Position Based Dynamics for Character Effects

Aina Nicolau Orell

Master Thesis

MSc Computer Animation and Visual Effects

Bournemouth University

NCCA

August, 2016

Contents

Abstract

This project presents the implementation of cloth, hair and soft bodies simulation using Position Based Dynamics (PBD). The report makes an overview of the mathematical and physical knowledge necessary to understand how PBD works as well as it shows other techniques that have also been used for character effects. Then, it gives a detailed explanation of the implementation process. Firstly, it shows how to manipulate the geometry to be able to use it for the simulation. After that, it describes the PBD solver, constraints and how that module is reused for the different effects. In addition, it discusses some ways to shader the results. The implementation has been done in Houdini using VEX language. Finally, it analyses the results and proposes some ways to improve it.

Keywords: Position Based Dynamics, Character Effects, Distance Constraint

Chapter 1

Introduction

Character effects are very important features to bring characters to life. There are a lot of advances in visual simulation for these effects, allowing the creation of life-like characters with clothing in high quality animation productions. However, simulating these kind of effects in a production environment is a challenging problem. For example, hair simulation is a challenging phenomenon because of the big amount of number of hairs on the head and the complexity of their movement. Soft bodies are also a very interesting topic to cover on character animation to represent the movement of deformable skin, for instance the belly and breast for humans or for the simulation of viscous animals or monsters.

Probably, the most popular approaches for dynamics simulation are forced-based. They built up external and internal forces and compute the acceleration by Newton's second law, using an integration method of time to update the velocities. In contrast to these methods, that are focused on accuracy, trying to create simulations as close as possible to real life, there are other ones more focused on stability, robustness and speed. The method used in this project is located in the second group that focuses onr efficiency remaining the results as visually plausible as possible. This method, called Position Based Dynamics (PBD), works directly on positions in order to have an easy manipulation of the simulation. These algorithms have gained a lot of attention in the past years, specially in computer games and interactive applications for its simplicity and stability.

The language chosen to programme this method is VEX language. This is a highly optimized geometry manipulation language. I decided to use it because it is very efficient due to its simplicity and for being a function based language. In addition, it is very powerful; it is possible to create nodes from the scratch and combine them to existing Houdini nodes. This allows an exact control over the desired points and the creation of costume nodes. Moreover, it has a very fast performance.

The main objectives of this project are listed bellow:

- Perform three different effects for characters: hair, cloth and soft bodies.
- Create a common PBD network for the different effects extensible for the improvement of each effect separately.
- Solve collisions for imported objects in order to test our results in real character models.
- Achieve a plausible result for both simulation and visual look level.

This project is structured in 6 sections. Chapter 2 provides an overview of the related work in the research literature, the firsts applications of the method and how it was formalized. Chapter 3 explains the mathematical background needed to understand the implementation process. It explains the algorithm and how to solve the specific constraints for the effects that we want to simulate. Chapter 4 shows the different steps that we follow to implement the project. This explanation is divided into the implementation of PBD itself and the process that the geometry needs to be ready for the simulation as well as the shading aspect. In Chapter 5 we show the results obtained with our implementation, making some observations and discussing some problems and improvements. Finally, Chapter 6 concludes the project and analyse some possible future work.

Chapter 2

Related Work

Dynamic objects have been traditionally simulated working with forces. Recently, some approaches work directly on the positions avoiding the problems about explicit integration that these force based systems have. The main advantages are the simplification of handling collision and the direct and immediate control of the animated scene. This new method is called Position Based Dynamics and was first introduced by Jackobsen (2001). He simulates a cloth system using constraints instead of springs. Using Verlet integration he manipulates directly the positions and keeps particles at a certain distance. In that moment, he did not explain how more general constraints could be handled.

There were more approaches that started to use constraints. Desbrun (1999) and Provot (1995) use constraint projection in mass spring systems to prevent springs from overstretching. Bridson et al (2002) use them to resolve collision. Clavet (2005) use constraints to simulate viscoelastic fluids.

Müller et al. (2007) formalize Position Based Dynamics. They present a framework that can handle general constraints formulated via constraint functions as well as a solver-algorithm that is independent of the type of constraints. He implemented different effects for instance cloth simulation. That solver could handle non-linear and inequality constraints but its convergence was slow. Müller (2008) proposed a multi-grid based process to speed up the convergence of PBD significantly while keeping the power of the method to process general non-linear constraints.

Based on these work, Bender et al. (2014) and Bender et al. (2015) provide some tutorials introducing the basic concept of the method, comparing different solvers and discussing approaches to improve them as well as some problems that the method presents. They show how we can use Position Based Dynamics to simulate different effects like hair, cloth, volumetric deformable bodies, rigid body systems and fluids. Macklin (2014) modify from Gaussian iterations to Jacobi-style iterations, meaning that each constraint is solved independently.

The fast, unconditionally stable and controllable algorithms that Position Based Dynamics produces are well established in games, but they are not widely used in film production. Steele (2014) wanted to reduce artist iteration time for simulate character effects such as ropes, chains and cloth for the film *How To Train Your Dragon 2* and he implement a new simulation pipeline for deformable solids inspired by PBD.

Müller (2007) uses stretching and bending constraints to simulate cloth, and defines some constraints for self collision and collision with solid objects. There are some approaches to handle inextensibility, a fundamental property of the cloth. Kim (2012) uses "Long Range Attachment" (LRA) method that applies unilateral distance constraint between free particles of the cloth to distant attachment point on the character, preventing them from stretching away from the attachments. For example, Macklin (2014) exposes these unilateral distance constraints and use them as long-range attachments. Another method is described by Müller (2012), who proposes a robust method for simulating hair and fur that guarantees inextensibility with a single iteration per frame. The technique is called "Follow The Leader" (FTL) and moves the particles such that all the constraints are satisfied but with the restriction that we are only allowed to iterate through all the particles once.

These kind of effects like cloth have been a topic of research since 1980's. We have said that the other alternative to simulate them are physically based models. Mass-spring systems are conceptually simpler and easier to implement than more physically consistent models derived from continuum mechanics using the finite element method. They are widely used for one and two-dimensional structures, such as hair and cloth. For hair, Iben (2013) proposes a hair model based upon an extensible elastic rod. Related to this, Umetani (2014) presents a new application of PBD to elastic rod simulation, which is essential for animating thin strands such as hair, fur, ropes, and so on. They introduce ghost points, which are additional points defined on edges, to naturally endow continuous material frames on discretized rods. Since PBD simulation is fast and robust, it can be used inside interactive modelling tools. They implement a PBD model with a simple collision model in a 3D modelling tool, to allow the user to create hair volumes. For soft bodies, Mesit (2010) presents previous work on deformable objects related to the methods. Some of the common approaches to soft body modelling include mass-spring systems, finite element methods, finite volume methods and finite difference methods.

Chapter 3

Technical Background

This section explains the theory necessary to understand how PBD works before the implementation step. It is based on the papers (Müller, 2007), (Macklin, 2014) and (Nicolau O., 2016).

3.1 Position Based Dynamics

Position Based Dynamics is a technique used for simulating dynamics that computes the position of a group of particles or vertex over time. Then, the velocity is updated as the difference of positions. These positions change affected for the internal and external forces. The advantage of PBD is the simplification of the computation of internal forces, like elasticity, pressure or viscosity, into constraints projected into the positions. Therefore, it computes the position and velocity taking only into consideration external forces for instance gravity and wind.

The new positions represent candidates for the final location of the particles in the time step. Then, to compute the final positions we need to apply the internal forces projecting some constraints into these candidates. The constraints used depend on the effect that we want to simulate. For example, the constraints used for grain materials are different from the constraints for cloth simulation. Some examples of constraints are fixed position, distance, friction, rigid and deformable bodies and fluid. There is a special case of constraint, the collision constraint, that needs to be computed for each time step such that it depends on the position of the particle regarding the environment in any moment. Finally, in the solver the positions are successively updated by being repeatedly projected for each constraint using Gauss-Seidel method.

