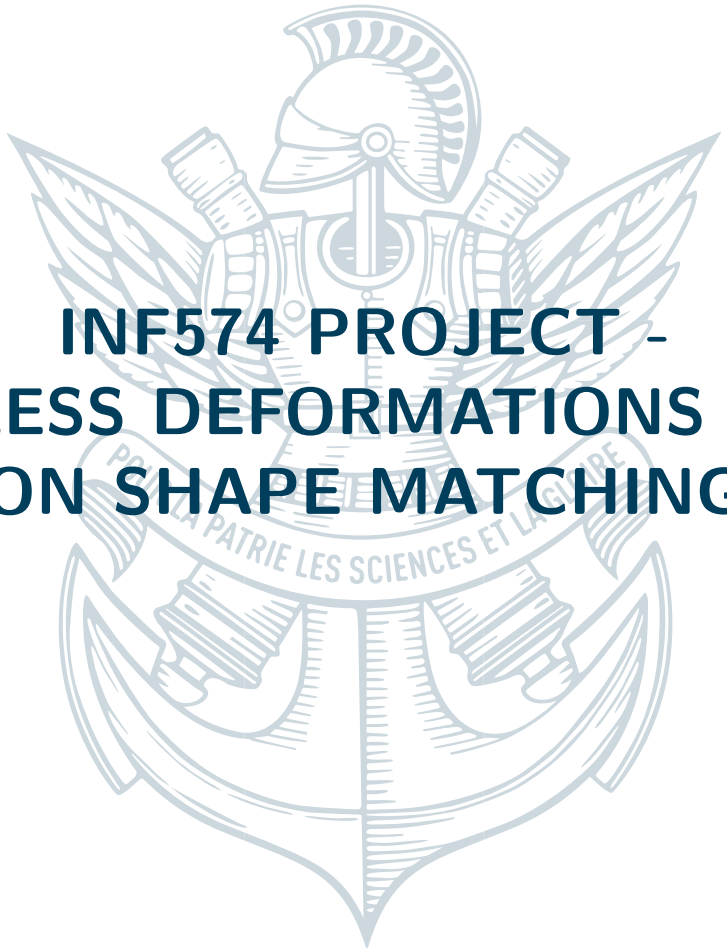

INF574 PROJECT - MESHLESS DEFORMATIONS BASED ON SHAPE MATCHING

December 2022

Mergui Hanna
Bohm Maximilien

Table of Contents

List of Tables

Introduction

In most interactive computer graphics applications, objects are considered as rigid bodies with rigid body motions.
In the real world, on the contrary depending on the amplitude of the force applied almost any object possesses elastic properties to some point. Therefore, if one wishes to represent interactions as real as possible, elastic deformations should be considered. However, older techniques were not very efficient and very computationally expensive. Therefore, finding a stable solution i.e. an elastic equilibrium in complicated objects was near impossible.

This paper by Müller in 2005 [1] claims to have found a better solution based on creating an elastic model based on the conservation of the shape.
Instead of using interacting with the mesh itself, we model the object as a particle system. Thus any interaction with the object will be 2-fold.
First, the force is applied to each particle independently of the shape.
Second, the 'return' of the particles will be dealt by finding the rotation of the original shape that minimizes the sum of the distances and then a 0-rest length spring model is applied to each particle to it's 'goal' position to model the elastic interaction as shown below.

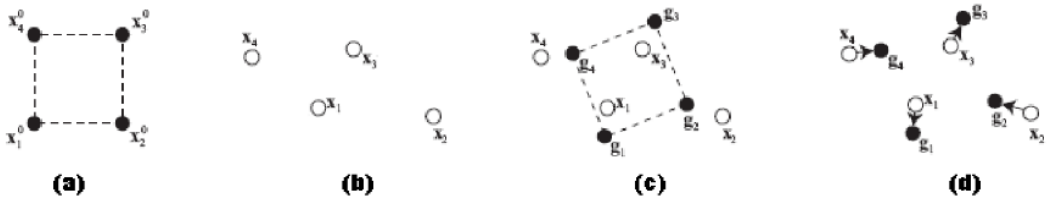

Figure 1: Overview of Shape Matching

1 Description of the methods found in the paper

1.1 Objects as particle systems

Contrarily to the typical LibIGL methods, objects in interactions here are encoded using a system of particles.
Within each particle is encoded the following:

- positions: a vector $\vec{x} = (x, y, z)$;
- velocities: a vector $\vec{v} = (v_x, v_y, v_z)$ calculated by taking the derivative of the respective positions;
- acceleration: a vector $\vec{a} = (a_x, a_y, a_z)$ computed by taking the derivative of the velocities;

In the initialization of the object, each particle has 3 datapoints set and all other quantities are initialized as 0.

Using this paper's method, at each time step, we update the informations backward, we compute the acceleration using the force and Newton's second law ($\vec{F} = m\vec{a}$) and update the positions by integrating twice.

