Softcell constraints for robust real-time deformation

Teo, Edmund Yew Kuan

2009

Teo, E. Y. K. (2009). Softcell constraints for robust real-time deformation. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/19018>

<https://doi.org/10.32657/10356/19018>

NANYANG TECHNOLOGICAL UNIVERSITY

Softcell Constraints for Robust Real-time Deformation

A Masters Thesis
Submitted to the School of Computer Engineering
of the Nanyang Technological University

by

Teo Yew Kuan Edmund

for the Partial Fulfillment of the Requirements
for the Degree of Masters of Engineering

April 21, 2009

Acknowledgments

This thesis project is part of a Masters programme sponsored by TQ Global Pte Ltd. I thank Parimal Aswani for his meticulous guidance and expressing his confidence in my work. I thank my supervisor, Dr Seah Hock Soon, who gave me complete flexibility in pursuing my research interest.

Last of all, I would like to thank all my friends who have given me their valuable pointers on improving this report.

Contents

Acknowledgments	i
List of Figures	iv
Abstract	v
1 Introduction	1
1.1 Problem Statement	1
1.2 Objectives	2
1.3 Thesis Organization	3
2 Background And Related Work	4
2.1 Continuum Mechanics	4
2.1.1 Stress Tensor	5
2.1.2 Strain Tensor	7
2.1.3 Generalized Hooke's Law	9
2.2 Deformation Model Classification	10
2.3 Finite Element Methods	11
2.3.1 Overview	11
2.3.2 Linear Tetrahedron Element	12
2.3.3 Analytical Formulation	14
2.3.4 Numerical Computation	17
2.3.5 Advantages and Limitations	18
2.4 Mass Spring Systems	19
2.4.1 Overview	19
2.4.2 Advantages and Limitations	21
2.5 Meshless Deformation Based on Shape Matching	22

2.5.1	Overview	22
2.5.2	Formal Definition	23
2.5.3	Optimal Translation	24
2.5.4	Optimal Rotation	25
2.5.5	Advantages and Limitations	26
2.6	Coupled Prisms Method	27
2.6.1	Overview	27
2.6.2	Advantages and Limitations	29
3	Softcell Constraints	30
3.1	Design Considerations	30
3.2	Simulation Overview	32
3.3	Cell Discretization	33
3.4	Constraint Overview	36
3.5	Constraint Notations	36
3.6	Point to point Constraint	37
3.6.1	Motion of a fixed point in a rigid body	37
3.6.2	Constraint force equation	40
3.7	Orientation Constraint	43
3.7.1	Finding average orientation	44
3.7.2	Integration for orientation	46
4	Extension	49
4.1	Fracturing	49
4.2	Clustering	49
5	Results	51
6	Conclusion and Future Work	58
A	Derivation of orientation constraint	61
	References	63

List of Figures

2.1	A material element in a continuum	5
2.2	Components of stress tensor	6
2.3	A tetrahedralized cylinder	11
2.4	Linear tetrahedron element	12
2.5	Simulation stages	22
2.6	Prisms extrusion and deformations. Taken from [8]	27
3.1	Cells voxelization. Taken from [33]	31
3.2	Cut section of an adaptive discretization	34
3.3	Application of constraints	36
3.4	Graphical solution of C	41
4.1	Cubic regions with 4 clusters	50
4.2	Cubic regions with 6 clusters	50
5.1	Fracture	52
5.2	Unclustered deformation	53
5.3	Clustered deformation	54
5.4	Object under robust deformation	55
5.5	Complex simulation environment	57
6.1	Tunneling artifacts	59

Abstract

This thesis presents softcell constraints, a novel set of formulations for robust real-time deformation. The underlying model handles volumetric objects in the form of surface mesh. The main theory behind the approach is to voxelize the surface mesh into cuboid cells. Neighbouring cells are elastically coupled together through constraint forces and torques. Based on geometric formulations, the constraint forces/torques are designed to minimize the global elastic energy.

The rigidity of the cells is inspired by the coupled prisms method and has made the method highly robust even when under extreme deformation. The simulation cost is linear to the number of cells. Thus, it is capable of handling a reasonable number of complex objects. Collisions are easy to handle as it is based on cuboid primitives. Thus, it eliminates the burden to implement complex collision detection technique and contact handling.

This combination of robustness, performance and ease of implementation makes the method particularly interesting for games.

Chapter 1

Introduction

1.1 Problem Statement

Classical physics engines only support rigid body simulation. Over the years, highly efficient methods for collision detection and collision response for rigid bodies have been developed.

In comparison, deformable bodies are rarely seen because they are a much more complicated problem. Since their shapes can change frequently, there is a need to spatially arrange the geometries for efficient collision detection. Often, it requires considerably more memory storage and computational effort. Furthermore, self-intersection must be handled to prevent the deformable bodies from penetrating themselves. Deformable bodies can undergo elastic or plastic deformation. In both cases, the internal elastic forces must be computed.

Fortunately, new powerful CPUs, GPUs and sufficient storage have given us better capabilities to solve many deformation problems. One representative example is the NVIDIA PhysX[35] technology. It is a powerful physics engine which utilizes the power of multi-core CPUs, GPUs and dedicated physics processor units (PPUs) to enable real-time physics. Most notably, it allows the simulation of volumetric deformable

bodies.

We can also see that deformation features are highly sought after in games. Gran Turismo 5[46] and Fracture[25] are two recent games which are highly acclaimed for their unique deformation features. Thus, it is worth developing a deeper understanding on how to utilize the power of current technologies for deformation.

1.2 Objectives

The first aim of this work is to survey different deformation techniques. Within the survey, suitable methods for games are to be identified. After that, we create a technique that can apply deformation in real-time to collidable objects in games.

The requirements of the technique include:

- **Support complex objects:** The technique should handle a large variety of complex objects in 2D and 3D.
- **Simulation Interactivity:** The technique should simulate complex simulation scenarios that are considered typical for games at interactive frame rates.
- **Stability:** The technique should be stable for a reasonable wide range of time steps while maintaining interactivity.
- **Simple Implementation:** The technique should be easy to implement and integrate into a physics simulation framework.

1.3 Thesis Organization

The remainder of the thesis is organized as follows.

Chapter 2 presents a basic introduction to continuum mechanics - an essential topic for understanding the current areas in deformation. Four important areas of development are discussed - finite element method, mass spring system, shape matching and coupled prisms method. Challenges for real-time deformation are also identified.

Chapter 3 describes the theories behind softcell constraints, a new real time deformation method for volumetric bodies. Speed and stability are critical in physics simulation for games. These serve as the main design consideration. The strength and limitations of the approach are also analyzed.

Chapter 4 proposes additional extensions to the softcell constraints.

Chapter 5 presents the results of the new technique.

Chapter 6 concludes the thesis and discusses the future work.

Chapter 2

Background And Related Work

For researchers unfamiliar to the field, two surveys[16, 34] on the study of deformation within the computer graphics research community serve as a good entry point.

Simulating deformation in computer graphics is an inherently interdisciplinary field - namely a combination of concepts from mechanical engineering and computer science. It is a problem which requires intricate knowledge of many areas such as Newtonian dynamics, continuum mechanics, numerical computation, differential geometry, vector calculus, approximation theory, computer graphics, etc. Each of them deserves a lengthy discussion for a proper introduction.

While it is not possible to cover every topic, this chapter strives to give a treatment on elementary concepts that will complement the two survey reports for those without mechanical engineering background.

2.1 Continuum Mechanics

Continuum mechanics is the study of kinematics and mechanical behavior of materials modeled as a continuous body, e.g. solids and fluids. Such bodies are commonly called

CHAPTER 2. BACKGROUND AND RELATED WORK

continuum. Interested readers may refer to [14] for further reference.

Figure 2.1: A material element in a continuum

A continuum is considered as a set of infinitesimal volumetric elements which are called material elements in this thesis. The material elements are connected to each other with no gaps or breaks between them and thus satisfy the continuity assumption. Figure 2.1 shows that an element is treated as a rectangular parallelepiped in the continuum.

2.1.1 Stress Tensor

There are two types of external forces acting on the material elements:

- Body forces which act on every material elements. An example is gravitational force.
- Surface tractions which act on the material elements on the body surface. An example is the collision force resulting from a collision between two bodies.

The external forces result in stresses acting on the elements. Stress is defined as the measure of amount of force acting on a body per unit area. In general, it is expressed as:

$$\sigma = \left(\frac{F}{A} \right) \quad (\text{Eq. 2.1})$$

CHAPTER 2. BACKGROUND AND RELATED WORK

where:

- σ is the stress.
- F is the acting force.
- A is the area that F is acting on.

Figure 2.2: Components of stress tensor

It is common to represent the stresses acting on a material element as a stress tensor.

Figure 2.2 shows an element is defined in a reference frame with coordinate axes x , y and z parallel to its edges. The stress tensor components are labeled on the element.

The stress tensor is given as

$$\begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \quad (\text{Eq. 2.2})$$

Let ΔS be the surface of the element with an outer normal vector pointing in the positive direction of the x axis. The equivalent of σ_x is τ_{xx} which is in turn the stress

CHAPTER 2. BACKGROUND AND RELATED WORK

vector acting on ΔS in the positive x direction. Similarly, τ_{xy} is the stress vector acting on ΔS in the positive y direction and τ_{xz} is the stress vector acting on ΔS in the positive z direction.

