Universität des Saarlandes

# Software project: Articulatory Speech Analysis

Report

Author: Manisha Gandhi, Olga Gedsun, & Marina Oberwegner
Email: manishagandhi@msn.com, gedsunolga@yahoo.de, s9mmober@stud.uni-saarland.de

Instructor: Dr. Ingmar Steiner

# 1 Introduction

In the software project 'Articulatory Speech Analysis' we learned how to develop a program which analyzes speech by using the Blender engine. Our project is based on previous studies and developments of how to analyze speech visually.

For example Steiner et al. introduce a model which only uses the tongue, as well as the mandible and maxilla to visualize what happens during the production of a specific sound. The authors used motion capture data which got its information from electromagnetic articulography (EMA). The authors used a mngu0 articulatory corpus which contains 3D EMA data information. Further, the authors state that they used volumetric magnetic resonance imaging scans of the vocal tract of the speaker during the speech production of the speaker. Moreover, the authors used an articulatory animation framework to animate the tongue etc. by using motion capture data which they achieved from the EMA subset. An example of the model they developed could be seen in Figure 1 (Steiner et al., 2012).

Figure 1: Example of an animated articulatory model using EMA data.
**Source:** Steiner et al. (2012)

Figure 2: Example of the 3D model.
**Source:** Steiner and Ouni (2012)

Further, our project is motivated by the study investigated by (Steiner and Ouni, 2012). They present a portable kinematic articulatory model which shows audiovisual speech synthesis based on actual speech data. The authors also only concentrated on showing the set of teeth and tongue movements in their model. The authors used a model containing the most important parts which they obtained from a stock 3D model website. The model is shown in Figure 2. For our project we imitated some of the methods presented by the authors mentioned previously.

# 2 Report written by Manisha Gandhi

### Data Processing

The motion capture data for the artimate project was taken from the msak0 corpus. They consisted of 460 EMA files, which specified the x and y positions

for ten articulators over a number of frames. The articulators included parts of the tongue, the lips and the teeth. To be read to Blender, the EMA files had to be converted to a bvh format. The code for converting the files was written in Groovy, inside the build.gradle script.

Before writing the script, one file was manually typed into a bvh formant, so Marina could test it in Blender. It helped in understanding how the two types of files were formatted and how one could be converted to the other. The header was typed directly, but the data had to be rearranged because the bvh file required z axis values, as well as having the x, y and z values of a root listed together, in that order.

The data was rearranged using a spreadsheet, which stored the values of each channel in a column, meaning that the order of the channels could easily be switched for the whole file and the new z axis columns could be inserted. After some attempts at importing it into Blender and some necessary tweaking, the hand-written file was a success. I moved on to the main task of automatically generating all the files.

The EMA files were numbered from '001' to '460', so they could be read into the script using a loop. The text of the EMA files was processed in two parts: the headers and the data. The header section contained information about the number of frames, number of channels and the names of the channels, which were each x and y axis of the articulators. The data contained for each frame, a row of the channel values in the same order as the channel names in the header.

First empty collections were created, to be used for storing various important parts of the text, including a list for the root names, a list of the channels and the number of frames. A switch for separating the header and data was created in the form of a variable called 'processheader', which was set to true. Each line of the file was split into a list, and if the line contained two items, it was identified and processed as the header and otherwise it was processed as the data.

The information extracted from the header section of the text included the number of frames, the channel numbers and the channel names. The channel names were put into the 'channels' list by taking each line in the header that started with 'channel' and adding the second part of that line to that list. The 'header' map was filled by setting the first item in the line as the key and the second item as the value, so it contained every channel number with its name. The frame number was extracted by finding the line that began with 'NumFrames' and taking the second item in the line, which was a number and putting it into the string variable called 'frames'.

When the line of the text matched 'EST_Header_END', the processheader variable was set to false, so the rest of the text could be processed as the data. At the beginning of each line of data were two numbers which were not axis values, so they had to be removed. The lines, without the first two numbers, were added as lists of values to the data list. The values were divided by a thousand, so they would be appropriately scaled when the files were loaded into

Blender.

