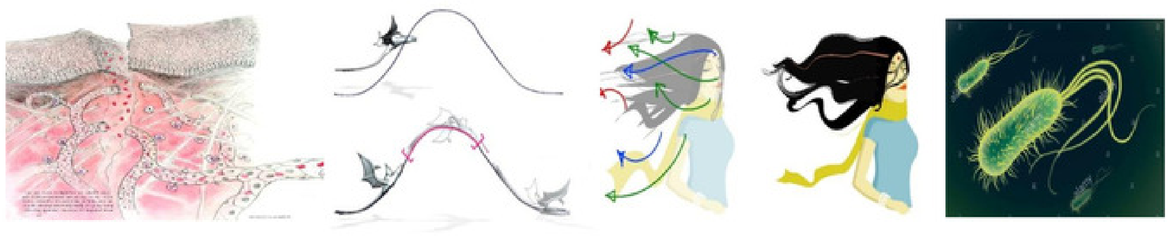

Illustrations from Renaud Chabrier, Guay, Ronfard, Gleicher, and Cani (2015b), Xing et al. (2016) and @Alamy

Project Report

Sketching Animations of Organic Shapes

Lauriane BERTRAND
IGD Master, IP Paris, Palaiseau, France
`lauriane.bertrand@inria.fr`

LIX, Polytechnique

Supervised by Marie-Paule Cani and Damien Rohmer

September 2, 2022

Acknowledgements

I would like to thank all the people who contributed to the success of my internship and who helped me in the writing of this report.

First of all, I would like to thank my professor and tutor, Marie-Paule Cani, a professor at École Polytechnique and member of the GeoVic team at LIX, for accepting me for this internship. I really enjoyed her teaching, which made me want to further develop my skills in this field by working with her team.

I would like to thank my second internship tutor, Damien Rohmer, also a professor at Ecole Polytechnique and a member of the GeoVic team, for his welcome, the time spent together and the sharing of his expertise on a daily basis. He was a precious help in the most delicate moments.

I also thank the whole team for their welcome and their sympathy. I had a great time with them and I hope to keep good links with them. Special mention to Tara Butler, Vincent Bonczak, Damien Rohmer, Maud Lastic, Mathieu Desbrun, Theo Braune, Pooran Memari, Eduardo Alvarado Pinero, James Gain, Jiayi Wei, and Pierre Poulin for their great company during coffee break.

Finally, I would like to thank Marie-Paule Cani and Damien Rohmer who advised and read me during the writing of this internship report.

Contents

1 Introduction

As an intern in the laboratories of Ecole Polytechnique, I spent 5 months and a half working in the GeoVic team. GeoViC is a Computer Graphics team, specialized in geometric modeling and animation. Their goal is to extract and represent the structure contained in manipulated 3D, static or animated contents (captured data, simulation results, imprecise user inputs such as sketches...) in order to unify their processing and find correspondences. Applications cover visualization, manipulation and modification of static or animated data, expressive design of content similar to an example, completion of missing data and replication of details, as well as transfer of shapes and motions with automatic adaptation to a new context. This research takes advantage of the broad skills of the permanent members of the team, which range from expressive design and 3D animation to geometry processing and algorithmic geometry. My internship is part of a long term project coordinated by Marie-Paule Cani, called Creative AI.

The goal of Creative AI is to advance towards intelligent systems that help users express the shapes in motion they have in mind, and their organization into complex virtual worlds. These users may be computer artists, but also the general public, engineers or scientists from other disciplines who have mental images of their objects of study, but cannot seamlessly communicate them. The general methodology is to provide them expressive, gesture-based control (such as sketching, deformation, copy-pasting) while augmenting the graphical models they manipulate with knowledge, was it laws from priors or statistics and correlations learnt from examples.

Within this project, my subject is ”Sketching Animations of organic shapes” and aims to bring contributions in the animation part. My topic being very broad I was free to choose the issue I wanted to contribute on. My only constraints were that the user had to interact by sketching. Thus, at first, my plan was to contribute on sketching methods for animation. Indeed, currently, most of the methods based on sketching simply use a 2D line as input to represent the object's motion. However, the lines depicted by artists in comics have much more meanings. They can give information about speed, rigidity, or behaviors (oscillations, bounces, etc), as depicted in Figure 1. My first idea was to interpret these 2D lines to animate 3D objects. However, as the system has to be useable by the general public I prefered to work on something that doesn't rely on artistic skills.

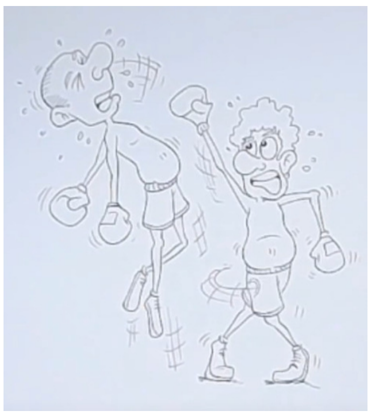

Figure 1: Motion information is given through specific sketch inputs

My second idea was to look for a way to do 3D sketching. As said before, in the most recent research, the input is given in the 2D space, because the interface they interact with is in 2D. However, since the objects to animate are evolving in the 3D space, the problem is ill-posed and is often solved by simplifying it (for example, by restricting the animation to the view plane). I therefore wished to study 3D sketching by interpreting multiple 2D lines into a 3D stroke, as shown in Figure 2. However, similarly to my previous idea, I found out that it would have recquired artistic knowledge, such as perspective.

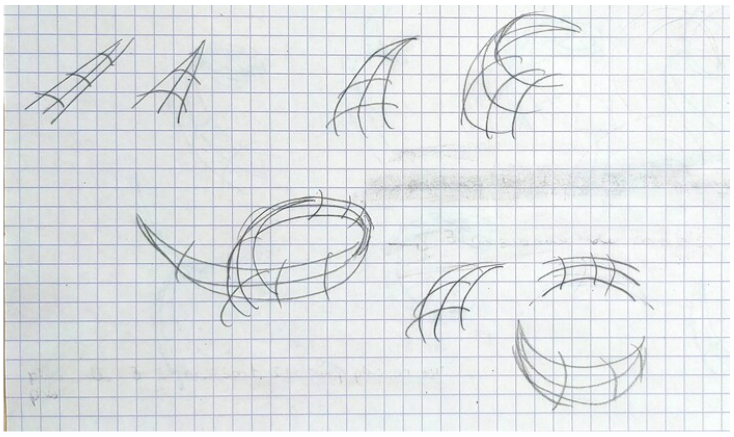

Figure 2: 3D line sketches using 2D strokes

Lastly, my final and selected idea was to do multi-resolution animation by sketching. By multi-resolution, I mean, in this context, the idea of animating a shape, then zooming in on smaller parts of it, animate them and repeat the process again. Many studies have been conducted in order to animate shapes from sketches but most of them are done on human models. However, there are no solutions for more organic shapes like biological or marine forms. Moreover, they propose ways to animate objects but barely let the possibility to control a sub-part of an already animated one. Thus, my goal is to propose a system allowing scientists as well as novices to animate organic shapes simply and quickly from a few sketched inputs. The particularity of these forms is that they very often have a hierarchical structure (hairs attached to tentacles attached to a body for a coral for example as shown on Figure 27). Using the notion of hierarchy for animation has never been studied so I aim to adapt sketch animation to hierarchical structures.

The main concept is to divide a shape into multiple objects having their proper skeleton and motion, using a hierarchical structure. By observing what exists in our environment we can reduce the amount of possible shapes and deformations to the ones described in Figure 3 and work only with cyclic motions. An illustration of the concept of hierarchy is presented in Figure 5 using an euglena (Figure 4) as a reference. In addition to being simple and cyclic, the deformations are also very alike between the parts that look similar (for example all the tentacles of an anemone undergo very similar deformations). In order to speed up the animation process, it is relevant to consider a copy and paste method and this will be part of the method I will present.

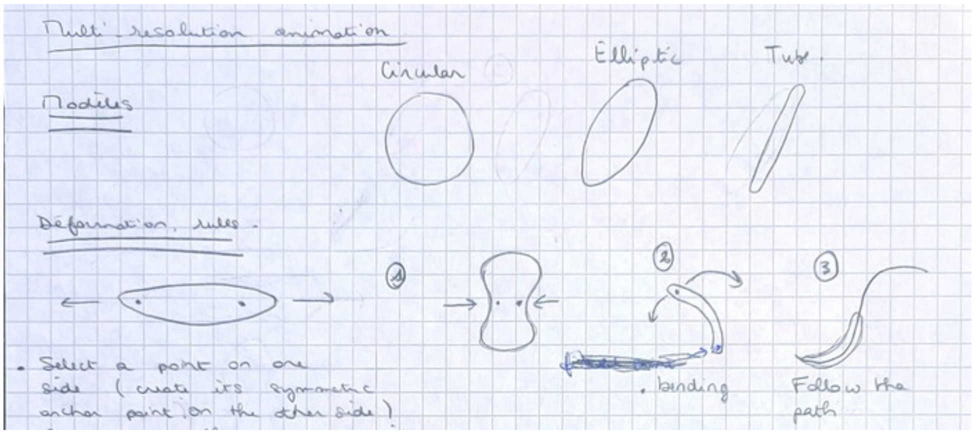
Multi-resolution animation
Modules
Circular
Elliptic
Tube.
Déformation rules.
1
2
3
• Select a point on one side (create its symmetric anchor point on the other side)
• bending
Follow the path

Figure 3: Possibilities of shapes and deformations of a part of an object

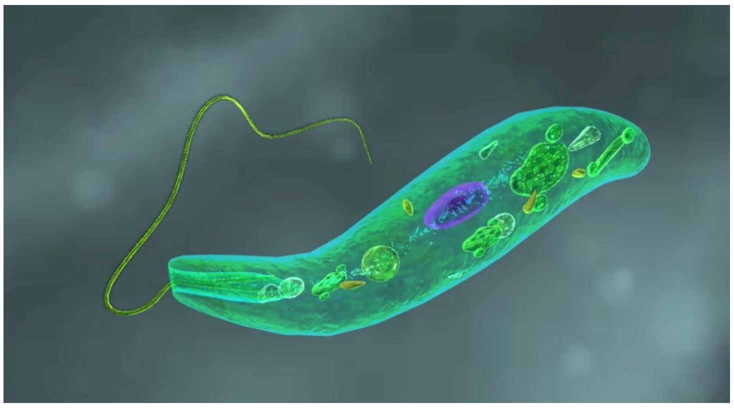

Figure 4: Euglena

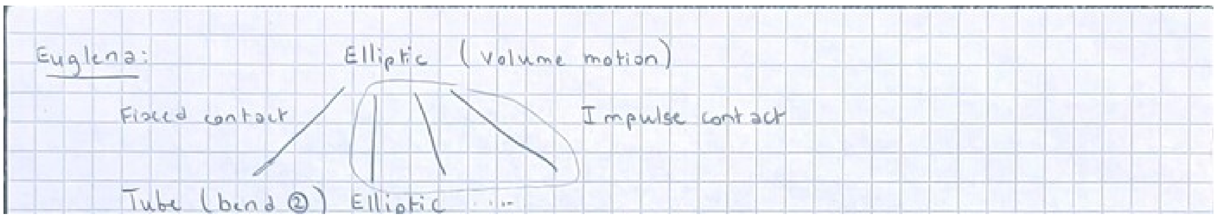

Figure 5: Division of the euglena into multiple objects using a hierarchical structure

Finally, we seek to keep a plausible animation, thus, the animations must remain coherent along the hierarchy. Coherence might be subjective to the user because it depends on what he has in mind but it can be characterized by the fact that if two motions are very similar in terms of frequency they must be in phase at the ends of their trajectory, i.e. they reach their extremums at the same time. However, as individual motions are induced independently from each other by the user, slight phase shifts can be observed. As this has, to my knowledge, never been studied, a second part of my work is to explore possibilities to synchronize the motions, to keep coherency along the hiearchy.

Furthermore, the system must work in real time so that the user can have a direct preview of what he does. My ultime goal is to integrate my system to an already existing software, called

Matisse and proposed by [Bernhardt, Pihuit, Cani, and Barthe (2008)], that allows the user to model 3D organic shapes through sketching.

My contributions are therefore the following:

1. Extend real-time sketching animation to hierarchical structure
2. Propose a method to synchronize independent motions by alterning between manual inputs and automatic interpretation
3. Propose a controlable copy-paste that retarget animation to similar objects
4. Conduct a user study to examine how one draws cyclic motion and validate the need of synchronization

My system is an online software working in real-time, implemented in JavaScript using the available library Three.js.

Section 2 will present the state of the art, then I will explain my work in Section 3 and give my results in Section 4. Finally, limitations and future work will be discussed in Section 5 and Section 6 will conclude my report.

2 Related Work

As previously introduced, this internship focuses on sketch-based techniques for easy and multi-resolution animation. To create dynamic scenes, we want users to have access to intuitive sketch-based tools to animate shapes and propagate the created motion to multiple similar objects. In particular, the animation process should be made possible for a wide variety of organic shapes through the interpretation of input strokes. In addition, we are seeking tools that offer the possibility to edit a motion in space and time and tools that enable the retargeting of an existing motion to similar shapes. In the following, we present an overview of the existing methods and advances that might be suited for our goal and discuss their limitations. This chapter is structured as follows: Section 2.1 is centered on sketch-based methods for motion creation, Section 2.2 focuses on specific tools for the edition process, and Section 2.3 is centered on the hierarchical nature of a couple of methods.

2.1 Motion

Traditionally, an animation was entirely hand-drawn by artists, image per image. Based on the well-known principles of animation, this creative process was either linear (straight ahead, Figure 6 left) or through keyframes and in-betweens (pose-to-pose, Figure 6 right). While straight ahead can be considered the most creative and spontaneous approach, some lack of coherence can appear between the beginning and the end of the animation. In contrast, the pose-to-pose technique requires defining the important steps at first, enabling better control of the animation as a whole.

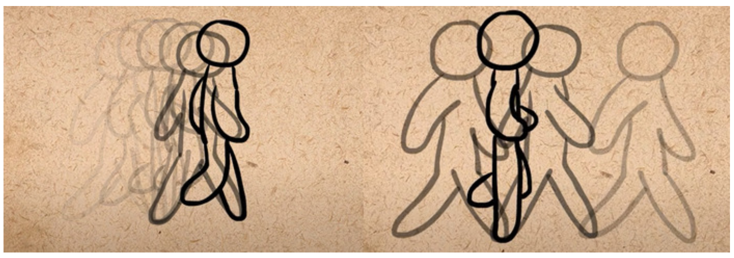

Figure 6: Left: Straight ahead principle, Right: Pose-to-pose principle

Regardless of the chosen approach, hand-drawn animation requires high artistic skills to generate the desired drawings and convey the animation rhythm from a set of fixed images and a timing chart. In complement, drawing each image individually can be very tedious and time-consuming, especially since small changes either in the visual characteristics of an element or in its motion will require redrawing some parts or even the entire set of drawings.

Recently, computer animation methods tried to ease this fully hand-drawn animation process by providing tools to animate either an individual object or a set of elements. These techniques can be classified according to the level of creativity and freedom they propose. For instance, parametric models such as procedural or physically-based systems [Rohmer et al. (2021), Kwon and Lee (2008), Kass and Anderson (2008)] have been intensively studied in the context of natural phenomena. However, they heavily rely on parameters, which can be difficult to estimate by the developer or to tune by the user. This indirect and not intuitive control is therefore not helping the creative process. One alternative, which is also the standard in industrial animation software, is to let the user interact with an environment through direct manipulation. Whereas it is easy to drag and drop an object in space, manipulating precisely a shape that can be complex, is more challenging. In particular, existing methods have mostly relied on the manipulation of anchors positioned on a model. The latter can be manually located by the user but also pre-defined by some support structures such as a bounding cage or more classically the model's rig. While the former can offer more freedom on the motion, extreme dragging can result in unnatural poses as the geometry consistency will not be preserved. In contrast, manipulating pre-defined anchors can be considered too rigid since the amount of control is restricted to the handles' degrees of freedom. In addition, independently of the chosen approach, it can require several iterations or even be impossible to create a specific expressive pose. In contrast to posing, timing can also be difficult to estimate from the user's inputs, and relying on a fixed timeline can be insufficient for an expressive animation. Therefore, a user might prefer relying on sketching (or even combining it with direct manipulation) to ease the animation process. Indeed, as for sketch-based modeling, it can be more intuitive for the user to use sketch inputs to express animations but more computationally complex to synthesize the desired motions and deformations. In the following, we focused on existing methods for the sketch-based animation of an individual object or a set of elements. More particularly, these approaches can be classified relatively to their use of sketching which can be to represent the line of action at keyframes (Section 2.1.1), characterize motion guidelines (Section 2.1.2), and create cartoon-styled animations (Section 2.1.3).

2.1.1 Line of Action

As previously introduced, the pose-to-pose approach consists in, first, defining keyframes (the main steps of the animation) and then, progressively smoothing the animation through the use of in-betweens. Related work targeting an animation per keyframes has mostly focused on approaches either to infer a transition (or in-betweens) out of pre-defined keyframes or to represent the key poses in a coarser and easier representation.

For a complex and detailed object such as 2D or 3D character, hand-drawing one keyframe after the other can be very laborious. Therefore, some attention has been given to an intuitive and abstract representation of key poses based on sketched input, in particular for 3D models. However, inferring the desired 3D pose from 2D sketches can be challenging. The objective is first to identify the user's desired pose from a sketched 2D abstract representation and then adapt the model accordingly.

Targeting the posing of 3D articulated characters from a single line of action, Guay et al. [Guay, Cani, and Ronfard (2013)] propose a sketch-based interface on which the user can sketch a line of action (LOA), and the system automatically aligns the 3D model to fit this line (Figure 7). In addition, secondary lines can be added to refine the generated pose from other viewpoints. They define an LOA as a C or S-shaped curve, guiding the pose both in terms of position and tangents. They define the region of deformation by a bodyline, that is the maximal, connected linear chain in a character skeletal kinematic tree. Following their formal description of the line of action, they determine the desired pose by solving an optimization problem. The strength of their system is to enable the global posing of a shape through only one single stroke even though they are only limited to C and S-shaped.

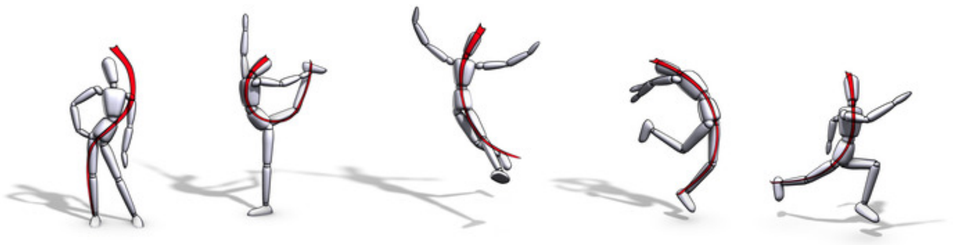

Figure 7: Expressive character poses created by sketching intuitive lines of action.

