Meshless Deformations Based on Shape Matching

Implementation by Seyed Sepehr Seyed Ghasemipour
University of Toronto
sepehr.seyedghasemipour@mail.utoronto.ca

Abstract

This project implements the simulation technique proposed by [Müller et al. 2005]. Motivated by stability and efficiency over physical accuracy, the proposed method makes use of a simple particle system and an integration scheme based on shape matching to generate real-time simulations of deformable objects. The approach expresses a high level of controllability, making it especially useful in domains such as games and entertainment where tunability is a priority. We explore the possibilities of the method within such areas by looking at linear and quadratic deformation modes, as well as clustering techniques.

1 Introduction

At the time when [Müller et al. 2005] published their work, deformable objects were not commonplace within games and animation. Though models of deformable objects had been developed, they primarily focused on capturing physically accurate representations, sacrificing other factors that were imperative to game contexts in the process. [Müller et al. 2005] address three of these factors; By adopting a geometric framework and relaxing the need for complete accuracy, they take large steps towards more efficient, stable, and versatile simulations.

1.1 Efficiency and Stability

Numerical integration of positions and velocities is an unavoidable part of any simulation technique. At the same time, integration schemes are often a bottleneck to the efficiency and stability of a method. In general, explicit integration schemes offer cost effective calculations at the expense of stability. On the other hand, implicit integration schemes allow for stability at the expense of cost. Hence, there is a trade-off. Yet interactive simulations are not possible without one or the other. In this context, it is critical to recognize that what underlies this trade-off is the need to evaluate solutions to complex physical equations. By eliminating this need, one can instead form a proxy to the original results while encouraging more efficient and stable behaviour. This is precisely what [Müller et al. 2005] do when they reformulate the process of calculating energies and forces as a shape matching problem.

1.2 Versatility

Versatility as it pertains to the work of [Müller et al. 2005] is two fold. Firstly, ”in games ... the magnitude and shape of a deformation need to be controllable by the developer, tolerating a degradation of realism as long as the result looks realistic” (Müller, 2005). To this end, the method being studied performs well and offers a natural set of extensions. Namely, the alpha parameter controlling the degree to which particles are pulled towards their goal positions allows for a wide range of simulation from rigid bodies to the utmost of deformable. Additionally, the method as described in Section 2 finds velocity updates by optimizing for a linear transformation. This can be leveraged to produce a linear deformation mode which can also be extended to a quadratic deformation mode with small modifications. Furthermore, the algorithm also lends itself nicely to clustering, allowing for more realistic outcomes.

The second notion of versatility relates to object representation. In particular, a significant advantage of this method is that it only uses vertex information and therefore can operate on any object no matter the format or complexity. This is once again ideal for game and entertainment settings, as it eliminates restrictions due to formatting.

2 The Algorithm

The premise of the algorithm relies on the tendency for elastic objects to return to their rest state. As such, the method takes in particles corresponding to an object, and at every time step pulls each in a direction that encourages the original shape. The following outlines the general procedure.

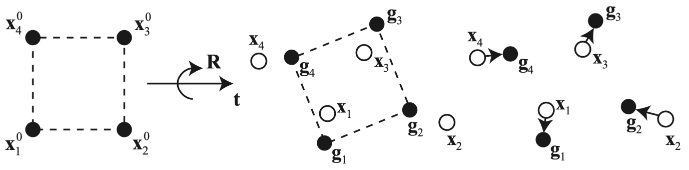


Figure 1: algorithm outline by [Müller et al. 2005]

For particle i,
Let x_i^0 denote the initial configuration.
Let m_i denote the particle mass.
Let x_i denote the current configuration.
At each step, we:

1. match the initial configuration of the object to the current configuration to obtain goal positions g_i
2. pull each position x_i towards g_i

Therefore, the only matter of concern is finding g_i.

