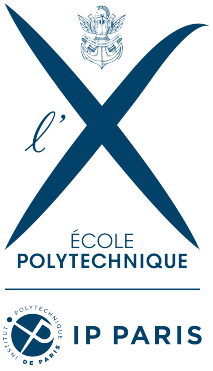

École Polytechnique

BACHELOR PROJECT RESEARCH LAB IN COMPUTER SCIENCE

Extending and prototyping Dynamic Skinning Deformation

Author:

Andreea Pătărlăgeanu, École Polytechnique

Advisor:

Damien Rohmer, LIX

Academic year 2023/2024

Abstract

In Computer Graphics, the so-called "Linear Blend Skinning" (LBS) is a widely method used to represent static virtual articulated character deformation. It binds the vertices of a 3D surface to a set of hierarchical skeletal joints associated with rigid transform. The recent velocity and acceleration skinning approaches extended this approach to dynamic deformers allowing them to represent cartoon-like deformation while preserving the hierarchical nature and real-time efficiency of LBS using joint velocity. An ongoing research project in the VISTA team proposes to further extend such deformation to time-dependent deformations via the use of linear time filters applied to the joint representation. This new approach defined as "Dynamic Skinning" can model various cartoon-like effects such as damped oscillations, and wave-like deformation propagation, which can be modeled in designing adapted convolution kernels defined as the impulse response of the time-filters. This project focuses on extending the method of Dynamic Skinning to incorporate different oscillation shapes. Specifically, I designed the kernels as the response of a unit step, rather than an impulse. This allows for the representation of new oscillation forms such as triangular, rectangular, pendulum, and rapid oscillation, expanding the potential of Dynamic Skinning.

Contents

1 Introduction

1.1 Context

This project had as an objective the expressive dynamic animations of 3D virtual characters. As in most of the cartoons, the goal is to obtain exaggerated deformations, usually more visible on static images (see Fig. 1). The expressive dynamic animation plays an important role in facial expression, by offering an emotion or feeling. The character animation is composed of energy (dynamics, velocity, weight, etc.) and material property (elasticity vs rigidity). The expressive animation is completely different from realistic animation and provides visual clues about the animation dynamics, using few details/elements/frames[9].

Figure 1: Expressive animation, exaggerated deformations.

1.2 Useful terminology

Let us first describe the fundamentals of expressive character animation, and review related works. In the following section, I will explain the terminology as outlined in *'Real-time Expressive Dynamic Animation of 3D Virtual Characters'*[9]:

- *Animation of 3D Virtual Character* refers to the characters' body: its pose (often a skeleton) and its shape/skin deformation
- *Expressive Dynamic Animation* comes from the notion of "facial expression". This term has been extended to the notion of expressive animation for the whole body. In this case, it refers to a notion of energy of motion: "Provide visual clues about the animation dynamics, using few details/elements/frames".
- *The skeleton* (see Fig. 2) is a set of joints organized in a hierarchy (tree, root=pelvis), *the joints* are frames, 4x4 matrix with rotation and translation, *the bone* is a segment between two joints and *the animated skeleton* are rigid transformations applied to the joints

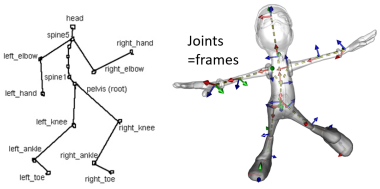

Figure 2: Skeleton, joints, bone.

- *A polygon mesh* (see Fig. 3) is a collection of vertices, edges and faces that defines the shape of a polyhedral object. The faces usually consist of triangles (triangle mesh), quadrilaterals (quads), or other simple convex polygons (n-gons).

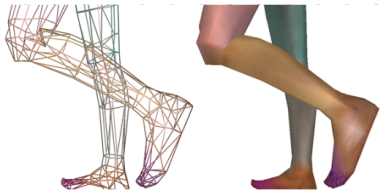

Figure 3: Representation of a polygon mesh

2 Related works

In standard industry, the most common approach to represent cartoon deformation remains the use of procedural deformers that are manually parameterized by artists. While highly general, these deformers remain time-consuming to set up and are not easily transferable from one character to another. In the academic world, the problem of cartoon deformation has been addressed by several authors and can be categorized along two main axes: (i) the level of automatization from purely manual to fully automatic, and (ii) the specialization toward animation ranging from specific pose-based, to fully time-based dynamic approaches.

