Animating AZee Descriptions Using Off-the-Shelf IK Solvers

Fabrizio Nunnari[1], **Michael Filhol**[2], **Alexis Heloir**[1,3]
[1]DFKI/MMCI/SLSI Group, Saarbrücken, Germany,
[2]LIMSI, CNRS, Université Paris–Saclay, Univ. Paris–sud, bât. 508, 91400 Orsay, France,
[3]LAMIH UMR CNRS 8201 Université de Valenciennes, 59300 Le Mont Houy, France
fabrizio.nunnari@dfki.de, michael.filhol@limsi.fr, alexis.heloir@dfki.de

Abstract

We propose to implement a bottom-up animation solution for the AZee system. No low-level AZee animation system exists yet, which hinders its effective implementation as Sign Language avatar input. This bottom-up approach delivers procedurally computed animations and, because of its procedural nature, it is capable of generating the whole possible range of gestures covered by AZee's symbolic description. The goal is not to compete on the ground of naturalness since movements are bound to look robotic like all bottom-up systems, but its purpose could be to be used as the missing low-level fallback for an existing top-down system. The proposed animation system is built on the top of a freely available 3D authoring tool and takes advantage of the tool's default IK solving routines.

Keywords: Sign Language Synthesis, Signing Avatar, AZee.

1. Introduction on Signing Avatars

In signing avatar technology, current approaches for the creation of sign repositories can be generally described as *pre-animated* or *synthesised*. Solutions embracing the pre-animated approach start from an analysis of full sentences, which are then segmented at a coarse, lemma level. Very large repositories are populated by captured or manually authored Sign Language (SL) animation clips. SL generation is then performed by sequentially stitching together the available segments. In contrast, solutions embracing the synthesized approach are derived from a linguistic representation. This leads to a concise set of atomic animation elements which are symbolically described in a high level declarative language. In this case, SL generation consists of a procedural realization of the symbols composing first signs and then full sentences.

Because they have been manually authored or captured on human performers, pre-animated approaches usually deliver animations which are perceived by end-users as more realistic and natural. However, they cannot extrapolate much beyond the set of low level pre-stored animation clips. On the contrary, synthesised approaches deliver procedurally computed animation which are perceived as stiff and robotic by end-users, but, because of their procedural nature, they are presumably capable of generating the whole possible range of gestures covered by their symbolic description.

In both cases, a clear requirement for any system today is that it be able to animate various articulators on the body and face, with flexible timing patterns. By this we mean, for example, be able to control all communicative channels simultaneously (hands, eyes, lips, and others), but *not* all sharing the same time boundaries.

In this paper, after a survey of the related work (Section 2.), we present in Section 3. how the AZee system is designed to: i) realize animations that feature interleaving communicative channels, and ii) is designed to be part of a generator interchanging synthesis with pre-animated sequences. This novel approach has been so far only a design proposal. In Section 4., we describe how we used a popular open-source 3D editor and its integrated Inverse Kinematic solver to generate AZee animations. The implementation is a work-in-progress. Section 5. illustrates preliminary results in the realization of static poses together with performance tests. Finally, Section 6. concludes the paper.

2. Related Work

Signing avatars have been under development for more than a decade. One of the first working systems was JASigning[1] from the Visicast (Jennings et al., 2010) and the DictaSign (Efthimiou et al., 2010) projects: it uses only the *synthesised* approach and is able to produce signing animation from SiGML (Hanke, 2004) statements as input. SiGML is a digital representation of the Hamburg Notation System (HamNoSys) (Prillwitz et al., 1989), which is a graphical formalism for the description of Sign Language using a set of pictograms. By design, HamNoSys is an oversimplification of sign language, describing sentences as a sequence of glosses. Only within glosses there is a parallelization between manual (hands, fingers) and non-manual features (eyes, lips, nodding, ...). As such, resulting animations are generally perceived as unnatural.

The following technologies tried to overcome the strong limitations of approaches based on pure-synthesis (i.e., the complete lack of recorded data) or pure-pre-animation (i.e., the impossibility to parameterized signs) by injecting elements of one approach into the other.