2.1 Shape Matching

The problem of shape matching can be thought of as finding the rigid transform of the original configuration that is closest to the current configuration. As such, we can formulate it as a least squares optimization problem by the following:

$$\sum_i m_i * (R(x_i^0 - t_0) + t - x_i)^2$$

where we try to find rotation matrix R and translations t and t_0 that minimize the above expression. As it turns out, the optimal choices for t and t_0 are the center of masses of the current and initial configuration respectively. Additionally, to obtain R it is easier to first solve for the linear transformation A in $\sum_i m_i * (A * q_i - p_i)^2$, where $q_i = x_i^0 - t_0$ and $p_i = x_i - t$, and then use polar decomposition to obtain R. Namely, the optimal solution to the corresponding linear transformation problem is given by

$$A = (\sum_i m_i * p_i \dot{q}_i^T)(\sum_i m_i * q_i \dot{q}_i^T)^{-1} = A_{pq} A_{qq}$$

A_{qq} is a symmetric matrix and cannot rotate. Thus, we can extract our rotation matrix R by applying polar decomposition to A_{pq}. Finally, we arrive at goal positions $g_i = R(x_i^0 - t_0) + t$.
Using this, we can define the position and velocity updates by

$$v_i(t+h) = v_i(t) + \alpha * (\tfrac{g_i(t) - x_i(t)}{h}) + \tfrac{h * f_{ext}(t)}{m_i}$$
$$x_i(t+h) = x_i(t) + h * v_i(t+h)$$

where α is a parameter between 0 and 1 that captures how deformable an object is.

3 Implementation

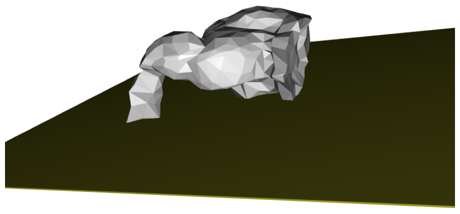


Figure 2: coarse bunny simulation under linear deformation mode

The proposed algorithm was implemented using Python alongside NumPy, Scipy, and Open3D packages. The bunny mesh was borrowed from coursework and rectangle models were developed using Blender. Gravity was implemented by adding a constant downward velocity at each time step. Collision with the ground was also handled. Detection was done by manually checking the vertex coordinates, while the response to each vertex was an upward velocity addition of equal magnitude. Lastly, we included a damping constant for more realistic effects.

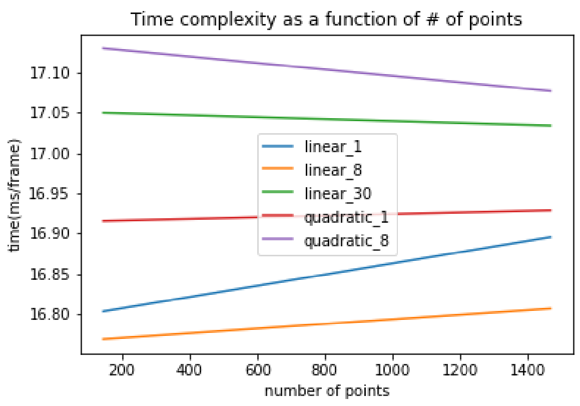
4 Extensions & Future Work

In their work, [Müller et al. 2005] discuss a linear deformation mode accomplished by calculating g_i using the linear transform A, as described in Section 2.1, as opposed to rotation matrix R. With further additions, a quadratic deformation mode can be attained as well; however the inner workings of that approach are not discussed in detail. Nonetheless, these features were implemented alongside the basic setup, and all instances boasted simplicity, versatility, and performance. Furthermore, a clustering method was applied to these simulations where separate overlapping regions of a mesh were matched individually. This allowed for more degrees of freedom and ultimately more natural simulations. Lastly, similar results to Figure 8 from the original paper, showcasing time complexity as a function of # of points for such clustering methods, was also reproduced using our implementation (in Appendix). Namely, we see that the performance of these methods demonstrates a linear relationship with respect to # of points being handled, and that quadratic deformation modes require more computation and thus more time. Stemming from this work, an interesting future direction would be to explore approaches for efficiently calculating higher order deformation modes, and to weigh the costs and benefits of incorporating such methods in interactive simulation frameworks such as the one studied in this project.

References

Müller, M. , Heidelberger, B., Teschner, M., and Gross, M. 2005. Meshless Deformations Based on Shape Matching. *ACM Trans. on Graphics 24*, 3 (Aug.), 471–478.

5 Appendix



A plot showcasing time complexity (in milliseconds per frame) as a function of number of points being simulated. Two rectangle meshes, with 144 and 1466 points respectively, were used to generate the results. In our implementation, the number of points does not seem to have much effect on performance.