Meshless Deformations Based on Shape Matching

SIGGRAPH 2005

Matthias Müller, Bruno Heidelberger,
Matthias Teschner, Markus Gross

COMPSCI 715 S2 C - Advanced Computer Graphics

Heiko Voigt, Daniel Flower, Daniel Weisser

14 October 2005

Contents

1 Introduction

1.1 Problem description

When objects in the real world have force applied to them, they change shape, i.e. deform, and then return completely or nearly back to their original shape, even if only a small amount. Therefore, graphical objects should also be able to change shape, because having deformable objects in computer games and animated movies, or even scientific simulations, would increase the realism a great deal.

However, up until now, most objects in interactive applications are not deformable. The main reason for this is probably efficiency. Techniques which try to simulate physical reality are very computationally expensive, not to mention difficult to program. Furthermore, having a stable solution (for example ensuring that the volume, mass or energy of an object stays constant) makes the technique even more computationally expensive. Finally, the deformation needs to be controllable by the programmer, so that the object behaves in a way that the developer intended.

The technique that we are studying [Müller et al., 2005] claims to solve all the above problems by using a simple geometric model. The simplicity ensures that the technique is efficient, easy to program, stable, and controllable by varying certain parameters.

1.2 Solution description

The main point of difference with this technique compared to others is that rather than representing the object with a mesh or surface representation, it is represented by a particle system. The idea is that the normal shape of the object is specified by some arrangement of particles, but when forces are applied to the particles they lose this shape.

The main focus of the paper is on finding the best transformation to describe the change in location of the particles from their original shape to their deformed shape, which is called "shape matching". Each particle is then pulled from its current position towards its corresponding position in the shape created by the transformation (i.e. towards its goal position), and the particles eventually return to their original positions. This is shown graphically in figure 1.1.

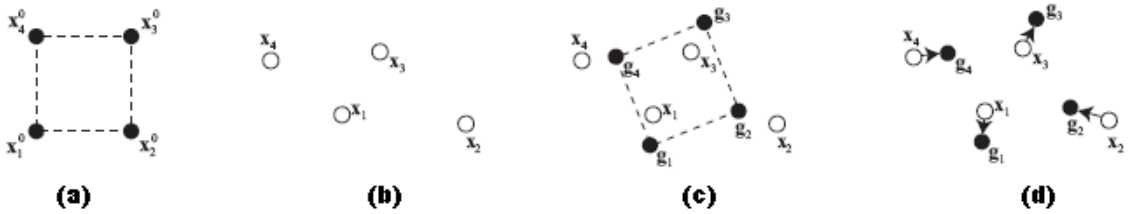

Figure 1.1: Overview of the shape matching algorithm. A shape starts in its normal shape, as shown in (a). After a force is applied, the particles lose their shape, as shown in (b). The shape matching algorithm finds the best transformation which will convert the original shape to the current shape with the least error, as shown in (c). This step is what makes the goal positions g_i that the particles are pulled to. Finally, in (d), the particles move towards their goal positions. The graphic is derived from [Müller et al., 2005]

1.3 Overview

This report gives a detailed overview on the theoretical aspects of the paper in section 2. After that our problems with the given are described and the algorithm is evaluated. Section 5 mentions the flaws and limitations that we found, when we tried to implement the algorithm described in the paper. The concept of clustering, as described in the paper, is ignored in this report.

For mathematical formulae, where it is necessary to distinguish between vector and non-vector quantities, the vector quantities are indicated in bold type.

2 Theoretical background

2.1 Physical background

2.1.1 Motion of objects

The paper requires a basic understanding of physics. The most important aspects to understand are the relationship between distance, velocity and acceleration, and Newton's second law of motion.

Distance, velocity and acceleration

Although we all know what these words mean, it is important to understand their relationship with each other. Velocity v describes how distance x changes over time, and acceleration a describes how velocity changes over time. In other words, velocity is the derivative of distance over time x', and acceleration is the derivative of velocity v' over time. Integration goes the other way, so integrating acceleration gives velocity, and integrating velocity gives distance. Mathematically, this can be shown as

$$v(t) = x'(t)$$
$$a(t) = v'(t)$$

and so also

$$a(t) = x''(t)$$

Figure 2.1 explains this connection graphically.

Newton's second law of motion

Newton's second law of motion states the relationship between the mass of an object, its acceleration, and the force applied to achieve this acceleration, which is:

$$F = ma$$

This is an important equation because it allows us to calculate how the velocity of an object will change when a force is applied to it. Another way to look at it is that acceleration is equal to force divided by mass, which is important because it shows that acceleration, and hence velocity, can only be changed when a force is applied.

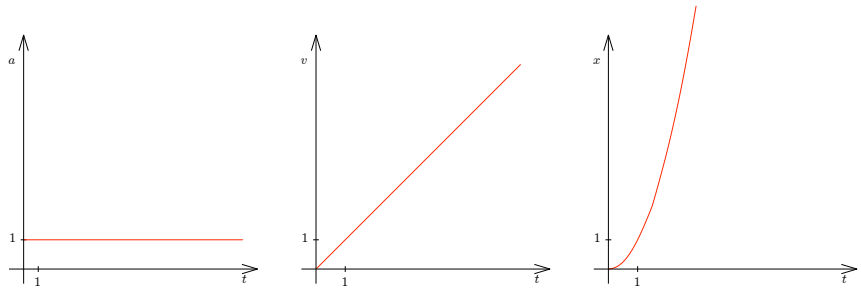

Figure 2.1: This illustrates the integration of all three items assuming a constant acceleration. ($a(t) = 1$,$v(t) = t$,$x(t) = t^2$)

2.1.2 Particle systems

The solution given in this paper requires that the objects be represented as a particle system. In graphics, objects are generally represented as meshes (i.e. vertices and information on how the vertices are connected) or by some surface representation.

In a particle system, an object simply has a set of particles. A particle might be a single point in space, or it may be a shape such as a small sphere. Each particle maintains its own position, velocity, acceleration and other properties depending on the application. The meshless deformations solution requires that a particle also keep track of its original position, goal position, and mass.