Therefore, we present the dynamic object by:

- N particles:

  Each particle $i \in [1, ..., N]$ consists of:

  - a mass m_i
  - a position x_i

- a velocity v_i

- M constraints

  Each constraint $j \in [1, ..., M]$ consists of:

  - a cardinality n_j
  - a function C_j: $\mathbb{R}^{3nj} \rightarrow \mathbb{R}$
  - a set of indices $\{i_1,...i_{nj}\}$, $i_k \in [1, ..., N]$
  - a stiffness parameter $k_j \in [0, ..., 1]$
  - a type of either *equality* or *inequality*

  The constraints are represented as follows:

$$C(x_1, ..., x_n) = 0 \quad (1)$$

or

$$C(x_1, ..., x_n) \geq 0 \quad (2)$$

The first one corresponds to a constraint of the type equality whereas the second one to the type inequality.

The process that the PBD follows to simulate the dynamic object for each time step Δt is defined by the following pseudo-code proposed by (Müller, 2007):

```
(1)  forall vertices i
(2)     initialize x_i = x_i^0, v_i = v_i^0, w_i = 1/m_i
(3)  endfor
(4)  loop
(5)     forall vertices i do v_i ← v_i + Δt w_i f_ext(x_i)
(6)     dampVelocities(v_1, ..., v_N)
(7)     forall vertices i do p_i ← x_i + Δt v_i
(8)     forall vertices i do generateCollisionConstraints(x_i → p_i)
(9)     loop solverIterations times
(10)       projectConstraints(C_1, ..., C_{M+M_coll}, p_1, ..., p_N)
(11)    endloop
(12)    forall vertices i
(13)       v_i ← (p_i − x_i)/Δt
(14)       x_i ← p_i
(15)    endfor
(16)    velocityUpdate(v_1, ..., v_N)
(17) endloop
```

Figure 1 *Pseudo-code of the PBD algorithm (Müller, 2007)*

Giving an overview of the steps in the algorithm:

- Lines 1-3: the features of the particle *i* are initialized
- Line 4: Loop for the computation of the parameters
- Line 5: Computation of velocity for external forces
- Line 6: The velocities are damped if it is necessary
- Line 7: Computation of the candidate positions
- Line 8: Detection and generation of collision constraints
- Lines 9-11: The iterative solver manipulates the candidate positions in order to satisfy the constraints
- Lines 12-15: Update of the final position and velocity
- Line 16: Final velocity, depending on particle sleeping effect

(Macklin, 2014) made some changes to the initial algorithm. The main difference is that it uses a Jacobi iteration instead of Gauss-Seidel. That means that the candidate positions are not updated immediately, and rather than projecting one constraint after the other, all constraints are computed based on the same position.

Algorithm 1 Simulation Loop

1: **for all** particles i **do**
2: apply forces $\mathbf{v}_i \Leftarrow \mathbf{v}_i + \Delta t \mathbf{f}_{ext}(\mathbf{x}_i)$
3: predict position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
4: apply mass scaling $m_i^* = m_i e^{-kh(\mathbf{x}_i^*)}$
5: **end for**
6: **for all** particles i **do**
7: find neighboring particles $N_i(\mathbf{x}_i^*)$
8: find solid contacts
9: **end for**
10: **while** $iter < stabilizationIterations$ **do**
11: $\Delta\mathbf{x} \Leftarrow \mathbf{0}, n \Leftarrow 0$
12: solve contact constraints for $\Delta\mathbf{x}, n$
13: update $\mathbf{x}_i \Leftarrow \mathbf{x}_i + \Delta\mathbf{x}/n$
14: update $\mathbf{x}^* \Leftarrow \mathbf{x}^* + \Delta\mathbf{x}/n$
15: **end while**
16: **while** $iter < solverIterations$ **do**
17: **for each** constraint group G **do**
18: $\Delta\mathbf{x} \Leftarrow \mathbf{0}, n \Leftarrow 0$
19: solve all constraints in G for $\Delta\mathbf{x}, n$
20: update $\mathbf{x}^* \Leftarrow \mathbf{x}^* + \Delta\mathbf{x}/n$
21: **end for**
22: **end while**
23: **for all** particles i **do**
24: update velocity $\mathbf{v}_i \Leftarrow \frac{1}{\Delta t}(\mathbf{x}_i^* - \mathbf{x}_i)$
25: advect diffuse particles
26: apply internal forces $\mathbf{f}_{drag}, \mathbf{f}_{vort}$
27: update positions $\mathbf{x}_i \Leftarrow \mathbf{x}_i^*$ or apply sleeping
28: **end for**

Figure 2 *Pseudo-code of the PBD algorithm (Macklin, 2014)*

Other change presented in this second algorithm is the optimization of the processes finding the nearest neighbours to the current particle. These two commented changes are the changes that we also applied to our implementation.

3.1.1 Computing the Candidates

The algorithm starts computing the candidates' positions and the velocities, considering the external forces. We can relate these parameters using Newton's Second Law, that computes the acceleration with the equation

$$a = \frac{F}{m} \quad (3)$$

where a represents the acceleration, F the external forces that affect the particle and m the mass related to that particle. Then, we can find the velocity and accordingly the position, by the integration of the acceleration value.

3.1.2 The Solver

The aim of the solver is to recompute the candidate position in order to satisfy the constraints. The resulting system of equations is non-linear and to solve such a general set of equations and inequalities, we use a Jacobi iteration method.

The final correction Δx_i for each particle is computed as the accumulation of all the corrections of the position of i by constraint j (Δx_{ij}) and divided into the amount of constraints that affect i (n):

$$\Delta x_i = \frac{\sum_j \Delta x_{ij}}{n} \quad (4)$$

3.1.3 Constraint Projection

Having this set of candidate points, we have to move them depending on the constrictions. In these simulation, it is important to consider the conservation of linear and angular momentum. The method proposed for constraint projection conserves both momenta for internal constraints. It is based on a point-based approach such that we can directly use the function constraint C.

Consider a constraint with a cardinality n on the points $\mathbf{p}_1, ..., \mathbf{p}_n$ with constraint function C and stiffness k. $\mathbf{p}$ is the concatenation of $[p_1^T,\ldots, p_n^T]^T$. The gradient $\nabla_{\mathbf{p}}C$ is perpendicular to rigid body modes because it is the direction of maximal change.

We want to find the correction $\Delta\mathbf{p}$ satisfying $C(\mathbf{p} + \Delta\mathbf{p}) = 0$. An approximation to this equation is:

$$C(\boldsymbol{p} + \Delta\boldsymbol{p}) \approx C(\boldsymbol{p}) + \nabla_p C(\boldsymbol{p}) \cdot \Delta\boldsymbol{p} = 0 \qquad (5)$$

Then a solution $\Delta\mathbf{p}$ is found along the direction $\nabla_{\mathbf{p}}C$, reducing the number of unknowns in the equation to one Lagrange multiplier λ. Substituting that into the formula (5) we have:

$$\Delta\boldsymbol{p} = -\frac{C(\boldsymbol{p})}{|\nabla pC(\boldsymbol{p})|^2}\nabla pC(\boldsymbol{p}) \qquad (6)$$

Then, the correction for a particle i affected by the constraint:

$$\Delta x_i = -\lambda\omega_i \nabla_{x_i} C(\boldsymbol{p}) \qquad (7)$$

where ω_i is the inverse mass of particle i and

$$\lambda = \frac{C(\boldsymbol{p})}{\sum_j \left|\nabla_{x_i} C(\boldsymbol{p})\right|^2} \qquad (8)$$

After that, we have to consider the type and the stiffness k of the constraint. For the type equality we can always perform a projection. Otherwise, for the type inequality, the projection is only performed if $C(p_1,...,p_n) < 0$. We can incorporate the stiffness parameters in different ways but the simplest one is multiplying the corrections $\Delta\boldsymbol{p}$ by $k \in [0 \ldots 1]$.

