Escola Universitària d'Enginyeria Tècnica
de Telecomunicació La Salle

Bachelor's Thesis (TFG)
Degree in Animation

Planning a Modular Rig

Student
Enrique Velasco Mairal

Supervising Professor
Ignasi Duelo Fandos

This document is an AI-assisted translation of the original thesis, written in Catalan. Technical terminology has been adapted to standard English-language rigging and VFX usage. Some wording may differ from the original.

Contents

Abstract

Planning and building a rig involves many processes that stay hidden and remain largely unknown. The goal of this work is to surface, analyze and learn all of those steps, finding the most optimal way to develop the characters for the fourth-year projects.

We will see that there are many ways to proceed, and that none is less valid than another. To know how to proceed, you have to understand which techniques exist, what each character's actions and limitations are, what its attitude will be, how much time is available and how much time there is before layout begins. Once all of these concepts are under control, a rig can be designed at the level of the character that will carry it.

All of this is explained in detail in the document that follows.

Acknowledgements

I would like to thank a number of people without whom this work would not have been possible, whether through their guidance, support or experience.

Sergio Graña, Ona Molas, Enrique Alberola, Judit Navarro and Sancho Albano took care, day after day, of every step before and after the rig so that the characters could move through the pipeline. My thanks also to my classmates and project teammates, and to the mentors and tutors.

On the other hand, to my work colleagues: Felix Balbas, Vincenzo Leombruno, Alessandro Boschian, Albert Vivó, Roure Ossó, Xavier Roca Crespo and Sara Saez Hoces, for their guidance and for helping me solve rig situations and understand the process as a whole.

My thanks too to external mentors: Iker J. de los Mozos for the experiences he shared in interviews and our long conversations; and to the Ilion mentors, Silvia Montes and Isabel Bértolo, for sharing their knowledge of how the groom and cloth-simulation pipelines work.

On the code side, I would like to thank David González Cuéllar and Vasil Shotarov for the foundations they provided in the systems for saving and reading deformer weights and for the system that saves and loads control shapes.

Acronyms

ACES — Academy Color Encoding System

API — Application Programming Interface

FK — Forward Kinematics

FPS — Frames per second

GPU — Graphics Processor Unit

IDE — Integrated Development Environment

IK — Inverse Kinematics

PyCharm — Python-based programming IDE

Python — High-level programming language

TFG — Bachelor's Thesis (Treball Final de Grau)

USD — Universal Scene Description

UV — 2D texture coordinate space; U and V define the two components

Introduction

This work focuses on the technical production of the fourth-year projects, specifically: Last in Battle, Path of Sand and World Of Lost Things.

The areas covered are: rig, groom, pipeline and tools. For each one, the process followed, its definition and the difficulties encountered are explained. The goal is not only to explain the process, but also to expand on the more theoretical parts of the fields mentioned.

The resulting deliverables are 6 rigs — 5 bipeds and one quadruped — together with scripts and utilities collected under the LS_MENU inside Maya.

A project on rigs prepared to capture mocap data, carried out jointly with Cardiff School of Art & Design, is also discussed.

All concepts described in this document refer to and are drawn from the 3D software Autodesk Maya 2018.5.

The reel will focus on the rigging side, which is the part the work is most directed towards and to which the most time has been dedicated.

General concept

Before tackling the area of rigging, we have to look at what it is, where it comes from and how it has evolved.

Definition

Rigging is the part of 3D and VFX production responsible for creating the logical deformation structures of a character. Using the program's resources, it is able to give the character the ability to move “anatomically”.

It does not need to be anatomically perfect, nor to respect every rule of the character's anatomy. What matters more is that the character has appeal. If the rules of anatomical mechanics have to be broken to achieve this, the artistic license to do so can be taken.

A character built on a correct anatomical structure will have more accurate deformations than one that is not. More variables come into play than just those belonging to the rig in order to make the character work properly — variables that are not controllable within the department, such as design, topology, groom and animation.

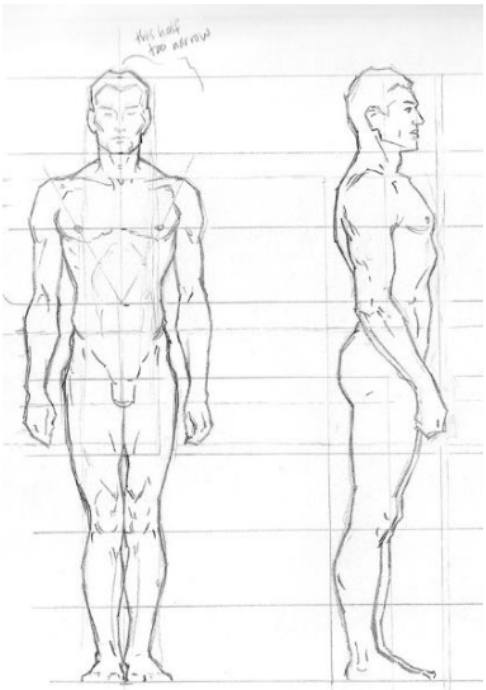

Illustration 1: Body anatomy diagram by Andrew Loomis

Objective

We can define the main objective of the rig as giving the animator controls, attributes and tools to bring expressiveness to the character.

It must be kept in mind that the geometry sculpted by the animator in the viewport will most likely not be the one that goes to render; instead, it may pass through muscle systems, cloth, hair and shot fixes.

On cartoon characters, care must be taken so that all deformations are clean and rounded, and so that wrinkles serve to mark the lines of expression.

One of the important premises is that the rig must be able to run at 25 fps. This is problematic because, however much the characters are optimized, everything depends on:

- Number of characters on screen
- Viewport display features
- Object subdivision
- Object visibility
- Active deformations
- Program evaluation (DG, Parallel, Serial)
- GPU evaluation (override)

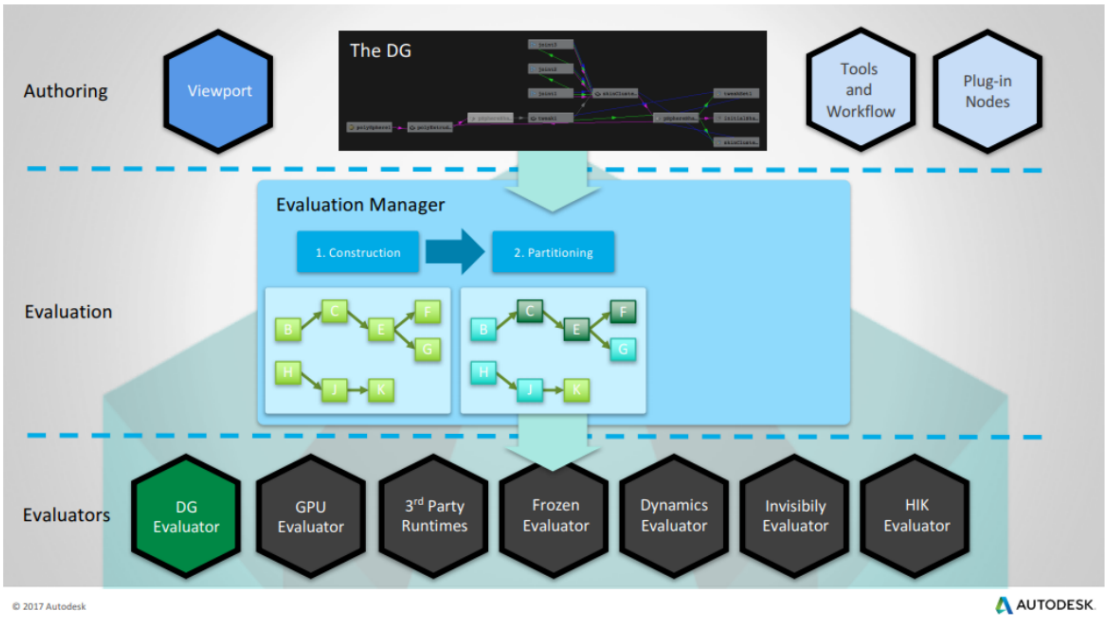

Illustration 2: Diagram of node evaluation order inside Maya

Origins

The concept of rigging took a while to emerge in production, even though it was present from the very beginning, even before 3D production began.

Rigging can be considered to start with the first stop-motion animations, when the need arose to be able to articulate characters in order to optimize production and thus avoid building each figure for each frame.

Little by little it evolved according to the needs of each production, moving from stop-motion to the articulation of robots and spaceships in the first experiments of 3D production.

Illustration 3: Armature of the King Kong animatronic built by Ray Harryhausen (1933)

Types and Techniques

By aesthetic

Focusing only on the 3D environment, two different typologies can be drawn out.

Realistic: All bodily and facial expressiveness is closer to reality. Cloth and hair are left for simulation environments. From the rigging side, systems are provided to fake muscle deformations.

Cartoon: Expressions are more extreme; systems that give the animator the ability to draw whatever shape they want with the character are applied more drastically. The concepts of squash and stretch are more present.

Illustration 4: On the left is the wireframe of Gollum from "The Lord of the Rings" developed by Weta Digital, referring to the display of a realistic geometry. On the right is Captain Charles T. Baker from "Planet 51" developed by Ilion Animation Studios, showing one of the character's maximum deformations.

By relevance

We can find:

Characters

Leads (Hero/A). Those that appear on screen the most and that carry the most narrative weight. They have the most advanced and customized deformation systems.

Secondary (B). Those that appear less frequently but interact with the leads. They have a complete set of deformations, but not as extreme as the A characters.

Tertiary. They appear in the background or to fill the scene. They tend to be generic characters with variations of clothing and hairstyles.

Props

Static. Based on pivoting systems, since they will serve for characters to grab or interact with them.

Animated. They have simple mechanical deformation systems.

Vehicles. They have systems for interacting with the terrain, in addition to advanced mechanical deformation systems.

Directed simulation. Systems in which the animator has the ability to control the generic path of each deformation but which, on top of that, have dynamic systems. Deformation or expression fields

are applied to the dynamic systems, since this makes animation and final display easier.

By camera

Finally, they can also be separated by their proximity to camera: depending on the distance from camera to character, a more complex or a simpler system will be designed.

Procedure

There are several ways to proceed when creating rigged assets. Depending on each production, the most optimal method or methods to deliver the story are chosen. They can be rigs based on systems of:

- Several geometries modeling the deformations the geometry can reach
- Heavy skinClusters with several deformation layers, where the joints hold the control
- Attributes that drive various deformations of any kind
- Limited cranks that drive deformations
- Wire deformers in which dynamic controls live
- Displacement of geometries by the movement of other, simpler ones (ShrinkWrap, WrapReversed)

Apart from those mentioned, there are others that also generate interesting deformations on characters. There is no single generic way to build a rig; it has to be adapted to the needs of each production.

