Report
Wavelet and applications

Jonathan JULOU Wilfried BROISSART

February 2021

Contents

1 Setting up the project

If it is not installed already, install Blender : https://www.blender.org/

We developed our addon on Blender 2.91.0. The ideal would be to have the most recent version of Blender, but given what functionalities we used our code should work from Blender 2.8 onward.

To run our code, it is necessary to replace Blender's bundled python by another environment where it would be possible to install the pyWavelet library. From what we tested with anaconda, running blender from a command prompt with the desired environment activated seems to work. On windows, you might have to replace Blender's python by a hardlink to the environment that has pyWavelet if it does not work.

If you want to quickly test our code, we put the blender file we used for testing, which contains some motion capture data and the addon pre-configured, in the project archive. In that case, you can run it from the script tab, by selecting the script "debug.py" that will link the files of the project to the debug environment when run. You have to modify the absolute path to the addon folder in that case. Otherwise, it is possible to install our project using the Add-ons menu in Edit/Preferences from the zip file included.

2 Introduction

Parametric Motion Blending refers to the controlled combination of two animations on the same armature. Roughly, the idea of the article is to take two animations: the main movement (the low frequencies are interesting) and a detailed movement (the high frequencies are interesting), and to combine those two motions to create a motion that feels natural. When blending the motions, the large scale movements would be primarily taken from the edited animation, while for the small scale ones the original motion capture would be favored.

The paper proposes to use wavelets to achieve that. By applying the DWT at multiple scales, it is possible to isolate each resolution in the motion, and for each one do a barycentric combination using a different coefficient at each scale. The new motion would be the result of this multiresolution interpolation.

We did our implementation as a Blender plugin in python. Therefore, we already have a 3D environment, with armatures, animation and display. The core of the computations described in the article is done using the pyWavelets library.

3 Implementation

3.1 Working with Motion Data

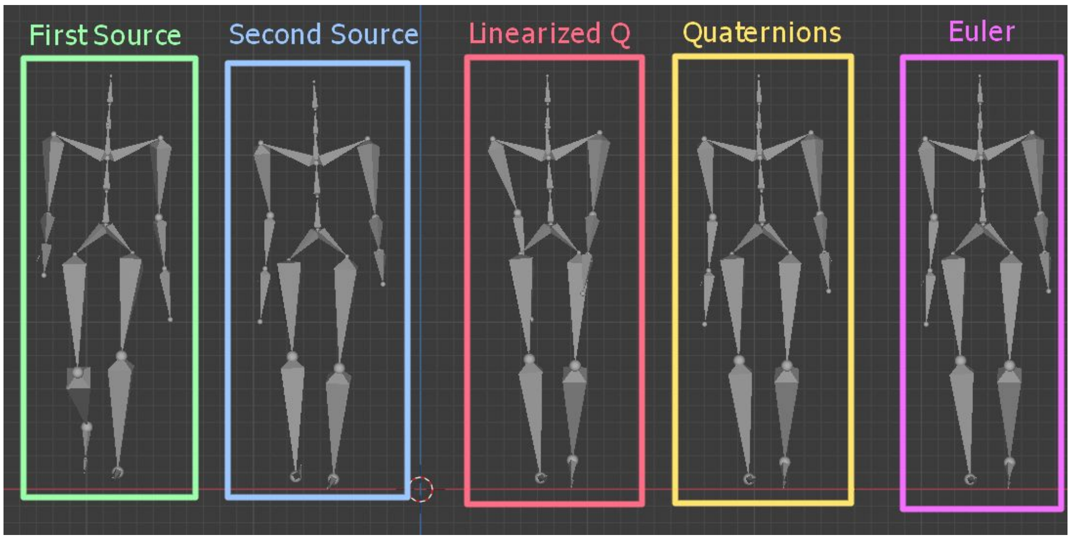
As mentionned in our implementation expectations, the motion curves used in the article are not clearly defined. From what we understood, it is of utmost importance that the representation of the motion be linearly separable since the article seems to use one-dimensional wavelet decompositions for each coordinate.

The XYZ coordinate system is already linearly separable.

This should however exclude using Euler Angles for rotations. The thesis we wrote about in the expectations gives a way to linearise quaternions, so we thought it their motion-curve space was what was used in the article we would implement. However our implementation of their transform gives unexpected orientations in some places. On the other hand, directly using the quaternions, by separating the 4 components, gives very good results.

We go over what we tried for quaternion linearisation. The functions we wrote to go in and out of the linearized quaternion space are strongly inspired by the construction given page 29 of the thesis.

To test which representation would be better, we designed a simple test that does not involve wavelets. The principle is to simply take the mean for each component of the motion representation to animate what should be the mean motion of the two source animations.



For the three, the movement is pretty smooth and logical, but with the linearized quaternions, there seems to be an unwanted inward attraction. Also Euler seems to do well on screenshots, but we saw some occasional visible bone flipping in the animation even if the global movement was identical to quaternions. In conclusion of this test, we thought that using standard quaternion representation would do well, and contrary to what we expected euler is not too bad. We therefore used the quaternions for our wavelet interpolation since it seemed a little more stable.

The results for the linearized quaternions seem strange given the expected properties of the linearized quaternion space described in the thesis, so we surely made mistakes in the code or forgot some important detail. From what we understood, the main idea behind their transform is actually not to make the quaternion linearly separable, but to have all their space represent rotations, which is not the case with the 4 dimentional quaternion space, to help with interpolation. So since we already get good interpolation with quaternions we thought it did not seem necessary to delve further into their process.