Physically based animation

Fully automatic and time-based approaches are generally addressed by the so-called physically-based simulation. These methods represent the dynamic of the character using physical material properties such as shape inertia and elasticity, and the actual deformation is obtained by computing a numerical solution of the equations of motion, which is represented as a partial differential equation. These methods have been widely studied in research [7], but remain limited by their computational time and the difficulty to explicitly control the resulting deformation. Fast simulation methods have been proposed and rely typically on either a pre-computation of the space of possible deformation [13, 15, 16], or on a simplification of the physical model using, for instance, position-based dynamics [14]. Nevertheless, these methods remain difficult to control, and the user has to rely on a trial-and-error process to obtain the desired result, which makes them less suited to the production pipeline.

Alternative approaches

On the other side of the spectrum, artist-oriented models have been proposed in order to leverage the sketching and drawing skills of artists to set the static pose of a character, where the user draws a curve, and the deformation is computed by solving an optimization problem that matches the user input. The method was successfully developed to express the notion of line-of-action [3] for characters, or impact-based deformation. Extensions have been proposed to extend such a pose-based deformation to dynamic trajectories, where the sketch of the artist can depict the space-time motion of the character [4, 1]. This approach eases the control of the deformation but remains limited to a specific deformation attached to a given character, and therefore, is time-consuming to set when dealing with multiple characters. A last type of approach consists in transferring a reference animation to a new character, thus allowing an automatic generation of animation to multiple characters. Thes approaches, often described as *retargeting* [11, 17], allows very efficient animation generation, and is the core of the widely used tool Mixamo. However, the retargeting process can be seen as a successive pose-transfer of the skeleton and is not designed to handle the specificity of cartoon deformation related to the character's skin and flesh reacting to the skeleton's motion.

Current positioning

The approach we use in this work called *Dynamic Skinning*, can be seen as an intermediate between fully automatic methods and manual ones, as the character's deformation is fully computed from the skeleton motion but relies on an existing character rig and skeletal motion. Dynamic Skinning doesn't aim to change a character's pose but only acts as a deformation response to the skeleton's deformation. The key advantage of this approach is that it remains as efficient as pure standard linear blend skinning while providing automatic dynamic deformation that remains easy to tune.

A few works have proposed fast and controllable deformers on top of linear blend skinning. Among them, Noble and Tang [8] proposed a procedural bending parameterized by the velocity between two consecutive joints, but were not taking advantage of a full character rig and could not handle time-related deformation. Interestingly Wang et al. [12] and Kass and Anderson. [5] proposed a time-based oscillation model able to propagate cartoon-like deformation along a shape. They rely on a time filter to compute the oscillation. While these approaches naturally handle time-related phenomena, the design of such a time filter and the spatial propagation of the deformation remain difficult to control. Our approach is inspired by such a time filter, but it adapts and extends to be easy to control. First, the Dynamic Skinning framework allows to seamlessly propagate a deformation along the character geometry based on its rig. Second, we provide an easy way to control the time-based deformation in designing the oscillating behavior as a linear filter defined as a response to a unit impulse of velocity. As such, the time response will be handled by a simple filter that can be represented as an oscillating curve, while the spatial propagation is handled via the Dynamic Skinning framework. The current work will focus **on the design of such time-filters in order to model various oscillating-like behavior**.

3 Background on expressive animation

The general objective of the expressive animation is to obtain the cartoon effects (drag, stretch, oscillation, delay)[10] that automatically adapt to standard 3D Animated Character (typical Skeleton-Based Skinning used in production). However, there are some constraints, like real-time (>25 fps) efficiency and user control/controllability. We get the desired results through skinning. Skinning is deforming the "skin" surface on top of an articulated skeleton[9].

3.1 Linear Blend Skinning (LBS)

Linear Blend Skinning is the standard industry method for character animation. It is used in video games, animation cinema, etc. One disadvantage is that it is limited to static key-framed poses. The LBS is a linear blending between transformation matrices (see Fig.4), creating a smooth blend deformation near joints [6]. At each frame, the position p_i is computed as follows [9]:

$$p_i = \sum_{\text{joints } j} \alpha_{ij}\, p_{i/j}$$

where α_{ij} are skinning weights and $p_{i/j}$ are positions from rigid skinning relative to joint j

The skinning weight α_{ij} is a coefficient that determines the influence of a specific joint *j* on a particular vertex *i* of the mesh, and $p_{i/j}$ represent positions from rigid skinning relative to joint *j*.

