It's all particles

Yassine Mankai[1] and Mohamed Rached Waly[1]

[1]Telecom Paris, Institut Polytechnique de Paris, Paris

yassine.mankai@telecom-paris.fr
mohamed.waly@telecom-paris.fr

March 2022

1 Introduction

This project aims at implementing different features explored by Müller et al in the field of position based dynamics. The main idea behind their work is to create a unified solver to simulate the interaction of different types of 3D objects at once (rigid bodies, cloths, fluids..) given some environmental constraints.

In order to do so, they proposed to model each object with a set of basic particles. They opted for particles for their simplicity, versatility and specially for their ability to simulate a wide range of shapes. Constructing objects starting from particles reduces the number of collisions to process compared to complex algorithms for generating contacts between mesh based representations. Finally, the most important reason behind this idea is the fact that handling each particle separately induces parallelism and it is doable with the presence of modern graphics processors.

In this report, we are going to present how we approached the subject. We mainly focused on position based dynamics' pipeline, shape matching, cloth simulation and different types of collisions.

2 Implementation

2.1 Data representation

One integral part of the implementation phase was the choice of the data structures to manage the simulation. We decided to opt for what we found more representative of the core idea of these kind of systems. Shapes are internally a set of particles, forces and constraints are then applied on each particle. For that, our scene stores:
- a list of all the particles as a buffer called *all_particles* of a structure called *particle_element*
- a list of all the shapes as a buffer called *all_shapes* of a structure called *shape_structure*

2.1.1 Particle structure

We went for spherical particles where each particle structure store different attributes(see Fig.1).

```
Struct Particle contains
    vec3 position;
    vec3 velocity;
    vec3 allforces;
    float Mass;
    int phase;
    // indicates the index of shape containing the particle
    int nbrConstraints;
    // indicates whether we need to take in consideration a
       correction of speed (like friction  restitution)
end
```

Figure 1: Particle structure

2.1.2 Shape structure

Each particle is linked to a shape structure by its phase (index of the shape). The shape structure contains data needed for the calculation of the different shape-related constraints that will be applied on the correspondent particles. For now, we managed to implement 3 types of shapes(see Fig.2). For simplicity, we used a generic data structure that contains attributes for all different types of shapes. A memory optimisation could be achieved using a better software design with inheritance and polymorphism.

```
Struct Quadratic Deformation contains
    ShapeType type;
    vec3 com0;
    int nbParticles;
    vector<vec3> relativeLocations;
    vector<vec3> qQuad;
    mat39 optimalRotation;
    mat39 A;
    mat9 Aqq;
    mat3 Apq;
end
```

```
Struct Cloth contains
    ShapeType type;
    // enum type
    int height;
    int width;
    float bendLength0;
    // initial bending length
    float structLength0;
    // initial structure length
    float shearLength0;
    // initial shear length
    helpfull functions;
    // facilitates the acces of the neighbouring particles
end
```

```
Struct Linear Deformation contains
    ShapeType type; // enum type
     vec3 com0; // initialcenter of mass
     int nbParticles;
    vector<vec3> relativeLocations;
    // inital relative locations to the com0
    mat3 optimalRotation;
    mat3 A; // optimal linear transformation
    mat3 Aqq;
    mat3 Apq;
end
```

Figure 2: Different shape structures used in the implementation

2.2 The Simulation Loop

We tried to implement the modified position-based dynamics pipeline proposed in Macklin et al. (2014). For each time step, we do a symplectic integration for each particle in order to predict a target position x^*. At this level, we only include external forces that can't be translated into position constraints.

Next, we do a few stabilization iterations to avoid bad initial conditions. In this step, for each initial position x_i contradicting a contact constraint, we apply position variations to both the prediction x_i^* and the initial position x_i. This idea was discussed in the article and it aims at avoiding the contribution of the stabilization phase into the total energy of the system. We only include contact constraints as they're the most visible ones.