EMBR (Heloir and Kipp, 2009) was born as a tool for the synthetic animation of interactive virtual agents and was later employed in the generation of sign language animation (Heloir et al., 2011). It has been recently extended to support the playback of pre-animated facial movement (Kacorri and Huenerfauth, 2014), thus has become a mix between the two techniques. Its animation description language is not overspecialized for sign language, hence it offers more flexibility in the configuration of body postures. However, the SL generation is basically still performed

[1] `http://vh.cmp.uea.ac.uk/index.php/JASigning` – 23 Feb 2018

through a concatenation of poses and as such suffers of the same limitations as JASigning.
The player developed for the project ATLAS (Lombardo et al., 2010; Lombardo et al., 2011) is based on a repository of pre-animated clips. It supports sign parameterization in two ways. First, it allows for the overlapping of different animation tracks on top of the sign animation clip, allowing, for example, to control the animation of eyebrows and facial expressions independently from the arms. Second, it performs on-the-fly editing of the stored animation curves. For example, a sign can be relocated by applying an offset to the position of the hands for every frame of the clip. If the edits are applied on a limited range of deformation, the resulting motion looks still very natural and pleasant. However, the system lacks of any pure procedural synthesis of movement which is not exported from an animation editor.
The Paula system (Wolfe et al., 2011; McDonald et al., 2016), too, uses a multi-layered animation approach. The various layers can control the full body, with layers at higher priority overriding the animation of more basic layers. Each layer can playback pre-animated data as well as apply procedural control to body parts with routines tailored to support sign language animation (e.g., eyebrows adjustment and spine/torso rotation). Again, the range of possibilities of the player are limited to its database of motion clips and its hard-coded animation controllers, but can not be scripted via a high-level language.
These three above mentioned applications present a hybrid approach between the synthesis and the pre-animation. However, rather than deliberate architectural planning, it appears that one approach has been integrated into the other as a later attempt to overcome the limitations of an initial design choice. Additionally, none of those systems gives the possibility to independently drive different communicative channels on different time boundaries.
The only exeption we are aware of, is the design proposed by Filhol et al. (Filhol et al., 2017), where they explicitly designed from the very beginning a generative strategy accommodating both approaches together. A system that: i) is based on pre-animated signs, but ii) is ready to fall back on pure synthesis when pre-recording is not possible. Additionally, the system iii) do not enforce shared time boundaries on all the communicative channels.

3. The missing bottom-up system

Generic Sign synthesis platforms are designed to combine low-level (roughly, phonetic) features into larger pieces (lexical signs for the most part), stitched together in sequence to build utterances. For example, JASigning takes a string of units for input, labelled with glosses, each described with HamNoSys. In such phonetically inspired descriptions, a "Z" movement drawn in the vertical plane is composed with two horizontal strokes separated by a downward diagonal stroke. These individual strokes are part of the relatively small number of primitives of the language, but highly reusable for all sorts of movement descriptions. In this paper, we call such approach *bottom-up* animation, because it builds from the smallest possible features. Provided enough of these primitives—though by design, not necessarily plenty—the advantage is that one can describe everything by combination of the primary features. The work we propose here will also fall in this category.
On the other hand, the problem with bottom-up systems is that they inevitably render robotic animations. It comes from the fact that while humans may think of and describe certain movements as circles, straight movements or fixed orientations, actual human motion never follows its idealised geometric description. Animated as such, they do not look human, and there is no known generic way of taking the intended geometry and distorting it to look natural. The solution to provide naturalness is rather to make use of larger dedicated procedures or even full play-back of pre-recorded chunks, already implementing the human deviation from the idealised forms. This mostly advocates against bottom-up approaches all together, and for use of higher-level entries to avoid building complex movements from scratch.
Filhol et al. (Filhol et al., 2017) have recently reported testing such approach to naturalness for Sign Language with the Paula animation system. It works from:

- AZee, a language to specify linguistic input without fixed lexical signs and allowing more than merely listing Stokoe-style parameter values;
- the principle "the coarser the better", by which the larger the chunk of pre-animated data is, the better candidate it is for natural output;
- and the animation system Paula.

3.1. AZee input

AZee is a language to write parameterised signed forms for semantic functions. This can capture descriptions such as HamNoSys lexical entries (the fixed Stokoe-style description is the form; the gloss the meaning), but also more relational functions such as "not-but(X, Y)", meaning "not X but Y" and producing the form (say F) synchronising Y after X and a headshake and deep gaze in between, over a manual hold of X.
An AZee input for synthesis is typically a recursive nesting of semantic functions capturing the meaning of the production. For instance with the function above, plus "tree" and "wardrobe", one can build the following expression to mean "not a tree but a wardrobe":

not-but(tree(), wardrobe())

Evaluating this expression with an AZee parser produces a score, in this case F with blocks X and Y instantiated with the results of "tree" and "wardrobe" respectively, as illustrated in Fig. 1. The contents of a block is either:

- itself a score of the same recursive type, thus itself synchronising blocks on its nested time line (see outer boxes in the diagram);
- or a set of low-level constraints, which stops the recursion (grey-filled boxes).

The relevant constraints for this work are:

- placements of linguistically relevant body points (called *sites*) in target locations;
- bone orientations, e.g. orient normal vector of palm upright.