3.1.4 Collision Detection

One advantage of PBD is the simplicity to compute collision avoidance. We have said that the collision is a special type of constraint because it is not static, it changes over the time, so we need to detect them in every simulation step.

We can differentiate two types of collision: particle-particle interaction and collisions with the environment. We can compute the first type comparing the distance of the particles to the sum of the two radii. The distance needs to be grater than or equal that sum to avoid the intersection. The constriction generated between particle p_1 and p_2 is the following:

$$C(p_1, p_2) = |p_1 - p_2| - (r_1 + r_2) \geq 0 \quad (9)$$

In the second type of collisions, we check if the particle collides with an infinite plane or a triangle when moving from its position to its candidate position. If the collision exists, an infinite plane constraint is generated, as this works for both cases. With the following constriction the particle is always in the same part of the plane:

$$C(p) = n^T p - d_{rest} \geq 0 \quad (10)$$

where x is the position of the colliding particle, n is the normal on the infinite plane and d_{rest} is the distance along that normal between the plane and the origin, including the radius of the particle.

3.2 Character Effects

The aim of this project is the creation of some character effects using different distance constraints, such as the simple distance constraint and unilateral distance constraint. Applying these constraints into different dimension levels we can achieve a variety of effects. Firstly, when we apply the constraints into 1D we obtain a string simulation, which represents the hair effect. Secondly, if we apply these constraints into 2D, for instance into a grid, we have the cloth effect, as a consequence of being a composition of stings on a plane. Thirdly, applying them into 3D we obtain soft bodies effect. The objects are deformed but they come back to their initial shape as a result of keeping these distances. In this section we make an overview of the necessary constraints used to represent every effect.

3.2.1 Constraints

Cloth is generally represented by a network of stretching and bending constraints along triangles of the grid mesh. We model stretching by the following distance constraint function:

$$C_{stretch}(p_1, p_2) = |p_1, p_2| - d \quad (11)$$

where d is the initial length of the edge. Therefore, what the correction performs is the difference of the actual distance and the initial length (see figure)

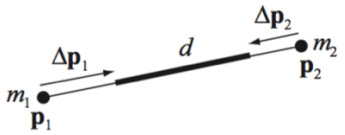


Figure 3 *Representation of distance constraint*

The bending constraint is defined by the following constraint function:

$$C_{bend}(p_1, p_2, p_3, p_4) = \text{acos}\left(\frac{(p_2 - p_1) \times (p_3 - p_1)}{|(p_2 - p_1) \times (p_3 - p_1)|} \cdot \frac{(p_2 - p_1) \times (p_4 - p_1)}{|(p_2 - p_1) \times (p_4 - p_1)|}\right) - \varphi_0 \quad (12)$$

where φ_0 is the initial dihedral angle between the two triangles.

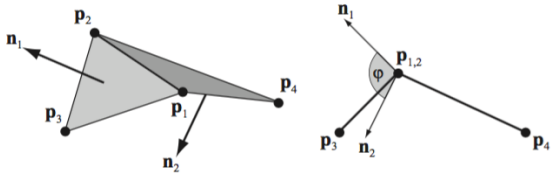


Figure 4 *Representation of bending constraint*

In our model we want to be based on distance constraints so we are not using bending constraints. However, to achieve more realism, we apply inextensibility for the reason that it is a fundamental property of cloth, specially for cloth that is attached to some character. To do that, Long Range Attachment (LRA) method is used. It applies unilateral distance constraints between free particles and attachments in order to prevent them to have a larger distance that the initial one.

The LRA method works as follows. Firstly, we compute the initial distance r_i from every free particle i to the attachment point. If in any moment of the simulation some particle is moving beyond this distance, it is projected back to the surface of a sphere centred at the attachment point with radius r_i.

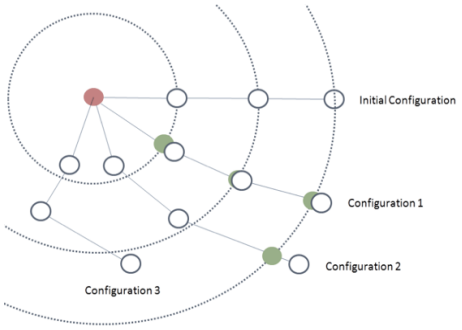


Figure 5 *LRA method : attachment point (red) - target positions (green)*

To include LRA in the simulation algorithm we perform the constraint projection step (Figure 6), which includes LRA as well as other local constraints.

Algorithm 1 Our Constraint Projection Step

for all constraints between particle p_i and p_j **do**
  apply local constraint projection to p_i and p_j
end for
for all unconstrained particle p_i **do**
  $d_i \leftarrow 0$
  $N \leftarrow$ number of LRA constraints assigned to p_i
  for all LRA constraint c_{ij} **do**
    apply constraint and compute displacement d_{ij}
    $d_i \leftarrow d_i + d_{ij}$
  end for
  if $N \neq 0$ **then**
    $p_i \leftarrow p_i + d_i/N$
  end if
end for
apply collision and other constraints

Figure 6 *LRA integration to the algorithm (Kim, 2012)*

Chapter 4

Implementation

This chapter covers how the techniques explained in the previous one are implemented. The implementation of this project is done in Houdini. We create a POP Solver from the scratch using VEX language to simulate PBD. There is already a Solver in Houdini to generate PBD effects, called POP Grains. It has been used as a reference to know how the results had to look. In spite of being also based on PBD, this Solver has a network too complex to reuse it, so we preferred to create a simple version of it. A good guide to start the implementation is the video (Magee, 2016), that explains how POP Grains works, talking a bit about PBD in Houdini.

We divide this explanation into three process steps, needed to achieve the final result from the scratch. These steps are represented in the following figure:

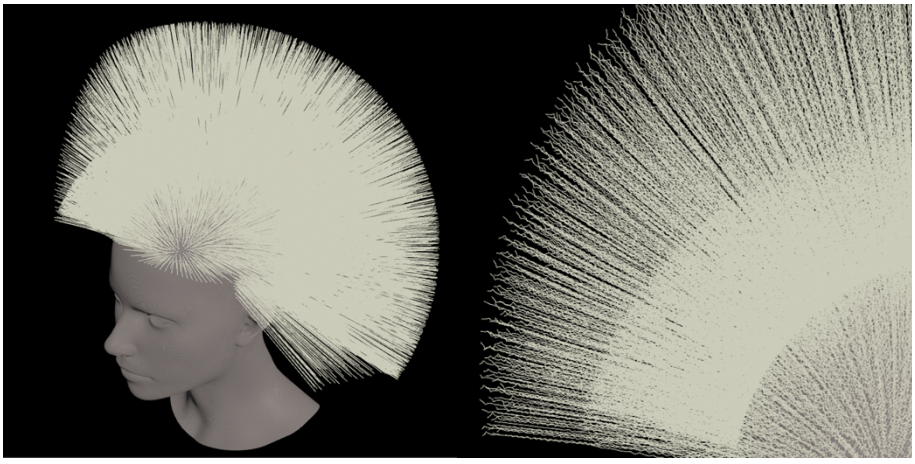
Figure 7 *Implementation Steps Diagram*

In the first step we are uncharged to create the necessary geometry to simulate. Before starting with the POP network, where the simulation is computed, we need to create the particles and the shape of our geometry. We create a geometry in the Object context and there, we can create any object to represent our effects. Then, we convert the model into a network of polylines that are able to be modified in the PBD network. Therefore, the program is ready to add the 'popnet' and simulate the created particles. So, in this second step, we move the points of the geometry such that the simulation is generated. Finally, in the step number 3, we create some shaders to achieve a more realistic result for the final simulation look.