The original position is needed to be able to calculate the displacement of a particle, and the goal position is simply the position that the particle should move to (this is explained in subsequent sections). The mass is required so that the object's centre of mass can be calculated.

At each time step, each particle is updated separately. It's velocity is incremented by its acceleration, and its position is updated by its velocity. External forces such as gravity can also be added by increasing the velocity. Obviously if all particles have the same velocity and acceleration, then the object will move as if it were a solid object. It is only when some force disrupts this uniformity of movement, that things begin to get interesting...

2.1.3 Handling collisions and gravity

Common sense and basic physics tells us that no object will deform without an external force acting upon it. At first read then, it seems most curious that a paper dealing in deformations only mentions forces in passing. To understand why such little attention is given to forces, it is important to understand what exactly is the concern of this paper.

As explained earlier, an object is a group of particles, where each particle has its current position, and its goal position. When all the particles are in their goal positions, the

object will be in its regular shape, and the method this paper describes simply moves a particle from its current position towards its goal position. The paper is not concerned with how a particle became displaced from its goal position, even though without any displacement no deformations will occur.

There are several sources of deformation, however in this section we will look at just external, constant forces acting on all particles equally, such as gravity, and the forces experienced through collisions by individual particles[1].

Gravity

The effects of gravity are introduced in the **Integration** section[2]. The term that adds the force to a single particle is $h\mathbf{f}_{ext}(t)/m_i$, where h is the amount of time passed since the last update, $\mathbf{f}_{ext}(t)$ is the external force at that time, which is the same for all particles, and m_i is the mass of the particle.

Because force equals mass multiplied by acceleration, this equation can be simply thought of as the acceleration caused by gravity multiplied by the time step amount. Note that this equation is completely independent of the shape matching, and so adding gravity becomes the simple task of incrementing the velocity of each particle by a certain amount.

Collisions

Dealing with collisions in a computer simulation involves two steps: collision detection and collision response. Collision detection involves determining which, if any, objects are colliding with each other, and where the collision takes place. Collision response applies appropriate forces to the colliding objects so that they behave realistically. Although response obviously comes after detection, we will look at response first as this is more closely related to the preceding paragraphs.

Collision response Collision response involves applying an *instantaneous* force to a colliding object. An instantaneous force, as the name suggests, is a force that lasts for only a very short time, for example when a bat hits a ball, or when you punch your best friend in the face. Note that an instantaneous force cannot be added in the same way as a constant force like gravity. This is because an instantaneous force should be applied during just one time step, and because the time step length changes as the frame rate changes, depending on the frame rate a different amount of force would be added, and would never be the same amount of force as was requested.

[1] Another source of force comes from user interaction, such as dragging a particle with the mouse. This can be thought of as the equivalent of a collision response.

[2] Section 3.4, Equation 9 of the SIGGRAPH paper [Müller et al., 2005]

While the paper does not mention how to add instantaneous forces, once again because the shape matching does not take into account any forces, nor the velocity of the particle, we can simply set the velocity to a new value, and the next time the velocity and position of this particle is updated, it will be moving with a new velocity. This will cause the centre of mass of the object, and hence the goal positions, to change, and deformation will occur.

One way to calculate the response force vector is to get the normal of the point that the particle collided with. The response vector is simply the velocity vector reflected on the normal. This can be multiplied by an elasticity value between zero and one to simulate the energy lost in the collision, where zero means all energy was lost (and so the particle does not rebound) and one means no energy was lost (and so the particle will bounce around forever). The response vector can simply be added to the particle's velocity vector.

Collision detection Collision detection is another area which was glossed over in the paper. Collisions with the environment and other mesh-based objects are straightforward. Each particle independently checks whether it is colliding with another object, and numerous techniques are available for determining the point of impact (for example there are equations to calculate the distance between a point and a plane, among many others).

While the paper states that there are no particle-to-particle interactions within an object, the particles of two separate objects *do* need to somehow react to each other. One possibility is that particles of separate objects simply detect collisions between themselves, but because an object is not represented by a surface, there are many empty spaces where collisions cannot be detected (see figure 2.2).

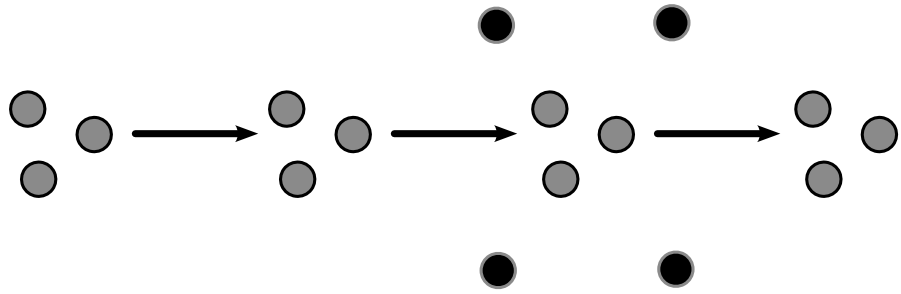

Figure 2.2: Because shapes are represented by particles, rather than surfaces, they can easily pass through each other. In this example, the small, brighter object passes right through the larger dark object without colliding with any particles.

There are ways to mitigate this problem. One such way is to add repulsive forces between particles so that when particles become close they push each other away. However this can give the unrealistic effect whereby two objects respond to each other without making

any contact, however it does make it more difficult for a particle to travel "inside" another object. Another possible solution is to simply add more particles, in effect simulating a surface (see figure 2.3). Combining these two approaches, particularly having particles throughout the volume of the object which repulse the particles in another object, makes it very difficult for two objects to intersect each other.