Finally, the data we take into consideration is composed for each bone of a 3-dimensional position and a 4-dimensional rotation. We split them into 7 one-dimensional vectors along time, and feed them into a 1D wavelet decomposition. We do different operations to the decomposed coefficients depending on which of the three functionalities described in the next sections is chosen, and finally reconstruct a 1D vector from the operated-on coefficients. This is written onto a target armature to be able to see the new animation and compare it to the original motions.

3.2 Movement Compression inspired by Lab 2

The idea of the algorithm of the paper is to use a similar decomposition to that of the beginning of lab 2, decomposing a movement into a coarse and a set of detailed resolutions of the wavelet decomposition.

So to get started with a more familiar process, we implemented a compression of the movement akin to what is done in lab 2, with a thresholding of the coefficients. We did it as two functionalities :

The first one is literally the compression function implemented in lab 2 applied on each bone's keyframes (the keyframes values for each axis are concatenated to create multiple one-dimensional array). The compression ratio can be set in the interface.

The second one corresponds to a "resolution level thresholding" : coefficients from high level resolution are set to 0. The resolution level until which coefficients are conserved can be set in the interface. It gives more control over what happens to the wavelet coefficients than the compression ratio.

3.3 The Algorithm from the Article

The five main steps are:

1. Time warping to synchronize times of the key events

2. Apply the DWT on both signals : we obtain $\forall i \in \{1;2\} S_i = [C_i^N, D_i^{n-1}, ..., D_i^N]$

3. Apply the motion blending to S_1 and S_2 to get $S = [X_C C_1^N + (1 - X_C)C_2^N, X_{n-1}D_1^{n-1} + (1 - X_{n-1})D_2^{n-1}, ..., X_N D_1^N + (1 - X_N)D_2^N]$

4. Apply the IDWT to S to get the final motion

5. Inverse time-warping to adjust the timing of the resulting motion

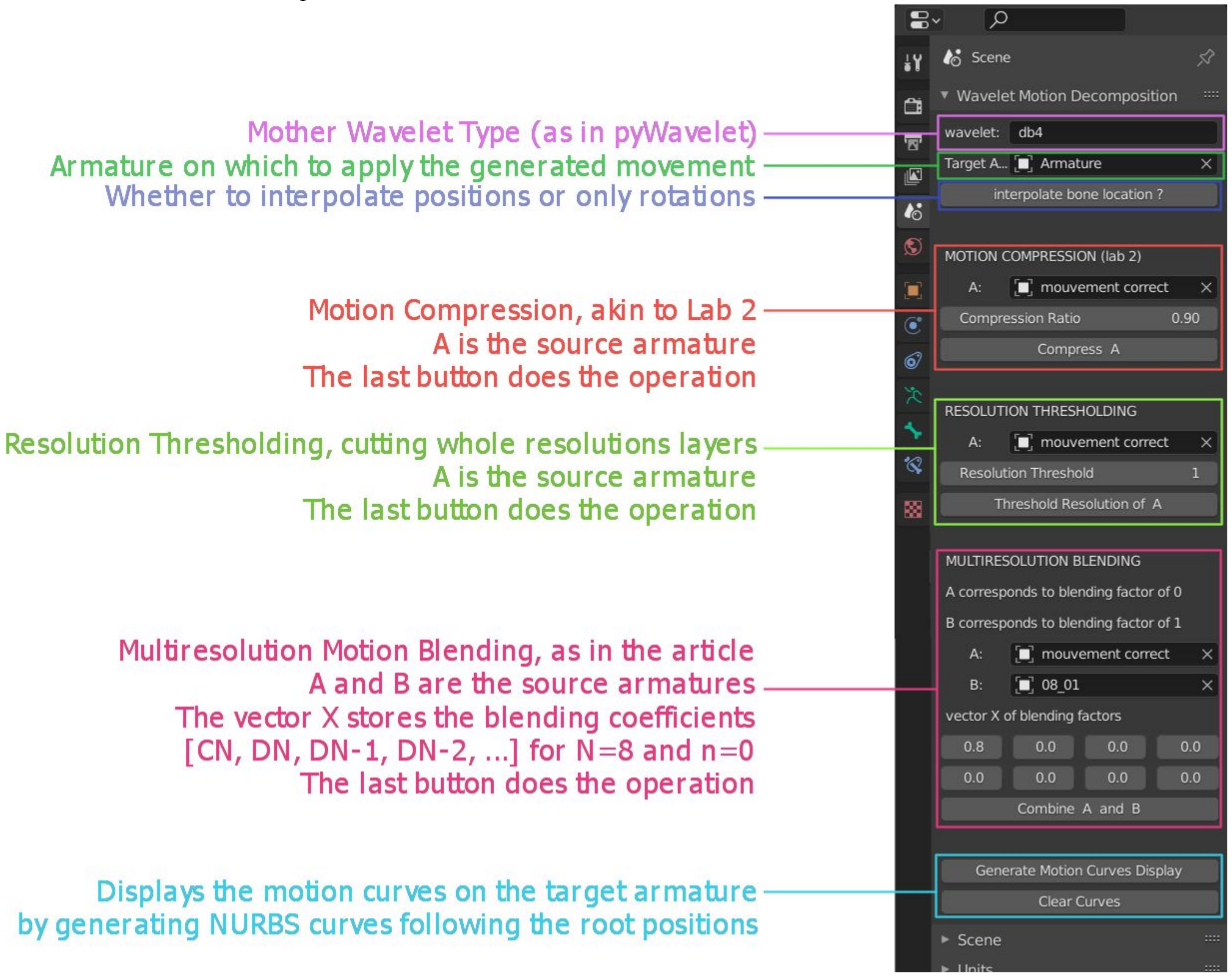
In the notation of DWT and motion blending, C stands for coarse, D for details and X is a vector of blending factors by resolution level. The most refined level of details is n and the less refined is the coarse one at level N.

