A Position Based Approach
to Ragdoll Simulation

Fiammetta Pascucci

LiTH-ISY-EX- -07/3966- -SE

June 2007
A position Based Approach to Ragdoll Simulation

Fiammetta Pascucci

June 2007

Supervisor: Ingemar Ragnemalm

Examinator: Ingemar Ragnemalm
Abstract

Create the realistic motion of a character is a very complicated work.

This thesis aims to create interactive animation for characters in three dimensions using position based approach. Our character is pictured from ragdoll, which is a structure of system particles where all particles are linked by equidistance constraints.

The goal of this thesis is observed the fall in the space of our ragdoll after creating all constraints, as structure, contact and environment constraints.

The structure constraint represents all joint constraints which have one, two or three Degree of Freedom (DOF). The contact constraints are represented by collisions between our ragdoll and other objects in the space. Finally, the environment constraints are represented by means of the wall constraint.

The achieved results allow to have a realist fall of our ragdoll in the space.

Keywords: Particle System, Verlet’s Integration, Skeleton Animation, Joint Constraint, Environment Constraint, Skinning, Collision Detection.
I would like to thank all the people who have contributed to making my master thesis work. In particular, I want to thank the person who has always believed in me. This person has always encouraged me to move forward, has always given me a lot of love and has been with me throughout difficult times. This person is everything to me, and I hope we can stay together for the rest of our lives. Piergiorgio, thank you very much.

I would like to thank my parents because they gave me the opportunity to come to Sweden and they have always been there for me, supporting me in my goals.

This work could not have been completed without the encouragement, support, enthusiasm, and expert guidance of my brilliant supervisors Ingemar Ragnemalm. I thank him for the patience he showed in reading the report and answering my questions and for his valuable suggestions.

I would like to give thanks to my Italian Professor Marco Shaerf because he gave me this great opportunity to pursue my Master's thesis in Sweden.

Many thanks to Marco Fratarcangeli for his suggestions and his continuous helpfulness.
Contents

1 Introduction ................................................................. 1
   1.1 Animation and tools articulated characters ..................... 1
   1.2 Goals's thesis ..................................................... 3
   1.3 Structure of the thesis ........................................... 4

2 Ragdoll physics ............................................................ 7
   2.1 Introduction to Ragdoll system ................................. 7
   2.2 Our Ragdoll ....................................................... 8
       2.2.1 The structure of a Ragdoll ............................. 8
   2.3 Conclusion ....................................................... 9

3 Mathematical tools ........................................................ 11
   3.1 Particle System .................................................. 11
   3.2 Numerical Integration ............................................ 12
       3.2.1 Euler’s method ............................................ 13
       3.2.2 Verlet’s method .......................................... 14
   3.3 Rotation in 3D space ............................................ 16
   3.4 Euler angles ...................................................... 17
   3.5 Matrix ............................................................. 18
   3.6 Quaternion ......................................................... 20
   3.7 Conclusion ....................................................... 22

4 Character articulation ..................................................... 23
   4.1 Articulated figure ............................................... 23
   4.2 Model of skeleton ............................................... 24
   4.3 Ragdoll’s constraints ............................................ 24
       4.3.1 Principles of implementation ........................... 24
       4.3.2 Resolve the problem that one constraint breaches another 25
       4.3.3 Sequence joints .......................................... 26
       4.3.4 Constraint for Shoulder joint ........................... 26
       4.3.5 Constraint for Elbow joint .............................. 31
   4.4 Conclusion ....................................................... 33
## CONTENTS

5 Coat the ragdoll 35  
5.1 The ragdoll’s shape 35  
5.2 Rigid body parts 35  
5.3 Skinning 36  
5.4 Other methods 37  
5.5 Implementation with rigid body parts 38  
5.6 Simplified skin from points method and implementation 43  
5.7 Conclusion 45  

6 Collision handling 47  
6.1 Collision detection 47  
6.2 Problem of nonpenetration constraints 48  
6.3 Handling collision and penetration by projection 49  
6.4 Conclusion 50  

7 Project and Implementation 53  
7.1 Used tools 53  
7.2 Code Structure 54  
7.2.1 Insert constraint 55  
7.2.2 Coat the ragdoll 59  
7.3 Conclusion 60  

8 Result 61  
8.1 Summary 61  
8.1.1 Motion constraints 61  
8.1.2 Coat the ragdoll 62  
8.1.3 Fall in the space and between objects 62  
8.2 Conclusion 63  

9 Conclusion and Future works 67  

A Software 69  
A.1 Set up the development environment 69  
A.1.1 Microsoft Visual Studio 2005 69  
A.1.2 TortoiseCvs 69  
A.1.3 IdoLib 70  
A.1.4 VcgLib 70  
A.1.5 SDL 73  
A.1.6 GLUT 73  
A.1.7 GLEW 73  
A.2 Run the project 74
## CONTENTS

### B OpenGL

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1 OpenGL</td>
<td>75</td>
</tr>
<tr>
<td>B.2 OpenGL Structure</td>
<td>76</td>
</tr>
<tr>
<td>B.2.1 Convention</td>
<td>76</td>
</tr>
<tr>
<td>B.2.2 Syntax</td>
<td>77</td>
</tr>
<tr>
<td>B.3 Libraries on OpenGL</td>
<td>77</td>
</tr>
</tbody>
</table>

### Bibliography

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>79</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter makes a brief introduction to the project developed in this thesis. Firstly, there will be showed a general panoramic of animation techniques and besides we will showed the main goals of this work, after we will be described the chapter’s structure together with a short introduction of them.

1.1 Animation and tools articulated characters

The animation of a virtual characters is a process very complex.

The human motion [1] is the result of many degrees of freedom. The human body is made up about 206 bones and it’s controlled by a complex nervous system.

In computer graphics the definition and control of motion are two basic themes of notable interest in scientific research. The animation’s problem of virtual characters is a continuous interest for researcher, because the main trouble is to decide the level of detail with which to cope with problem.

In this work we use a simple representation of the human body, in fact we used a "skeleton". The character’s skeleton [2] is a pose-able framework of bones connected by articulated joints. The joints allow relative movement within the skeleton.

We have created our skeleton according to the article of Thoman Jakobsen [3]. This article builds his physical engine on a position based approach.

His central idea is the right of combination of several techniques that all benefit from each other.

The techniques mentioned are: Verlet integration, handling collision and penetration by projection, simple constraint solver using relaxation, modeling rigid body as particles with constraints.

In the following we make a brief introduction to different animation techniques, the animation technique called key-framing, another technique called motion capture and in the last the animation technique called physically based animation.
Introduction

Key-framing

The technique called Key Framing is the simplest form of animating an object. An object has a beginning state and will be changing over time, in position and in the color. The most important concept in this technique are the keyframes.

These keyframes are the steps to create the animation. Another important concepts are the extremes of keyframes, in fact these represent the frames necessary to fix the critical point in the animation among keys.

The sequence of keyframes defines which movement the spectator will see. In the Figure 1.1 is shown the particular sequence of a set of keyframes.

![Figure 1.1: A particular sequence of keyframes](image)

Motion Capture

Another important animation technique is called Motion Capture. This is a technique of digitally recording movements.

Is a tool to realize realistic animation. This technique is very important because realizes a similar human animation because by means of special dress captures the motion from human actors to create natural movements.

This animation technique has been used in many film for example in: "The Lord of The Rings", where the actor Andy Serkis represents the character of "Gollum".

This technique has a big drawback, in fact the cost of the software and equipment can be prohibitive for small productions.

In the Figure 1.2 is shown the actor Andy Serkis, he dresses with a special equipment for representing the special character.
1.2 Goals’s thesis

Figure 1.2: The actor Andy Serkis dressed with a special equipment for representing the character Gollum in the film “The Lord of The Rings”

Physically based animation

In this type of animation is integrated the animations with physically simulated movements. This technique make possible a realistic animations because includes the physical simulation of the motion.

A particular case of physically based animation is a Ragdoll physics. This type of animation is used in many video games. The character is not conscious but he is submitted to the laws of physics. In Figure is shown a possible fall of the ragdoll.

Figure 1.3: The fall of ragdoll

More about that is explained in the next chapter 2.

1.2 Goals’s thesis

The goals of the presented work are:

- Create motion constraints.
Introduction

- Coat the skeleton with geometric primitives.
- Realize a realistic fall.

In order to realize the first goal, the motion constraints are specified by limiting the angle rotation belonging to the skeleton’s joints. Each joint has got an independent degree of freedom (DOF\(^1\)).

The second goal is coated the skeleton with a geometric primitives, which will be done by a proposed method. This latter is a simplified skin from points.

The third goal is to realize a realistic fall, we presented the skeleton’s fall on the plane, and the skeleton’s fall between some objects in the space as well.

1.3 Structure of the thesis

The schema belonging to the report is the following:

- **Chapter 2** contains a description of our Ragdoll, in which we explain the structure of it.

- **Chapter 3** enunciates some important mathematical concepts. In fact, in this chapter, we explain firstly the numerical integration and then we discuss the advantages and the drawbacks of Euler integration and Verlet integration. Moreover, we describe the most common approaches used today for making rotations in three dimensions: Euler angles, matrix representation and the quaternion representation.

- **Chapter 4** describes the tools used for character’s articulated. We outline the problem of constraints and we present the method that we used to articulated our ragdoll. In particle we explain the method constraint for shoulder joint and constraint for elbow joint.

- **Chapter 5** contains a brief introduction about some methods to coat the skeleton and then we present our used method for coating our Ragdoll.

- **Chapter 6** describes the problem of collision between our Ragdoll and others objects. We present our approach as well.

- **Chapter 7** explains the total project. We outline the development environment, the libraries used and the method that we implemented.

- **Chapter 8** describes the test case and we comment some screenshots.

- **Chapter 9** is the last part of this report and contains the conclusion and possible future improvements of the work.

\(^1\)Degree of freedom could be one, two or three, it is depend of type of considered joints. The DOFs represents the possible range of motion joint.
1.3 Structure of the thesis

- *Appendix A* contains a detailed description of the installation software and the development environment that we used. We outline all libraries that we used.

- *Appendix B* contains the brief introduction of library OpenGL.
Chapter 2

Ragdoll physics

In this chapter, we give a brief introduction to ragdoll physics explaining the departure’s point of this work citing the important article of Thomas Jakobsen [3]. In conclusion we explain how we have made the our Ragdoll, modelling the skeleton’s topology by interconnection particles with stiff constraints.

2.1 Introduction to Ragdoll system

Behind the name Ragdoll is implicit the concept of ragdoll physics[4]. This implies a procedural animation system 1 replacing old static animation of characters.

The term ragdoll [5] comes from the fact that the articulated systems, due to the limits of the solvers used, tend to have little joint stiffness. It’s simply to make a death sequence more realistic.

The first game to exhibit ragdoll physics was the Jurassic Park licensed game Jurassic Park: Trespasser, which received very polar opinions though most were negative. The game had terrible bugs but was remembered for being a pioneer in video-game physics engines.

The important advantage that ragdoll offers over traditional animations is that it allows much more correct interaction with the surrounding environment.

In general, a ragdoll is a collection of multiple particles where each pair of particles is interconnected by bones. In computer graphics, such technique is called skeletal animation system2.

By connecting pairs of particles by a stiff constraints, it possible to develop a complete model of articulated human body.

The particles are the significant parts of human body, and the stiff constraints represent the bones.

---

1 A procedural animation is a type of computer animation. This approach is used to generate animation dynamically in real-time, not limited to a fixed set of actions.

2 In computer animation skeletal animation [6] is a technique in which a character is represented in two parts: a surface representation used to draw the character (called the skin) and a hierarchical set of bones used for animation only (called the skeleton).
2.2 Our Ragdoll

In this work, the ragdoll model that we use was created according to the approach described of Thomas Jakobsen\(^3\) in [3]. This method has been used to create a series of video game with name Hitman [9]. Jakobsen developed a physics engine with a position based approach. His main idea was to use verlet integration and manipulate positions directly, more about verlet integration is explained in chapter 3.