3.2 Velocity Skinning

Velocity Skinning is a simple technique to add exaggerated deformation triggered by skeletal velocity on top of standard skinning animation (See Fig.5). [2] It consists of a deformation for each moving joint and is split between linear and angular velocity. Therefore, we get that the formula for the velocity skinning is computed as a sum of the LBS position and a deformation depending on the vertex velocity. It is computed as follows[2]:

$$p_i^{VS} = p_i^{LBS} + \varphi(\dot{p}_i) \quad \text{where} \quad \dot{p}_i = \sum_{\text{joints } j} \beta_{ij}\, v_{i/j}$$

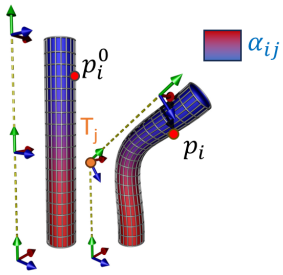

Figure 4: Linear Blend Skinning on a cylinder.

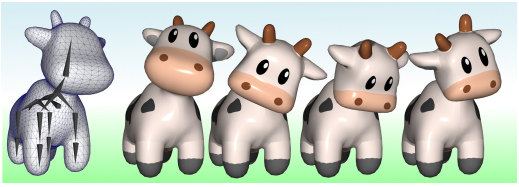

Figure 5: Velocity Skinning on a cow mesh.

3.3 Dynamic Skinning

Dynamic skinning is a time-filtered [12, 5] version of the initial velocities seen as a time signal instead of the instant velocities. If the instantaneous velocity is being used, then the deformation can only reflect this instantaneously. For example, if the velocity is 0, then there is no deformation at all, which doesn't allow an oscillation-like behavior. The filter considers the velocity of the joint as a time signal and uses its current and past value (in a limited window of a few seconds) to generate such oscillation behavior. This can be formalized as using a time filter over the velocity signal and tuning the filter such that the filtered velocity oscillates when we have a sudden change of instantaneous velocity. It can model various cartoon-like effects such as damped oscillations, and wave-like deformation propagation.

From velocity skinning, we extend the formula $\sum_{\text{joints } j} \beta_{ij}\, v_{i/j} to \sum_{\text{joints } j} \left(\beta_{ij} \cdot \mathcal{F}\left(v_{i/j}, t\right)\right)$. See Fig.6:

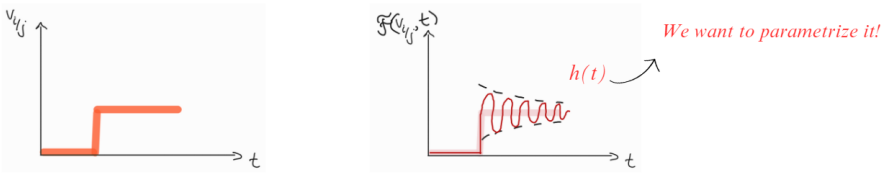

Figure 6: Graphs for Velocity Skinning and Dynamic Skinning.

By default, the oscillation is a sinusoidal wave and the user can adjust the frequency base, frequency increase slope, damping and magnitude (see in the next chapter more details about this). We can see how the frequency impacts the graph of the wave in Fig.7:

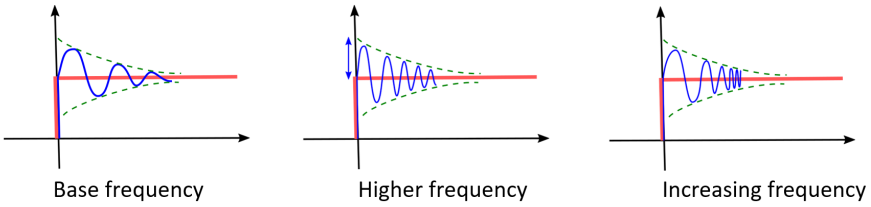

Figure 7: Different frequencies.