We implemented steps 2, 3 and 4 in our Blender plugin. The time-warping steps 1 and 5 don't involve wavelets and are only necessary to merge movements with different timings, so we did not implement them.

In practice, we imposed n=0 and N=8 to simplify the interface and treatment. It is possible to go further in the details, but for the motion capture data we used for testing, changes in the detail levels beyond 4 are absolutely imperceptible.

3.4 Overview of the Interface

Below is an overview of the interface we designed to easily have access to every functionality from the Blender viewport.



Mother Wavelet Type (as in pyWavelet)

Armature on which to apply the generated movement

Whether to interpolate positions or only rotations

Motion Compression, akin to Lab 2
A is the source armature
The last button does the operation

Resolution Thresholding, cutting whole resolutions layers
A is the source armature
The last button does the operation

Multiresolution Motion Blending, as in the article
A and B are the source armatures
The vector X stores the blending coefficients
[CN, DN, DN-1, DN-2, ...] for N=8 and n=0
The last button does the operation

Displays the motion curves on the target armature
by generating NURBS curves following the root positions

Scene

Wavelet Motion Decomposition

wavelet: db4

Target A... Armature

interpolate bone location ?

MOTION COMPRESSION (lab 2)

A: mouvement correct

Compression Ratio 0.90

Compress A

RESOLUTION THRESHOLDING

A: mouvement correct

Resolution Threshold 1

Threshold Resolution of A

MULTIRESOLUTION BLENDING

A corresponds to blending factor of 0

B corresponds to blending factor of 1

A: mouvement correct

B: 08_01

vector X of blending factors

0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Combine A and B

Generate Motion Curves Display

Clear Curves

Scene

Units

To give an example of how to set the vector X, putting 1 in the first box and 0 in all others would mean that we take the coarse level from armature B and every detail level from armature A, effectively resulting in a transfer of quick and small movements from A onto the larger, slower movements of B.

Such an example is what the article presented as desirable considering motion capture data, taking the global movement of a handmade animation to precisely adjust the positions, and transferring the small movements from a motion capture doing roughly the same action to make it more humanlike.

We will see later in the section about results of our experimentation that interpolating bone locations with wavelets can give a very perceptible distortion on the time boundaries of the motion depending on the padding used in the decomposition. It seems overall less stable than rotation only. This is why we created the option to toggle bone location interpolation off. If so, the location blending is done through a weighted average of the positions in the two source animations, by the coarse coefficient X_0.

In the aforementioned case of making a motion look more human while keeping control over it, if the handmade animation takes into account realistic weight transfers, which causes small accelerations and slow-downs, using rotation only would give more control. However, if it is not the case, interpolating position can bring those small changes in position at the cost of exact position control.

3.5 Structure of our Code

For each one of the main functions (compress, resTres, combine), the global structure for wavelet operating on motion data is the following :

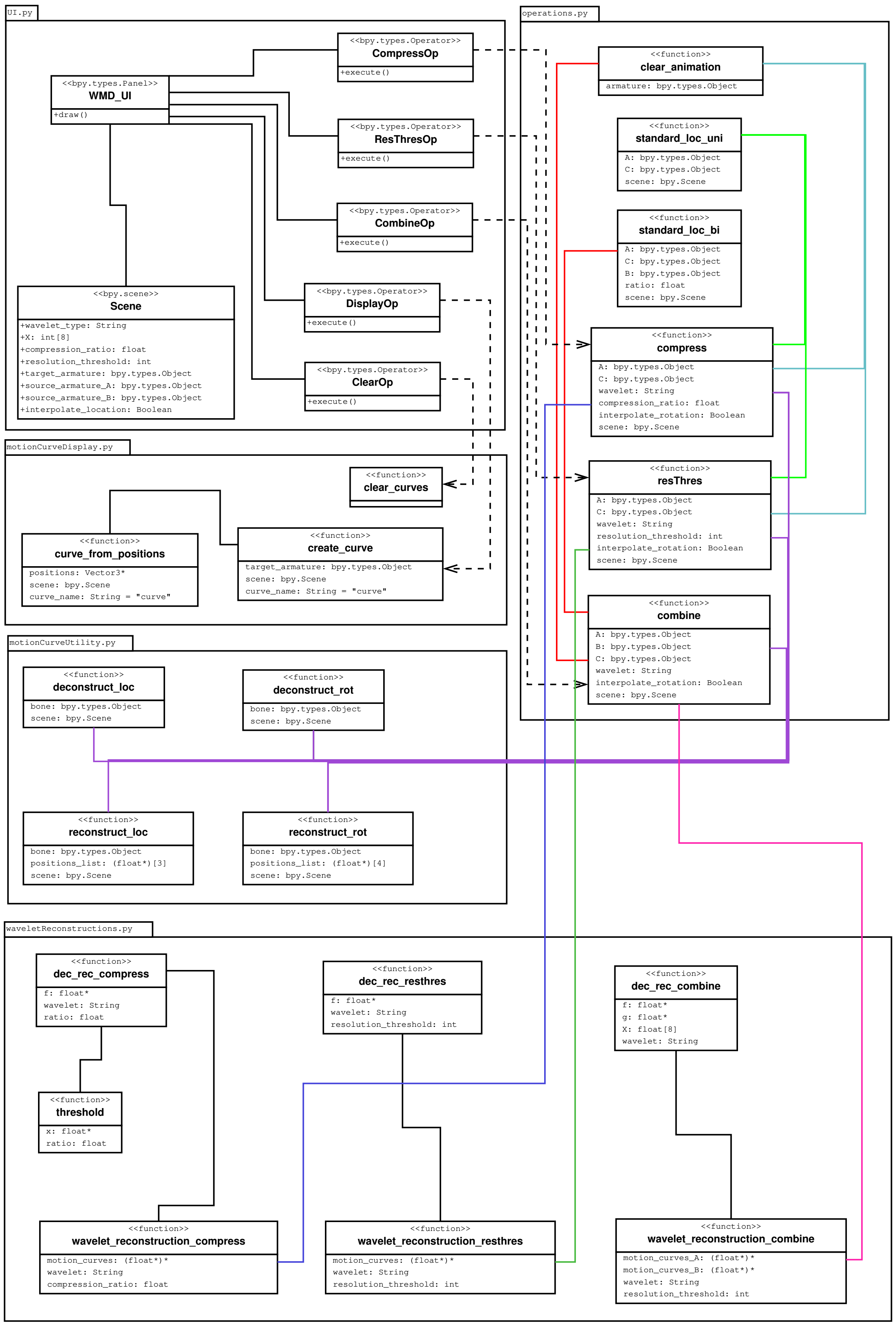
- deconstruct motion data (location or rotation) into a list of 1D vectors (for example $position \rightarrow (v_1, v_2, v_3)$)
- do the wavelet operation on each 1D vector
- reconstruct motion data from the modified 1D vectors (for example $(v'_1, v'_2, v'_3) \rightarrow position$)