In the following we explain how our ragdoll has been designed.

We made the type of ragdoll described of Thomas Jakobsen’s article [3]. Figure 2.1 shows a sketch of a ragdoll with particles and constraints. This is the same configuration used in Hitman for representing the human anatomy.

![Figure 2.1: Outline of the ragdoll (adapted from [3])](image)

2.2.1 The structure of a Ragdoll

We will now describe the structure of our ragdoll in more detail. Our physical model is composed by system of particles [10]. Each particle, on the whole system, has got an own mass, an own position, and finally an own velocity. All particles respond to a set of force, such as the gravity force, damp force, friction force, etc., but these have not spatial dimension. These particles are points that can move along the space, the movement of them is checked by the three principles of dynamic.

In our case, we have sixteen particles and twentyfour stiff constraints. The particles represent: head, right shoulder, left shoulder, right elbow, left elbow, right hip, left hip, right lamb, left lamb, right knee, left knee, right foot, left foot. The system of interconnections between two particles are called stiff constraints.

---

\(^3\)Thomas Jakobsen [7] [8] is a mathematician, and computer programmer, assistant professor at the Technical University of Denmark and head of research and development at IO Interactive. His notable work includes designing the physics engine and 3-D pathfinder algorithms for Hitman: Codename 47.
2.3 Conclusion

which represents an equidistance constraint. The equidistance constraint is a rigid link among two particles with fixed distance:

- equidistance constraint: \( |X_2 - X_1| = d \)

where the \( X_2 \) and \( X_1 \) represent the position of two particles and \( d \) represents the distance between them. During the simulation, after one integration step, the separation distance between the particles might have become invalid.

In fact we might have two invalid cases:

- the particles are too close each other.
- the particles are too far each other.

To obtain the right distance, in order to satisfy the constraints, we project the particles in a valid position.

For creating our ragdoll, we insert sixteen particles and we put equidistance constraints between selected pairs of particles. Figure 2.2 shows the front view of the ragdoll, and here is shown also the lateral view of the ragdoll.

![Figure 2.2: Front view of ragdoll and lateral view of ragdoll](image)

2.3 Conclusion

In this section we explained the point of departure for this work. We gave a general panoramic about the state of the art. We discussed how we created our ragdoll.

In the next section, we will discuss about some important mathematical tools.
Ragdoll physics
Chapter 3

Mathematical tools

In this chapter, we explain some mathematical tools really important for this work and in general on the computer graphic field.

Firstly, we make a brief overview of our particle’s system discussing about numerical integration methods and representation for rotations on three dimensions[11].

We will explain the advantages and the drawbacks of some integration methods, such as:

- Euler Integration (with a little variations)
- Verlet Integration

Moreover, we will discuss about the advantages and the drawbacks of the most common approaches used today in order to make a rotations in three dimensions, these are:

- Euler Angles.
- Matrix representation.
- Quaternion representation.

3.1 Particle System

As mentioned in chapter 2, we consider our ragdoll shaped from a set of particles and a set of stiff constraints. All particles have one position \( x \) and velocity \( v \), these two parameters are two vectors in the space.

If we consider the Newton’s second law of motion [10] \( f = ma \) we have \( \ddot{x} = \frac{f}{m} \), this formula is the differential equation of second order in \( x \).

This formula can become of first order if we consider these relation:

- \( \dot{v} = \frac{f}{m} \)
- \( \dot{x} = v \).
These two equations define the movement of one particle in the space.

The movement of one particle is described from the pair $[\dot{x}, \dot{v}]$. The position and the velocity can be concatenated from a six vector. This position/velocity product space is called *phase space*. In conclusion a system of $n$ particles is described by $n$ copies of equation, concatenated to form a $6n$-long vector.

### 3.2 Numerical Integration

Numerical integration [12] takes care studies the numerical solution of ordinary differential equations (ODEs). Often differential equations cannot be solved analytically, in which case we have to satisfy ourselves with an approximation to the solution. In a canonical initial value the behavior of system is described by an ordinary differential equation (ODE) to the form:

- $\dot{x} = f(x, t)$.

where $f$ is a known function, $x$ is the state of the system and $\dot{x}$ is the derivate. Typically, $x$ and $\dot{x}$ are vectors. In an initial value problem we are given $x(t_0) = x_0$ at some starting time $t_0$.

The problem in two dimension is easy because $x(t)$ sweeps out a curve that describes the motion of a point $p$ in the plane. At any point $x$ the function $f$ can be evaluated to provide a two-vector, so $f$ defines a vector field on the plane. The Figure 3.1 describes what said.

![The derivative function \( \dot{x} = f(x, t) \) forms a vector field.](image)

Figure 3.1: Vector field generated of the derivative function (adapted [10] )

In all points $x$ we can figure out the function $f$, which describes the vector field on the plane. The vector $\dot{x}$ is the velocity that the moving point $p$ long the curve $x$.

The numerical solution of differential equation we take discrete interval of time, called *time step*. In any step, by means of $f$, we figure out the increment $x$ ($\Delta x$) in a interval of time $\Delta t$. After we increment of $\Delta x$ the value of $x$:

$$x(t + \Delta t) = x(t) + f(x, t)\Delta t$$
3.2 Numerical Integration

In conclusion in numerical methods the function $f$ is regarded as a black box where we provide numerical values for $x$ and $t$, receiving in return a numerical value for $\dot{x}$.

In the following we show Euler’s method (with a little variations) and the Verlet’s method.

3.2.1 Euler’s method

*Euler’s method* [10] is the simplest numerical procedure for solving ordinary differential equations (ODEs) with a given initial value. This method started to the idea of using linear approximations to model the path of a function, Euler proposed the use of successive linear approximations to model a function’s path.

In the following we explain the advantages and the drawbacks.

Let our initial value for $x$ be denoted by $x_0 = x(t_0)$ and our estimate of $x$ at a later time $t_0 + \Delta t$, that we called $x(t_0 + \Delta t)$.

Euler’s method simply computes $x(t_0 + \Delta t)$ by taking a step in the derivate direction:

$$x(t_0 + \Delta t) = x_0 + \dot{x}\Delta t$$

Instead of the real integral curve, $p$ follows a polygonal path, obtained by evaluating the derivative at the beginning of each leg, here the accuracy of the solution degrades as the size of the time step increases.

This is shown in the Figure 3.2 in the first panel. Although the Euler’s method is very simply, this method commits a some error not unimportant. In fact this method in not accurate.

We consider the case of a two dimensions function $f$ whose integral curves are concentric circles. A point $p$ governed by $f$ is supposed to orbit forever on whichever circle it started on. Instead, with each Euler step, $p$ will move on a straight line to a circle of larger radius, so that its path will follow an outward spiral. Reducing the stepsize will slow the rate of this outward drift, but never eliminate it.

Figure 3.2, second panel, shows the curves from concentric circles.

Euler’s method can be unstable and is not even efficient.

To understand the Euler’s method we must know the error that this method produces. For this reason we must understand the *Taylor series*:

$$x(t_0 + \Delta t) = x(t_0) + x'(t_0)\Delta t + \frac{\Delta t^2}{2} x''(t_0) + \frac{\Delta t^3}{6} x'''(t_0) + ... + \frac{\Delta t^n}{n!} \frac{\partial^n x}{\partial t^n} + ...$$

We get the Euler update formula by truncating the series, discarding all but the first and the second term on the right hand side. Euler’s method would be correct only if all derivatives beyond the first were zero, i.e. if $x(t)$ were linear. The *error term*, the difference between the Euler step and the full, untruncated Taylor series, is dominated by the leading term, $(\frac{\Delta t^2}{2} \ddot{x}(t_0))$. We can describe the error as $O(\Delta t^2)$ (read Order $\Delta t$ squared.)
We need to limit the problem with a little time step, in this way reduce the error but not delete. The error that linearly accrues rest with time step. In practice, a great many timesteps might be required, depending on the error and the function $f$.

The Euler method can be improved by using the midpoint method. In this case, the derivative is first evaluated at the current time, then the simulation is linearly extrapolated by half of a time step; then the derivative is evaluated again and the simulation is re-extrapolated from its original state using the new derivative. This is actually a second-order Runge-Kutta method.

A second variation on Euler integration is called the leapfrog method. Instead of updating the velocities and the locations of the simulated bodies at the same time, we first update the locations, then update the velocities using force calculated from the new value of the locations. This has about the same computational cost as the Euler method, but is much more stable.

### 3.2.2 Verlet’s method

Verlet’s method \[13\] is a method for calculating the trajectories of particles in molecular dynamics simulations.

The method was developed by French physicist Loup Verlet\(^1\). The Verlet’s

\(^1\)Loup Verlet [14] is a French physicist who pioneered the computer simulation of molecular dynamics models. In 1967 he developed what is now known as Verlet integration (a method for the numerical integration of equations of motion) and the Verlet list (a data structure that keeps track of each molecule’s immediate neighbors in order to speed computer calculations of molecule to molecule interactions).
3.2 Numerical Integration

Integration offers greater stability than the much simpler Euler’s integration. Stability of the technique depends a uniform update rate, or the ability to accurately identify positions at a small time delta into the past. In this method does not store explicit velocities. Instead, the positions of all objects at both time $t_n$ and $t_{n-1}$ are stored. This "velocityless" representation allows Verlet to be extremely stable in cases where there are large numbers of mutually interacting particles, such as in a piece of cloth or a ragdoll. In the following, we explain the algorithm to simply calculate trajectories using Euler integration, which is defined by:

- $x(t_0 + \Delta t) = x(t_0) + v(t_0)\Delta t$.
- $v(t_0 + \Delta t) = v(t_0) + a(t_0)\Delta t$.

$t_0$ is the current time and $\Delta t$ is the time step.

The Verlet algorithm reduces the level of errors introduced into the integration by calculating the position at the next time step from the positions at the previous and current time steps, without using the velocity:

$$x(t_0 + \Delta t) = x(t_0) + (x(t_0) - x(t_0 - \Delta t)) + \frac{\Delta t^2}{2} a(t_0) = 2x(t_0) - x(t_0 - \Delta t) + a\Delta t^2.$$  

The velocity at each time step is then not calculated until the next time step.

$$v(t_0) = \frac{x(t_0 + \Delta t) - x(t_0 - \Delta t)}{2\Delta t}$$

A related algorithm is the velocity Verlet algorithm:

$$x(t + \Delta t) = x(t) + v(t)\Delta t + \frac{1}{2} a(t)(\Delta t)^2$$

$$v(t + \Delta t) = v(t) + \frac{a(t) + a(t + \Delta t)}{2}\Delta t$$

The Verlet integrator derived from the Taylor series. Let $x(t)$ be the trajectory of a particle at time $t$.

The Taylor expansion around time $t_0$ then gives:

$$x(t_0 + \Delta t) = x(t_0) + \Delta t x'(t_0) + \frac{1}{2} \Delta t^2 x''(t_0) + \frac{1}{6} \Delta t^3 x'''(t_0) + O(\Delta t^4)$$

and

$$x(t_0 - \Delta t) = x(t_0) - \Delta t x'(t_0) + \frac{1}{2} \Delta t^2 x''(t_0) - \frac{1}{6} \Delta t^3 x'''(t_0) + O(\Delta t^4)$$

Adding these together gives:

$$x(t_0 + \Delta t) + x(t_0 - \Delta t) = 2x(t_0) + \Delta t^2 x''(t_0) + O(\Delta t^4)$$

and the final conventional format:

$$x(t_0 + \Delta t) = 2x(t_0) - x(t_0 - \Delta t) + \Delta t^2 x''(t_0) + O(\Delta t^4)$$
Mathematical tools

The term $O(\Delta t^4)$ represents fourth-order and higher terms in the Taylor expansion. This offers the clear advantage that the third-order term from the Taylor expansion cancels out, thus making the Verlet integrator an order more accurate than integration by simple Taylor expansion alone.