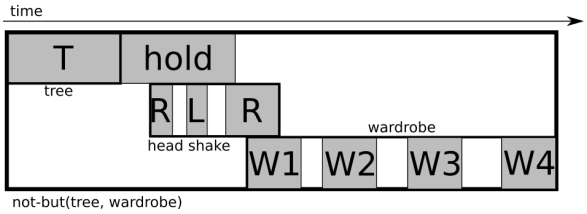

Figure 1: Example of AZee score.

For example, T in the diagram, representing the set of necessary and sufficient articulations constraints for the meaning "tree", is likely to include an orientation of the strong forearm up, a placement of the strong elbow on the weak palm, etc.

3.2. A top-down system

The idea Filhol & McDonald (Filhol et al., 2017) follow is to work with larger blocks rather than low-level features combined to produce synthesis. But it is not trivial to decide which blocks should be used. While the larger they are the more natural they look, the less feasible it becomes too because it is never possible to have everything prerecorded. Taking advantage of the recursive structure of AZee resulting scores, the authors address this problem in the following fashion.

What they do is start at the top level of the AZee score and work their way down the nested block structure until matches are found for blocks they have animations for (they call this a "short-cut"). At each level, if the full block is not matched, it is looked into and its constituents (sub-blocks) are layered on the animation score, each checked individually for a match, and so on. In contrast to what is done when building from small features to reach large blocks of utterances (bottom-up), this opposite approach can be called *top-down* animation.

3.3. What is missing

A top-down search for the most natural chunks guarantees that the highest usable blocks is used when appropriate, and never used when not. However, this system for the moment assumes a shortcut is possible at some point down the block structure, and that the bottom (low-level constraints specified in the non-breakable blocks) will not be met.

The problem is that actual SL generation systems involve blocks which cannot all be fully listed or recorded beforehand: infinite variation in continuous spaces, depiction, etc. AZee captures well these features with arbitrarily complex geometric expressions that are generally impossible to shortcut.

We propose to implement a bottom-up AZee animation system. No low-level AZee animation system exists yet, though it would ensure that anything can be animated from AZee, regardless of what we are ever able to shortcut. The goal is not to compete on the ground of naturalness since movements are bound to look robotic like all bottom-up systems. But its purpose could be to be used as the missing low-level fallback for a top-down system like Paula. This way we would take advantage of top-down shortcutting whenever possible, and still guarantee an output from AZee input when it is not, using bottom-up generation from low-level specifications.

4. Synthesising animations from AZee scores

One part of AZee which was missing so far–and described in this paper–is the implementation of the code realizing the skeletal poses for the keyframes delimiting the interpolation blocks.

The implementation of a high-quality gesture realizer encompasses a number of non trivial Computer Graphics and Animation techniques such as real-time rendering and shading, direct and inverse kinematics of complex kinematic chains, declaration and management of joint boundaries, collision detection, keyframe and timeline management, parameterizable interpolaton between animation curves, etc. All these features being available in the open source and liberally licensed Blender 3D editor[2], our AZee realizer is built in Python on top of the Blender API and could be viewed as an interface between AZee and Blender.

4.1. From AZee scores to an animation timeline

The AZee parser translates an AZee expression into a *score*, which is a set of timed intervals (blocks) whose boundaries are layed out on a timeline and whose contents is either:

- itself a nested score, such as "headshake" inside the "not-but" box in Fig. 1 (recursive case);
- a set of low-level constraints such as "R" inside the "headshake" box (base case).

The base case constraints include articulations (bone orientations and site placements) that can be thought as inverse kinematics (IK) problems in computer animation terms. Our goal is to translate those IK problems into (forward kinematics) joint rotations, and to position the full-body posture correctly on the final animation timeline.

This means create the right keyframes with the right set of constraints pulled and translated from the AZee resulting score, then relying on the system's native interpolation capability to fill the intermediate frames on the timeline. The first step to do so is to *flatten* the score from its nested and multi-linear structure so that it is projected on a single time line, as illustrated in Fig. 2.

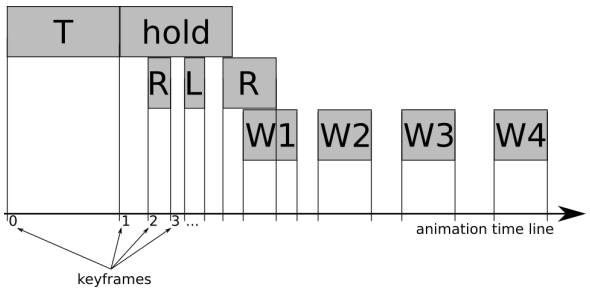

Figure 2: A *flattened* AZee score.

For every animation keyframe created on the timeline, we must copy all constraints that apply at that moment in time.