The other components of the stress tensor are defined in the similar way. σ_x , σ_y and σ_z are called the normal stresses while the remaining components are shear stresses.

By considering the equilibrium of the equations of motion of the element and setting the limit of the element volume to zero (i.e. $dV \rightarrow 0$), it can be proven that the stress tensor is symmetric. That is $\tau_{ij} = \tau_{ji}$ for all $i, j \in \{x, y, z\}$.

2.1.2 Strain Tensor

Strain is a measure of deformation which represents the relative displacement between particles in a deformed body.

To further illustrate, let $m = (m_1, m_2, m_3)$ be the material coordinates of points in a body Ω . The world coordinate at m will be $X(m) = (x(m) \ y(m) \ z(m))^T$.

When forces are applied, the body deforms and a point at material coordinate m will be displaced. The displacement vector from its original position in world space will be $U(m) = (u(m) \ v(m) \ w(m))^T$.

$U(m)$ is known as the displacement field. The gradient of the displacement field is given by

$$\nabla U = \begin{bmatrix} u_{,x} & u_{,y} & u_{,z} \\ v_{,x} & v_{,y} & v_{,z} \\ w_{,x} & w_{,y} & w_{,z} \end{bmatrix} \quad (\text{Eq. 2.3})$$

CHAPTER 2. BACKGROUND AND RELATED WORK

where $u_{,x}$ is defined as the partial derivative of u with respect to x . The other components are defined in a similar way.

A strain tensor represents the strain state of a body and the general form is given as

$$[\epsilon] = \begin{bmatrix} \epsilon_x & \frac{\gamma_{xy}}{2} & \frac{\gamma_{xz}}{2} \\ \frac{\gamma_{yx}}{2} & \epsilon_y & \frac{\gamma_{yz}}{2} \\ \frac{\gamma_{zx}}{2} & \frac{\gamma_{zy}}{2} & \epsilon_z \end{bmatrix} \quad (\text{Eq. 2.4})$$

(Eq. 2.4) can be expressed as a function of the displacement field ∇U . There are several choices of representation. For example, the Cauchy's linear strain tensor and the Green's nonlinear strain tensor are

$$\epsilon_c = \frac{1}{2} (\nabla U + [\nabla U]^T) \quad (\text{Eq. 2.5})$$

$$\epsilon_G = \frac{1}{2} (\nabla U + [\nabla U]^T + [\nabla U]^T \nabla U) \quad (\text{Eq. 2.6})$$

respectively.

The choice of strain tensor has significant implications. For instance in some deformation models, the nonlinear strain tensor offers better accuracy at the cost of solving higher order partial differential equations (PDEs).

2.1.3 Generalized Hooke's Law

The generalized Hooke's Law allows us to relate the stress tensor to the strain tensor through the following equation:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{xz} \end{Bmatrix} = [C] \begin{Bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{xz} \end{Bmatrix} \quad \text{Or } \sigma = C \cdot \epsilon \quad (\text{Eq. 2.7})$$

where $[C]$ is the 6x6 constitutive matrix which determines the material behavior of the body. The law states that the stress tensor is linearly proportional to the strain tensor.

2.2 Deformation Model Classification

Early researchers have provided a meaningful classification[34] for the diversity of deformation models. One major distinction is between Lagrangian methods and Eulerian methods.

In Lagrangian methods, deformable bodies are modeled as a set of points with different positions and properties. In Eulerian methods, a set of fixed points in a fixed reference frame are associated with material properties. Eulerian methods are suited for simulating fluids and gases where the flow of fluid/gas particles in a region is of interest.

This report will focus on the following area of developments:

- Lagrangian Mesh Based Methods
 - Finite Element Method (FEM)
 - Mass Spring Systems
- Lagrangian Mesh Free Methods
 - Meshless Deformation Based on Shape Matching
 - Coupled Prisms Method

2.3 Finite Element Methods

2.3.1 Overview

In FEM, the continuum is discretized into interconnected geometric elements.

Figure 2.3: A tetrahedralized cylinder

Figure 2.3 is an example of a tetrahedralized cylinder and the cut section of the cylinder is on the right. There are many other forms of elements that can be used. The elements are generally characterized by how many nodes each element contains. The higher the number of nodes in the element the higher the accuracy in the simulation is. However, the tradeoff is the computational cost.

4-node tetrahedrons are the most commonly used geometric elements in computer animations because they don't incur too much overhead with sufficient accuracy. The next section will discuss more on the properties of tetrahedron elements.

2.3.2 Linear Tetrahedron Element

Figure 2.4: Linear tetrahedron element

Figure 2.4 shows a tetrahedron element with 4 nodes. Any point p within the tetrahedron can be expressed in world space as

$$p = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \sum_{i=1}^4 N_i p_i = \sum_{i=1}^4 N_i \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \quad (\text{Eq. 2.8})$$

where:

- $N_1 = 1 - \xi - \eta - \zeta$
- $N_2 = \xi$
- $N_3 = \eta$
- $N_4 = \zeta$
- $\xi, \eta, \zeta \in [0, 1]$

The functions denoted by N_i for $i = 1$ to 4 are called isometric shape functions.

Elements defined by such shape functions are called isoparametric elements.

CHAPTER 2. BACKGROUND AND RELATED WORK

It can be seen that these functions interpolate the position vectors of the 4 nodes to represent a point in the element. Any attribute associated with the nodes (i.e. stress or strain) can be interpolated too.

Similarly, the displacement $\{U\} = \begin{Bmatrix} u \\ v \\ w \end{Bmatrix}$ of an internal point from its original position will be

$$\{U\} = [N]\{u^e\}$$

$$\Downarrow$$

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 & 0 & 0 \\ 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 & 0 \\ 0 & 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ w_1 \\ u_2 \\ v_2 \\ w_2 \\ u_3 \\ v_3 \\ w_3 \\ u_4 \\ v_4 \\ w_4 \end{Bmatrix} \quad (\text{Eq. 2.9})$$

The procedure for FEM simulation has two stages:

- Analytical Formulation
- Numerical Computation

CHAPTER 2. BACKGROUND AND RELATED WORK

2.3.3 Analytical Formulation

The objective in analytical formulation is to find the element stiffness matrix that governs the nodal forces acting on each element under deformation.

In the most basic form of FEM, the generalized Hooke's Law from Eq. 2.7 is used.

Materials whose mechanical properties do not depend on directions are considered isotropic. Their behaviors are governed by Eq. 2.7 in the following form:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{xz} \end{Bmatrix} = \frac{E}{(1+v)(1-2v)} \begin{bmatrix} 1-v & v & v & 0 & 0 & 0 \\ v & 1-v & v & 0 & 0 & 0 \\ v & v & 1-v & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}-v & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}-v & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}-v \end{bmatrix} \begin{Bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{xz} \end{Bmatrix} \quad (\text{Eq. 2.10})$$

where:

- E is the Young's modulus constant which specify the stiffness of the material
- v is the Poisson's ratio in the range $[0, 0.5]$

Based on the Cauchy's linear strain tensor, which was described in section 2.1.2, the strain tensor components can be expressed as:

$$\begin{Bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{xz} \end{Bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \Leftrightarrow \{\epsilon\} = [L]\{U\} \quad (\text{Eq. 2.11})$$

CHAPTER 2. BACKGROUND AND RELATED WORK

Combining Eq. 2.9 and Eq. 2.11,

$$\begin{aligned}\{\epsilon\} &= [L]\{U\} \\ \{\epsilon\} &= [L][N]\{u^e\} \\ \{\epsilon\} &= [B]\{u^e\}\end{aligned}\tag{Eq. 2.12}$$

where:

$$[B] = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial y} & \frac{\partial N_4}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial z} & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial z} & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial z} & \frac{\partial N_3}{\partial y} & 0 & \frac{\partial N_4}{\partial z} & \frac{\partial N_4}{\partial y} \\ \frac{\partial N_1}{\partial z} & 0 & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial z} & 0 & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial z} & 0 & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial z} & 0 & \frac{\partial N_4}{\partial x} \end{bmatrix}\tag{Eq. 2.13}$$

$[B]$ is the matrix form of the shape function derivatives. To find each derivative component in $[B]$, the jacobian matrix $[J]$ needs to be derived from Eq. 2.8.

$$[J] = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} = \begin{bmatrix} (x_2 - x_1) & (y_2 - y_1) & (z_2 - z_1) \\ (x_3 - x_1) & (y_3 - y_1) & (z_3 - z_1) \\ (x_4 - x_1) & (y_4 - y_1) & (z_4 - z_1) \end{bmatrix}\tag{Eq. 2.14}$$

Finally,

$$\begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{Bmatrix} = [J] \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix}\tag{Eq. 2.15}$$

CHAPTER 2. BACKGROUND AND RELATED WORK

$$\begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{Bmatrix} \quad (\text{Eq. 2.16})$$

We can get the final form of $[B]$ by substituting Eq. 2.16 into Eq. 2.13. $[B]$ is eventually a function of the node positions.