The map of headers was used to extract a list of the nine root names. They were part of the channel names, so the values of all the keys starting with 'channel' were accessed and split on the underscore, to separate them from the 'x' and 'y' parts of the string. It was added to the 'root' list if that list did not contain an identical string, to prevent the roots appearing in the list twice. There were originally ten roots, but one was not named after any speech articulator, so it had to be removed. It did not contain an underscore and so could not be split in two, so that was made a condition of it being added to the list.

The next step was to create a map of the channel names as keys and a list of their values for each frame as the map values. An iterator was used over the list of channels and for each channel number, the same numbered item was taken from each list within the data list, and added to a new list. The key was set as the name of the channel and the new list of data was set as the value.

The header and data were ready to be written to file. A string of the name of the new file was created, which was the original file name but with a .bvh extension. The lines of the new files were written within a withWriter method. First the header was set out in a bvh format by listing the name of each root, the x, y and z channels for that root and the offset positions. The header also included the total number of frames and the duration of a frame.

The data in the bvh files is also laid out by one frame per line, so it made sense to write the data to file by iterating over each frame. The number of frames was converted to an integer, then for each frame, each root was iterated over. The x and y values for each root were taken from the map of channels and data, by accessing the key with the root name and the item of data with the frame number. The z value was set as 0 for all roots and frames. A list of the x, y and z values was joined together by a tab and each time the loop moved on to a new frame, a new line was added.

The script was run and the bvh files were successfully generated. I compressed the files into a folder and shared them with the group.

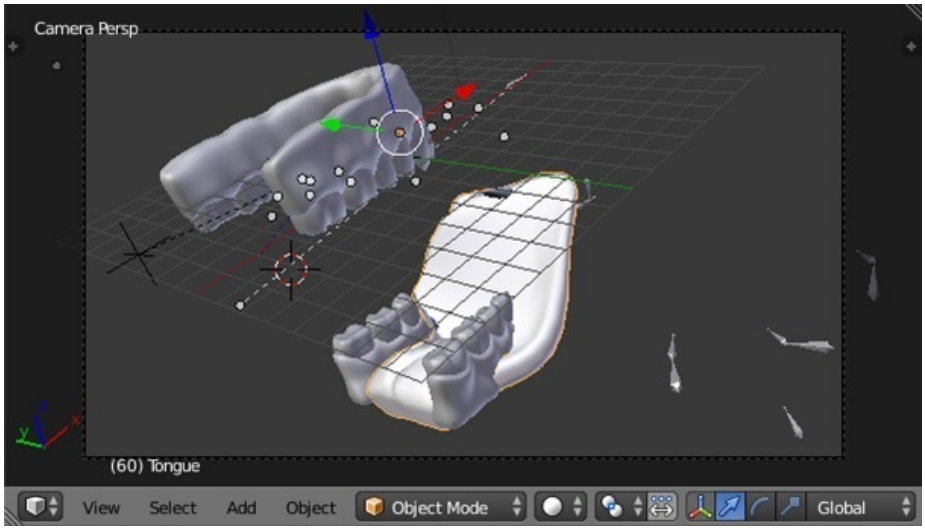
# 3 Report written by Olga Gedsun

My part of the software project was to create a demo standalone. For the realization basis there was a Blender File with an imported mngu0 collada file from the previous works. This collada file consists of a model with a set of teeth with a tongue. To make the demo standalone with the Blender animation software I had to add some game logic to the imported objects. The model should be seen from all different ankles and play the animation with the sound.

I have done the actions described further first manually with the Blender Software and then written a python script for the demo. First to add the game logic for the animation playback I have selected the tongue object, added the logic sensor, controller and actuator. Then linked up the actuator with the

controller and the controller with the sensor, this has to be made with all new added sensors to enable their functions. The actuator had to be specified by type, action, play mode, frame mode and frame start. The action starts an animation playback, which is bound to the 'ULipCoilArmature' (the coil that allows moving all the objects at once, it bound together the tongue, maxilla, mandible and all the teeth). Now that we have the game logic for one object we can copy this logic bricks to all other objects. For the sound playback it needs only one object so I add the sensor controller and actuator. The actuator needs to be specified so that the chosen sound file is played to the end.