Within a last notation we have the Verlet integration:

$$x' = 2x - x^* + a \Delta t^2 + O(\Delta t^4)$$

where

- $x'$ = new state.
- $2x$ = current state.
- $x^*$ = past state.
- $x^* = x$
- $x = x'$

The term of velocity disappeared, in a single step figure out the new position in at next step. It is a reversible in time in fact if a negative time step is used, the system rolls back exactly to the start point; this means that the energy is conserved and thus the method is very stable.

In conclusion the Verlet integration rather than Euler integration is better for creating the constraints between particles, because is very easy to do. A constraint is a connection between multiple points that limits them in some way, perhaps setting them at a specific distance or keeping them apart, or making sure they are closer than a specific distance. Often physics systems use springs between the points in order to keep them in the locations they are supposed to be. However, using springs of infinite stiffness between two points usually gives the best results coupled with the Verlet algorithm.

### 3.3 Rotation in 3D space

In this section we explain how make rotations in three dimensions. We discuss three commonly methods for making rotations as Euler angles, matrix, and quaternion (read more in [15]).

When we used a solid three dimensions objects we need a way to specify, store and calculate the orientation\(^2\) and subsequent rotations of the object.

A rotation is a movement of an object in a circular motion.

A two-dimensional object rotates around a center (or point) of rotation.

A three-dimensional object rotates around a line called an axis.

The rotations can have been made with three different approaches:

\(^2\)The orientation of an object in space is the choice of positioning it with one point held in a fixed position. Usually, an orientation is defined by a rotation from the initial system.
Rotations can be implemented using Euler angles.

Rotations can be implemented using Matrices representation.

Rotations can be implemented using Quaternions representation.

### 3.4 Euler angles

The point of departure for introducing the Euler angles is explaining the considerable theorem by Euler:

**Euler’s Theorem** [16]: *Any two independent orthonormal coordinate frames can be related by a sequence of rotations (not more than three) about coordinate axis, where no two successive rotations may be about the same axis.*

This theorem warrants the mere existence of a rotation sequence, therefore we can find a sequence of rotations successive coordinate axis. The maximum number of rotations is three, and in practice two or even one rotation may suffice. The angle of rotation about a coordinate axis is called an Euler Angle. The Euler angles [17] were developed by *Leonhard Euler*³ describe the orientation of a rigid body in three-dimensional Euclidean space.

The idea of Euler angles is to split the complete rotation of a cartesian coordinate system into three simpler rotations about the axis of this system. When Euler angles are used, a general orientation is written as a set of rotations about three mutually orthogonal axis in space.

Usually the X, Y, and Z axis in a Cartesian coordinate system are used. To give an object a specific orientation it may be subjected to a sequence of three rotations described by the three Euler angles. This means that we can represent an orientation with three numbers.

A sequence of rotations around principle axis is called an *Euler Angle Sequence* [18]. Recall that a triple of Euler angle \([\theta_1, \theta_2, \theta_3]\) are interpreted as a rotation by \(\theta_1\) around an axis \(A_1\), then a rotation by \(\theta_2\) around an axis \(A_2\) and finally a rotation by \(\theta_3\) around \(A_3\), with \(A_2\) different with \(A_1\) and \(A_3\).

The axis are restricted to the coordinate axis, X, Y and Z, giving twelve possibilities as shown in Table 3.1. This gives us 12 redundant ways to store an orientation using Euler angles.

In conclusion, the full space of orientations can be parameterized by Euler angles[15], but in the following we showed the advantages and drawbacks of this approach.

**Advantage**  The traditional approach of Euler angles is often used in one DOF rotation joints, because there is a single axis of rotation and would be faster and simpler using this approach instead of quaternion. Moreover, it is simple to build the matrix from a set of Euler angles.

³Leonhard Euler [1707-1783] was a swiss mathematician who made enormous contributions to a wide range of mathematics and physics including analytic geometry, trigonometry, geometry, calculus and number theory.
**Mathematical tools**

<table>
<thead>
<tr>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable X</td>
</tr>
<tr>
<td>XYZ</td>
</tr>
<tr>
<td>XZY</td>
</tr>
<tr>
<td>XYX</td>
</tr>
<tr>
<td>XZX</td>
</tr>
<tr>
<td>Variable Y</td>
</tr>
<tr>
<td>YXZ</td>
</tr>
<tr>
<td>YZX</td>
</tr>
<tr>
<td>YXY</td>
</tr>
<tr>
<td>YZY</td>
</tr>
<tr>
<td>Variable Z</td>
</tr>
<tr>
<td>ZXY</td>
</tr>
<tr>
<td>ZYX</td>
</tr>
<tr>
<td>ZXZ</td>
</tr>
<tr>
<td>ZYZ</td>
</tr>
</tbody>
</table>

Table 3.1: All representation with three variables

**Drawback**  One potential problem that Euler angles can have is *gimbal lock*. The Gimbal lock [19] is caused by the alignment of two of the three gimbals together so that one of the rotation. This results when two axis effectively line up, resulting in a temporary loss of a degree of freedom.

This problem derivate from the rotation made around one axis can be cover up the rotation made around another axis. In the Figure 3.3 is shown this problem. For example if the pitch is 90° the yaw and roll can nullify each another.

Read more about Gimbal locks in [19]

![Figure 3.3: Gimbal lock](image)

The second problem is that there is no simple way to concatenate rotations.

### 3.5 Matrix

In three dimensions the rotation is determined by a arbitrary axis and one rotation angle $\theta$. Rotation matrices are the typical choice for implementing Euler angles. The rotation matrices are often used in computer graphics because they can represent both the position and orientation of an object in space as well as a number of other operation.

A rotation matrix is a $3 \times 3$ matrix, but usually homogeneous $4 \times 4$ matrices are

---

4A matrix is a rectangular array of elements which are operated on as a single object.
used instead. Read more in [20].
In general the matrices take the following format:

\[ M = \begin{bmatrix}
    a_{11} & a_{12} & a_{13} & 0 \\
    a_{21} & a_{22} & a_{23} & 0 \\
    a_{31} & a_{32} & a_{33} & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \]

where each set of rows and columns are an orthonormal base.
In the following we show the possible rotation on the three different axes. The first rotation around x-axis:

\[ R_x = R_x(\theta) = \begin{bmatrix}
    1 & 0 & 0 & 0 \\
    0 & \cos \theta & -\sin \theta & 0 \\
    0 & \sin \theta & \cos \theta & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \]

The second matrix rotation around y-axis:

\[ R_y = R_y(\theta) = \begin{bmatrix}
    \cos \theta & 0 & \sin \theta & 0 \\
    0 & 1 & 0 & 0 \\
    -\sin \theta & 0 & \cos \theta & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \]

The third matrix rotation around z-axis:

\[ R_z = R_z(\theta) = \begin{bmatrix}
    \cos \gamma & -\sin \gamma & 0 & 0 \\
    \sin \gamma & \cos \gamma & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \]

We show the conversions between different representations for rotation.

**Euler angles to matrix** Rotation about the x-axis by the angle \( \alpha \) followed by rotation about the y-axis by the angle \( \beta \) concluded by rotation about the z-axis by the angle \( \gamma \) is written in rotation matrix (homogeneous matrices the rotation matrices are \( 4 \times 4 \)):

\[ R_{\alpha,\beta,\gamma} = R_z(\gamma)R_y(\beta)R_x(\alpha) \]

\[
= \begin{bmatrix}
    \cos \gamma & -\sin \gamma & 0 & 0 \\
    \sin \gamma & \cos \gamma & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
    \cos \beta & 0 & \sin \beta & 0 \\
    0 & 1 & 0 & 0 \\
    -\sin \beta & 0 & \cos \beta & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
    1 & 0 & 0 & 0 \\
    0 & \cos \alpha & -\sin \alpha & 0 \\
    0 & \sin \alpha & \cos \alpha & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
= \begin{bmatrix}
    \cos \beta \cos \gamma & \cos \gamma \sin \alpha \sin \beta - \cos \alpha \sin \gamma & \cos \alpha \cos \gamma \sin \beta + \sin \alpha \sin \gamma & 0 \\
    \cos \beta \sin \gamma & \cos \alpha \cos \gamma + \sin \alpha \sin \gamma & -\cos \gamma \sin \alpha + \cos \alpha \sin \beta \sin \gamma & 0 \\
    -\sin \beta & \cos \beta \cos \alpha & \cos \alpha \cos \beta & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

In the following we show the advantages and drawbacks of this approach.
Mathematical tools

**Advantage**  The matrices are among the most commonly used technique, they are computationally efficient way to apply rotations to geometric data. This is advantage of matrix implementations because is that the mathematics is well-known, and that matrix applications are relatively easy to implement using standard packages.

But the real advantage of the matrix representation is the capacity of the homogeneous matrix to correspond to all the other basic transformations, for example shearing, scaling, projection, translation. An other advantage of matrices is the matrix multiplication to combine in one matrix, and also the operation like "move forward" have all information in the matrix.

**Drawback**  The matrix multiplication is not generally commutative, that is $AB$ is not equal to $BA$. This means that we attentive to order of rotations.

### 3.6 Quaternion

The quaternions were first described by the Irish mathematician Sir William Rowan Hamilton [21] in 1843 and applied to mechanics in three-dimensional space. Hamilton’s aim was to generalize complex numbers to three dimensions, numbers as $a + ib + jc$ where $a, b, c \in \mathbb{R}$ and $i^2 = j^2 = k^2 = 1$.

Quaternion are an extension of complex numbers that provide an alternative method for describing and manipulating rotations. A quaternion is a vector in four dimension space that can be used to define a three dimension rigid body orientation. A quaternion has 4 components:

$$ q = [q_0, q_1, q_2, q_3] $$

Quaternions are actually an extension to complex numbers. Of the 4 components, one is a real scalar number, and the other three form a vector in imaginary $ijk$ space.

$$ q = [q_0, iq_1, jq_2, kq_3] $$

In the following we give a set of equality:

$$ i^2 = j^2 = k^2 = ijk = -1 $$

$$ i = jk = -kj $$

$$ j = ki = -ik $$

$$ k = ij = -ji $$

Sometimes we can come across the written as the combination of a scalar value $s$ and a vector value $v$:

$$ q = <s, v> $$
Figure 3.4: Quaternion represent a rotation by an angle $\theta$

where $s = q_0$ and $v = [q_1, q_2, q_3]$

Often we will use only unit length quaternions: $|q| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1$

A quaternion can represent a rotation by an angle $\theta$ around a unit axis $v$: $q = [w, x, y, z] = [\cos(\theta/2), v(\sin(\theta/2))]$

where rotation axis is $v = [ax, ay, az]$ and $\theta$ is angle and the value of $w, x, y, z$ are:

- $w = \cos(\frac{\theta}{2})$
- $x = ax\sin(\frac{\theta}{2})$
- $y = ay\sin(\frac{\theta}{2})$
- $z = az\sin(\frac{\theta}{2})$

In Figure 3.4 there is a reference intuitive of concept of quaternion. A unit quaternion $q = (q_0, (q_1, q_2, q_3))$ is equivalent to following the matrix

$$
\begin{bmatrix}
1 - 2q_2^2 - 2q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 & 0 \\
2q_1q_2 + 2q_0q_3 & 1 - 2q_1^2 + 2q_3^2 & 2q_2q_3 - 2q_0q_1 & 0 \\
2q_0q_3 - 2q_0q_2 & 2q_2q_3 + 2q_3q_0 & 1 - 2q_1^2 - 2q_2^2 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
$$

This matrix should be compared with the one given for Euler angles; note the increased symmetry of the quaternion matrix, which reflects the isotropy of the representation. Read more in [15]

In the following we show the advantages and drawbacks of this approach. Read more in [16]

**Advantage** The quaternions are used when there is a need to support interpolation between arbitrary orientations without suffering from Gimbal lock and the
order dependent problems we find with Euler angles. The rotation with quater-
nions not influenced by the choice of coordinate system as Euler angles. The user
of an animation system does not need to worry about a certain convention of the
order of rotation about explicit axis. An other advantage is that the quaternions
are more compact (and faster) than matrices. Particularly good for rotation and
interpolation, Slerp

The method called SLERP (Spherical Linear intERPolation) interpolates be-
tween two unit quaternions along the shortest arc on the unit sphere. The SLERP
function is defined as:

\[ \text{Slerp}(q_1, q_2, u) = q_1 \frac{\sin((1-u)\phi)}{\sin\phi} + q_2 \frac{\sin u\phi}{\sin\phi} \]

where

- \( q_1 q_2 = \cos(\phi) \)
- \( u \in [0, 1] \)
- \( q_1 \) and \( q_2 \) are two quaternions (rotations)
- \( u \) is the fraction between these rotations along the four dimension sphere.