The structure for each wavelet operation on a 1D vector is the following :

- deconstruct the 1D vector(s) into wavelet coefficients with DWT
- operate on the coefficients
- reconstruct a 1D vector from the coefficients with IDWT

Concerning wavelets, the interesting part is in the file "waveletReconstructions.py". Overall the code is not very advanced since it relies on pyWavelets to do all the hard work.

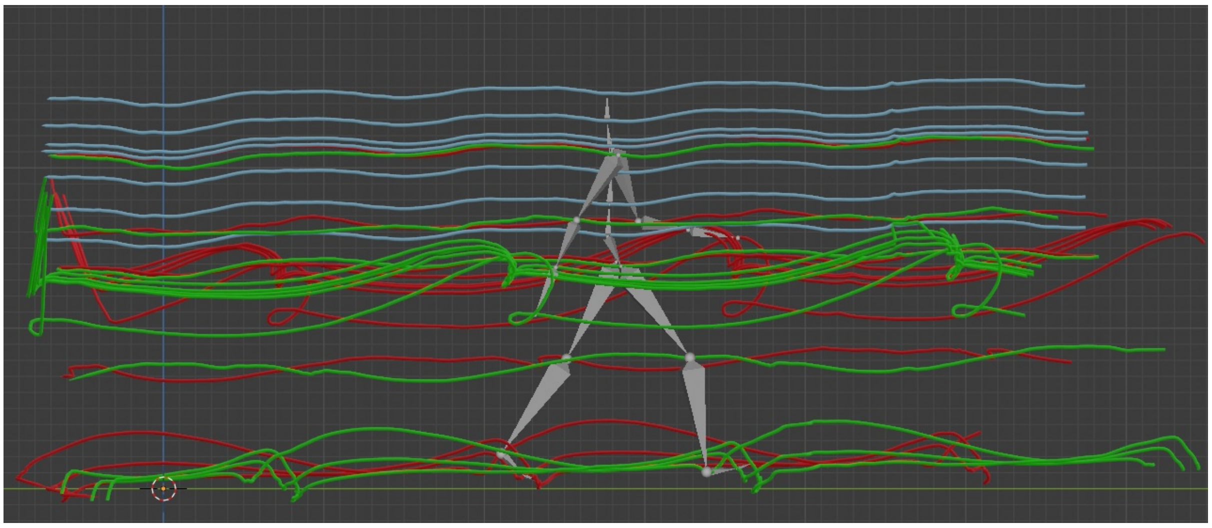
Below is a schematic representation of the interactions between files in our code :