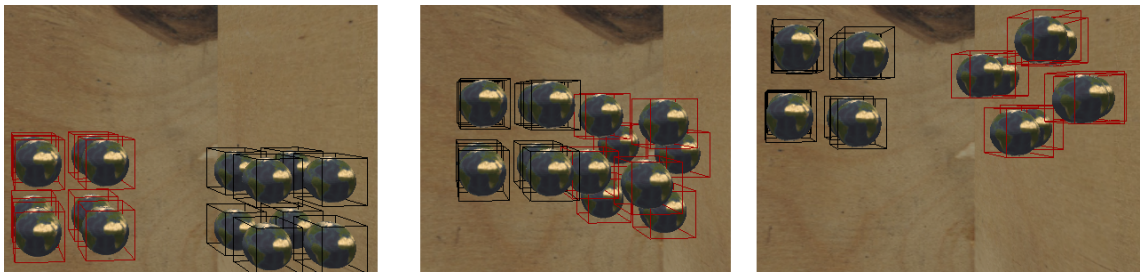

Figure 2.3: An example of collision detection working only on the particles of the objects. The particles are large and bunched closely together so that when the two objects meet, they do not intersect.

A problem with reducing the likelihood of intersection, is that it also reduces the likelihood that two objects that *do* manage to intersect each other can untangle themselves.

Simple particle-to-particle collisions are not satisfactory when dealing with object-to-object collisions. In the example simulations shown in the paper (for example where hundreds of shoes successfully deform each other without intersecting each other) the objects are loaded as meshes, and a subset of the vertices become the particles used in the shape matching section. Conventional collision detection techniques can then be used.

While loading objects as a mesh and using the mesh for collision detection solves the above problems, it makes irrelevant one of the supposed advantages of meshless deformations, which is that no connectivity information is needed between the points (i.e. no mesh is needed). It would seem that this claim is only true when there is only a single deformable object interacting with non-deformable objects[3].

Collisions and gravity overview

- Shape matching involves moving a particle from its current, deformed position to its undeformed position
- Objects only get deformed when forces are applied to them
- Forces and collision detection/response are handled independently of the shape matching

[3]The usefulness of not requiring connectivity information is possibly moot anyway due to the fact that some kind of mesh is needed in order to texture map the object.

- Forces are applied to single particles, rather than the object as a whole, by changing the velocities of the particles
- Because a particles system does not represent a continuous surface, collisions between objects can only be properly handled when each object has a mesh defined

2.2 Integration methods

The position x, velocity v and acceleration a are all dependent on each other by being different derivatives ($x''(t) = v'(t) = a(t)$) in time (see section 2.1.1). We need a method to numerically calculate the integration from one to another. There is no analytical solution due to fact that the acceleration at each time step t is only known at runtime. The acceleration consists of different possible forces. For example there are gravity, shape matching and drag forces.

2.2.1 Explicit Euler Integration

The explicit Euler integration is a linear integration scheme which is based on only one sample point $f(x)$ plus its difference $\triangle h f'(x)$.

$$f(t + \triangle h) = f(x) + \triangle h f'(x)$$

Example: The integration scheme for the velocity of a spring system is based on the acceleration force $f = -k(x(t) - l_0)$. The system is fixed at the origin and the other point with mass m is free. l_0 is the length of the spring and its resting state. k is the spring constant. The acceleration $a(t)$ is the first derivative of the velocity $v(t)$ with respect to time t. This gives

$$v(t + \triangle h) = v(x) + \triangle h a(x)$$

since the acceleration is $a = f/m$. The explicit Euler integration for the velocity is:

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

See [Weisstein, 2005] for Reference.

2.2.2 Implicit Euler Integration

Another approach is the implicit Euler integration. The term looks similar to the explicit integration but takes the difference of one step ahead $\triangle h f'(x + \triangle h)$ to integrate. This cannot be explicitly solved and usually leads to linear equations.

$$f(t + \triangle h) = f(x) + \triangle h f'(x + \triangle h)$$

The *Example* for this equation is

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

for the velocity. [Wikipedia, 2005]

2.2.3 Explicit vs. Implicit Integration

To compare both integration methods we take the example function $f(x) = x^2$. The derivative is $f'(x) = 2x = 2\sqrt{f(x)}$ this leads to the following equations:

- Explicit Integration:
$$f(x + \triangle h) = f(x) + \triangle 2\sqrt{f(x)}$$
- Implicit Integration:
$$f(x + \triangle h) = f(x) + \triangle 2\sqrt{f(x + \triangle h)}$$
$$0 = f(x + \triangle h)^2 - 4(\triangle h)^2 f(x + \triangle h) + f(x)^2$$
This is then solved to
$$x_{1,2} = 2(\triangle h)^2 \pm \sqrt{4(\triangle h)^4 - f(x)^2}$$
The term underneath the square root can not be negative because we are integrating in the positive direction.

Figure 2.4 shows the comparison of both integration methods.

The calculation of the Implicit Integration is more complex than the Explicit scheme but provides more accurate results in general. This evaluation scheme was described in Kesselheim [2005]

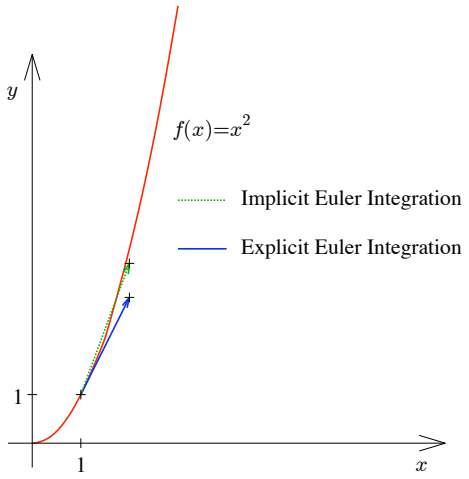

Figure 2.4: Comparing Euler Integration Methods

2.3 Stability of Euler integration

Stability is a key issue for 3D animation and modelling. It is not acceptable for animated models to behave physically incorrectly if parameters of it are changed. The Euler integration is known to be stable just under certain circumstances (see [Weisstein, 2005]) so it is not in general suitable for integrating from the applied forces to the position of an object.