1.2 Integration methods

We wish to integrate the acceleration to find the positions at each time-step t. Now, as the forces applied by the user are done at runtime, analytical formulas cannot be used and we must therefore rely on numerical integration. For this sake, in this paper two methods are used (Implicit and Explicit Euler Integration Methods) and then built apon to more specifically match our use.

1.2.1 • Explicit Numerical Integration based on Euler Integration

The explicit Euler integration method is based on knowing one sample point of the integrated function $f(x)$ as well as knowing the function to integrate $f'(x)$.
Mathematically, it is written as follows

$$f(t + \delta h) = f(x) + (\delta h)f'(x) \tag{1}$$

where δh represents the time-step. Let us apply this method to an undamped mass-spring system to see how it works.

Remark. *Suppose we have an undamped mass-spring system. For simplication, we suppose it is placed along the x-axis.*
The acceleration force applied to the system is thus,

$$F = k(l_0 - x(t)) \tag{2}$$

where l_0 is the resting length and k is the spring constant. Knowing that the acceleration $a(t)$ is the derivative of the velocity $v(t)$, using Euler's integration method, we have that,

$$v(t + \delta h) = v(x) + (\delta h)a(x). \tag{3}$$

Now, using Newton's second law, we know that,

$$a = F/m, \tag{4}$$

hence we get that,

$$v(t + \delta h) = v(x) + (\delta h)\frac{k(l_0 - x(t))}{m} \tag{5}$$

Finally, to get the positions, we would need to integrate once more, and that is where we see that the explicit method cannot be used again.

Indeed, once the explicit Euler integration method is used, we get the value of the integrated function at the time-step $t + \delta h$ and not at the time step t.
Hence to calculate the position of the particle at the time step $t+\delta h$, implicit Euler integration method is used.

1.2.2 • Implicit Euler Integration

Implicit Euler Integration, also known as Backward Euler Method is very similar to the explicit method with the only difference being that the function to be integrated is evaluated at $f'(x + \delta h)$.
Hence, the new equation becomes,

$$f(t + \delta h) = f(x) + (\delta h) f'(x + \delta h). \tag{6}$$

The main difference between the two methods is that while implicit Euler integration leads to more stable solution than explicit methods, in practice, it requires solving a linear equation which usually is more costly than the explicit method.
For the best results, we thus use a mix of both methods for the two integration steps required.

2 Shape Matching

In the paper's approach, the shape matching problem is stated as follows:

Definition (Shape Matching)**:** *Given two sets of points $\vec{x}_i^0$ and $\vec{x}_i$.*
Find the rotation matrix $\mathbf{R}$ and the translation vectors $\vec{t}$ and $\vec{t_0}$ which minimize,

$$\sum_i w_i (\mathbf{R}(\vec{x}_i^0 - \vec{t_0}) + \vec{t} - \vec{x}_i)^2, \tag{7}$$

where w_i are the weights of individual points. For our case, we will use $w_i = m_i = 1$.

It turns out that the optimal translation vectors can be analytically computed and it has been found that they represent the center of mass of the initial shape and the center of mass of the actual shape,

$$\vec{t_0} = \vec{x}_{cm}^0 = \frac{\sum_i m_i \vec{x}_i^0}{\sum_i m_i}, \quad \vec{t} = \vec{x}_{cm} = \frac{\sum_i \vec{x}_i}{\sum_i m_i}. \tag{8}$$

Finding the optimal rotation requires a few more computation. Defining the relative positions $\vec{q}_i = \vec{x}_i^0 - \vec{x}_{cm}^0$ and $\vec{p}_i = \vec{x}_i - \vec{x}_{cm}$ and doing some mathematical magic that can be found in the paper by Müller, we find that the optimal transformation is

$$\mathbf{A} = \left(\sum_i m_i \vec{p}_i \vec{q}_i^T \right) \left(\sum_i m_i \vec{q}_i \vec{q}_i^T \right)^{-1} = \mathbf{A}_{pq} \mathbf{A}_{qq}, \tag{9}$$

leading to the optimal rotation being,

$$\mathbf{R} = \mathbf{A}_{pq} \left(\sqrt{\mathbf{A}_{pq}^T \mathbf{A}_{pq}} \right)^{-1}. \tag{10}$$

Finally, we get that the goal positions are,

$$\vec{g}_i = \mathbb{R} \left(\vec{x}_i^0 - \vec{x}_{cm}^0 \right) + \vec{x}_{cm} \tag{11}$$

2.1 Extended Integration Method

Now that we know the goal positions $\vec{g}_i$ we can construct an integration method that does not exceed the goal positions.

$$\vec{v}_i(t+\delta h) = \vec{v}_i(t) + \alpha \frac{\vec{g}_i(t) - \vec{x}_i(t)}{\delta h} + (\delta h) F_{ext}(t)/m_i \tag{12}$$

$$\vec{x}_i(t+\delta h) = \vec{x}_i(t) + (\delta h)\vec{v}_i(t+\delta h), \tag{13}$$

with $\alpha \in [0,1]$ as a stiffness parameter, and F_{ext} being the force created by the user.
We can see that if $\alpha = 1$, the position $\vec{x}_i$ will directly move to the goal position $\vec{g}_i$.
However, if $\alpha < 1$, the point will move to the goal position $\vec{g}_i$ slower.

