact points, and the EMG value at each iteration step of the simulation. The main differences between restorations are expected to be noticed in the number of contact points it has with the substrate and the resulting stress that it inflicts in the object. Since the maximum stress value is used to fracture the object, by monitoring this value, we can infer when the object would fracture given the teeth configuration.

A gallery of screenshots is saved for each simulation run, which captures the scene at each chewing iteration step, from different viewpoints. An example of a screenshot gallery is shown in Figure 5.25. In order to test the results that different restorations produce, we have taken the restorations shown in Figure 5.23 and smoothed them to varying degrees using the HC Laplacian Smoothing option provided by MeshLab [53] following the ideas by Vollmer et al. [79]. These smoothed restorations, see Figure 5.24, are not supposed to represent valid restoration models, but their purpose is to test in a controlled experimental set-up if and how the shape of the restorations influences the
5.4. COMPARING RESTORATIONS

Figure 5.23: Sample restorations created with Cerec which fit on top of a prepared area of the lower jaw

Figure 5.24: Smoothed restorations

overall chewing process.
5.4.1 Stress values and contact points

For multiple given restorations, we can compare the simulation results based on the stress values that were recorded, the number of contact points, and the shape of the substrate at different iteration steps. Since there are several parameters which can influence the results, we designed a few different tests, each focused on a specific variable. The main test though, for our chewing simulation, is to set all parameters, run the simulation once for each of the considered restorations, and assess the results. In Chart 5.26, a comparison is shown between the maximum stress values recorded at each step in the deformable object. The blue line corresponds to the simulation run with a set of restorations created with CEREC [1] and the orange line corresponds to a set of test restorations. The test restorations which can be seen in Figure 5.24, have been created by smoothing the original restorations, as described in the previous section.

From the chart we can observe a different pattern for the maximum stress value. The original restorations caused on average higher stress values than the smoothed versions. For the same simulation pair, Chart 5.27 shows a comparison between the amounts of contact points recorded at each step. For the original restorations a smaller number of contact points with the teeth is observed throughout the simulation due to the fact that a smoother area intersects the substrate in a larger number of points when compared to the original surface. In other words, sharper teeth have less contact points with the substrate. In order to compare the stress values properly, the fracture option was turned off for this example. The stress values cannot be directly compared any more after the deformable substrate has split into multiple pieces, as the stress is reduced by the fracture
and afterwards each piece has its own stress values. The end result of the mesh can be observed in Figure 5.28a and Figure 5.28b. We can observe that the sharper teeth have left a stronger imprint into the mesh when compared to their smoother counterparts.
Figure 5.27: Comparison of number of contact points between original and smoothed restorations

(a) Tetrahedral mesh of substrate deformed by original restorations  
(b) Tetrahedral mesh of substrate deformed by smooth restorations

Figure 5.28: Comparison of deformations caused by different restorations, after the same amount of chewing iterations
5.4. COMPARING RESTORATIONS

5.4.2 Comparing fracture events

We are also interested to see how the fracturing process differs when different restorations are used in the chewing process. In order to analyse this, we have conducted several simulation runs for pairs of original and smoothed restorations. For each simulation run, we placed the substrate at a different starting position on the restorations atop the lower jaw, then ran the simulation twice, once with the original restorations and once with the smoothed restorations. For a given starting position, we can then compare the results by noticing the differences in the substrate shape around the fracture event.

![Original Restorations](image1)

![Surface Mesh Iteration 18](image2)

![Tetrahedral Mesh Iteration 18](image3)

![Fractured Surface Mesh Iteration 19](image4)

**Figure 5.29:** Original restorations breaking up substrate into multiple pieces when the lower jaw moves from position 18 to position 19. Multiple deformation steps are performed between the two iterations shown in the images.

For example, Figure 5.29 illustrates how the teeth fitted with the original restorations break up the substrate into multiple pieces, while Figure 5.30 shows that the smoothed restorations (for the same simulation set-up) just produce further deformations in the substrate. One iteration represents one position for the lower jaw, therefore what the two figures illustrate, is that the smoothed restorations did not manage to split the substrate while the original restorations have done so.

Figure 5.31 shows that the smooth restorations also produce a fracture eventually, but only at a later point in time, meaning the smooth restorations required the jaws to be closer together in order to break up the substrate. One possible reason for this is that the original restoration models are larger in size and therefore put more pressure earlier on
the substrate while the smoother, smaller restorations reach the critical pressure point at a later time point. Another possible reason is that the original restorations, being sharper, inflict stronger forces in the model which cause it to fracture sooner, effectively piercing its surface, as opposed to the smoother teeth for which the chewing force is more spread out.