The principle of minimum potential energy is stated as

Theorem 2.1 *Of all the admissible configurations of a conservative system, those satisfying the equilibrium equations make the total potential energy minimum with respect to small variations of displacements about the stable equilibrium configuration.*

Based on the aforementioned principle and variational calculus, it can be proven that the 12×12 stiffness matrix of a tetrahedron element is:

$$[K^e] = V[B]^T[C][B] \quad (\text{Eq. 2.17})$$

where:

- V is the volume of the tetrahedron
- $[C] \in \mathbb{R}^{6 \times 6}$ is the constitutive matrix

2.3.4 Numerical Computation

From the discretization of a continuum, the element nodes can be shared by several elements. The elastic forces from adjacent elements must be added up at each node. Thus, the first operation in the numerical computation is to assemble the individual element stiffness matrix into a $3n \times 3n$ stiffness matrix $[\mathbf{K}]$ for the continuum.

The linear system equation of motion for the whole continuum becomes:

$$[M]\{\ddot{U}\} + [D]\{\dot{U}\} + [\mathbf{K}]\{U\} = \{f_{ext}\} \quad (\text{Eq. 2.18})$$

where:

- $[M] \in \mathbb{R}^{3n \times 3n}$ is the diagonal mass matrix. The diagonal entries are the masses of the individual nodes
- $[D] \in \mathbb{R}^{3n \times 3n}$ is the damping matrix.
- $\{f_{ext}\} \in \mathbb{R}^{3n}$ is the external force vector acting on the nodes.
- $\{U\} \in \mathbb{R}^{3n}$ is displacement vector of all the nodes.

For every simulation step, the only unknown in Eq. 2.18 is $\{\ddot{U}\}$. Readers should be able to see that the problem is reduced to solving a matrix equation of the form $[A]\{x\} = \{b\}$. Matrix solvers like LU decomposition and Gaussian elimination are well discussed in [38].

2.3.5 Advantages and Limitations

FEMs are often the first choice when a realistic result is desired. It offers high accuracy in approximating the physical model. However, a lot of preprocessing work is involved. The continuum mesh model must be discretized into elements. Repeated mesh refinement may also be required to reduce modeling errors[3].

It is widely known that linear FEM (see Eq. 2.5) is only limited to simulating deformations less than 1% of the object dimensions. Otherwise, the objects will increase in volume unnaturally[28].

While non-linear FEM (see Eq. 2.6) can account for large rotational deformations, it results in a non-linear system of equations that will slow the simulation down greatly. Furthermore, non-linear elasticity can make the system numerically unstable for large deformations. Special treatment would be required[15].

Linear FEM approach can be extended with stiffness warping[29] to prevent the artifacts from large rotational deformations while keeping the simulation almost as efficient as the original approach. To further optimise the computation for interactive applications, it is common to link every vertex of the high resolution mesh to the closest tetrahedron in the low resolution mesh and its barycentric coordinates is stored. The low resolution mesh will be used for the FEM simulation and also act as a control mesh to deform the high resolution mesh.

FEM remains a complicated and expensive method. Furthermore, graphic requirements for modern games are ever growing with higher surface mesh details. There are still a lot of challenges to adopt FEM in this area.

2.4 Mass Spring Systems

2.4.1 Overview

The procedures in mass spring system can be generalized as follows:

- (i) Discretize the objects
- (ii) Define the connectivity
- (iii) Define the deformation parameters
- (iv) Compute the spring forces
- (v) Force and velocity integration

A continuum is discretized into mass points. The adjacencies of the mass points will be defined to form simplices. It can be line segments in 1D, triangles in 2D, and tetrahedralized meshes in 3D (Section 2.3.2).

Each point has mass, position and velocity while each spring is typically specified by its stiffness and original length. On every simulation step, the spring forces are computed and summated at the mass points. Let x_1 and x_2 be the positions of two points connected by a spring with an original length L . Based on the linear elastic model, the forces acting at x_1 and x_2 will be:

$$\begin{aligned} F_d(x_1) &= k_d (|x_2 - x_1| - L) \frac{x_2 - x_1}{|x_2 - x_1|} \\ F_d(x_2) &= -F_d(x_1) \end{aligned} \tag{Eq. 2.19}$$

where k_d defines the stiffness of the spring. Note that the forces preserve the distance between the two points. The forces tend to pull the two points back to their original

CHAPTER 2. BACKGROUND AND RELATED WORK

distance but are zero if the spring length is exactly the same as the original length. In more advanced systems, constraint forces that conserve surface area and volume are defined. Based on the continuum's material behavior, constraint functions to preserve surface area or volume are applied to the system to restrict its movement in response to any strain. These functions can be defined at element level[45] or object level[21].

Suppose this constraint function is $C(x) \in \mathbb{R}^{3n}$, the continuum's potential energy arising from this constraint is $E_c(x) = \frac{k}{2}C(x)^T C(x)$ where the mass point positions are collectively $x \in \mathbb{R}^{3n}$ for n points and k is a stiffness constant. By finding the total potential energy $E(x)$ caused by the constraint functions, the net force f_i for point i arising from the energy is $f_i = -\frac{\partial E}{\partial x_i}$.

The remaining step is to choose an integration scheme. Stability is an important consideration especially when the stiffness constants are large values. Starting with the known positions $x(t_0)$ and velocities $\dot{x}(t_0)$ at time t_0 , the goal is to find $x(t_0 + h)$ and $\dot{x}(t_0 + h)$ where h is the time step.

$$\begin{pmatrix} x(t_0 + h) - x(t_0) \\ \dot{x}(t_0 + h) - \dot{x}(t_0) \end{pmatrix} = h \begin{pmatrix} \dot{x}(t_0) \\ f(x(t_0), \dot{x}(t_0)) \end{pmatrix} \quad (\text{Eq. 2.20})$$

$$\begin{pmatrix} x(t_0 + h) - x(t_0) \\ \dot{x}(t_0 + h) - \dot{x}(t_0) \end{pmatrix} = h \begin{pmatrix} \dot{x}(t_0 + h) \\ f(x(t_0 + h), \dot{x}(t_0 + h)) \end{pmatrix} \quad (\text{Eq. 2.21})$$

The explicit (Eq. 2.20) and implicit (Eq. 2.21) Euler schemes are widely discussed and compared. The latter scheme is often much preferred as it allows for large time steps while maintaining stability[2].

2.4.2 Advantages and Limitations

In contrast to FEM, mass spring systems are easier to implement and faster. They have been applied to a wide of range of deformable bodies such as cloth, muscles, etc. The nature of mass spring systems allows highly parallelizable computation.

The behaviour of mass spring systems is tuned through the spring constants. Achieving isotropic and anisotropic properties can be very challenging[9].

Large spring constants tend to introduce instability to the system and limit it to small time steps. This will adversely affect the interactivity of games[1].

2.5 Meshless Deformation Based on Shape Matching

2.5.1 Overview

Müller et al. are arguably the first to propose a deformation method that is specifically suited for games[30]. Figure 2.5 summarizes the stages in meshless deformation based on shape matching.

Figure 2.5: Simulation stages

CHAPTER 2. BACKGROUND AND RELATED WORK

Two other works have strong connections to aforementioned approach. In the fast lattice shape matching method (FLSM)[39], Rivers et al. has extended the shape matching method by constructing a lattice of cubic cells containing the mesh. The embedded mesh is then deformed using trilinear interpolation of the lattice vertex positions. The lattice cells are grouped into overlapping shape matching regions. For each region, FLSM performs the shape matching operations as per normal. The per-node goal positions are obtained by averaging the transformations of all influencing regions. FLSM exploits the summation redundancy of the regions and provides large-region matching at constant per-vertex cost.

Steinemann et al. relied on an octree-based hierarchical sampling and an interval-based region definition to perform shape matching[44]. Similar to FLSM, they summed the transformations of the regions for the goal positions. Their method supports high degrees of freedom owing to the adaptive octree discretizations. The underlying data structures can also be reused for efficient handling of collisions. As such, it is an attractive option for video games.

The shape matching technique will be discussed as it is one of the central concepts of the coupled prisms method to be discussed in Section 2.6.

2.5.2 Formal Definition

The problem is modelled as follow: Given the original shape x_i^0 and deformed shape x^i , find the optimal rotation matrix R and translation vector t which minimize

$$\sum_i m_i \|e_i\|^2 = \sum_i m_i (R(x_i^0 - t_0) + t - x_i)^2 \quad (\text{Eq. 2.22})$$

where m_i are the mass of each point. t_0 is chosen to be the center of mass of the original shape for convenience, i.e.

$$t_0 = x_{cm}^0 = \frac{\sum_i m_i x_i^0}{\sum_i m_i} \quad (\text{Eq. 2.23})$$

To compute the optimal rotation and translation, a popular technique is the iterative closest point algorithm[7, 40]. Given two point clouds, the algorithm attempts to associate the points by the nearest neighbour criteria. Then the estimated transformation parameters are obtained by using a mean square function in an iterative fashion.

As the correspondences between the points are a priori known in the above case, the problem can be reduced to finding the optimal rigid transformations between two point clouds with known correspondences. Related approaches can be found in [47, 12, 22].

The next two sections will develop the approach used by Müller et al. to find the optimal translation and rotation.