To simulate the motion induced by multiple LOA, [Guay, Ronfard, Gleicher, and Cani (2015a)] propose a physically-based line interpolation guiding the transitions between two key poses (each defined by a line of action) while preserving bone constraints. This is done by considering the 2D line of action as a piecewise rigid chain with elastic behavior, transitioning from one key pose to another by forward simulation (Figure 8). While this method results in a smooth motion without length discontinuities, the deformation/posing is realized according to the view plane only, because it makes it easier to infer the depth. Thus, the system does not enable non-planar or more complex deformations. In addition, the use of a physically-based system makes it harder to create the motion, as the user has to tune the parameters, which can be a tedious process. Finally, even though these tools provide interesting solutions to ease the creation of keyframes, the inherent problem of this concept is that the focus is only on the precise spatial features for a specified time. Due to the decorrelation between space and time, transitioning between two keyframes can lack rhythm which is an important feature of expressive animation.

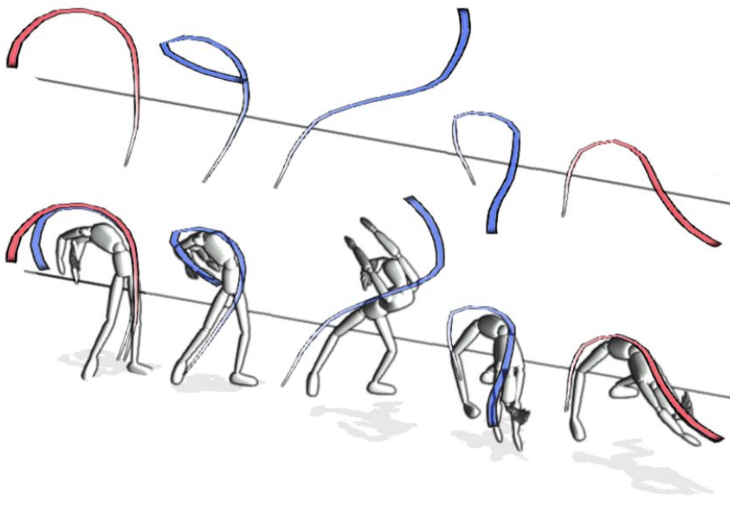

Figure 8: The lines in blue are frames of the physically interpolated line of action, generated between two drawn line of action strokes in red.

Extending the keyframe abstraction concept to dynamics, [Guay et al. (2015b)] introduces a space-time sketching abstraction encoding a full coordinated motion into a single sketch curve called space-time curve (STC) that encodes both trajectory and speed (Figure 9 top). Other curves can then be added to the environment to refine the motion. The authors rely on some assumptions about the desired animation behavior such as the matching in both space and time of the shape and the provided keyframe, to define the concept of dynamic line of action (DLOA). The latter is a 2D parametrized surface where time is seen as a spatial parameter, and constant-time slices are static lines of action. In contrast to [Guay et al. (2013)], the full DLOA, i.e., the sequence of keyframes and the timing, is extracted from the input "space-time curve" drawn in the screen space. In complement, they extend the linear interpolation to preserve local length and C1-smooth motion. From the user's input, they use projective constraints to compute the DLOA that drives the motion. In particular, this is achieved by linear parametrization, followed by a parameter splitting into a spatial and timing variable, from a warping (Figure 9 bottom). This linear parametrization enables squash and stretch effects since the drawing speed is not constant. In addition, this solution allows bouncing and rolling effects by recognizing singular points or loops and by respectively adding a correction term to handle take-off and landing and an additional constraint to the DLOA. With the STC concept, they introduce a way to correlate space and time under the same single curve. However, their method is limited to simple shapes (without limbs) and is not yet extended to 3D curves.

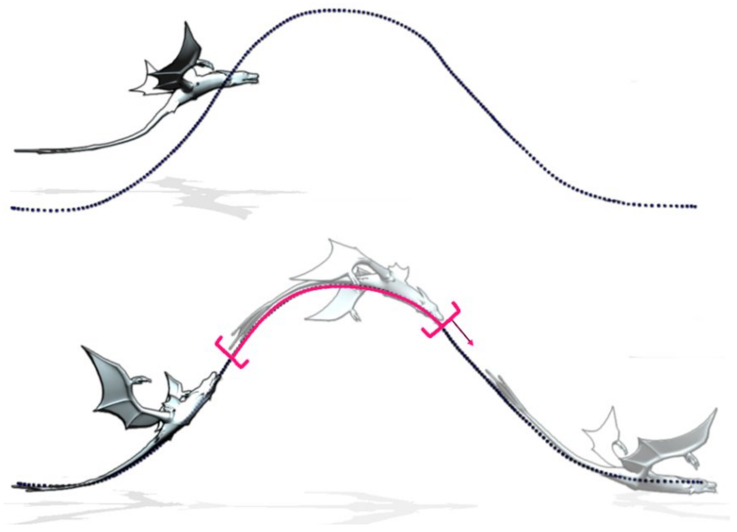
Figure 9: Top image: character model and space-time curve (STC). Bottom image: dynamic warping (red window) within the parametric space of the STC to produce a dynamic line of action.

2.1.2 Cyclic motion

As explored previously, multiple methods have been developed to instantiate general motion through sketching. However, specific motions such as oscillatory and cyclic motions are also very common in computer graphics. Specifically, many biological forms have cyclic or nearly cyclic behaviors. Using the additional information derived from their characteristics, one can find methods that are more suitable for this type of animation. Thus, we'll now explore some sketching techniques specific to motion cycle design.

[Pentland and Williams (1989)] propose to describe geometry and dynamics separately, in order to use a vibration-mode representation of the dynamics. Using its cyclic behavior, they separate the non-rigid motion into a sum of independent vibratory modes (Figure 10). On the other side, the geometry is handled using a volumetric model, called superquadrics. Both representations are then coupled with a polynomial deformation mapping to obtain the final deformation. The modal representation of the dynamics has great qualities as it produces great gains in efficiency, reduces temporal aliasing, allows stretching and squashing, and handles object collisions. However, like all traditional forward simulation methods, they fail to provide the kind of control we require. There is no way for an animator to craft a series of poses at different times with these modal techniques and know that the dynamics will accurately recreate them. In addition, this method provides only the original simulation parameters as a way of changing the motion.

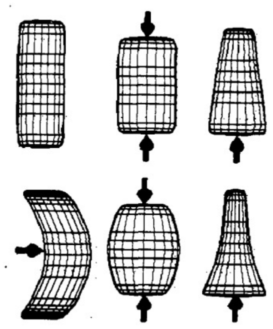

Figure 10: (a) A cylinder, (b) a linear deformation mode in response to compression, (c) a linear deformation mode in response to acceleration, (d) a quadratic mode in response to a bending force, (e) superposition of both linear and quadratic modes in response to compression, (f) superposition of both linear and quadratic modes in response to acceleration.

As an improvement of the previous technique, [Kass and Anderson (2008)] introduce the concept of Wiggly Splines. The spline generalizes traditional piecewise cubics when its resonance and damping are set to zero, but creates oscillatory animation when its resonance and damping are changed. They create the spline by applying a spacetime constraint formalism based on a physical system composed of a single mass and a spring with constraints and minimize an objective function that penalizes non-physical and inefficient motion. Constraints consist of positions and tangents and are provided by the animator (Figure 11). Moreover, Wiggly Splines address a key issue known to animators by the term “overlap”, by using complex-valued filters. Overlap generally refers to the fact that different degrees of freedom of a given model need to accelerate and decelerate at different times. Associated with a phase shift along the surface geometry, the method can model drag effects followed by wiggling motion. While their technique offers a clean and fluid oscillation behavior, the magnitude and phase field should however be parameterized by the user on every given model and do not automatically take advantage of the existing rig. Moreover, the use of Wiggly Splines is more suited for secondary motion than the main motion of a given shape.

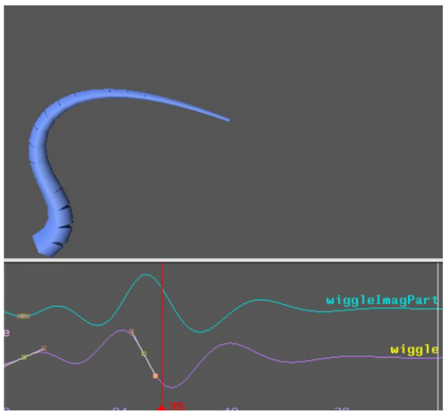

Figure 11: Animation of a procedural tail.

Alternatively, [Ciccone, Guay, Nitti, and Sumner (2017)] animate an item (full 3D model, rig controller, bone, etc) by sketching several loops of the same motion around the concerned entity. A motion cycle is defined by a set of sketched nearly cyclic repetitions. The user sketches as many sets of cycles as wants, which reduces inaccuracies and potential noise in the input data. From this data, the system identifies repetitive patterns in three steps. First, an analysis of the spatial and velocity variations of each pair of points is used to estimate the cycle period. Then, correspondences between cycles are computed to find the similarity in the extracted cycles and thus define an average cycle. Finally, a Bezier curve is fitted to each component of this cycle (translation, rotation, scaling). As illustrated in Figure 12, several motion cycles can be sketched for different items, enabling the creation of more complex animation. Even though the authors provide some editing tools that will be discussed later on, their system is highly dependent on the quality of the input and an inaccurate spline model could be found in the case of too strong heterogeneity between the user cycles.

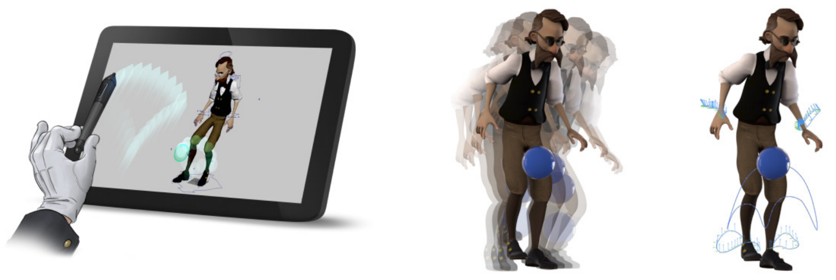

Figure 12: The user draws several loops (left), and a looping motion cycle is extracted from the noisy input (middle); the user can combine different motions to create a complex animation (right)

Similarly, [Dvorožňák et al. (2020)] propose a tool enabling 3D modelization followed by direct animation through sketching (Figure 13). They combine a 3D inflation for modelization with a layered ARAP (ARAP-L) for animation where the constraints are both positions and depth-ordering. In the animation mode, the user can create a control point anywhere on the character's surface, and animate it by moving it in real-time while the software records the motion. The user can create and animate additional control points as desired, and their movements will then be automatically synchronized with the timing of the first (master) point. In this way, the user can build up a complex action like a walk by layering cycles, one body part at a time. Furthermore, the authors propose three recording modes to ease the animation and help with synchronization: (1) overwriting fixed-length cycles, where the control point's movement is recorded continuously in a loop, constantly overwriting the previous cycle; (2) averaging fixed-length cycles, where the current movement is blended with previous cycles, which can serve to smooth out kinks and pops, and (3) time-scaled cycles, where each cycle begins on the same frame, but all the cycles are stretched or compressed in time to match the timing of the master control point. Although their method makes it possible to rapidly create consistently animated meshes from a few hand-drawn strokes, they are restricted to a single view.

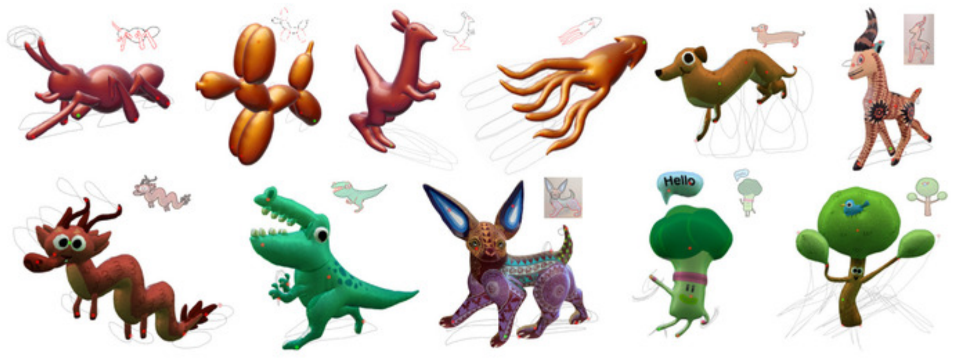

Figure 13: A selection of animated characters created using Monster Mash by animators during user study. For each example, the original hand-drawn outlines are visible in the inset above (superimposed over the original drawing or a source photo). From these outlines, Monster Mash can inflate a smooth, consistent 3D model that can immediately be animated using several control points (red and green dots). Their trajectories are visualized as grey curves.

2.1.3 Cartoon animation

A lot of research has been conducted to create realistic motion data for character animation. However, while such founded methods are excellent in generating realistic animations, they are not suitable for producing cartoon-style animation which has a different nuance. The principles of traditional cartoon animation have been widely introduced into computer graphics, and some mathematical techniques for non-photorealistic animation have been developed, such as squash and stretch, anticipation, and follow-through.

[Kwon and Lee (2008)] achieve to give a cartoonish effect through the creation of a sub-joint hierarchy (a superset of the original joint hierarchy of the character) and the smooth exaggeration of the motion using trajectory-based motion exaggeration and physical simulation (Figure 14). The algorithm consists of two steps. First, trajectory-based motion exaggeration is applied to make a distorted motion which involves the stretching of links. Next, they divide the links of the original joint hierarchy into small unit joints (sub-joints) and use them to mimic the bending effect of rubber. This sub-joint hierarchy can be precomputed, and the motion can be improved by adding in a mass-spring simulation. While they achieve to create a cartoon-like animation, they can't guarantee volume preservation, as the length of the links is modified. Moreover, careful control of the exaggeration parameter is needed for generating collision-free and plausible results. Finally, the amount of data required for exaggerated motions is much greater than that required for the original motion because of the additional joints and the need for 6 DOF channel data.

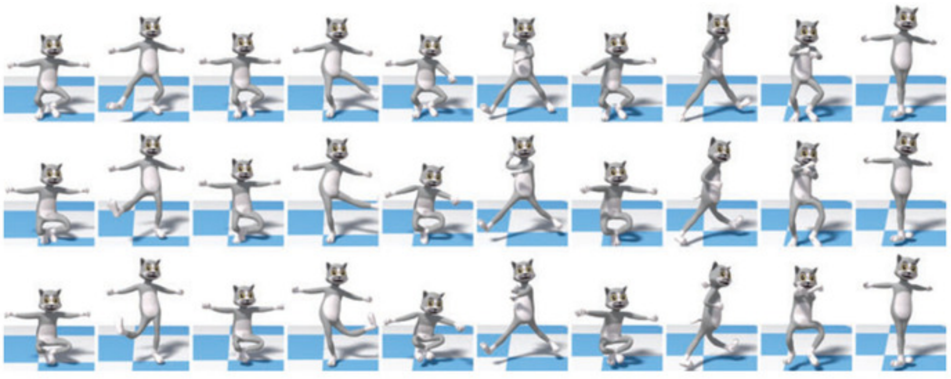

Figure 14: The Cossack dance motion of this cat character is exaggerated. Top row: original motion capture data. Middle row: applying rubber-joint group positioning. Bottom row: applying mass-spring simulation and blending

On the same topic, [Rohmer et al. (2021)] creates a cartoon-like effect by interpreting the speed of a character's rig. Their method, called Velocity Skinning, takes a standard skeleton animation as input, along with a skin mesh and rig weights. They then derive per-vertex deformations from the different linear and angular velocities along the skeletal hierarchy. By choosing meaningful deformers, they manage to mimic the cartoonlike "squashy" and "floppy" effects. Thus, they can enhance and stylize physical-looking behavior within a standard animation pipeline, for arbitrary skinned characters and in real-time (Figure 15). Animator control is supported through a simple interface toolkit, enabling to refine the desired type and magnitude of deformation at relevant vertices by simply painting weights. The resulting rigged character automatically responds to new skeletal animation, without further input. Velocity skinning enriches standard skinning with automatic stylization, without significantly impacting computational times. However, they don't preserve volume and can't ensure great deformation in extreme scenarios. Moreover, motion applied to the end effectors of a character will not propagate backward within the hierarchy, thus, they cannot achieve contact-like effects and IK-guided motion. Finally, as it depends on the character's skeleton continuity, the behavior of one 3D shape won't have any impact on another 3D shape whose skeleton isn't linked. I will present in my work an adaptation of Velocity Skinning to multiple shapes that have independent skeletons but linked behavior.

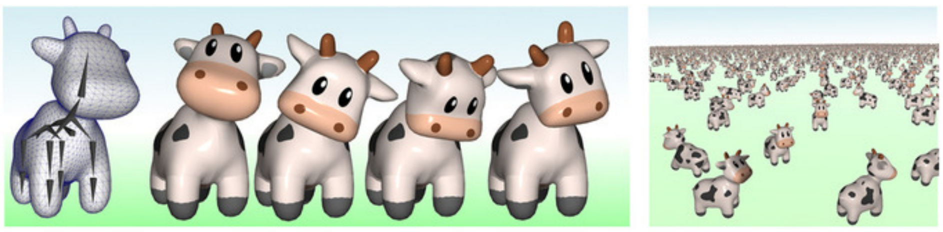

Figure 15: Left: Skeletal rig, with a single bone in the head: When animated using velocity skinning, secondary animation effects are automatically added to the ear, and face, while the horn can be set as rigid. Right: The native efficiency and simplicity of the method are compatible with GPU implementation used to compute thousands of animated cows in real-time.

2.2 Interface

All the methods presented above introduce a new way of sketching and interpreting the motion of a single 3D element. However, other tools might be necessary to ease the animation of a complete scene. For instance, to animate an octopus, one might want to sketch the motion of one tentacle and then propagate the motion to the other ones. Also, the motion resulting from the sketch could be acceptable but not perfect and the user would like to modify, re-draw or create the continuation of the trajectory.

Some articles propose smart and easy ways to interact with an existing motion. In this section, I will introduce different useful tools to speed up the animation of an entire scene. Section 2.2.1 is centered on the selection of a sub-part of an element or a scene, Section 2.2.2 focuses on motion editing, and Section 2.2.3 is centered on motion retargeting.

2.2.1 Selection

One might want to animate a small part of an element or a scene. The first step is to select the interesting part. While it seems to be an easy task, it is actually quite tricky to automatize it as, with the same input, the part to select can vary depending on what the user wants to do.

[Davis, Colwell, and Landay (2008)] propose a simple yet effective solution for selection. Objects are selected by drawing a loop around them. If 60% of a stroke lies inside the selection loop, it is selected. Then, as seen in Figure 16, a manipulator appears on top of it and allows the object to be animated in a variety of ways (translation, rotation, scaling, orient to the path...). Even though this solution sounds great in 2D, it becomes ubiquitous in 3D as the user's loop doesn't provide any information about the selection in the third dimension. Moreover, it's a manual selection that is not as efficient as an automatic one.