Joint-based rigs

This type of rig gives great freedom of movement in the controls, since they work from the transformations of one or several controls that point to logical systems ending in the skinning joints.

Bone-based systems can be classified into two broad categories:

- Following the rig
- World-relative (Local)

World-relative systems

These types of systems are the most common in facial rigs. The logical and skinning system is computed at (0,0,0) and is connected as a blendshape layer onto the body rig.

They allow the deformation computation to be split into several blocks and to have one layer for each computation system.

Blendshapes

Shape-based systems are a type of world-relative system in which the shapes are modeled and connected. They are very useful for facial systems.

Prerequisites

Before starting any rig, you have to consider which upstream departments are involved in the creation of the asset and which are downstream. It is important to know where our data comes from and where ours will go.

Currently we can find the following upstream departments:

Modeling. Provides the input of the character's geometry. They will iterate many times on the character. On some productions they also provide the facial deformation shapes.

Texturing. Handles the textures together with the UV unwrapping.

Groom. Provides the scalp pieces. In the case where the hair is pre-animated, the simulation curves.

And as downstream departments:

Animation. Uses the rig controls to bring expressiveness to the character.

Simulation. Uses the animated scalp geometries and the curves that were extracted from the groom. It also uses the cloth animation to already have a base.

Shot Sculpting. Using the alembic file (.abc) extracted from animation, it corrects interpenetrations.

We must make sure that all the information coming in is in good condition and that what we deliver is too.

The considerations to be made are the following:

- Modeling geometry must have a smooth topology, where the loops are spread homogeneously across the surface of the model and do not drift.
- The topology must be straight: the straighter and more relaxed the topology is, the better deformations it will have.
- At the rotation points (elbows, knees, groin and shoulder) there must be a higher topology density to compensate for the loss of it during deformations.
- If the UVs are not in the first quadrant, a dedicated UV set for the rig will have to be made — even automatically — so that tracking systems such as follicles can work without errors.
- The scalp geometries should preferably have a topology similar to the geometry they are attached to.
- Groom curves should be separated by description and ordered in the outliner by groups.
- Have a group in the rig containing only the clean geometry for cache.

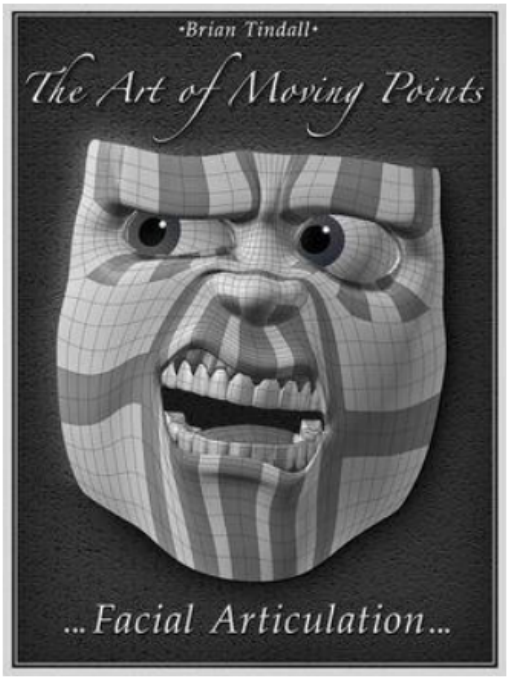

Illustration 5: Example of facial topology, The Art of Moving Points

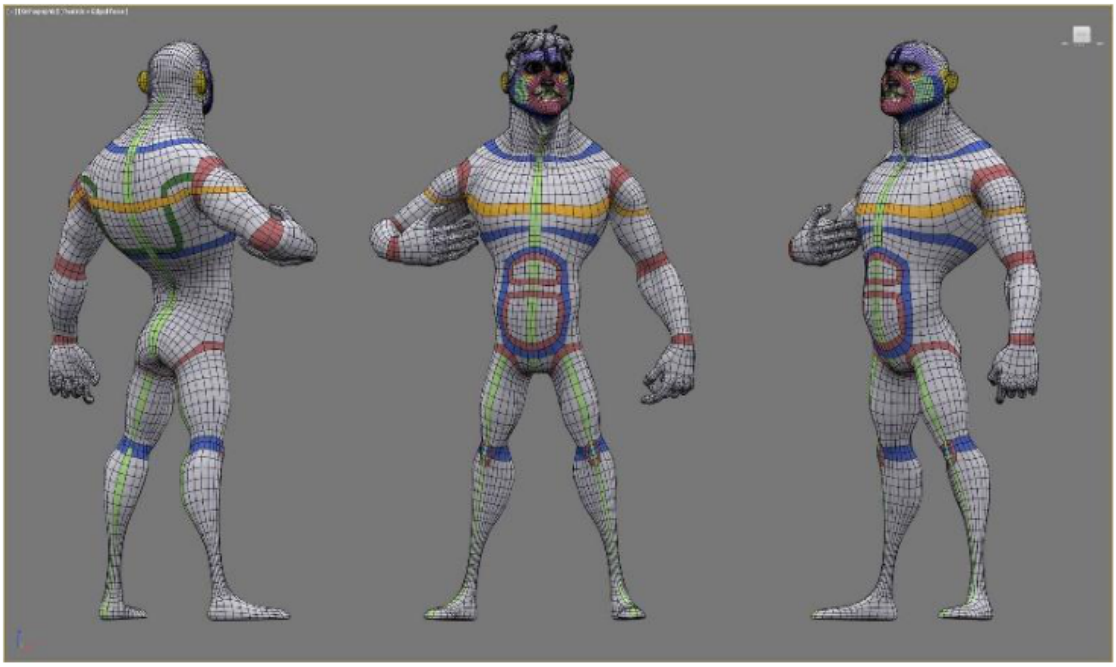

Illustration 6: Example of topology loop flow on a cartoon man, author: Héctor Sanz DOrs

Creating the rig (Cases)

There are many factors involved in building a rig. They can be split into 4 areas:

- Type of program to be used
- Type of animation
- Animator profiles in the animation department
- Available technology

The type of program to be used defines which tools can be used. Not all programs have the same tools; each one has its strengths and the things that do not quite work. Knowing the program to be used also opens up a wide range of plug-ins that we can incorporate into the pipeline.

It is important to know whether the production can tolerate the use of plug-ins, since plug-in performance often improves on the systems provided by the program itself and, in addition, noticeably simplifies the bulk of the work. Plug-ins have to be compiled for each version of the program and of the operating system, which makes their implementation difficult on productions that have different machines.

The way of animating plays a very large role when preparing the character for animation. A realistic, cartoon, semi-realistic, hard-edged or 3D stop-motion type of animation are not the same thing. There are countless ways to animate and it must be clear which one is used in each production.

The animators' level of experience is another point that must be considered. Having a team where most animators have a Junior or Mid profile is not the same as having a team in which the bulk of the animation falls on people with senior or higher profiles.

On a team where most are senior or above, more advanced logical systems of joints, wires, etc. can be used, taking the deformations to an extreme where the characters feel more alive and have fewer

limitations. This can be done because a more experienced person will know how to adapt their animation style to that of the production, making it harder to lose the character's appearance.

On the other hand, with a team of more junior profiles, the rig needs to be limited and not allow as much freedom of movement in the deformations, since the chance of losing the character's attitude is higher. The maximum deformations are therefore pre-established by the project's animation supervisor. They can be stored in a pose catalog or already built into the deformations of the controls.

Technology keeps advancing, and year after year new paradigms are implemented that revolutionize the way assets are produced. Currently, the field is moving towards parallel evaluation with threads and the computation of deformations on the GPU. There are several animation and effects studios that have been betting on this approach for years and have developed proprietary software and plug-ins. In the case of “Pixar”, they have “Presto”, a system for evaluating scenes and characters through geometry caching and the implementation of “USD” systems.

Character analysis

Before starting a rig, you have to observe how the model is built, what topology it has, what its limitations are and what is wanted in the asset.

You also have to take into account all of the extras the character will and will not have. Doing too much and investing time in systems that will not be used, or that are unnecessary, takes time away from dedicating it to the necessary systems.

Doing the analysis well is very important, since the following phases are structured from it.

In the case of the project characters, we can analyze the following for each of them:

Ledyan (Biped/LIB): The body is mainly athletic and needs a rig so that the body mechanics works correctly. In terms of facial expressiveness, it has little, so the facial can be more limited. As extras it has: the arrow quiver, the arrows, the protections, the pauldron and the bracers. The clothing will be attached on top of the body with skinning.

Yura (Biped/LIB): Essentially the same as Ledyan, and with less facial expressiveness due to the kerchief covering the face. As extras, the shoes, the waist protections, the breasts, the pauldrons and the bracers must be considered. The clothing will also be attached on top of the character.

Illustration 7: The character Yura on the left and Ledyan on the right. Both from “Last in Battle”

Ekko (Biped/Path of Sand): The same as the other two bipeds as a base. This character has more facial expressiveness, so it will need more care on the facial. As extras it has: the belt, the dagger, the belt pendant, the cloth hanging from the belt, the cape, the buff, a clothing change in shot, and a hood.

Monster (Quadruped/Path of Sand): In the shot it appears walking, so it will need a flexion system in the rear and front legs, a scapula on the front leg and a clavicle on the rear leg, a breathing system, a system to draw the tail into an appropriate shape, directed control systems on the tentacles and automatic systems on the tentacles.

Leo (Biped/WOLT): Basic body and facial rig. The clothing will be simulated on top of the animation.

Druidac (Biped/WOLT): Basic body and facial rig. The clothing will be simulated on top of the animation.

Nut (Robot/WOLT): A character with a certain relevance in the project. Since it is not a biped, it will require a custom rig in which it can move and where the arms articulate elastically. A lot of expressiveness in the eyebrows and the ability to make many different shapes. Retractable pupil.