[2] `https://www.blender.org` – 23 Feb 2018

In our example, keyframes 0 and 1 will contain the set of constraints "T". Keyframes 2 and 3 will require not only a copy of "T" but also those from "R" since both apply at those dates on the flattened timeline.
At this point it becomes impossible to determine to provenance of the constraints packed in the keyframes. This is the step that would involve a crucial loss if we were aiming at naturalness because we will be relying on simple interpolation to fill all blanks. It is the reason for the expected robotic motion, and why it is not done by the top-down system presented above (working with the nested block structure). On the contrary, we are aiming at enabling synthesis from native constraints, so we accept this loss of naturalness to secure the possibility of an output.

4.2. Definitions: joints, sites, and IK-Problems

AZee defines the skeletal structure of a human signer in terms of *joints* and *sites*.
A **joint**, as in any 3D skeletal animation system, is a data structure characterized by:

- a *name*, unique for each joint of a skeleton;
- a *parent* joint, possibly *null* if the join is the root of the skeletal system;
- an *offset* from the parent joint. For the *root*, the offset represents the joint's absolute position relatively to the origin of the axes frame;
- a *rotation*, expressed by Euler angles for pose editing and then translated into a quaternion for better automatic interpolation;
- a set of *rotation limits*

  $$\{(R^x_{min}, R^x_{max}), (R^y_{min}, R^y_{max}), (R^z_{min}, R^z_{max})\}$$

  expressed as minimum and maximum rotation angle along the three Euler axes.

In this structure, a **bone** is essentially the segment connecting two joints with a parent-child relationship.

In addition to joints and bones, a **site** is defined as:

- a *name*, unique for each site;
- a *parent* joint from which the site depends;
- an *offset* from the parent joint.

Essentially, a site is a point in space whose absolute location depends from the global position and rotation of the parent joint. It is used to identify key points on the skin of the virtual human (e.g., the tip of a finger, or a corner of the mouth) that are used as reference for placement tasks. When a joint rotates, the linked bone, together with all the children joints and sites, change their absolute position.

The set of available bones and sites is described in two dedicated maps (*bones-map* and *sites-map*) thus separating the high-level namespace of the entities which can be addressed by AZee statements from the low-level hierarchical skeletal structure. This makes possible to seamlessly substitute the underlying character if a proper remapping is performed.

When AZee generates a keyframe on the timeline, the posture of the signer at that specific keyframe is described by a set of **IK-Problems**. An IK-Problem is defined by:

- an *IK-chain*, which is an ordered list of joints where the *start* of the chain, its first joint, is the parent of the second, the second is parent of the third, and so on. The last element of the chain can be a *site*, in which case it is called the *end-effector* of the chain;
- one optional *place constraint*. If present, it requires the end-effector of the chain (a site) to be positioned at a specific 3D point in space;
- a set of *rotation constraints*, possibly empty. Each rotation constraint requires either the *direction* or the *normal* of a bone to be *parallel* or *perpendicular* to a given 3D vector. The bone must be part of the IK-chain. Each bone can have at maximum two rotation constraints, in which case they will constrain independently both the direction and the normal of the bone.

AZee instantiates IK-Problems as an ordered list, taking into account their inter-dependencies. For instance with the set of constraints T in our previous example of the tree, at least two IK problems will be built, involving respectively:

- a placement of the weak hand at the target location of the tree;
- a placement of the strong elbow in contact with the weak hand palm.

The latter depending on the former, they will appear in this order in the list so that they can simply be applied in the given sequence. This way, the target placement of the strong elbow becomes a mere look-up of the current state of the avatar.
Determining the final pose of the virtual signer now requires solving, in the given order, all the IK-Problems defined on a given keyframe.

4.3. Solving single constraints

As introduced before, an IK-Problem is composed of one optional place-constraint and zero or more rotation-constraints.
The resolution of a place constraint yields to the resolution of the most classical of the IK problems, similar to robotics, where the end-effector of the ik-chain must be positioned in a 3D location. The IK solver calculates the rotation of all the joints of the chain.
Differently, solving a rotation-constraint for a bone means setting the absolute rotation in space of the bones's parent joint (e.g. to orient the forearm, we need to set the absolute rotation of the elbow). The IK-solver will determine the relative rotation of all the joints of the IK-chain up to the beginning of the chain. There are two cases, the first being when two rotation constraints are set at the same time on a bone. Forcing both the direction and the normal of a bone implies a unique possible absolute rotation for the bone's parent joint. In the second case, when only either the direction or the normal of a bone are set, the constraint is more relaxed and there are infinite solutions.
The three cases presented above (one for placing and two for rotation) can be individually solved by existing IK libraries. In our case, we use the iTaSC solver integrated in

```
1  def apply_ik_problem(skeleton, ik_prob):
2
3      # Outer iteration: apply all constraints of an IK-Problem.
4      iter_count = 0
5      while iter_count < MAX_ITER:
6          # Inner iterations: use the Blender IK to satisfy the single constraints.
7          if ik_prob.place_constr != null:
8              ik_prob.place_cstr.apply_to(skeleton)
9          for rot_constr in ik_prob.rot_cstr_list:
10             rot_cstr.apply_to(skeleton)
11
12         # Measure the ''distance'' between the current skeleton configuration
13         # and the the desired position/orientations
14         place_offset = ik_prob.place_cstr.offset_to_goal(skeleton) if ik_prob.place_cstr
               != null else 0.0
15
16         rot_offsets = [rot_cstr.offset_to_goal(skeleton) for rot_cstr in ik_prob.
               rot_cstr_list] if len(ik_prob.rot_cstr_list) > 0 else 0.0
17
18         # If all the distances are below the threshold, break the iteration
19         if place_offset < PLACE_THRSHLD and max(rot_offsets) < ROT_THRSH:
20             break
21
22         iter_count += 1
```