4.1 Model Generation

Before performing the simulation, it is necessary to create the geometry and process it in order to be able to move its points and deform it. For this reason, we convert the geometry into a network of polylines that creates a lot of triangles connected by points. We have previously commented that this project generates different effects applying the same PBD network in different dimension levels. Therefore, the generation of the geometry is a bit different for each case.

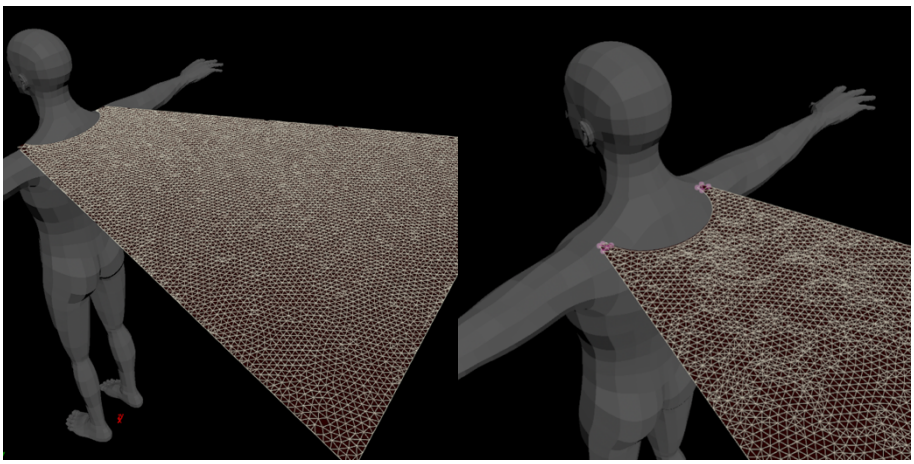
If we apply PBD to 1D, that is a line, we obtain the effect of a string of hair. To have the entire geometry of hair we only need to scatter points on the head surface and copy there that line. The best way to do it without having intersections is to the direction of the normal of the surface. Some things that we should take into consideration are the length, the number of points and the attachment of these strings. There is a relation between the length of the line and the number of points that it has. One of the constraints that we are applying to it is for self-collision, as a consequence the points has to be separated a specific distance range. If the distance between these points is not within the range, that results in deformations of the string, like zigzag effect (Figure 8 (b)). The other challenge is finding the points to attach. One way is making a group using different patters depending on the length and the number of points, in order to pick only the numbers that are supposed to be at the beginning of the strings. This method can be a bit tedious because it needed to be calculated for every geometry. A more proper solution is to add a vertex attribute to know when the string is changing. This attribute can be the same as the one used for the rendering. You can find a further explanation in 4.3 Shading section.



(a) (b)

Figure 8 *(a) Hair Geometry (b) Zigzag effect problem*

If we apply PBD to 2D geometry such as a grid, we obtain cloth effect. In this case we do not create the lines directly as we did with the hair. To create the network of lines we need to scatter points on the surface and then connect them creating explicit lines. We do that with a Wrangler node, using VEX language. This node that we create is based on Houdini's node 'Connect Adjacent Pieces'. Firstly, it defines a search radius and creates a vector with the nearest points to the current one. Then, it loops for each point of the vector. There, it has two conditions in order to not take either the current point or a point higher that the current one, because we are supposed to make the connection in only one direction. If that point satisfies these conditions, we create a polyline, and this point and the current one are defined as its vertices.



(a) (b)

Figure 9 *(a) Polyline Network on cloth (b) Attachment point selection*

Finally, if we apply it to 3D geometry we obtain soft bodies effect. It is not enough to follow the previous explanation changing the grid for a 3D object to create this geometry, because in this way we only obtain an empty piece of cloth with the object shape. To solve that, we need to scatter points also inside the surface, for this reason we use "Points from Volume" node. It is needed to apply before a "VDB from Polygon" to convert the geometry and also get rid of some unnecessary details of the object.

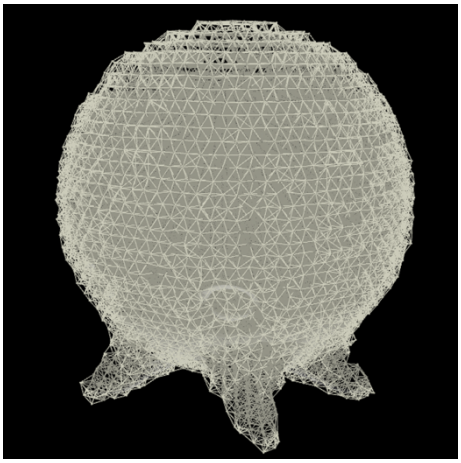


Figure 10 *3D Polyline Network for Soft Bodies*

As one of the main points of the project is to test the effects on character models, we need to import .obj geometry into the scene. We downloaded some examples from (TurboSquid, 2016) and (CgTrader. 2016) and converted them into alembic using Maya. When this geometry is imported to Houdini, it is important to convert it into 'Polygon' with a 'Convert' node to be able to use it correctly. It is also useful to subdivide the geometry to have better visualization and more precision in collision detection.

4.2 PBD Network

This is the most important part of the project as it is where the simulation takes place. The aim of this section is to implement the algorithm represented in Figure 2, that moves directly the positions of the model to achieve the desired effect. We implement that inside a POP network using VEX Language. We create seven main Geometry Wrangle nodes to implement the Position Based Dynamics method on these particles. Therefore, we represent each constraint in a new node. It is important to mention that some of these nodes that we have created, such as collision, have their version in Houdini nodes, but we have implemented them from scratch to use the exact PBD code that we explained in Chapter 3. The POP network is shared for all three character effects, but at the same time, it is extensible to fit better to each one. The following figure shows the final network to understand the composition of nodes. This node has two inputs: firstly, the polyline network that we have to deform and secondly, the collider geometry. It is important to add a 'normal' node to the collider before attach it to the PBD node.