2.5.3 Optimal Translation

It can be proven that the optimal translation t is simply the center of mass x_{cm} of the deformed shape. i.e. $x_{cm} = \sum_i \frac{m_i x_i}{m_i}$.

Proof:

CHAPTER 2. BACKGROUND AND RELATED WORK

Let $p_i = x_i - x_{cm}$ and $q_i = x_i^0 - x_{cm}^0$. Note that

$$\sum_i m_i p_i = 0, \sum_i m_i q_i = 0 \quad (\text{Eq. 2.24})$$

e_i can now be rewritten as

$$e_i = R(q_i) - p_i + t - x_{cm} \quad (\text{Eq. 2.25})$$

(Eq. 2.22) will expand to

$$\sum_i m_i \|R(q_i) - p_i\|^2 + 2 \cdot (t - x_{cm}) \sum_i m_i (R(q_i) - p_i) + \sum_i m_i \|t - x_{cm}\|^2 \quad (\text{Eq. 2.26})$$

Since R is a linear operation, the middle term is zero. The first term does not depend on t and the third must be positive. Therefore, (Eq. 2.22) is minimized when the third term is zero or $t = x_{cm}$. ■

2.5.4 Optimal Rotation

With $t = x_{cm}$, the term to be minimized is relaxed by replacing the rotation matrix R with an optimal linear transformation matrix A :

$$\sum_i m_i (Aq_i - p_i)^2 \quad (\text{Eq. 2.27})$$

Müller et al. set the derivatives with respect to all coefficients of A to zero and let A becomes

$$A = \left(\sum_i m_i p_i q_i^T \right) \left(\sum_i m_i q_i q_i^T \right)^{-1} = A_{pq} A_{qq} \quad (\text{Eq. 2.28})$$

A_{qq} only contains scaling transformation as it is symmetric. Thus, the optimal rotation

is found by breaking A_{pq} into $A_{pq} = RS$ where R is the rotational part and S is the scaling/shearing part. The rotation matrix can be extracted from A_{pq} via polar decomposition[42].

2.5.5 Advantages and Limitations

Shape matching deformation is a very attractive approach for games. It has been proven to simulate complex scenes with hundreds of objects and tens of thousands of points.

As discussed in the preceding sections, large deformations and high stiffness constants often challenge the stability and speed of traditional methods. Since the deformed shape is relaxed towards a rigid shape, the simulation is unconditionally stable. Shape matching deformation sets itself apart by assuring stability in all conditions.

By working on only point clouds, shape matching deformations avoid the need of elaborated preprocessing that is seen in FEM. It also does not require connectivity information in contrast to mass spring system.

However, the model is not physically based. The parameters used in the mathematical model has no physical meaning. As such, the result of the simulation is not governed by the stress-strain relationship from continuum mechanics. Nonetheless, this inaccuracy is acceptable for games since it suffices to provide a plausible effect.

The computations for shape matching deformation are mostly parallelizable and can be handled by parallel hardware such as the GPU.

2.6 Coupled Prisms Method

2.6.1 Overview

Botsch et al designed the coupled prisms method as a shape modeling tool for freeform deformation[8]. The method extends the idea of thin shells for the underlying surface deformation model. A thin shell is a thin volumetric layer wrapped around a center surface[5]. Shell simulations have been discussed in recent works such as [18] and [48].

Figure 2.6: Prisms extrusion and deformations. Taken from [8]

The prisms based method builds the layer by extrusion along vertex normals. A prism is generated for each mesh face. See top of Figure 2.6. The prisms are kept rigid so that they do not suffer degenerations. Botsch et al. had demonstrated that this can ensure numerical robustness even when under extreme deformations.

CHAPTER 2. BACKGROUND AND RELATED WORK

The prisms are connected along their shared faces by elastic joints. The joints are stretched when the prisms undergo bending and stretching deformations. The red shaded regions in Figure 2.6 are the stretched joints. The authors defined the amount of stretching in the joints as the desired deformation energy.

For each adjacent prisms P_i and P_j , the region of their connecting sides are parameterized by the function $f^{i \rightarrow j}(u, v)$ and $f^{j \rightarrow i}(u, v)$ respectively where $(u, v) \in [0, 1]$. The energy between P_i and P_j becomes:

$$E_{ij} = \int_{[0,1]^2} \|f^{i \rightarrow j}(u, v) - f^{j \rightarrow i}(u, v)\|^2 dudv \quad (\text{Eq. 2.29})$$

Looking at the top of the Figure 2.6 again, the layer thickness h defines the local surface thickness. For thicker prisms, the same bending or stretching deformation induces a higher intensity of joint stretching. In other words, the larger the thickness the higher the stiffness is. The deformation energy E of the system is a weighted summation of the pairwise energies E_{ij} . The weight is a function of the squared length of their shared edge and the area of their shared face.

$$E = \sum w_{ij} \cdot E_{ij} \quad (\text{Eq. 2.30})$$

The user controls the surface deformation by constraining the position and/or orientation of selected prisms. Then, the prisms method seeks to find the optimal rotation and translation of the unconstrained prisms so that the global deformation

CHAPTER 2. BACKGROUND AND RELATED WORK

energy is minimized. That will give the numerical solution of the deformed shape in equilibrium.

Note from (Eq. 2.29) that the deformation energy for a pair of adjacent prisms is zero when the sample points $f^{i \rightarrow j}(u, v)$ completely matches $f^{j \rightarrow i}(u, v)$. Thus, the problem has been effectively reduced to a global shape matching problem. The shape matching method to match one set of sample points to another set was discussed earlier in Section 2.5.

2.6.2 Advantages and Limitations

The main strength of the coupled prisms method is the robustness. This is because the individual prisms are kept rigid. Even when all the prisms are collapsed into one point and given random orientations, this method is still able to fully recover the deformed mesh within finite iterations. This is unlikely possible with conventional methods such as FEM.

Another important property of the method is the prisms generation is not restricted to triangular meshes.

The limitation of the method is its computational performance due to the need to solve the global shape matching problem.

Chapter 3

Softcell Constraints

3.1 Design Considerations

This chapter defines a set of dynamic constraints for robust and real-time deformation. These dynamic constraints are applied to rigid cuboid cells during the simulation and are thus called "Softcell Constraints". The inspiration came from the coupled prisms method introduced in Section 2.6.

A potential extension of the prism based framework is the application to physically based dynamic simulation, since numerical robustness is also very important in that context. Several challenges were identified in this thesis.

The discretization of the surface mesh into prisms is based on the mesh faces. This coupling of the mesh geometry to the discretization is prohibitive to the simulation of large scale deformation and high detail meshes.

In a dynamic simulation, collision handling of softbodies are inevitable. Since the surface mesh is updated through the prisms' rigid motion, it is most intuitive to treat the prisms as rigid bodies for simulations. The joints between the prisms should provide the restoring elastic forces when under deformations. However, the stiffness of

CHAPTER 3. SOFTCELL CONSTRAINTS

the mesh object is controlled by the thickness of the layer extrusion (see Section 2.6.1). The disadvantage is that the error in approximation of the actual geometry becomes proportional to the layer thickness. For near rigid bodies, the layer will scale up much larger than the actual geometry.

Figure 3.1: Cells voxelization. Taken from [33]

In this thesis, an alternative discretization is proposed. The author has chosen to voxelize the underlying mesh into cuboid cells in similar fashion described in [33]. The details will be explained in the Section 3.3. An example of their discretization is shown in Figure 3.1. Similar to the coupled prisms method, each individual cell is kept rigid to preserve the robustness property. The author also defines the softcell constraints to couple the adjacent cells.

The resulting cells can provide a reasonable approximation of the embedded geometry for collisions. Furthermore, a coarser discretization on a high detail mesh can be adopted for higher interactive rates. This effectively addresses the approximation error and geometry coupling issues that were mentioned earlier.

Although the elastic joints on the prisms are governed by non-linear deformation energies, the numerical solution for the equilibrium is geometrically based. Recall that the shape matching technique was used. It does not consider the mass, inertia and collisions. Thus, it cannot be used to couple the cells for dynamic simulation. The softcell constraints are designed to replace the prism joints and are based on Newtonian mechanics. They will fully describe the motion of the mesh object due to the effect of inertia, forces, torques and deformation.

3.2 Simulation Overview

The proposed top level simulation stages are straightforward. At the preprocessing stage, the mesh is discretized into cells and softcell constraints are defined to couple every adjacent pair. The main simulation loop comprises 4 steps:

- (i) **Predict unconstrained motion:** Forward step the simulation in time by integrating the equations of motion for each rigid body including the rigid cells.
- (ii) **Collision Detection:** Determine the points of contact for every pair of rigid bodies in collision. Collision detection for the softbodies is through box intersection tests on the rigid cells. Support for box primitive in physics engines is ubiquitous. Thus, it is fairly easy to implement and integrate this approach into most physics libraries.
- (iii) **Collision Response:** Resolve the points of contact to prevent intersection between the bodies.
- (iv) **Maintaining softcell constraint:** The cells may be displaced during collisions and result in deformation. Restoring forces and torques are computed based on the constraint formulation to simulate the internal elastic forces.