3.4 Objective and method

The objective of this project is to implement different types of oscillations such that the user can independently choose the type of oscillation, frequency, damping, and magnitude. There are some proposed oscillations with different-shaped waves that I am going to detail in the next chapter.

The method I followed in order to achieve these results is the following:

- Study how to design the time filter, a response to a unit step.
- Study the case of the sinusoidal oscillation.
- Model generalized responses:
  - triangular oscillation
  - pendulum oscillation
  - Beep-Beep oscillation
  - rectangular oscillation
- Evaluate the code and debug using C++ and Python: after each oscillation, the code generates a CSV file that I use to plot the results and see the shape of the wave.
- See the results: check each type of oscillation for different models (giraffe, whale, ray-fish, character, clown, etc) and observe how they behave.
- Take videos of the same model for the sinusoidal oscillation and the new ones implemented and compare: which one is more "natural", or "suited" for the model or action.

4 Oscillations

4.1 Time filter design

The fundamental idea of this work is to design different time filters $\mathcal{F}$ to be applied to the velocity (either linear velocity or angular velocity) of the joints. While the Dynamic Skinning framework handles the deformation's spatial propagation thanks to the weights β_{ij}, the time filter enables the representation of a time-varying oscillating behavior.

Designing in general time-filter can be complex. In our case, we consider a linear causal filter, i.e., a linear filter that only considers past values of the joint velocity. To represent this filter in an intuitive way, we propose to design it as the response to a unit impulse of velocity, i.e. a sudden change of velocity going from 0 to a constant unit value. Such a response can then be expressed as a parametric curve oscillating around the unit step distribution, and the response to an arbitrary signal is then obtained via standard linear signal filtering process.

We call $\kappa(t)$ a user-defined oscillating curve. We then define the response of the linear filter to the unit step as

$$h(t) = (1 + \kappa(t))\, U(t), \tag{1}$$

where $U(t)$ is the unit step distribution

$$U(t) = \begin{cases} 0 \text{ if } t < 0, \\ 1 \text{ if } t \geq 0. \end{cases}$$

We note that using the form $(1+\kappa(t))$ allows us to ensure that h oscillates around the value 1 when κ oscillates around 0. In the next part we will assume that κ satisfies $\kappa(0) = 0$, and remains null after a maximal time window W of a few seconds ($t > W \Rightarrow \kappa(t) = 0$). Once the user has defined the function κ, the response to an arbitrary velocity signal $s(t)$ can then be obtained as the convolution between the derivative of h (i.e. the response to a Dirac impulse in the sense of the distributions) and the input signal

$$\begin{aligned} \mathcal{F}_\kappa(f,t) &= (\dot{h} \star f)(t) = \int_{u\in\mathbb{R}} \dot{h}(u)\; s(t-u)\,\mathrm{d}u \\ &= f(t) + \int_{u\in[0,W]} \dot{\kappa}(u) s(t-u)\mathrm{d}u \end{aligned} \tag{2}$$

where $\dot{h} = \frac{\mathrm{d}h}{\mathrm{d}t}$. The actual steps of the remaining work are:

1. Pre-define an appropriate parametric function κ representing the expected oscillation style (choice detailed in the following sections).
2. At any time t_0 of the animation, compute numerically
   - The time derivative $\dot{\kappa}(t_0)$
   - The filtered value as the convolution

$$\mathcal{F}_\kappa(f,t_0) = f(t_0) + \sum_{u\in[0,W]} \dot{\kappa}(u)\; s(t_0-u)\; \Delta t \tag{3}$$

   with Δt being the time between two consecutive frames (typically 1/60 second).

4.2 Implementation and Purpose of Additional Oscillation Types

The purpose of implementing more types of oscillations is to help the user decide on which oscillation he thinks suits the model and the movement he wants to show. Therefore, we decided on 4 more types of oscillations, besides the sinusoidal one which was already implemented: triangular, rectangular, pendulum, and a rapid oscillation resembling the famous oscillation of the bluebird in the animated movie "The Road Runner Show".

I created a new function, called *convolution*, which stores in a CSV file the values returned by the *kappa* function. The *kappa* function is a time filter defined as a response to a unit step, and it is parameterized in the following way

$$\kappa(t) = A \cdot e^{-\frac{t}{\tau}} \cdot \sin(2\pi f t)$$

where A is the magnitude, τ is the damping, and f is the frequency (see Fig.8).

