# Real-Time 3D Position-Based Dynamics Simulation for Hydrographic Printing

Leonardo Thomas *, Karl Apaza Agüero †, Antonio Lopes Apolinário Jr ‡
Department of Computer Science
Federal University of Bahia (UFBA)
Bahia, Brazil
Email: *leothomas77@yahoo.com.br, †kaguero@ufba.br, ‡antonio.apolinario@ufba.br

***Abstract*—Hydrographic Printing, also called Hydrographics, is a viable method for coloring objects created with 3D printers. However, executing the hydrographic technique leads to a complex interaction between a thin film and a 3D printed object, in which the image in the film must adhere to the object. To handle the difficulty of predicting the final result of hydrographic printing, we propose a 3D computational simulation that uses Position-Based Dynamics, a popular technique for simulating deformable bodies and widely used in physics engines. We take advantage of this technique running in parallel a GPU-based simulation with suitable performance. We simulate the film behavior and its interaction with the printed object, as an interaction between a soft-body colliding with a rigid one. To evaluate the achieved performance consistency, we varied the number of vertices and voxels in the bodies involved and observed that the simulation kept running in real-time. We also execute the hydrographic technique in different printed models and compare these results with the simulated models.**

## I. Introduction

The 3D printing manufacturing process had a fast development in the last years. The advances of this technique influenced many areas such as product design, engineering, architecture, medicine, cultural heritage [1] [2] [3].

Unfortunately, despite the advances in complex shapes fabrication, creating colored objects by additive manufacturing is still difficult. Some high-end 3D printers address the problem of creating the shape and coloring the surface as a single task [4]. Often, this approach is undesirable because it forces the choice of a specific equipment model, leading to a budget that often exceeds the project expectations. Among the manufacturing process, when we search for a more accessible and cheaper way of coloring 3D printed objects, we often deal with two methods: the traditional painting and the decal techniques [5].

With the traditional painting, we have to deal with material loss, ink preparation, ink mixing, solvents, and surface preparation. Moreover, skilled people are essential to this technique. This approach needs an initial setup that requires a set of assets.

Decal techniques work with a substrate (a plastic or paper sheet) that carries a printed pattern or image. This substrate reacts with an activator (water, heat, or another reagent) to transfer the pattern to a surface through contact [6]. One advantage of this process is that we can apply it to a previously 3D printed object. Another advantage is the possibility of adoption in a large scale industrial production, with a semi-automatic process, minimal material loss, and a more comfortable way.

Among the decal techniques, Hydrographic Printing [7], also called Water Transfer Printing (WTP) or Hydrographics, is a known technique made to coloring 3D objects. It is viable to a wide variety of materials such as ceramic, fiberglass, plastic, wood, and metal. To perform this technique it is initially necessary to print a 2D image over a polyvinyl film. This film is put on a vat of water and kept floating over the water surface, as shown in Figure 1a. The film reacts with an activator solution and becomes a thin sheet that carries the ink of the previously printed 2D image. The object is deepened in the vat, and the colors are transferred when it touches the film. Figure 1b shows the final aspect of the 3D printed object decorated with a pattern by the hydrographics technique.

(a) (b)

Fig. 1. A 3D printed object covered with carbon fiber pattern by a hydrographic printing technique. (a) Hydrographic film floating. (b) 3D printed object colored with hydrographics.

With hydrographics, it is usual to transfer a repetitive pattern (like carbon fiber, camouflage, wood, stone, animal skin) to the object. Thus, there is no need for accurate alignment between the color pattern contained in the film and the object in the coloring process. However, when it is necessary to color specific areas of a piece printed in 3D, we have to provide an accurate mapping of the colors from the 2D image contained in the film to the surface of the 3D object. Performing this operation with the usual hydrographic printing is hard because there is no control of how this mapping will occur.

Our proposal aims to simulate virtually the hydrographic printing process in real-time, opening new possibilities. We believe that the most interactive process allows simulating the immersion with different positions quickly. This way, after simulations by varying the immersion positions, it will be possible to choose a more suitable way to use the hydrographics, by considering the use of the largest possible area of the film, keeping the stretch controlled. With the approach, it will be possible to reduce the common issues of hydrographics, avoiding material loss, and improving the prediction of the process. Also, unlike previous works [8], [9], that use a full physical model tailored with Finite Element Method (FEM) on the Central Processing Unit (CPU), our main contribution is to propose a real-time simulation on the Graphics Processing Unit (GPU), by adopting the Position-Based Dynamics (PBD) technique combined with a soft-body collision detection approach.

The organization of this paper is as follows: first, we present

related works (section 2), followed by the background knowledge that enabled our implementation (section 3), then we show assumptions, parameters, and definitions to the simulation made (section 4). After that, we describe the tests and compile results with the solution adopted (section 5), finally presenting the conclusions (section 6).

## II. Related Work

Zhang et al. [8] present an approach of computational simulation for hydrographics. The simulation proposed can predict the distortion of the film and the color variation during the hydrographic printing process. Furthermore, their work shows a method to automatize the physical process of the hydrographics partially by hardware that controls the immersion into the water. Finally, they propose a solution with multiple immersions to provide a suitable treatment that covers more complex concave objects.

Panozzo et al. [9] present relevant contributions to the same problem. Their work makes a detailed study of the physical conditions (like water temperature, the quantity of activator, dipping velocity) in which hydrographic printing occurs. This descriptive survey proposes a set of physical and computational tools that generate more accurate results with hydrographic printing. Their algorithmic solution treats the problem differently from the work of Zhang et al. [8]. Finally, they describe an algorithm to simulate the printing process and the film distortion when it adheres to the arbitrary 3D surface.

Regarding the implementation of computational simulation, Zhang et al. [8] proceed into the CPU. Their work addresses the problem to a 2D domain, representing the film. Thus, the simulation consists of calculating the discrete velocities and positions over the film. The 2D velocity field calculation simplified the problem. The simulation was enough for their needs. However, simulation examples took up to 5 minutes to perform a single immersion. This duration can be a very time-consuming process when an interactive simulation is necessary or in a real-time 3D simulation.