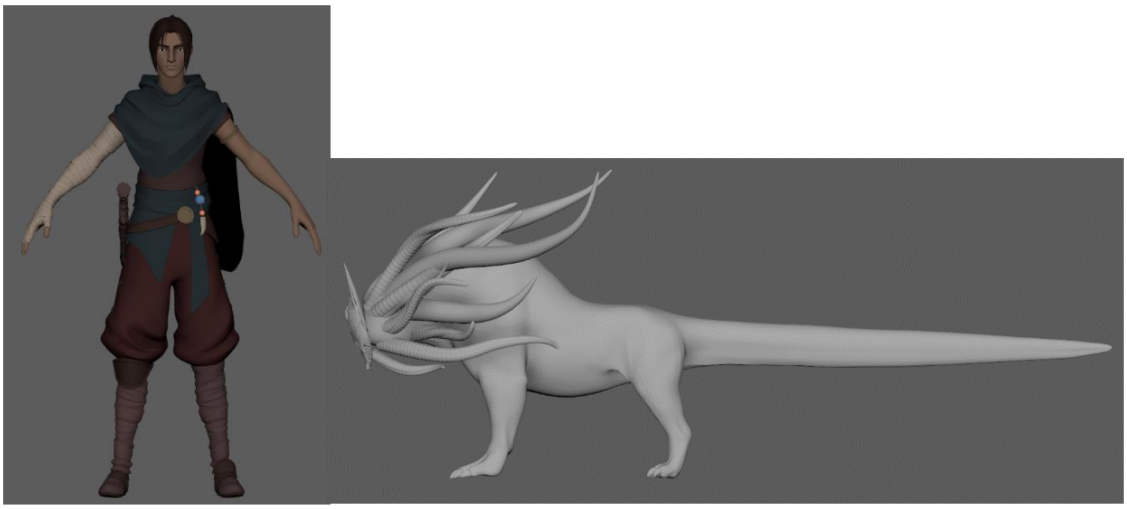

Illustration 8: Ekko on the left and the monster on the right. The scale is not representative between the two characters. Characters from "Path Of Sand"

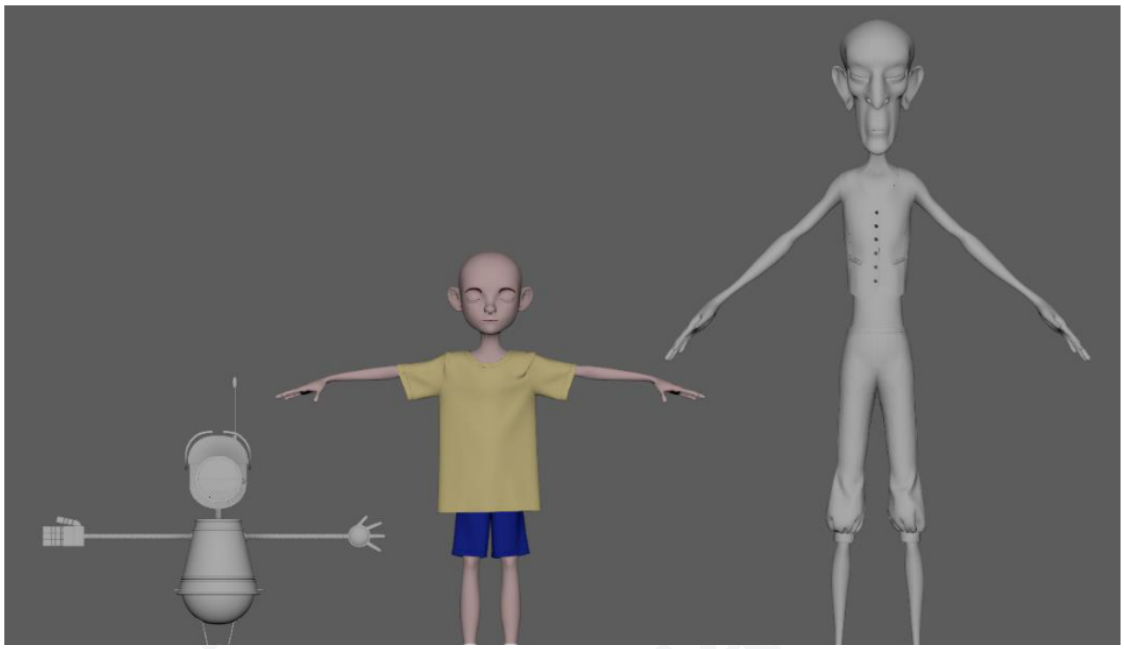

Illustration 9: From left to right, Nut, Leo and Druidac. Characters from World Of Lost Things

Approach

In order to be able to control all the version changes of model, UVs and groom, and to be able to start building the rigs in advance, before having the final models, the rig is set up to be built procedurally (Python). And as a version-control system: Git, specifically a repository on GitHub.

This way we can split the creation of the rig into the following parts:

Model. Folder with a set of .ma files containing the various versions of the model

Shapes. Folder with a set of .ma files containing the various versions of the model

Groom. Folder with a set of .ma files containing the scalps and the curves

Body guides. Folder with a set of .ma files containing joints that define the input position so the modules know where to build themselves

Facial guides. Folder with a set of .ma files containing locators and NURBS that define the position of the facial logic systems

Map weights. Sets of folders with the different weight maps saved in binary

Settings. Folder with a set of .json files with properties defined for each character

Control Shapes. Folder with the construction information of the control shapes

All of these parts are collected inside a Python file (.py) that will build a new rig each time it is executed, with whatever inputs are present. To access Maya's actions, the libraries “maya.cmds”, “OpenMaya” and “OpenMayaAnim” are used, and for file management and other utilities the libraries os, “shutil”, “json”, “cPickle”, “StringIO”, “time” and “logging” are used.

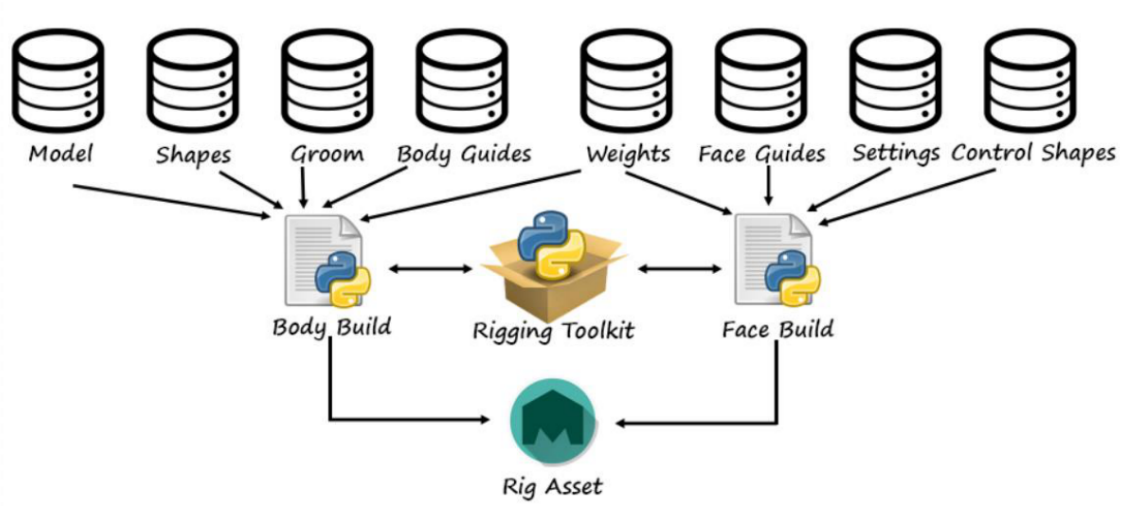

Illustration 10: Representation of the structure for generating a rig according to the established approach

Modularity and procedurality

A procedural rig is achieved when it is made not only through code, but also when the rules of object-oriented programming are applied to its structure.

It must satisfy the premises of:

Abstraction. Extracting the common needs of the objects. Separating behavior from its implementation.

Encapsulation. Information hiding. The animator does not know the internal implementation.

Inheritance. Relationship between the different rig objects. Modules that go from generalization to specialization.

Polymorphism. The same rig object behaves differently depending on the operation applied to it.

It can be established that a limb system is the same whether it is a leg or an arm. Thus, with a single limb rig object the needs of both arms and legs can be satisfied. The same happens with the other parts of the body. Only one finger system needs to be implemented, from which the others depend.

This does not only happen with humans. The concept of comparative anatomy describes that, among all living beings, there is a similar pattern in terms of structures. We can therefore reuse the modules created across several characters of the same type.

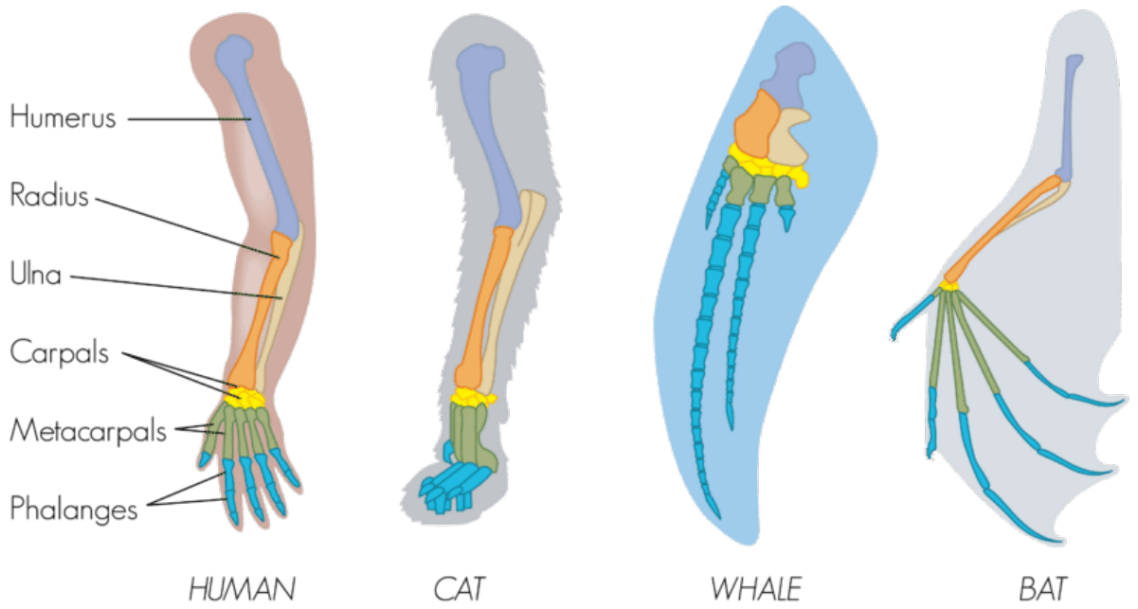

Illustration 11: Comparison of the limbs of various animals according to the theory of comparative anatomy.

Basic body systems

The biped systems will have the following characteristics:

The setup of a biped will be similar to that of the Mary Project, a character modeled and textured by José Manuel García Alvarez and rigged by Antonio Méndez Lora. It is a character everyone is familiar with and that they will have no trouble using.

For the general body rig there are IK and FK systems for the limbs. The IK system also has a stretch system, to take the deformations further.

On the limb system there is an extra control that allows the upper or lower part of the limb to be made longer or shorter, with or without maintaining volume. It also allows it to be arched.