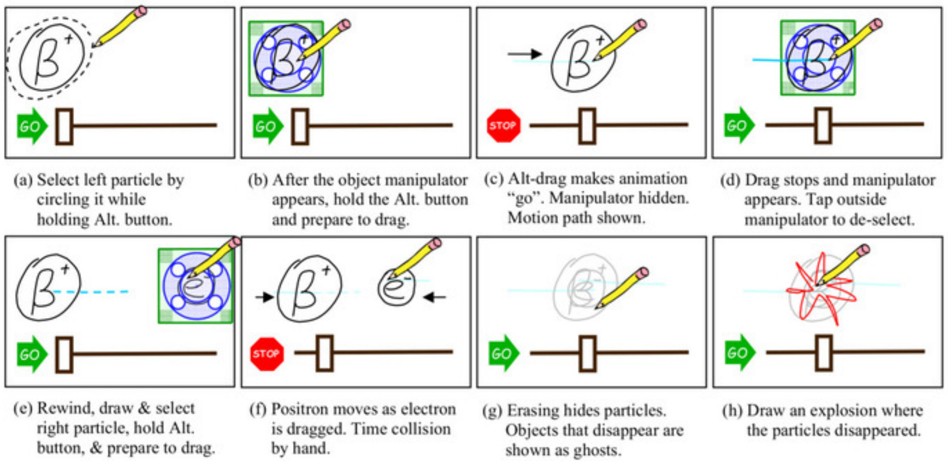
GO
STOP
(a) Select left particle by circling it while holding Alt. button.
(b) After the object manipulator appears, hold the Alt. button and prepare to drag.
(c) Alt-drag makes animation "go". Manipulator hidden. Motion path shown.
(d) Drag stops and manipulator appears. Tap outside manipulator to de-select.
(e) Rewind, draw & select right particle, hold Alt. button, & prepare to drag.
(f) Positron moves as electron is dragged. Time collision by hand.
(g) Erasing hides particles. Objects that disappear are shown as ghosts.
(h) Draw an explosion where the particles disappeared.

Figure 16: Creating a particle collision with K-Sketch

Although they also worked on modeling and animation [Jin, Gopstein, Gingold, and Nealen (2015)] suggest a semi-automatic selection by relying on a source shape (either live bone-skeleton motion data or existing animation attached to a skeleton). As explained in Figure 17, to select a portion of the model (target) for animation, as well as find a suitable skeletal chain for live motion retargeting and visual animation feedback, they find and score candidate partial matches between the source and target at various scales (LOD). The score depends on both skeletons' topology and geometry and the top match is displayed in real-time. Then, the user cycles through and selects one of a number of viable candidate matches. While this method ensures a quick and smart selection of interesting parts of the shape, the user still needs to provide initialization data, which can be tedious to obtain. Moreover, both source and target skeletons are limited to trees, i.e. connected and acyclic skeletons.

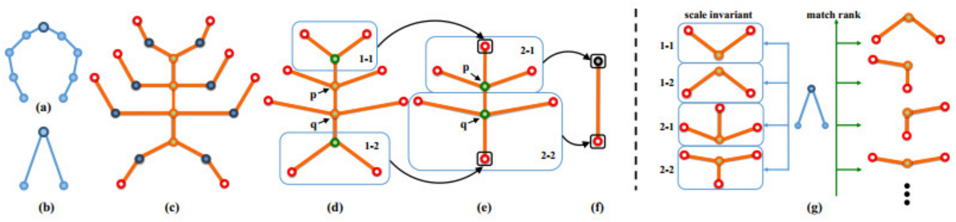

Figure 17: Finding and scoring candidate partial matches between the source and target skeletons on various scales. They use the upper body skeleton from Kinect (a) as the source, with its abstraction in (b). The LOD hierarchy of the target skeleton (c) is identified iteratively (d)–(f). (g, left) They collect sub-trees (1-1, 1-2, 2-1, and 2-2) at each iteration, and use them to find the candidate mapping with the simplified source (b) in a scale-invariant manner. Mappings are ranked using our proposed similarity metric (g, right).

Working on liquid animation, [Manteaux, Vimont, Wojtan, Rohmer, and Cani (2016)] present a semi-automatic solution to detect salient regions from which space-time features, i.e. a sequence of sub-parts of the liquid surface, will be automatically computed. The user can then easily select them using picking: a click at a specific location and time results in automatic selection of the associated feature with its full range in space and time. Feature extraction is done on a sequence of meshes without correspondences representing the liquid surface over time in three steps: detection, segmentation, and aggregation (Figure 18). Detection is conducted in a coarse-to-fine manner, through curvature analysis to automatically detect salient features, combined with topological filtering to interactively adjust the selected region. A painting tool is also available to fine-tune the selection. The segmentation step decomposes the selection into connected components, called frame features, using a straightforward breadth-first search. Finally, space-time features are created by aggregating frame features by computing a vertex-disjoint path cover of a graph representing all possible frame feature connections. Temporal coherency of the resulting paths is enforced by minimizing a geometric matching cost. The aggregation of ROI into space-time features is a key component, however, as it is based on geometrical similarities between two consecutive frames it might fail if the time step is too large or if the water body is moving too quickly.

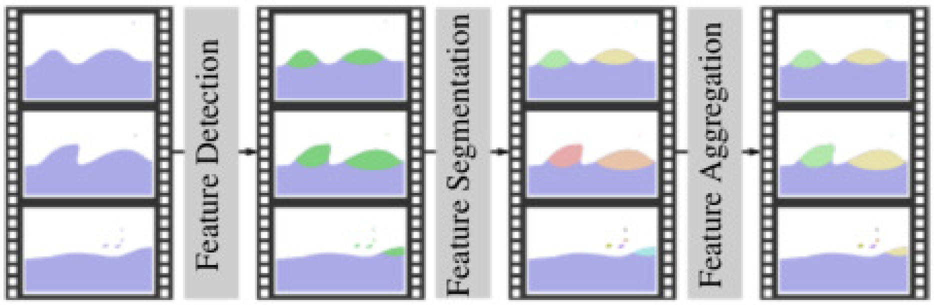

Figure 18: Feature extraction process

2.2.2 Motion editing

Even though it is easier for a user to infer a motion through sketching, it is often necessary to make several attempts or modify a correct but not perfect animation. It is therefore necessary to offer an interface allowing to interact with previously created motions and to propose tools allowing to modify them easily and quickly. Some of the previously presented articles provide such interfaces.

[Kass and Anderson (2008)] provides a familiar interactive interface supporting direct manipulation of the wiggly splines while at the same time embedding the physical realism of an oscillatory differential equation into the spline itself. The user can modify position and tangent constraints by direct manipulation of the curve, and the phase shifts can be created directly by procedural techniques or through a paint-like interface.

The resulted cycle obtained by [Ciccone et al. (2017)] is then inserted in an interactive interface, enabling the edit of a cycle both in time and space and in the current viewport using direct manipulation. As each Bezier spline maps time with position, orientation, or scale, they compute the bijective function giving the time of each curve at a specific value. Thus, they provide correspondences between time and space, orientation and scale. On their interface, the user can update the cycle point positions by direct manipulation, the orientation, and scale by interacting with their oriented and lengthed arrows and propagating the transformations all along the cycle. In addition, the user can sketch a projective line to highlight the presence of contact constraints. Finally, the method lets users update the cycle timing by moving points closer or further away which results in a smooth time warping expressed in the form of the minimization of an energy function.

Focused on the instantiation of motion, [Milliez et al. (2014)] rely on a painting metaphor to enable users to brush animated content directly on a 3D scene. During the interactive session, the user selects a motion brush, sets the brush parameters (size, spacing, and opacity), and draws a stroke that is embedded in 3D space. The repartition of the 3D content, also referred to as motion stamp, is instantiated in the local frame of the stroke relative to the brush spacing value. The user can then repaint over existing stamps to update the parameters, coordinate the parameters of one brush or stamp relatively to a predefined one, or draw additional field strokes from which a scalar or vector field is determined and map to the parameter of the chosen stamp. In complement to these spatial tools, a curve editor allows the adjustment of the animation timing as well as the possibility to arrange the ordering of the content of several brushes.

As an extension of [Guay et al. (2015b)] to more complex characters such as articulated ones, [Choi et al. (2016)] present SketchiMo, a system dedicated to the edition of a pre-defined 3D articulated motion, using sketch inputs and direct manipulation. During the editing process, the user keeps alternating between defining a sketch target and editing the motion using the three sketch spaces, as depicted in Figure 19. Four types of sketch targets are available: body line, joint path, link selected joints, or connecting end-effectors. They rely on [Guay et al. (2013)]'s bodyline definition but let the user select the desired one. Joint paths are then deduced from the extremities of this bodyline and act as visual guidance of the joint trajectory. The remaining sketch targets serve in linking the motion of two joints of the skeleton. Through the provided sketch spaces, the user can edit the global motion of the bodyline (global space), the local motion of an extremity joint (local space) or dilate the time to ease the fine editing of both these motions. In addition, they provide a brush for re-timing some sections of a path and a noise removal tool for the motion data. The multi-resolution property of their system makes it really efficient to act on different levels of motion. They rely on minimizing an energy function under constraints to update the current motion. While their interface enables them to easily alternate between different levels of details, providing real freedom in the creative process, their system is only focused on motion editing and doesn't provide the generation of new content from the user's sketch.

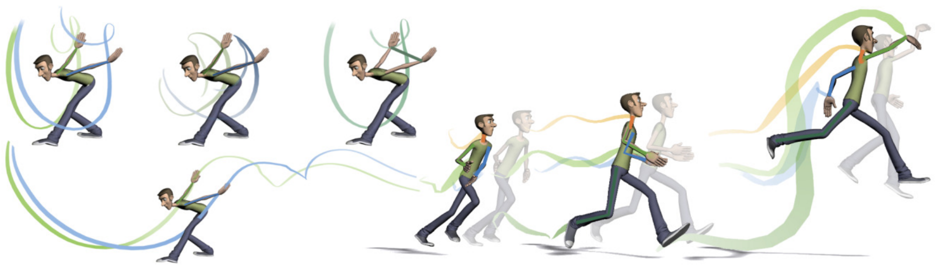
Figure 19: Left: SketchiMo offers different visualizations that accentuate different aspects of a motion: a joint path in world space (top left), the relationship between a joint and its parent (top middle), between two coordinated body parts (top right), or the temporal character of the movement (bottom). All the highlighted lines are editable via sketch input. Right: An edited dunk motion performed with SketchiMo compared with the original motion

2.2.3 Motion retargeting

When the user manages to obtain a suitable animation on a part of the object he would like to transfer it to similar objects in order to speed up the process. Furthermore, besides the fact that it takes time to repeat the same animation on all similar objects, it is tedious to reproduce exactly the same motion. Especially, organic shapes such as microscopic or marine animals often have many repeating elements like spines or tentacles. It is therefore necessary to offer the possibility to copy and paste an animation from one element to another.

In their article, [Jin et al. (2015)] focus on motion transfer between skeletons. Motion can be copied and pasted between kinematic chains with different skeletal topologies, and entire model parts can be cut and reattached, while always retaining plausible, composite animations. In AniMesh, motions are stored as rotations over time for each skeletal bone. Due to the piece-wise rigidity of a bone skeleton, orientations change only at skeletal nodes and are therefore piecewise constant along the source polyline. Figure 20 illustrates the retargeting, done by simply sampling the piece-wise linear function along the target polyline, using a unit arc-length parameterization. However, the target polyline might not be uniquely defined, as shown on Figure 21 and the choice is done manually by the user. While AniMesh is a useful tool for fast animation and retargeting, it is restrained to polylines and can't be combined with inverse kinematics. Moreover, the retargeting doesn't take into account the intrinsic rigidity of a sub-part of a mesh. It depends only on the number of bones and not on other more meaningful parameters such as material or thickness.

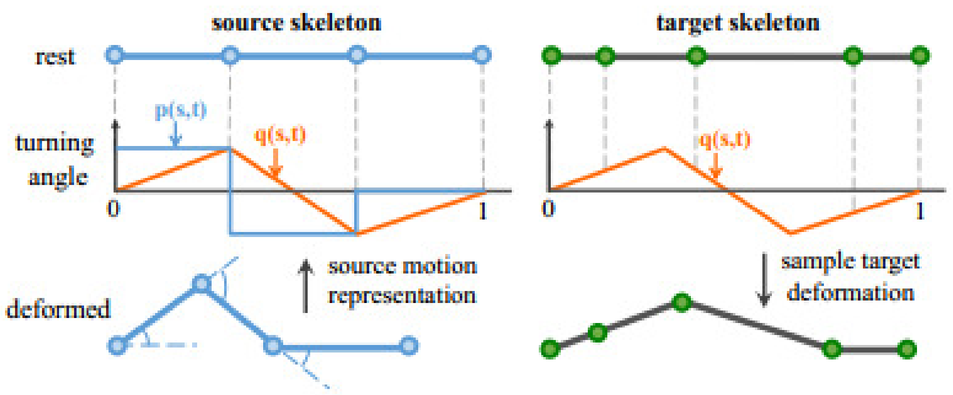

Figure 20: The source and target polylines' rest poses are shown in the top row. At time t, the source polyline is deformed (bottom row, left). Its animation (middle row, left) is represented as p(s,t) at time t, stored as the sum of turning angles for each bone, and q(s,t), the interpolation of p(s,t). The target polyline is deformed (bottom row, right) by sampling from the source q(s,t) for each bone.

.

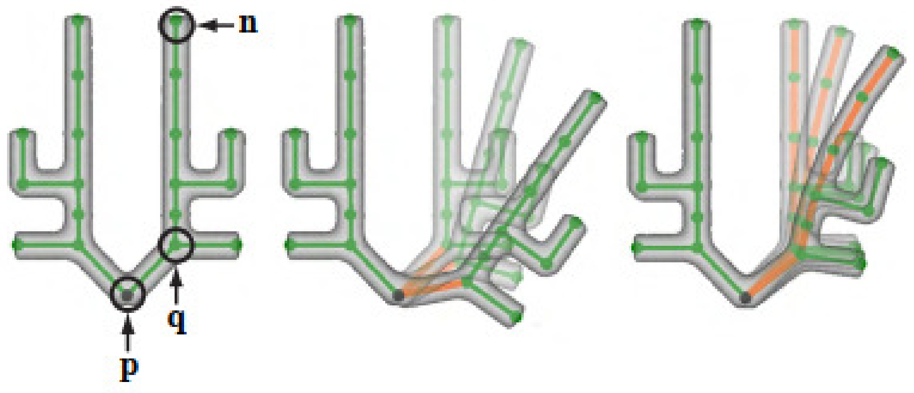

Figure 21: The animator can choose how far motion is retargeted along a skeletal chain. Left: the target shape at rest pose. Middle: the animator can terminate the skeletal chain at the sub-tree root q (the entire sub-tree rotates rigidly). Right: alternatively, he can extend the skeletal chain along the longest branch of the sub-tree to a leaf node n

.

Using space-time features presented in Section 2.2.1, [Manteaux et al. (2016)] propose an interactive sculpting system for seamlessly editing pre-computed animations of liquid without the need for any re-simulation. The computed space-time features are separated into two categories depending on whether the frame feature has boundaries (mesh representation) or not (differential representation). Mesh representation is copied and pasted by simply transforming it into an independent mesh and inserting it. Differential representation is represented using textures of vertex displacement and normals, and these are used to deform the target surface as seen on Figure 22. However, even though space-time features capture realistic behavior, the way they are edited and inserted may spoil the realism of the resulting animation. Moreover, some geometrical details may be lost when copying because of the resolution of the stamp compared to the resolution of the triangulation.

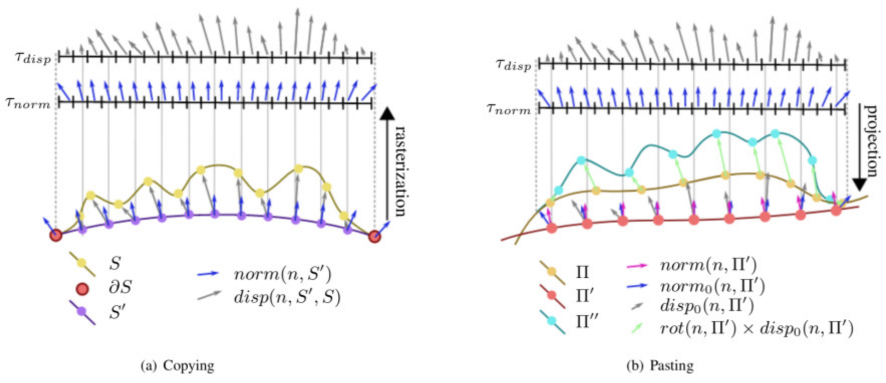

Figure 22: Left: displacement and normal maps on the original surface. Right: Insertion on the wanted surface. The final result is represented by Π''.

.

Based on a completely different approach, [Hecker et al. (2008)] present a way to infer a motion to an unknown character using authorization. The key idea is the use of two semantics to record the motion: a generalized form that is character-independent and a specialized form, used to generate the individual retargeted motions. Animations contain an arbitrary number of channels, and for each channel, the animator tells which parts of the character to select and which aspects of the motion are important. The semantic information is then transformed from the specialized space to the generalized space. At runtime, the generalized animation curves are specialized onto the characters. As shown on Figure 23, the resulting pose preserves the overall motion and stylistic details of the authored animation. Note that this method is also compatible with inverse kinematics. Furthermore, some tools are also available to modify the copy-pasted motions, such as mirroring. The problem with this approach is that the result of the authorization on other shapes may not correspond to what the user wanted, because there are many ways to describe the same movement.

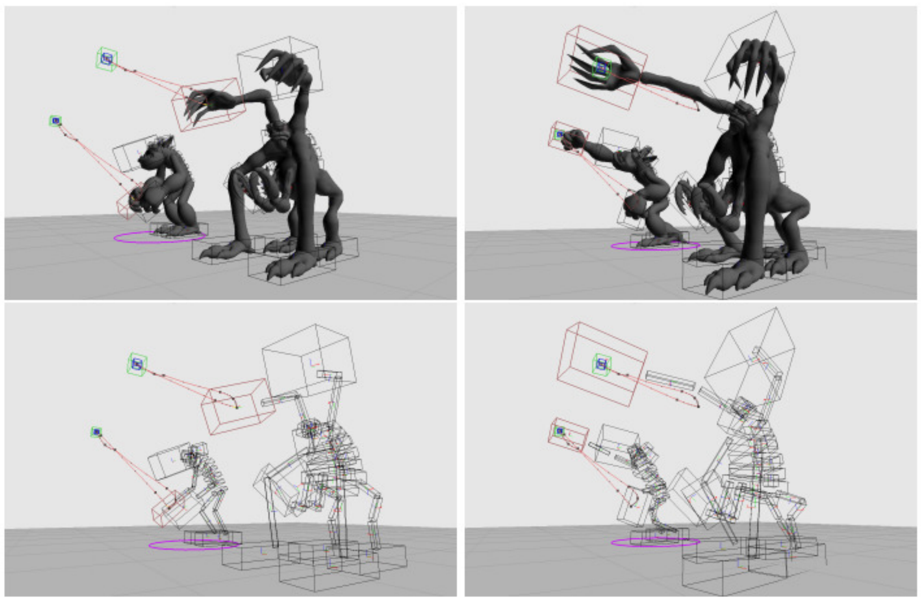

Figure 23: Two different characters posed with the same generalized input

.