Panozzo et. al [9] implemented the simulation in the CPU. They adopted a generic solver for linear systems in which it is possible to run processes in parallel by using threads. In their work, the complete simulation for a model took between 7 and 10 minutes.

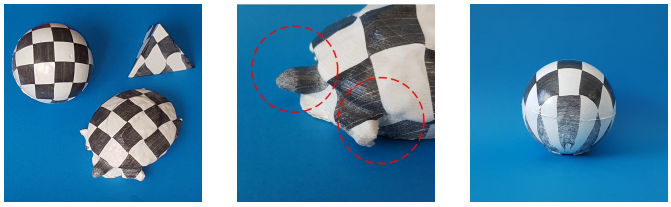
Our work aims to show that it is possible to provide a hydrographic printing simulation in the GPU adopting a different algorithmic solution with PBD [10], yielding better performance while still giving accuracy.

## III. Background

This section presents some relevant aspects of the hydrographic printing process and its physical simulations, particularly applying PBD. We use this knowledge as foundations to implement the simulation proposed.

### A. Hidrographic Printing

Experiments with complex objects like the one in Figure 2a with convexities and concavities, shows that hydrographics bring better results in objects with a convex main face (or at least one significative face). Also, objects with great concavity or occlusions will have areas where the film will not stick properly (Figure 2b). So, predominantly concave objects are inappropriate for hydrographics. Another issue in the hydrographic printing process is that some regions of the film undergo a more significant stretching than others, which leads to a color whitening (Figure 2c).



(a) (b) (c)

Fig. 2. (a) Tested objects. (b) Concavities and occlusions cause problems with hydrographics. (c) The film is highly stretched in some regions of the sphere.

During the hydrographic process, when the film touches the object, it immediately adheres to the object and continues dipping together. Other released parts of the film begin a stretching movement, and it begins to move but on the waterline. The water does not mix with it. Water behaves as a support for the film and helps the film embrace the model. The film behaves like a thin, elastic object that immediately adheres when it touches the object. The film stretches slowly as the immersion movement. When the process ends, the image's points printed over the film occupy some specific position in the object surface. A computational simulation of this process must consider these observations to give a realistic movement between the film and the object.

Based on the physical hydrographic process, it is clear that the simulation modeling would lead mainly to an interaction between a soft-body - the hydrographic film - and a rigid-body moved with constant velocity - the object to be painted. Although there is the interaction between object and the film with fluid - the water - we observe the system formed and considered that should not suffer significant interference of the water [8] [9], since the film remains on the surface without mixing with it.

### B. Physical Simulation

When we use the classical approach with Euler integration [11], the update of the physical system involves calculating the sum of forces that result in the accelerations that allow us to compute the velocities, and finally, the velocities allow the calculation of the positions. This approach can lead to instabilities: explicit integration extrapolates the right side of equations blindly, and for example, by a large $\Delta t$, a spring overshot its equilibrium, causing an internal gain of energy and increasing the instability [12]. In physical simulations that involve computer graphics, for each time-step of a scene, it is essential to update the frame by directly knowing the new position of each vertex, considering that each vertex represents a particle. The PBD approach can address these two problems: provides a more stable and controllable algorithmic solution with the advantage of directly update the positions [10].

PBD was first presented by Müler et al. as an alternative way to process physical simulations on particle based systems. PBD determines a trade-off between numerical precision versus better stability and suitable performance. Instead of applying the full force-based system concepts, driving to a second-order derivative system, PBD by-pass the derivative steps acting directly on the positions of the particles in the system. The PBD simulation starts with forces (externals and internals) acting over the particles and the motion equations. Initially, the classical Eulerian integration handles external forces (like gravity and wind). Internal forces

(like elastic forces in a soft-body) were represented by functions called *Constraints*.

*Constraints* are restrictions used to model the dynamic interaction between the particles in a system. In a simulation, they are used, for example, to stop the particle move when it reaches an obstacle. We also can use constraints to handle the distance allowed between two particles, making it easier to control the maximum deformation of a soft-body in a simulation [13]. Functions usually express constraints. These functions can be an *equality* if it is satisfying when the computed value reaches zero or an *inequality* if it is satisfying when the function reaches a value greater or equal than zero. To simplify the notation, we can represent the *equality* and *inequality* constraints with the $C(p) \succ 0$ notation, where $p$ is a set of references to particle positions bounded to the particular constraint. The set of *constraints* applied to the particles generates a system of equations that can be solved by an iterative way, usually Gauss or Jacobi. This decision gives greater importance to the stability of the system by approximations and corrections around the expected positions.

## IV. Real-Time 3D Position-Based Dynamics Simulation for Hydrographic Printing

The computational steps in our proposed simulation are as shown in Figure 3. The simulation scene has a rigid model with a texture and a deformable triangle mesh representing the film with particles and distance constraints (Figure 3a). Then, the rigid model is pre-processed, creating a voxelization to handle the collision (Figure 3b). The simulation starts entering the condition-controlled loop, updating the rigid model position in each time step, checking the collision, and the film constraints (Figure 3c). When the collision occurs, the simulation proceeds with the collision response, and the film stretches (Figure 3d). Finally, the simulation stops, and we transfer the texture of the rigid model to a simulated flat film (Figure 3e).