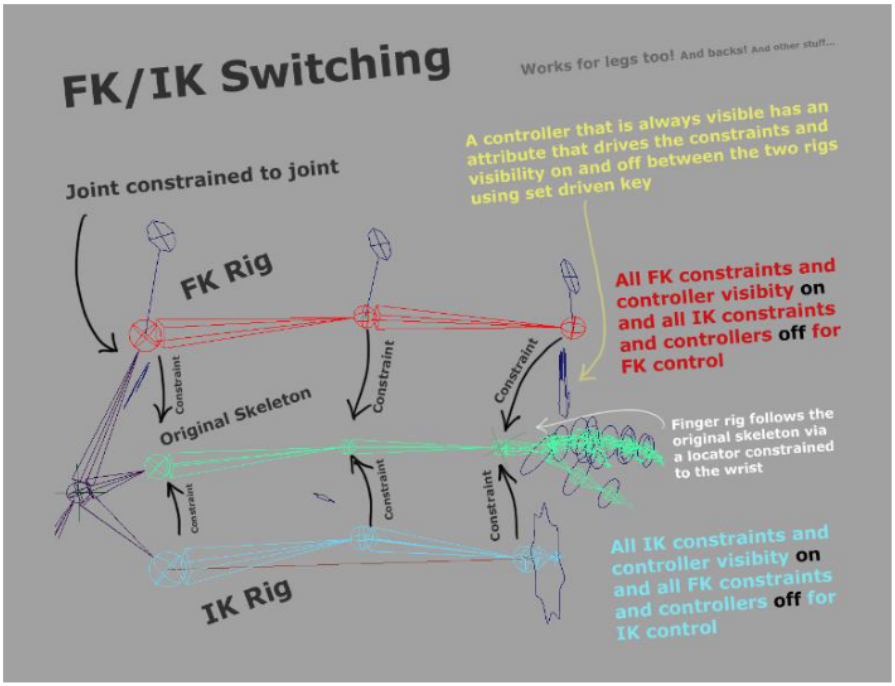

Illustration 12: IK-to-FK switching diagram established by Intro Rigging [1]

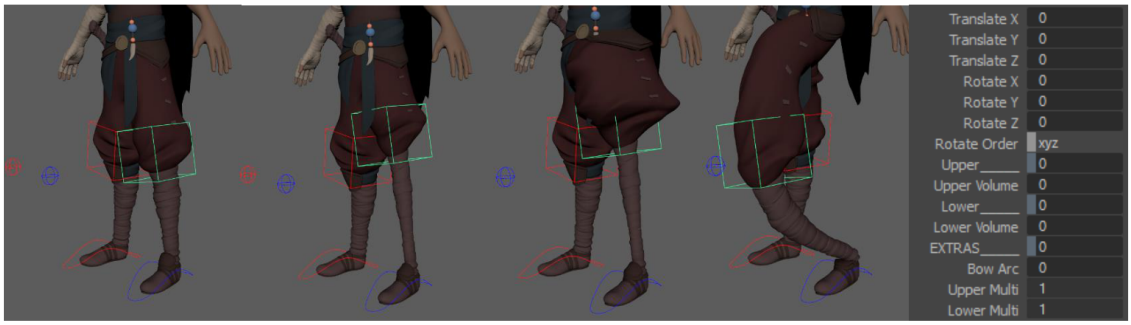

Illustration 13: Example of a limb in IK, where there is a control that moves the leg with the IK, a pole vector that defines the direction the leg looks at, and a control that allows an offset to be given to the knee.

For the spine there is an IK and FK system, but this one is different because the FK controllers are positioned after the IK, as can be seen in the following image. It also has squash and stretch parameters with minimum and maximum values, to give more dynamism to the animation.

The clavicle system carries the arms. For the arms in FK, there is an option for the arms to follow the rotations of the clavicle or not, giving this system another effect.

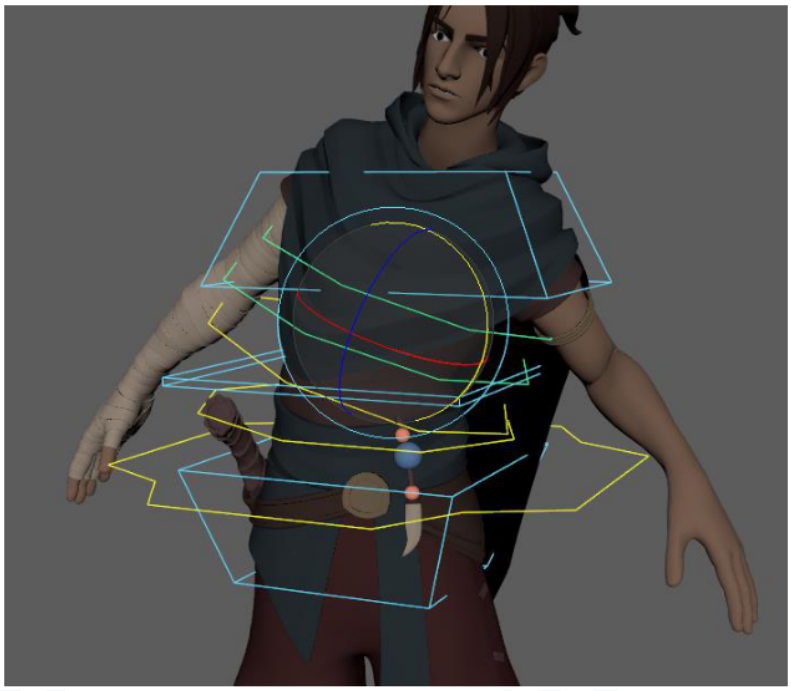

Illustration 14: Example of the spine in the live IK/FK system

Illustration 15: Example of how the clavicle works with the arm-follow system in FK

The fingers of the hands have control systems, both in IK and FK. When in FK, they have spread, fist and twist attributes.

For the foot in IK there is a reverse-foot IK system that gives the animator the following options: Ankle Swivel, Ball Swivel, Toe Swivel, Ankle ZRoll, Ankle XRoll, Toe Tilt, Foot Bank, Ball Lock Swivel, Ball Lock ZRoll, Foot Roll, Toe Tap. All of these systems are formed by direct connections to offset groups on top of the leg IK.

The head system has an FK control from the base of the neck and an IK control from the tip. It also has an option for the head not to follow the rotations of the chest.

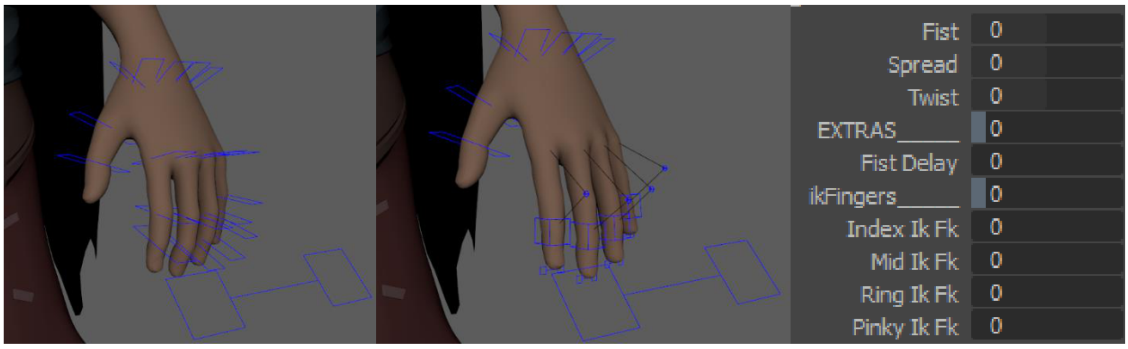

Illustration 16: Hand finger control system, with the attributes present in the generic control. In the center, the IK system on the fingers can be seen.

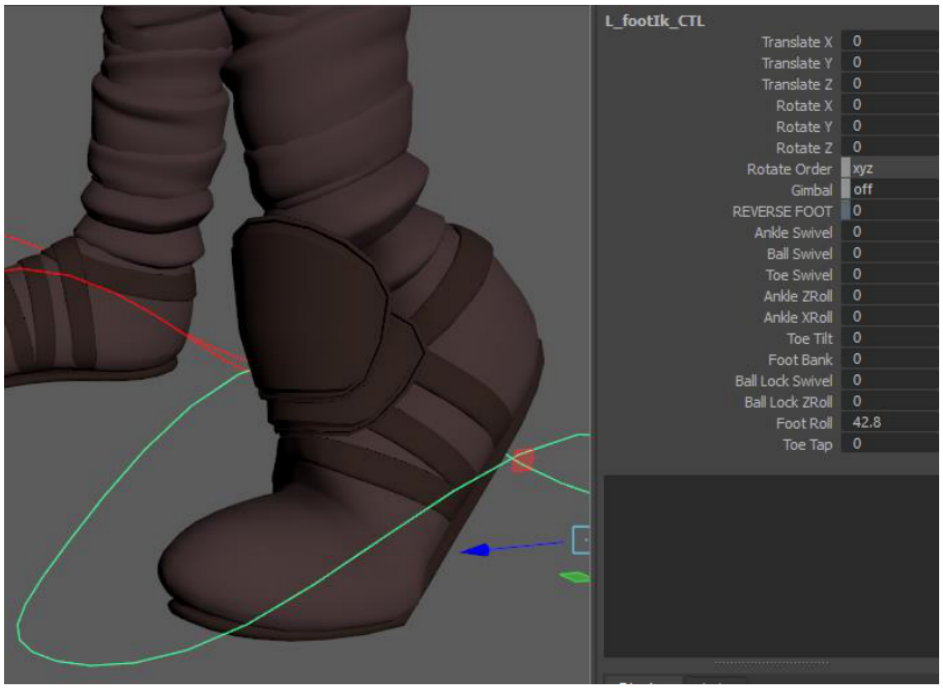

Illustration 17: Reverse-foot system; on the left the foot roll is placed, on the right the control's specific attributes are shown.

On the eyes there is a system to control their aim, separated into right and left and a global one that carries both. They also have control over the dimensions of the pupils and the iris.