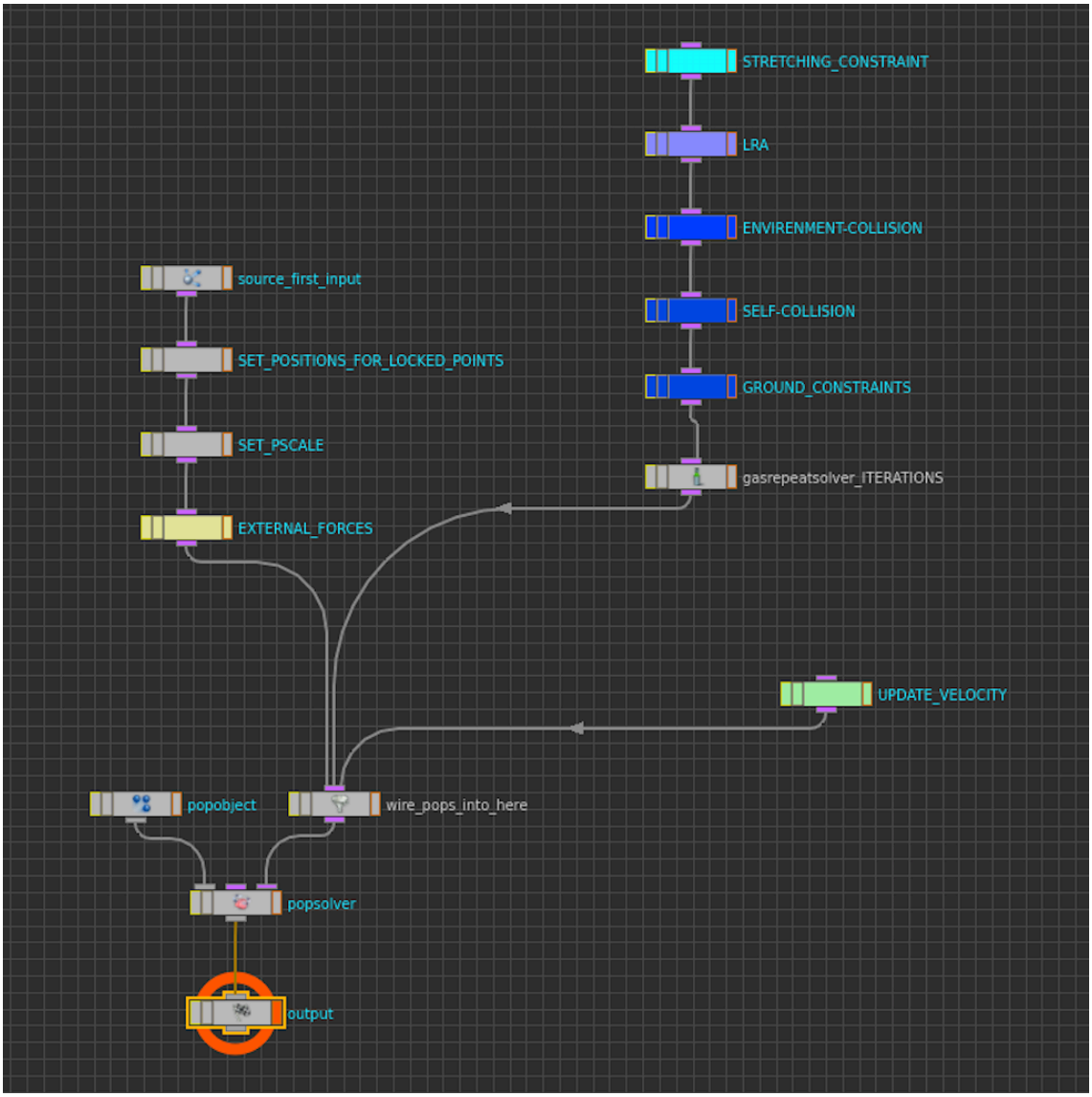


Figure 11 *PBD Houdini Network*

4.2.1 External Forces

As a first step on our simulation we compute the position of the particles depending on the external forces. In this case we only take into consideration the gravity force. To do that, we compute the velocity variable using the gravity acceleration multiplied by the time. Then, the position can be updated multiplying this velocity also by the time. This node that we have written has to be placed after the particle source and it makes the same function that the POP Force.

4.2.2 Stretch Constraint

This is the constraint that maintain the particles in the same geometry such that the model is deformed but not broken. It checks if the distance between two connected particles of the polyline network have increased over the time step, if the case is true, it applies a correction in order to come back to the initial distance. This correction is computed from the distance constraint function (11) (Müller, 2007).

The derivate with respect to the points are

$$\nabla_{p_1} C(p_1, p_2) = n \tag{13}$$

$$\nabla_{p_2} C(p_1, p_2) = -n \tag{14}$$

where

$$n = \frac{p_1 - p_2}{|p_1 - p_2|} \tag{15}$$

The inverse of the mass is *w* and the scaling factor *s* is

$$s = \frac{|p_1 - p_2| - d}{w_1 + w_2} \tag{16}$$

Therefore, we can compute the constraint correction

$$\Delta p_1 = -\frac{w_1}{w_1 + w_2}(|p_1 - p_2| - d)\frac{p_1 - p_2}{|p_1 - p_2|} \tag{17}$$

$$\Delta p_2 = +\frac{w_1}{w_1 + w_2}(|p_1 - p_2| - d)\frac{p_1 - p_2}{|p_1 - p_2|} \tag{18}$$

To apply this correction, we have to check for every point which primitives it belongs to. For each primitive we compare the position of the point to the position of the other point of the primitive in order to know the distance between them. This distance is compared to the initial one, and the error is computed. To move the particle, we have to do it in the direction of the line between both particles and the error represents the magnitude of that correction vector (Figure 3). We compute all the corrections for this point respect to all points which it is connected with. Then, we can compute the final correction with the average of all of them. After that, we update the position of the point adding to it the final correction.

4.2.3 Long Range Attachment

This is a specific constraint node that we only use for cloth because in the other effects it causes some problems. It is also based on distance constraint, in particular on unilateral distant constraint. The performance is not exactly as the previous one. To compute Long Range Attachments (LRA) we need to store data in three different vectors: one with the attachment points, one with the initial distances between every free point and every attachment point, and finally, the same distance as the vector before but for the current frame. Therefore, the first two vectors are static while the third one is changing every frame.

In the first frame we compute the initial radii, while in all the other frames we compute the current ones. This current distance is compared with the initial radius and if it is bigger, we apply a PBD correction:

$$\Delta p = \left((p_1 - p_2) - r_{initial}\right) \cdot \frac{(p_1 - p_2)}{|p_1 - p_2|} \quad (19)$$

As we can see, this correction is the same as the distance constraint, but we substitute *d* for the initial radius $r_{initial}$. The correction is computed for each attachment point, so the final displacement is the average of the different results.

There is the possibility to have some control over the stretching parameter inflating the value of initial radii (Kim, 2012). For this reason, we created a user parameter in the LRA node to change this value and have different stretching results easily (Parameter References in VEX Snippets - Houdini Tricks, 2016).

4.2.4 Environment Collision

This constraint is necessary for having interactions with the environment. It checks the distance between our model's particles and the environment collider. We define a research radius and move all the particles within this distance further away. In other words, we apply a new correction to the nearest particles. The correction is the following (see equation 10):

$$\Delta p = (p - q_c)\, n_c \cdot \frac{(p - q_c)}{|p - q_c|} \quad (20)$$

The final position can be updated adding the average of all corrections. To obtain good results, the points can not be too separated because the detection do not happen and there is collision. For this reason, the more points, the more precise, but we need to find a good compensation. If the results are no good enough with this method, it is possible to use another method based on triangles (Müller, 2007).

4.2.5 Self Collision

This type of collision is computed as particle-particle collision, like the equation (9). The particles have a uniform radius, so we check which of the pair of particles are in a distance lower than the double of the radius value. If there is the case, we move the particle along the normal collision the distance enough to have a separation of both radii. Therefore, the correction is

$$\Delta p = \big((p_1 - p_2) - d\big) \cdot \frac{(p_1 - p_2)}{|p_1 - p_2|} \quad (21)$$

where *d* represents the distance between the particles.

As before, we define a search radius to apply the correction to the nearest particles. Finally, the position is updated in the same way than the previous one.

4.2.6 Ground Collision

It is useful to add a constraint to avoid the collision of the ground. We use a simple method to place the particles always on the infinite plane of the ground. The positions of the particles are checked and if any of them is on a position under the plane, it is projected on the plane. We compare the position with the radius instead of 0 to not have intersection.

4.2.7 The Solver

More less all nodes follow the same process. There is a variable to store particle neighbours. With this vector we only need to check collisions between neighbour particles and save a lot of computation. After that, we create two variables that are needed for the Jacobi iteration: one to store the accumulation of correction displacement and the other one for the number of constraints that affect the particle in order to divide the accumulation. The iteration starts and the correction is saved for each loop. When it finishes, the position and velocity are updated. In addition, the damping is applied if necessary.

In addition to the the main nodes previously explained there are other simple nodes needed to have a correct result. The most important is node is the 'gasrepeatsolver'. This represents the loop of the algorithm itself. As we have said before, we compute collisions only for neighbours, so we have to propagate this movement to the rest of the particles, iterating the constraint system with this node. When we use more iterations the simulation is more stable. Another node that we have is needed when the input geometry is moving, for example when the head for the hair is rotating. The attachment points have to follow the movement of the geometry instead of staying at the same position. We also define the attribute *pscale* in the PBD node, even though it can be defined before. Finally, we can also add a node to compute the masses of the particles before the constraint projections if it is necessary.