After that, we have the solver. Here, we solve for the prediction x_i^* all the constraints. In our case, we have: collisions with the environment, particle-particle collisions if they have different

phases, shape matching constraints for rigid and deformable shapes and mass spring systems for cloth shapes. For each constraint group (shape matching / contact/ springs) affecting a particle, we calculate an averaged position correction term $\frac{\Delta x_i}{n_i}$. During the implementation, we noted that to have good results, the average should be multiplied by a factor ω. This was evidently explained in the article with the need for over-relaxation. Finally, in a classic PBD manner, we update the velocity based on the change in position in the current frame. We also implemented the particle freezing step discussed in the article to avoid positional drift. *The details are explained in the algorithm 1 in the attached annex.*

2.3 Collision handling

We implemented two types of collision-based constraints for our particles: Particle-environment collision Particle-Particle collision. We tried to keep the pipeline as general as possible. The idea is to calculate. For each particle, we average over all the contact constraints a position correction term (dx_i) that should project the sphere outside the constrained domain.

2.3.1 Particle-environment collision

- Plane: $dx_i = (\langle x_i - point_{wall}, normal_{wall} \rangle + radius_{particle}) * normal_{wall}$

- Sphere: $dx_i = center_{sphere} + (radius_{sphere} + radius_{particle}) * \frac{x_i - center_{sphere}}{\|x_i - center_{sphere}\|} - x_i$

+ In both cases: $dx_i += (-\gamma * v_{normal} + \theta * v_{tangential} - v) * \frac{\Delta t}{NStabilization}$

2.3.2 Particle-Particle collision

To solve the collisions between particles, we had to search each step for the neighborhood of each particle. The brute force approach was computationally heavy. We opted for the simple solution of regular grids as an optimization data structure. The idea is to divide a space into fixed sized cells. Each cell stores the indices of particles in it. Then, to query a particle neighborhood, we look for its cell and the ones next to it. To have good results, we needed a cell size small enough to hold few particles but big enough to avoid multiple empty cells. A way to achieve such results was to generate a fixed resolution regular grid for each shape and to make the query iterate over each one of them. The choice of the resolution was made in order to have the potentially colliding particles only in directly neighboring cells (For the moment, it is hard coded but can be improved for general purpose). Access and updates are implemented using simple hash functions.
Having this range query implemented, we can detect interpenetration between any two particles. Projection constraints are implemented for any couple of particles having different phases. This should allow to simulate the interaction between two different objects in the scene for simple situations.

2.4 Shape matching

The shape matching presented in Müller et al. (2005) can be stated as the following , find the optimal rotation R and translation t to transform the points x_i(current representation) as close

as possible to x_i^0 (initial representation).
The problem is formulated as follows: find R and t that minimizes:

$$\sum mi(R(x_i^0 - t_0) + t - x_i)^2$$

Optimal translation

the optimal translation is the translation between the center of mass of the initial shape and the current center of mass.

Optimal rotation

Centering the initial and the actual points lets us determine only the optimal linear transformation A and not worry about the translation.
The formulation of the problem becomes: finding A that minimizes:

$$(\sum m_i p_i q_i^T)(\sum m_i q_i q_i^T)^{-1} = A_{pq} A_{qq}$$

A_{qq} terms contains only scaling, we will be focusing only on A_{pq}. Extracting the rotation part of this matrix consists in applying a polar decomposition $A_{pq} = RS$ where $S = \sqrt{A_{pq}^T A_{pq}}$
The target points are then calculated doing:

$$g_i = R(x_i^0 - x_{cm}^0) + x_{cm}$$

In the paper Müller et al. (2005), they mentioned that this method can simulate only small deviations from the rigid body to extend the range of motion. To extend the range of motion they used the linear transformation matrix A to calculate the targets. They used not precisely A but A divided by $\sqrt[3]{det(A)}$ to ensure volume preservation ($det(A) = 1$)

The target in that case is:

$$g_i = (\beta A + (1 - \beta)R)(x_i^0 - x_{cm}^0) + x_{cm}$$

The details are explained in the algorithm 2 in the attached annex.