**Figure 5.30:** Substrate deformation caused by smoothed restorations
5.4. COMPARING RESTORATIONS

5.4.3 Comparing starting positions

The purpose of the next test was to identify if the starting position of the substrate influences the differences observed in the collected stress values. In order to gather these data, the substrate has been placed in four different starting positions as shown in Figure 5.32, fracture has been turned off, and the simulations were run in pairs of smooth/original restorations.

The material properties of the substrate were fixed. Chart 5.33 and Chart 5.34 show the results for the different starting positions illustrated in Figure 5.32. We observed that, although there are some variations in the number of contact points, the stress values recordings are consistently showing higher stress levels for the original restorations. In Chart 5.34, we observe that for the starting position 4, the contact points become
close to 0 for the original restorations, for a portion of the simulation. The illustrated that the shape of the restorations influences the manner in which the substrate moves during the chewing process. A small number of contact points means that the substrate was temporarily pushed away from the teeth, due to the chewing forces it experienced. During these iterations, the substrate was still deforming, but only under the influence of its accumulated internal stress, basically slowly returning to its original shape, while being at the same time attracted back towards the teeth by the tongue force. Once the substrate is having again contact with the teeth, we notice that the amount of contact points increases again in the plot. From these measurements we can conclude that, while the starting point makes a difference in the results, as it was of course to be expected, the influence of the shape of the restorations is consistently felt. Therefore, we suggest that the substrate model can be placed approximately on top of the restorations in order to observe the influence their shape has on the chewing process.

Figure 5.33: Maximum stress values recorded for different starting points of substrate
5.4. COMPARING RESTORATIONS

5.4.4 Comparing material properties

This test aimed at comparing how the stress values incurred by the substrate are influenced by the material properties of the substrate. For this purpose, we have chosen a fixed starting point, a fixed set of restorations, and varied the material properties of the substrate, namely the mass density and the Young’s modulus value. Table 5.6 shows the different material properties which were applied to the model as well as the average value for the maximum stress encountered in the model as it progressed through the chewing process.

<table>
<thead>
<tr>
<th>Material ID</th>
<th>Material Density</th>
<th>Young’s Modulus</th>
<th>Average Max Stress Original Restorations</th>
<th>Average Max Stress Smooth Restorations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200000</td>
<td>200000</td>
<td>414844.8782</td>
<td>333981.5887</td>
</tr>
<tr>
<td>2</td>
<td>200000</td>
<td>400000</td>
<td>660732.4189</td>
<td>594570.4866</td>
</tr>
<tr>
<td>3</td>
<td>400000</td>
<td>400000</td>
<td>638110.878</td>
<td>556360.6359</td>
</tr>
<tr>
<td>4</td>
<td>500000</td>
<td>400000</td>
<td>642439.4847</td>
<td>552223.3055</td>
</tr>
<tr>
<td>5</td>
<td>700000</td>
<td>400000</td>
<td>604539.5895</td>
<td>553032.8801</td>
</tr>
</tbody>
</table>

Table 5.6: Average maximum stress values recorded in substrates with different material properties

We can observe that on average the stress values were higher for the original restorations, for all the different material combinations. Chart 5.36 shows the recorded stress values for
all of these materials and we notice a trend for the original restorations to produce higher values. It is also interesting to notice how varying the material properties influences the stress values even when the same set of restorations is used, as it is shown by Chart 5.35. We notice that the higher stiffness (Young’s modulus) is associated with higher stress, which is to be expected. In Chart 5.35, Material 1 has the smallest Young’s modulus value and it is visible that it recorded smaller stress values when compared to the other materials. This difference is even more clear in Table 5.6. From this test we can infer that even though the material properties affect the stress values recorded inside the substrate, we can notice a trend across all tested materials, to consistently experience elevated stress when the original restorations are used, as opposed to when the smoothed restorations are used. A preliminary conclusion is reached that the differences noticed when varying the shape of the restorations, are not an artefact of the material properties.

Figure 5.35: Stress values recorded in substrates with different material properties
5.4. COMPARING RESTORATIONS

Substrate material 1

- Stress Original Restorations
- Stress Smooth Restorations

Deformation step

Maximum stress scale
Figure 5.36: Comparison between recorded stress values for different material properties
### 5.4.5 Comparing restoration smoothness levels

Moreover, we wanted to see how different levels of smoothness in the restoration models influence the result of the chewing. For this purpose, we have chosen a fixed starting point, fixed material properties, and varied the restoration models. An overview of the models with 6, 16, 28 and 40 smoothing iterations used for this test can be seen in Figure 5.37.