2.3 Hierarchy

The hierarchical structure is widely observed in our environment. In a multi-resolution animation context, it seems natural to think about using a hierarchical pattern. It is already commonly used in computer graphics, especially when studying repetitive objects like trees. As an example, [Vimont, Rohmer, Begault, and Cani (2017)] use a hierarchical structure to create a procedural framework, called deformation grammars, enabling to sculpt hierarchical 3D models in an object-dependent manner. Object deformations are processed as symbols thanks to user-defined interpretation rules, are used to define hierarchical deformation behaviors tailored for each model, and enable any sculpting gesture to be interpreted as some adapted constraint-preserving deformation. A variety of object-specific constraints can be enforced, such as maintaining distributions of sub-parts, avoiding self-penetrations, or meeting semantic-based user-defined rules. An element, i.e. any object of the scene, can be either a simple or a complex object. A simple object is a geometric object in the classical sense, fully defined by the set of internal parameters of its visual representation (such as a triangular mesh defined by the position of the vertices and their indexing into faces). A complex object is defined recursively as a set of internal parameters, plus a set of hierarchical parameters which are references to sub-objects (children of the complex object that is the parent). Lastly, a semantic type is associated with each element. A complex object with children of the same semantic type is called homogeneous, otherwise, it is called heterogeneous. Figure 24 shows an example of a tree model using the presented hierarchy. This allows to individually consider each level of the hierarchy and the relationships between levels. To keep consistency, the deformation is also applied hierarchically. An element deformation is defined as the edit of the element's internal parameters, called the internal deformation, followed by the element deformations applied to its children, called hierarchical deformation. On the same principle, element consistency, i.e. the set of properties that an element must satisfy to be considered as valid, is also split into two sub-concepts: internal consistency and hierarchical consistency. The hierarchical structure proposed by the authors is well suited for objects that are hierarchical by nature such as trees, however, it would become more tedious for free shapes such as organic shapes.

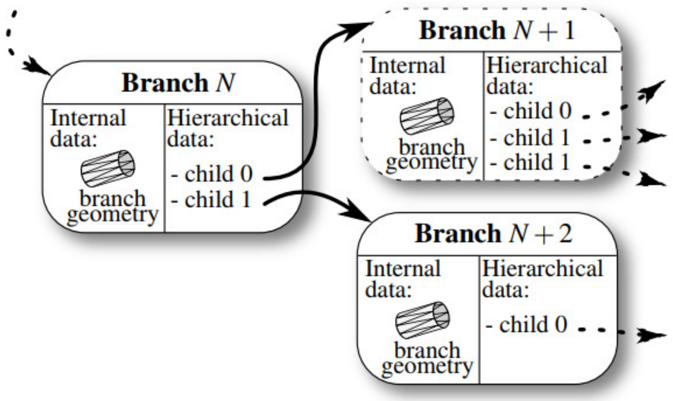

Figure 24: A tree model organized hierarchically into a complex object

[Milliez et al. (2014)], presented briefly in Section 2.2.2, use the hierarchical pattern on the animated brushes. A brush can be stored into a higher-level brush to favor coarse-to-fine animation and promote both the creation and reuse of animated content at different levels of detail. They define a motion brush as an elemental digital scene composed of geometries that move over time. The atomical content of this type of brush is created offline using the traditional animation tools to define an animation sequence of 3D content, and an additional stylization of this sample can be realized using the Overcoat 3D painting system. As illustrated in Figure 25, the user can also combine several brushes and choose which brush should be used on which portion of the strokes to be drawn. Finally, the hierarchical structure enables the instantiation of complex and multi-resolution animated 3D content. Providing an intuitive authoring tool that encoded a set of hierarchical animated 3D content answers an important challenge in the coarse-to-fine design of animated distributions of elements. However, in their current system, the atomical level of animated 3D content is defined offline and there is no data processing regarding potential structures in the sample distribution.

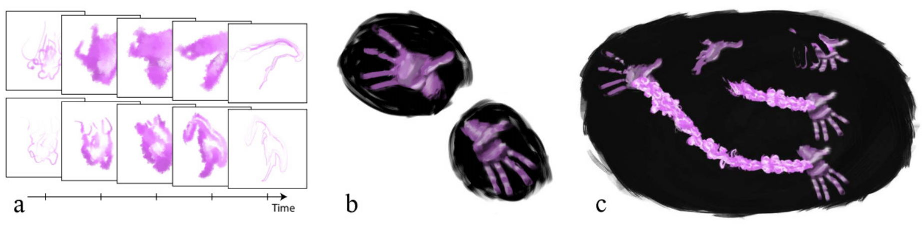

Figure 25: The user paints a set of motion brushes (a) and uses them in a scene containing two hands (b). By only painting one stroke, a complex fire effect erupts from one hand and reaches the other over time (c).

Through this detailed related work section, we have presented the existing approaches in sketch-based animation and useful editing tools. However, the current state-of-the-art methods is not sufficient for our objective of multi-resolution animation of organic shapes. Indeed, in most cases, the animations instantiated by the user are independent of each other. However, in the case of an anemone, for example, we would like to be able to give a movement to the tentacles as well as the body of the anemone while keeping it realistic. [Rohmer et al. (2021)] allows the motion of the tentacles via the movement of the anemone's body using velocity skinning, but the resulted motion is not controllable. Conversely, [Ciccone et al. (2017)] allows the creation of animations for both parts but they are uncorrelated and this can lead to synchronization problems. In the following Section, I will present my method for multi-resolution animation. Section 4 will present my results and they will be discussed in Section 5. Section 6 will conclude my work.

3 Method

The scope of my research is to animate organic shapes such as marine animals and microscopic forms using sketch-based inputs. By studying different species, a general structure can be identified. Indeed, the organism is often composed of the main body, limbs on the outside, and sometimes parts on the inside (Figure 26). As shown on Figure 27, the external and internal parts can also have sub-parts, which justifies the use of a hierarchical pattern. Firstly, I will explain in more details in Section 3.1 the structure that I use in my system. Then, I will introduce the animation process of a single object in Section 3.2. In Section 3.3, I will present method to keep coherency of the animations between different levels in the hierarchy. Finally, Section 3.4 will focus on the features available to modify the animation of the entire scene in a consistent manner.

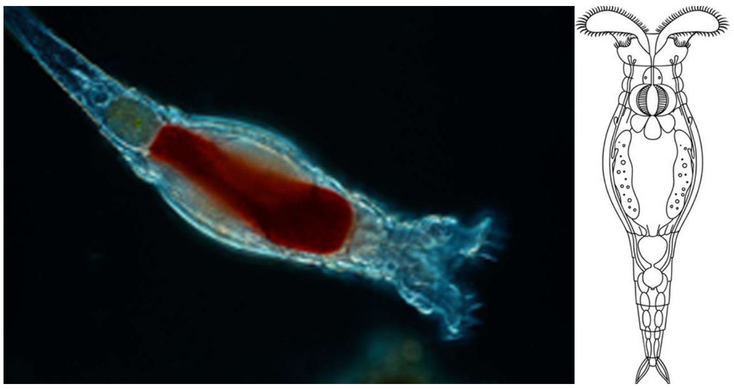

Figure 26: Photo of a rotifer with its schema. You can see both small tentacles on its head and limbs inside its body.

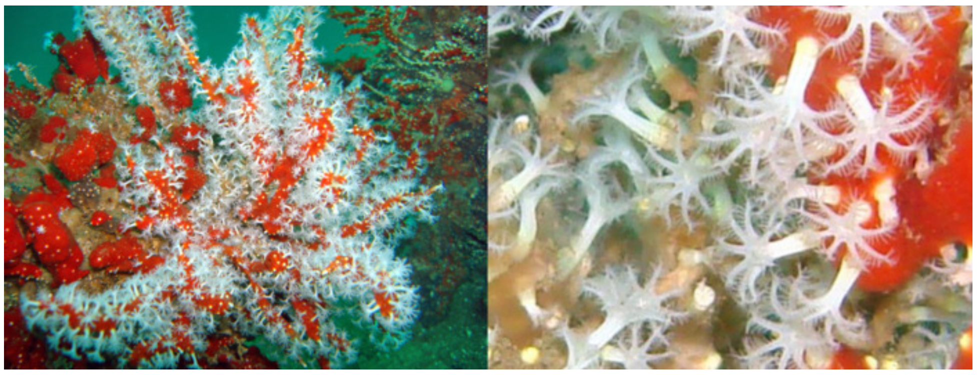

Figure 27: Photo of a coral with a close-up on its outside limbs.

3.1 Hierarchical Structure

As explained in previous sections, due to the intrinsic structure of organic shapes, the use of a hierarchical pattern to animate them is relevant. In fact, most motions can be separated into several deformations of subparts of the global shape. For example, when an anemone moves, we can separate its global motion into a deformation of its body and an individual response of each of its tentacles, ending up with one deformation per body part. To this end, we establish a hierarchical framework by dividing the shape into multiple *objects*, following rules based on what we can observe in our environment.

1. An *object* is represented with a linear skeleton.
2. Its deformations are either bending, contraction-dilation, or a combination of both and are repeated in a loop.

Therefore, we can restrict the representation of a shape to a combination of parts (in the case of anemone, one part for the body and one part per tentacles) that can be represented with a linear skeleton and have their own cyclic motion: either bending, contraction-dilatation or a combination of both. Each skeleton is attached to its proper mesh with skinning weights, and the deformation of the mesh is done with LBS, using the deformed skeleton. Contraction-dilatation is an important topic but I chose to focus on bending and let the second possible deformation for future work. Moreover, the deformation of the creatures is not done at constant volume but I consider that this is taken into account in the contraction-dilation motion. Consequently, the bending is done by trying to keep the volume constant. Even though LBS doesn't exactly preserve it, the deformations are simple enough to make the assumption that the volume is preserved.

The main body of the creature represents the root of the hierarchy and will be referenced as *root*. It has inner and outer limbs, defined as *details*, that are represented as the children of the object it is attached to in the hierarchy. A *detail* must follow the rules described above, have no branches other than those of its respective children, and can be either "inner", i.e. inside its parent, or "outer", i.e. outside its parent. As I did not have time to explore internal details in more depth, in the following I will use *detail* to describe an outer detail. A *detail* can also have children. An example of the hierarchy of the coral shown in Figure 27 is given in Figure 28.

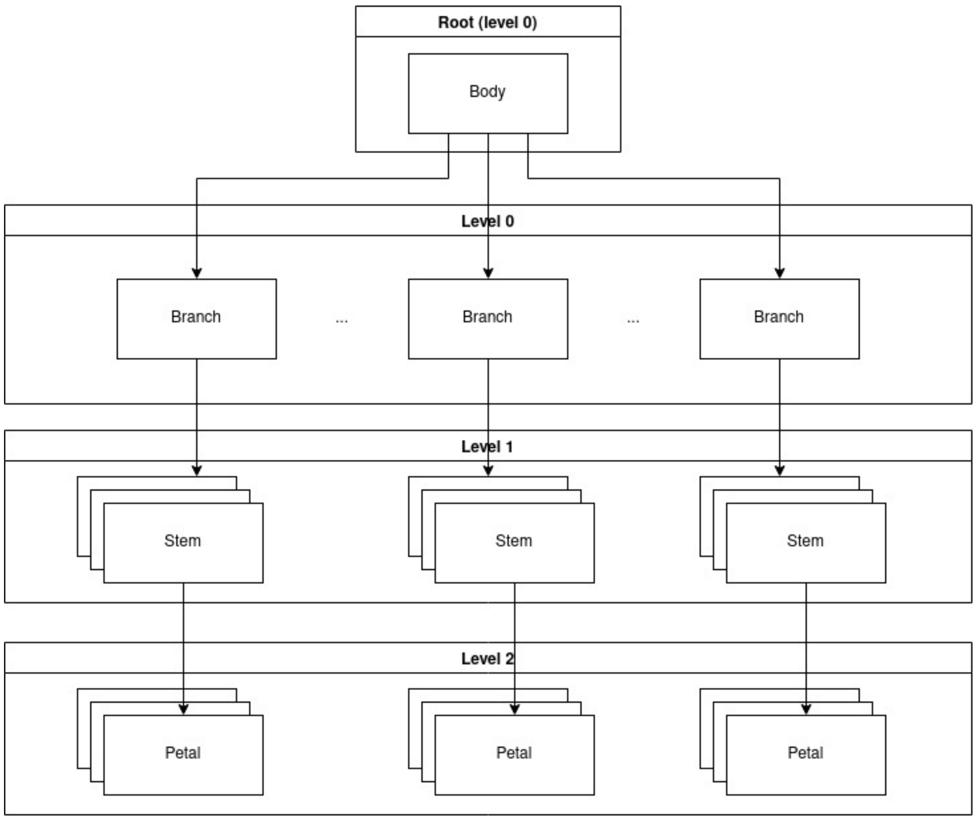

Figure 28: The hierarchy of the coral shown in Figure 27.

A scene is now composed of multiple objects that have children and one parent (except for the *root* which doesn't have parent). The particularity of this approach is that one shape can be described with multiple objects, hence multiple skeletons (one per object) that are not linked together. This way of presenting the scene leaves a lot of freedom since we can animate shapes that are not attached (like the inside of the rotifer Figure 26) as well as attached shapes (for example the tentacles of the anemone). The disadvantage is that it is necessary to preprocess the input mesh so that they meet the required constraints. Throughout my internship, I assumed that the skeletons had already been simplified and that the scene has been divided in multiple objects following the rules presented previously.

As the skeletons of each object are not linked, it is necessary to find a point of correspondence between a *detail* and its parent. Indeed, if we translate or deform the skeleton of the parent, its child should remain attached to its surface when necessary. To this end, we link each skeleton root to the closest vertex of its parent, which we call *correspondence*. By storing the translation and rotation offset between the root bone and its *correspondence*, it is easy to determine the rigid motion of a *detail* from the deformation of its parent.

3.2 Object animation

In this Section, we will consider as an *objet*, a part of a shape that is represented by a mesh with a linear skeleton and skinning weights and without branching as described in the previous section. It can be either a *detail* or the *root* of the shape.

In Section 3.2.1, I will introduce how to interact with the object through sketching as well as how the processing of the input is done. More complex methods such as cycle generation and path removal will be explained in Section 3.2.2. In Section 3.2.3, I will present my method to procedurally bend an object. Finally, in order to add dynamics to the bending and to allow more expressive animations, I propose an adaptation of velocity skinning in Section 3.2.4.

3.2.1 Sketch-based motion processing

Based on the methods used in the articles presented in the state of the art [Dvorožňák et al. (2020), Ciccone et al. (2017)], the user input is a simple 2D sketched line that gives as the output a cyclic motion, derived from the projection of the 2D line into the 3D space. Because of the difference of the number of dimension between the input and the output, the projection from the 2D space to the 3D one doesn't have a unique solution. Similarly to the articles, I solve the problem by working in the view plane.

As people are more used to representing a trajectory relative to an object, the input is drawn and stored in the space local to the object. The user selects one of the joints of the skeleton and moves on the canvas. A 2D stroke will be drawn based on the mouse trajectory and the object will be deformed as described later in Section 3.2.3. When the user raises the mouse, the drawn trajectory is saved and the object follows a cyclic motion based on the input.

A trajectory is stored as follows: when the user clicks on the screen, the current time is recorded as a reference. Then, when he draws, the position of the mouse, projected into the view plane, as well as the time, relative to the referent time, are stored. Therefore, each trajectory has its own timeline, independent of the others and the speed of the animation depends on the user drawing (Figure 29). The choice of not having a shared timeline is due to the fact that we are not trying to make a linear animation, like the editing of a movie for example, but rather to animate small parts that will run in a loop, in order to highlight particular behaviors.

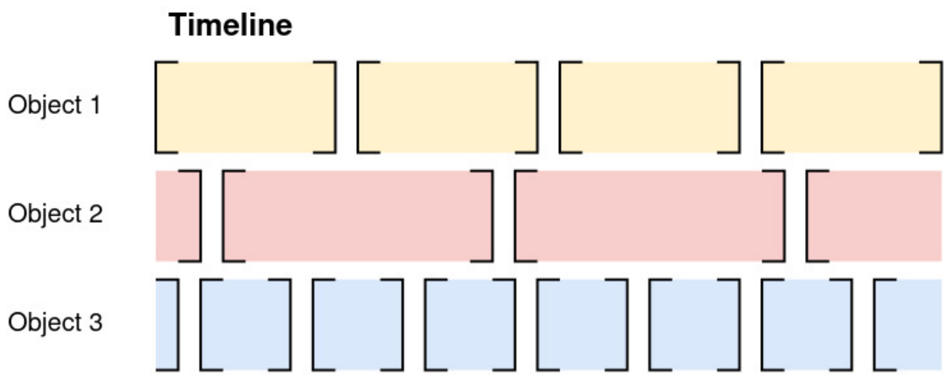

Figure 29: A colored rectangle represents a cycle. Each object evolves in its own timeline.

Once the trajectory is drawn, several preprocessing steps are applied in order to clean and store the user's trajectory correctly. First of all, a filtering step allows to smooth the curve. Indeed, the user's drawing can sometimes be shaky and create artifacts on the deformation. Making a simple average on the neighboring positions and timings is enough to remove the artifacts. Then, a retiming step is conducted so that the trajectory uses a fixed timestep of 16ms. Finally, as I work with periodic motions, I duplicate and mirror the trajectory in order to avoid jumps when the cycle starts again.

3.2.2 Generation of clean cyclic motion from dirty inputs

Some motions can be tedious to draw perfectly and give the system an idea of the motion, followed by an automatic interpretation of the input might be more relevant. For example, to do a jiggle movement, one might want to draw the cycle multiple times to give the "idea" of the jiggle, then the system would automatically generate the trajectory. To this end, the principle is to extract the different cycles. We assume that the input goes back and fourth (which seems coherent because of the cyclic aspect of the animations). We separate the cycles into different curves by studying the angles formed by two points on the trajectory and the root of the object. Each local maximum indicates the end of a way (or a way back). Then, for each point in each curve, I look for a correspondent in the curve whose angle is the global maximum, by taking the closest point. I end up with several clusters of points, which I average to obtain the final cycle. This method works relatively well when the input is sufficiently clean but quickly gives bad results in other cases. Indeed, the average is very sensitive to outliers and sometimes a cluster can contain only one point which leads to averaging with only one value (which is most probably an outlier). A solution could be to group the values alone with the closest clusters and to take the median rather than the mean. Finally, I don't currently analyze the timings so the final timings are simply those of the maximum angle curve. We could use the timings to find better matches (minimizing the distance and time difference) and also average the timings. The entire process is depicted in Figure 30.

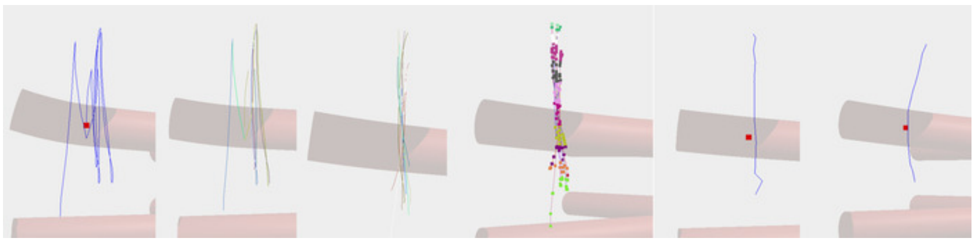

Figure 30: From left to right: Initial path, Curve extraction, Curve centering around the barycenter, Clusters, Auto-generated cycle, Effector path. The colors are used only to better see the groups but they are chosen randomly in each picture.