**Drawback** The quaternion are not necessary for one DOF rotation joints. The
mathematics back to quaternion isn’t intuitive and simple. Some operation are eas-
er to perform with matrices. The multiplication of quaternions is non-commutative.

### 3.7 Conclusion

In this chapter we discussed in detail some important mathematical concepts.
We use the integration methods in order to create the physics model, and we explain
in which way we can made rotations in the space. In order to articulate our ragdoll,
we used both Euler angles both quaternions.

In fact for creating the motion constraints, we generate two important regions
(Region Begin and Region End) with quaternion and inside them we figured out (by
means of Euler angles) the position of a particle. If the particle’s position is inside
the right region (defined between region_begin and region_end) then the position
will be good; otherwise, the computed position will result not good and then it will
be applied the constraint. More about that is explained in the next chapter 4.

In this work, we used the Euler angles to have the particle’s positions because it
is the simplest (and quickest) method to figure out that; moreover, we used the
quaternions for making the region constraints, because they yield lightest to make
vector rotations in three dimensions.

---

5In computer graphics, Slerp[22] is shorthand for spherical linear interpolation, introduced by
Ken Shoemake in the context of quaternion interpolation for the purpose of animating 3D rotation. It
refers to constant speed motion along a unit radius great circle arc, given the ends and an interpolation
parameter between 0 and 1.
Chapter 4

Character articulation

This chapter gives an overview of a convenient model to synthesize moving character. Mainly, we explain the tools used to create our constraints and in which manner we limit the rotations of joints. Moreover we highlighting the concept of Degree Of Freedom (DOF) of all joints. In conclusion that we will discuss how we developed our constraints.

4.1 Articulated figure

The purpose of this work is to generate ragdoll movement similar to human movement. In computer graphics and animation, articulated characters are built as hierarchical structures consisting of a set of segments connected by joints. The joint allows relative movement within the skeleton.

All joints can allow one, two or three DOFs, which define its possible range of motion. The DOFs depend on the type of joints in consideration. We use the following rules:

- Three DOF for the right shoulder.
- Three DOF for the left shoulder.
- Three DOF for the right hip.
- Three DOF for the left hip.
- Three DOF for the right leg.
- Three DOF for the left leg.
- Three DOF for the neck.
- One DOF for the right knee.
- One DOF for the left knee.
Character articulation

- One DOF for the right elbow.
- One DOF for the left elbow.

The constraint for shoulder joint has got three DOFs; which means that it can rotate along three perpendicular axis. The rotation can be decoupled into a spherical motion.

Another example could be constraint for elbow joint, it has one rotation DOF\(^1\) which means that it only can rotate along a single axis.

In the following we explain as we can use one parameter to specify the angles of rotation for each DOF.

The aim of this work to control the motion of character controlling only the movement joint of abstract skeleton.

4.2 Model of skeleton

The skeleton [2] is represented from a set of bones linked by means of joints. A joint defines the possible range of motion.

As above mentioned the joints could move around one, two, or three axis. These axis are called degrees of freedom. If we make an approximation much exhaustive of our skeleton we can have two hundred degrees of freedom. We limit the angle of rotation of all joints in each directions.

The skeleton’s topology is an open directed graph or tree. One joint is selected as the root and the other joints are connected up in hierarchical fashion. A node in the tree represents the joints of the skeleton.

4.3 Ragdoll’s constraints

As mentioned before, the purpose of this work is create realistic movement of our ragdoll. We discuss in which manner our ragdoll is composed and how the rotations of joints have been checked.

We highlight the problem for limiting the movement of the joint and the methods used for creating joint constraints. In the following we explain the principles of implementation.

4.3.1 Principles of implementation

Our ragdoll is composed two different constraints:

- Equidistance Constraints.
- Joint Constraints.

\(^1\)In computer graphic the character animation with one DOF rotation joint, is sometimes called a hinge joint, so constraint for elbow joint and the constraint for knee joint are good examples of hinge joints. The hinge joint can be specified to rotate about any axis.
The **equidistance constraints** are constraints that represent the bound of two particles. In particular we call these constraints **stiff constraint**. The latter permits to insert the constraint between two particles; knowing in fact the position of two particles in the space, we figure out the distance that exit between particles. The distance represent the length of bound between particles. These constraints built in according to [3]

**The joint constraints**, instead represent the movement constraints. We interested to limit the possible movement of single joint. In fact at the beginning we have our ragdoll that can move in totally liberty, our ragdoll at the beginning did all wrong movement because there aren’t the joint constraints in three dimensions.

We have drawn on article [23] of Qiang Liu and Edmond C.Prakash.

In this article in fact there is a good description for rotation limits for joint parameterized with the unit quaternion. The method that they present imposes limits on the amount of rotation for each joint. We follow the idea of this method with our considerations. In the following we explain in which manner we resolve the constraint that breach with another.

### 4.3.2 Resolve the problem that one constraint breaches another

As above mentioned we created the equidistance constraints in according to [3] and we created the joint constraints in according to [23] (with a our considerations).

The methods to insert these constraints is very simple and linear because, in fact, the first constraint we must only figure out the distance between two particles then insert the stick inside the particles, instead for the second constraint we consider the possible degree of freedom of the joint and we limit the movement of it. But during the simulation, after one integration step, it possible for example the separation distance between the particles might have became invalid.

In fact we might have two invalid cases:

- the particles are too close each other.
- the particles are too far each other.

To obtain the right distance, in order to satisfy the constraints, we use matter of solving a system of equations. However, we choose to proceed indirectly by local iteration. We simply repeat the satisfy equidistance constraints a number of times after each other in the hope that the result is useful.

At the beginning this approach (pure repetition) might appear somewhat unsophisticated, but it demonstrates that it actually converges to the solution that we are looking for. This approach is often called in literature relaxation (or Jacobi or Gauss-Seidel iteration).

If the initial conditions are right and the consecutive satisfying various local constraints and then repeating carry out a global configuration that satisfies all constraints at the same time.
Character articulation

The number of necessary iterations varies depending on the physical system simulated and the amount of motion. In our case the number of iterations are ten. If we stop the iterations early, the result might not end up being quite valid but because of the Verlet scheme, in next frame it will probably be better, next frame even more so etc. This means that stopping early will not ruin everything although the resulting animation might appear somewhat sloppier.

4.3.3 Sequence joints

We implemented a simple and efficient method that permits limit the movement of character. The sequence of joint is in the following:

- Constraint for Shoulder joint.
- Constraint for Hip joint.
- Constraint for Leg joint.
- Constraint for Neck joint.
- Constraint for Knee joint.
- Constraint for Elbow joint.

Since the constraint for Shoulder joint, constraint for Hip joint, constraint for Leg joint and constraint for Neck joint have three degree of freedom we illustrate only one of this. In the following we present an example of this case, we show the constraint for Shoulder joint. Moreover we discuss also the constraint for elbow constraint that has only one degree of freedom.

4.3.4 Constraint for Shoulder joint

In the following, it will be described the constraint for shoulder joint both right shoulder and left shoulder.

The shoulder’s joint constraint is often called ball-socket because the latter has three DOFs.

Now, we explain how this joint constraint has been implemented underlying some important fragment of the implemented code.

In order to implement the shoulder constraint, we thought that to figure out such constraint it’s important to know the elbow’s position. Such latter position is useful to know the angle between the shoulder and each degree of freedom.

Then, each time that the shoulder constraint is computed, in the beginning we obtain the spacial coordinate of three particles: shoulder, elbow, hand.

The code which is used to take the spacial coordinate is the following:

```cpp
RefIterator pIt = ref_particles.begin();
ParticleType &p0 = **pIt;
```
ParticleType &p1 = **(pIt + 1);
ParticleType &p2 = **(pIt + 2);
CoordType space_shoulder = p0.P();
CoordType space_elbow = p1.P();
CoordType space_hand = p2.P();

In order to have our coordinate, we translate the spacial coordinate into the origin coordinate. We have supposed that the shoulder is placed in the origin of axis:

CoordType temp_elbow = space_elbow - space_shoulder;
CoordType temp_hand = space_hand - space_shoulder;

Figure 4.1 shows the particles in the space and the particles in the origin coordinate. At this moment we know if the created structure belong to right shoulder or left shoulder.

Now we call three methods which define the allowed movement (region in the space) for the elbow compared to the shoulder. We split the problem on three different planes XY, YZ, XZ. In each plane has been considered the allowed rotations.

The method shoulder XY

The first method is called shoulderXY. Such method creates two points, which represent elbow and hand, and then creates the plane XY. After that it figures out the normal belonging to the defined plane XY.

The normal is important because is been used:

- For computing the projection of particles on the plane XY.
- Become the direction for the quaternions, which are created to define the borders of the allowed area.
Character articulation

After figure out the normal we make the projections of the former two points over the plane XY. On the former plane, there are defined two important points which define two regions, named:

- Region Begin.
- Region End.

These regions define the allowed area where the particles, elbow and hand, can move. The region_begin and region_end represent the bounds where the particles are possibility to move.

Both the region_begin and the region_end are created using the concept of quaternion. In this case the region_begin coincide with the point (0.0, -1.0, 0.0) while the region_end is created using the concept of quaternion. In fact we create a quaternion with angle equal to a angle_region_end = 120.0 degree and vector equal the normal of the plane XY.

In the following, it showed the code:

```cpp
Point3<double> region_begin ( 0.0, -1.0, 0.0 );
Quaternion<double> constraint_end(math::ToRad(120.0) ,norm);
constraint_end.Normalize();
Point3<double> region_end = constraint_end.Rotate(region_begin);
```

Now we realize a test where we ascertain if our particle elbow is inside the region_begin and region_end.

If the result is true we can move the particle else we return in the allowed position namely the last allowed position.

The Figure 4.2 shows the initial position of particles. We define with red line the region begin and with blue line the region end, the particles can stay only in the zone. The zone represent the controlled rotation of the constraint’s shoulder.

![Figure 4.2: Permitted zone on the plane XY](image)

The Figure 4.3 shows the possible motion of particles elbow and hand inside the allowed area, we applied a force that moves the particle hand inside the region, when we try to move the particle outside the allowed area it is not possible because there is the bound, and the particles hand and elbow stay in the last valid position.
4.3 Ragdoll’s constraints

Figure 4.3: The possible motion of arm on the plane xy

**The method shoulder YZ**

The second method that we call is **shoulderYZ**. The steps are similar to the precedent method but change the plane in consideration and the region_begin and region_end. In fact, in this case we have:

```cpp
Quaternion<double> constraint_begin(math::ToRad(50.0), norm);
constraint_begin.Normalize();
Quaternion<double> constraint_end(math::ToRad(225.0), norm);
constraint_end.Normalize();
Point3<double> defRegion(0.0, 0.0, -1.0);
Point3<double> region_begin = constraint_begin.Rotate(defRegion);
Point3<double> region_end = constraint_end.Rotate(defRegion);
```

Also here we do test for ascertaining if our particle are inside in the permitted zone. The Figure 4.4 shows the initial position of particles. Also here we define with red line the region begin and with blue line the region end, the particles can stay only in the zone. The zone represent the controlled rotation of the constraint’s shoulder. The Figure 4.5 shows the possible motion of particles elbow and hand inside the allowed area.