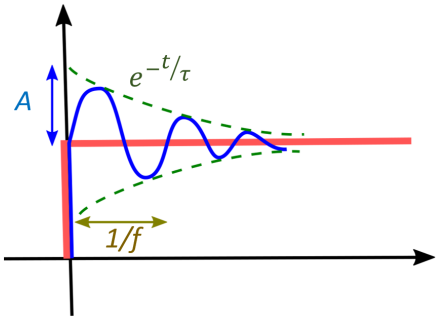

Figure 8: Graphical illustration of an oscillation.

4.3 Sinusoidal oscillation

This oscillation was already implemented and its purpose is to make the part of a mesh, depending on the selected joint, oscillate by generating a periodic signal in the shape of a sinusoidal wave whose amplitude slowly decreases to 0 (see Fig.9).

Using the *convolution* function, I plotted the graph generated by the values of the *kappa* function when we have a sinusoidal oscillation of the mesh. The result is computed using the formula:

$$e^{-attenuation \cdot k \cdot 0.017} \cdot \sin(u)$$

where $u = frequency \cdot 0.017 \cdot k \cdot 2 \cdot \pi$.

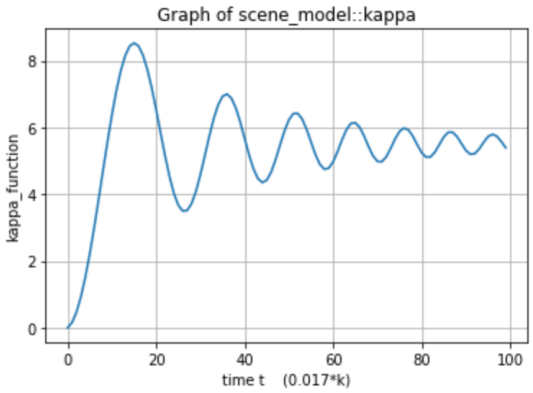

Figure 9: Wave of the sinusoidal oscillation.

4.4 Triangular oscillation

For the triangular oscillation, we wanted as a result to have a mesh that oscillates in a linear increasing function, then decreases linearly and so on, by generating a periodic signal in the shape of a triangular wave whose amplitude slowly decreases to 0 (see Fig.10). Depending on the phase within one period of the waveform, less or greater than π, we set the direction to be 1, indicating an increasing phase, or -1 respectively for a decreasing phase. To ensure that the peak of the sinusoidal value matches the peak of the triangular one, we

will use the term $\frac{2}{\pi}$ and to transform the sinusoidal wave into a triangular one, we extract the $\arcsin$ of the $\sin$. Therefore,we have:

$$phase = u \mod 2\pi$$

$$direction = \begin{cases} 1 & \text{if } phase < \pi \\ -1 & \text{otherwise} \end{cases}$$

$$\text{and thus } result = \left(\tfrac{2}{\pi}\right) \cdot \arcsin(\sin(u)) \cdot e^{-attenuation \cdot k \cdot 0.017} \cdot direction$$

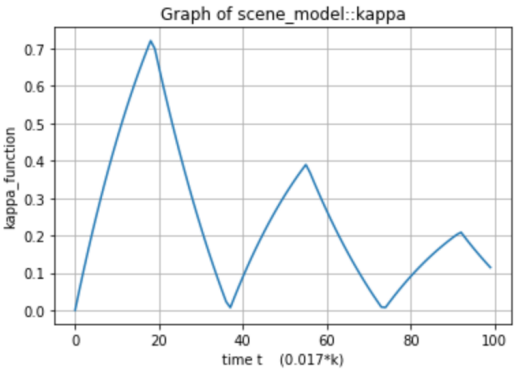

Figure 10: Wave of the triangular oscillation.

We see that the graph is not perfectly straight but slightly curved. This is due to the increase of time *t*.