Another method for cycle generation could be to study the timing of each curve and to deduce a frequency. We would take as extremums the average of the first and last points of each exctracted curves. This ensure to have a clean curve, however, we lose the speed variation induced by the user's input.

A second tool is available to delete the possible unwanted part at the beginning of the path. Indeed, when the user starts to draw, the recording begins immediately. However, the object might not be at the starting position wanted by the user. Hence, he has to bring it to the desired starting point before drawing its path. In this case, it is relevant to remove the excess path. Based on the same angle study as before, the first detected curve can be deleted (result displayed in Figure 31). This method only works if there is a change of direction between the excess part and the beginning of the trajectory. We could improve the tool by studying the timings because when the user arrives at the starting point he can make a small pause to mark the beginning of his trajectory. This would allow to clean up even when the path starts in the same direction as the excess part.

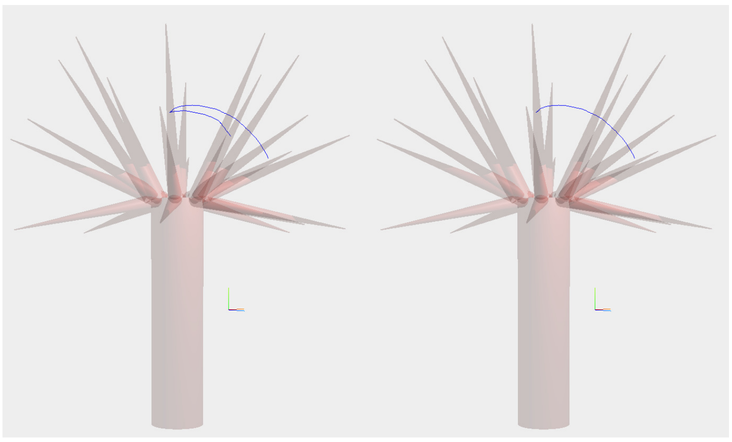

Figure 31: Left: Sketched stroke without the clean path tool. Right: Same input with clean path tool.

Finally, other tools could be added as they are useful but I have not implemented them because they do not make a contribution. By default, when the user draws, the path is immediately duplicated to create a cycle. However, if the user comes back to the starting point by himself (by drawing a circle for example) we don't want the trajectory to be duplicated. We could therefore study the distance between the start and end positions of the trajectory in order to determine whether or not it is necessary to duplicate the user input. Finally it would be interesting to be able to decelerate or accelerate the animation, either because the movement is so slow/fast (jiggle for example) that it is difficult to draw it by hand, or to better study the animation. In the first case, it is the trajectory itself that is modified (there is a transformation of the movement). In the second case, it is only a change of time step (initially 16ms), but the trajectories do not change.

3.2.3 Modeling bending deformation from the sketch

Now that the input is clean, the goal is to extract information to bend the object. As I want to keep my system as simple as possible computally speaking and in real-time, I opted for a procedural method applied on the skeleton. The mesh deformation will be achieved through a simple LBS using the skeleton positions and the skinning weights. Even though bending can be seen as a translation of the joints with respect to their parent it can also be seen as a rotation of their parent joint. I chose to work with angles because it seemed more natural to me to find a bending equation based on rotations.

By making the assumption that the rotation angle at each joint is the same, I was able to deduce an explicit formula that creates a bending deformation. The hypothesis of constant angle is debatable because the angle could actually depend on the local diameter of the mesh around the joint. However, in practice, we find that keeping the angle constant provides convincing results.

To create a bending deformation at a time t, I retrieve the 3D position (*target*) at t on the drawn curve. The goal is to bend the object in such a way that the joint of the skeleton selected by the user, called *effector*, comes as close as possible to the *target*, while using only rotations and giving a bending effect. Thus, what we are trying to do is to "project" the *effector* onto the axis formed by the root of the object and the *target*, by rotating iteratively the joints between the root and the *effector* (Figure 32).

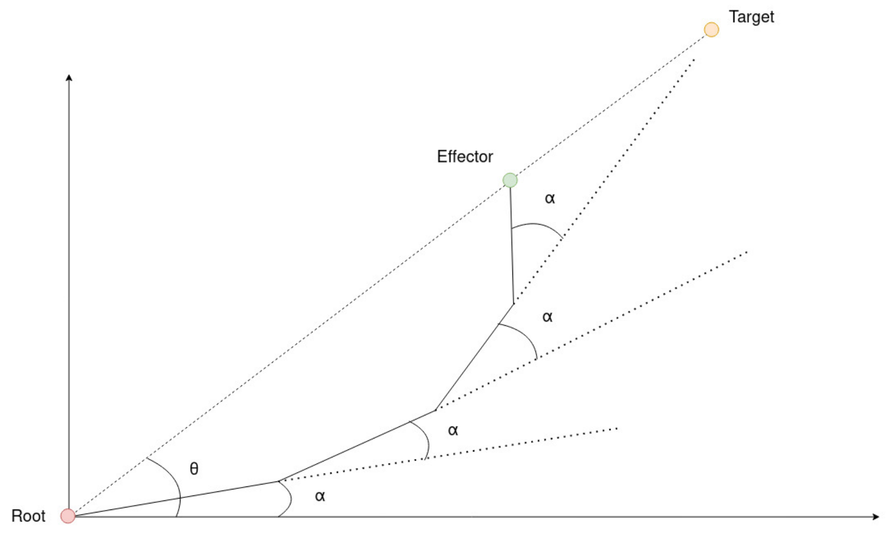

Figure 32: Representation of a bended skeleton where the axis root-*effector* is aligned with the axis root-*target*

Using the constant angle and constant volume hypotheses, we can translate it into equation 1:

$$\tan\theta = \frac{\sum_{i=1}^{N} \sin i\alpha}{\sum_{i=1}^{N} \cos i\alpha} \tag{1}$$

where N is the number of bones between the root and the *effector*, θ is the rotation between the axes root-*effector* and root-*target*, and α is the constant angle that we are looking for.

The numerator and denominator are telescoping series that can be computed as follows:

$$\sum_{i=0}^{N} \cos(i\theta + \phi) = \frac{\sin((N+1)\frac{\theta}{2})}{\sin\frac{\theta}{2}} \cos(N\frac{\theta}{2} + \phi) \tag{2}$$

$$\sum_{i=0}^{N} \sin(i\theta + \phi) = \frac{\sin((N+1)\frac{\theta}{2})}{\sin\frac{\theta}{2}} \sin(N\frac{\theta}{2} + \phi) \tag{3}$$

Using Equations 2 and 3 we obtain:

$$\tan\theta = \tan(\frac{N+1}{2}\alpha)$$

Finally the angle formula is given in equation 4.

$$\alpha = \frac{2\theta}{N+1} \tag{4}$$

This formula is applied iteratively, from the root to the leaf of the skeleton and ensures that the effector is on the root-*target* axis and gives results such as the one shown in Figure 33.

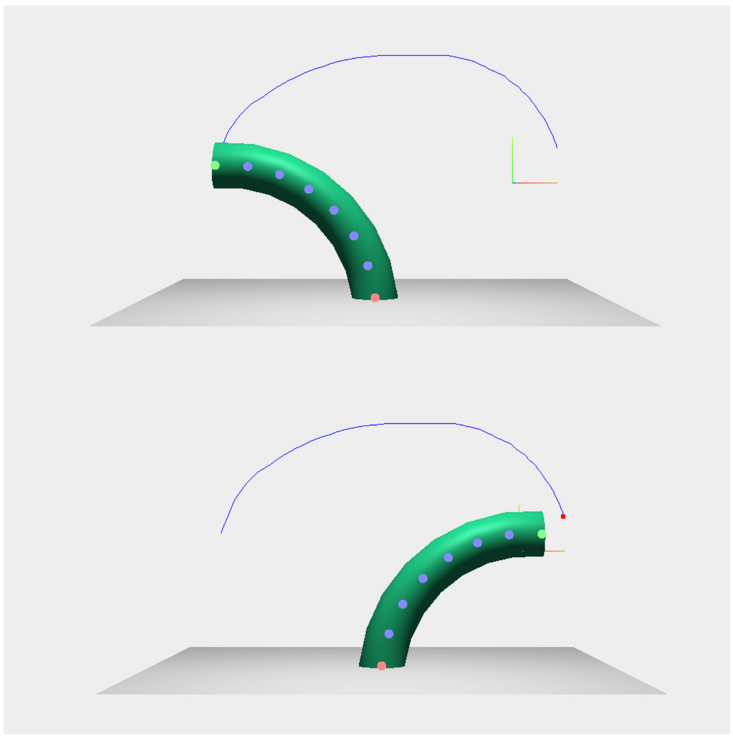

Figure 33: Result of animation of bending. The top image shows the first frame and the bottom image the last frame. The green point shows the effector.

3.2.4 Dynamic effect modeling through adapted velocity skinning

Even if the presented method is convincing to represent most of the bending deformations, it lacks dynamism. For example, when an anemone undergoes a marine current, we observe a movement of inertia at the level of its tentacles. However, we can't reproduce this "water current" effect with bending only. For this reason, I decided to adapt velocity skinning on the *effector*'s descending hierarchy, to add expressiveness.

First of all, [Rohmer et al. (2021)] apply velocity skinning on the vertices of the mesh. However, as my shapes are mostly symmetric, my skeletons are linear and each joint moves the same way, it is sufficient to apply it to the skeletons and use the speed at the joint rather than propagating speeds along the skeleton. Finally, as the bending is created from a change of angle, I chose to compute an angle instead of a position displacement. The angle is computed from the linear velocity of the joint whose formula is :

$$v_i = w_i \times (p_i - p_0)$$

where v_i is the linear velocity, w_i the angular velocity (illustrated in Figure 34) and p_i the position at bone i, and p_0 the position of the root of the skeleton.

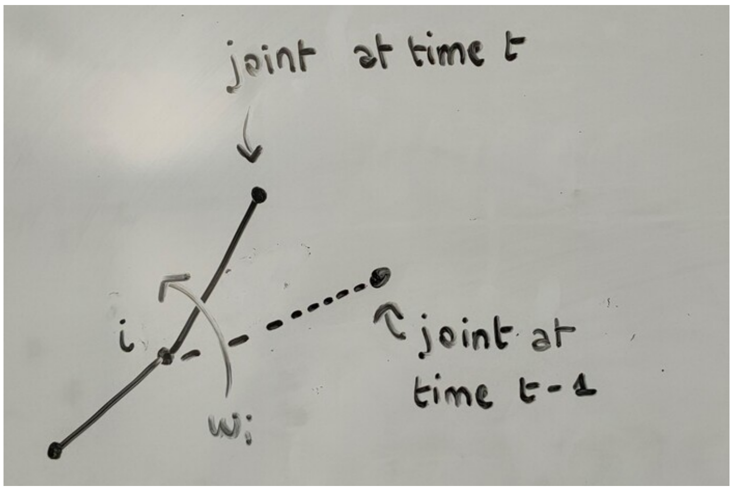

Figure 34: Illustration of angular speed w_i at bone i.

The angle is then retrieved using the linear velocity (Equation 5).

$$\theta = -k||v_i|| \tag{5}$$

where k is a parameter.

Even if the result is not based on physics, we observe on Figure 35 that the final deformation is plausible (the results will be further discussed in Section 4). Moreover, my contribution being not focus on the deformation of an object but on the interaction between different animated objects, this simplified version is more than sufficient.

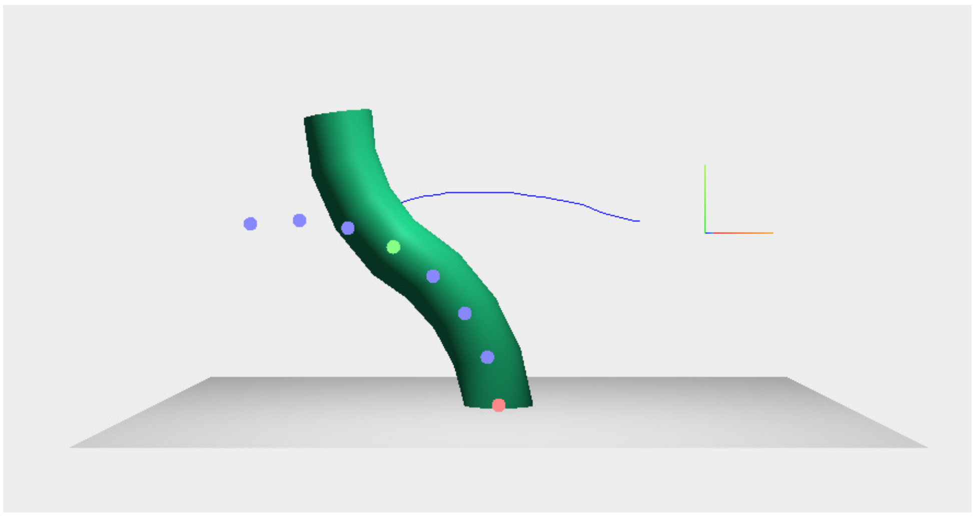

Figure 35: The skeleton shows the original bending while the mesh represents the bending with velocity skinning

3.3 Hierarchical coherency

At all levels in the hierarchy, a *detail* can have its own intrinsic motion, as we want a multi-resolution animation. However, because it is attached to an also moving part, its own motion should be coherent with the parent's one. Coherence might be subjective to the user because it depends on what he has in mind but it can be characterized by the fact that if a *detail*'s motion is very similar in terms of frequency to its parent's motion, they must be in phase at the ends of their trajectory, i.e. they reach their extremums at the same time. However, as individual motions are induced independently from each other by the user it is hard to perfectly manually synchronize them, and slight phase shifts can be observed.

I propose two solutions to this end:

1. Switch between its *own motion*, i.e. the one drawn by the user, and an automatic motion simulating the inertia induced by the parent motion, depending on the speed of the *correspondence*.

2. Synchronize its *own motion* with respect to the behavior of its parent so that it seems realistic by automatically modifying the user input.

In the first method, the idea is to combine the motion given by the user (*own motion*) with the motion induced by the parent, by making an interpolation between the two deformations, whose parameter α depends on the speed of the *correspondence* (Equation 6). When the parent's motion is slow, we can keep the user's input because the parent doesn't induce inertia on the *detail* motion. However, when the parent's motion is fast, one must see a response on the *detail* deformation, to keep the coherency. The inertia is obtained by replacing the manual deformation given by the user with an automatic one. That way, even though the motions are not synchronized, the automatic motion takes over the input motion when coherency is required. This gives a lot of freedom to the user who can give any kind of motion to an object without worrying about synchronization but he has less control on the *detail* motion when its parent moves fast since the simulation will take over.

$$q_i = q_{o_i}(q_{o_i}^{-1} q_{s_i})^{\alpha} \tag{6}$$

where q_i represents the quaternion of the bone i of the final deformation, q_{o_i} the quaternion of the bone i of its own motion deformation and q_{s_i} the quaternion of the bone i of the deformation induced by the motion of its parent.

To obtain automatically the deformation of the *detail* induced by its parent, I tested two approaches which will be explained in the following sections: the use of a mass-spring system (Section 3.3.1) and an adaptation of velocity skinning to hierarchical structures (Section 3.3.2).

The second method that I propose to keep coherency between the animations is to automatically synchronize the user-drawn *detail* motion with the one of its parent. This method will be described in more detail in Section 3.3.3. Unlike the first process which keeps the user's input intact, this one modifies it. This leaves less control to the user over the input of the *detail* trajectory but ensures that the movement looks plausible.

3.3.1 Automatic motion through mass-spring system

The first attempt to create the automatic motion used in the solution based on a switch between user input and automatic deformation was done by using a mass-spring system. This ensures that the children's response to the parents' motion is physically realistic. To do this I created a particle at each node, to which I imposed a length constraint of the size of the bone between the two neighbors and angle mass-springs at each joint to keep the simulation close to the skeleton shape. The system has drag forces but I assumed that there was no external forces such as gravity. This gives the following formula:

$$F_{d_i} = -\mu v_{d_i} + k_1(||p_{d_{i+1}} - p_{d_i}|| - L)\frac{p_{d_{i+1}} - p_{d_i}}{||p_{d_{i+1}} - p_{d_i}||} + k_2(||p_{d_{i-1}} - p_{d_i}|| - L)\frac{p_{d_{i-1}} - p_{d_i}}{||p_{d_{i-1}} - p_{d_i}||} + k_3 \alpha w$$

where μ, k_1, k_2, k_3 are parameters, L the length of the bones, α is the angle between the axes $a_1 = p_{d_i} - p_{d_{i-1}}$ and $a_2 = p_{d_{i-1}} - p_{d_{i-2}}$, and

$$w = \frac{(a_1 \times a_2) \times a_1}{||(a_1 \times a_2) \times a_1||}$$

I also added a zero length mass-spring between the *target* position and the *effector*, to make it follow the trajectory.

The algorithm is as follows:

1. Initialization of the particles at the position of the joints of the skeleton
2. Application of forces and constraints on the particles
3. Calculation of the new positions by semi-implicit resolution
4. Application of the bending presented in Section 3.2.3 by using for each bone its own particle as the target. Therefore, we ensure a constant length of the skeleton.

Although reliable in most cases, this method poses several problems regarding my initial objective. First, a mass-spring system requires a lot of tuning, however, my goal is to allow the user to animate quickly and simply without having to spend time on parameter management. Secondly, since the mass-spring system doesn't have a unique solution, there is no guarantee on the stability of the animation. For example, if we assume that the base of the body is animated and that the details follow a mass-spring system, simply translating the body in the scene can change the response of the simulation when coming back to the initial position. Therefore, even if the method ensures a realistic result, it is not suitable in this study.

3.3.2 Automatic motion through hierarchical velocity skinning

Rather than working with a simulation that is tedious to control, I chose to use the velocity skinning method presented by [Rohmer et al. (2021)] and adapt it to a hierarchical structure. For the same reasons as the ones presented in the Section 3.2.4, I work on the skeletons and don't propagate the speeds.

Unlike the input used in the article, I work with several skeletons. Moreover, even though the velocity skinning is induced by the parent motion, it must have an effect only on the *detail*. Using the initial formula would deform the parent as well. For this reason, I modified the formula so that the root bone of the *detail* skeleton has a zero displacement. This is obtained simply by subtracting the displacement of the root of the detail from all the calculated displacements.

Consequently, the new formulas are as follows and are illustrated on Figure 36:

$$v_{d_i} = w_{d_i} \times (p_{d_i} - p_{p_0}) \tag{7}$$

where v_{d_i} is the speed, w_{d_i} the angular speed and p_{d_i} the position at bone i of the *detail*, and p_{p_0} is the position of the root bone of the parent of the *detail*.

$$\begin{aligned} d_{d_i} &= (R - Id)(p_{d_i} - p_{p_0}) - (R - Id)(p_{d_0} - p_{p_0}) \\ &= d(p_{d_i}) - d(p_{d_0}) \end{aligned}$$

where d_{d_i} is the final displacement at bone i, R the rotation matrix around the rotation axis with angle

$$\theta = -k||v_{d_i}|| \tag{8}$$

where k is a parameter.