3.3 Cell Discretization

The cell discretization works in both 2D and 3D (square cells in 2D, cuboid cells in 3D). In this thesis, the proposed discretization is characterized by the maximum side lengths of the cuboid cell and the maximum number of times the initial cell can be split. Note that a cuboid can have unequal side lengths.

The mesh is first voxelized into cuboid cells with the maximum side lengths. The result is a set of equally sized cuboid elements, but the adjacent cells do not share their common vertices like in [33]. This is in order to keep the cells as individual elements. It must be stressed that in a pair of adjacent cells, one cell will have a face that coincides with a corresponding face of the other cell. The author calls it the common face but it should be regarded as two separate faces since the cells are individual elements.

This uniform discretization is straightforward but can be more efficient if an adaptive discretization is used. Zhuang et al pointed out that the cutoff spatial frequency of an object in response to external loads decreases in proportion to the distance away from the mesh surface[51]. Their approach was FEM based and they suggested that there is little accuracy lost if they increase the size of the interior elements.

In the same intuition, this thesis proposes to further split the surface cells in an octree manner (quadtree in 2D) until they reach the predefined maximum number of times they can be split. Surface cells are cells that intersect the surface of the embedded mesh. This will result in a coarser discretization in the mesh interior and finer control on the surface.

CHAPTER 3. SOFTCELL CONSTRAINTS

Figure 3.2: Cut section of an adaptive discretization

CHAPTER 3. SOFTCELL CONSTRAINTS

Figure 3.2 shows the cut-section of an adaptive discretization. The cyan cells are the surface cells while the blue cells are the interior cells.

The author found that the choice of the cuboid cell as a basic unit is a good compromise between efficiency, spatial adaptivity and ease of implementation. As the design goal is to allow arbitrarily shaped volumetric meshes, there are two commonly used primitive units - cuboids and tetrahedrons. For cuboids, the octree refinement allows easy updates of neighbouring cell information since it can be represented in an array. Thus, it also allows easy splitting of large cuboids into smaller ones. A cuboid cell will break exactly into 8 similar cuboid cells. The ratio of the side lengths of a child cell to that of its parent is always 1:2.

In contrast, discretizing a mesh into tetrahedrons is much more involved. Due to the lack of uniform discretization, it is more difficult to update the neighbourhood information during tetrahedron refinements. Each tetrahedron is fully described by its position vector, orientation and the 4 vertex position vectors. The cuboid is more memory efficient as it only requires the position vector, orientation and the 3 side length scalars.

Owing to its simplicity, sphere primitives are also used in bounding volume hierarchies for geometric deformations[43]. Since a sphere cannot be split into smaller spheres, it cannot take advantage of adaptive discretization. Furthermore, bounding sphere hierarchies are usually too loosely fit. One simple example is the bounding of a cube mesh with a sphere. Hubbard has introduced random changes to the spheres for a tighter fit in [23]. The process can be quite slow and can produce irregular distributions of spheres. It was commented that the practical value of this technique is limited[13].

3.4 Constraint Overview

The motivation behind the softcell constraints is to give rise to constraint forces/torques. On each simulation loop, the constraint forces/torques are integrated to reduce (Eq. 2.29) towards zero. The constraint forces/torques are analogous of the internal elastic forces to restore the body to its original shape. Within finite iterations, the system deformation energy will reach the desired global minimum.

Figure 3.3: Application of constraints

From the geometric point of view, the deformation energy between two adjacent cells is zero when their common faces completely meet. To achieve that, it suffices to constrain a point on the common faces of the two cells and also constrain their orientation to align. Refer to Figure 3.3.

Constrained dynamics is not a new topic. It has been found in related works such as [37, 24, 4, 50, 49]. However, the author has not seen it being used to derive elastic forces for softbodies. The following sections will develop the formulation of the point to point constraint and orientation constraint for explicit use in computation of elastic forces.

3.5 Constraint Notations

This thesis uses the following symbols for the state of a rigid body:

CHAPTER 3. SOFTCELL CONSTRAINTS

- X - position vector of center of mass
- R - orientation matrix of body
- v - linear velocity
- L - angular momentum
- w - angular velocity
- m - mass
- I_{body} - local inertia tensor
- I - world inertia tensor

3.6 Point to point Constraint

For each adjacent cuboid cell, its common face region is always a square. The center of the square region is chosen as the constrained location which will correspond to two points - one on the common face of each cell. The constraint force attempts to keep the two points to stay in contact.

3.6.1 Motion of a fixed point in a rigid body

The motion of a rigid body is governed by the Newton-Euler equations which are explained in [17, 26, 11]. The constraint formulation relies on the equation of motions of a fixed point in a rigid body. Therefore, several derivatives of the state properties are required.

The derivative of the I^{-1} is as follows

CHAPTER 3. SOFTCELL CONSTRAINTS

$$I^{-1} = RI_{body}^{-1}R^T \quad (\text{Eq. 3.1})$$

$$\begin{aligned} \frac{d}{dt}I^{-1} &= \left(\frac{d}{dt}R\right) I_{body}^{-1}R^T + RI_{body}^{-1} \left(\frac{d}{dt}R^T\right) \\ &= w^*RI_{body}^{-1}R^T + RI_{body}^{-1}R^Tw^{*T} \\ &= w^*I^{-1} + I^{-1}w^{*T} \end{aligned} \quad (\text{Eq. 3.2})$$

where w^* is the dual of vector w .

The derivative of w is as follows.

$$w = I^{-1}L \quad (\text{Eq. 3.3})$$

$$\begin{aligned} \frac{d}{dt}w &= \left(\frac{d}{dt}I^{-1}\right) L + I^{-1} \left(\frac{d}{dt}L\right) \\ &= \left(\frac{d}{dt}I^{-1}\right) L + I^{-1}T \\ &= w^*I^{-1}L + I^{-1}w^{*T}L + I^{-1}T \\ &= w^*w + I^{-1}(w^{*T}L + T) \\ &= I^{-1}(L \times w + T) \end{aligned} \quad (\text{Eq. 3.4})$$

where $T = \frac{d}{dt}L$ is the torque.

Let b_{body} be the fixed point of a body in its local coordinates. The first and second derivatives of $b = Rb_{body}$ are as follow.

CHAPTER 3. SOFTCELL CONSTRAINTS

$$\begin{aligned}
 \frac{d}{dt}b &= \frac{d}{dt}(Rb_{body}) \\
 &= \left(\frac{d}{dt}R\right)b_{body} \\
 &= w \times b
 \end{aligned} \tag{Eq. 3.5}$$

$$\begin{aligned}
 \frac{d^2}{dt^2}b &= \left(\frac{d}{dt}w\right) \times b + w \times \left(\frac{d}{dt}b\right) \\
 &= \left(\frac{d}{dt}w\right) \times b + w \times (w \times b) \\
 &= (I^{-1}(L \times w + T)) \times b + w \times (w \times b) \\
 &= (b^{\star T}I^{-1})T + (b^{\star T}I^{-1}(L \times w) + w \times (w \times b)) \\
 &= HT + \beta
 \end{aligned} \tag{Eq. 3.6}$$

where $H = b^{\star T}I^{-1}$ and $\beta = (H(L \times w) + w \times (w \times b))$.

Let $X_b = X + b$ be the coordinate of the fixed point in world space. The first and second derivatives of X_b are as follow.

$$\begin{aligned}
 \dot{X}_b &= \frac{d}{dt}X + \frac{d}{dt}b \\
 &= v + w \times b
 \end{aligned} \tag{Eq. 3.7}$$

$$\begin{aligned}
 \ddot{X}_b &= \frac{d^2}{dt^2}X + \frac{d^2}{dt^2}b \\
 &= \frac{1}{m}F + HT + \beta
 \end{aligned} \tag{Eq. 3.8}$$

where F is the external force.

3.6.2 Constraint force equation

The author proposes the general equation form to be:

$$\ddot{C}(Y, t) + k_d \dot{C}(Y, t) + k_s C(Y, t) = 0 \quad (\text{Eq. 3.9})$$

where:

- Y represents the state variables of the rigid cells
- t is the time
- C is a 3D vector representing the constraint energy
- k_d is the damping scalar constant in the range $[0,1]$
- k_s is the spring scalar constant which is a positive number

(Eq. 3.9) is based on the general form by Barzel et al except for the constant coefficients[4]. Figure 3.4 shows the general graphical solution for this second order differential equation (DE).

CHAPTER 3. SOFTCELL CONSTRAINTS

Figure 3.4: Graphical solution of C

It can be seen that the solution will reduce the constraint C to zero. In order to make the two points of the adjacent cells (cell 1 and cell 2) to meet, C can be defined as:

$$C = X_b^1 - X_b^2 \quad (\text{Eq. 3.10})$$

where X_b^1 and X_b^2 are position vectors of the fixed points on cell 1 and cell 2 respectively.

By substituting (Eq. 3.7), (Eq. 3.8) and (Eq. 3.10) into (Eq. 3.9), the resulting form for a pair of adjacent cells 1 and 2 will become:

CHAPTER 3. SOFTCELL CONSTRAINTS

$$\frac{1}{m_1}F_1 + H_1T_1 + \beta_1 - \frac{1}{m_2}F_2 - H_2T_2 - \beta_2 + k_d\alpha + k_s\chi = 0 \quad (\text{Eq. 3.11})$$

where:

- $\alpha = v_1 + w_1 \times b_1 - v_2 - w_2 \times b_2$
- $\chi = X_1 + b_1 - X_2 - b_2$

It suffices to apply forces at the two constrained points to keep them together.