4.5 Pendulum oscillation

The main idea for the pendulum oscillation was to have an oscillation that makes the mesh rest more at the extremities and go fast in between (see Fig.11). To achieve such an oscillation, the graph should resemble a trapezoidal wave but the corners to be smoother. Therefore, we take just the first two Fourier components of the trapezoidal wave and we get the equation:

$$f_{\text{trap}}(x) = \tfrac{8\sqrt{2}}{\pi^2}\left(\sin(x) + \tfrac{\sin(3x)}{9}\right)$$

However, the result on the mesh was not what we wanted, so the constant $\frac{8\sqrt{2}}{\pi^2}$ was deleted. Remember that the graph just helps us to achieve the desired result, but it should not be perfect in order to achieve a specific oscillation. Thus, I modified the formula such that the oscillating mesh has the behavior described above. Hence, we get:

$$constant = -attenuation \times k \times 0.017$$

$$\text{and thus } result = e^{constant}\left(\sin(u) + \tfrac{\sin(3u)}{9}\right)$$

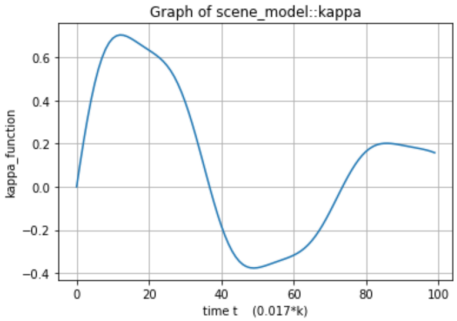

Figure 11: Wave of the pendulum oscillation.

4.6 "The Road Runner Show" oscillation

This oscillation was probably the easiest one to implement. "The Road Runner Show" oscillation refers to a very fast one, similar to the oscillation of the neck of the bluebird in the famous animated movie (see Fig.12). Thus, since the oscillation is still a sinusoidal wave, the only term that needed to be modified was f_{slope} and we gave it a hard-coded value of 3 (in the previous examples it was 0).

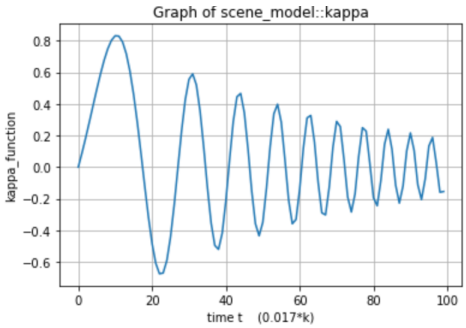

Figure 12: Wave of the Beep-Beep oscillation.

4.7 Rectangular oscillation

This oscillation was the most difficult one to implement. We wanted an oscillation that rests at the extremity for some time and goes almost instantly to the other extremity where it rests again and so on (see Fig.13). I modified the waveform to be a rectangular one by adjusting the amplitude based on the sign of the sin function. If the sin function is positive, it returns the calculated result, otherwise, it returns the negated result, effectively inverting the waveform.

Since the waveform has to be constant for half a period, we compute the $constant$ parameter which depends on $k - remainder$. The $remainder$ represents how much of the current period has already been completed. Subtracting the $remainder$ from k effectively resets the wave number to the start of the current period. Thus:

$$period = \frac{1}{frequency} \text{ and } remainder = k \bmod \left\lfloor \frac{period}{dt} \right\rfloor$$

$$\text{and thus } result = \begin{cases} e^{-\text{attenuation}\cdot(k-\text{remainder})\cdot 0.017} & \text{if } \sin(u) > 0 \\ -e^{-\text{attenuation}\cdot(k-\text{remainder})\cdot 0.017} & \text{otherwise} \end{cases}$$

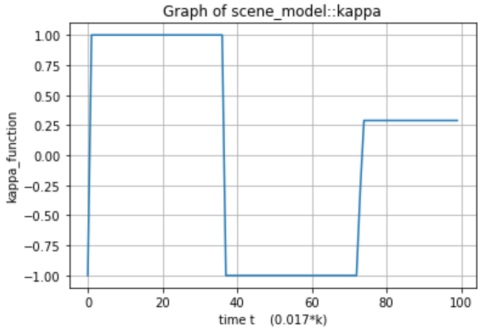

Figure 13: Wave of the rectangular oscillation.

However, we see that we have the same problem as for the triangular wave, which is that the rectangles are not perfectly straight, having a slight curve. This is due to the increase of time *t*.

5 Speedup filter

The speed-up filter is designed to simplify and accelerate the computation process. The standard numerical convolution, computed as the product of the filter kernel and the input values, takes around 3 seconds. Since this computation must be performed for each joint and vertex, the speed-up filter is introduced as another option in order to reduce this time-consuming process.