Also here we apply a force that moves the particles elbow and hand. The limit of the motion depends of the region begin and the region end.
The method shoulder XZ

The third method that we call is shoulderXZ. The plane is XZ and the region_begin and region_end are in the following:

```cpp
Quaternion<double> constraint_begin(math::ToRad(80.0),norm);
constraint_begin.Normalize();
Quaternion<double> constraint_end(math::ToRad(225.0),norm);
constraint_end.Normalize();
Point3<double> defRegion(1.0, 0.0, 0.0);
Point3<double> region_begin = constraint_begin.Rotate(defRegion);
Point3<double> region_end = constraint_end.Rotate(defRegion);
```
4.3 Ragdoll’s constraints

Now we realize a test where we ascertain if our particle elbow is inside the region_begin and region_end.

The Figure 4.4 shows the initial position of particles. Here we define with red line the region begin and with blue line the region end, the movement of particles is limited from the two regions. The Figure 4.7 shows the possible motion of particles elbow and hand inside the allowed area.

![Figure 4.6: Permitted zone on the plane XZ](image)

![Figure 4.7: The possible motion of arm on the plane xz](image)

4.3.5 Constraint for Elbow joint

The joint constraint for right and left elbow has only one DOFs. In the following we explain the method used.
Character articulation

The joint constraint for the elbow it has been create a structure with two particles:

- Elbow.
- Hand.

We have now the spatial coordinate of our particles:

```cpp
RefIterator pIt = ref_particles.begin();
ParticleType &p0 = **pIt;
ParticleType &p1 = **(pIt + 1);
CoordType space_elbow = p0.P();
CoordType space_hand = p1.P();
```

We obtain the origin coordinate:

```cpp
CoordType temp_hand = space_hand - space_elbow;
```

The Figure 4.3.5 shows the particle in the space and the particle in the origin coordinate. Now we have two structures belong to right elbow or left elbow. Now we call only one method where define the possible movement of hand. The method that we call is `elbowYZ`, this create two point that represent elbow and hand, and then create the plane YZ.

In this case we calculate the normal of plane and we make the projection of two point on the plane.

In this moment we define `region_begin` and `region_end`.

Both the `region_begin` and the `region_end` is created using the concept of quaternion.

In fact we create two quaternion, the first for `region_begin` is created with angle equal to a angle `angle_region_begin = 10.0` degree and vector equal the normal of the plane YZ, while the quaternion for `region_end` is created with angle equal to a angle `angle_region_end = 130.0` degree and vector equal the normal of the plane YZ. The Figure 4.8 shows the initial position of particles. We define with red line the region
4.4 Conclusion

Figure 4.8: Permitted zone on the plane YZ

begin and with blue line the region end, the particles can stay only in the zone. The zone represent the controlled rotation of the constraint’s elbow. In the following we present the code:

```cpp
Quaternion<double> constraint_begin(math::ToRad(10.0), norm);
constraint_begin.Normalize();
Quaternion<double> constraint_end(math::ToRad(130.0), norm);
constraint_end.Normalize();
Point3<double> defRegion(0.0, -1.0, 0.0);
Point3<double> region_begin = constraint_begin.Rotate(defRegion);
Point3<double> region_end = constraint_end.Rotate(defRegion);
```

We realize a test where we ascertain if our particle hand is inside the region_begin and region_end. If the result is true we can move the particle else we return in the allowed position namely the last allowed position.

4.4 Conclusion

This chapter we have presented a complete overview of constraints. We explain the initial problem for limiting the movement joint. In conclusion we discuss in detail, showing also the code as we have make the joint constraint, we explain the constraint for shoulder joint and the constraint for elbow joint to comprehend better all allowed region, where the particles can move in total liberty.
Character articulation
Chapter 5

Coat the ragdoll

In this chapter we will explain the second part of this work: the ragdoll’s shape. We will make a brief introduction about some methods which have been used in the history of computer graphics.

Moreover, we will present two approaches for coating our ragdoll which both use a set of geometric primitives.

The first approach is very complicated, and it has no great visual result. Differently, the second approach is much more simple with good visual result.

5.1 The ragdoll’s shape

Our goal is to coat a character in three dimension. Our character is composed by articulated joints and bones. The location of each particle coincides with a joint, and the distance between two particle defines the length of the bone.

There are many approaches to make a Ragdoll’s shape like geometric primitives, coat the ragdoll with a mesh 1, or another used method is the skinning 2.

In the following, we will present the commonly methods for coating a ragdoll, and then we will illustrate two methods that we implemented. We will show the advantages and the drawbacks of these two different approaches, which have given unlike result.

5.2 Rigid body parts

The method, which we will illustrate, was a popular method in the game industry in the mid 90’s and one of the first used method in the world of computer games

---

1 The mesh is a series of polygons (triangles) grouped to form a surface. An other definition of mesh is a digital representation of a surface or solid consisting of multiple, possibly curved, line segments whose intersections form a regular grid.

2 The skinning is a the process where a model is wrapped around a skeleton, when the skeleton moves, the model will move correspondingly. The model effectively forms a skin over the skeleton joints.
Coat the ragdoll

industry. This method is conceptually simple and very used in literature. This
method used the simple geometric primitives for coating a 3D character.
Many video games used this method in the past, for instance, we want to men-
tion the Weekend Warrior\(^3\) and the more famous MechWarrior [25] and [26]. In
each game have been used the segmented body parts approach. The first Weekend
Warrior is a 1997 computer game (now freeware) for the Macintosh developed by
Pangea Software and published by Bungie Studios. The author of this game is
Brian Greenstone.
Figure 5.1 shows the realization of this game. This game was the first one which
used the former method, and all of the games after that, went with the single object
skinned approach, which has got much better results.

![Weekend Warrior](image)

Figure 5.1: The game version of Weekend Warrior (adapted[26])

The second MechWarrior is the title of a number of games set in the fictional uni-
verse created for the tabletop wargame BattleTech. The term "MechWarrior" is
also used to describe the pilot or operator of a BattleMech. The use of the geo-
metric primitives is very simple, but unfortunately, it presents many problems. For
example, when we coat the body’s part (arm or leg), that results difficult shape
them. Therefore, it has been replaced by skinning.

5.3 Skinning

Skinning [27](deformable mesh) is a popular method for performing real time de-
formations of polygon meshes by way of associated bones and joints of an articu-
lated skeleton.

Skinning is the process of binding a skeleton to a single mesh object, and skin-
ing deformation is the process of deforming the mesh as the skeleton is animated

\(^3\)The Weekend Warrior is a video game made of Pangea Software. The Pangea Software [24] is
an Macintosh game company that is owned and operated by Brian Greenstone. Formed in 1987, the
company began by writing a number of shareware games for the Apple II GS computer, with their
first commercial game, Xenocide, being released in 1989. Their first published Macintosh game
came in 1993.
other methods

or moved. As the skeleton of bones is moved, a matrix association with the vertices of the mesh causes them to deform in a weighted manner. Skinning is a popular method for doing deformations of characters and objects in many 3D games. The skinning is the process of attaching a renderable skin to an underlying articulated skeleton.

There are several approaches to skinning with varying degrees of realism and complexity. Our main focus will be on the smooth skinning algorithm, which is both fast and reasonably effective, and has been used extensively in real time and prerendered animation. The smooth skinning algorithm goes by many other names in the literature, such as blended skinning, multi-matrix skinning, linear blend skinning, skeletal subspace deformation (SSD), and sometimes just skinning.

But although the skinning is still the most popular method for the animation of deformable human and creature characters, it suffers from a number of problems, such as the collapsing elbow and candywrapper effect. In the Figure 5.2, we show an example of this technique and also the problem.

![Figure 5.2: Skinning problem (adapted [27])](image)

In our project we use a simplified skin from point. More about that is explained in next section 5.6

### 5.4 Other methods

There are some methods able to coat a Ragdoll. In [28], for example, is cited an approach which use the finite element\(^4\). Such article presents a framework for the

---

\(^4\)The finite element analysis was first developed in 1943 by Richard Courant, mathematically, the finite element method (FEM) is used for finding approximate solution of partial differential equations (PDE) as well as of integral equations such as the heat transport equation. This method is a procedure that permits to round the values of one function in determined point. Determining the value of function in a set of point (this depends of precision of our mesh) we determine the behavior of variable into domain by means of linear interpolation (cubical interpolation for problem of major order)
Coat the ragdoll

skeleton-driven animation of elastically deformable characters. In fact, a human or animal character is embedded in a big volumetric control lattice, which provides the structure needed to apply the finite element method. In this article there is a concept of skeleton in which all bones are made to coincide with edges of the control lattice, which permits us to apply the constraints using algebraic methods.

There is an association between a regions of the volumetric mesh with particular bones and perform locally linearized simulations, which are blended at each time step. In this article is introduced a good method for interactive simulation of deformable bodies controlled by an underlying skeleton. The Figure 5.3 shows a special situation.

![Figure 5.3](image)

Figure 5.3: The skeleton of animal and the model embedded in half of control lattice and the skeleton coincides with edges and vertices of control lattice(from [28])

5.5 Implementation with rigid body parts

In this approach, we used simple geometric primitives, such as cubes. The point of departure has been becoming familiar with OpenGL (more detail in Appendix B.1) and then understanding the geometric primitives in order to coat our ragdoll.

Now we explain the method that uses the rigid body part and this method hasn’t got any result although this approach is very complex. In the approach, we wanted to use geometric primitives, in order to cover the arm, the leg, the neck and trunk. Unfortunately, we met many problems. Figure 5.4 shows an example of two particles.

For simplicity, we consider the particles shoulder and elbow. We know only the relative position in the space of these, but if we subtract from the coordinate of elbow particle and the coordinate of shoulder particle we obtain the relative origin coordinate.
5.5 Implementation with rigid body parts

Figure 5.4: Insert the bar line between the particles shoulder and elbow

The first step has been to define the distance between the shoulder particle and the elbow particle. In the space, the distance between two points expressed in three dimension is figured out by the formula:

$$\text{distance (p1 and p2)} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Such distance between particles represents the length of our bar line that we will use for coating the distance between the particles.

Below, there are shown the steps used to draw the final coating bar.

- At the beginning bar line between two particle lie on axis X.
- Figure out the angle rotation on axis Z.
- Figure out the angle rotation on axis Y.
- Figure out the angle rotation on axis X.

The first thing that we made was position the bar line on axis X only to simplify the issue.

The big problem that we have, it was to understand in which way to insert the cube between particle starting from initial position. The problem was to understand, which rotation and which angle of rotation, the bar line must do from initial position to the final position. On Figure 5.5 is shown an example of such problem.

We afforded the general method that can be used for all particles.

The second important consideration has been decide how to figure out the first angle of rotation around the Z axis. The order of rotation is totally indifferent but for simplicity we consider the first rotation on axis Z. In the following we showed the step fundamental that we followed to make the all rotation of our bar line in the space.
Coat the ragdoll

Figure 5.5: The initial position of bar line between the particle shoulder and particle elbow that lie on axis X, the final position of bar line in the space.

**Rotation around Z axis** We started with the bar line which lies on X axis. The problem is rotate the bar line according to the final position. The step that we following are:

- Figure out the angle of rotation on Z axis.
- Use the quaternion to rotate the bar line.

In fact the first step has been to figure out the angle of rotation. We started to consider the matrix of rotation on Z axis:

\[
egin{bmatrix}
 x' \\
 y' \\
 z' \\
 1
\end{bmatrix} = \begin{bmatrix}
 \cos \gamma & -\sin \gamma & 0 & 0 \\
 \sin \gamma & \cos \gamma & 0 & 0 \\
 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
 x \\
 y \\
 z \\
 1
\end{bmatrix}
\]

where \( \cos \gamma = \frac{xx' + yy'}{x^2 + y^2} \) and \( \sin \gamma = \frac{xy' - yx'}{x^2 + y^2} \).

We figure out the \( \tan \gamma = \frac{\sin \gamma}{\cos \gamma} \) and then the \( \arctan \gamma \).