Calculating targets will enable us to calculate the x associated to the shape matching constraints which will be:
$\triangle_{shapeMatching} x_i = \alpha * (g_i - x_i)$ where α is a parameter which simulates stiffness [Müller et al. (2005)] and in $[0, 1]$

Quadratic variation

Linear transformations presented above can only represent shear and stretch. According to Müller et al. (2005), we can extend the version to a quadratic version where the initial relative location will encapsulate also quadratic variations of the initial relative location.
In this version the q_i is $[q_x, q_y, q_z, q_x^2, q_y^2, q_z^2, q_x q_y, q_y q_z, q_z q_x]$
The details are explained in the algorithm 3 in the attached annex.

2.5 Cloth

We model cloth as a purely elastic shape. We used a 2D mass spring system with structural, shearing and bending springs. This choice is due to the simplicity of the formulation of such system as distance constraints which fits well with the particle solver we used. The particles are sampled on an N*N grid and all have the same mass. The spring constraints are implemented as

position variations in the general shape constraints of the solver loop (algorithm 1). A distance constraint between two particles p_i and p_j is expressed for p_i as:

$$\Delta_{spring} x_i = k * (\|x_i - x_j\| - L) * normalize(x_j - x_i)$$

2.6 Implementation details

To simulate rigid shape, we used either the linear or quadratic version of the shape matching. To simulate deformable shape, we used the quadratic version of the shape matching for more satisfying results.
$\alpha \simeq 1$ and $\beta \simeq 0.5$ means we want to simulate a rigid body.
Low values of α and β means we want to simulate a deformable body.

3 Results

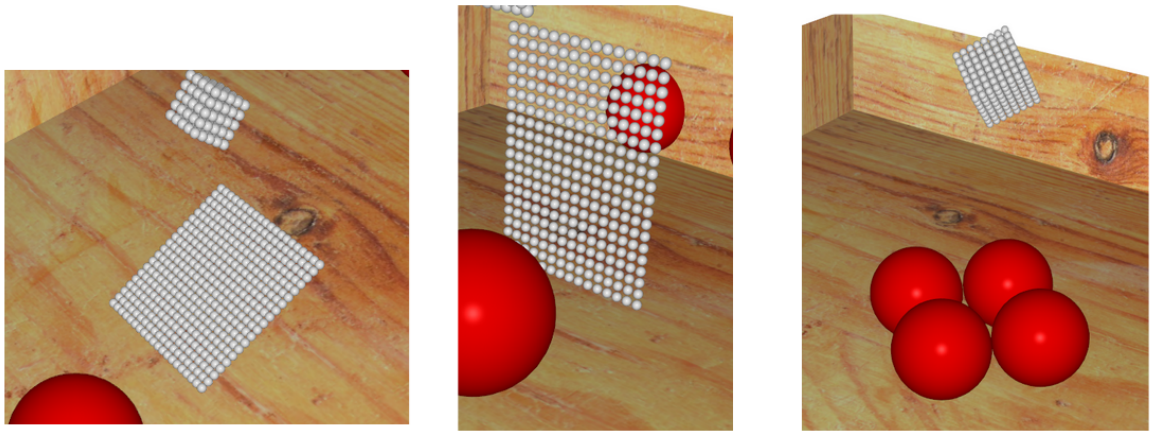

Figure 3: 3 scenes: Trampoline,Curtain,Shape Matching (left to right)

All the results we will be discussing below are accompanied with a video. We will be only discussing the context of each scene.
The red spheres present in the scene are environment's obstacles and not shapes made by shape matching. They are defined by the user's input.
We also provided some parameters to tweak in the graphical user interface.

3.1 Shape matching

The linear and quadratic shape matching are both well suited for rigid bodies. But if we would like to create a not fully rigid body, the use of the quadratic form of the shape matching is mandatory. To test the shape matching we created a scene where a cube drops to get squeezed between four spheres(environment obstacles).

3.2 Cloth

We wanted to test the constraints that we implemented for the cloth. We fixed all the boundary particles of the cloth which was oriented horizontally and we pushed an environment's obstacle to go through it.