Standard convolution computation takes into account the entire motion, but we can achieve a similar effect by focusing on the most "influential" part of the input motion. By identifying and retaining the maximum influence (the highest velocity in the time window), we can get the desired effect. Instead of convolving with the entire signal, we perform the convolution with a Dirac δ function positioned at the most influential input value. This influential value is defined as the maximum of the input signal, scaled by the decay over time. The process involves creating a new signal $-e^{-\alpha \cdot t}$, identifying the maximum value of $f(t)$, and convolving the kernel with this Dirac position and magnitude. If we replace the input signal with the Dirac function, we don't need the full numerical convolution, simplifying the computation, so we evaluate the kernel at a single time point. This approach speeds up the process and obtains the desired effects.

6 Structure of the code

During the implementation of these oscillations, I modified the code to save the points returned by the oscillation functions into a CSV file. This method helped debugging and verification processes and improved my skills in C++ and Python. Using Python and the *numpy* library, these values were plotted to verify if the resulting wave matched the desired output.

In order to provide a comprehensive structure of the code for this project, I implemented a distinct function for each oscillation type. These functions are all of the type *vec3 maxjointfilteredvalue* (see Fig.14). When the user selects a specific oscillation type, the corresponding function is called.

In this structure, *evaluateSinusoidal, evaluateTriangular, evaluateRectangular, evaluatePendulum*, and *evaluateBipBip* are the functions corresponding to each type of oscillation. When the user selects an oscillation type by clicking on the radio buttons and the respective evaluation function is called, producing the desired output.

```
struct max_joint_filtered_value{
    float t0;
    vcl::vec3 value;

    void update(vcl::vec3 const& new_candidate, float t_current);
    vcl::vec3 evaluateSinusoidal(float t_current) const;
    vcl::vec3 evaluateTriangular(float t_current) const;
    vcl::vec3 evaluateRectangular(float t_current) const;
    vcl::vec3 evaluatePendulum(float t_current) const;
    vcl::vec3 evaluateBipBip(float t_current) const;
    float evaluate_damping(float t_current) const;

    static float attenuation;
    static float frequency;
    static float frequency_slope;

};
```

Figure 14: Definition of the structure.

As I previously mentioned, I created radio buttons such that the user can choose one oscillation type at a time. The moment the oscillation starts, the function intended for the selected type of oscillation is called.

6.1 Updating the GUI

To facilitate user interaction and the selection of oscillation types, I updated the GUI (Graphical User Interface). This update also allows users to adjust the frequency, damping, and magnitude (see Fig.15). The radio buttons ensure that the user can only choose one oscillation type at a time, thus preventing multiple selections that could be followed by errors in the system.

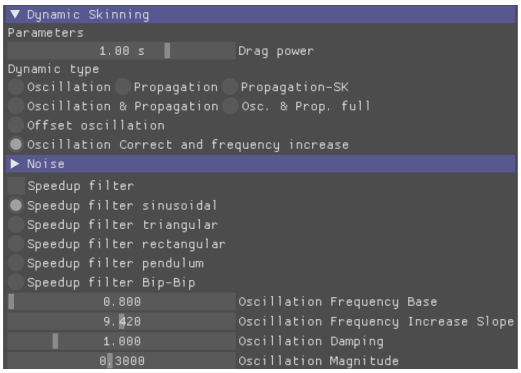

Figure 15: Updated GUI.

We observe the 5 radio buttons for each of the oscillations and the sliders that allow the user to modify the frequency, damping and magnitude.

7 Conclusion

In conclusion, this project has successfully introduced four additional oscillation effects (see Fig.16, providing users with a broader range of options to achieve their desired outcomes.

In the end, I propose some future potential extensions: developing more effects and determining the appropriate effect based on the physical properties of the represented material. For example, automatically adjusting the effect according to the length and width of the skeleton.