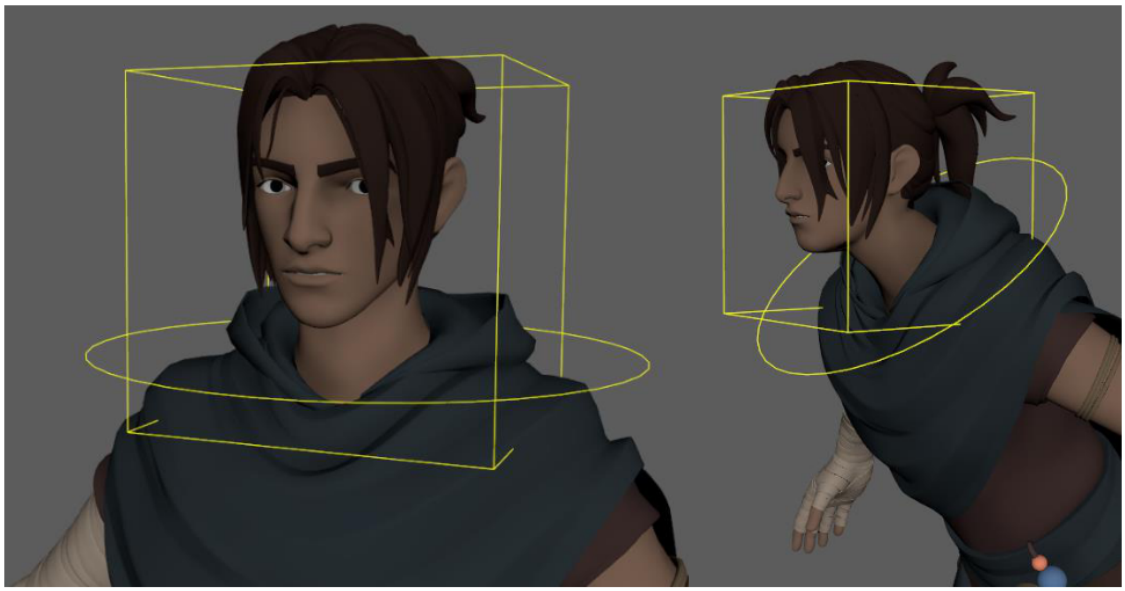

Illustration 18: Head and neck control system. On the right, you can see how the head's no-follow system to the chest rotations is acting.

Illustration 19: Eye control system. You can see what control there is with the iris and pupil systems.

Basic facial systems

The facial rig is world-relative, so it is connected to the body deformations through a blendshape. All facial controllers are mirrored.

The order in which the layers are presented is not the order in which they are implemented in the rig; they are merely logical groupings of ideas.

Layer 1

Bone system for the basic face: lower part of the jaw, upper part of the jaw and nose. The upper jaw never goes below the lower jaw, which makes the mouth close and push with collision.

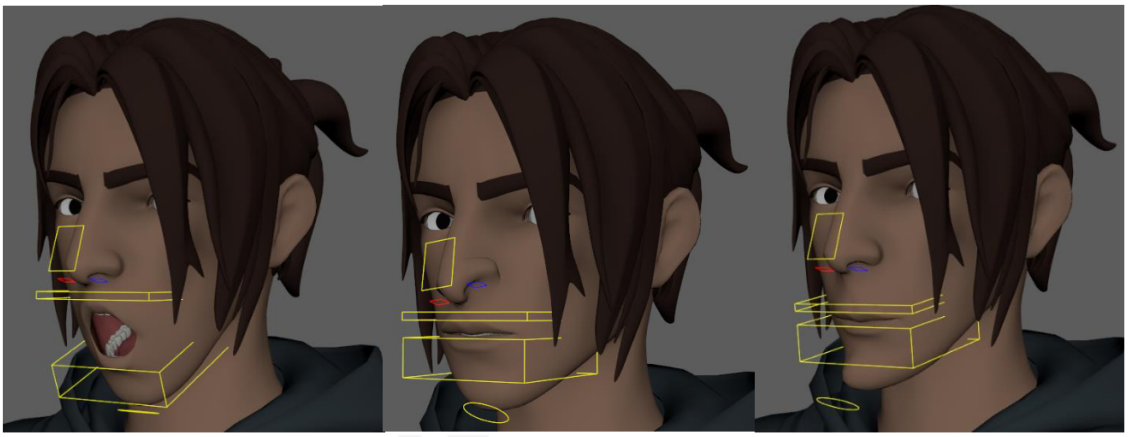

Illustration 20: Jaw and nose control system

Layer 2

Bone system for the upper part of the jaw, the eye sockets and ears. To be able to move them as a whole and position them.

Illustration 21: Control system for eyes and ears.

Layer 3

Corrective shape for the opening of the jaw to help achieve cheek compression when opening the mouth and getting the O shape. It will be made with a joint and will be scaled.

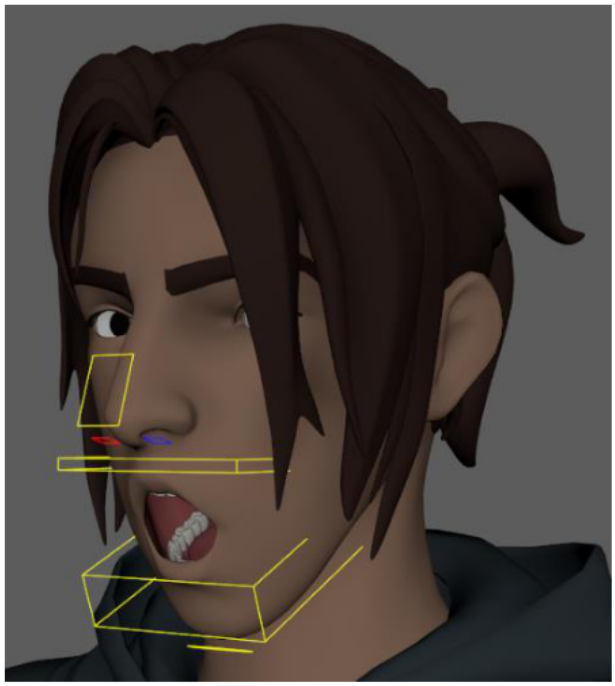

Illustration 22: Lip compression system when opening the mouth

Layer 4

Joint system for the eyelids, rivet and blendshape between the upper and lower eyelid edges. The blink height can be defined, or it can be done separately.

Illustration 23: Eyelid control system. The attributes can be seen.

Layer 5

Fine shape-tweak system. Tweak controls for the lip area, nasolabial fold and eyebrows.

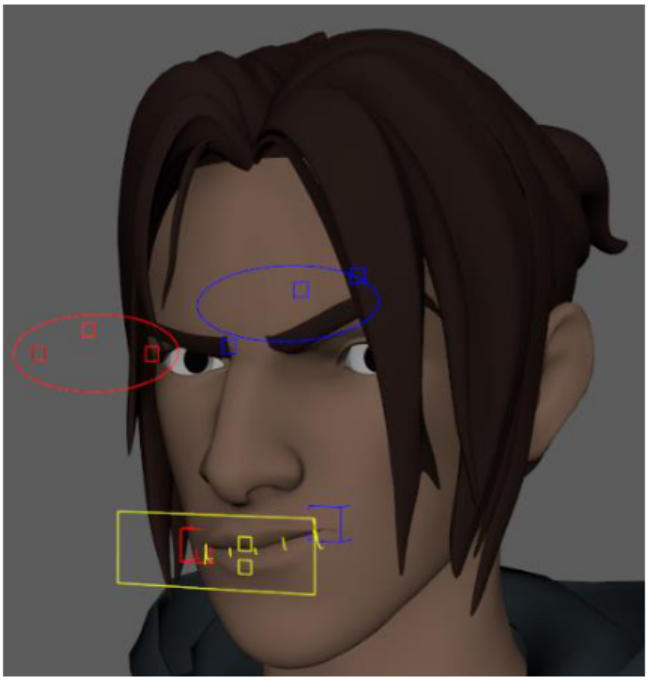

Illustration 24: Eyebrow and lip tweak system

Layer 6

Blendshape system (the basic and symmetric shape models are by Enrique Alberola; my role was to design the shapes, how to connect them, and small tweaks and passes on the shapes):

Mouth

It will have a system of 8 blendshapes; the modeled forms are corner stretched, corner squashed, corner raised, corner lowered, and their combinations. When modeling the mouth shapes, it must be kept in mind that the lips have to slide over the jaw that forms the teeth area. It is important that the upper and lower lips do not separate.

Corner stretched. It is important that the deformation is parallel to the base lips, since combined with the others it will generate other shapes. Mainly the lip corners move and carry a bit of the middle along. The nasolabial area sinks horizontally and very gently stretches the nostril area.

Corner squashed. It is the inverse of corner stretched; it is about making the lip corners almost touch. The central part of the lips compresses (it must slide over the surface of the jaw).

Corner raised. The lip corners lift, slightly accompanying the nose and the nasolabial area. The cheek slightly accompanies the deformation; it must reach the U shape.

Corner lowered. The opposite of the previous one; it has to lower the lip corners and stretch the loops of the nasolabial area. The cheek stretches a little.

Illustration 25: Shapes from left to right: corner stretched, corner squashed, corner raised and corner lowered

The union of the 4 previous deformations is interpolated through a controller that moves in y and x translation. Even though the deformations are interpolated, the resulting shapes at the extremes [(10,10), (-10,10), (-10,-10), (10,-10)] do not quite have an appropriate deformation regarding the cheeks and the harmony between them. To fix the resulting deformations, corrective shapes that force the shape we determine have to be generated. And their combinations: to generate the shapes, you have to start from the union of:

- Stretched + Up
- Stretched + Down
- Squashed + Up
- Squashed + Down

The goal of the correctives is to mark the cheek area. They are key shapes for marking the character's attitude.

Illustration 26: Corrective shapes, from left to right: stretched up, stretched down, squashed down and squashed up

Cheek

It has 4 shapes (cheek puffed, cheek sucked, cheek raised and cheek slid)

Cheek puffed. Spherical shapes work well. To make it, it helps to place a sphere as a guide. It also tends to work to pull the shape out with a “softMod” deformer.

Cheek sucked. It must be kept in mind that the zygomatic bone is below the eyes, so the cheek has to be placed a bit lower.

Cheek raised. It has to make the cheek compress with the eye.

Cheek slid (4). It is about the sliding, in y and x translation, of the cheek over the face.

Illustration 27: Cheek shapes, from left to right and top to bottom: puffed, sucked, slid up, slid down, slid forward and slid backward.

Lip shapes

(Positive roll, negative roll, kiss) They serve to make the phonetic deformations and to give intention to lip movements. They will be driven by attributes limited from 0 to 10.

Positive roll. It is about the lips rolling outward, both lower and upper; part of the teeth must be visible.

Negative roll. Like the previous one, but in this case inward. It serves to tighten the lips and make bilabial phonemes (p, b).

Kiss. Similar to a narrow mouth, this one acts as a kiss, that is: besides tightening the corners and the lips, it pulls more volume out of the lips and compresses the cheeks.