Additional torque is not required. By Newton's third law, a constraint force F_c should be applied to the constrained point of cell 1 and $-F_c$ should be applied to the constrained point of cell 2.

The Newton-Euler equations allow the resultant torque from an applied force at a point that is at perpendicular distance of b from the center of mass of a rigid body to be expressed as $T = b^*F$. Thus, the final form of the point to point constraint equation is:

$$\begin{aligned} \frac{1}{m_1}F_c + H_1(b_1^*F_c) + \beta_1 - \frac{1}{m_2}(-F_c) - H_2(-b_2^*F_c) - \beta_2 + k_d\alpha + k_s\chi &= 0 \\ \left[\frac{1}{m_1} + H_1b_1^* + \frac{1}{m_2} + H_2b_2^* \right] F_c + \beta_1 - \beta_2 + k_d\alpha + k_s\chi &= 0 \quad (\text{Eq. 3.12}) \\ F_c &= \left[\frac{1}{m_1} + H_1b_1^* + \frac{1}{m_2} + H_2b_2^* \right]^{-1} (-(\beta_1 - \beta_2) - k_d\alpha - k_s\chi) \end{aligned}$$

3.7 Orientation Constraint

It seems convenient to reuse the general form in (Eq. 3.9) since the point to point constraint is developed from there. The most natural way to define an orientation constraint from that approach is to align the 3 principal axes of the two cells. Given that u_{ix} , u_{iy} and u_{iz} are the direction cosines of the principal axes of cell i .

The constraint function is:

$$C = \begin{Bmatrix} u_{1x} \cdot u_{2x} \\ u_{1y} \cdot u_{2y} \\ u_{1z} \cdot u_{2z} \end{Bmatrix} - \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} \quad (\text{Eq. 3.13})$$

When the direction cosines are aligned, all dots product becomes 1 and C becomes a zero vector. Subsequently, the first and second derivatives of C can be developed. By substituting the 3 derived terms back into (Eq. 3.9), it can be simplified to the following form (see appendix A for detailed steps):

$$[M] \{T_c\} = \{B\} \quad (\text{Eq. 3.14})$$

where $[M]$ is a 3×3 matrix and $\{B\}$ is a 3D vector. T_c is the unknown torque to be applied to cell 1 and $-T_c$ to cell 2.

However, the author found that it is disadvantageous to use the above approach. The main reason is the costly computation of the matrix $[M]$ for every iteration. $[M]$ is also often ill-conditioned because the system can become overconstrained or

underconstrained. A simple case is when the adjacent pair are already aligned. Any torque that makes the cells rotate one complete turn in opposite directions is a valid solution. Barzel et al proposed to use singular-value decomposition (SVD) to solve the system equation as it usually yields the solution with the smallest magnitude.

SVD is a relatively slow method and will affect interactivity. This thesis presents a novel method that attempts to relax all the cells towards an approximation to the average cell orientation of the softbody.

3.7.1 Finding average orientation

The input required is the set of cell orientations, q_i , represented in unit quaternion form. The author has chosen the quaternion form for its computational advantage. See [41] for an introduction to quaternion. Buss et al. proved the existence and uniqueness properties of a weighted spherical average (average quaternion in this case). They have also developed an iterative algorithm to find the average with linear and quadratic convergence rates [10]. The author found no computational advantage for such accuracy. A simple and efficient approximation is presented here as an alternative.

Let m_i be the mass of cell i and M be total mass of the cells. Since the unit quaternion represent rotation, the author defines the average orientation q_{avg} as the normalized result of the weighted sum of the rotations, i.e.

$$q_{avg} = \frac{\sum_i q_i s_i}{|\sum_i q_i s_i|} \quad (\text{Eq. 3.15})$$

CHAPTER 3. SOFTCELL CONSTRAINTS

where $s_i = \frac{m_i}{M}$.

This definition is reasonable as the author's objective is a plausible behaviour. Results have shown that the cells are still relaxed towards well-defined orientations and producing acceptable visual results that match a normal person's intuition of angular dynamics.

The following procedure finds q_{avg} .

Algorithm 1 To find the average orientation

```

N - total number of cells
M - total mass of the cells
 $q_{identity}$  - identity quaternion
 $q_i$  - orientation of cell i
 $q_{avg} \leftarrow \{0, 0, 0, 0\}$ 

for  $i = 1$  to N do
   $m_i :=$  mass of cell  $i$ 
   $s_i = m_i/M$ 
  if  $q_i \cdot \text{dotproduct}(q_{identity})$  is less than 0 then
     $q_i \leftarrow -q_i$ 
  end if
   $q_{avg} \leftarrow (q_{avg} + q_i s_i) \setminus \setminus \text{sum the weighted quaternions}$ 
end for
 $q_{avg} \leftarrow \frac{q_{avg}}{|q_{avg}|} \setminus \setminus \text{normalize the result}$ 

return  $q_{avg}$ 

```

There may be a chance that q_{avg} turns out to be a zero quaternion, i.e. $|q_{avg}| = 0$.

Thus, normalizing it will result in a divide by zero condition. This is inevitable due to the approximation nature of the algorithm. The author recommends that the last known non-zero q_{avg} is always saved in the simulation. Owing to the time coherence of physics simulation, the next average quaternion should be close to the average

CHAPTER 3. SOFTCELL CONSTRAINTS

quaternion from the previous iteration. Thus, the last saved q_{avg} can be used when the zero quaternion occurs.

It is also important to highlight that for each unique rotation, there are two possible quaternions whose components have opposite signs. This is owing to the fact that a rotation about an axis u by θ angle is identical to a rotation about $-u$ by $-\theta$.

Therefore, it is necessary to choose the sign σ such $\sigma(q_0 \cdot q_{identity}) \leq 0$ before summing the quaternions together. This explains the dot product and negation operations in algorithm 1. Otherwise, it may result in extra spinning[11].

3.7.2 Integration for orientation

q_{avg} acts as a goal orientation for the cells to move towards. Suppose cell i is at orientation q_i , the shortest arc rotation to orientate the cell to q_{avg} is

$$q_{shortestArc} = (q_{avg})(q_i)^{-1}.$$

Let the w_i^g be the required angular velocity for cell i to reach q_{avg} . With w_i^g , the rotation of the body after a time step h is represented by the quaternion

$$\left[\cos \frac{|w_i^g| h}{2}, \sin \frac{|w_i^g| h}{2} \frac{w_i^g}{|w_i^g|} \right] \quad (\text{Eq. 3.16})$$

Let u and s be the axis of rotation and angle of rotation that specifies $q_{shortestArc}$ respectively. w_i^g is given as:

$$w_i^g = \frac{s \cdot u}{h} \quad (\text{Eq. 3.17})$$

CHAPTER 3. SOFTCELL CONSTRAINTS

Setting all cell angular velocities to their respective w_i^g will cause the cells to be at q_{avg} after every iteration. However, the angular momentum of the body is not conserved.

For plausible behavior, the average angular velocity w_{avg} is defined as:

$$w_{avg} = \sum_i \frac{m_i}{M} w_i \quad (\text{Eq. 3.18})$$

The proposed integration equation is given as:

$$w_i(t+h) = w_i(t) * (1 - k_s) + (w_{avg}(t) + w_i^g(t)) * k_s \quad (\text{Eq. 3.19})$$

In (Eq. 3.19), k_s is the stiffness constant in the range $[0,1]$. When $k_s = 1$, the body exhibits rigid behaviour. All cells are aligned to the goal orientation after the end of the iteration. They will continue to move at the angular rate of w_{avg} if they reach the goal orientation. When $k_s < 1$, the second term of $(w_{avg}(t) + w_i^g(t))$ is added to relax the cell orientation towards the goal orientation.

The following algorithm summarizes the integration procedure.

CHAPTER 3. SOFTCELL CONSTRAINTS

Algorithm 2 Integration for orientation

N - total number of cells
 M - total mass of cells
 q_{avg} - average orientation
 w_i - angular velocity of cell i
 m_i - mass of cell i
 q_i - orientation of cell i
 h - timestep
 k_s - stiffness constant

$w_{avg} \leftarrow \sum_i \frac{m_i}{M} w_i$
for $i = 1$ to N **do**
 if $q_i \cdot \text{dotproduct}(q_{avg})$ is less than 0 **then**
 $q_i \leftarrow -q_i$
 end if
 // $q_{shortestArc}.u$ is the unit axis and $q_{shortestArc}.s$ is rotation angle
 $q_{shortestArc} \leftarrow (q_{avg})(q_i)^{-1}$
 $w_i^g \leftarrow \frac{(q_{shortestArc}.u)(q_{shortestArc}.s)}{h}$
 $w_i \leftarrow w_i(1 - k_s) + (w_{avg} + w_i^g)k_s$
end for

Chapter 4

Extension

4.1 Fracturing

In continuum theories, fracturing occurs when the stress in a softbody builds up beyond the fracture stress point[14]. The magnitude of the constraint function in (Eq. 3.10), i.e. $|C|$, provides a local stress measure that quantifies how far apart the cells are. To support fracture, the point to point and orientation constraints for an adjacent pair are both removed when the following condition is met:

$$|C| \geq \sigma_{fracture} : \{\sigma_{fracture} > 0\} \quad (\text{Eq. 4.1})$$

The positive scalar $\sigma_{fracture}$ determines the strength of the material. Re-tessellation of the fractured bodies is beyond the scope of this thesis. Interested readers may refer to other existing works[33, 32, 36, 27].