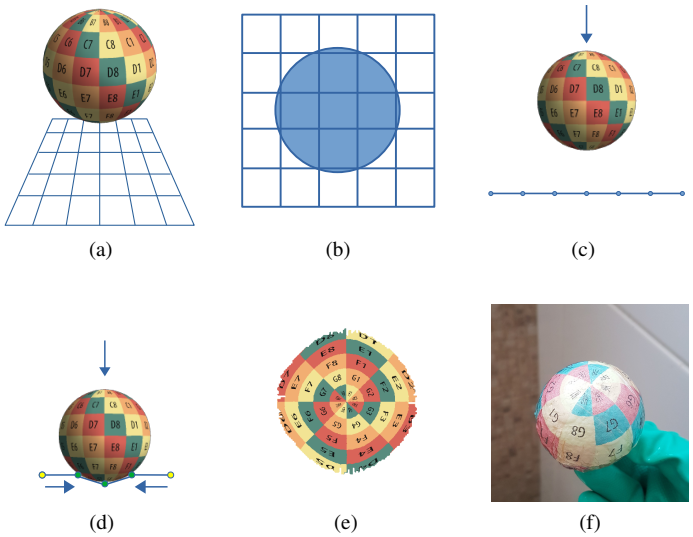


Fig. 3. The simulation starts with (a) texturized rigid model and hydrographic film model. The rigid model (b) is voxelized to collision handling. The simulation moves the rigid body down (c), and handles the collision between the film and the rigid body (d). The texture is transferred to the film with reverse-texture mapping (e). Finally, hydrographic printing is executed in the physical object (f).

### A. Simulation models

The simulation receives as input the rigid body model with a mapped texture. Also, it is necessary to provide the parameters to create the plane mesh considering scale factors and the relative position between the rigid body and the film mesh. We parameterize the number of points in plane mesh width W and its height H, together with the horizontal and vertical distance between each vertex. The plane mesh has its normal vector parallel to the Y-axis and a parameter to define its center. It is possible to define the 2D texture that the plane mesh will assume. We apply a chessboard texture to observe the film stretching when the simulation run. Modeling the scene as in the PBD scheme, the points of the film are equivalent to particles, and each particle has distance constraints in the vertical and horizontal directions (determined by the vertical and horizontal spacing parameter).

### B. Rigid model pre-processing

In a computer simulation, due to discrete updates of the positions, eventually, some collisions are lost, and one body can across another, generating the *tunneling* effect [14]. To address this problem, the pre-processing step (Figure 3b) stores the closest distance $\phi$ from the surface of an object in the grid representation. With this representation, called Signed Distance Field (SDF), negative values are usually the distances within the object, and positive values are distances outside the object [15]. Applying a distance function by traversing the object with a previously created Axis-Aligned Bounding Boxes (AABB) tree can determine the $\phi$ value in each position in the grid. This representation also stores $\bigtriangledown\phi$ as the normal on the surface of the object. $\bigtriangledown\phi$ values make it possible to resolve the collision response by knowing the point of contact and moving the particle by reflecting the normal at this point of contact [14].

### C. Physical simulation

The simulation we set the gravity force to zero. With this, we aim to simulate the flat film floating in the vat. So, when the mesh collides with a rigid body, then the film model with the chessboard texture follows the mesh deformation, creating the visual effect of a thin sheet stretching and adhering to an object. We assume that the film's particles that collide with the object will follow the same vertical velocity of the object. This assumption would be sufficient to represent the adhesion of the film to the rigid body. The particles that have not collided yet are the ones free to move and deform. Their new positions are calculated by the iteration process described in the Algorithm 1.

The Algorithms 1 and 2, adapted from [10] and [14], illustrate how the PBD simulation works. The deformable plane is represented by a set of *N* particles and *M* constraints. Each particle $i \in [1..N]$ has as attributes: its position $x_i$, its velocity $v_i$, the inverse mass $w_i$ given by the mass $m_i$. Lines (1) to (3) just initializes these attribute values.

Each iteration of main loop on lines (4) to (22) runs with a given time-step $\Delta t$ like an Eulerian method. The value of $f_{ext}$ is given by the sum of external forces (in our simulation, we set zero).

Line (7) computes a predicted position $p_i$ for each particle by an explicit Euler integration. Lines (9) to (13), executes the collision handling. These collisions are generated dynamically in every $\Delta t$. Lines (14) to (16) apply the constraints to the

particle through an iterative solution to satisfy the final positioning of $N$ vertex to $M$ constraints. We set the number of iterations when the simulation starts. This number will determine the dynamic of iteratives corrections applied to the predicted position $p_i$. The operation of $\Delta x/n$ represents the applying of under-relaxation, considering the $n$ the number of particles involved. The $solveConstraints$ function at line 15 is detailed in Algorithm 2. Lines (18) to (21) computes the final values to positions and velocities, after running the final correction.

**Algorithm 1** Position-Based dynamics simulation algorithm

1: **for all** $particle_i \in particles$ **do**
2: initialize $x_i = x_{i0}, v_i = v_{i0}, w_i = 1/m_i$
3: **end for**
4: **loop**
5: **for all** $particle_i \in particles$ **do**
6: apply forces $v_i \leftarrow v_i + \Delta t w_i f_{ext}(x_i)$
7: predict positions $p_i \leftarrow x_i + \Delta t v_i$
8: **end for**
9: **for all** $particle_i \in particles$ **do**
10: find neighboring particles ($p_i$)
11: find solid contacts
12: **end for**
13: solveContacts
14: **loop** solverIteractions **times**
15: solveConstraints($C_1$, ..., $C_M$)
16: $x_i \leftarrow x_i + \Delta x/n$
17: **end loop**
18: **for all** $particle_i \in particles$ **do**
19: velocity update $v_i \leftarrow (p_i - x_i)/\Delta t$
20: $x_i \leftarrow p_i$
21: **end for**
22: **end loop**

**Algorithm 2** Constraint solver algorithm (scatter)

1: **procedure** SOLVECONSTRAINTS($constraints$)
2: **for all** $particle_i \in particles$ **do** in parallel
3: initialize position delta $\Delta x_i \leftarrow 0$
4: **end for**
5: **for all** $constraint_j \in constraints$ **do** in parallel
6: compute $\lambda_c$
7: **for all** $particle_i \in constraint_j$ **do**
8: compute constraint gradient $\bigtriangledown x_i C$
9: atomically update $\Delta x_i \leftarrow \Delta x_i + w_i \lambda_c \bigtriangledown x_i C$
10: **end for**
11: **end for**
12: **end procedure**