4.3 Shading

The methods used for shading are different depending on the effect. Firstly, in the case of the hair, we can render the lines directly, because they are the geometry itself. The problem is that if we do not modify any parameter, the lines appear too thick and unrealistic. To correct that, we can add an attribute that the curves have for default called *width*. The difficulty is that we want to change the width from the root to the tip and repeat the same process for every string. For this reason, we create an attribute 'pct' (percentage of the curve) with the type 'vertex'. We need to do it as vertex type because these numbers are repeated over the strings, while the points are unique. The expression that we use to define the parameter 'pct' is $VTX/($NVTX-1). However, the width attribute needs to be defined as a point type, for this reason we promote this attribute from vertex to point. After that, we create an Attribute VOP to be able to modify the shape of the hair string using a ramp function. The network is shown in the Figure 12. There, we do a ramp to generate the shape and bind the result as the width attribute (Figure 13).

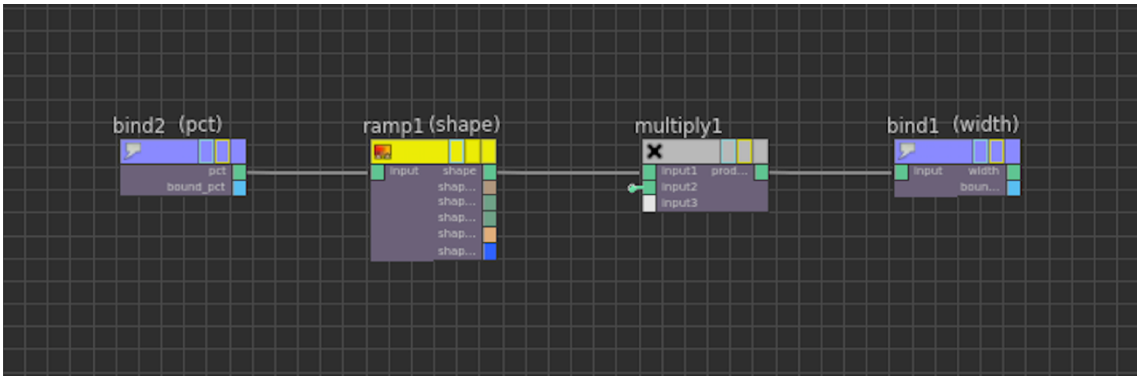


Figure 12 *Houdini VOP Network for Hair Shape*

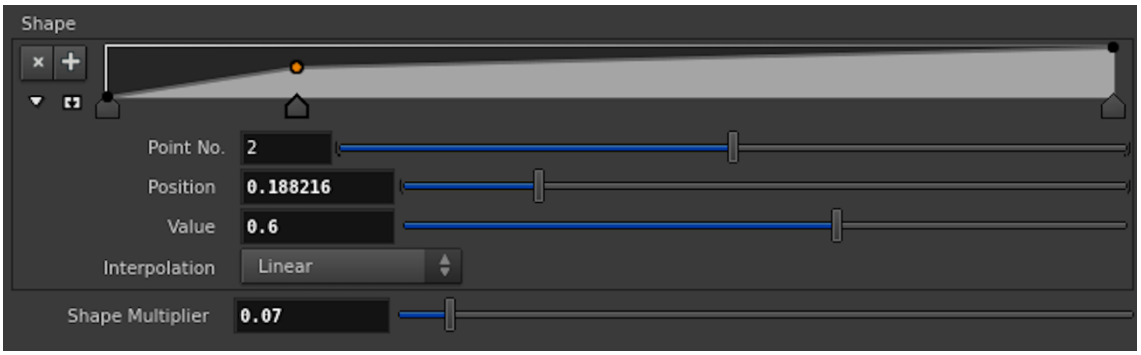


Figure 13 *Houdini VOP Network Interface for Hair Shape*

Finally, using the hair shader that Houdini has for default, we add colour and achieve a nice result. Secondly, for cloth simulation we need to recover the geometry that we lost creating the network of polylines. The method chosen has been using the "Point Deform" node, that combines the model with the simulation. Houdini has several proper cloth shaders. We decided to use the Velvet shader. Finally, soft bodies are rendered in the same way than the cloth.

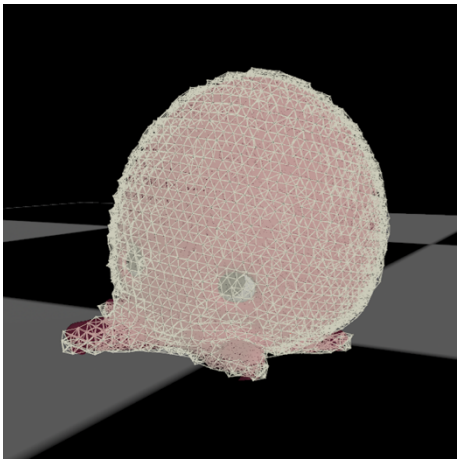


Figure 14 *Result of Point Deform Houdini node*

Chapter 5

Results

In this chapter we analyse the results generated in this project. We discuss the efficiency, performance, parameters and scenarios used for the three simulations: hair, cloth and soft bodies. We include several screenshots of the results in order to appreciate the shading part, but since the results are simulations, it is preferable to check the videos attached. The output of the project is a set of videos showing the results of the simulations on proper characters with the correspondent shading. The first tests were simulated in simpler scenarios so the result is not relevant compared to the complex ones. We discuss the number of particles used in each case and the time needed to render the specific amount of frames in the Table 1.

The first effect is the hair. We have three different scenarios. Firstly, we simulate a 7000 short hairs with 40 points each. Secondly, we combine long hair with that short hair. We use 5000 long strings with 100 points each plus 2000 short strings. In the following images there are the results of both cases. We can see how the constraints are working, specially the collisions with the head are very precise, there is collision even in the ear.

(a) (b)

Figure 15 *(a) Short Hair Simulation (b) Long Hair Simulation*

In the shading section we said that we are able to control the width of the hair. There are two examples of the hair width with different values.

(a) (b)

Figure 16 *Examples of different hair width*

The second effect is the cloth. We start with a simple scene with a squared piece of cloth attached on the corners. This simulation without the LRA node looks a bit unnatural. It is not very normal to see cloth with this level of elasticity. After applying LRA the movement is much more smooth. Thanks to the user parameter we can control the stretch value and obtain results in between both.