4.2 Clustering

To extend the range of deformation, the cells can be divided into overlapping clusters. The orientation of each cluster is updated separately based on the method discussed in

CHAPTER 4. EXTENSION

Section 3.7. One way to generate the clusters is to subdivide the axis aligned bounding box of the mesh into overlapping cubic regions. All cells whose cell center is in the same cubic region are grouped as one cluster.

Figure 4.1 and 4.2 illustrate how the cubic regions of a bounding box can be divided into different numbers of overlapping regions. The higher the number of clusters the better the flexibility of deformation is.

Figure 4.1: Cubic regions with 4 clusters

Figure 4.2: Cubic regions with 6 clusters

Chapter 5

Results

The author has integrated the proposed method into a simulation environment based on the bullet physics engine(<http://www.bulletphysics.com>). All test scenarios presented in this thesis are experimented on a PC Core2 Duo 2GHz and NVIDIA GeForce 8600 graphic card. A video demonstration accompanying this thesis illustrates the results better.

CHAPTER 5. RESULTS

Fracturing

Figure 5.1: Fracture

Figure 5.1 shows an example of the fracture effect. The object fractures when its top right corner was pulled off with a force.

CHAPTER 5. RESULTS

Clustering

Figure 5.2: Unclustered deformation

CHAPTER 5. RESULTS

Figure 5.3: Clustered deformation

Figure 5.2 shows the screenshots of a cuboid mesh with 290 vertices and was discretized into 325 cells. A force was applied to the top of the cuboid by user control. Figure 5.3 shows the same cuboid but with clustering. 192 clusters were used. The experiment shows that clustering results in more deformation details. Notice that the clustered

CHAPTER 5. RESULTS

cuboid is able to undergo more twisting deformation in contrast to the unclustered cuboid. The downside is that the increase in the number of clusters will also increase the simulation cost.

Robustness

Figure 5.4: Object under robust deformation

From left to right, Figure 5.4 is a sequence of images of successive iterations. The leftmost image shows that the cells of the snowman model were initially collapsed into a single point and were given random orientations. The soft constraint forces were able to restore the model at the end. This proves that the simulation can remain stable even if the object is under extreme deformation.

The same experiment in Figure 5.4 was repeated with a unit cube. The cube was discretized uniformly with a dimension of $4 \times 4 \times 4$ cells. The unit cube was tested for congruency with the original shape. Based on the theory of isometry, two objects are congruent if the distance between any pair of points in one object is the same as the corresponding pair of points in another object[6]. The author established a metric of congruency, D , as the sum of the absolute difference between the distance of every unique pair of points in the original shape and deformed shape, i.e:

CHAPTER 5. RESULTS

$$D = \sum_{i=1}^n \sum_{j=i+1}^n ||x_i - x_j| - |x'_i - x'_j|| \quad (\text{Eq. 5.1})$$

where:

- n is the number of vertices of the underlying shape.
- x_i and x_j are the vertices of the original shape.
- x'_i and x'_j are the vertices of the deformed shape.

After 500 iterations, D converged to approximately 0.00001 for the unit cube. This suggests that the approach restores the deformed model to the original shape.

Performance

Figure 5.5: Complex simulation environment

Figure 5.5 depicts a scene of 50 snowmen, 13000 vertices and 2000 softcells. The simulation was running at 35 frames per second in real time. On average, the computation for the softcell constraints accounted for approximately 25% of the simulation time. The main bottleneck of the simulation in the test scenario was resolving the contact during collision. Solving the collision contacts took up 45% of the simulation time. The simulation cost is linear to the number of cells as each cell is coupled to at least 1 other cell and to at most 6 cells.

Chapter 6

Conclusion and Future Work

This thesis has presented softcell constraints, a novel set of formulations for fast and robust simulation of deformable bodies. The formulations assume the underlying model representation is a collection of elastically coupled cuboid cells. Even when under extreme deformation, the method is highly robust. This is because of two features:

- The cells are rigid. Thus they do not suffer from degeneracies like FEM elements.
- The cells are always relaxed towards well-defined positions and orientations.

Collisions are easy to handle as it is based on cuboid primitives. Thus, it eliminates the burden to implement complex collision detection technique and contact handling. Furthermore, complex objects and a modest number of cells can also be handled at interactive rates. In summary, this method combines ease of implementation, robustness and modest performance. This makes it particularly interesting for games.

CHAPTER 6. CONCLUSION AND FUTURE WORK

Figure 6.1: Tunneling artifacts

The main limitation is that when an object is greatly stretched, the cells will become far apart. Tunneling artifacts will manifest as other objects may pass through the cell gaps. This is illustrated by figure 6.1. Refinement of the discretization during the simulation is necessary to maintain effective collision handling. However, the author believes this is a challenging task. The problem is less visible if the object is not stretched too far.

Recall that the constraint forces/torques affects only the cells and not on the mesh directly. Furthermore, the deformation energy does not give a measure of the volume of the deformed object. No method was successfully found to guarantee volume conservation. Thus, the method can only produce plausible behavior at best. It is generally acceptable in games but unsuitable for applications that require precise modeling.

Although the method can handle complex scenes, collision handling was found to be a performance bottleneck. On average, it takes up 45% of the simulation time. Further research on the collision handling will make this method more attractive. Another potential solution to improve performance is to parallelize the implementation. Based on two observations, the author believes this is feasible:

CHAPTER 6. CONCLUSION AND FUTURE WORK

- The data structures for the formulations are fairly simple.
- The computation of the constraint forces do not depend on the results of each other.

While fracture has been explored as one of the extensions, the author has not focused on effective re-tessellation of the fractured mesh. It is another promising direction for future work.

Plasticity is also a very useful extension that is worth investigating. It only requires minor changes to the formulations. However, the cell gaps will potentially grow under plastic deformation. This will also lead to the tunneling artifact that was mentioned previously.

Point sampled surfaces are also increasingly popular in computer animations due to the availability of advanced 3D scanning devices. Readers are referred to [19] for a good introduction to point based graphics. To perform deformation simulation on point sampled surfaces, the most popular approach is the Moving Least Square (MLS) method[31]. Guo et al. have coupled the octree structure with the MLS method for efficient deformation of point sampled surfaces[20]. It will be interesting to see how the softcell constraints on an adaptive octree can lend itself to real time meshless deformation of point sampled surfaces.

Appendix A

Derivation of orientation constraint

Given that u_{ix} , u_{iy} and u_{iz} are the direction cosines of the principal axes of cell i . The orientation constraint is given as:

$$C = \begin{Bmatrix} u_{1x} \cdot u_{2x} \\ u_{1y} \cdot u_{2y} \\ u_{1z} \cdot u_{2z} \end{Bmatrix} - \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} \quad (\text{Eq. A.1})$$

Consider u_{ix} . It can be treated as a fixed vector from the center of mass of the rigid cell. Therefore, $\dot{u}_{ix}$ and $\ddot{u}_{ix}$ follows from (Eq. 3.5) and (Eq. 3.6) respectively.

$$\dot{u}_{ix} = w^* u_{ix} \quad (\text{Eq. A.2})$$

$$\ddot{u}_{ix} = H_{ix} T + \beta_{ix} \quad (\text{Eq. A.3})$$

where:

- $H_{ix} = u_{ix}^{*T} I^{-1}$
- $\beta_{ix} = H_{ix} (L \times w) + w \times (w \times u_{ix})$

CHAPTER A. DERIVATION OF ORIENTATION CONSTRAINT

The derivatives of u_{iy} and u_{iz} are defined in the same way. By using product rule, $\dot{C}$ and $\ddot{C}$ can be expanded.

$$\dot{C} = \begin{Bmatrix} u_{1x} \cdot \dot{u}_{2x} + u_{1x} \cdot \dot{u}_{2x} \\ u_{1y} \cdot \dot{u}_{2y} + u_{1y} \cdot \dot{u}_{2y} \\ u_{1z} \cdot \dot{u}_{2z} + u_{1z} \cdot \dot{u}_{2z} \end{Bmatrix} \quad (\text{Eq. A.4})$$

$$\ddot{C} = \begin{Bmatrix} u_{1x} \cdot \ddot{u}_{2x} + u_{2x} \cdot \ddot{u}_{1x} + 2u_{1x} \cdot \dot{u}_{2x} \\ u_{1y} \cdot \ddot{u}_{2y} + u_{2y} \cdot \ddot{u}_{1y} + 2u_{1y} \cdot \dot{u}_{2y} \\ u_{1z} \cdot \ddot{u}_{2z} + u_{2z} \cdot \ddot{u}_{1z} + 2u_{1z} \cdot \dot{u}_{2z} \end{Bmatrix} \quad (\text{Eq. A.5})$$

$$\ddot{C} = \begin{Bmatrix} u_{2x} \cdot H_{1x}T_c - u_{1x} \cdot H_{2x}T_c + u_{1x} \cdot \beta_{2x} + u_{2x} \cdot \beta_{1x} + 2u_{1x} \cdot \dot{u}_{2x} \\ u_{2y} \cdot H_{1y}T_c - u_{1y} \cdot H_{2y}T_c + u_{1y} \cdot \beta_{2y} + u_{2y} \cdot \beta_{1y} + 2u_{1y} \cdot \dot{u}_{2y} \\ u_{2z} \cdot H_{1z}T_c - u_{1z} \cdot H_{2z}T_c + u_{1z} \cdot \beta_{2z} + u_{2z} \cdot \beta_{1z} + 2u_{1z} \cdot \dot{u}_{2z} \end{Bmatrix}$$