The goal of this step is rotate the initial bar line as much as the final position. Although in this case we know the initial position of our bar line (it lies on X axis), and then we must consider the final position. For this reason, we want know the position where the initial and the final bar could be.

Figure 5.6 shows the possible position where the bar line could assume. We subdivided the circumference in four quadrants.

Firstly, we figure out the angle between the initial bar line and the final bar line. After known the angle, we create quaternion which will be used in order to make a

\[\text{For us the final position is the position on the origin that we obtain subtracting the spacial coordinate belonging to the elbow and the shoulder particles}\]
5.5 Implementation with rigid body parts

rotation around Z axis. The created quaternion will bring the initial bar position in a new “intermediate” position. The latter position is the new position assumed by the bar; such position will be modified by the rotation around the Y and after the X axis in order to have the final bar position.

![Figure 5.6: Definition of quadrant for rotation on axis Z](image)

**Rotation around Y axis**  Here, we have the position figured out on the previous step, which was the rotation around the Z axis of the bar line. In this new step, we have figure out the new position after to consider the rotation around the Y axis. The consideration made in this step are quite similar to the previous one, obviously changing the rotation axis. Firstly, in fact, we figure out the angle between the update bar line and the final bar line.

\[
\begin{bmatrix}
  x' \\
  y' \\
  z'
\end{bmatrix} =
\begin{bmatrix}
  \cos \beta & 0 & \sin \beta & 0 \\
  0 & 1 & 0 & 0 \\
  -\sin \beta & 0 & \cos \beta & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]

where \( \cos \beta = \frac{x'x + z'z}{x'^2 + z'^2} \) and \( \sin \beta = \frac{z'y' - x'y}{x'^2 + z'^2} \).

We figure out the \( \tan \beta = \frac{\sin \beta}{\cos \beta} \) and then the \( \arctan \beta \).

Like previously, we consider the respective position of the update bar line and the final bar line and we consider the quadrant where is the bar line. Figure 5.7 shows the quadrant’s definition.

Now we create a quaternion that rotate around axis Y the update bar line with angle figure out as above mentioned. In the following we complete the approach with last rotation around the axis X.

**Rotation around X axis**  Now we figure out the last rotation angle, which will be used to give the final position to the bar line. For this reason we figure out the
Figure 5.7: Definition of quadrant for rotation on axis Y

\[
\begin{bmatrix}
  x' \\
y' \\
z'
\end{bmatrix}
=
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & \cos \alpha & -\sin \alpha & 0 \\
  0 & \sin \alpha & \cos \alpha & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

where \( \cos \alpha = \frac{yy' + zz'}{y^2 + z^2} \) and \( \sin \alpha = \frac{yz' - y'z}{y^2 + z^2} \).

We figure out the \( \tan \alpha = \frac{\sin \alpha}{\cos \alpha} \) and then the \( \arctan \alpha \). Here, we want to know where are the new position of bar line and the final bar line, also here we consider the possible region inside a quadrants where the bar line can stay. Figure 5.8 shows the definition of quadrant.

Figure 5.8: Definition of quadrant for rotation on axis X

Now we create a quaternion that rotate around axis X of angle figure out.
After we considered the relative angle rotation around axis Z, axis Y, axis X, the last step that we made, has been to transport the bar line in relative position in the space between two particle. This approach has been used for all case (arm, leg,
5.6 Simplified skin from points method and implementation

The Figure 5.9 shows the front lateral and the lateral view.

![Image of Ragdoll with front and lateral views](image)

Figure 5.9: Front view of Ragdoll and lateral view of Ragdoll

### 5.6 Simplified skin from points method and implementation

This method is a simplified skin. This approach has been introduced because we want a simple method for coating the part of the trunk and abdomen and also because the precedent approach presented some problems, for instance, with the final part between knee and foot, and also between elbow and hand.

This method considers the skin from knowledge of a some point of the shape. We discuss the method and we explain his advantages.

This method is very simple for covering part body. In the following we explained which are the part covered by this second approach:

- Trunk.
- Abdomen.
- Right and Left Arms.
- Right and Left Legs.
Coat the ragdoll

- Neck.

In this approach, we coat differently and separately each single part, not all part together.

**Trunk**  We presented the method to coat the central part of our Ragdoll. If we look with attention the Figure ?? in chapter 2, we notice that the position of five particles (neck, right shoulder, left shoulder, right high hip and left high hip) form a pentagon.

The step which we followed to coat the trunk are:

- Consider five points.
- Figure out the center of pentagon.
- Figure out the five points between corners and center.
- Insert a set of triangles between all founded points.
- Repeat these property also on the back of the trunk.

Figure 5.10 shows the fundamental step that we followed. Obviously this step are pertaining only front plane, we considered also the back plane.

Totally, we have seventeen vertexes (five front, five back, five center plane, one front center and one back center) and thirty triangles (fifteen front triangles and fifteen back triangles). This method is very simple because there are neither rotation nor transformation.

**Abdomen**  The steps to coat the abdomen are similar compared to the steps to coat the truck.

In fact, Figure ?? showed in chapter 2, we look with attention our ragdoll, it could be notice that the position of four particles (right high hip, left high hip, right leg and the left leg) form a quatrilateral.

The step that we followed to coat the abdomen are:

---

6This point represent the position of five particles
5.7 Conclusion

- Consider four points\(^7\).
- Figure out four points on the back.
- Insert six quadrilateral.

**Arm, leg and neck**  In this case, we consider how to coat the arms but the steps for coating the legs and the neck are similar. Firstly, we consider the position of the shoulder and elbow particles, then we figure out other two points with the same coordinate on Y and Z axis but with different coordinate in the X axis (the new position of such new point depend to the length that we want). The last step is to insert six quadrilateral.

**The advantages**  The advantage of such method compared to the first approach is very simple and very intuitive. This approach warrants a good result without trouble us to make a rotation or transformation. In fact we must figure out only new vertex for creating a triangles or quadrilater.

But there is a little drawback, this method is not general in fact for insert the triangle and quadrilateral we must individualize each part that we cover computing the vertexes, then for this reason this method is not general.

The Figure 5.11 shows the result of the second approach we show the front view and the lateral view.

**5.7 Conclusion**

In this chapter, we explained two different approaches which have been implemented to cover all part of our ragdoll.

The first approach is very complex because we have to figure out all possible angle of rotations, and after that we must make transformations in order to move the bar line from origin coordinate to spatial coordinate.

Differently, the second approach is very simple providing good results using only a set of triangles and a set of quadrilaterals. The improvement of this work will be use the complete method of skinning (we have implemented a simplified skin from point).

In the next chapter, we will discuss about the collisions of our ragdoll with other objects. For this reason we have inserted the ragdoll in a room in which we will insert a set of big spheres.

Figure 5.12 shows the ragdoll covered with second approach inside one room.

\(^7\)This point represent the position of four particles
Coat the ragdoll

Figure 5.11: Front view of Ragdoll and lateral view of Ragdoll

Figure 5.12: Ragdoll inside a room
Chapter 6

Collision handling

In this chapter we explain the third part of this work, the collision handling, to properly handle collision between our ragdoll and other objects. The case that we have chosen in consideration has been the collision of our ragdoll with a plane and with a big sphere. We discuss the problem that we met when the our ragdoll collides with other object in the space in according to[10].

6.1 Collision detection

In physics, collision means the action of bodies that strike together. The collision detection has been a fundamental problem in computer animation. This is a big problem physically-based modeling, geometric modeling, and robotics. The problem is understand given two or most moving objects, which will be the final configuration of this object. The response to collisions is the actual physics problem of determining the unknown forces of the collision between objects. We focus the attention on two problems, the first is the penetration with the plane and the second is test the behavior our particles that collide with solid objects for example the sphere. A particle can be considered as a line segment from it’s previous position to it’s current position, for this reason if we are colliding against static objects, then we just need to test if the line segment intersects the object (it is a one possible idea).

In physics simulation, there is usually a distinction the collision in two part the impacts and the contacts[29].

Impacts are instantaneous collisions between objects where an impulse must be generated to prevent the velocities at the impact location from allowing the objects to interpenetrate.

Contacts are persistent and exist over some range of time. In a contact situation, the closing velocities at the contact location should already be zero, so forces are needed to keep the objects from accelerating into each other. With rigid bodies, contacts can include fairly complex situations like stacking, rolling, and sliding. When two solid objects collide (such as a particle strike a solid surface), forces are
Collision handling

generated at the impact position that impede the objects from interpenetrating. In the following we explain the big problem of not penetration.

6.2 Problem of nonpenetration constraints

As above mentioned we make a reference[10] for explain the concept of nonpenetration.

We don’t want to allow any inter-penetration at all particles when the latter strike the floor. This is very important because at the instant that the particles actually comes into contact with the floor, what we would like is to suddenly change the velocity of the particles.

When two objects are in contact at some point $p$, and they have a velocity towards each other, we call this colliding contact. Colliding contact requires an instantaneous change in velocity.

Whenever a collision happens, the state of an object, which describes both position, and velocity, stands a discontinuity in the velocity. The numerical routines that solve ODE’s$^1$ do so under the assumption that the state $X(t)$ always varies smoothly. The problem is simplify in this manner:

- If a collision happens at time $t_c$, we tell the ODE solver to stop.
- We take the state at this time, $X(t_c)$.
- Figure out how the velocities of objects involved in the collision must change.
- We call the state reflecting these new velocities $X(t_c)^+$
- We then restart the numerical solver, with the new state $X(t_c)$, and instruct it to simulate forward from time $t_c$.

When the objects are at rest$^2$ we say that the objects are in resting contact.

Now we consider two important problem that we meet we have the collision between object:

- Figure out the velocity that changes for colliding contact.
- Figure out the contact forces that prevent inter-penetration.

We consider our simulation, we figure out precise position of our particle, the which strike on the floor at specific time. The Figure 6.1 shows previously told. At time $t_0 + \Delta t$, the particle is found to lie below the floor. The actual time of collision $t_c$ lies between the time of the last known legal position, $t_0$ and $t_0 + \Delta t$.

We make a set of consideration:

$^1$Ordinary Differential Equation

$^2$The object are resting at one point $p$ e.g imagine the particle in contact with the floor with zero velocity
6.3 Handling collision and penetration by projection

Consider the particle at times $t_0$, $t_0 + \Delta t$, $t_0 + 2\Delta t$.

Consider the time of collision, $t_c$, time between $t_0$ and $t_0 + \Delta t$.

After the time $t_0$ the particle lies above the floor.

At the next time step $t_0 + \Delta t$ the particle is under the floor.

The last step means that inter-penetration has happened. If we are going to stop and restart the simulator at time $t_c$, we will need to compute $t_c$.

A simple manner of figure out $t_c$ is to use a numerical method called bisection[30]. If at time $t_0 + \Delta t$ we found inter-penetration, we inform the ODE solver that we wish to restart back at time $t_0$, and simulate forward to time $t_0 + \frac{\Delta t}{2}$. Now we have two possible cases:

- If there isn’t the inter-penetration we know the collision time $t_c$, which is lies between $t_0 + \frac{\Delta t}{2}$ and $t_0 + \Delta t$

- If there is inter-penetration e try to simulate from $t_0 + \Delta t$ to $t_0 + \frac{\Delta t}{4}$

The accuracy with which $t_c$ is found depends on the collision detection routines. The Figure 6.2 shows the zone of collision detection routines, which has a parameter $\varepsilon$. In this moment we decide if our computation of $t_c$ is good when the particle inter-penetrates the floor by no more than $\varepsilon$, and is less than $\varepsilon$ above the floor.

The method of bisection is a little slow, but its easy to implement and quite robust.

6.3 Handling collision and penetration by projection

In previous section we explain the method propose by Baraff [10], now we explain our solution of collision in reference to [3].
In this article there is a complete description of a new strategy. When the particle is a non valid position (under floor), the particles are simply projected out of the obstacle. By projection, we mean moving the point as little as possible until it is free of the obstacle. Normally, this means moving the point perpendicularly out towards the collision surface. Figure 6.3 shows a particle that inter-penetration the plane and by means of projection the particle puts on a valid position. This method is very simple in fact when the particle is in a not valid position, the position of the particle is projected in the nearest valid position. But can be a some problem. The principal problem that this method suffers is for example to project the particle in a bad direction. The Figure 6.4 shows this possible problem, and other problem is risk of adding energy. This method however is very simple and very quick to implemented.