Illustration 28: Specific lip shapes, from left to right: positive roll, negative roll, kiss

Eyebrows

There are 8 shapes (raised, lowered, together, apart, rotated positive, rotated negative, forward and back). It is important that they slide over the fictitious surface of the character's skull.

Raised. The eyebrows have to rise vertically and symmetrically until they match the topology. The eyebrows have to be left slightly curved.

Lowered. The eyebrows lower to the height of the middle of the eyes; care must be taken that they do not collapse with the eyelashes, in case they are long. The glabella (between the brows) must not be marked.

Glabella marking. The glabella must be marked, making the eyebrow collapse.

Apart. The eyebrows have to slide over the skull outward.

Together. The eyebrows have to slide over the skull inward.

Rotated positive. They have to make a positive sine.

Rotated negative. They have to make a negative sine.

Forward. The eyebrows have to come outward; it serves to correct other deformations.

Back. The eyebrows have to go inward; it serves to correct other deformations.

Illustration 29: Eyebrow shapes, from left to right and top to bottom: raised, lowered, forward, back, apart, together, glabella marking, rotated positive, rotated negative.

Nose

It has 3 shapes (sneer, flare, sniff)

Sneer. Raising of the nostrils

Flare. Flaring of the nostrils

Sniff. Inhalation of the nostrils

Illustration 30: Nose shapes, from left to right: sneer, flare, sniff

Execution

Body

The first step is to set up the body guides. They are a set of joints, locators and NURBS that establish the position and orientation information so that the various modules (classes) have the data to operate on.

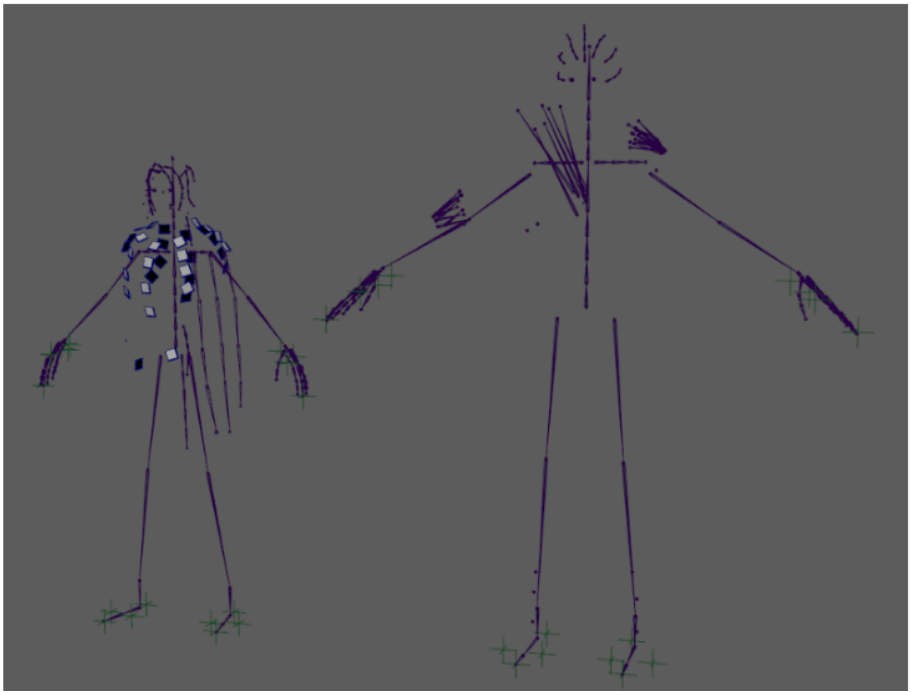

Illustration 31: Example of body guides. Ekko's on the left and Ledyan's on the right

This data is the basis for the creation of all body systems, both generic and specific.

The second step is to design the structure of the Python file that will be responsible for creating all the systems using the designed “riggingToolkit”. This is an example of the start of the build, where the model and the guides are loaded, the basic outliner structure is created, and the spine module is created.

The third step is to define the spaces each control, or set of controls, will have. There are dynamic spaces, such as the arm or the head, and follow spaces, such as the hand or the foot.

```
def build(character_name, body=False, falece=False, publish=False):
    # create new file
    mc.file(new=True, f=True)
    for plugin in ['matrixNodes.mll', 'MayaMuscle.mll']:
        if not mc.pluginInfo(plugin, q=True, l=True):
            mc.loadPlugin(plugin)

    # create base rig modules
    arm = asset.Asset(character_name)
    char = character.Character(character_name=character_name, asset=arm, extra_scale=20)

    # load models and guides (body)
    char.load_model()
    char.load_guides(body=body)

    # create base structure
    char.prepare(body=body)
    # place geometry group inside the geo transform
    mc.parent(character_extras.character_geometries(character_name=character_name), char.geometry_grp)

    ##############################
    # create spine
    first_joint = 'C_spineRoot_JNT'
    childs = mc.listRelatives(first_joint, allDescendents=True)
    childs.reverse()
    spine = spineCmds.Spine(name='spine',
                            start_joint=first_joint,
                            mid_joints=childs[:-1],
                            end_joint=childs[-1],
                            character=char,
                            hook_to=char.masterWalk.transform)
    spine.create()
```

Illustration 32: Example of the start of the asset build file

```
    ###################################
    ###################################
    ###################################
    # Define space switches
    for index, arm in enumerate(arms):
        spaceSwitchesCmds.space_switches(node=arm.pv_control.transform,
                                         space_dict={
                                             'hand': arm.ik_control.transform,
                                             "clavice": clavices[index].end_joint,
                                             "world": char.masterWalk.transform,
                                             "pelvis": spine.start_joint
                                         },
                                         default='clavice'
                                         )
        spaceSwitchesCmds.space_switches(node=arm.ik_control.transform,
                                         space_dict={"clavice": clavices[index].end_joint,
                                                     "world": char.masterWalk.transform,
                                                     "pelvis": spine.start_joint,
                                                     "chest": spine.end_joint,
                                                     "head": head.start_joint,
                                                     },
                                         default="world"
                                         )
        spaceSwitchesCmds.orient_switches(node=arm.fk_start_control.transform,
                                          space_dict={"clavice": clavices[index].end_joint,
                                                      "world": char.masterWalk.transform
                                                      },
                                          default="clavice"
```

Illustration 33: Example of code defining the controllers' follow spaces

The third step is to run the build file and adjust the dimensions of the controllers that come out by default. Going from the controllers on the left to those on the right.

The fourth step becomes the skinning of the geometries, which is when the area of influence of each bone is defined. To do this, proxy meshes and tools such as “ngSkinTools” and “brSmoothWeights” are used. To speed up reading and saving the maps, they are stored in binary files created with

“cPickle”.

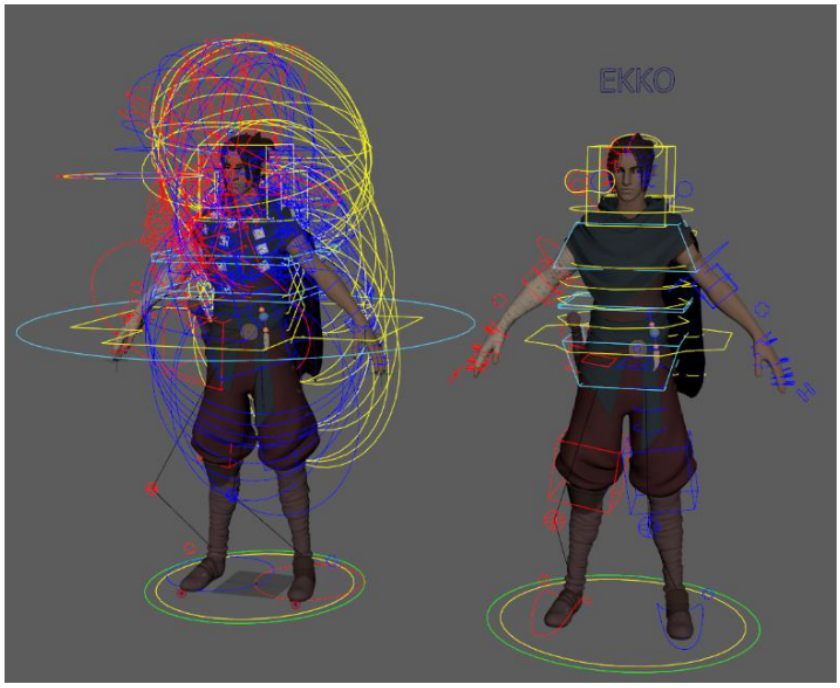

Illustration 34: Example of the character's guides; on the left the default guides, on the right the placed guides

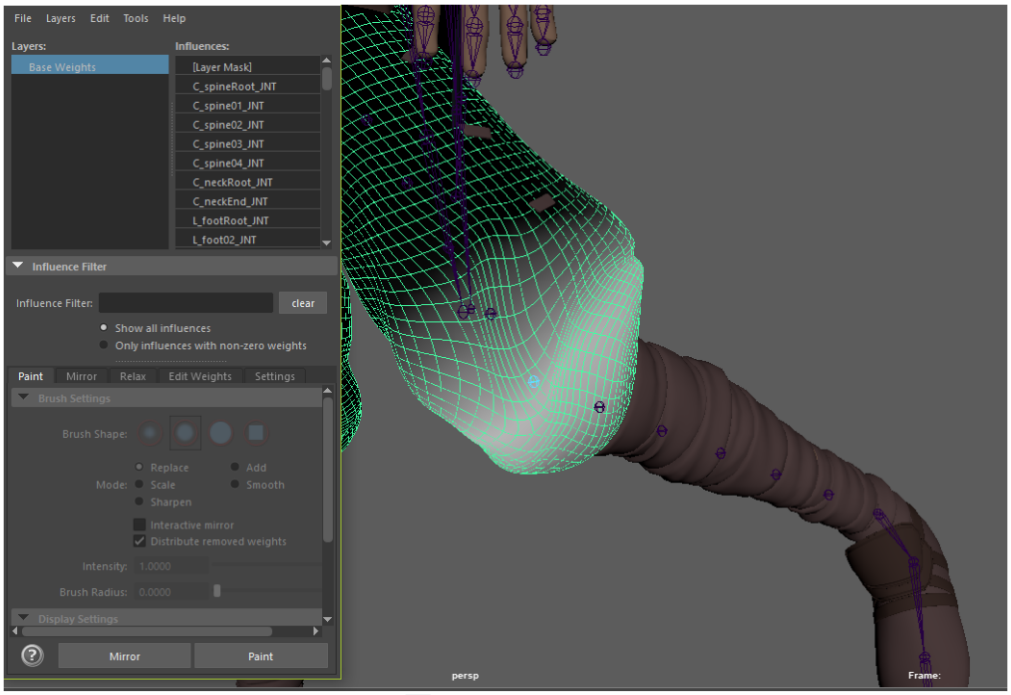

Illustration 35: Example of painting weights with ngSkinTools, where there is one weight layer.