This section describes the flaws of the standard Euler integration with an example where it can be seen that this integration method is not suitable for modelling animations that have to follow Newton's laws.

The example that is used is based on the example of the paper [Müller et al., 2005] but tries to elaborate more on the instability of the Euler integration for a spring. The physical system that is used is a simplified mass-spring system. Figure 2.5 shows the relevant parameters to perform the calculation. The spring is classified by a resting length l_0 and a spring constant k. Physically the spring connects two points. One of the points is fixed and the other point is free and connected with the mass m.

The free point $x(t)$ is always pulled towards the equilibrium l_0 of the spring.

For further calculations this is simplified and the resting length is set to 0 and the block is initially not moving ($v(0) = 0$). The described example is an extended version of [Kesselheim, 2005] and is based on the example found in [Müller et al., 2005, Section 3.1].

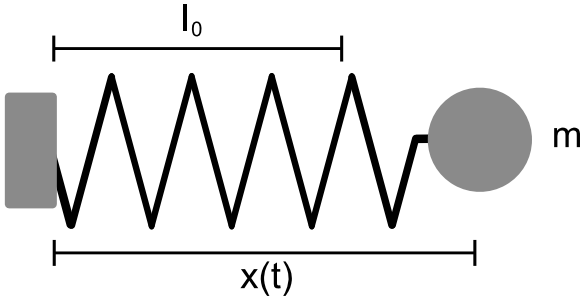

Figure 2.5: Overview of the mass spring system

2.3.1 Force calculation

The force $F(t)$ that is applied to the spring is proportional to $x(t)$. By adding the spring constant to the equation the force can be described as

$$F(t) = -k \cdot x(t)$$

Based on Newton's second law of motion (see section 2.1.1) the acceleration for the system can be determined.

$$a(t) = \frac{-k \cdot (x(t))}{m}$$

With $a(t) = x''(t)$ (see section 2.1.1) we get the differential equation

$$x''(t) = \frac{-k \cdot (x(t))}{m}$$

This ordinary differential equation can be solved analytically[4] and leads to the following equation for the position of x.[5]

$$x(t) = x_0 \cdot cos\left(\sqrt{\frac{k}{m}} \cdot t\right)$$

So for this physical system it is possible to calculate the current position for an arbitrary point in time when the initial position is given.

[4]The steps are described at http://www.myphysicslab.com/spring1_analytic.html

[5]For the solving of differential equations that are derived from Newton's laws the paper "Ordinary Differential Equations (ODEs)" at http://chaos.swarthmore.edu/courses/phys6_2004/QM/ODEs.pdf provides a good overview.

2.3.2 Euler integration

In order to compare the exact result to the Euler integration a modified Euler integration scheme is applied. The modification states that for integrating the position the velocity of the next time step is used instead of the current velocity ($v(t+h)$ instead of $v(h)$).

$$\begin{aligned} v(t+h) &= v(t) + h \cdot a(t) = v(t) + h \cdot \frac{-k \cdot x(t)}{m} \\ x(t+h) &= x(t) + h \cdot v(t+h) \end{aligned}$$

2.3.3 Example calculation

In this example two different parameters for the time step are used to perform the numerical integration. Both calculations are based on the same spring system (i.e. spring constant $k = 1$, mass $m = 1$ and original point $x(0) = 10$).

The first calculation is done with time step $h = 0.1$. Here it can be seen in table 2.1 and in figure 2.6 that the Euler integration is approximating the real values quite accurately.

t	0.1	0.1	0.2	0.3	0.4	0.5	0.6
correct $x(t)$	10.00	9.95	9.80	9.55	9.21	8.78	8.25
calculated $x(t)$	10.00	9.90	9.70	9.40	9.01	8.53	7.97
calculated $v(t)$	0.00	−1.00	−1.99	−2.96	−3.90	−4.80	−5.66

Table 2.1: Example integration with $h = 0.1$

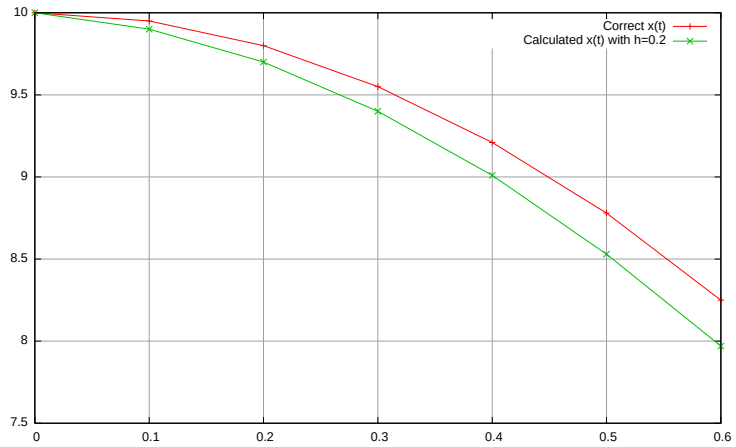

Figure 2.6: Differences in positions for $h = 0.1$

For the second calculation the time step is set to $h = 2.0$. The calculations (table 2.2 and figure 2.7) show that the Euler integration explodes and calculates values far away from the correct ones.

t	0.00	2.0	4.0	6.0	8.0	10.0	12.0
correct $x(t)$	10.00	-4.16	-6.54	9.60	-1.46	-8.39	8.44
calculated $x(t)$	10.00	-30.00	50.00	-70.00	90.00	-110.00	130.00
calculated $v(t)$	0.00	-20.00	40.00	-60.00	80.00	-100.00	120.00