The next step was to add a new camera and a new empty object, in my case a cube, and parent this cube to all objects. This is made to let the new camera capture the objects and their motions. I chose one object and add a new constraint 'Child Of' the empty cube and copy this constraint for all the other objects. The new camera has to be tracked to the empty cube; all other objects have to be parent to the empty cube.




In the picture above you can see the view on the object from the new camera.

Now I can add some logic to the camera, so we move the camera and see the targeted objects from different perspectives. For that we need four sets of "sensor, controller and actuator" to precise the motion to the left, to the right, up and down. To do so we set the type of the actuator to 'Motion', the mode to 'Simple Motion' and change the coordinates of location and rotation for the specified motions. The same is made for all other objects, but with different motion coordinates and another set of keyboard keys as for the camera. In the end this has to be saved and exported as a Game Engine Runtime.

# 4 Report written by Marina Oberwegner

During the semester, we were introduced to the basic idea of articulatory speech analysis. We were introduced to different methods of recording speech and tongue, as well as jaw movements, such as the x-ray method. Further, we were introduced to the study of our professor. Hence, we got an inside of how to record the tongue and set of teeth movements using the electromagnetic articulography. We also learned that the motion capture data for these movements is based on the recordings of the electromagnetic articulography. We used example capture data from EMA to test its movements in Blender. Blender is a professional free and open-source 3D computer graphics software [1]. After also being introduced and following tutorials about Blender, we were able to implement a model. Besides following the instructions during class and doing tasks at home, I had to implement our actual model, whereas the rest of the group converted some bvh files for the motion capture, as well as developing the demo.

In order to implement the model, I used a pre-defined model consisting of the set of teeth and the tongue which I received from my professor (see Figure 3). Further, I used the bvh files for implementing the movements of the parts of the model (see Figure 4).

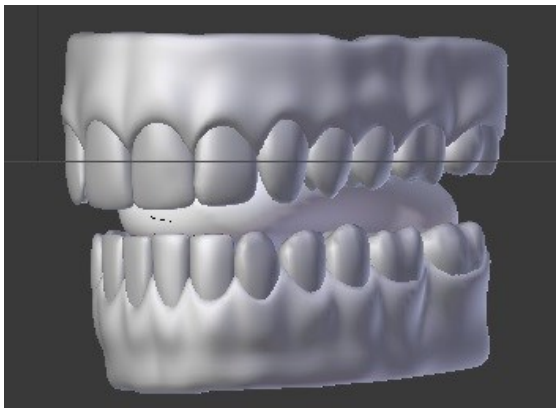


Figure 3: Model already created for implementation. It consists of the set of teeth and the tongue.

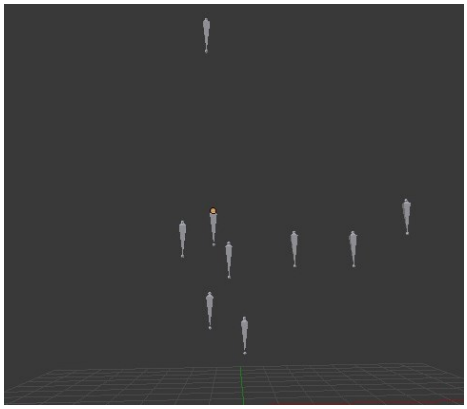


Figure 4: An example of a bvh file imported into Blender.

Before I was able to use the model together with the motion capture file, I had to create a Nurbs path to control and deform the tongue. Hence, I added a Nurbs path in ca. the length of the tongue mesh of the model (later I placed the path into the tongue mesh and connected them by using the 'Add Modifier' constraint). Further, I created three armatures which were hooked to the Nurbs path. One armature represented the tongue tip, the second one the tongue blade, and the last one the tongue dorsum. Moving one of the armatures, deformed the tongue according to the armatures movement. The armatures were also needed to be attached to the corresponding bvh parts which included the

[1] You could download it here: http://www.blender.org/download/

tongue tip, tongue blade, and tongue dorsum movement.