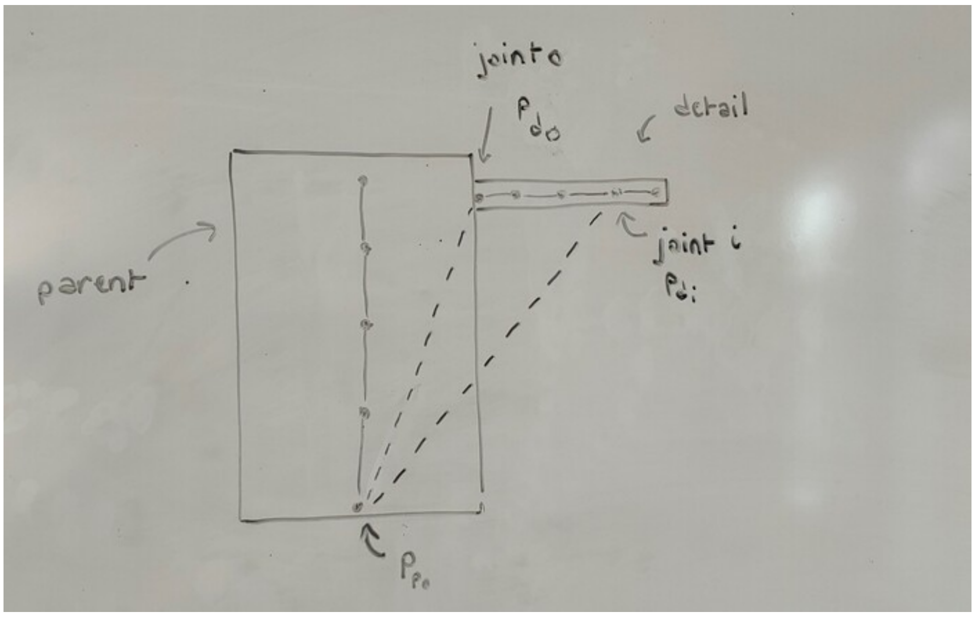

Figure 36: Schema of a detail (small branch) and its parent (big cylinder) with used annotations.

However, the result presents several problems. First, as the velocity is not propagated, the deformation is identical whatever the position of the detail is on its parent. To take it into account, I recover the index of the joint of its parent closest to the root bone of the *detail* and I add this factor to the formula. Moreover, if we use the presented formula, we observe on the left mesh of Figure 37 that the details perpendicular to the parent trajectory don't move. To make sure that the displacement of the *details* that are along the rotation axis is not zero, we add as a factor the distance between the joint of the *detail* and its root bone (right mesh in Figure 37).

Finally we obtain the following formula:

$$v_{d_i} = c_{d_i} ||p_{d_i} - p_{d_0}|| w_{d_i} \times (p_{d_i} - p_{p_0}) \tag{9}$$

where c_{d_i} is the index of the closest bone of the parent to the root bone of the *detail*.

The displacement formula becomes:

$$d_{d_i} = (R - Id)(p_{d_i} - p_{p_0}) \tag{10}$$

Although more realistic, this formula does not preserve the length of the original skeleton at all. We must therefore add a function that rescales the skeleton, followed by the bending function, applied in the same way as presented in the mass-spring system paragraph (Section 3.3.1).

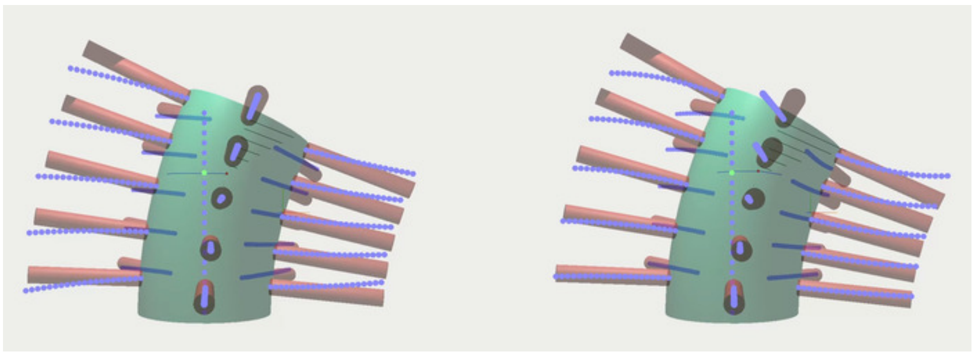

Figure 37: Left: First proposed formula. *Details* aligned with the rotation axis don't move. Right: Final formula. Every *details* respond to its parent motion.

Even if this method is not based on physics, unlike the mass-spring system, there is only one parameter to tune and for all my tests, using 25 $rad.m^{-1}.s$ for the paramter k works.

Now that I have explained the method for two levels it remains to adapt it for 3 levels or more. Indeed, when we move the *root* object only, the *details* of level 2 in the hierarchy must also react. However, if we apply the method as described above we do not get the expected result (left image in Figure 38).

In fact, we are calculating velocity skinning from an object whose LBS skeleton doesn't move hence the speed is null. Therefore, rather than taking the parent's speed as the basis, it is preferable to take as the basis the first object, when we go up in the hierarchy, that has a motion given by the user. The resut is shown on the right image in Figure 38.

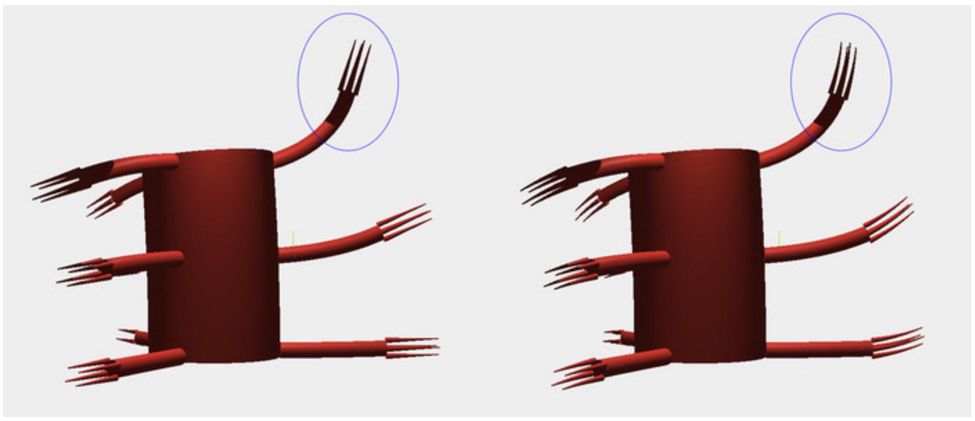

Figure 38: Left: Hierarchical velocity skinning on 3 levels without adaptation. Right: Adaptation for more than 2 levels.

This ends the explanation of the first way to keep coherency between animations. In the next Section, I will present the second method which is to synchronize automatically the different user's input.

3.3.3 Synchronize the motion of a *detail* to its parent

Rather than replacing the motion given by the user with velocity skinning when it is necessary, it would be interesting to directly modify the user input. As presented earlier, coherency is described by the fact that if a *detail*'s motion is very similar in terms of frequency to its parent's motion, they must be in phase at the ends of their trajectory. Thus, our goal is to synchronize the *own motion* of an object with that of its parent by stretching/shortening the *detail* trajectory so that its extremums are reached at the same time than the parents ones, as illustrated in Figure 39.

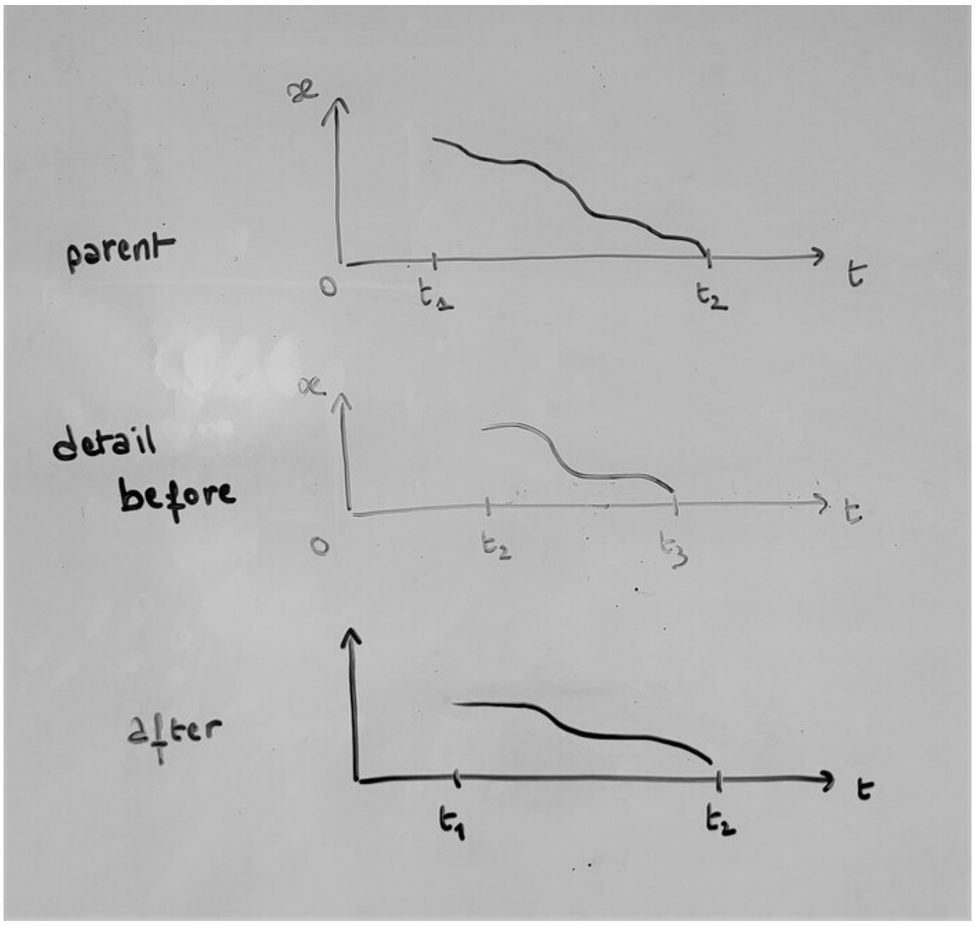

Figure 39: Illustration of synchronization in 1D. Top: The parent motion through time. Middle: The initial *detail* motion through time. Bottom: The synchronization *detail* motion.

As we are studying bending, the user input will be mainly composed of back and forth movements. The idea is to separate the sketched line into different parts, each representing either a forward or a backward movement using the same method proposed in Section 3.2.2. Once the different parts have been obtained, it is sufficient to reparameterize the trajectory of the *detail* so that the beginning and the end of each forward (resp. backward) motion takes place at the same time as the forward (resp. backward) motion of its parent. I didn't have the time to implement all this part during my internship, that's why currently the synchronization works only when the detail and its parent have only one forward/backward cycle (Figure 40).

This method allows the user to automatically synchronize the input motion and therefore gives them more freedom on the aspect of the animation. However, synchronization can sometimes greatly change the timing and speed of the input if it's very far from a synchronized version.

Another solution could be to display, while the user draws his new trajectory, either a shadow representing the motion of what's already been animated at the current time, or a point on the trajectories indicating the current timing. He could then synchronize manually more easily. However, a tool to automate the process is still relevant since it is more accurate and easy than doing it by hand.

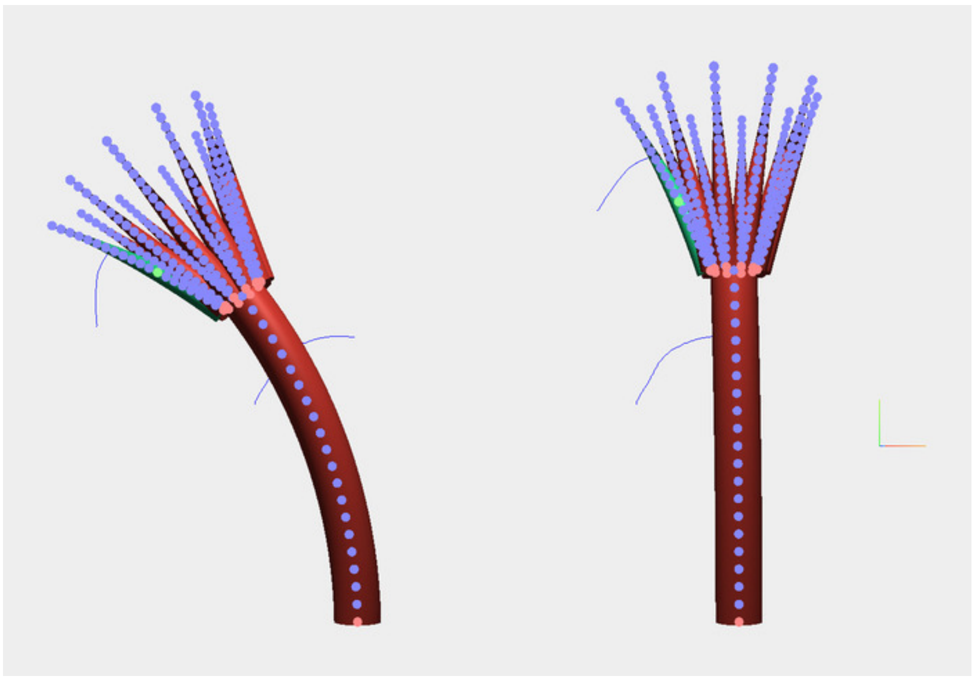
Figure 40: Left: Two deformations without synchronization. Right: The two deformations after synchronization. They arrive at their extremums at the same time.

3.4 Group behavior

We have seen in Section 3.2 how to animate an object alone. We have also studied the functioning of the hierarchy and its behavior on the animation. Now it is good to look at some relevant tools to modify the animation of a group of objects. Indeed, as explained in the introduction, organic shapes can have a lot of details, for example tentacles on an anenome. We don't want to let the user animate each tentacle one by one because it would be very time consuming while the individual movement of each tentacle is similar. Therefore, I will present four categories of tools that can be used to make an animation more complex. Section 3.4.1 will detail the copy and paste feature, then mechanisms for modifying trajectories will be presented in Sections 3.4.2 (spatial modification) and 3.4.3 (temporal modification).

3.4.1 Motion retargeting

The most useful feature of this interface is the copy-paste feature. The idea is to copy the trajectory of the first selected object, described as the *reference* and paste it onto similar objects. This feature saves a lot of time because it allows the user not to create each trajectory by hand. Indeed, in biological and marine environments, similar shapes have very generally the same behavior. Therefore, it is relevant to retarget the movement on similar parts. As the trajectories are stored locally to the object, the copy-paste is very simple, since it is enough to put the stroke in the local reference frame of the new object (Figure 41). If the two are perfectly identical, the positions do not change. In the case of an object of different size, the local trajectory must be scaled. To do this, multiply the local positions by the ratio between the size of the skeleton of the *reference* and that of the target object is enough.

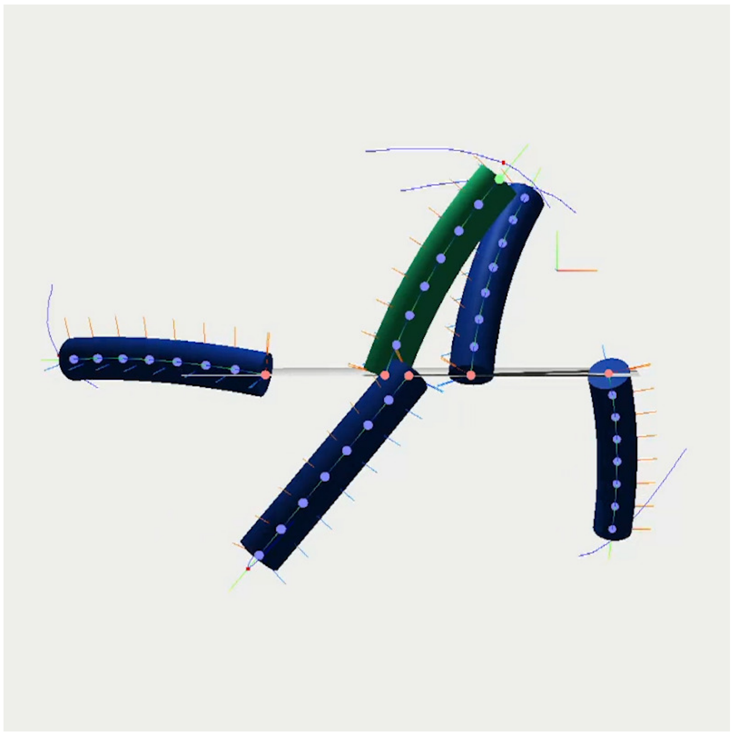

Figure 41: Retargeting the trajectory of the *reference* (green object) to every object.

Another element to copy is the *effector* joint. Indeed, even if the objects are the same size, their skeleton may be different and therefore the *effector* may not be the same joint. To retrieve the best suited joint, we use the rest position of the *reference* and we compute the distance between its root joint and its *effector*. The joint of the target object whose distance to its root joint is the closest to the computed one will be the *effector*. Finally, bending is done as presented in Section 3.2.3 and the result is shown on Figure 42.

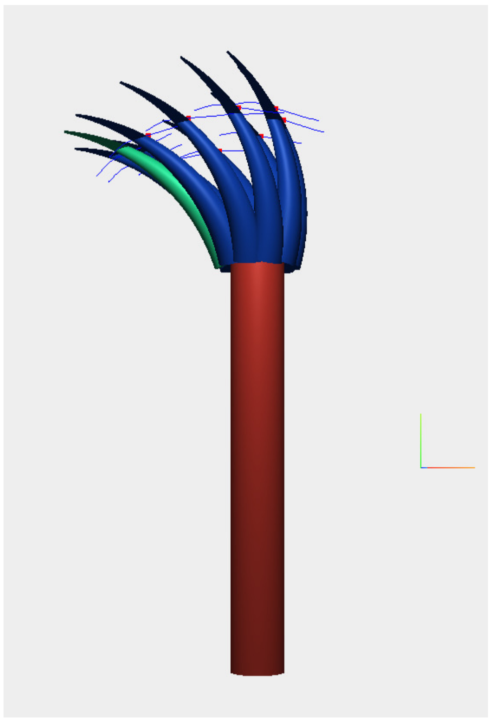

Figure 42: Tre trajectory of the green object has been pasted on blue objects.

By default the timing of the trajectory is the same and its orientation is identical in the local reference frame of the object. However, it is rare that this kind of behavior is observed in our environment. Very often, the timing varies, either randomly or following a specific pattern. The same is true for the orientation of the bending. Therefore, other tools are available in the interface to modify, not the animation of an object, but a group of motions.