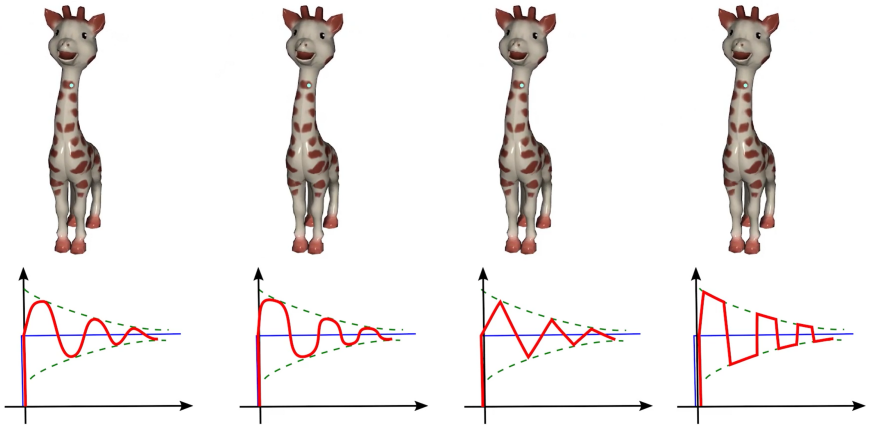

Figure 16: Result of the different oscillations.

8 References

[1] Byungkuk Choi, Roger Blanco i Ribera, J. P. Lewis, Yeongho Seol, Seokpyo Hong, Haegwang Eom, Sunjin Jung, and Junyong Noh. SketchiMo: Sketch-based Motion Editing for Articulated Characters. *ACM SIGGRAPH, Proc. Transactions on Graphics*, 2016.

[2] Niranjan Kalyanasundaram Faezeh Moshfeghifar Marie-Paule Cani Victor Zordan Damien Rohmer, Marco Tarini. "velocity skinning for real-time stylized skeletal animation". *Eurographics, Computer Graphics Forum, 2021*.

[3] Martin Guay, Marie-Paule Cani, and Rémi Ronfard. The line of action: an intuitive interface for expressive character posing. *ACM SIGGRAPH Asia, Proc. Transactions on Graphics*, 2013.

[4] Martin Guay, Rémi Ronfard, Michael Gleicher, and Marie-Paule Cani. Space-time sketching of character animation. *ACM SIGGRAPH, Proc. Transactions on Graphics*, 2015.

[5] Michael Kass and John Anderson. Animating oscillatory motion with overlap: Wiggly splines. *ACM SIGGRAPH, Proc. Transactions on Graphics*, 2006.

[6] THALMANN D. MAGNENAT-THALMANN N., LAPERRIRE R. Joint-dependent local deformations for hand animation and object grasping. *Graphics interface'88*, 1988.

[7] Matthias Muller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM SIGGRAPH, Proc. Transactions on Graphics*, 2005.

[8] P. Noble and W. Tang. Automatic expressive deformations for stylizing motion. *GRAPHITE*, 2006.

[9] Damien Rohmer. Real-time expressive dynamic animation of 3d virtual character (presentation slides). 2024.

[10] Frank Thomas and Ollie Johnston. The illusion of life: Disney animation. 1981.

[11] Ruben Villegas, Jimei Yang, Duygu Ceylan, and Honglak Lee. Neural kinematic networks for unsupervised motion retargetting. *CVPR*, 2018.

[12] Wang, S. Drucker, M. Agrawala, and M. Cohen. The cartoon animation filter. *ACM SIGGRAPH, Proc. Transactions on Graphics*, 2006.

[13] Ying Wang, Nicolas J. Weidner, Margaret A. Baxter, Yura Hwang, Danny M. Kaufman, and Shinjiro Sueda. REDMAX: Efficient Flexible Approach for Articulated Dynamics. *ACM SIGGRAPH, Proc. Transactions on Graphics*, 2019.

[14] Yuhan Wu and Nobuyuki Umetani. Two-way coupling of skinning transformations and position based dynamics. *ACM SIGGRAPH, Proc. Transactions on Graphics*, 2023.

[15] Hongyi Xu and Jernej Barbic. Pose-space subspace dynamics. *ACM SIGGRAPH, Proc. Transactions on Graphics*, 2016.

[16] Jiayi Eris Zhang, Seungbae Bang, David I.W. Levin, and Alec Jacobson. Complementary dynamic. *ACM SIGGRAPH, Proc. Transactions on Graphics*, 2020.

[17] Ynbo Zhang, Deepak Gopinath, Yuting Ye, Jessica Hodgins, Greg Turk, and Jungdam Won. Simulation and retargeting of complex multi-character interactions. *ACM SIGGRAPH*, 2023.