Table 2.2: Example integration with $h = 2.0$

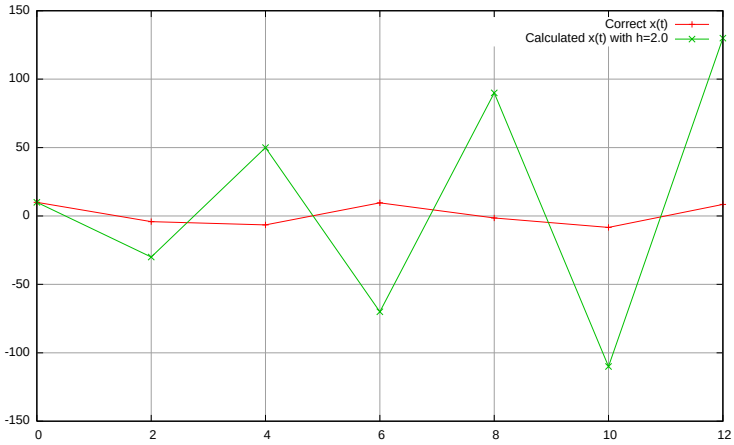

Figure 2.7: Differences in positions for $h = 2.0$

The overshooting can be especially seen for the first calculated value of v. The value is here bigger than the maximum that the real velocity will ever reach. The maximum velocity is the point where $x(t) = 0$. This point is reached at $t = \frac{\pi}{2}$ with a velocity of $(v(\frac{\pi}{2}) = x'(\frac{\pi}{2}) = -10)$. So the energy of the system increases which is physically incorrect.

2.4 Stable integration scheme

The proposed algorithm extends the explicit numerical integration by introducing shape matching. The shape matching step allows a more exact calculation of the actual destination for an animation step. Another advantage is that it does not overshoot like the method shown in section 2.3.

2.4.1 Shape Matching

The idea for the calculation of the target points is based on a multiple-step calculation. A graphical overview on the idea can be seen in figure 2.8.

- An object is divided in different particles that are not linked to each other. Then the physical simulation based on Newton's laws are performed and calculated.

- The positions for the particles are now taken and the algorithm tries to match the shape of the new object with the shape of the original object.
- Based on the difference to the original shape the velocity of the particle is altered and the object tries to regain its original shape.

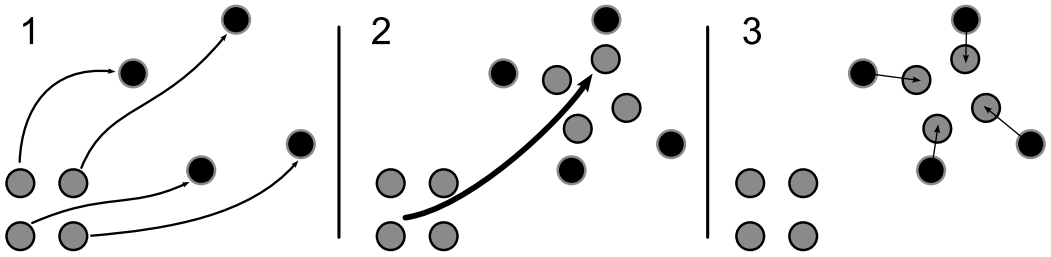

Figure 2.8: Idea of shape matching of particles

In order to solve the problem it is formulated mathematically. Therefore the basic idea of shape matching is restricted to two translations and a rotation. The restrictions can be easily explained, by translating the original object to the zero-point, rotating it and then translating it to the destination point (see figure 2.9).

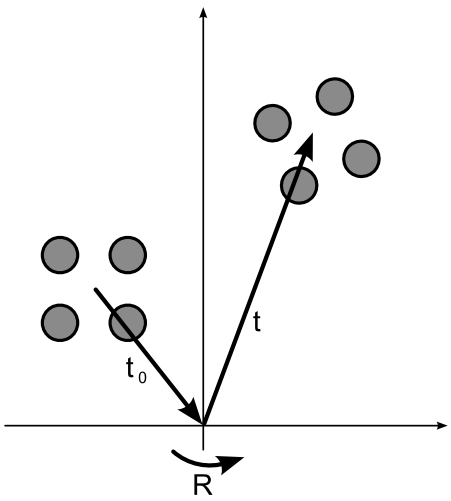

Figure 2.9: Shape matching (rotation, translations)

For the given set of original points ($\mathbf{x}_n^0$) and the set of deformed points ($\mathbf{x}_n$) of the restricted problem we can now formulate an optimization problem. We try to find a rotation matrix $\mathbf{R}$ and two translation vectors $\mathbf{t}$ and $\mathbf{t}_0$ which minimise the following equation:

$$\sum_i w_i \left(\mathbf{R}\left(\mathbf{x}_i^0 - \mathbf{t}_0\right) + \mathbf{t} - \mathbf{x}_i\right)^2$$

w_i allows one to give the different particles different importance. The natural choice for w_i is to take the mass m_i of a particle so the formula that will be used later is reformed to $\sum_i m_i \left(\mathbf{R}\left(\mathbf{x}_i^0 - \mathbf{t}_0\right) + \mathbf{t} - \mathbf{x}_i\right)^2$

In order to solve the optimisation problem we start with finding the translation vectors. For finding the translation vectors the partial derivatives with respect to $\mathbf{t}$ and $\mathbf{t}_0$ are calculated. Due to the fact that we look for the minimum the solution for the two vectors have to give the result 0 when they are inserted in the first derivatives.

Therefore we get the solutions

$$\mathbf{t}_0 = \frac{\sum_i m_i \mathbf{x}_i^0}{\sum_i m_i} = \mathbf{x}_{cm}^0 \text{ and } \mathbf{t} = \frac{\sum_i m_i \mathbf{x}_i}{\sum_i m_i} = \mathbf{x}_{cm}$$

Physically this is the logical choice because the object will be rotated around its center of mass.