To put everything together, I constructed the model using Blender and later I also wrote a python script to build the model. In order to do so, I had to open my model and imported a bvh file of my choice (e.g. msak0_001.bvh).

```
# imports .bvh file into .blend project
def readBvh(bvhFilename):
    bpy.ops.object.select_all(action='DESELECT')
    bvhFilepath = "C:\\Users\\Marina\\Documents\\artimate\\src\\"
    if bvhFilename is "":
        bvhFilename = "msak0_001"
    bpy.context.scene.objects.active
    bpy.ops.import_anim.bvh(filepath=r'C:\Users\Marina\Documents\artimate\src\msak0_001.bvh', axis_forward='Z', axis_up='Y')

# load model by opening .blend file containing the model
def loadModel():
    bpy.ops.wm.open_mainfile(filepath="C:\\Users\\Marina\\Documents\\artimate\\src\\projectModel_withBvh.blend")
```

Figure 5: Open the model in Blender and import a bvh file.

The code given in Figure 5 in the method 'importBvh' throws an error, if the bvh file is not already imported into the model beforehand. The error message is shown in Figure 6. In order to get rid of the error, I imported the bvh file manually into the blend file.


Figure 6: Error when importing the bvh file automatically.

After attaching all parts of the model to an empty object (constraint 'Child Of'), I added several constraints to the mandible, maxilla, and the tongue which is done in the method called 'trackBvh'. First of all, I made sure that the bvh file was put into the right position. For this I selected the bvh object and set,

under relations, its parent to the 'Empty' object. This is shown in Figure 9.

```
#bvh
bpy.ops.object.select_all(action='DESELECT')
bvh = bpy.data.objects.get(bvhFilename)
bvh.select  = True #selects bvhInput object
bpy.data.objects[bvhFilename].parent = bpy.data.objects['Empty'] #put the imported file into correct position
bpy.ops.object.select_all(action='DESELECT')
```

Figure 7: Setting the bvh relation.

```
#armature tongue tip
armature_tt = bpy.data.objects.get("armature_tt")
armature_tt.select = True
constraint_tt = armature_tt.constraints.new('COPY_LOCATION')
bpy.data.objects['armature_tt'].constraints["Copy Location"].target = bpy.data.objects[bvhFilename]
bpy.data.objects['armature_tt'].constraints["Copy Location"].subtarget = "tt"
bpy.ops.object.select_all(action='DESELECT')
```

Figure 8: Adding armature constraint 'Copy Location'; tt stands for tongue tip

After positioning the bvh file, I added the 'Copy Location' constraint to the three different armatures (e.g. tongue tip). This could be seen in Figure 10. The 'Copy Location' constraint is responsible for copying the movement of the bvh armatures to the attached armatures. In this case, copying the movement of the tongue tip, tongue blade, and tongue dorsum of the bvh armatures to their corresponding armatures of the path inserted into the tongue.

```
#bvh
bpy.ops.object.select_all(action='DESELECT')
bvh = bpy.data.objects.get(bvhFilename)
bvh.select  = True #selects bvhInput object
bpy.data.objects[bvhFilename].parent = bpy.data.objects['Empty'] #put the imported file into correct position
bpy.ops.object.select_all(action='DESELECT')
```

Figure 9: Setting the bvh relation.

```
#armature tongue tip
armature_tt = bpy.data.objects.get("armature_tt")
armature_tt.select = True
constraint_tt = armature_tt.constraints.new('COPY_LOCATION')
bpy.data.objects['armature_tt'].constraints["Copy Location"].target = bpy.data.objects[bvhFilename]
bpy.data.objects['armature_tt'].constraints["Copy Location"].subtarget = "tt"
bpy.ops.object.select_all(action='DESELECT')
```

Figure 10: Adding armature constraint 'Copy Location'; tt stands for tongue tip

Moreover, I tracked the mandible to the jaw by using the 'Track To' constraint. It tracks the mandible to the jaw armature of the bvh in order to get the movement of the mandible according to the movement of the jaw. The python code for doing this is given in Figure 11. Finally, the rotation of the jaw was taken into account. For this I already added an armature into the model before creating the correct model. Then I added the code given in Figure 12. This piece of code selects the jaw armature I added and adds the constraint 'Locked Track' constraint to the jaw armature of the bvh file ('li'). Further, the jaw armature is selected to be the 'Child Of' the mandible. Hence, the jaw is rotating according to the 'li' movement.