*1) The Solver:* PBD solves non-linear equations with equality and inequality constraints. First, it becomes the problem linear. After that, it is necessary to find the solution for an under-determined system of equations by converting it to a minimization problem. The Equation 1 expresses the linearization process. It applies the linear approximation in a multi-variable function to an equality constraint that suffered a correction of $\Delta p$ to its position in the $\bigtriangledown C(p)$ direction [10].

$$C(p + \Delta p) \approx C(p) + \bigtriangledown C(p) \times \Delta p = 0 \quad (1)$$

$$\Delta p = \lambda M^{-1} \bigtriangledown C(p)^T \quad (2)$$

$$\lambda = -\frac{C(p)}{\bigtriangledown C(p) M^{-1} \bigtriangledown C(p)^T} \quad (3)$$

The correction $\Delta p$ shown in Eq. 2 is made with the Lagrange multiplier $\lambda$, restricted to the constraint gradient $\bigtriangledown C(p)$. The Eq. 2 is weighted by the inverse mass of the particle, represented by the matrix $M^{-1}$, where $M = diag(m_1, m_2, ..., m_N)$ and $m_i$ is the mass of each particle. Combining Eq. 1 and 2, we have Eq. 3 [10].

The simulation uses the distance constraint $C(p_1, p_2) = |p1 - p2| - d$, keeping particles $p_1$ and $p_2$ at distance $d$. Satisfying this constraint implies in apply the derivatives in each point. They are $\bigtriangledown_{p_1} C(p_1, p_2) = n$ and $\bigtriangledown_{p_2} C(p_1, p_2) = -n$ respectively, given $n = \frac{p1-p2}{|p1-p2|}$. With $\lambda$, the scale factor weighted by the inverse masses $w_i$, we have $\lambda = \frac{|p_1-p2|-d}{w_1+w_2}$ , the formulations 4 and 5 to correct its positions [10].

$$\Delta p_1 = -\frac{w_1}{w_1 + w_2}(|p1 - p2| - d)\frac{p1 - p2}{|p1 - p2|} \quad (4)$$

$$\Delta p_2 = +\frac{w_2}{w_1 + w_2}(|p1 - p2| - d)\frac{p1 - p2}{|p1 - p2|} \quad (5)$$

*2) Parallel computing of constraints:* Usually, a particle is associated with many constraints. Gauss-Siedel (GS) solvers can process each constraint sequentially. To improve the performance, the GS solver can use the GPU to process grouped constraints in parallel with a coloring graph fashion [16]. On the other hand, Jacobi solvers use the GPU to process constraints in parallel, with an under-relaxation solution based on the constraint deltas average. First, computing constraints associated with a particle in parallel and accumulating the position deltas for each particle. After that, computes the average value dividing the accumulated deltas by the total number of constraints affecting the particle. This processing can be parallel with a particle-centric (gather) approach or constraint-centric approach (scatter). We use a constraint-centric way, as shown in Algorithm 2, with each GPU thread to process all the associated particles. After processing all position deltas, we execute an atomic update. This solution has less performance and is harder to converge [17], but considering the number of particles and constraints involved, we consider this solution suitable.

### *D. Collision handling*

Physical-based simulations in real-time needs of an efficient collision handling method. Checking the collision between two bodies with a naive implementation can lead to a $O(m \times n)$ computational effort, where *m* and *n* are the number of vertices of each body, this makes it unsuitable for maintaining a real-time simulation for large values of $m$ and $n$. To face this problem, many approaches can be applied, providing efficient data structures and adopting parallel checking in the GPU [18] [19] [20].

We adopt a grid-based method, subdividing a limited space with volumes (as cubic voxels) in a uniform data structure [19]. The voxel structure has 128 voxels in each axis on a 3D space, with 2,097,152 voxels representing the cube containing the rigid body, with regions inside and outside this body. The neighbor finding for collision checking was provided by the spatial hash solution, as presented in [20]. This method reduces the time drastically to search particles in the neighbor because the given data structure stores the particles in a hash structure, with spatially

close particles occupying the same hash cluster. In addition, it is possible to process this task in parallel.

### E. Reverse texture-mapping

The computational simulation steps provide the visualization in real-time how the hydrographic film deforms when adheres with a rigid body. However, this simulation only provides the movement between the two bodies and does not perform the automatized generation of the planned texture with a suitable deformation to apply the hydrographic printing in a real 3D printed object.

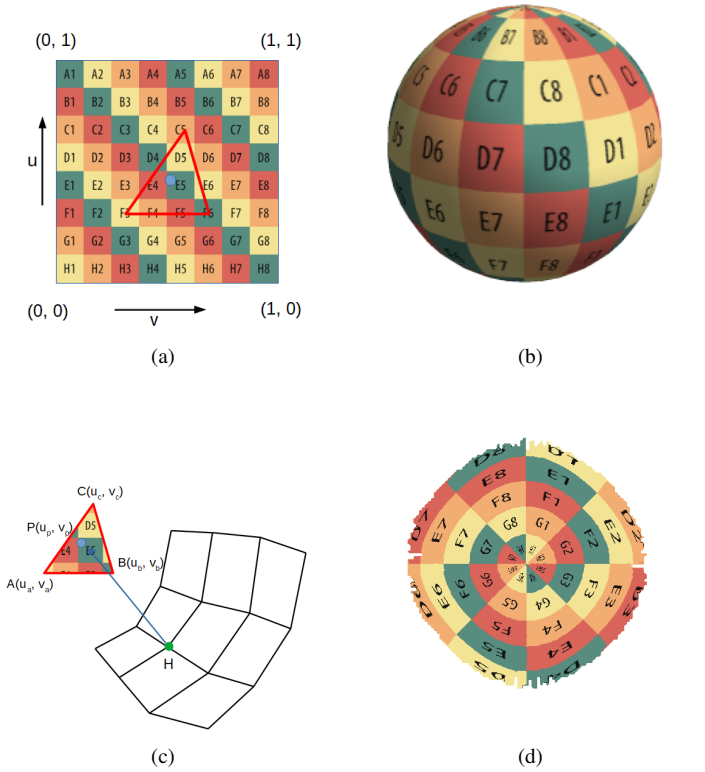



Fig. 4. Reverse texture-mapping process: (a) Texture, (b) The texture applied to the rigid model, (c) Reverse-texture process to obtain texture coordinates for the film mesh from textured rigid model (d) The reversed texture.