In order to minimize $\mathbf{R}$ the solutions for $\mathbf{t}$ and $\mathbf{t}_0$ are put back into the equation. This can be seen as a rotation around the relative coordinates $\mathbf{q}_i$ and $\mathbf{p}_i$, with

$$\begin{aligned} \mathbf{q}_i &= \mathbf{x}_i^0 - \mathbf{x}_{cm}^0 \\ \mathbf{p}_i &= \mathbf{x}_i - \mathbf{x}_{cm} \end{aligned}$$

As a result we now try to minimize

$$\sum_i m_i \left(\mathbf{R}\mathbf{q}_i - \mathbf{p}_i\right)^2$$

Finding the solution to this problem is simplified. Instead of just allowing a rotation $\mathbf{R}$ an arbitrary linear transformation $\mathbf{A}$ is allowed.

$$\sum_i m_i \left(\mathbf{A}\mathbf{q}_i - \mathbf{p}_i\right)^2$$

For finding the solution the same approach is used that has been used for finding the translation vectors. So the derivatives for the entries of the matrix $\mathbf{A}$ are set to 0.

The solution for this is

$$\mathbf{A} = \left(\sum_i m_i \mathbf{p}_i \mathbf{q}_i^T\right)\left(\sum_i m_i \mathbf{q}_i \mathbf{q}_i^T\right)^{-1}$$

The rotation matrix $\mathbf{R}$ can now be found by polar decomposition. Every invertable matrix $\mathbf{A}$ can be decomposed to an orthogonal matrix $\mathbf{R}$ and a symmetric positive definite matrix $\mathbf{S}$ so that $\mathbf{A} = \mathbf{R}\mathbf{S}$. The following applies

$$\begin{aligned} \mathbf{S} &= \sqrt{\mathbf{A}^T\mathbf{A}} \\ \mathbf{R} &= \mathbf{A}\mathbf{S}^{-1} \end{aligned}$$

$\sqrt{\mathbf{A}^T\mathbf{A}}$ can be calculated with the help of diagonalization with Jacobi rotations.

After having calculated the rotation matrix $\mathbf{R}$ and the translation vectors $\mathbf{t}$ and $\mathbf{t}_0$ the goal positions $\mathbf{g}_i$ can be calculated (see figure 2.9)

$$\mathbf{g}_i = \mathbf{R}\left(\mathbf{x}_i^0 - \mathbf{x}_{cm}^0\right) + \mathbf{x}_{cm}$$

The matrix calculations in the implementation use the library developed at [Davies, 2005].

2.4.2 Extended Euler integration

Based on the calculated goal position the Euler integration scheme is altered. An additional velocity is added that changes the velocity so that the shape tries to regain its original shape (see figure 2.10).

$$\begin{aligned}\mathbf{v}_i(t+h) &= \mathbf{v}_i(t) + \alpha \cdot \frac{\mathbf{g}_i(t) - \mathbf{x}_i(t)}{h} + h \cdot \frac{\mathbf{f_{ext}}}{m_i} \\ \mathbf{x}_i(t+h) &= \mathbf{x}_i(t) + h \cdot \mathbf{v}_i(t+h)\end{aligned}$$

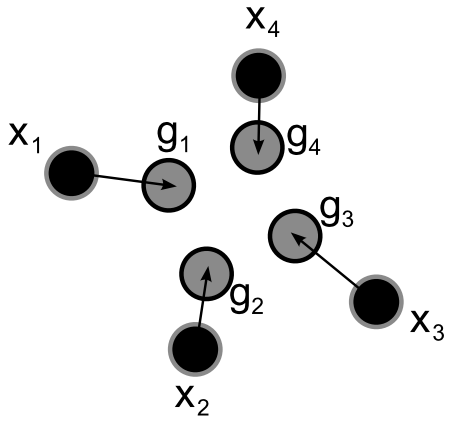

Figure 2.10: Integration extension based on goal positions.

2.5 Extended shape matching

With the "basic" shape matching explained above, we can only allow for translations and rotations. The authors give several techniques to extend the shape matching algorithm, three of which will be explored in this section.

2.5.1 Rigid body dynamics

Rigid bodies, or non-deformable objects, can be simulated by simply setting the value for α to 1. This value controls how quickly a particle reaches its goal position, where setting it to a small value will cause it to slowly return, while a larger value will speed it up. The maximum value allowed is 1, which means each particle will instantly reach its respective goal position, i.e. the object is never deformed, so it acts like a rigid body.

2.5.2 Linear deformations

Linear deformations allow the shape to scale and shear while they undergo deformation. In the basic shape matching, only the rotation is taken from the best matching linear transformation, while this extension involves calculating the entire transformation. Along with translation and rotation, a linear transformation can also express scaling and shearing.

Because the goal positions are calculated using the transformation, the introduction of scaling and shearing means that the shape created by the shape matching algorithm will be deformed from the original shape. In other words, the shape will never return to its original shape. To control the amount of influence the linear transformation has on the shape matching, the parameter β is used, and so the equation to calculate the goal position for a particle becomes:

$$\mathbf{g}_i = (\beta\mathbf{A} + (1-\beta)\,\mathbf{R})\left(\mathbf{x}_i^0 - \mathbf{x}_{cm}^0\right) + \mathbf{x}_{cm}$$

By using a smaller value for β, the ratio of rotation-to-pure deformation will increase, and so the goal positions will more closely match the original, undeformed shape of the object.

2.5.3 Quadratic deformations

The linear transformation is described using a 3 by 3 matrix. A transformation using a matrix of this size is restricted to only translations, rotations, scaling and shearing, however there are more kinds of deformations an object can undergo, such as twisting and bending. By moving from a linear to a quadratic transformation, these extra deformations can be simulated.

As with the linear transformation, once again the equation to calculate the goal positions needs to be updated. This time, the transformation matrices are a lot larger (3 by 9) in order to allow greater freedom of movement, and once again the β parameter is used to control the amount of deformation.

Figure 2.11 shows the different deformations techniques based on the same particle system. For all pictures the β-value has been set to 0.8. In the first picture the deformation method (rigid body) tries to approximate the deformed shape by rotating the original shape. For the second picture linear deformation was used. Here it can be seen that the shape is approximated better by the goal positions but still not close enough to the deformed object. For the last example quadratic deformation was used and here it can be seen that with this method the deformed shape can be approximated very closely.