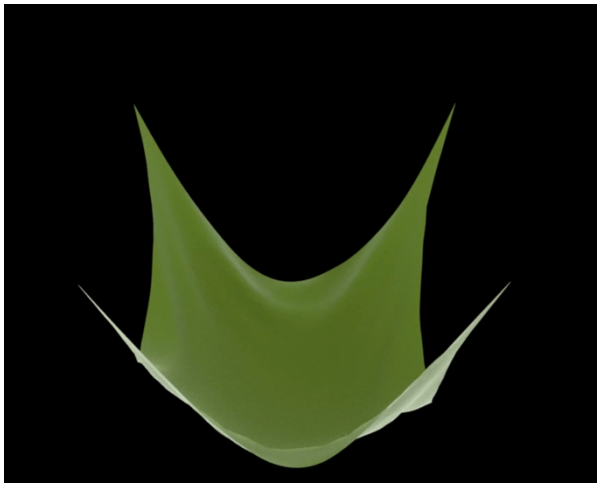
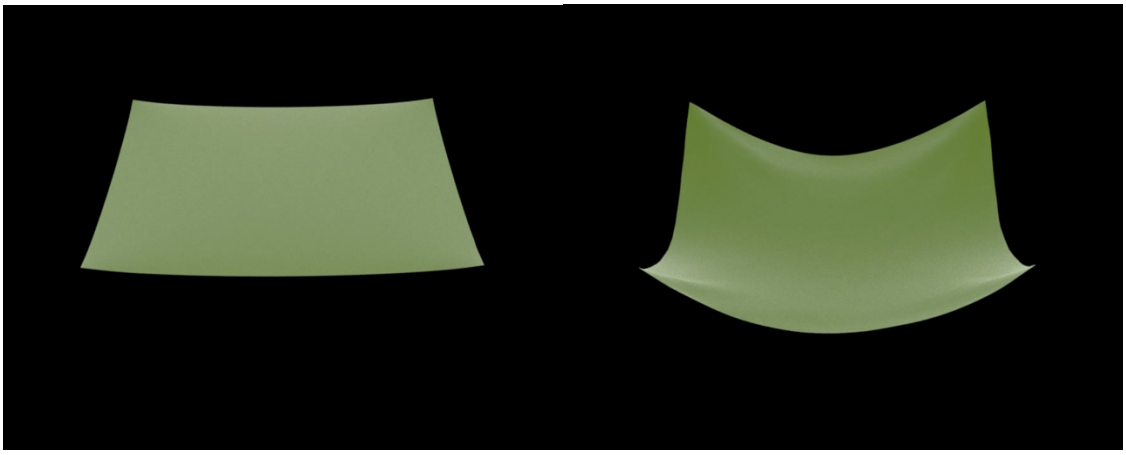


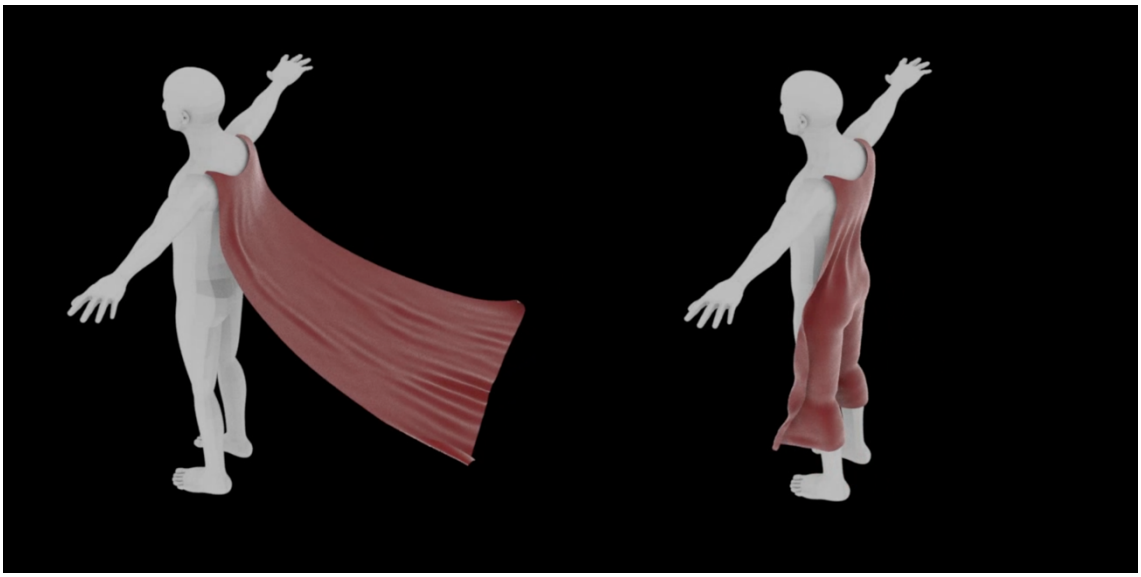
Figure 17 *Cloth Simulation without LRA*



(a) (b)

Figure 18 *(a) Cloth with LRA (b) Cloth with customised stretching*

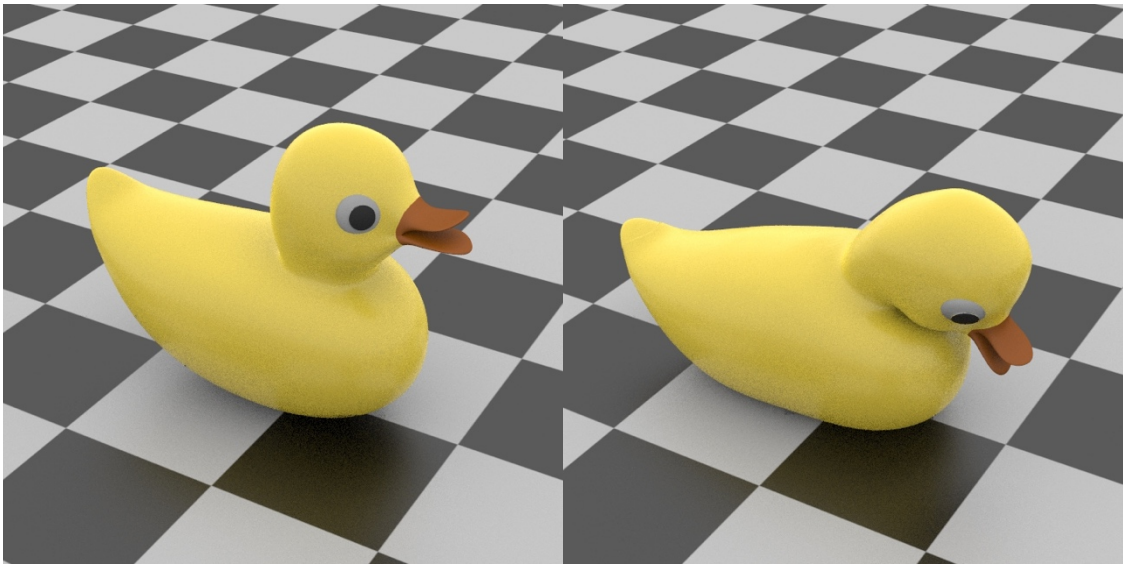
Moreover, having control over the stretch parameter, we solve collision problems. We compute the environment collision comparing the distance between the points of the cloth and the points of the collider. If the cloth is stretching, its points get away from each other, so the collision detection lose precision and fail. With LRA this problem is solved. The next scenario is useful to see that effect. It consists on a cape attached to a man model. Without LRA, the cape fells and its surface is increasing and reaches the floor, but with LRA the cape maintain its original size.



(a) (b)

Figure 19 *Frames of Cape Simulation*

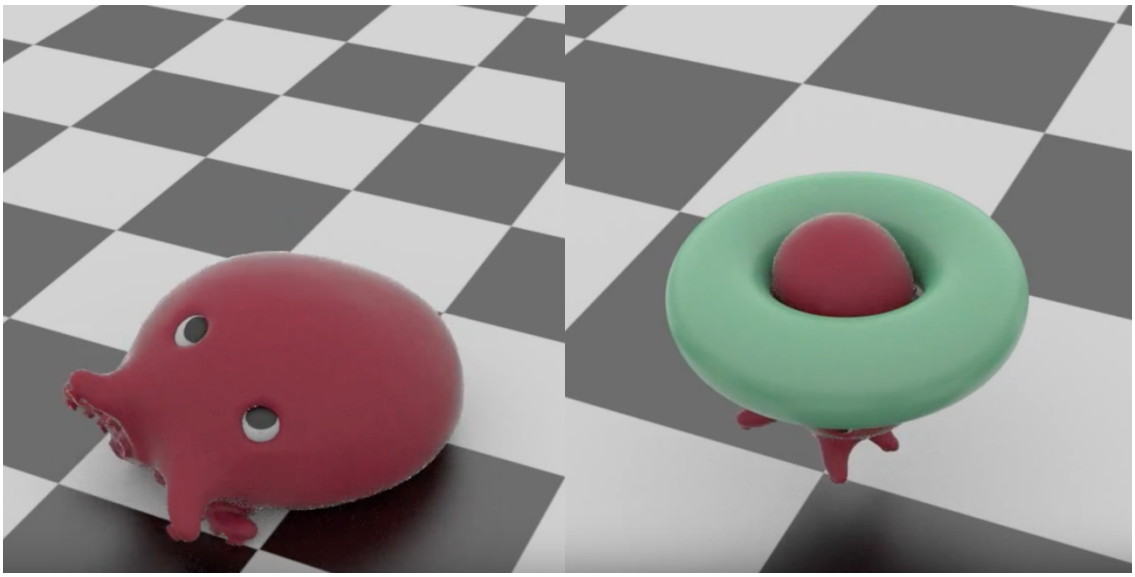
Finally, we have the soft bodies effect. For the first scenario we use a geometry of a duck. We can see how the shape is deformed for the force of the gravity and the compression of the floor. One problem that we can find in this simulation is the friction. When the object finished the deformation, it does not remain stopped on the floor.