UI.py
<<bpy.types.Panel>>
WMD_UI
+draw()
<<bpy.types.Operator>>
CompressOp
+execute()
<<bpy.types.Operator>>
ResThresOp
+execute()
<<bpy.types.Operator>>
CombineOp
+execute()
<<bpy.types.Operator>>
DisplayOp
+execute()
<<bpy.types.Operator>>
ClearOp
+execute()
<<bpy.scene>>
Scene
+wavelet_type: String
+X: int[8]
+compression_ratio: float
+resolution_threshold: int
+target_armature: bpy.types.Object
+source_armature_A: bpy.types.Object
+source_armature_B: bpy.types.Object
+interpolate_location: Boolean
operations.py
<<function>>
clear_animation
armature: bpy.types.Object
<<function>>
standard_loc_uni
A: bpy.types.Object
C: bpy.types.Object
scene: bpy.Scene
<<function>>
standard_loc_bi
A: bpy.types.Object
C: bpy.types.Object
B: bpy.types.Object
ratio: float
scene: bpy.Scene
<<function>>
compress
A: bpy.types.Object
C: bpy.types.Object
wavelet: String
compression_ratio: float
interpolate_rotation: Boolean
scene: bpy.Scene
<<function>>
resThres
A: bpy.types.Object
C: bpy.types.Object
wavelet: String
resolution_threshold: int
interpolate_rotation: Boolean
scene: bpy.Scene
<<function>>
combine
A: bpy.types.Object
B: bpy.types.Object
C: bpy.types.Object
wavelet: String
interpolate_rotation: Boolean
scene: bpy.Scene
motionCurveDisplay.py
<<function>>
clear_curves
<<function>>
curve_from_positions
positions: Vector3*
scene: bpy.Scene
curve_name: String = "curve"
<<function>>
create_curve
target_armature: bpy.types.Object
scene: bpy.Scene
curve_name: String = "curve"
motionCurveUtility.py
<<function>>
deconstruct_loc
bone: bpy.types.Object
scene: bpy.Scene
<<function>>
deconstruct_rot
bone: bpy.types.Object
scene: bpy.Scene
<<function>>
reconstruct_loc
bone: bpy.types.Object
positions_list: (float*)[3]
scene: bpy.Scene
<<function>>
reconstruct_rot
bone: bpy.types.Object
positions_list: (float*)[4]
scene: bpy.Scene
waveletReconstructions.py
<<function>>
dec_rec_compress
f: float*
wavelet: String
ratio: float
<<function>>
threshold
x: float*
ratio: float
<<function>>
dec_rec_resthres
f: float*
wavelet: String
resolution_threshold: int
<<function>>
dec_rec_combine
f: float*
g: float*
X: float[8]
wavelet: String
<<function>>
wavelet_reconstruction_compress
motion_curves: (float*)*
wavelet: String
compression_ratio: float
<<function>>
wavelet_reconstruction_resthres
motion_curves: (float*)*
wavelet: String
resolution_threshold: int
<<function>>
wavelet_reconstruction_combine
motion_curves_A: (float*)*
motion_curves_B: (float*)*
wavelet: String
resolution_threshold: int

4 Experimentation

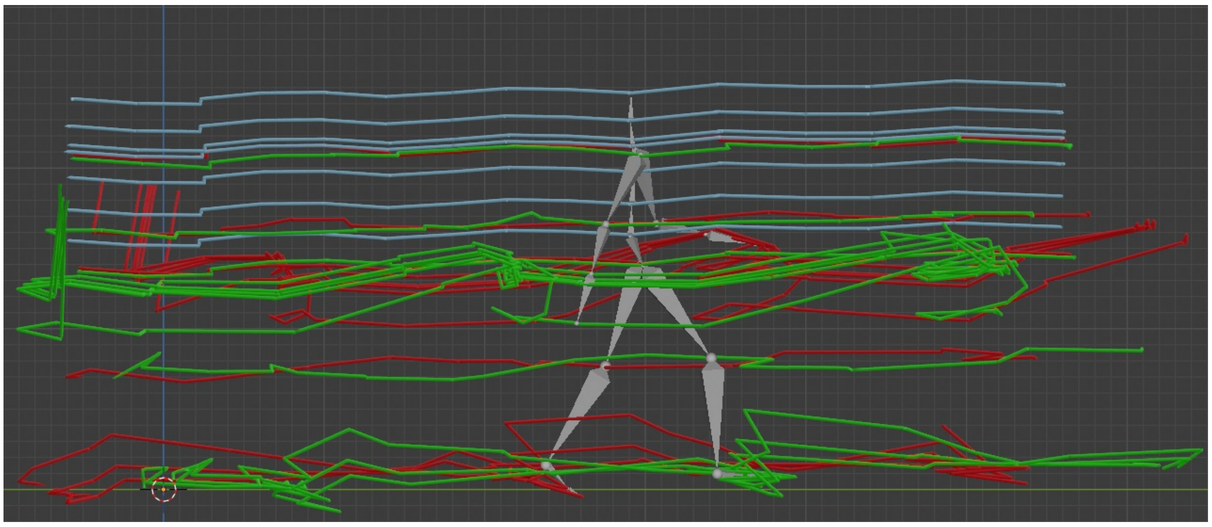
4.1 Compression and Resolution Thresholding

Here is a display of the original motion curves for the source motion capture we used for our tests : (green for the right side, red for the left side, and blue for the middle)