By substituting the derived terms into (Eq. 3.9), the result is:

$$\begin{bmatrix} u_{2x} \cdot H_{1x} - u_{1x} \cdot H_{2x} \\ u_{2y} \cdot H_{1y} - u_{1y} \cdot H_{2y} \\ u_{2z} \cdot H_{1z} - u_{1z} \cdot H_{2z} \end{bmatrix} T_c = - \begin{Bmatrix} u_{1x} \cdot \beta_{2x} + u_{2x} \cdot \beta_{1x} + 2u_{1x} \cdot \dot{u}_{2x} \\ u_{1y} \cdot \beta_{2y} + u_{2y} \cdot \beta_{1y} + 2u_{1y} \cdot \dot{u}_{2y} \\ u_{1z} \cdot \beta_{2z} + u_{2z} \cdot \beta_{1z} + 2u_{1z} \cdot \dot{u}_{2z} \end{Bmatrix} \quad (\text{Eq. A.6})$$

T_c is the restoring torque to be applied to cell 1. $-T_c$ will be applied to cell 2.

References

- [1] D. Baraff and A. Witkin. Dynamic simulation of non-penetrating flexible bodies. *ACM SIGGRAPH Computer Graphics*, 26(2):303–308, 1992.
- [2] D. Baraff and A. Witkin. Large steps in cloth simulation. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54, 1998.
- [3] A.W. Bargteil, C. Wojtan, J.K. Hodgins, and G. Turk. A finite element method for animating large viscoplastic flow. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), 2007.
- [4] R. Barzel and A.H. Barr. A modeling system based on dynamic constraints. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 179–188. ACM New York, NY, USA, 1988.
- [5] K.J. Bathe. Finite element procedures. *Englewood Cliffs, New Jersey*, 1996.
- [6] FS Beckman and DA Quarles Jr. On isometries of Euclidean spaces. *Proceedings of the American Mathematical Society*, pages 810–815, 1953.
- [7] P.J. Besl and H.D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on pattern analysis and machine intelligence*, 14(2):239–256, 1992.
- [8] M. Botsch, M. Pauly, M. Gross, and L. Kobbelt. Primo: Coupled prisms for intuitive surface modeling. *Geometry Processing 2006*, page 11.
- [9] D. Bourguignon and M.P. Cani. Controlling Anisotropy in Mass-Spring Systems. *Computer Animation and Simulation 2000: Proceedings of the Eurographics Workshop in Interlaken, Switzerland, August 21-22, 2000*, 2000.

REFERENCES

- [10] S.R. Buss and J.P. Fillmore. Spherical averages and applications to spherical splines and interpolation. *ACM Transactions on Graphics (TOG)*, 20(2):95–126, 2001.
- [11] D.H. Eberly. *Game physics*. Morgan Kaufmann, 2004.
- [12] DW Eggert, A. Lorusso, and RB Fisher. Estimating 3-D rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, 9(5):272–290, 1997.
- [13] C. Ericson. *Real-time collision detection*. Morgan Kaufmann, 2005.
- [14] YC Fung. A first course in continuum mechanics. *Englewood Cliffs, NJ, Prentice-Hall, Inc., 1977. 351 p.*, 1977.
- [15] J. Georgii and R. Westermann. A multigrid framework for real-time simulation of deformable bodies. *Computers & Graphics*, 30(3):408–415, 2006.
- [16] S. Gibson and B. Mirtich. A survey of deformable modeling in computer graphics. *MERL, TR-97*, 19, 1997.
- [17] H. Goldstein, C. Poole, J. Safko, and S.R. Addison. Classical mechanics. *American Journal of Physics*, 70:782, 2002.
- [18] E. Grinspun, A.N. Hirani, M. Desbrun, and P. Schröder. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 62–67. Eurographics Association Aire-la-Ville, Switzerland, Switzerland, 2003.
- [19] M. Gross and H. Pfister. *Point-Based Graphics (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2007.
- [20] X. Guo and H. Qin. Real-time meshless deformation. *Computer Animation and Virtual Worlds*, 16(3-4):189–200, 2005.
- [21] M. Hong, S. Jung, M.H. Choi, and S.W.J. Welch. Fast Volume Preservation for a Mass-Spring System. *IEEE COMPUTER GRAPHICS AND APPLICATIONS*, pages 83–91, 2006.

REFERENCES

- [22] B.K.P. Horn et al. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.
- [23] PM Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, 1995.
- [24] P.M. Isaacs and M.F. Cohen. Controlling dynamic simulation with kinematic constraints. *ACM SIGGRAPH Computer Graphics*, 21(4):215–224, 1987.
- [25] LucasArts. Lucasarts fracture, <http://www.lucasarts.com/games/fracture/>. 2008.
- [26] I. Millington. *Game physics engine development*. Morgan Kaufmann, 2007.
- [27] N. Molino, Z. Bao, and R. Fedkiw. A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics (TOG)*, 23(3):385–392, 2004.
- [28] M. Muller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler. Stable real-time deformations. *Symposium on Computer Animation: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 21(22):49–54, 2002.
- [29] M. Müller and M. Gross. Interactive virtual materials. *Proceedings of Graphics Interface 2004*, pages 239–246, 2004.
- [30] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. *Proceedings of ACM SIGGRAPH 2005*, 24(3):471–478, 2005.
- [31] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 141–151, 2004.
- [32] M. Müller, L. McMillan, J. Dorsey, and R. Jagnow. Real-time simulation of deformation and fracture of stiff materials. In *Computer Animation and Simulation 2001: Proceedings of the Eurographics Workshop in Manchester, UK, September 2-3, 2001*, page 113. Springer, 2001.

REFERENCES

- [33] M. Muller, M. Teschner, and M. Gross. Physically-based simulation of objects represented by surface meshes. In *Computer Graphics International, 2004. Proceedings*, pages 26–33, 2004.
- [34] A. Nealen, M. Muller, R. Keiser, E. Boxerman, and M. Carlson. Physically Based Deformable Models in Computer Graphics. *Computer Graphics Forum*, 25(4):809–836, 2006.
- [35] NVIDIA. Nvidia physx, <http://developer.nvidia.com/object/physx.html>. 2008.
- [36] J.F. O’Brien and J.K. Hodgins. Graphical modeling and animation of brittle fracture. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 137–146. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 1999.
- [37] J.C. Platt and A.H. Barr. Constraints methods for flexible models. *ACM SIGGRAPH Computer Graphics*, 22(4):279–288, 1988.
- [38] W.H. Press, S.A. Teukolsky, and W.T. Vetterling. *Numerical recipes in C++: the art of scientific computing*. Cambridge University Press.
- [39] A.R. Rivers and D.L. James. FastLSM: fast lattice shape matching for robust real-time deformation. *ACM Transactions on Graphics (TOG)*, 26(3), 2007.
- [40] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proc. 3DIM*, pages 145–152, 2001.
- [41] K. Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254. ACM New York, NY, USA, 1985.
- [42] K. Shoemake and T. Duff. Matrix animation and polar decomposition. *Proc. of the 1992 Graphics Interface Conference*, pages 245–254, 1992.
- [43] J. Spillmann, M. Becker, and M. Teschner. Efficient updates of bounding sphere hierarchies for geometrically deformable models. *Journal of Visual Communication and Image Representation*, 18(2):101–108, 2007.

REFERENCES

- [44] D. Steinemann, M.A. Otaduy, and M. Gross. Fast adaptive shape matching deformations. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 87–94, 2008.
- [45] M. Teschner, B. Heidelberger, M. Muller, and M. Gross. A versatile and robust model for geometrically complex deformable solids. *Computer Graphics International, 2004. Proceedings*, pages 312–319, 2004.
- [46] Gran Turismo. Gran turismo: The official site, <http://www.gran-turismo.com/>. 2008.
- [47] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, 1991.
- [48] M. Wicke, D. Steinemann, and M. Gross. Efficient animation of point-sampled thin shells. In *Computer Graphics Forum*, pages 667–676. Blackwell Publishing, 2005.
- [49] A. Witkin, M. Gleicher, and W. Welch. Interactive dynamics. *ACM SIGGRAPH Computer Graphics*, 24(2):11–21, 1990.
- [50] A. Witkin and M. Kass. Spacetime constraints. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 159–168. ACM New York, NY, USA, 1988.
- [51] Y. Zhuang and J. Canny. Real-time simulation of physically realistic global deformation. In *IEEE Vis*, volume 99, pages 24–29, 1999.