![Figure 5.37: Original restorations smoothed by repeatedly applying Laplacian smoothing operation [79]](image)

In Chart 5.38 we can observe a comparison of the number of contact points recorded during the standard simulation run for restorations of slightly different shape. The smoother restorations record on average more contact points with the substrate, which is to be expected. In Chart 5.39 and Chart 5.40 the stress values recorded for these doctored restorations are compared with the stress values recorded for the original set of restorations. Overall the chart lines follow a similar pattern, with some small differences for the restorations smoothed 6 and 16 times and somewhat larger differences for the restorations smoother 28 and 40 times. This test shows that the more the restorations are smoothed, the more different their influence in the simulation is compared to the original restorations.
5.4. COMPARING RESTORATIONS

**Figure 5.38:** Contact points for different restorations

**Figure 5.39:** Comparing stress values recorded in restorations with 6 and 16 smoothing iterations
Figure 5.40: Comparing stress values recorded in restorations with 28 and 40 smoothing iterations
5.4. Comparing Restorations

5.4.6 Comparing substrate model resolutions

Finally, we wanted to investigate how the resolution of the substrate model mesh influences the results. Since the simulation is based on a finite element model, we do expect that the finer the mesh, the more accurate the results are. In addition, the resolution of the mesh representing the teeth must be correlated to the resolution of the substrate mesh, otherwise the subtle influence of the fine details in the teeth surface would not be captured. For this purpose we have designed a simple test, which compares the stress results recorded for 3 substrate meshes of different resolutions, which can be seen in Table 5.7.

<table>
<thead>
<tr>
<th>Substrate Mesh ID</th>
<th>Substrate Mesh Name</th>
<th>Number of tetrahedra</th>
<th>Number of vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cube40</td>
<td>413526</td>
<td>74088</td>
</tr>
<tr>
<td>2</td>
<td>Cube30</td>
<td>178746</td>
<td>32768</td>
</tr>
<tr>
<td>3</td>
<td>Cube20</td>
<td>55566</td>
<td>10648</td>
</tr>
</tbody>
</table>

Table 5.7: Resolutions of the cubic substrate meshes used in testing

The results obtained for the meshes with a smaller resolution, namely Meshes 2 and 3 are shown in Charts 5.42 and 5.43. It is difficult to decide whether the comparison between the original and the smoothed restorations is successful for these meshes. As one can notice from the charts, while a difference is noticeable, it does not seem to be consistent with the expectations. What is most obvious, is the fact that the lowest-resolution mesh (Mesh 3) produces very similar results for both the original and the smoothed restorations. This is consistent with the expectation that if the mesh is too coarse, it cannot capture the subtle surface features of the teeth. The differences noticed for the Mesh 2 are not conclusive. While it does show as expected that more contact points are recorded for the smoother restorations, whether the stress levels are sufficiently different to suggest a difference in behaviour is not completely clear. In addition, we notice in Chart 5.43 that the original restorations pushed the substrate away from teeth, which explains the nearly 0 contact points shown in this chart. This can happen due the different shape that the original restorations have compared to the smooth restorations. The substrate is constantly attracted back towards the teeth by the tongue force, which is why we see more contact points being detected as the simulation progresses. However, Chart 5.44 shows the same comparison for the finest mesh (Mesh 1) and here we notice that higher stress levels are induced by the original restorations. Figure 5.41 shows the shape of the substrates at the end of the simulation for the 3 meshes of different resolutions. In conclusion, this test has shown that the resolution of the substrate mesh has an impact in the results of the simulation and should be chosen carefully. For results that can be relied upon, a finer mesh should be used. The only reason for not using the finest mesh possible, is the time constraints, because the finer the mesh is, the more time the
simulation will take.

**Figure 5.41:** Substrate shape at the end of the simulation for different resolution levels
Figure 5.42: Comparison between recorded stress values for smaller resolution meshes (Meshes 2 and 3, cf. Table 5.7)
Figure 5.43: Comparison between recorded number of contact points for smaller resolution meshes (Meshes 2 and 3, cf. Table 5.7)
5.4. COMPARING RESTORATIONS

Figure 5.44: Comparison for highest-resolution substrate mesh (Mesh 1, cf. Table 5.7)
Chapter 6

Future work

In order to improve the accuracy of the chewing simulation, finer meshes should be employed, for both the substrate models and the restoration meshes. This however requires further efforts to speed up the simulation, by employing more efficient algorithms, perhaps using parallelization and GPU processing. There is potential to speed up the computations of the deformations by internally using a solver which supports multithreading, such as the PARDISO solver [29]. At the moment the application uses the default, Vega provided Jacobi preconditioned conjugate gradient solver [67]. The collision detection algorithm could be sped up possibly by employing a spatial partitioning method, however considering that we also need to calculate the forces that result from these collisions and decide on which vertices to apply them, this problem can expand into a research topic in itself. In order to further improve the realism of the deformations, additional efforts are needed to detect and prevent self intersections in the substrate mesh.