(a) (b)

Figure 20 *(a) Duck Soft Body in the initial frame (b) Duck during the Simulation*

As this effect depends on the complexity of the geometry we test it on two different objects. In second scenario the object is an octopus falling down. Finally, we use this same object as well as a collider to see the reaction of the soft body.



(a) (b)

Figure 21 *(a) Octopus Simulation (b) Octopus Simulation with Collider*

	NAME	N. POINTS	FRAMES	RENDER TIME	S / FRAME
1	Short Hair	280.000	250	2h 14min	42
2	Long Hair	580.000	250	3h 21min	49
3	Long Hair Rotation	580.000	250	3h 15min	47
4	LRA 0 Cloth	3.000	100	15min	9
5	LRA 1 Cloth	3.000	100	12min	7
6	LRA 0.5 Cloth	3.000	100	15min	9
7	Cape Cloth	6.000	250	2h 3min	30
8	Duck Soft Body	34.000	250	2h 8min	30
9	Octopus Soft Body	20.000	100	35min	21
10	Octopus Collision Soft Body	20.000	175	1h 10min	24

Table 1 *Overview of Results*

Chapter 6

Conclusion

This section is a critical evaluation of the project. It discusses the difficulties that we had during the implementation. Moreover, it talks about some possible points to improve and the way to develop them. We had around three months to develop this project. We think that, in our case, the time has been organized correctly and the previous research in PBD during the Simulation Assignment was so useful to have a faster implementation and understanding of the problem.

The output of this project was a bit open because there were lots of things to add to it if there was time enough. We think that one of the most stressful aspects of this project was that we could not be focused on one thing, because we have three different effects. However, the main objectives were well defined and they have been achieved. We have been able to simulate three different effects for characters (hair, cloth and soft bodies) using a common PBD network based on distance constraints. It is useful to create a simple version to tackle the complex network of Houdini POP Grains. The effects can collide and interact with the environment and we have a proper shading as well as they have been tasted in character models.

We are quite satisfied with the results. On the one hand, we achieved all our objectives as well as we like the videos for both simulation and visual look aspects. In addition, we learned how to use Wrangler nodes and code with VEX Language, that we think it can be very useful in the industry.

On the other hand, there are still some details that we could not add or achieve for the project. The most important one is the simulation of curly hair. We wanted to use distance constraints to connect some points of the hair as a spring. We start the implementation but the results were not good enough to present. In addition, we wanted to add more user interaction. It has been difficult to modify easily some parameters of the simulation having plausible results. The problem was the dependency of the *pscale*, the amount of points and the dimensions. Therefore, we could relate these parameters with a ‘separation’ value that defines the distance between particles and have an easier control over the simulation.

References

ANON. 2016. Houdini Introduction to POP grains - Part 001. [online] YouTube. Available from: https://www.youtube.com/watch?v=pV6HJKM1XZo [Accessed 27 May 2016].

ANON. 2016. CIS563, 2015, Position Based Dynamics. [online] YouTube. Available from: https://www.youtube.com/watch?v=fH3VW9SaQ_c [Accessed 27 May 2016].

ANON. 2016. Houdini Scripting - Introduction to Vex Part 001. [online] YouTube. Available from: https://www.youtube.com/watch?v=SHWNYuD5tPU [Accessed 27 May 2016].

BENDER, J., MÜLLER, M., OTADUY, M., TESCHNER, M. AND MACKLIN, M. 2014. A Survey on Position-Based Simulation Methods in Computer Graphics. *Computer Graphics Forum*, 33 (6), 228-251.

BENDER, JAN, MATTHIAS MÜLLER, AND MILES MACKLIN. 2015 Position- based simulation methods in computer graphics. *EUROGRAPHICS Tutorial Notes*.

BARTELS, PIETERJAN. 2015. Position Based Dynamics.

BRIDSON, R., FEDKIW, R., & ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics (ToG)*,*21*(3), 594-603.

CGTRADER. 2016. *Cgtrader.com*. https://www.cgtrader.com/free-3d-models/character-people?keywords=body.

CLAVET, S., BEAUDOIN, P., & POULIN, P. 2005. Particle-based viscoelastic fluid simulation. In Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation (pp. 219-228). ACM.

DANESH, A. 2016. SideFX - Creating Grass Procedurally in Houdini Using Only Curves. *Lesterbanks*. http://lesterbanks.com/2013/04/creating-grass-procedurally-in-houdini-using-only-curves/.

DESBRUN, M., SCHRÖDER, P., & BARR, A. 1999. Interactive animation of structured deformable objects. In *Graphics Interface* (Vol. 99, No. 5, p. 10).

FREE 3D MODELS AND TEXTURES | TURBOSQUID.COM. 2016. *Turbosquid.com*. http://www.turbosquid.com/Search/Index.cfm?keyword=free+.

IBEN, H., MEYER, M., PETROVIC, L., SOARES, O., ANDERSON, J., & WITKIN, A. 2013. Artistic simulation of curly hair. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (pp. 63-71). ACM.

JAKOBSEN, T. 2001. Advanced character physics using the fysix engine.

KIM, T. Y., CHENTANEZ, N., & MÜLLER-FISCHER, M. 2012. Long range attachments-a method to simulate inextensible clothing in computer games. In*Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (pp. 305-310). Eurographics Association.

LIU, T., BARGTEIL, A. W., O'BRIEN, J. F., & KAVAN, L. 2013. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, *32*(6), 214.

MACKLIN, M., MÜLLER, M., CHENTANEZ, N. AND KIM, T. 2014. Unified particle physics for real-time applications. *TOG*, 33 (4), 1-12.

MESIT, J. 2010. *Modeling and Simulation of Soft Bodies* (Doctoral dissertation, University of Central Florida Orlando, Florida).

MÜLLER, M. 2008. Hierarchical position based dynamics.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., & GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Transactions on Graphics (TOG)*,*24*(3), 471-478.

MÜLLER, M., HEIDELBERGER, B., HENNIX, M. AND RATCLIFF, J. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18 (2), 109-118.

MÜLLER, M., KIM, T. Y., & CHENTANEZ, N. 2012. Fast Simulation of Inextensible Hair and Fur. *VRIPHYS*, *12*, 39-44.

NICOLAU O., A. 2016. PBD Sand Simulation. Unpublished.

PARAMETER REFERENCES IN VEX SNIPPETS - HOUDINI TRICKS. 2016. Houdini Tricks. http://houdinitricks.com/parameter-references-in-vex-snippets/.

PROVOT, X. 1995. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In *Graphics interface* (pp. 147-147). Canadian Information Processing Society.

STEELE, T., LEPREVOST, J. C., & GOSSART, K. 2014. A position-based dynamics system for animated character effects. In *ACM SIGGRAPH 2014 Talks* (p. 55). ACM.

UMETANI, N., SCHMIDT, R., & STAM, J. 2014. Position-based elastic rods. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (pp. 21-30). Eurographics Association.