3.4.2 Change orientation of a group of motions

We've seen in the previous section how to modify spatially each point of a trajectory. Now, we'll focus on spatial transformations applied to the full trajectory. By default, when copying and pasting a path, its *orientation* remains the same relative to the object's reference frame. I call *orientation* the axis formed by the first and last points of the stroke. However, very often the *orientations* of a group of objects differ. By observing videos of marine animals and organic shapes I have derived different behaviors leading to the following list of tools: local space, random orientation offset, target, global orientation, mirroring. Not all of them have been implemented, but I will present them all in this part.

Local space (or shape similarity). This is the default orientation. Each object has its own reference frame and this tool puts the trajectories of the selected objects in the same orientation as the *reference* (Left image in Figure 43).

Random orientation offset. It consists in giving a random *orientation* to a trajectory. Indeed, it is common to see bending going in all directions, such as the jiggle movements of anemone tentacles. To allow this kind of animation, each object sees its trajectory rotated by a random angle around the axis of the rest pose of the object (Middle image in Figure 43).

Target. It orients the trajectory towards a 3D point in the scene. Indeed, many of the corals and anemones that I have studied have tentacles that all move towards the same point.

In particular, the small "flowers" seen in Figure 27 have a motion directed towards its center. The user can therefore add a target point towards which all the trajectories converge. Like the other tools, the *orientation* can only be changed with respect to the object's axis at its rest pose, since the object must keep the same bending effect and mustn't expand. The angle of rotation is obtained by minimizing the distance between the last point of the trajectory and the target. Moreover, the user can select the target and move it in the scene, the trajectories' *orientations* will be updated in real time (Right image in Figure 43).

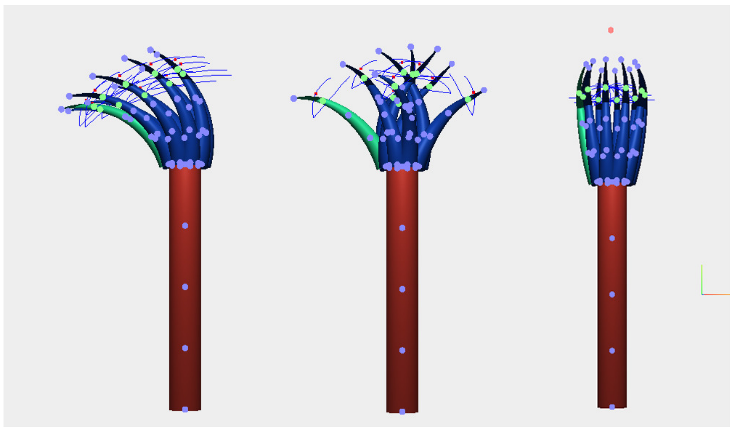

Figure 43: Left: Local space, Middle: Random Orientation Offset, Right: Target

Global orientation. Sometimes, rather than having an orientation relative to the object's local space, the user may prefer to copy its trajectory with respect to the global space. For example, if he wants to give the impression of a sea current, the direction depends not on the object's local space but on the scene's one. The tool is similar to the **local space** one but aligns all *orientations* to the one of the *reference* with respect to the global space instead of the local one.

Mirroring. It mirrors a trajectory, with respect to a plane in the scene. Indeed, many beings are symmetrical. Even if a semblance of mirroring can be obtained with the target tool, it remains very limited. This is why a specific tool can be used. Based on a plane in the scene, the trajectories can be mirrored with respect to it as explained in [Hecker et al. (2008)]. We can rely on meaningful planes such as sagittal, front and back planes.

Due to lack of time, I could not implement the last two tools presented but it would be good to add them to propose a richer and more complete system.

3.4.3 Change timings of a group of motions

Spatial modifications, although important, are not the only relevant transformations. Indeed, timing transformations are also necessary in order to generate complex and realistic animations. By default, when copying and pasting, the timings are not modified, therefore all the objects in the copy and paste will have the same speed. However, it is rare in nature to observe this kind of phenomena. Usually, either the timing is random or the gap between each motion follows a particular pattern (such as a linear offset). I therefore propose several tools to imitate this kind of behavior.

Same timing. This is the default timing, all selected objects have the same timings (Left image in Figure 44).

Random timing offset. Like **random orientation offset**, the starting timing of a group of objects can have a random offset. To reproduce this process, I shift all the timings of the trajectory by a random value (Right image in Figure 44).

Stroke offset. It creates a timing offset by following a particular pattern. For example, if the tentacles are subjected to a current, the force of the current will reach each tentacle with an offset. This pattern can be represented by a gradient or a vector field. In my case, I was inspired by the linear gradients proposed in Photoshop. In Photoshop, to draw a gradient, the user draws a line and the canvas is colored progressively from the starting color to the ending one by following the line. Even if I don't display the gradient, the concept remains similar. The user draws a curve, as he was drawing an object's path but without selecting a joint. The interpretation of this curve is done in the same way except that it is not attached to any object and, therefore, is described in the global frame. Then, for each selected object, we look for the point in the curve closest to its *effector*. The difference between the timing of the exctracted point and the first point of the curve gives the time offset of the object. The shift is finally applied like in the **random timing offset** tool (Middle image in Figure 44).

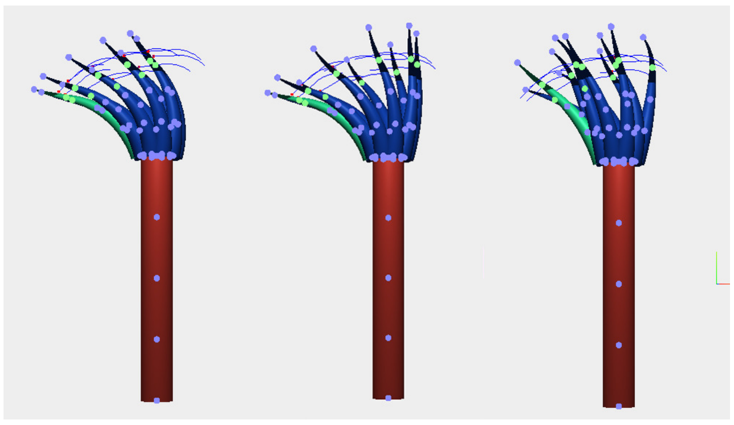

Figure 44: Left: Same timing, Middle: Stroke offset, Right: Random Timing Offset

Anchor. It makes the selected objects start at a precise time in the timeline of the *reference*. Sometimes we would like a movement to start at a specific moment of an already created animation. For example, we assume that the user has animated the body and tentacles of his anemone. But he might want the bending of the tentacles to start when the anemone has its body bent to the maximum. To do this, he can select the anemone's body as the *reference*, move to the desired timing and use the **anchor** tool. This will take as initial timing the current timing of the *reference*. Although quick to set up, I didn't take the time to implement it because I chose to focus more on the synchronization tool which seemed more relevant to my contribution.

4 Results

My system has been implemented with JavaScript and a library called Three.js. It works in real-time and is usable through a webpage (Figure 45). Even though Three.js doesn't provide an optimal solution for computational and rendering speed we can evaluate in more depth the performances, as real-time is an important criteria. To this end, I tested my system on scenes with different number of objects. I tried to have the best performance by undisplaying the trajectories and the joints of the objects. Although, the animation remains relatively fast (46,06 ms for 56 objects), we can note that the real timestep moves away from the initial one of 16ms. The consequence is that the motion that the user draws does not correspond in term of speed to the animation that he sees. Note that the system has not been optimized and Three.js is not known to be optimal on this criterion so it is possible to further improve theperformance. I conducted the performance test with an Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz x 32 processor and an NVIDIA Quadro M2000 graphic system, and the results are displayed in Table 1.

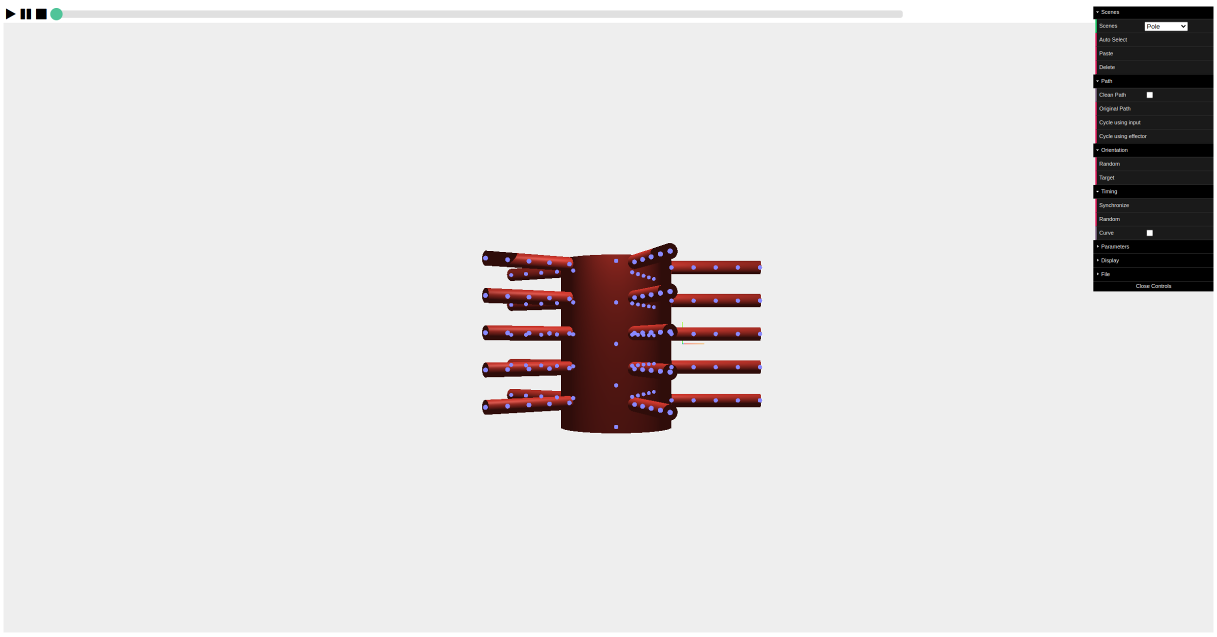

Figure 45: Proposed webpage system.

Nbr objects	31	46	56	64
Timestep (ms)	16,18	26,11	46,06	41,70

Table 1: Real timestep duration of one system call, by averaging 5 calls.

To study the suitability of my system, I propose a user study aiming at validating the need of automatic synchronization as well as examining the simplicity of inducing animations and the inputs patterns. I asked 8 participants to reproduce two movements of anemones. The objectives were to see if they were able to reproduce them, with what difficulty, how many tries and in what way.

The majority of them didn't know the system at all so they had free time to experiment it before the test. In order to save time, I did not show them all the tools, except the copy-paste one. The motions to be reproduced didn't need the use of orientation and timing tools anyway. Note that they did not have access to the synchronization tool or the hierarchical velocity skinning either. However, in case they couldn't do the wanted deformation due to the excess path explained in Section 3.2.2, I let them use the tool. They had as much time as they wanted to get used to the system and then I asked them to reproduce the animations they saw on the videos illustrated in Figure 46 and 47. I observed that 5 to 10 minutes was enough to understand the system and they reproduced each animation in 1 to 3 minutes.

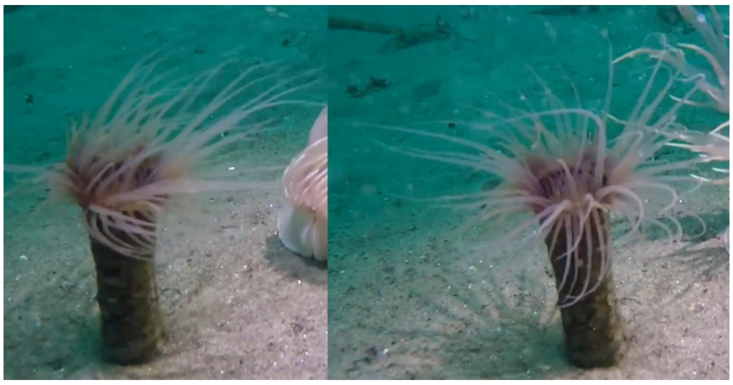

Figure 46: First video that users has to reproduce. They could only move the tentacles.

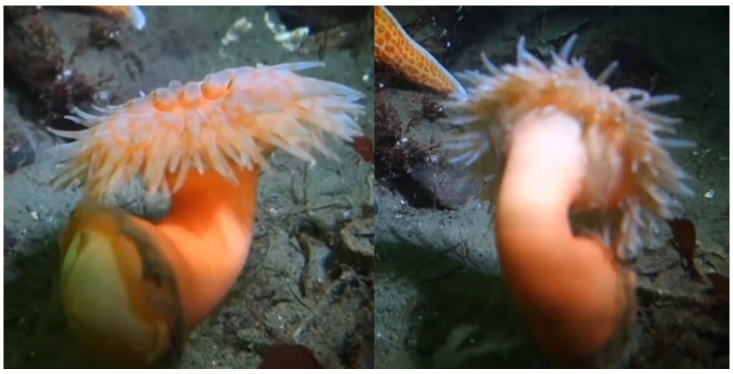

Figure 47: Second video that users has to reproduce. They had to move both tentacles and main body.

Figure 48 and49 displays the input results:

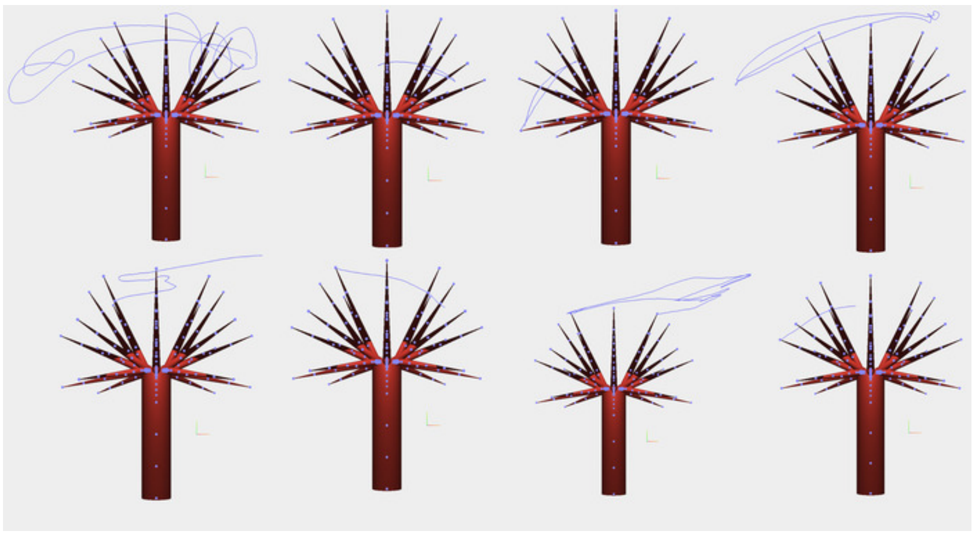
Figure 48: Input strokes of the 8 participants for the 1st test.

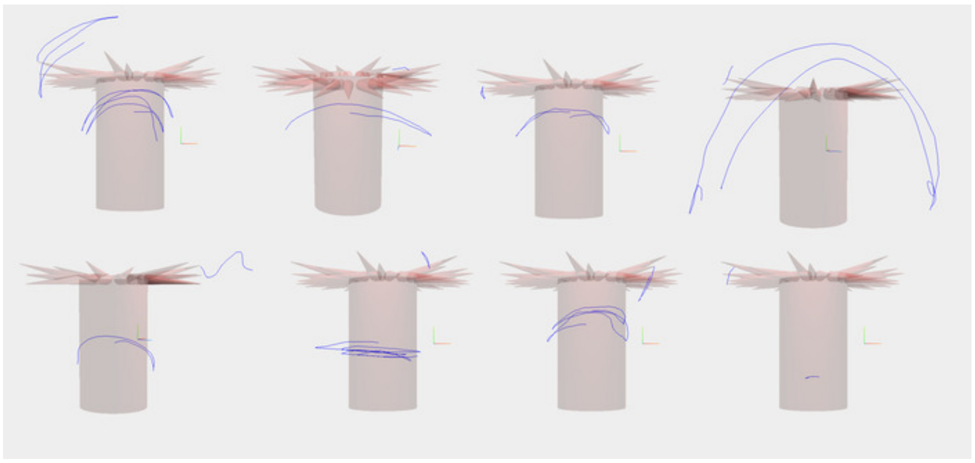
Figure 49: Input strokes of the 8 participants for the 2nd test.

Although the way of drawing is very different according to the user, all of them managed to create an animation similar to the original video with only a small amount of strokes (Table 2). Only 2 participants didn't use the final joint as *effector* to create the current effect on the first video, and 3 participants did a bending only in one side whereas the tentacles bend on both sides.

Participant	1	2	3	4	5	6	7	8
Video 1	6	14*	2	2	2	1	1	8
Video 2	7	9*	7*	9	18*	4	4	4

Table 2: Number of strokes of each participant for tests 1 and 2. * indicates the tests where participants asked for the cleaning path tool.

However, a general remark was made about the difficulty of manually synchronizing a new trajectory with the existing ones, which explains the high number of trials for some participants. This indirectly reinforces the need for the synchronization tool in the system.

On my side, I tried to reproduce more complex animations, using all the available tools (you can see orignal motion and their results in Figures 50 to 53. Although I can reproduce the movement of most of the marine animals, I was not able to animate the tentacles like the ones of the euglena which have a very particular movement.

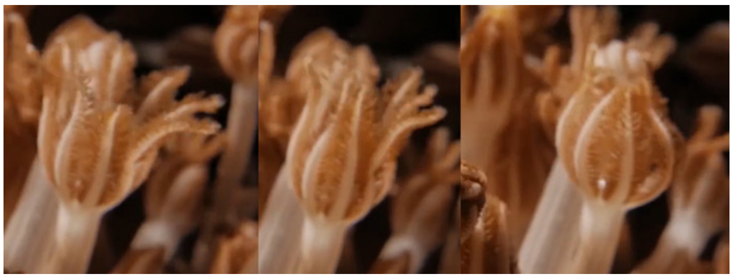
Figure 50: "Flower" motion at different timesteps.

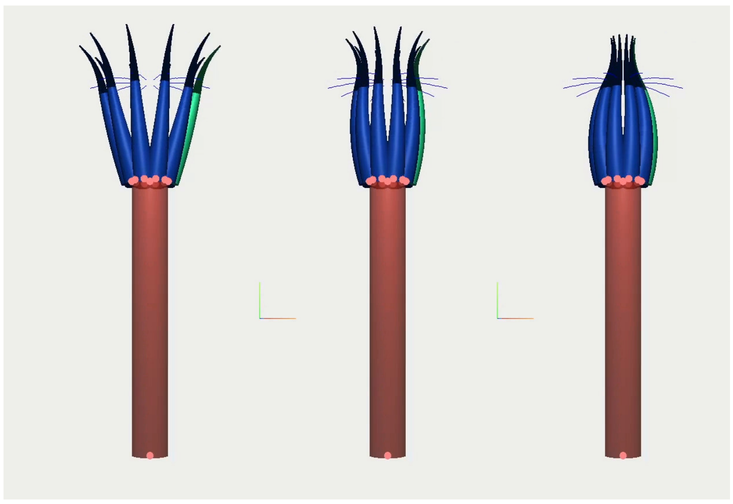
Figure 51: Result of the Figure 50 motion using the system.