This final step, called reverse texture-mapping, consists of solving the following problem: given a rigid mesh with a mapped texture, determine a 2D image to be printed on a real hydrographic film, so that when performing the hydrographic painting through a controlled movement correctly transfer colors to the correct points of the real object. This 2D image is a flattening of texture applied to the rigid body, with an adequate deformation according to the stretch of the film and with the curvature of the rigid body at the point where it touched the film during the immersion.

For this, we determine a function that performs the reverse mapping of the texture, as shown in Figure 4. The input of this function is the vertices of the rigid body with the mapped texture coordinates, as shown in Figure 4b, and its final objective is to determine the equivalent texture coordinates at the vertices of the film. When the bodies collided, the simulation generates contact points between the rigid body and the deformable one. For each contact point $H$ present in the mesh of the film, there is a triangle in the rigid body mesh with vertices $A$, $B$, and $C$ with that can be touched by this contact point, as shown in Figure 4c. This triangle is reached at $P$ through a ray-triangle intersection $\vec{HP}$ casting a ray in the direction of contact point $H$ normal (considering the deformed film) to the rigid body with the Möller-Trumbore algorithm [21].

Then, first we use the Equation 6 representing the arbitrary point $P$ in the triangle $A$, $B$ and $C$, considering $P$ with barycentric coordinates$u$, $v$ and $w$. After that, whe generate the Equation 7 to interpolate the texture coordinates $u_p$ and $v_p$ of this arbitrary point belonging to the triangle with vertices $A$, $B$ and $C$ with respective texture coordinates $u_a$, $v_a$, $u_b$, $v_b$ and $u_c$, $v_c$.

$$P = w \times A + u \times B + v \times C \tag{6}$$

$$P(u_p, v_p) = w \times A(u_a, v_a) + u \times B(u_b, v_b) + v \times C(u_c, v_c) \tag{7}$$

At the end of this process, the procedure generates the reverse texture coordinates of the film dynamically at the points where the film touched the rigid body. Thus, when considering the vertices of the film in its original flat position as shown in the Figure 4d, keeping the reverse texture coordinates mapped dynamically, we render a plane with the reverse texture mapping and the expected distortion.

This interpolation process has two consequences: (1) possibly, there will be regions where some vertex of the film may be left without a coordinate because the ray-triangle intersection does not occur. (2) eventually, a triangle of the film mesh will fall on an area where there is a seam in the texture-mapped in the rigid model (Figure 5a), generating artifacts as shown in Figure 5b. Similar problem was already described in [22] and [23].

We address the first problem by performing a post-processing procedure filling the reverse texture coordinates. If a triangle has a vertex with missing texture coordinates, then we use the texture coordinates already filled in to interpolate the missing texture coordinate and fill it.

For the second problem, we use a *geometry shader* in the reverse texture mapping with two aims: (a) detect film triangles that generate seams; (b) subdivide the film triangle creating new triangles with new interpolated texture coordinates from a subdivision scheme based on the midpoint of each edge of the triangle (see Figures 5c and 5d).

## V. Results

We execute tests aiming to evaluate the proposal considering the performance and accuracy. To do the performance tests, we vary the film resolution and the uniform grid resolution. To perform the accuracy tests, we execute the physical hydrographic printing in three different models. Then, we simulate the hydrographic printing to the same models, and we compare reference points in the pictures taken from the physical and the simulated result.

### A. Setup to physical hydrographic experiment

The reference models used as ground truth in the physical experiment were a sphere with a diameter of 70mm, a tetrahedron with an edge of 60mm, and a turtle model with dimensions of 110mm x 75mm. We use a squared plastic vat plastic box with 29 liters, a tripod connected with a linear actuator (for controlling the dipping around Y-axis), adapted support to the object, plastic barriers adapted with aluminum profiles (for limit the film movement over the waterline), hydrographic blank film A4 sized, with the pattern previously printed with a pigment ink printer and varnished, activator reagent spray.

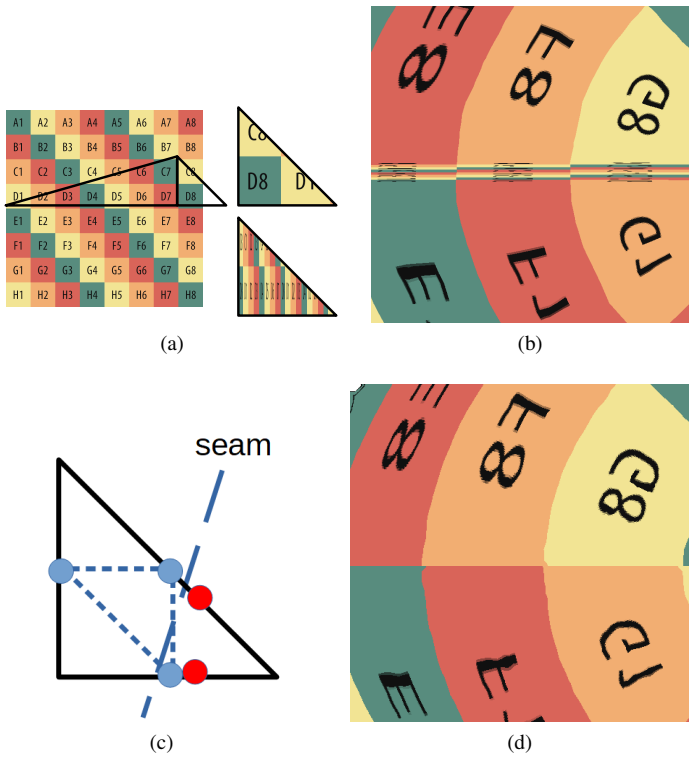



Fig. 5. Some texture coordinates can generate seam problems: (a) triangle with seam problem, (b) seam artifacts are generated, (c) a subdivision scheme is applied by using the midpoint of each edge of the triangle, (d) final result.