Listing 1: The algorithm describing the strategy to solve an IK-Problem.

the Blender software in SDLS mode. The iTaSC IK solver was originally developed by De Shutter et al. (De Schutter et al., 2007) and its integration in Blender is documented online[3].

4.4. Solving an IK-Problem

While solving the single constraints composing an IK-Problem is doable with off-the-shelf libraries, fulfilling the whole set of constraints in a single pose is not supported by the Blender iTaSC solver. In order to solve a whole IK-Problem into a final posture, we elaborated an algorithm whose pseudo code is shown in Listing 1.

The general strategy is to start by sequentially applying both the placement and all of the rotation constraints on the target skeleton. Each time a single constraint is applied, it is likely to break the position achieved by a previous constraint. Hence, we re-iterate the application of the single constraints until the final desired position is achieved.

We call this approach *two-level iteration*. The first *outer* level of the iteration begins at line 5. The second *inner* levels are immersed in the two invocation of the `apply_to` function at lines 8 and 10, where the basic (place or rotation) constraints composing an IK-Problem are solved using the Blender iTaSC solver. The `apply_to` function uses the Blender API to: i) create a *bone IK Constraint* on the Blender skeleton; ii) create a *target* object to drive the end-effector position or rotation; iii) trigger the execution of the iTaSC IK-solver, which solves the problem through a number of iterations (whose maximum value is set in the Blender properties); and iv) remove both the target object and the bone IK constraint.

[3] `https://wiki.blender.org/index.php/Dev:Source/GameEngine/RobotIKSolver` – 23 Feb 2018

The resolution takes automatically into account the joint rotation limits, which are applied as IK rotation limits in each Blender bone properties during a setup stage.

Line 14 computes the distance between the desired and the actual position of the site/end-effector.

Line 16 computes, for each joint in the IK-chain, the rotational distance between the current and the desired rotation. The rotational distance, which is an angle, is computed by first computing the quaternion needed to shift from the current to the desired rotation. The quaternion is then decomposed into an axis-angle representation and the angle in degrees returned.

Line 19: if all the distances are below the respective thresholds, the current pose is considered to be close enough to the desired one and the outer iteration breaks.

4.5. Implementation

The AZee animation system presented in this paper is implemented as add-on for Blender (v2.79). Figure 3 shows the GUI for AZee authors. On the right side, the author can move a virtual camera and see a 3D preview of the generated animation. A side panel shows buttons to setup the system, tune IK convergence parameters, and other debug flags. On the left, the author can insert AZee statements and execute them by clicking a button. At the bottom, a timeline marks where the AZee interpreter creates keyframes.

The current implementation operates on a prototype skeleton specifically developed for the AZee development. Future versions will address the problem of directly animating any imported human skeleton.

In our tests, we were able to solve an IK-Problem in a fraction of a second on an Intel(R) Core(TM) i7-3635QM CPU@2.40GHz (computation is limited to one core by the Blender architecture) and DDR3 RAM@1600Mhz

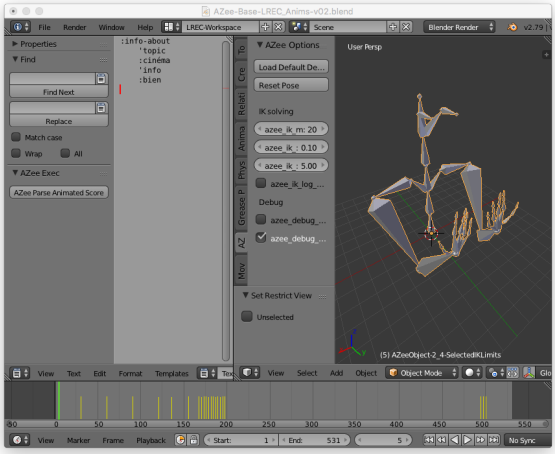

Figure 3: The AZee workspace in Blender.