3.3 Trampoline

We wanted to test the collision between shapes and the calculation of friction and restitution velocities. We created a scene where we fixed all the particles on the boundary of the cloth and tilted a bit its orientation. Then we dropped a cube on it to see the interaction between the two.
We noticed that the cube bounces on the trampoline, drops on the ground and stops moving.

3.4 Curtain

This is a second test of the collision between shapes in our scene and the impact of the stiffness of the cloth on the simulation.
We fixed the upper right and left particle of the cloth which was oriented vertically. We also reduced the spacing between particles comparing to the trampoline to have the correct behaviour. Then we moved some obstacles to bounce a shape on it in order for it to go through the curtain. We noticed that when the cube collides with an environment obstacle (red sphere) he is squashed but not when collided with the curtain.

4 Discussion

4.1 Friction and restitution

Taking in consideration friction and restitution plays a great part in the credibility of the scene. Without them, shapes won't bounce correctly when they hit an obstacle and will keep sliding on the floor. Collision between particles could also be improved with the use of a proper friction model. To solve this issue, we made a slight change in the pipeline presented in Macklin et al. (2014) (see 2.3). When handling contacts in the solver phase, we add a correction term to the position delta. This approach presents an issue: It is highly dependent on the scene we want to implement. Different friction and restitution models were discussed in other research papers. Some changes were also proposed on the PBD pipeline to incorporate such phenomenon.

4.2 Entanglement Effect

Even with considering friction and restitution, some times the shapes doesn't collide correctly. In our implementation, the collision between two particles happens along side the normal of the plane tangent to the sphere. This is the main to reason behind the entanglement effect. In fact Macklin et al. (2014) proposed an approach based on signed distance field within each shape to remediate this problem. The idea is to change the normals direction on each particle of the boundary of the shape. Because, in the case of a particle that penetrated another shape (let's

say fault of the time step of the simulation), this type of normals can help guide it to the outside.

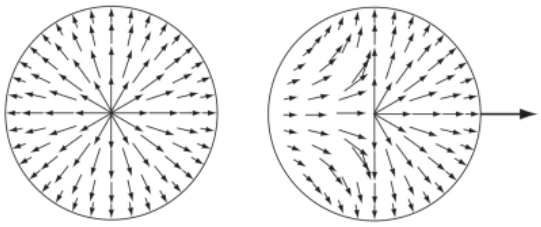

Figure 4: Sphere normals proposed in Macklin et al. (2014)

4.3 GPU parallelization

For now, we are processing the main loop and the calculations on the cpu side. This limits our number of shapes present in the scene, even now with only two shapes we have some drastic fps drop.
This approach will be fully exploitable if we passed the heavy calculation to the gpu. Being a system composed only by particles, we can process each one seperately on the gpu. This is can be a major improvement for our implementation and thus we can add multiple shapes at once.

4.4 Impact of variables

The simulation implemented is heavily dependant on the initial variables given by the user i.e number of iterations, time step, stabilization and solving iterations. We couldn't manage to find a general solution to this problem. So we add some presets to the simulation where the user can switch between predefined scenes to explore the features that we implemented.

References

Macklin, Miles et al. (July 2014). "Unified Particle Physics for Real-Time Applications". In: *ACM Trans. Graph.* 33.4. ISSN: 0730-0301. DOI: 10.1145/2601097.2601152.

Müller, Matthias et al. (July 2005). "Meshless Deformations Based on Shape Matching". In: *ACM Trans. Graph.* 24.3, pp. 471–478. ISSN: 0730-0301. DOI: 10.1145/1073204.1073216.