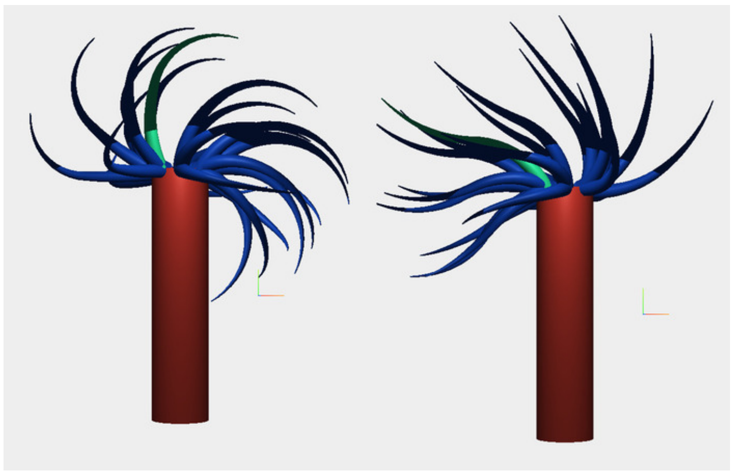
Figure 52: Result of the Figure 46 motion using the stroke offset tool.

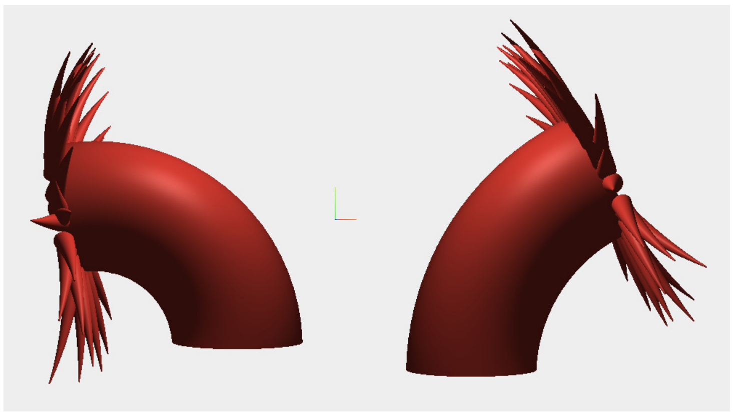

Figure 53: Result of the Figure 47 motion using the hieararchical velocity skinning (No motion was induced manually on the tentacles).

Furthermore, in the above examples it was necessary to use the velocity skinning adaptations explained in Sections 3.2.4 and 3.3.2. As explained previously, I had chosen to use velocity skinning rather than a mass-spring system because of performance and controllability issues. Although we can see that the velocity skinning effect is suitable for reproducing the movements of organic shapes, it is also interesting to compare it to the mass-spring system that I implemented. In Figure 54, we can see that the behavior of velocity skinning is very similar to that of the mass-spring. Therefore, although we do not rely on physical equations, we manage to obtain a sufficiently realistic behavior in real time.

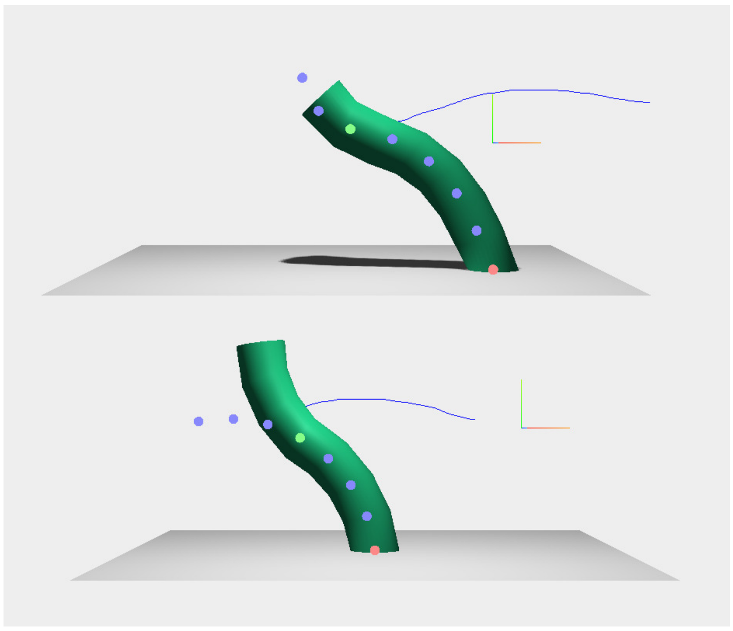

Figure 54: Top: Use of a mass-spring system for the bending motion. The points shows the particles positions. Bottom: Use of velocity skinning to add dynamism to the bending. The points shows the bending without velocity skinning

Finally, I wanted to study the user inputs and the way they draw their trajectory. Indeed, as my system is based on cyclic events, there are two ways to draw the input: 1) draw a single cycle or 2) draw several cycles and expect to have an average cycle in the output, like [Ciccone et al. (2017)]. To this end, in addition to studying whether the testers were able to reproduce the motion, I also looked at how they created it. According to Figure 48 and 49, although each user drew in a very different way, an overwhelming majority drew only one cycle. When conducting the study, I then wondered whether the result would be different if, rather than asking them to draw a trajectory, I asked them to draw a behavior. For the last 5 participants, I added a test and asked them to represent a jiggle effect by showing them a reference video. Contrary to the previous tests, the participants were more likely to draw a large number of cycles (Figure 55). Due to lack of time, I couldn't do new user studies but I came out with two hypotheses to study from this experiment: either users tend to draw only one cycle when the animation is slow and large and several when it is small and fast, or we have to separate the animations into two types, the trajectory type which represents a path followed by an object and the behavioral type which represents a behavior of the object (jiggle, bounce,...)

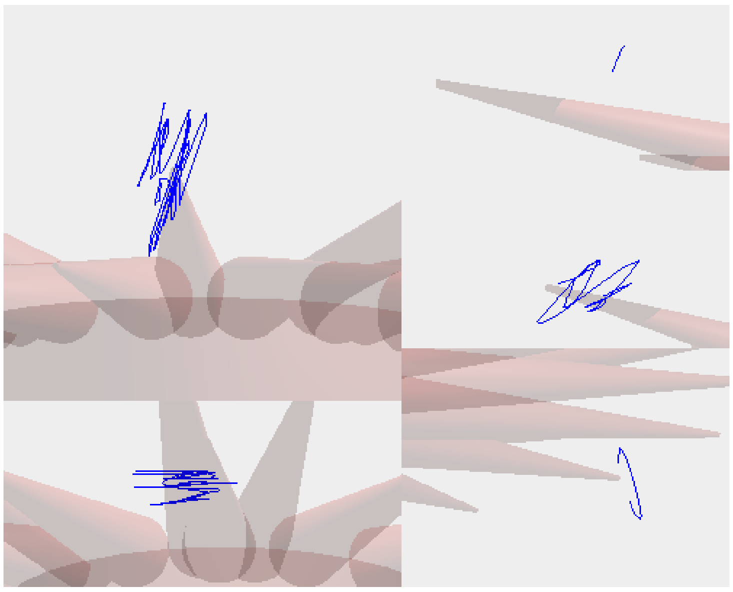
Figure 55: Input stroke of the participants when asked to do a jiggle effect.

A last study I wanted to do was to examine the real measurements and speeds of the animals I reproduce, in order to optimize the parameters. Indeed, I explained in Section 3.3 that the movement of the tentacles of the anemone follow either the trajectory drawn by the user or the velocity skinning, depending on the speed of its parent. However, it is necessary to determine the speed at which it goes from one to the other. For that purpose, it is possible to reproduce the scene at the right scale and to use the real speed from which it must switch.

5 Discussion

My system offers many tools to create complex and coherent animations in a very short time and very simply. My objectives concerning my contributions have been completed since the animation of the scene is done from sketches, has a hierarchical structure whose individual animations are consistent with each other and the whole thing runs in real time. Moreover, many tools are available to enrich the animations given by the user. However, there are still a lot of work that I didn't have time to do and many limitations.

First of all, concerning the objects, I start from already simplified and separated skeletons which will rarely be the case. In a real situation, the scene is not already separated into objects with linear skeletons, and the hierarchy has to be constructed based on their geometry. It would be interesting to add this processing in order to be able to take as input a larger panel of objects, using for example the method proposed in [Jin et al. (2015)], that can also serve for automatic selection. In my case, as my subject is part of the Creative AI project, it would be even more relevant to make an implicit surface input possible. Moreover, I only worked with *details* external to its parent whereas many organic forms such as the euglena have internal objects. Finally, the deformations I apply are reduced to bending and velocity skinning while other types of deformations such as contractions-dilatations are also observed in natural environment. Also, currrently the objects can't be translated in the scene. It would require to adapt velocity skinning to translation motions as it was implemented for rotations only.

Regarding the sketched-input, the main limitation is its 2D property. As in the works presented in the state of the art, it makes the problem ill-posed so we have to limit the trajectory to the view plane. It would be relevant to study 3D inputs in order to make the problem solvable without having to add assumptions. Moreover, we can note some limitations of the tools that modifies the sketch input. To work correctly, the tool that auto-generate a cycle must have a clean input. Otherwise, the auto-generated curve will be noisy. A solution to avoid noise could be not to average the user defined positions but to interpolate on the angles between the starting and ending positions using the mean timing of each cluster. Indeed, since the object cannot expand and follows a 2D trajectory, it is enough to have the starting point, the end point and the timings of each cluster to deduce the curve. Moreover, this function only works with inputs having a very precise shape (with forward and backward strokes). If the user draws his bending in another way, it is not guaranteed to have a suitable result. Although the user study showed that they tend to draw the trajectories with forward and backward strokes, there are still cases where the input differs a lot (input 1, 5, and 7 of Figure 46).

Furthermore, the synchronization tool currently only works when the trajectory is composed of one clean cycle (created, for example, with the tool mentioned above). It still needs to be adapted to synchronize all local extremums. One method to extend my current implementation would be to synchronize by chunk and only when necessary. The piecewise synchronization would be done using the same curve extraction as for cycle auto-generation. The idea is to synchronize the curve in which the *detail* is located with the curve in which its parent is located. Then, when both objects switch to the next curve (note that they change at the same time since they are synchronized) we synchronize the new ones. This would ensure the robustness of the synchronization; indeed, if the *detail* and its parent have a different number of input curves, the animations will gradually desynchronize. For example, if the parent has 5 curves and the child 4, we will synchronize the 4 curves of the child with the first 4 curves of the parent. However, when entering the 5th curve, the parent will again be in its 1st one. The 1st curve of the child should therefore not be synchronized with the 1st curve of the parent but with the 2nd.

On the same principle, my implementation does not ensure a cyclic animation of the scene. Indeed, as there is no common timeline and each object has its own timeline, if we start two trajectories at the same place and at the same time, nothing ensures that they will be at the same place at the same time later, because each cycle has its own duration. This limitation seems to me to be coherent because each object has its own behavior and if we want them to follow the same rhythm, we just have to use the synchronization tool.

Another limitation concerns group modifications. Currently, it is possible to modify the timings of a group of objects. However, the user might want to modify the timings of a group of group of objects. Let's take the example of the flowers on the anemone Figure 50. We want the flowers to close at different times, but all the petals of the same flower must close at the same time. It would therefore be necessary to add another hierarchical system to be able to group the trajectories of each petal and apply the timing modification tool to these groups instead of individual objects, which is not possible for the moment in my system. Also, the modification tools are currently applied using buttons. However, we aim to have a system working with only sketching inputs. Thus, I would have to think about other ways to use the proposed tools (use different brush types, draw specific shapes,...)

Furthermore, it is not possible to modify or continue a trajectory. It would be important to have a tool similar to the ones proposed by [Ciccone et al. (2017), Choi et al. (2016)] to modify the animation as well as a way to continue a trajectory because it would be possible to draw a 3D trajectory. Also being able to draw one trajectory per joint instead of one trajectory per object would make it possible to create more complex animations.

Finally, I didn't work on collisions at all even though it's a very important topic since it adds new types of deformation. It would be a good subject for further work because dealing with collisions in real time remains an open problem.

In conclusion, as future work it would be interesting to finish implementing the tools presented in Section 3.4 that have not been implemented, add mesh processing to automatically separate the scene into multiple objects with linear skeleton and to automatically deduct the hierarchy, adapt the system to shapes that have internal elements, use 3D user input instead of 2D strokes, allow modification of these inputs, and take collisions into account. The ultimate goal would be to include my system inside Matisse, he software developed within the framework of the Creative AI project.

6 Conclusion

In conclusion, I propose a multi-resolution animation system based on a hierarchical structure that allows to create cyclic motions throug sketching to any part of the shape while keeping them coherent along the hierarchy. One can animate coarsely an object and then refine little by little the animation to make it more and more rich. My contributions are the adaptation of animation methods to organic shapes that follows a hierarchical pattern, as well as the proposition of synchronization and copy-pasting features to ease the animation process. Synchronization helps to keep the coherency of the animations along the hierarchy while the copy-pasting feature speeds the animation process by retargeting a motion of an object to similar ones. Moreover, I proposed mechanisms to enrich the copy-pasting such as ways to modify the orientations and the timings of a group of trajectories. Finally, I validated the relevancy of my system by conducting a user study which confirmed the necessity of an automatic synchronization.

My work is part of the bigger projet, Creative AI, which aims at allowing the user to bring to life any idea he has in mind. Currently, it is already possible thanks to the Matisse software to model an organic shape from a sketch. It is in the process of being improved to be able to refine this shape by zooming in and drawing smaller and smaller details, allowing to model in a coarse-to-fine manner. My work is in line with this, since it allows to animate using the same hierarchical pattern.

It remains to ameliorate the system by improving the tools already present and by adding other ones necessary to simplify the animation process, as well as to integrate the system in the Matisse software.

References

Bernhardt, A., Pihuit, A., Cani, M.-P., & Barthe, L. (2008, June). Matisse : Painting 2D regions for Modeling Free-Form Shapes. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)* (pp. 57–64). Annecy, France. Retrieved from https://hal.inria.fr/inria-00336688

Choi, B., i Ribera, R. B., Lewis, J. P., Seol, Y., Hong, S., Eom, H., ... Noh, J. (2016, jul). Sketchimo: Sketch-based motion editing for articulated characters. , *35*(4). Retrieved from https://doi.org/10.1145/2897824.2925970 DOI: 10.1145/2897824.2925970

Ciccone, L., Guay, M., Nitti, M., & Sumner, R. W. (2017). Authoring motion cycles. In *Proceedings of the acm siggraph / eurographics symposium on computer animation.* New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3099564.3099570 DOI: 10.1145/3099564.3099570

Davis, R. C., Colwell, B., & Landay, J. A. (2008). K-sketch: A 'kinetic' sketch pad for novice animators. In *Proceedings of the sigchi conference on human factors in computing systems* (p. 413–422). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1357054.1357122 DOI: 10.1145/1357054.1357122

Dvorožňák, M., Sýkora, D., Curtis, C., Curless, B., Sorkine-Hornung, O., & Salesin, D. (2020, nov). Monster mash: A single-view approach to casual 3d modeling and animation. *ACM Trans. Graph.*, *39*(6). Retrieved from https://doi.org/10.1145/3414685.3417805 DOI: 10.1145/3414685.3417805

Guay, M., Cani, M.-P., & Ronfard, R. (2013, November). The Line of Action: an Intuitive Interface for Expressive Character Posing. *ACM Transactions on Graphics*, *32*(6), Article No. 205. Retrieved from https://hal.inria.fr/hal-00861503 DOI: 10.1145/2508363.2508397

Guay, M., Ronfard, R., Gleicher, M., & Cani, M.-P. (2015a, June). Adding dynamics to sketch-based character animations. In *Sketch-Based Interfaces and Modeling (SBIM) 2015* (p. 27-34). Istanbul, Turkey: Eurographics Association. Retrieved from https://hal.inria.fr/hal-01154847

Guay, M., Ronfard, R., Gleicher, M., & Cani, M.-P. (2015b, August). Space-time sketching of character animation. *ACM Transactions on Graphics*, *34*(4), Article No. 118. Retrieved from https://hal.archives-ouvertes.fr/hal-01153763 DOI: 10.1145/2766893

Hecker, C., Raabe, B., Enslow, R. W., DeWeese, J., Maynard, J., & van Prooijen, K. (2008). Real-time motion retargeting to highly varied user-created morphologies. In *Acm siggraph 2008 papers.* New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1399504.1360626 DOI: 10.1145/1399504.1360626

Jin, M., Gopstein, D., Gingold, Y., & Nealen, A. (2015, oct). Animesh: Interleaved animation, modeling, and editing. *ACM Trans. Graph.*, *34*(6). Retrieved from https://doi.org/10.1145/2816795.2818114 DOI: 10.1145/2816795.2818114

Kass, M., & Anderson, J. (2008, aug). Animating oscillatory motion with overlap: Wiggly splines. *ACM Trans. Graph.*, *27*(3), 1–8. Retrieved from https://doi.org/10.1145/1360612.1360627 DOI: 10.1145/1360612.1360627

Kwon, J.-y., & Lee, I.-K. (2008). Exaggerating Character Motions Using Sub-Joint Hierarchy. *Computer Graphics Forum.* DOI: 10.1111/j.1467-8659.2008.01177.x

Manteaux, P.-L., Vimont, U., Wojtan, C., Rohmer, D., & Cani, M.-P. (2016). Space-time sculpting of liquid animation. In *Proceedings of the 9th international conference on motion in games* (p. 61–71). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2994258.2994261 DOI: 10.1145/2994258.2994261

Milliez, A., Noris, G., Baran, I., Coros, S., Cani, M.-P., Nitti, M., ... Sumner, R. W. (2014).

Hierarchical motion brushes for animation instancing. In *Proceedings of the workshop on non-photorealistic animation and rendering* (p. 71–79). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2630397.2630402 DOI: 10.1145/2630397.2630402

Pentland, A., & Williams, J. (1989). Good vibrations: Modal dynamics for graphics and animation. In *Proceedings of the 16th annual conference on computer graphics and interactive techniques* (p. 215–222). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/74333.74355 DOI: 10.1145/74333.74355

Rohmer, D., Tarini, M., Kalyanasundaram, N., Moshfeghifar, F., Cani, M.-P., & Zordan, V. (2021, May). Velocity Skinning for Real-time Stylized Skeletal Animation. *Computer Graphics Forum*, *40*(2). Retrieved from https://hal-polytechnique.archives-ouvertes.fr/hal-03195315 (Proc. Eurographics)

Vimont, U., Rohmer, D., Begault, A., & Cani, M.-P. (2017). Deformation grammars: Hierarchical constraint preservation under deformation. *Computer Graphics Forum*, *36*(8), 429-443. Retrieved from https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13090 DOI: https://doi.org/10.1111/cgf.13090

Xing, J., Kazi, R. H., Grossman, T., Wei, L.-Y., Stam, J., & Fitzmaurice, G. (2016). Energy-brushes: Interactive tools for illustrating stylized elemental dynamics. In (p. 755–766). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2984511.2984585 DOI: 10.1145/2984511.2984585