Figure 4: Sign *tree* applied to a skinned skeleton.

with the following constant values: `MAX_ITER=35`, `PLACE_THRSHLD=1cm`, `ROT_THRSHLD=5degs`. As reported in detail in the next section, the computation time raises to several seconds when realizing a full signs or sentences. We are performing further tests in order to determine a trade-off between high precision (lower thresholds) and low realization time (lower number of iterations allowed).

5. Examples

In this section we present two working examples of the AZee animator. The first example is a static pose and aims at illustrating the mechanism of the IK resolution algorithm. The second example presents a more complex animated sentence and aims at illustrating the capability of AZee at interleaving communication channels.

5.1. Example1: tree

We report the results for the realization of the sign *arbre* (tree) in French Sign Language (see Figure 4). As shown in Figure 5-top, the sign requires the computation of 11 IK-Problems. The first two contain only a placement constraint: IK-Problem #0 asks for the palm of the left hand to

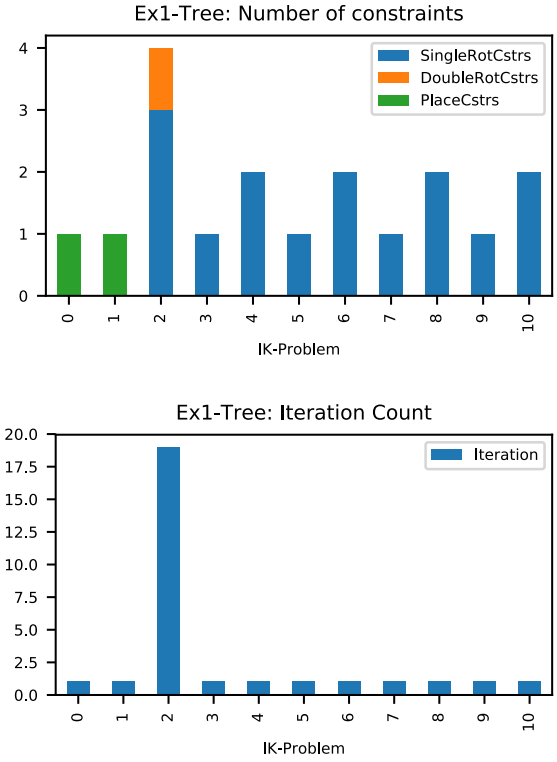

Figure 5: Sign *tree*. For each IK-Problem: (top) the count and the type of constraints, and (bottom) the number of iterations needed to solve the problem.

be located in front of the signer, while IK-Problem #1 asks for the tip of the right elbow to touch the palm of the left hand. IK-Problem #2, the most complex, aligns at the same time the right forearm, hand, and thumb. The remaining IK-Problems straighten the four remaining fingers along the direction of the hand.

Figure 5-bottom reports the number of outer iterations needed to solve each IK-Problem. Ten out of the 11 IK-Problems solve with only 1 iteration, while IK-Problem #2 needs 19 iterations. This is expected, because problem #2 has 4 constraints and the application of each constraint is likely disrupting the orientation of the previous ones. In detail, IK-Problem #2 operates on an IK-chain going from the tip of the thumb to the elbow, and its constraints involve:

1. The orientation of the thumb tip, which must point outwards;
2. The direction of the normal of the thumb middle phalanx, which must face up;
3. The orientation of the hand, which must be aligned with the forearm;
4. The orientation of the forearm, which must point vertically up.

Figure 6 shows, for the sign *tree*, the side view of the position of the skeleton after a progressive number of iterations. Figure 7-top shows how the maximum rotational distance decreases as more iterations are executed. Figure 7-bottom shows the execution time for each iteration: for IK-Problem

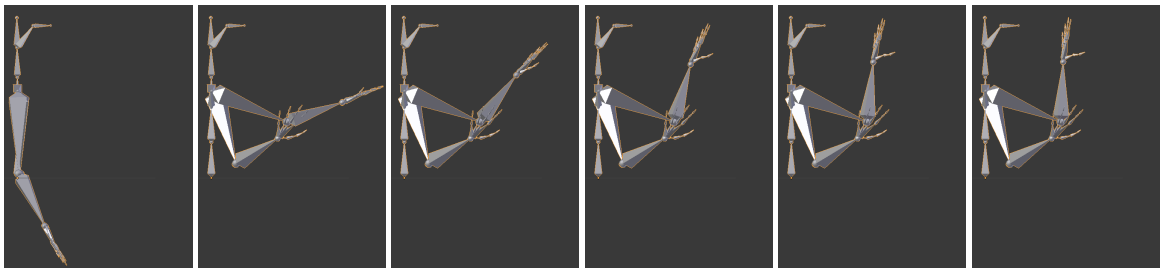

Figure 6: Sign *tree*: convergence to the final position from the resting pose (left) and after 1, 5, 10, 15, and 19 iterations.