The fifth step is to load the settings and weights that have been established. For example, those of fist, spread and twist on the hand fingers.

The sixth step, to finish building the body, is to lock and place certain attributes to the animation supervisor's measure. This makes it harder for the animator to break the rig by accessing options that are not visible at first.

```
# set freeze joint for skinning
freeze_jnt = mc.createNode('joint', n='C_bodyFreeze_JNT')
mc.parent(freeze_jnt, char.body_skel)
char.skinning_objects.append(freeze_jnt)

# label joints
joints.label_joints()

# load saved curves
curves.load_all_curves(char)

# load settings
settings.load_all_settings(char)

# load skinCluster.
# load last available skin
skinCluster.load_david_cuellar(character=char)
```

Illustration 36: Example of loading control shapes and weight maps

```
if publish:
    # lock everything for the animator
    [mc.setAttr(o + '.ihi', 0) for o in mc.ls('*Shape*', '*_REV*', '*_MDN*', '*Constraint*', '*_SKN*', 'bindPose*',
                                              '*IF*', '*MDL*', '*_PMA*', '*RMV*')]

    mc.hide(mc.ls(type='ikHandle'))
    mc.hide(mc.ls('*_LOC'))

    # set geometry grp to reference mode
    mc.setAttr(char.geometry_grp + '.overrideEnabled', 1)
    mc.setAttr(char.geometry_grp + '.overrideDisplayType', 2)

    # set skeleton grp to reference mode
    mc.setAttr(char.body_skel + '.overrideEnabled', 1)
    mc.setAttr(char.body_skel + '.overrideDisplayType', 2)
    mc.hide(char.body_skel)

    mc.hide(char.body_rig)

    mc.select(d=True)

    mc.setAttr('L_armExtra_CTL.IkFk', 1)
    mc.setAttr('R_armExtra_CTL.IkFk', 1)
    mc.setAttr('L_legExtra_CTL.IkFk', 0)
    mc.setAttr('R_legExtra_CTL.IkFk', 0)

    mc.setAttr('L_legExtra_CTL.stretch', 1)
    mc.setAttr('R_legExtra_CTL.stretch', 1)
    mc.setAttr('L_armExtra_CTL.stretch', 1)
    mc.setAttr('R_armExtra_CTL.stretch', 1)
    if mc.objExists('working'):
        mc.hide('working')

    # cleanup things
    cleanup.delete_unknown_nodes()
    try:
        cleanup.removed_unsused_influences([char.geometry_grp])
    except:
        mc.warning('Cannot remove influences')
    # cleanup.delete_unused_nodes()

    # # add character name
    curve_name = textToCurves.text_to_curve(text=character_name.upper(), curve_name='C_characterName_CRV', size=100)
    mtx_bb = mc.exactWorldBoundingBox(char.geometry_grp, calculateExactly=True)
    mc.xform(curve_name, t=(0, mtx_bb[-2] * 1.1, 0), ws=True)
    mc.makeIdentity(curve_name, apply=True, t=True)
    mc.parent(mc.listRelatives(curve_name, ad=True, s=True), char.global_ctl.transform, r=True, s=True)
    mc.delete(curve_name)
```

Illustration 37: Example of preparing the rig for animation

Facial

Building the facial rig is similar to building the body one.

The first step is also to define the guides for where the systems will be positioned. The locators mark positions and orientations, and the surfaces mark projection spaces for the facial systems. The upper one marks the range of movement of the eyebrows and the lower one does the same with the lips. This way, the character's shapes are better preserved.

The second step is to generate the facial build file. The difference from the body one is that the body rig is loaded and deformation layers are created.

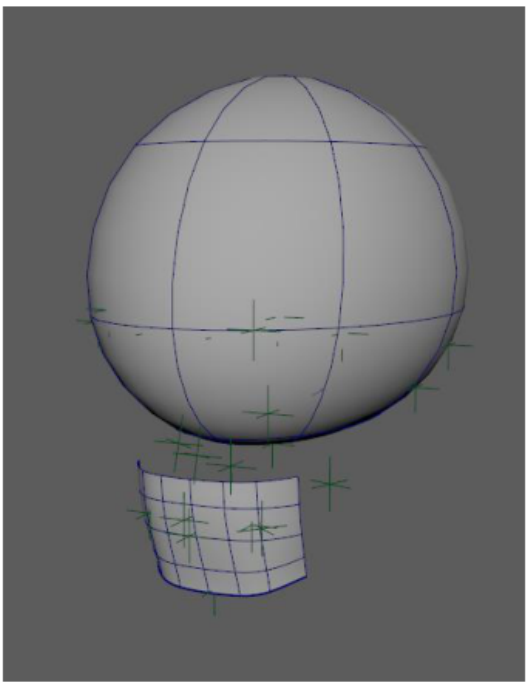

Illustration 38: Example of facial guides, Ekko

```
def build(character_name, body=False, face=False, publish=False):
    # create new file
    mc.file(new=True, f=True)
    for plugin in ['matrixNodes.mll', 'MayaMuscle.mll']:
        if not mc.pluginInfo(plugin, q=True, l=True):
            mc.loadPlugin(plugin)

    # create base rig modules
    a = asset.Asset(character_name)
    char = character.Character(character_name=character_name, asset=a, extra_scale=20, face=True)

    # load models and guides (body)
    char.load_body_rig()
    char.load_guides(face=True)

    # create base structure
    char.prepare(body=body, face=True)
    main_blendshape = blendShape.BlendShapeCommand(name='faceDeformation',
                                                   base=character_extras.character_body_geo(character_name)
                                                   )
    tune_blendshape = blendShape.BlendShapeCommand(name='tune',
                                                   base=character_extras.character_face_geo(character_name)
                                                   )
    tune_mesh = tune_blendshape.duplicate_and_add(name='C_tune_GEO')
    main_blendshape.add_shape(tune_mesh.split('.')[1])
    mc.parent(tune_mesh.split('.')[1], char.face_geo)
    sticky_blendshape = blendShape.BlendShapeCommand(name='sticky', base=tune_mesh.split('.')[1])
    sticky_mesh = sticky_blendshape.duplicate_and_add(name='C_sticky_GEO')
    local_blendshape = blendShape.BlendShapeCommand(name='local', base=sticky_mesh.split('.')[1])
    local_mesh = local_blendshape.duplicate_and_add(name='C_local_GEO')
    jaw_layer = char.face_layer(geometry=local_mesh.split('.')[1], name='jawNoseEars')
    sliding_layer = char.face_layer(geometry=local_mesh.split('.')[1], name='sliding')
    face_shapes_layer = char.face_layer(geometry=local_mesh.split('.')[1], name='faceShapes')
    eyelids_layer = char.face_layer(geometry=local_mesh.split('.')[1], name='eyelids')
    move_layer = char.face_layer(geometry=local_mesh.split('.')[1], name='move')

    char.load_blendshapes()
    face_shapes_blendshape = blendShape.BlendShapeCommand(name='faceShapes',
                                                          base=face_shapes_layer.split('.')[1]
                                                          )
    character_extras.wrap_stuff(character_name=character_name)

    # extra layers
    character_extras.extra_layers(character_name=character_name, char=char)
```

Illustration 39: Example of the creation of the deformation layers for the facial

As the third step, the shapes, weights and deformation settings are loaded, the same as in the body system.

Checks

Even though the rig is made procedurally, you have to check each time whether the deformations of the new build work with the old animations. It is therefore important to have a system such as “Studio Library” or “Atom” exports and imports, with saved animations, to verify that the new version is compatible with the previous ones.

Extras

Apart from the generic systems for the characters, specific systems have been created for each of them.

Ledyan

FK control systems have been created for the pauldron, for the pauldron's leaves, for the quiver, for its arrows and for the hair.

The shin guards and the straps have a system of a joint resting on a NURBS surface that has the same weights as the geometry underneath.

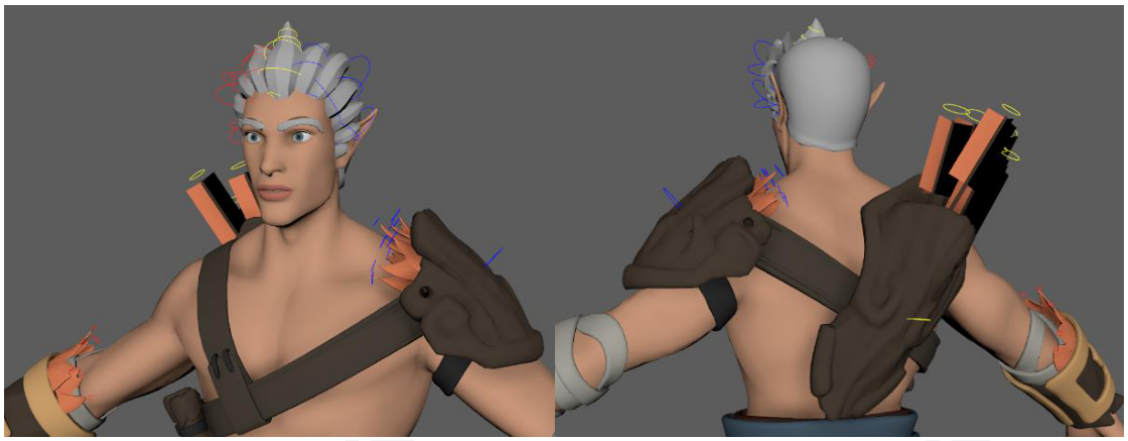

Illustration 40: Example of the FK systems on Ledyan

Illustration 41: Example of joint-on-a-NURBS systems.

Yura

It has FK systems for the pauldrons, the breasts and the hair locks. For the bun it has a system similar to the spine: an IK with squash and stretch to give it secondary movement.

Illustration 42: FK and spline systems for Yura

Ekko

All of the things hanging on this character — belt amulet, belt cloth, sword, hair — are FK chains attached to various parts of the body.

The hood is made with the same IK system with tangencies and squash as Yura's bun.

For the buff, the buckle and the point where the dagger attaches, there is a system of a NURBS skinned with the same deformations as the geometry underneath, which moves, on a second layer, the geometry it controls.

Ekko has the ability to put the hood on and take it off. The hood-on state has a set of FK systems so that its dynamic behavior can be animated without having to simulate it. Some controls follow various body bones (neck (4), head (2)).