### B. Hardware and software configurations

The simulator runs on an Intel Core i5-7200U CPU, in an NVDIA GeForce 940MX GPU with OpenGL 4.3 and Compute Unified Device Architecture (CUDA) 9.2. The software performs a simulation of the hydrographic printing process in real-time for a rigid body mesh and an arbitrary texture.

### C. Simulation parameters

To perform the accuracy and performance tests, we created a chessboard texture proportional to an A4 paper, each square with a 17mm edge. We built the simulated film by a plane mesh with M x N vertices, as shown in Table I. Then we apply the chessboard pattern in this plane. Each vertex in the plane is connected vertically and horizontally with another by distance constraints. These constraints are processed in parallel using the NVIDIA Flex solver.

Our simulation runs with a time-step of 16.67 milliseconds and a dipping velocity equivalent to 5mm/s in the negative direction of the Y-axis. In our tests, this velocity was suitable and similar to the physical process. We define empirically 5 solver iterations by time-step to provide a realistic behavior to the simulation.

### D. Performance Evaluation

To evaluate the simulation performance, we made tests with three rigid models (sphere, tetrahedron, and turtle) varying independently: (1) the number of vertices of the film, (2) the number of voxels in the rigid body grid used in the SDF collision.

In the first simulation, we used each rigid model and the uniform grid with $128^3$ voxels to handle the collision with the SDF method. So we varied the resolution of the film mesh and measured the frames per second (FPS), as shown in Figure 6.

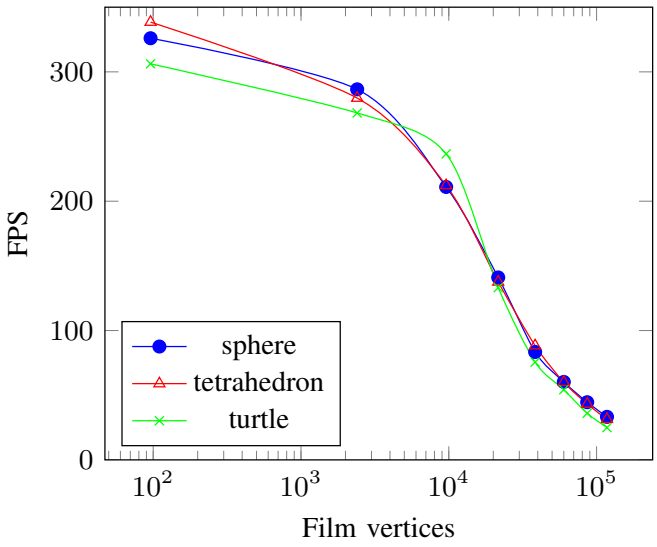



Fig. 6. Variation of FPS by increasing the number of film vertices and constraints (shown in logarithmic scale to $x$ axis) with sphere, tetrahedron and turtle models using uniform grid with $128^3$ voxels.

Taking the sphere model as an example, the number of constraints and FPS varied according to Table I. As expected, by increasing the number of film vertices and constraints, the time of processing each frame increased, but the FPS remained in real-time.

In the second test, we varied the number of voxels in the uniform grid used on SDF collision. Figure 7 shows the results as a line graph. This graph is stable, keeping the FPS in real-time. The spatial subdivision process influences directly in the time of nearby element finding and in the time of collision checking.

Some oscillation between measurements in Figures 6 and 7 can be explained by two factors: (1) when the simulation begins, the NVIDIA Flex library creates some internal structures at the first time, and this leads to higher time consumption. When simulation parameters change, the simulator empties dynamic structures such as hash tables to start a new simulation. However, the memory remains pre-allocated (by the performance definitions of the library implementation), and this process leads to less time consumption. (2) the simulation that updates positions and solves PBD is executed in parallel on GPU; it processes too many values asynchronously, and it sends the intermediate values that provide the taken performance measures to the CPU at irregular time intervals (according to callback functions triggered by the GPU), and this leads to fickle measurements. Despite these fluctuations, all measurements lead to a real-time simulation.

TABLE I
FPS VALUES ACCORDING THE NUMBER OF FILM VERTICES AND DISTANCE CONSTRAINTS. THE RIGID BODY CONSIDERED WAS A SPHERE WITH 25,921 VERTICES AND THE COLLISION MODEL USED A UNIFORM GRID WITH $128^3$ VOXELS.

| Film vertices | Constraints | FPS |
|---|---|---|
| 96 | 172 | 326.00 |
| 2,400 | 4,700 | 286.50 |
| 9,600 | 19,000 | 211.01 |
| 21,600 | 42,900 | 141.04 |
| 38,400 | 76,400 | 83.44 |
| 60,000 | 119,500 | 60.26 |
| 86,400 | 172,200 | 44.58 |
| 117,600 | 234,500 | 33.28 |

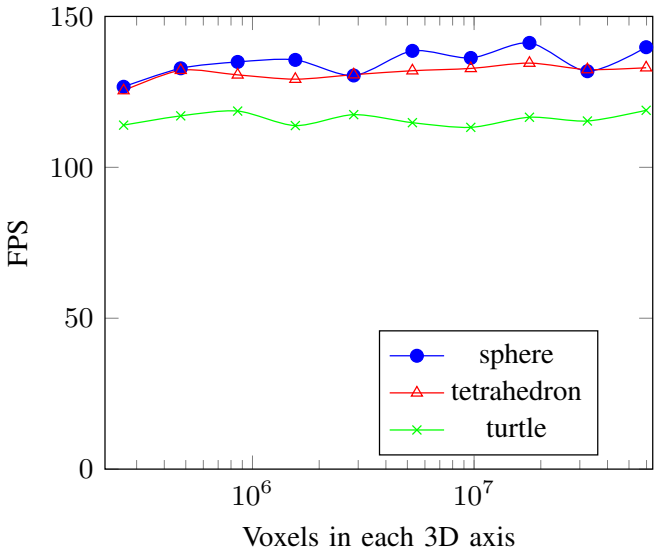



Fig. 7. Variation of FPS by increasing the voxels in each axis of 3D mapped space to collision detection (each value $N$ on $x$ axis corresponds to $N^3$ voxels). Tests were made with sphere, tetrahedron and turtle models colliding with the film mesh with 21.600 vertices and 42.900 distance constraints.