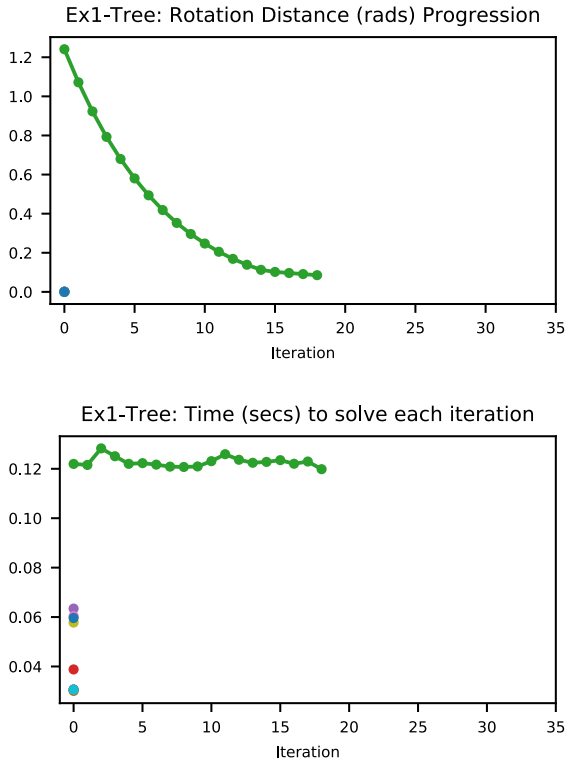

Figure 7: Test plot sizes.

#2 the execution time is stable at ca. 125ms for each iteration, while for all other IK-Problems the time lies between 25ms and 75ms for their first (and only) iteration. Overall, the interpretation of the sign pose took 2.76s to execute 29 iterations, with an average of 95ms per iterations (SD=39).

5.2. Example 2: cinema + good

The second example translates the sentence "The cinema is/was good":

```
:info-about
        'topic
        :cinema
        'info
        :bien
```

The top-level rule "info-about" produces a simple sequence of the two contained items, with a timed transition and an eye blink towards the end of the second item. IK is not be invoked for the final blink, and both contained items involve a classic sequence of manual postures at timed keyframes. At each keyframe, a number of IK problems is ordered.

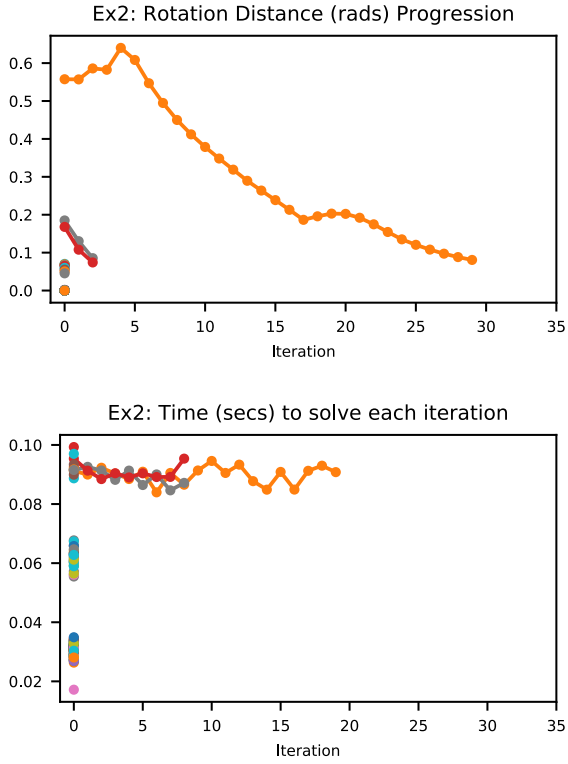

Figure 8: Test plot sizes.

The sentence required the resolution of 152 IK-Problems, of which 3 required multiple iterations. Figure 8-top shows the rotational distance progression, while Figure 8-bottom shows the execution time for each iteration. The hardest IK-Problem converged after 35 iterations. In general, all iterations are executed within ca. 100ms. Overall, the interpretation of the sentence took 9.97s to execute 185 iterations, with an average of 52ms per iteration (SD=26).

6. Conclusion

The work presented in this paper is the latest significant progress achieved in the implementation of a complete AZee realizer. This realizer is a step forward towards the implementation of a new generation of Sing Language synthesizers, allowing for the animation of different

communicative channels which interleave on the timeline without the limitation of the hard boundaries dictated by lemma-based transcriptions.