### 6.4 Conclusion

In this chapter we explain the important environment constraints. We discuss the problem that we met when the system of particles collided with the floor and the
other object and we exposed our method for collision. In the next chapter we outline our project and our implementation.
Chapter 7

Project and Implementation

In the previous chapters, we described the theoretical construct of our study; in this chapter, instead, we will explain some details about our implementation. We outline the used tools, such as library and development environment.

7.1 Used tools

This application has been implemented on a personal computer Intel Pentium M processor 2.00 GHz and 1.0 GB of RAM using as operating system Windows XP. The implementation was made with Visual C++.NET 2005 using Microsoft Visual C++ 6.0 [31]. The libraries installed for our system are:

- Idolib "Interactive Deformabile Object Library".
- VcgLib "Visual Computer Graphics Library".
- Sdl "Simple DirectMedia Layer".
- Glut "OpenGL Utility Toolkit".
- Glew "OpenGL Extension Wrangler".

In this work has been necessary to use two important engines: Idolib (available [32]) for the simulation of the physics laws, and VcgLib (available [33]) used to make a count mathematical. The latter library is used to make a handling of vector, matrix and more other.

The library Sdl (available [34]) is a library similar to Glut but it has more functionality. Finally, we used also the library Glut (available [35]) and Glew (available [36]).
7.2 Code Structure

This work is inserted in a previous project. The name of the previous project is \textit{idolib\projs\m4\ParticlesBox}. The structure of final project is explained in the following. Figure 7.1 shows the sequence diagram of first step when we open the \texttt{m4\ParticlesBoxMain.cpp}.

![Sequence Diagram of the main part](image)

The first method that we met is: \texttt{dobj.Init()}. This method loads from external file the physical parameters. Such loaded parameters are fifteen and represent the possible integration method, the number of time step, the number of particle, different force, etc. Moreover, the method reseted the structures (this function to destroy the mesh) and initialize the simulation. This last method is very important because we create a pointer to the integrator:

\begin{verbatim}
ParticleMeshIntegrator *pde_p;
\end{verbatim}

now if the value of pde\_p is true we delete pde\_p and

\begin{verbatim}
pde_p = new ParticleMeshIntegrator(relations, vert, constraints);
pde_p->Reset();
pde_p->SetPDESolver(integration_method);
pde_p->NbMaxRelaxationIts() = NbMaxItsRelaxation();
\end{verbatim}

After that we invoke the method \texttt{dobj.Select( DemoObj::NECKLACE )}; in this case the default demoObj is a Necklace, but for this work we use only DemoObj Ragdoll.

Finally, we call \texttt{sdl\_idle()}, this method takes the messenger of processed the library of sdl.
7.2 Code Structure

The method `Select(DemoObjType _object)` is very important. In this method the first step is reset the structure and we can choose three different objects:

- NECKLACE
- CLOTH
- RAGDOLL

after that, we set the integration method

```cpp
pde_p->SetPDESolver(integration_method);
```

When we choose the method `_DefRagDoll()` we create the structure of ragdoll, in fact this method offers a set of methods useful for this purpose. Figure 7.2 shows the sequence diagram of method `Select(DemoObjType _object)`.

![Sequence Diagram Select](image)

The method `_InsertParticle(CoordType &_pos, int _index)` inserts the respective particles in a specified coordinate. This method set the position of particle, reset to zero the velocity of particle and learn the identification. Figure 7.3 shows the sequence diagram of the `_InsertParticle(CoordType &_pos, int _index)`

7.2.1 Insert constraint

In the following we explain all methods for inserting the constraints inside the system.

**Insert Stiff Constraint**

The method `InsertStiffConstraint(Particle* p1, Particle* p2)` inserts equidistance constraints between two particles; we figure out the distance between two
particle and insert the stiff constraint. This method (as others methods that insert any type of constraints) recalls the method SetConstraint() of class Constraint. The class Constraint defines a structure for all constraints. The Figure 7.4 shows the sequence diagram of method _InsertStiffConstraint(Particle* p1, Particle* p2).

Insert Plane Constraint

The method _InsertPlaneConstraint() inserts the floor constraints, it don’t allow the penetration of ragdoll inside the floor. We consider the distance of all particles with the plane and when the particle inter-penetrates the floor the particle is projected in a valid position. In particular when the distance from plane is minor of
7.2 Code Structure

zero we have:

```
distFromPlaneOpos = Distance<double>(_plane, (**ri).OPos());
(**ri).OPos() +=_plane.Direction()*(1 + refl_coeff)*(-disFromPlaneOpos);
(**ri).P() +=_plane.Direction()*(1 + refl_coeff)*(-disFromPlane);
(**ri).P() = (**ri).P()*damp_coeff+ (**ri).OPos()*(1 - damp_coeff);
(**ri).P() = _plane.Projection(unconstrained_position);
```

where the (**ri).OPos() is the old position particle, (**ri).P() is the current position of particle and refl_coeff is the reflection coefficient, damp_coeff is the damping coefficient. The Figure 7.5 shows the sequence diagram of method _InsertPlaneConstraint().

![Sequence diagram of InsertPlaneConstraint](image)

**Figure 7.5: Sequence diagram of InsertPlaneConstraint**

**Insert All Joint Constraint**

The method _InsertAllJointConstraint() permits to limit the movement of ragdoll in three dimensions. This method creates the movement constraints which depend of degrees of freedom of the particles.

As mentioned in chapter 4, we must limit the rotation in the space of ragdoll. The method implemented permit to make this.

It is obviously that the constraint depend to the considered particles. In general, the implemented structure is the following: we consider the particle that we linked by a stiff constraint, the child particle can rotate in according to degree of freedom.
of the father particle. We limit the movement of child particle as mentioned in Chapter 4.

When we recall a method `SetConstraint()` of class `Constraint`, we follow the same reasoning of the `_InsertStiffConstraint(Particle* p1, Particle* p2)`. Figure 7.6 shows the sequence diagram of method `_InsertAllJointConstraint()`.

![Figure 7.6: Sequence diagram of InsertAllJointConstraint](image)

**Insert Sphere Constraint**

The last method `_InsertSphereConstraint()` represents the environment constraint where our ragdoll does not penetration some objects in the space. Specifically, we want that our ragdoll collides with big or little spheres without penetrate them.

As mentioned in chapter 6, the collision detection is a big problem of computer graphics. In many video games the problem of collision detection is always on the upgrade.

In this work we consider the collision among the ragdoll and some spheres as depicted in the following. When one particle collides with a sphere (the sphere are in the space, and we know the sphere’s positions in the space), if the distance between the particle’s position and the centre of the considered sphere is less than the radius belonging to the considered sphere, then the new particle’s position will be figured out summing the center sphere plus Δ, Δ is equals:

\[
\text{int delta} = ((\text{pi}\rightarrow\text{P}() - \text{c}).\text{Normalize()} * r * 1.001) \\
\text{pi}\rightarrow\text{P}() = \text{c} + \text{delta} ;
\]

Figure 7.7 shows the sequence diagram of method `_InsertSphereConstraint()`.

![Figure 7.7: Sequence diagram of InsertSphereConstraint](image)
7.2 Code Structure

7.2.2 Coat the ragdoll

The other important method that we implemented is CoatRagdoll(). This method is inside the class my_sdl, in particular implemented on the method display().

As mentioned in chapter 5, we follow the simplified skin from point to coat the ragdoll. When we call the method CoatRagdoll(), it recalls all methods of draw the important part of the ragdoll.

Figure 7.8 shows the sequence diagram of method CoatRagdoll().

In this method we call all methods necessary for coating the ragdoll. The name of all methods are alone significant.
7.3 Conclusion

In this chapter we discussed our final project, we illustrated all important methods that we implemented, and we explained the flux of the code as well. In the next chapter we will show the final results commenting some screenshots.
Chapter 8

Result

This chapter makes some evaluations about the project made in this work. It will be presented and commented some screenshots about whole project. We show the results of motion constraints without and with the shape of ragdoll. Finally, we will show some screenshots with a realistic fall of the ragdoll between objects in the space.

8.1 Summary

In this thesis the goals that we want reach are:

- Build a ragdoll with motion constraints.
- Coat the Ragdoll with a geometric primitive.
- Develop a realistic three dimensions fall of the ragdoll.

The point of departure is create a ragdoll in three dimension.

The Figure 8.1 shows the beginning of our work. In this screenshot we show the point of departure of this work. We have a skeleton without movement constraint in fact our ragdoll at the beginning has only the equidistance constraint. In the system is presented only gravity force (-9.81). The ragdoll is a simplification of a character.

8.1.1 Motion constraints

After create a ragdoll in the space we have provided a limit the movement of the ragdoll.

In fact in Figure 8.2 we show a set of screenshots that represent one motion constraint. We represent the constraint for shoulder joint. The balls red and green represent the force that we applied for move the hand inside the possible allowed area. In this case our ragdoll is a fixed in the space. We have fixed all particles except the particle hand and elbow.
The movement of hand respects the limit of rotation of shoulder joint and the elbow joint. In fact in the Figure 8.2 is shown all possible motion of particles elbow and hand inside the allowed area.

8.1.2 Coat the ragdoll

After creating all motion constraints, we are focused on coating the ragdoll. We used a simple geometric primitives as parallelepipeds and triangles.

We used the method illustrated in chapter 4, such method is a simplified of skin from point.

8.1.3 Fall in the space and between objects.

After limit the movement of ragdoll and create a shape of ragdoll we consider also the fall of our ragdoll in the space and between objects in the space.

The simple fall in this space is not relevant. In this case all constraints are respected and there is a further constraint, the constraint of plane. The plane’s constraint represents the constraint which don’t allow penetration inside the ground to the ragdoll.

We considered the ragdoll’s fall in the space among objects (in this case we consider the fall between spheres) the case of test becomes very interest. Beyond to consider the plane we consider all objects in the space, the ragdoll in fact not penetrate neither plane nor objects.

Collision with a big sphere

The first test represents the fall of the ragdoll on the big sphere. The ragdoll after the collision not penetrate the sphere on the contrary the ragdoll stay on the sphere
8.2 Conclusion

in position seated. The Figure 8.3 shows this case of test, the ragdoll that falls on the big sphere.

**Collision with a three big spheres**

The second test represents the fall of the ragdoll among three big spheres. The ragdoll after the collision not penetrate the sphere and then the right arm and the left arm collision with the sphere and in conclusion the ragdoll collides with a big sphere. In this case the ragdoll is without the shape. The Figure 8.4 shows this case of test, the ragdoll that falls among the big spheres.

**Collision with a three big spheres and the ragdoll is shaped**

The third test represents the fall of the ragdoll among three big spheres with the shape of the ragdoll. The ragdoll is shaped with a method mentioned in the chapter 5, after the collision not penetrate the sphere and then the right arm and the left arm collision with the sphere and in conclusion the ragdoll collides with a big sphere. The Figure 8.4 shows this case of test, the ragdoll that falls among the big spheres.

8.2 Conclusion

We showed the results of our application. We discuss all goals which we have reached. In the next chapter we explain some possible future works, and the conclusion of this work.
Figure 8.2: Motion’s constraint, constraint for shoulder joint
8.2 Conclusion

Figure 8.3: The fall of ragdoll on the big sphere

Figure 8.4: The fall of the ragdoll among some sphere without the shape
Figure 8.5: The fall of the ragdoll among some sphere with the shape
Chapter 9

Conclusion and Future works

The aim for this thesis was to find a good way to represent a ragdoll’s fall as much realistic as possible. In order to do this, we studied some important mathematical concepts. After that, we figured out some constraints assigned to the ragdoll’s joints, and finally we coated the ragdoll with geometric primitives.