### *E. Accuracy evaluation*

For the evaluation of accuracy, we do not use the reverse texture-mapping, as shown in Figure 4d. Instead, we perform the physical hydrographic technique applying a flat chessboard pattern in the models as ground truth (see Figures 8a, 8c and 8e). Chessboard patterns can provide adequate insight into mesh distortion and help to find distance references between the texture and the 3D object used as a reference.

After that, we perform the simulation with the scale factors calibration to each model. Then, we apply the chessboard texture when the film still is flat and execute the simulation, observing the film stretch and the texture deformation. When the simulation stops, we take a picture of the result in the same position as the real model photo. Figure 8 shows these photos side by side for comparison.

Then, we take *N* measures comparing the pixel positions of the reference points in the two images. These reference points are the vertices of each chessboard square. Each pixel $p_i(x, y)$ at the real image, has an equivalent pixel $q_i(x, y)$ at the simulated image. We calculated the distance $R_i$ between each point of the real image and the simulated one in pixels. With these sample points, we compute the average distance $\bar{R}$ with its standard deviation and show the results in Table II. To evaluate a percentage representation between the average distance $\bar{R}$ and the maximum distance between two points in the used images, we consider the diagonal with 1,900.70 pixels. The calculated values of each percentage representation are in Table II.

Aiming to visualize how the film stretches in the models simulated, we calculate the stretch coefficient for each distance constraint when the simulation stops (by dividing the final length of each distance constraint and its initial length). We plot these coefficients in a heatmap, as shown in Figure 9. Thus, we observed that each tested model has its stretch fingerprint on the film. We observe that the points with the most significative stretch coincide with the points of greatest deviation comparing each simulated image and the respective photo taken, and consequently

TABLE II
MEASURES MADE BETWEEN THE ORIGINAL MODEL AND THE SIMULATED MODEL, CONSIDERING EACH IMAGE WITH 1,344 X 1,344 PIXELS AND RESOLUTION OF 300 DPI, MEAN ± STANDARD DEVIATION, PERCENTAGE RELATION BETWEEN THE MEAN AND THE MAXIMUM DISTANCE BETWEEN TWO POINTS CONSIDERING THE DIAGONAL OF THE WORKED IMAGE

| **Model** | **N** | $\bar{R}$ **(pixels)** | $\bar{R}$ **/ Diagonal (%)** |
|---|---|---|---|
| Tetrahedron | 14 | 10.62 ± 6.32 | 0.56 |
| Sphere | 16 | 14.78 ± 8.60 | 0.78 |
| Turtle | 13 | 10.92 ± 9.49 | 0.57 |

are the most difficult to predict the final position in the film. On the other hand, the points with a low stretch are better aligned compared to the simulation and the photo taken. Consequently, these points in the low stretch are easiest to predict the final position.

The Figure 10 shows initial results with two immersions, considering physical and simulated steps. For this, we use the 3D reconstructed model of the turtle and the proposed simulation to create the top and bottom reverse textures. After that, we apply the hydrographic printing and color the model with two immersions. However, this process with two immersions is complex, it requires precise alignment to avoid damaging the first painting during the second immersion.

## VI. CONCLUSIONS

We presented a computational simulator for Hydrographic Printing. In 3D printing, the use of hydrographics can be extended to coloring 3D printed objects, with a simulation to improve the precision of the physical hydrographic process with a low-cost setup. The implemented simulation produces results quite promising. Unlike other works, our contribution was to generate the simulation in real-time frame rates by using the GPU.

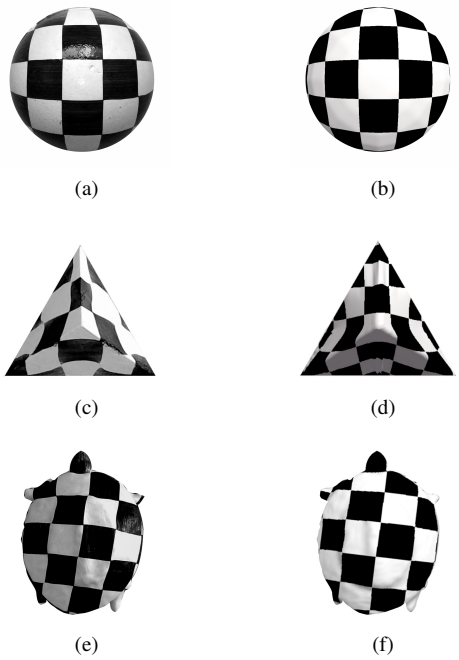
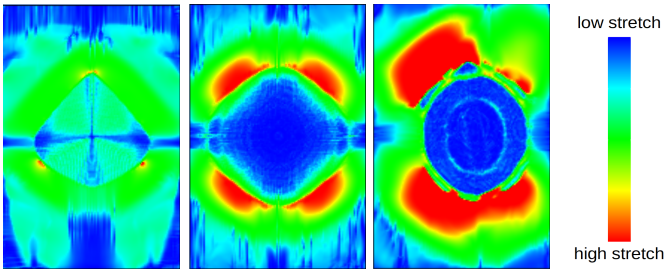



Fig. 8. Comparison between real and simulated models. At left, the sphere (a), a tetrahedron (c), and turtle (e) sample models used as ground truth. At right, the simulated models of the sphere (b), a tetrahedron (d) and turtle (f).