The python script could either be run in Blender under the 'Scripting' screen layout, or by calling the build.gradle script by calling the task 'gradlew runBlender'. Running the either of the possibilities, opens Blender containing the successfully build model which looks like the screen given in Figure 13.

```
# track mandible to the jaw
bpy.ops.object.select_all(action='DESELECT')
mandible = bpy.data.objects.get("Mandible")
mandible.select = True
constraint_mandible = mandible.constraints.new('TRACK_TO')
bpy.data.objects['Mandible'].constraints["Track To"].target = bpy.data.objects[bvhFilename]
bpy.data.objects['Mandible'].constraints["Track To"].subtarget = "li" #jaw
bpy.data.objects['Mandible'].constraints["Track To"].track_axis = 'TRACK_X'
bpy.data.objects['Mandible'].constraints["Track To"].up_axis = "UP_Z"
```

Figure 11: Tracking the mandible to the jaw armature 'li'.

```
# bone for jaw movement
bpy.ops.object.select_all(action='DESELECT')
armature_jawRotation = bpy.data.objects.get("Armature_jawRotation")
armature_jawRotation.select = True
constraint_jawRotation = armature_jawRotation.constraints.new('LOCKED_TRACK')
bpy.data.objects['Armature_jawRotation'].constraints["Locked Track"].target = bpy.data.objects[bvhFilename]
bpy.data.objects['Armature_jawRotation'].constraints["Locked Track"].subtarget = "li"
constraint_jawRotation = armature_jawRotation.constraints.new('CHILD_OF')
bpy.data.objects['Armature_jawRotation'].constraints["Child Of"].target = bpy.data.objects["Mandible"]
bpy.ops.object.select all(action='DESELECT')
```

Figure 12: Keeping track of the jaw rotation.

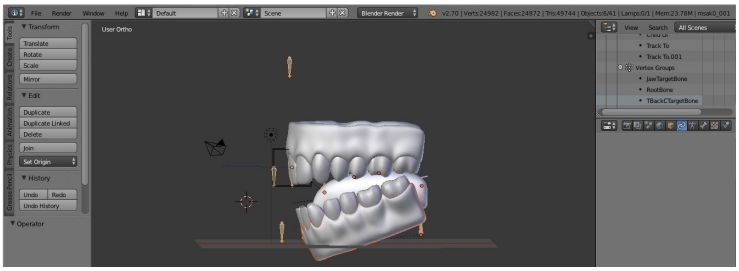


Figure 13: The successfully built model.

To sum it up, I learned how to use Blender and how to create a model using motion capture. I further got to know gradle and scripting with Blender. Moreover, I got a deeper inside into what articulatory speech analysis is about and which possibilities there are to visualize audiovisual speech information using motion capture data and a model using a set of teeth and a tongue only. The model we created is useful for getting a better inside into how the tongue moves while producing a special speech sound. It could be further developed to a model which shows people how to spell correctly, such as Prof. Olov Engwall [2] did.

[2]http://www.speech.kth.se/ olov/

# 5 Bibliography

Ingmar Steiner and Slim Ouni. Artimate: an articulatory animation framework for audiovisual speech synthesis. In John Kane and Joao Cabral, editors, *ISCA Workshop on Innovation and Applications in Speech Technology (IAST)*, pages 57–60. CNGL, Online, 3 2012.

Ingmar Steiner, Korin Richmond, and Slim Ouni. Using multimodal speech production data to evaluate articulatory animation for audiovisual speech synthesis. *CoRR*, abs/1209.4982, 2012. URL `http://arxiv.org/abs/1209.4982`.