Moreover, the former constrained ragdoll has been tested to see the fall results. We put some objects in the space, which are in particular spheres, in order to see the ragdoll’s behaviour after a collision with them.

In order to make this goal, we studied some mathematical concepts, such as Numerical Integration and rotations in the space. The Numerical Integration has been helpful for solving the ordinary differential equation (ODEs) compared to the system of particles. The rotations, instead, has been helpful both to limit the joint’s movements, and to create the region in order to implement the constraints.

We provided a simple method to limit the joint’s rotations of our ragdoll. Mainly, we constrained the Degree Of Freedom (DOF) belonging to each considered joint. For instance, the constraint for shoulder joint is composed by three different constraints, one for each Degree Of Freedom. Moreover, there are stiff constraint defined on the ragdoll which are used to have a equidistance constraints between two neighbor joints.

Another important step was to coat the ragdoll with geometric primitives. We used a method called simplified skin from points, which is a variation and simplification from the traditional skinning method. Such used method figures out some main points which will be used to draw geometric primitives for coating the ragdoll. In our case, the geometric primitives are triangles for coating the ragdoll’s trunk; differently, quadrilaterals has been used for coating arms, legs, neck, and abdomen.

We showed, by means of some screenshoots, that our ragdoll has got a good behaviour looking at its fall. Some other good results are achieved looking at the fall, when there are some objects in the space.

In the future, a possible work could be to make some improvements for coating
Conclusion and Future works

the ragdoll. In particular, the method which has been used could be replaced by the traditional skinning in order to have a more beautiful ragdoll’s shape.

Finally, this thesis work has been a really great life experience for me, because I learned a lot of things working in one of the most advanced country in Europe. Moreover, such experience has been great also for my personal appearance, in fact I worked every day with great passion for last university work.
Appendix A

Software

A.1 Set up the development environment

This appendix shows the all action to install the programme. The order of installation is the following:

- Install TortoiseCvs.
- Install the library IdoLib "Interactive deformable object Library".
- Install the library VcgLib "Visual computer graphics Library".
- Install the library SDL "Simple DirectMedia Layer".
- Install the library GLUT "OpenGl Utility Toolkit".
- Install the library GLEW "OpenGl Extension Wrangler".

A.1.1 Microsoft Visual Studio 2005

In this project has been used as the development environment Microsoft Visual Studio 2005 Professional edition. Visual Studio 2005 has been installed in C:\ProgramFiles.
The all project has been written using Visual C++.

A.1.2 TortoiseCvs

TortoiseCvs [37] lets to work with files under CVS version control directly from Windows Explorer. It’s freely available under the GPL.\footnote{GPL is acronym “General Public License” this license is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.}

With TortoiseCvs you can directly check out modules, update, commit and see
Software

differences by right clicking on files and folders within Explorer. You can see the
state of a file with overlays on top of the normal icons within Explorer.
TortoiseCvs is available on http://www.tortoisecvs.org/index.shtml.

A.1.3 IdoLib

IdoLib "Interactive deformable object Library "[32] is a C++ library which aims
to provide a simple and as general as possible tool for developing software for
simulating deformable objects.
In the following will be explained the procedure to install the IdoLib:

- Create one folder with name IdoLib.
- Click with right bottom of the mouse and press checkout.

After that it will pop up a window dialog as shown in the Figure A.1
Is very important make these instruction:

1. Where you see Protocol: put Password server(:pserver)
2. Where you see Server: put IdoLib.cvs.sourceforge.net
3. Where you see Repository: put/cvsroot/idolib
4. Where you see Username: put anonymous
5. Where you see Module: put idolib
6. Press ok

In the folder created at previous step after the update phase, there are tree folders:
idolib, projs and wrap.
In folder idolib there are many files with extension .h that they are used to imple-
ment simulating deformable object.
In folder projs at the moment there are eight project in continuous development.
For this thesis it has been modified the project number four, the name is M4_ParticleBox.

A.1.4 VcgLib

VcgLib "Visual computer graphics Library "[33]is a portable C++ templated li-
brary for manipulation, processing and displaying with OpenGl of triangle and
tetrahedral meshes.
The library, released under the GPL license is the result of the collaborative efforts
of the Visual Computing Lab "VCGLab "[38] of the ISTI "Institute of the Italian
National Research Council "[39]
In the following will be explained the procedure to install the VcgLib:

- Create one folder with name different from veg.
A.1 Set up the development environment

Figure A.1: Pop up of Idolib

Figure A.2: Pop up of VcgLib
Software

- Click with right bottom of the mouse and press checkout.

After that it will pop up a window dialog as shown in the Figure A.2

It is very important make these instruction:

1. Where you see Protocol: put Password server(:pserver)
2. Where you see Server: put vcg.cvs.sourceforge.net
3. Where you see Repository: put /cvsroot/vcg
4. Where you see Username: put anonymous
5. Where you see Module: put vcg
6. Press ok

In Figure A.3 is shown the result during the phase update. In the folder created at

![Figure A.3: Update of folder Vcg with TortoiseCvs](image)

previous step after the update phase there are four folder: apps, docs, vcg and wrap.

In the folder apps there are some application made with the library’s, as metro

standard tool for measuring differences between triangular meshes and ShadeVis a

tool for per-vertex computation of a static ambient occlusion term.

In the folder docs there is little documentation, styleguide and manual generated

by Doc++. 

72
In the folder vcg there is a library with all math definitions, complex definitions, space definitions.

In the folder wrap there are wrapping concepts, e.g. draw a mesh using OpenGL, trackball decorators, file import export.

A.1.5 SDL
SDL "Simple DirectMedia Layer" [34] is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer. It is used by MPEG playback software, emulators, and many popular games, including the award winning Linux port of "Civilization: Call To Power." SDL is distributed under GNU LGPL version 2. This license allows you to use SDL freely in commercial programs as long as you link with the dynamic library. The version of library SDL used is 1.2.11 stable.

After the download, put your file in these way:

- File.h: C:\ProgramFiles\MicrosoftVisualStudio8\VC\PlatformSdk\include
- File.lib: C:\ProgramFiles\MicrosoftVisualStudio8\VC\PlatformSdk\lib
- File.dll: C:\windows\system32

A.1.6 GLUT
GLUT "OpenGl Utility Toolkit" [35] is a window system independent toolkit for writing OpenGl programs. It implements a simple windowing application programming interface (API) for OpenGl. GLUT makes it considerably easier to learn about and explore OpenGl programming.

The version of library GLUT used is 3.7.6.

After the download, put your file in these way:

- File.h: C:\ProgramFiles\MicrosoftVisualStudio8\VC\PlatformSdk\include\GL
- File.lib: C:\ProgramFiles\MicrosoftVisualStudio8\VC\PlatformSdk\lib
- File.dll: C:\windows\system32

A.1.7 GLEW
GLEW "OpenGl Extension Wrangler" [36] is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGl extensions are supported on the target platform. OpenGl core and extension functionality is exposed in a single header file. GLEW has been tested on a variety of operating systems, including Windows, Linux, Mac OS X, FreeBSD, Irix, and Solaris.
Software

The version of library GLEW used is 1.3.6. After the download, put your file in these way:

- File.h: \ProgramFiles\MicrosoftVisualStudio8\VC\PlatformSdk\include\GL
- File.lib: \ProgramFiles\MicrosoftVisualStudio8\VC\PlatformSdk\lib
- File.dll: \windows\system32

A.2 Run the project

After all phase to set up the development environment the last phase is open the project. For this work thesis has been opened idolib\projs\mscv\all_projects.sln and after has been modified the project number four "M4_ParticleBox". In Figure A.4 has been shown the main.cpp of project with Microsoft Visual Studio 5.

![Figure A.4: M4_ParticleBox with microsoft Visual Studio 5](image-url)
Appendix B

OpenGL

B.1 OpenGL

OpenGL[40][41](Open Graphics Library) is a interactive computer graphic system that allows to write programs that access graphics hardware. OpenGL was developed by Silicon Graphics Inc¹.

Silicon Graphics Inc. in 1992 and is popular in the video game industry where it competes with Direct3D on Microsoft Windows platforms (see Direct3D vs. OpenGL). At the beginning her called IRIS GL OpenGL is widely used in CAD, virtual reality, scientific visualization, information visualization, flight simulation and video game development. Figure B.1 shows the logo of OpenGL. The library is composed from about 150 command and realized

![Figure B.1: Logo OpenGL](image)

in various language. This library is available ([43]) on platform Windows e UNIX. This is thought to be hardware-independent and for this reason doesn’t command to operate with windows.

¹Silicon Graphics, Inc.[42] (SGI) is a manufacturer of high-performance computing solutions, including computer hardware and software. SGI was founded by Jim Clark and Abbey Silverstone in 1982, initially as a maker of 3D graphics display terminals.
OpenGL

B.2 OpenGL Structure

OpenGL[44] is a library "rendering library". In this library there isn’t a predefined structure as in PHIGS\(^2\), OpenGL work in "immediate mode". To have the complex object we can use the more libraries above OpenGL.

OpenGL[46] [47] uses a set of moderate primitive, moreover he don’t write on the scheme but he accedes a framebuffer.

OpenGL [48] is a specific that defines a particular process of pipeline and for the visualization of image and geometric figures. OpenGL takes in input a point’s primitive, of line and polygon and he convert in pixel. This process is realizes by mean of graphic pipeline called OpenGL state-machine, where the state’s variable rule the rendering. In the Figure B.2 there is a scheme of execution of a OpenGL program. The specification of OpenGL is supervised from OpenGL Architecture Review Board (ARB). L’ARB is a set of companies that they interested to create a consistent API.

B.2.1 Convention

All command have the prefix gl (as an example: glClearColor). All constant and the variable of state are written in upper-case and the prefix is GL_ and the word are separate from the character "_" (example:GL_COLOR_BUFFER_BIT)

\(^2\)PHIGS[45] (Programmer’s Hierarchical Interactive Graphics System) is an API standard for rendering 3D computer graphics, at one time considered to be the 3D graphics standard for the 1990s. Instead a combination of features and power led to the rise of OpenGL, which remains the de facto 3D standard to this day. PHIGS is no longer used.
B.3 Libraries on OpenGL

In OpenGL the type of internal name have the prefix GL (as an example: GLbyte, GLshort, GLint, GLfloat, GLdouble)

B.2.2 Syntax

More command finished with two suffix (as an example: glVertex2f()). The first (2) is the number of argument instead the second (f) is the type(float). The type are:

- \( b \): 8-bit integer.
- \( s \): 16-bit integer.
- \( i \): 32-bit integer.
- \( f \): 32-bit floating-point.
- \( ub \): 8-bit unsigned integer.
- \( us \): 16-bit unsigned integer.
- \( ui \): 32-bit unsigned integer.
- \( d \): 64-bit floating-point.

A some command can have a third suffix (v) this is applied to vector.

B.3 Libraries on OpenGL

Many libraries have been developed on OpenGL. This gives a more functionality. The library GLU [49], are included in all implementation of OpenGL, an other library is added a development environment as GLUT and SDL (these two libraries are used to create user interface an control event, keyboard, mouse) Simple graphics interface can developed with libraries GLUI [50] FLTK [51].

Figure B.3 shows the libraries presented in OpenGL

![Figure B.3: The OpenGL libraries](image-url)
Figure B.4 shows the libraries for Windows, Linux and MacOSX.
Bibliography


BIBLIOGRAPHY


[23] Qiang Liu and Endmond C. Prakash. The parameterization of joint Rotation with the Unit Quaternion. School of Computer Engineering, Nanyang Technological University, Singapore 639798.


[27] Definition Skinning. URL:http://graphics.ucsd.edu/courses/cse169_w05/3-Skin.htm.


[34] SDL. URL:http://www.libsdl.org.


[38] VcgLab. URL:http://vcg.isti.cnr.it/joomla/index.php.
BIBLIOGRAPHY


På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Uphovsmannens ideella rätt innefattar rätt att bli nämd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/.

© Fiammetta Pascucci