(a) (b) (c)

Fig. 9. Fingerprint of the stretch in each model tested: (a) tetrahedron, (b) sphere and (c) turtle.

(a) (b) (c)

Fig. 10. Color process with two immersions for the turtle model: (a) As input the 3D printed model and the texturized model, (b) the proposed simulation generates the textures to top and bottom. After the physical process with two immersions by using these textures, the turtle is fully colored (c).

We introduce PBD to simulate a real-world physical process, in controlled conditions to reach the final results. Despite the approximations using the PBD, the simulation works with suitable accuracy, denoting that the simulation is useful to predict the distortion in the evaluated meshes. The quantitative evaluation made was the key to understand in which situations the simulation worked fine. We are working on enhancements over the actual results by fine-tuning parameters or adapting the algorithm. These enhancements will enable us to improve the simulation of the physical process, even more, keeping the purpose of real-time performance.

The color control on the hydrographic film was not the focus of this work, but it is desirable to print an image on the film that can balance the color distortions, generating a reverse compensation (increasing the pigmentation in regions with more significant stretching). For this, it will be necessary to investigate methods of simulating the variation of the thickness of the film. Then, it will be possible to propose a function that turns the color darker in a region of the film with more significant stretching.

## Acknowledgment

Thanks to reviewers for all the essential feedback received.

This research was supported by the Brazilian National Council for Scientific and Technological Development (CNPq) with the grant 372539/2018-1 under project 406719/2016-0.

Thanks to the Federal Data Processing Service (SERPRO), which supported the author Leonardo Thomas with the internal graduation program by granting hours to developing this research.

Thanks to the Museum of Archaeology and Ethnology (MAE) staff from the Federal University of Bahia (UFBA). They collaborate and grant permission for the 3D reconstruction of the turtle used in the results section.

## References

[1] B. Berman, "3-d printing: The new industrial revolution," *Business horizons*, vol. 55, no. 2, pp. 155–162, 2012.

[2] S. Mohr and O. Khan, "3d printing and its disruptive impacts on supply chains of the future," *Technology Innovation Management Review*, vol. 5, no. 11, p. 20, 2015.

[3] M. Neumüller, A. Reichinger, F. Rist, and C. Kern, "3d printing for cultural heritage: Preservation, accessibility, research and education," in *3D Research Challenges in Cultural Heritage*. Springer, 2014, pp. 119–134.

[4] A. Inc, "3d printing of colored models on multi-head printers," U.S. Patent US9 833 948B2, 2014.

[5] J. Weiler and S. Kuznetsov, "Crafting colorful objects: a diy method for adding surface detail to 3d prints," in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 2017, pp. 2217–2223.

[6] R. M. Arnold, "Dry transfer decal and method of manufacture," 1985, uS Patent 4,517,044.

[7] M. Nakanishi, "Printing apparatus," U.S. Patent US4 436 571A, 1984.

[8] Y. Zhang, C. Yin, C. Zheng, and K. Zhou, "Computational hydrographic printing," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 131, 2015.

[9] D. Panozzo, O. Diamanti, S. Paris, M. Tarini, E. Sorkine, and O. Sorkine-Hornung, "Texture mapping real-world objects with hydrographics," *Computer Graphics Forum (proceedings of EUROGRAPHICS Symposium on Geometry Processing)*, vol. 34, no. 5, pp. 65–75, 2015.

[10] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position based dynamics," *Journal of Visual Communication and Image Representation*, vol. 18, no. 2, pp. 109–118, 2007.

[11] A. Bargteil and T. Shinar, "An introduction to physics-based animation," in *ACM SIGGRAPH 2018 Courses*, 2018, pp. 1–1.

[12] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson, "Physically based deformable models in computer graphics," in *Computer graphics forum*, vol. 25, no. 4. Wiley Online Library, 2006, pp. 809–836.

[13] J. Bender, M. Müller, and M. Macklin, "A survey on position based dynamics, 2017," in *Proceedings of the European Association for Computer Graphics: Tutorials*, 2017, pp. 1–31.

[14] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, "Unified particle physics for real-time applications," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 153:1–153:12, Jul. 2014.

[15] K. Erleben and H. Dohlmann, "Signed distance fields using single-pass gpu scan conversion of tetrahedra," in *Gpu Gems 3*. Addison-Wesley, 2008, pp. 741–763.

[16] M. Fratarcangeli and F. Pellacini, "A gpu-based implementation of position based dynamics for interactive deformable bodies," *Journal of Graphics Tools*, vol. 17, no. 3, pp. 59–66, 2013.

[17] M. Fratarcangeli, V. Tibaldo, and F. Pellacini, "Vivace: A practical gauss-seidel method for stable soft body dynamics," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, pp. 1–9, 2016.

[18] R. Bridson, R. Fedkiw, and J. Anderson, "Robust treatment of collisions, contact and friction for cloth animation," in *ACM Transactions on Graphics (ToG)*, vol. 21, no. 3. ACM, 2002, pp. 594–603.

[19] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, and M. H. Gross, "Optimized spatial hashing for collision detection of deformable objects." in *Vmv*, vol. 3, 2003, pp. 47–54.

[20] S. Green, "Particle simulation using cuda," *NVIDIA whitepaper*, vol. 6, pp. 121–128, 2010.

[21] T. Möller and B. Trumbore, "Fast, minimum storage ray-triangle intersection," *Journal of graphics tools*, vol. 2, no. 1, pp. 21–28, 1997.

[22] M. Tarini, "Cylindrical and toroidal parameterizations without vertex seams," *Journal of Graphics Tools*, vol. 16, no. 3, pp. 144–150, 2012.

[23] M. Tarini, C. Yuksel, and S. Lefebvre, "Rethinking texture mapping," in *ACM SIGGRAPH 2017 Courses*, ser. SIGGRAPH 2017. New York, NY, USA: ACM, 2017. [Online]. Available: http://dx.doi.org/10.1145/3084873.3084911