In addition, it would be interesting to model the effect that the tongue has on the chewing, since in reality it does have a significant impact on the outcome of the mastication process. We have tried to simulate its presence by using a so called “tongue force”, but we feel there could be improvements in this area. While chewing, people often use their tongue to reposition the food, to move it from one side to the other and even to remove pieces which might have become stuck to the teeth. The tongue is an integral part of the chewing process and we assume it would be very difficult to actually chew anything successfully without help from the tongue. This could be modelled as an additional scene object which would collide with the substrate pieces and therefore move them around. Even though this addition would bring the simulation one step closer to realism, it is important to mention that the tongue in conjunction with the other muscles in the mouth also create a suction process which is very important in moving small pieces of food. To our knowledge, there is no research into recording tongue activity during a chewing process and therefore obtaining this kind of data could pose an insurmountable
problem at this time.

Last but not least, the influence of saliva in melting or binding the chewed pieces together is not modelled by our simulation and it would be very interesting to introduce it. We assume that for chewing a certain kind of food that does not immediately interact with saliva, this addition would not make a noticeable difference. However, for chewing a gummy bear, which incidentally was our main purpose, it might actually make a difference since this kind of material has a strong reaction to saliva, modifying its properties when it comes in contact with it.

In terms of user interface, more research is necessary to decide how this application could be used and under what circumstances. What elements would be of interest to the dentist and what would the patient like to see, are two important issues which require further investigation. This application could be integrated with CEREC and based on some computed parameters during the chewing cycle it could be used to decide in favour of one restoration over another. At this point it is not clear which parameters are the most important in deciding the fitness of a restoration and this is another issue which we leave for future work. We should however mention that the restorations should not be strongly optimized for chewing only gummy bears, since this kind of substrate is actually not a staple food in most diets. One suggestion would be to decide on a collection of regular food items which have different properties and run one chewing simulation for each of them. The restorations which perform reasonably well for all of these different foods should be chosen in the end.
Chapter 7

Conclusion

The application presented here is, to our knowledge, a first attempt to simulate the mastication process in a virtual environment, using deformable models and relying on chewing data acquired with a virtual articulator device, for the purpose of investigating the efficiency of different restorations created with CAD/CAM software.

Representing the deformable substrate with a model based on invertible FEM has enabled us to obtain a stable simulation which can handle severe deformations in a realistic manner. Continuous collision detection during the chewing process provides a realistic motion of the substrate pieces as they interact with the teeth and with each other. In addition to the forces resulting from collisions, the tongue force constantly pulls gently at the substrate pieces, ensuring they do not fly away from the scene and ensuring they remain in close proximity to the teeth. Fracturing of the substrate has been added in an effort to increase its physical accuracy and to allow the chewing process to achieve its purpose, that of breaking up the food into multiple smaller pieces. Analysis of the fracture event and its results add an extra layer of information to the simulation, providing more options for obtaining useful insight into the fitness of the restorations. By carrying out a number of different tests, we have tried to identify the factors that have the most significance in the outcome of the simulation, such as the shape of the restorations and the properties of the substrate model. We can conclude that we notice a difference in the results produces with restorations of different shapes and we expect that with finer meshes for the substrate model and the teeth, the impact of small topological changes in the restorations would be more visible.

In addition to comparing restorations, the user has the option to constantly interact with the scene, hide the substrate to highlight contact areas on the teeth or hide the teeth to bring the substrate into focus and observe it as it deforms. Stress colour maps can be enabled to highlight the inner working of the deformable model and observe first hand
which are the active areas. By observing these areas, the user can even predict where the substrate will fracture. All the geometry elements available in the scene such as arrows representing the different kind of forces could be used to directly observe how a certain tooth is interacting with the substrate and how often it does so.

The aim of this project was to research the feasibility of producing a chewing simulation which could be used to compare restorations. The current prototype application can be considered a first step on the path of creating a truly realistic and fast simulation which could be integrated with CEREC. While the individual components of this application can be improved, the main effort in this project was to put everything together, to research into the best methods for representing the various scene elements and to design the best fit collision detection algorithm and force calculation process. Last but not least, a significant effort was put into implementing the application in order to prove the feasibility of creating a chewing simulation. We have seen a lot of promise in using this application to compare restorations and have shown that producing such a simulation is possible and feasible, although more work is needed in order to meet the desired time constraints and degree of realism.
Appendix

This section contains a complete chewing progression, as recorded by the automatic simulation using the Cube40 model and the original restorations. This simulation used the first starting point as it is shown in Figure 5.32. The progression is shown from 3 different viewpoints: centred on the substrate, complete scene from the left side and the same complete scene rotate to 180 degrees.
Figure 1: Centered viewpoint, part 1
Figure 2: Centered viewpoint, part 2
Figure 3: Default scene viewpoint, part 1
Figure 4: Default scene viewpoint, part2
Figure 5: Rotated scene, part 1
Figure 6: Rotated scene, part2
Bibliography


129