3 Deformation Types

3.1 Linear deformations

Using the method of Shape Matching, we can only simulate small deviations from the rigid shape therefore we therefore use a combination of the linear transformation matrix $\mathbf{A}$ and the optimal rotation $\mathbf{R}$.
Using a control parameter $\beta \in [0,1]$, we use the matrix $\beta\mathbf{R} + (1-\beta)\mathbf{R}$ to compute the goal positions.
With this modified equation, this means that the shape of the object can undergo linear transformations.
As done in the paper, for conservation of volume, the matrix $\mathbf{A}$ is divided by $\sqrt[3]{\det(A)}$ making sure that the determinant stays 1.

3.2 Quadratic deformations

If we want to represent more than shear and stretch, linear transformations will not be enough.
In order to involve bending and twisting, we move towards quadratic transformations.
Defining the quadratic form,

$$\tilde{\vec{q}} = [q_x, q_y, q_x^2, q_y^2, q_z^2, q_x q_y, q_y q_z, q_z q_x]^T \in \mathbb{R}^9 \tag{14}$$

The problem now becomes finding the optimal quadratic transformation $\tilde{\mathbf{A}}$ minimizing $\sum_i m_i(\tilde{\mathbf{A}}\tilde{\vec{q}}_i - \vec{p}_i)$.
Doing the same trickary than before, we use $\beta\tilde{\mathbf{A}} + (1-\beta)\tilde{\mathbf{R}}$ to compute the goal positions, where $\tilde{\mathbf{R}} = [\mathbf{R}, 0, 0] \in \mathbb{R}^{3\times 9}$.

4 Extensions of these methods to different interactions

4.1 Plasticity

This methods of deformation based on shape matching has some fun applications. One of these applications relates to plasticity. Indeed, considering our model, if one takes the 'pure' deformation matrix $\mathbf{S} = \mathbf{R}^T\mathbf{A}$, testing it against the identity matrix, we can add a condition when it surpasses some threshold and update the matrix $\mathbf{S}$ such that this conditional identity is passed onto the relative positions $\vec{q}_i$.
This leads to permanent deformation when some threshold is passed.

4.2 Collisions

Another interesting application of this method is in the collisions. In computer graphics collisions are dealt with using a two-step method: collision detection and collision response.
For more complex animations, integrating this into our model would be very interesting.
Sadly, due to the time constraint and the limitations of LibIGL, we did not manage to implement this however we would like to share with you our thought of how to do it.
For the collision detection, a simple bounding box surrounding each object could be tested at each time-step to see if it interests another bounding box.
After such event, we can more specifically test what part of the objects are colliding.
As for the collision response, one way to do it would be the following method:
Suppose the collision happens at time-step t^*, then one would,

- First, compute the normal vectors to the surface of both object at the points of interactions;
- Second, set the velocity $v(t^* + \delta h)$ to be equal to the projection of the colliding velocity $v(t^*)$ onto the normal vector of the colliding surface at the point $\vec{x}_{i,target}$ such that,

$$v(t^* + \delta h) = p_{\vec{x}_{i,target}}(v(t^*)) \tag{15}$$

- Third, compute $v(t^* + 2\delta h)$ using the shape matching method.

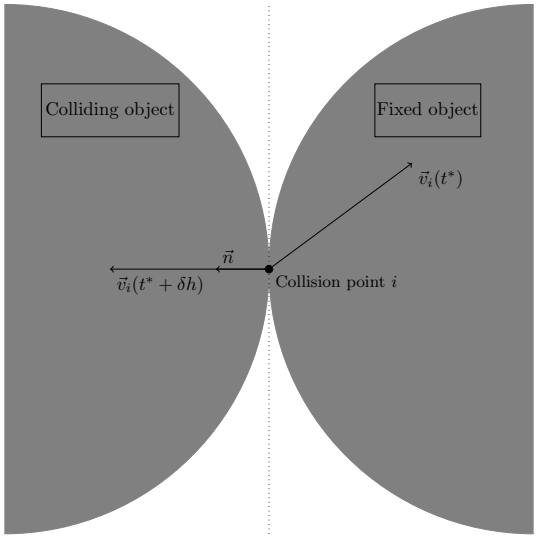

Figure 2: Overview of collision response

5 Limitations of this algorithm

Although arguably this paper provided a new means to involve elasticity in objects in computer graphics it's use is pretty limited.
As indicated in the paper, the main use of this algorithm would be in video games; however, for most cases this algorithm tends more towards a pleasing sight rather than a physically correct one.
Each object require a different tweak of all the parameters to look realistic. Each object is constituted of a complex inner structure that modify the different parameters in elastic collisions.
If the desired use is to seem pleasing and not necessarily realistic then this paper is extremely interesting as it is very efficient in nature but in other cases it will be more computationally expensive to correct the parameters rather than having a rigid-body animation instead.

References

[1] Muller M. Meshless deformations based shape matching. *ETH Zurich*, 2005.