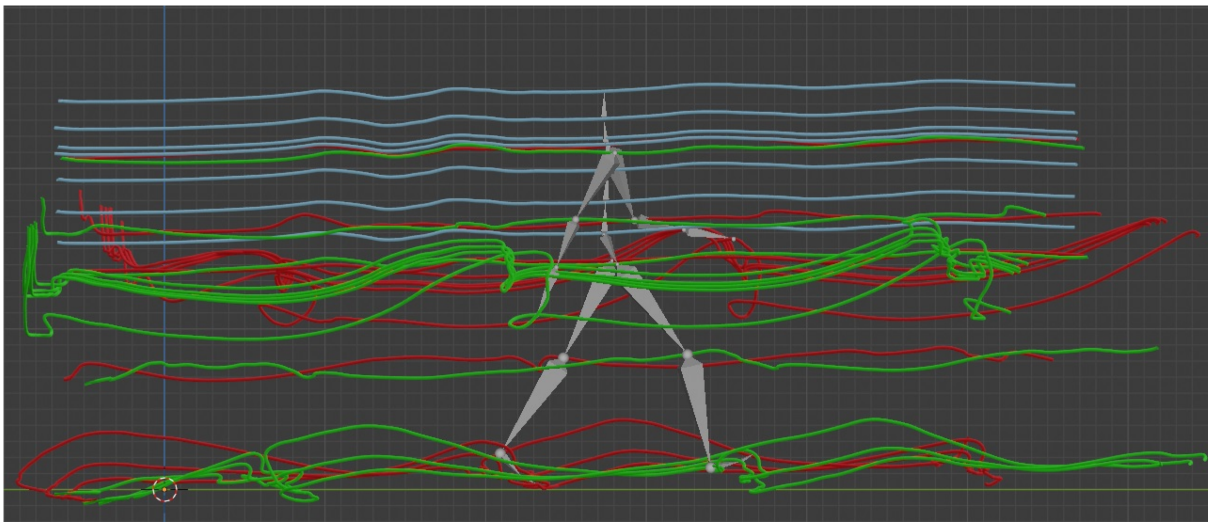
We did a test by compressing the movement with 95% compression (deleting 95% of the coefficients of the wavelet decomposition) on both rotations and positions, with constant padding in the decomposition. We discuss the padding later with videos.

With Haar mother wavelet :



The motion is discretised in time and space. The curve still looks continuous because we used NURBS, but the motion is not. The positions correspond to keyframed poses in the original animation, there are just less keyframes in a way. They don't necessarily fall in the same place because the location is discretised too.

With Daubechies 4 mother wavelet :



The motion is much smoother than with Haar, which is what we wanted given how Daubechies wavelets are constructed. It seems restricting the resolution tries to smooth the small movements and discontinuities of the original animation with larger curvatures, which leads to strange behaviors in places with small motion loops. Globally it is still much nicer to look at than the Haar compression, and is perceptively closer to the original motion, especially in the time dimension.

We did a little video to show what the compression does to the rotations of a motion :

https://youtu.be/aOh5H_wZ8MM

Globally it behaves as expected, removing details in the motion as the compression ratio increases, depending on what mother wavelet was used to compress. Note that it still translates smoothly with Haar because we only interpolated rotation. The position is directly copied from the source. If we compress positions with Haar, it gives discrete positions as expected.

Also, the offset between armatures is purely artificial. The global position of the objects is translated by a constant which has nothing to do with animation to allow for better visualisation (there is a difference between the armature object position and the hip bone position. The armature position is constant throughout animation, and thus can be changed without interfering with animation data).

An unexpected behavior we noticed while testing and wanted to show in this video was the time-boundary distortion for very high compression ratios. Noticing how there was only a problem at the extremities of the animation, we looked what padding we were using for the wavelet decomposition, and it happened to be 'zero-padding', that is it considers there are zeros on both end of the input vector to extend it in its calculations. Our hypothesis is that since it introduces a discontinuity, the wavelet representation might be disturbed.

For our next experiment, we will not use the compression functionality, but rather the resolution thresholding one. We motivate that choice by not wanting global distortion from removing coarse-level coefficients to interfere with the boundary distortion. We only keep the coarse level, setting all others to zero. Three paddings are compared : zero-padding, constant-padding, and symmetric-padding.

We introduce the position interpolation here, since as we quickly mentioned in last part it makes the boundary distortion much clearer. Here is the video :

https://youtu.be/mOQbo9Sor5Q

In conclusion, the zero-padding was responsible for the distortion. Symmetric-padding is better but still has perceivable boundary distortion. For us, in this test, constant-padding introduced the least positional distortion, so we decided to use it for the following experiments (it is also a hardcoded parameter in our script not modifiable via the UI since we felt the latter should be more user-oriented).

We also wanted to show a decomposition of the motion in the Haar basis, but it is not easily done, so we tried to give an idea of it with resolution thresholding from level 8 to 0 :

https://youtu.be/c3eX_8u9KGw

4.2 Motion Blending

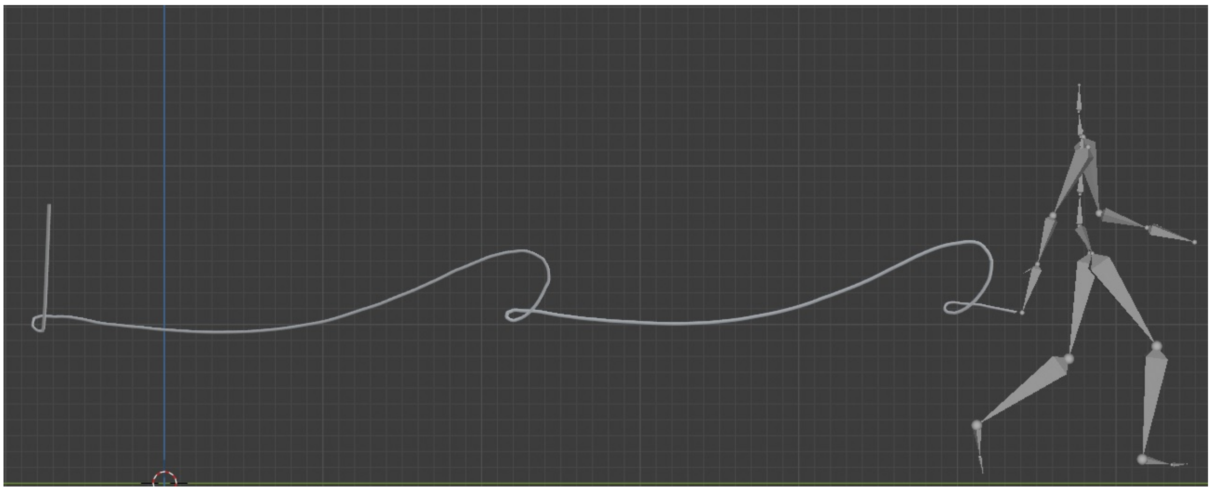
For a first test to show the most extreme case of motion blending, we transferred a very high frequency wrist movement from a static armature onto the motion capture animation. We also introduced a slow arm motion to show that by taking only high wavelet deconstruction levels, slow movements are lost.