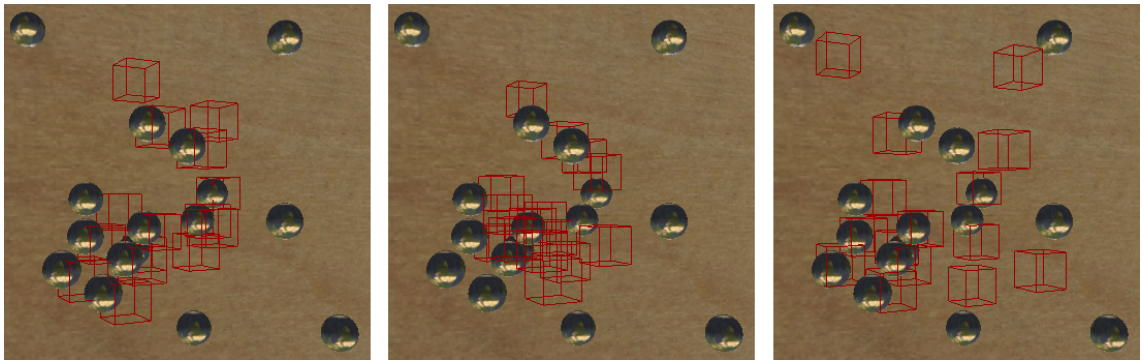

Figure 2.11: Comparison of the calculation of goal positions, where the spheres are particles in their current positions, and the red cubes show the goal positions

2.6 Plasticity

Imagine that you have a 30 cm metal ruler, that when bent returns to its original, flat shape. You are able to bend this ruler because it is elastic, and your bending is equivalent to the forces as described in section 2.1.3, while the return to the original shape corresponds to the shape matching problem.

Now imagine bending the ruler again, this time applying so much force that the ruler does not return to its original shape. This is the idea behind plasticity: when the amount of force exceeds some threshold, the shape of the object will be permanently changed. An object's propensity to undergo a permanent change in shape is referred to as that object's *plasticity*.

In a linear deformation, the linear transformation is made up of a rotational matrix multiplied by another matrix $\mathbf{S}$. In other words, $\mathbf{S}$ is the deformation that takes place before the particles are rotated, and because a rotated object is not deformed (for example turning a book on its side does not change the book's shape), we are only interested in $\mathbf{S}$, which represents the actual deformation.

The idea is that an object will keep track of the permanent change in shape in the matrix $\mathbf{S}^p$, this matrix transforming the original position of each particle relative to the original centre of mass. This matrix is initialised with the identity matrix, and of course when

you multiply a vector by the identity matrix, you are left with the original vector, and so to begin with there is no change in the shape due to plasticity.

The matrix $\mathbf{S}^p$ is only updated when the squared determinant of $\mathbf{S} - \mathbf{I}$ exceeds some threshold c_{yield}. A small value for c_{yield} means the object will undergo plastic deformations with only a small force applied; a large value will mean a lot of force will be required.

Once it has been determined that plastic deformation should take place, the matrix $\mathbf{S}^p$ needs to be updated. If we simply set $\mathbf{S}^p$ to be $\mathbf{S}$ then all deformations would be permanent. However, we probably only want to update it by a small amount, hence we multiply it by the parameter c_{creep}, and by multiplying it by the time step it means we change the shape no more than to where the actual particles have moved to at the current time. Because we only want to change it a small amount, the new value of $\mathbf{S}^p$ should be near to the identity matrix, but if c_{creep} and the time step are small, then the new matrix will be close to $\mathbf{0}$. Therefore, we subtract the identity matrix from $\mathbf{S}$, multiply this by c_{creep} and the time step, and then add the identity matrix back. Finally, we multiply this result with the previous value of $\mathbf{S}^p$, as plasticity accumulates over time.

By adjusting the values of c_{yield} and c_{creep}, we are able to control the plasticity of an object. For example, to simulate the bendy metal ruler used in the example above, we would set c_{yield} to be a relatively high value so that only a large force could permanently change its shape, and c_{creep} to a small value, so that when the shape changed, it would only change a small amount relative to the amount that the ruler was bent. Conversely, to simulate a piece of wire which would stay in the shape that it was bent to, c_{yield} would be close to zero so that even a small amount of force altered its shape, and c_{creep} would be close to one so that its shape would be permanently changed to very closely match the shape it was bent in.

3 Our Problems

3.1 Integration scheme

Unfortunately the integration scheme the authors present is not working in exactly the way they describe. We had to modify our integration scheme into two parts so they would be stable. The derived formulae term letters are related to the ones in the paper:

$$\mathbf{v}(t+h) = \mathbf{v}(t) + \frac{\mathbf{g}(t) - \bar{\mathbf{x}}(t)}{h} + h\mathbf{f}_{ext}(t)/m_i$$

$$\bar{\mathbf{x}}(t) = \mathbf{x}(t) + h\bar{\mathbf{v}}(t), \qquad \bar{\mathbf{v}}(t) = \mathbf{v}(t) + h\mathbf{f}_{ext}(t)/m_i$$

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\mathbf{v}(t+h)$$

$\bar{\mathbf{x}}$ and $\bar{\mathbf{v}}$ are intermediate calculations which are used to calculate the goal positions $\mathbf{g}_i$. The formulae for these are assimilated accordingly: $\mathbf{p}_i = \bar{\mathbf{x}}_i - \mathbf{x}_{cm}, \mathbf{x}_{cm} = \frac{\sum_i m_i \bar{\mathbf{x}}_i}{\sum_i m_i}$.

$$\mathbf{g}_i = \mathbf{R}\mathbf{q}_i + \mathbf{x}_{cm}$$

The authors used the same identifier for calculating the goal positions and decribing the integration scheme. This is basically only a flaw in their notation and does not violate their mathematical proves of stability.

3.2 Rotation matrix implementation

The mathematics behind the calculation for the rotation matrix to match the goal positions with the actual shape is not easily understandable. We understand roughly that they try to find a optimal linear transformation matrix with least square optimization. After this step the square root of $A_{pq}^T A_{pq}$ is used to extract the rotational part of A_{pq}. This part of the calculation is not understood by our group.