Thanks to its integration in the Blender software, the realizer will allow for a streamlined and integrated generation of high quality rendered animations. Concerning the performances, from our tests it appears possible, on modern hardware, to generate the animation curves of a full sentence with limited delay and export them for playback on a turn-based interactive system.

The current implementation, still under test and development, behaves well in the realization of static poses, but still presents some glitches during the animation process, mainly occurring when the joints orientation get close to the boundaries of the IK rotation limits.

As described before, AZee separates the low-level description of the skeletal structure from an high-level namespace of *bones* and *sites*. Hence, AZee signs can be applied to different characters of any size and proportion without the need of facing retargeting problems. Given a new virtual interpreter, only the bones and sites name mapping must be updated. Certainly, the application to arbitrary avatars will suffer of self-compenetration issues, which must be addressed, for example, through the use of collision-prevention systems and the simulation of soft bodies.

This addresses one of the most frequent limitations of SL projects, which is the impossibility to interchange data between different virtual interpreters and database of animations. It makes AZee a good candidate language to create a database of animated signs which can be reused by any research group on SL, to animate their own avatar.
From the point of view of signal processing, AZee can be considered as a *lossy animation compression* format. Although focusing on the description and compression of sign language animations, it is possible to imagine Azee (or a variant of it) applied in the description of casual gesturing in general-purpose conversational agents. In the future, it might be possible to work on a system which automatically derives AZee descriptions from existing animation curves, allowing for the seamless transfer of sign repositories across different virtual interpreters.

7. Bibliographical References

De Schutter, J., De Laet, T., Rutgeerts, J., Decr, W., Smits, R., Aertbelin, E., Claes, K., and Bruyninckx, H. (2007). Constraint-based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty. *The International Journal of Robotics Research*, 26(5):433–455, May.

Efthimiou, E., Fontinea, S., Hanke, T., Glauert, J., Bowden, R., Braffort, A., Collet, C., Maragos, P., and Goudenove, F. (2010). Dicta-sign–sign language recognition, generation and modelling: a research effort with applications in deaf communication. In *Proceedings of the 4th Workshop on the Representation and Processing of Sign Languages: Corpora and Sign Language Technologies*, pages 80–83.

Filhol, M., McDonald, J., and Wolfe, R., (2017). *Synthesizing Sign Language by Connecting Linguistically Structured Descriptions to a Multi-track Animation System*, pages 27–40. Springer International Publishing, Cham.

Hanke, T. (2004). HamNoSys-representing sign language data in language resources and language processing contexts. In *LREC*, volume 4.

Heloir, A. and Kipp, M. (2009). EMBR - A Realtime Animation Engine for Interactive Embodied Agents. In *Proceedings of the 9th International Conference on Intelligent Virtual Agents (IVA-09)*.

Heloir, A., Nguyen, Q., and Kipp, M. (2011). Signing avatars: A feasibility study. In *The Second International Workshop on Sign Language Translation and Avatar Technology (SLTAT), Dundee, Scotland, United Kingdom*.

Jennings, V., Elliott, R., Kennaway, R., and Glauert, J. (2010). Requirements for a signing avatar. In *Proceedings of the 4th LREC Workshop on the Representation and Processing of Sign Languages*, pages 133–136.

Kacorri, H. and Huenerfauth, M. (2014). Implementation and Evaluation of Animation Controls Sufficient for Conveying ASL Facial Expressions. In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility*, ASSETS '14, pages 261–262, New York, NY, USA. ACM.

Lombardo, V., Nunnari, F., and Damiano, R. (2010). A virtual interpreter for the Italian sign language. In *Proceedings of the 10th international conference on Intelligent virtual agents*, IVA'10, pages 201–207, Berlin, Heidelberg. Springer-Verlag.

Lombardo, V., Battaglino, C., Damiano, R., and Nunnari, F. (2011). An Avatar-based Interface for the Italian Sign Language. In *Proceedings of the 2011 International Conference on Complex, Intelligent, and Software Intensive Systems*, CISIS '11, pages 589–594, Washington, DC, USA, June. IEEE Computer Society.

McDonald, J., Wolfe, R., Schnepp, J., Hochgesang, J., Jamrozik, D. G., Stumbo, M., Berke, L., Bialek, M., and Thomas, F. (2016). An automated technique for real-time production of lifelike animations of american sign language. *Universal Access in the Information Society*, 15(4):551–566, Nov.

Prillwitz, S. L., Leven, R., Zienert, H., Zienert, R., T.Hanke, and Henning, J. (1989). *HamNoSys Version 2.0*. International Studies on Sign Language and Communication of the Deaf.

Wolfe, R., McDonald, J., and Schnepp, J. C. (2011). Avatar to depict sign language: Building from reusable hand animation. January.