List of Figures

Algorithm 1 The simulation loop

initialize $NStep$,$NStabilization$,$Nsolver$,ϵ
for k 0 : $NStep$ **do**
    $\triangle t \Leftarrow \triangle t_0 / NStep$ ▷ $\triangle t_0$ user's input
    for all particles **do**
        compute external forces
        store previous positions ▷ denoted x_i
        apply forces $v_i \Leftarrow v_i + \triangle t f_{ext}(x_i)$
        predict positions $x_i^* \Leftarrow x_i + \triangle t v_i$

    (Update neighbours) ▷ uniform grid on all particles

    (Stabilization phase) ▷ to avoid bad initial conditions
    for k 0 : $NStabilization$ **do**
        (Solve Contact Constraints for x_i) ▷ n_i number of contact constraints for i
        evaluate dx_i and n_i ▷ dx_i accumulated over all the contact constraints
        for all affected particles **do**
            $x_i^* \Leftarrow x_i^* + \frac{dx_i}{n_i}$
            $x_i \Leftarrow x_i + \frac{dx_i}{n_i}$

    (Solver)
    for k 0 : $Nsolver$ **do**
        (Solve Shape Matching Constraints) ▷ check algorithm 2
        for all shapes with shape matching constraint **do**
            calculate optimal rotation R_i and linear transformation A_i
        for all particles affected **do**
            update $x_i^* \Leftarrow x_i^* + dx_i^*$

        (Solve General Shape Constraints)
        for all general shape constraints **do**
            evaluate dx_i^* and n_i^* ▷ n_i^* number of shape constraints for i
            for all affected particles **do**
                update $x_i^* \Leftarrow x_i^* + \frac{dx_i^*}{n_i^*}$

        (Solve Contact Constraints for x_i^*) ▷ n_i^* number of contact constraints for i
        evaluate dx_i^* and n_i^* ▷ dx_i^* accumulated over all the contact constraints
        for all affected particles **do**
            $x_i^* \Leftarrow x_i^* + \frac{dx_i^*}{n_i^*}$

    (adjust Velocities)
    for all particles **do**
        $v_i \Leftarrow \frac{1}{\triangle t}(x_i^* - x_i)$
        (particle sleeping)
        if $||x_i^* - x_i|| < \epsilon$ **then**
            $x_i^* \Leftarrow x_i$

Algorithm 2 Shape Matching

(Preprocessing part)
for all shapes with shape matching constraint **do**
    calculate relative locations $q_i \Leftarrow x_i^0 - x_{cm}^0$
    calculate the symmetric part $A_{qq} = \sum m_i q_i q_i^T$ and store A_{qq}^{-1}
(Inside the simulation loop)
calculate $A_{pq} = \sum m_i p_i q_i^T$
polar decomposition of $A_{pq} = RS$ with $S = \sqrt{A_{pq}^T A_{pq}}$
calculate $\bar{A} = \frac{A}{\sqrt[3]{det(A)}}$ with $A = A_{pq} A_{qq}$ ▷ volume preservation
calculate targets $g_i = \beta R q_i + (1-\beta)\bar{A} q_i + x_{cm}$
$x_i \Leftarrow x_i + \triangle x_i$ with $\triangle x_i = \alpha(g_i - x_i)$

Algorithm 3 Shape Matching Quadratic

(Preprocessing part)
for all shapes with shape matching quadratic constraint **do**
    calculate relative locations $q_i = [q_x, q_y, q_z, q_x^2, q_y^2, q_z^2, q_x q_y, q_y q_z, q_z q_x]$
    calculate the symmetric part $A_{qq} = \sum m_i q_i q_i^T$ and store A_{qq}^{-1}
(Inside the simulation loop)
calculate $A_{pq} = \sum m_i p_i q_i^T$
same polar decomposition of $A_{pq} = RS$ ▷ same rotation R of algorithm 2
calculate $\bar{A} = A_{pq} A_{qq} = [AQM]$
replace A by $\frac{A}{\sqrt{det(A)}}$ in $\bar{A}$
calculate targets $g_i = \beta \bar{R} q_i + (1-\beta)\bar{A} q_i + x_{cm}$ with $\bar{R} = [R00] \in R^{3X9}$
$x_i \Leftarrow x_i + \triangle x_i$ with $\triangle x_i = \alpha(g_i - x_i)$