3.3 Frame rate-independent velocity updates

The authors state that the velocity update as they presented it[1] depends on the time step taken. They state that this can be solved by making $\alpha = h/\tau$ where $\tau \leq h$ is a "time constant". Aside from the fact that this would cause α to be greater than one, which is not desirable, this would cause the value of α to not be controllable in the program. It is also unclear what kind of value this constant should be.

3.4 Quadratic deformation of goal points

In section 4.3 of the paper the author introduces quadratic deformation of the goal points. This way the goal points can be matched along cubic curves. The object then deforms along these curves. It is unclear how the 9x3 transformation Matrix along with the 9 element vector $\tilde{q}$ is achieving this goal.

3.5 Plasticity

It is understandable that an object would experience plastic deformations when the force applied exceeds a certain threshold, but it is unclear exactly why it is the squared determinant of $\mathbf{S} - \mathbf{I}$ that is used[2]. While the determinant gives information about the volume of the object, and hence this equation is probably a measurement of the change in volume, it is unclear to us exactly how this works. Similarly, it is unclear why dividing $\mathbf{S}^p$ by the cube root of its determinant conserves its volume.

[1] Equation 9 in their paper [Müller et al., 2005]
[2] Equation 15, Section 4.5 in the SIGGRAPH paper [Müller et al., 2005]

4 Their evaluation of the algorithm

The evaluation the authors give can be divided into stability and performance.

4.1 Stability

The unconditional stability of this technique was an important improvement over previous techniques. To illustrate the success of this aspect, a completely squashed duck reforms to its original shape. While in the duck's case the technique appears to be stable, it is hardly a convincing proof of stability in all situations.

Fortunately, earlier in the paper we are shown a mathematical proof which convincingly proves the stability.

4.2 Performance

The performance is another very important section because the ability to use this technique in interactive applications was one of its main strengths. Consequently, the authors provide a more detailed analysis of the performance.

They supplied several different tests, for example animating 100 objects each with 100 particles (i.e. 10,000 particles altogether) and they reported that it ran at a respectable 50 frames per second. An important finding in those tests is that the time complexity is linear in respect to the number of particles used. More interesting is when they included collision detection and more objects. In this example, over 55,000 particles were used, and without collision detection the frame rate was around 25 frames per second, but with collision detection and response the scene could not be animated in real time (they do not give the frames per second achieved). This shows that the deformation calculations are more efficient than collision detection and response.

To illustrate that this technique is suitable for interactive applications, an example was shown where a human head with 66 particles could be manipulated with the mouse while handling collisions in real-time. They consider this situation "typical for games", however not many games consist of a single human head sitting inside a white box. They do not give the amount of processing power used in the example which would be useful

to see how much computation time would be required if, for example, in a game which took place in a complex environment the main character was deformable.

Overall though, their evaluation clearly shows that the performance achieved in this technique is very good.

5 Flaws and limitations

5.1 Physical correctness

The authors specifically said that they had games in mind when developing this technique. A common mantra in games is that correctness - whether it be in graphics, artificial intelligence, or physics etc - can be compromised as long as it appears correct. It would seem that this technique follows that rule, with its aim to be visually pleasing rather than physically correct. The benefit is of course efficiency, so this point is a limitation only if an exact physical representation is required.

5.2 Plasticity

Objects permanently change their shape only when the force applied to them exceed a certain value. Real objects also change their shape when a smaller force is applied for a long time (for example the indentations left on a sofa after sitting on it for a long time). The method given is unable to handle this type of plasticity.

5.3 Connectivity information

One of the advantages of this technique given was that an object could be a particle system where no connectivity information between particles is required. While technically this is true, without connectivity information, texture mapping and collision detection, and hence most useful applications for this technique, are at best problematic and at worst impossible.

Indeed, in their own examples it appears that they have loaded objects as a mesh and then used the vertices from that mesh to be the particles. This could be seen as an advantage though, because in graphics using meshes is very common and so this could be said to show how easy it is to extend a normal mesh to be a deformable object.

6 Impact of the paper

It is inevitable that in the competitive games industry, having deformable objects will become an important aspect of many games. Until now, the difficulty in implementation and the computational expense of deformations has stunted the use of these in games, and it may just be that the technique outlined in this paper will become very popular as a result of its ease of implementation and efficiency.

For these same reasons, deformable objects may be used in other applications, such as animations, where they were not used before.

Bibliography

[Davies 2005]
DAVIES, Robert: *Newmat C++ matrix library*. http://www.robertnz.net/nm_intro.htm. Version: 2005. – [Online; accessed 15-October-2005]. Newmat is a powerful matrix C++ library that allows to perform the necessary operations that the algorithm of the paper needs.

[Kesselheim 2005]
KESSELHEIM, Thomas: *Meshless Deformations Based on Shape Matching*. 2005. – A German presentation on the paper given at a seminar at 'Lehrstuhl für Informatik VIII, RWTH Aachen'. This presentation contains better explanations on the problems of Euler integration in comparison of solving the differential equation.

[Müller et al. 2005]
MÜLLER, Matthias ; HEIDELBERGER, Bruno ; TESCHNER, Matthias ; GROSS, Markus: Meshless deformations based on shape matching. In: *ACM Trans. Graph.* 24 (2005), Nr. 3, S. 471–478. – This is the original paper of SIGGRAPH 2005 and it is described in this report.. – ISSN 0730–0301

[Weisstein 2005]
WEISSTEIN, Eric W.: *Euler Forward Method*. http://mathworld.wolfram.com/EulerForwardMethod.html. Version: 2005. – [Online; accessed 15-October-2005] Mathworld contains a lot of useful mathematical background knowledge. For this report we had a look at the different Euler integration schemes.

[Wikipedia 2005]
WIKIPEDIA: *Implizites Euler-Verfahren*. http://de.wikipedia.org/wiki/Implizites_Euler-Verfahren. Version: 2005. – [Online; accessed 15-October-2005]