Here is a video of the experiment :

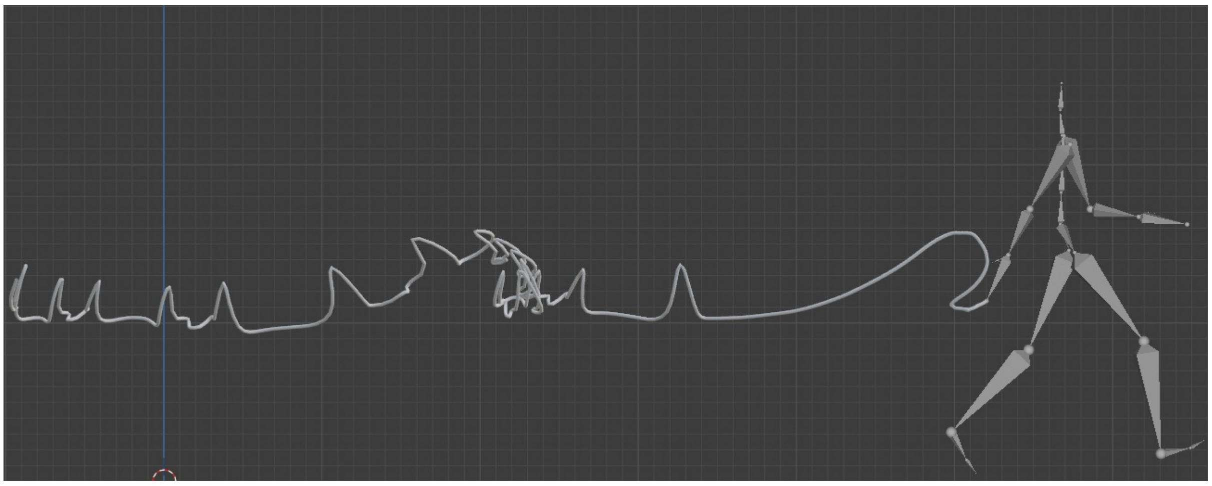
https://youtu.be/KtNqe5GUV7M

and below are static illustrations of the perturbations we introduced on the motion capture data, using motion curves display :

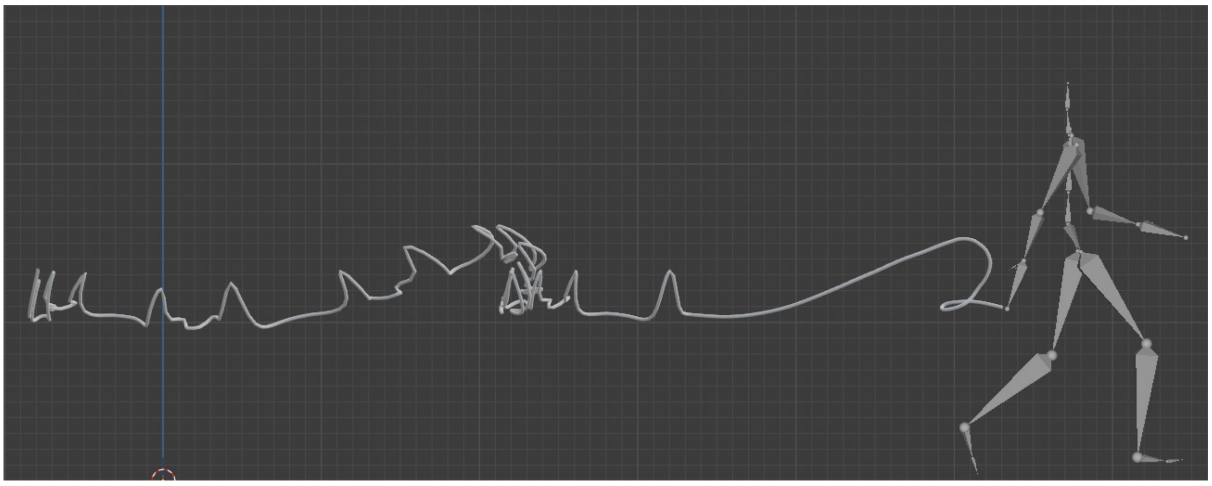
base motion



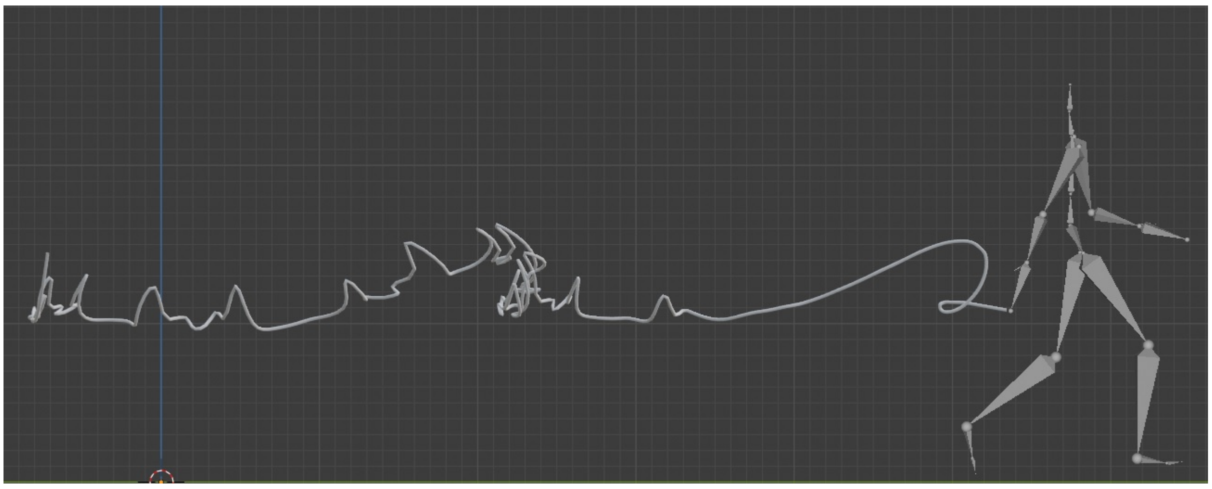
transferred from 2nd resolution level : all details from B



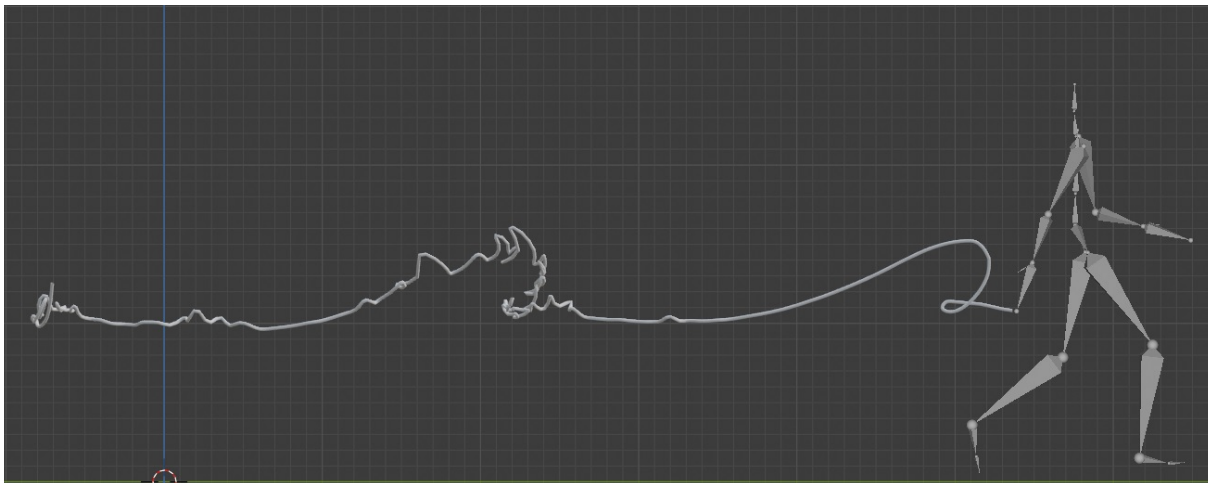
transferred from 3rd resolution level



transferred from 4th resolution level



transferred from 5th resolution level



Firstly, we see that we introduced shakiness while preserving the global motion, which is what we wanted.

A noticeable effect is that by taking all details from the static armature with wiggly wrist introduces differences in the motion capture animation just by removing the details from the motion capture. We see this effect in the second half of the motion curves, when the wrist armature is totally immobile. This perturbation is akin to compression. By taking a higher level, this effect is mitigated. With the 3rd level, we keep nearly all the wrist action whith a better preservation of the original global motion as seen at the end of the animation. However, at 4th level, the preservation is even better, at the cost of some more dampening in the wrist motion. Finally, at 5th level, we do not gain any noticeable preservation as we are already very close to the original, but the dampening becomes very noticeable.

It seems that with this technique everything is a matter of balance. If we want to add something at a time-resolution level, something has to be lost at that level in exchange. It is the principle of linear combination after all. We are not freed from this constraint using the method of the article, however we can offset its effects by selecting different resolutions, compared to a basic proportional blending on the whole motion.

4.3 Application Case

We tried the example use case stated in the article of transferring human motion details onto a hand-made animation. We are not professional animators so we do not expect it to look natural, but it should be a fun conclusion to our experiments.

The animation we made has two parts. In the first half, the character is somewhat walking, like in the motion capture, and in the second part, it plummets to the ground. The idea is to see how the transfer fares on a different kind of motion.

Here is the video :

https://youtu.be/pApMqEaVNVs

With this example, we saw how an artist might fiddle around with the different resolution levels to get the result they want from the combination using the method from the article and our interface. Actually a strong point of that method is that it is actually quite intuitive on the user end even without understanding how wavelets work.

Our plugin is not usable in a practical way though, mostly because it is too slow (the loading times were sped up in the videos), and because its range of applications is greatly shrunk from not having the time-warping quickly mentioned in the article.

Thank you for reading our report and watching the videos. You can install our project on your machine following the setup instructions at the beginning to try it out and test with other animations than the ones we show in the report.

We have heard other groups were going to use Blender so it might be worthwhile to install it and do the steps to have access to pyWavelets inside. If you have trouble setting up the project, do not hesitate to contact us.