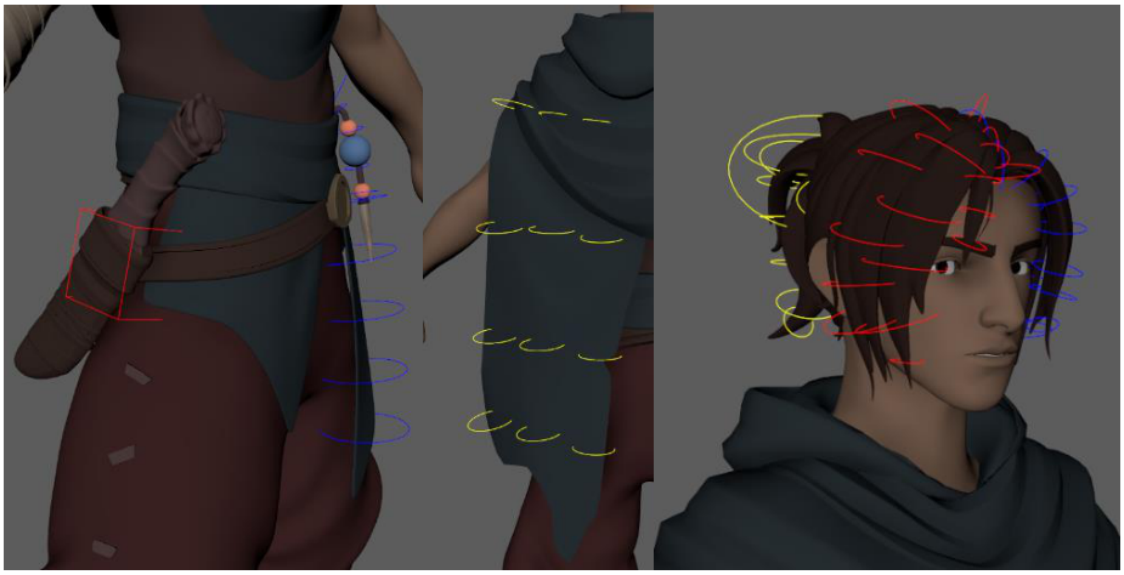

Illustration 43: FK systems on Ekko

Illustration 44: IK system of Ekko's hood

Illustration 45: Joint-on-a-NURBS systems on Ekko

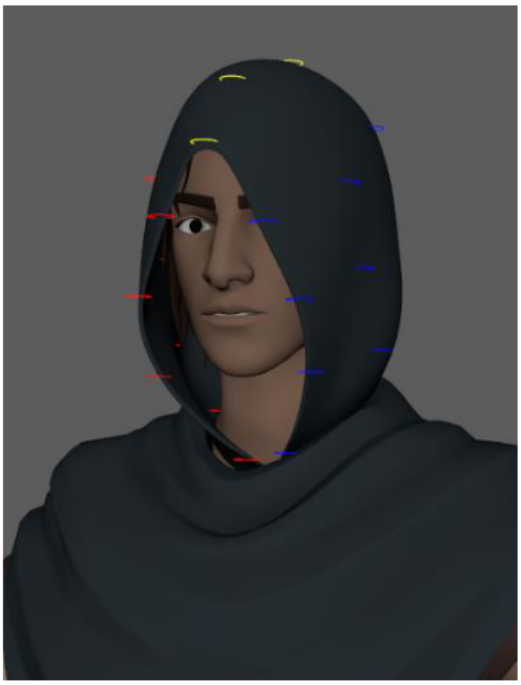

Illustration 46: FK systems for Ekko's hood-on state

Monster

The “Path of Sand” monster is not a biped but a quadruped, and it needs a different deformation system.

For the legs it has an IK system that allows the leg to articulate and another IK system that allows the heel to be rotated forward and backward. The rear clavicle system is a normal biped clavicle system. But for the front clavicle it has a reverse clavicle system. This allows the quadruped's scapula to be pulled out.

For the spine it has an IK system with a tangency control for the chest and one for the waist. The spine has the ability to lengthen by the chosen percentage. There is also a set of 9 controls to move the monster's belly and ribs.

The monster's tail has an IK system and an FK one that lives on top of the IK. It also has the ability to stretch and contract. The IK controllers are parented to one another so as to better define the shape of the path the tail will follow.

The monster has a set of 21 tentacles on its head; animating them entirely by hand is quite laborious. They therefore have their own control system. By default, there are controls to move each tentacle individually in its entirety. The controls that define the main trajectory of the tentacle can be extracted. Under the IK trajectory system there are two dynamic curves: one has a low-frequency turbulence field applied that gives it the general shape, and the other a high-frequency turbulence field that gives the tentacle its micro detail. For node evaluation to work, it must be set to Dependency Graph.

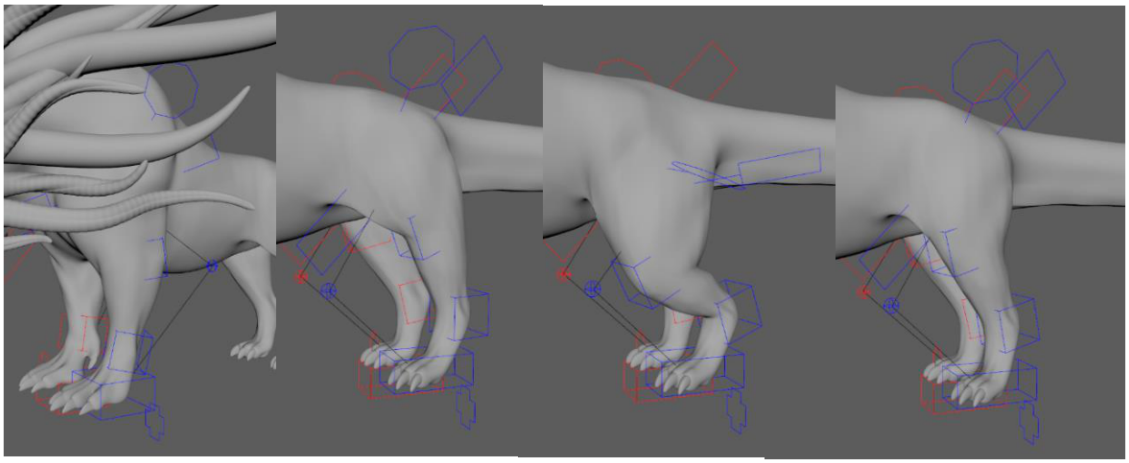

Illustration 47: IK system of the Monster's legs

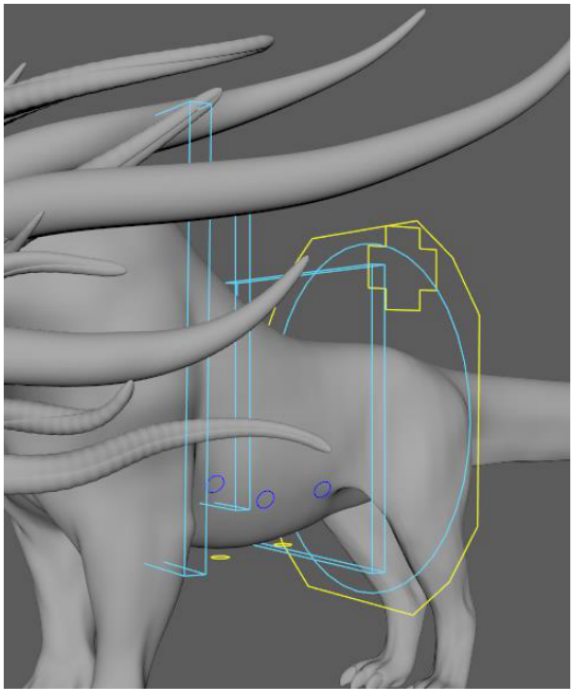

Illustration 48: The monster's spine system

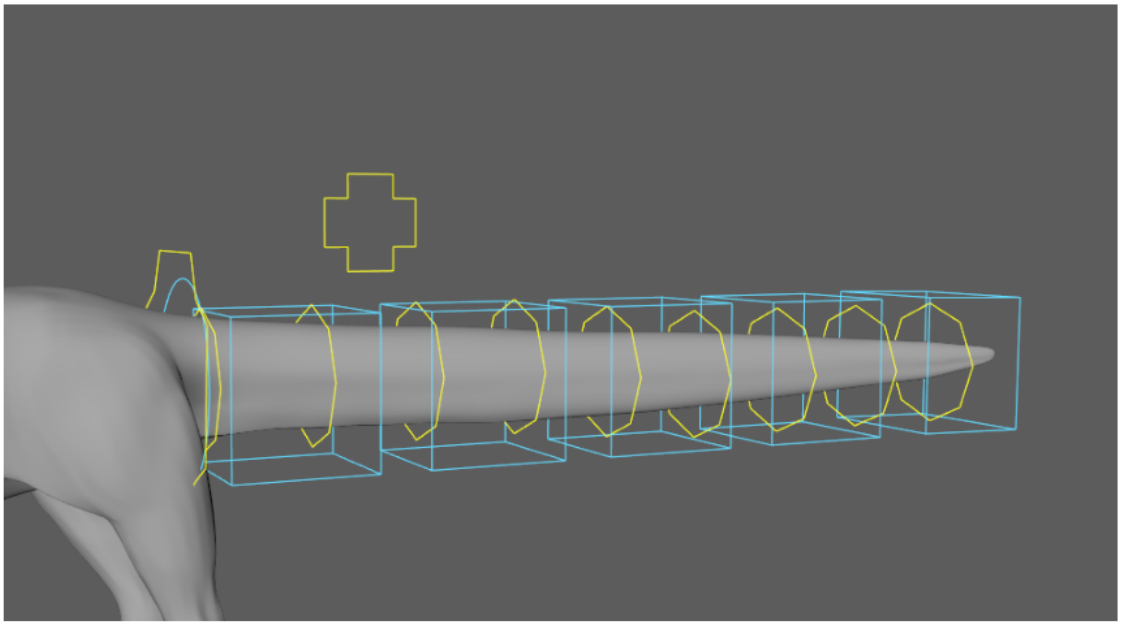

Illustration 49: IK system of the tail

El monstre té un seguit de 21 tentacles al cap, animarlos totalment a mà es bastant laborios. Per tant tenen un sistema propi de control. Per defecte hi ha els controls de moure cada tentacle individualment en la seva totalitat. Es poden extreure els controls que defineixen la trajectòria principal del tentacle. Sota de sistema de trajectoria del IK te dues curves dinàmiques, una té aplicat un camp de turbulencia de baixa frequencia que li dóna la forma general i l'altre un camp de turbulencia de altre frequencia que li dóna el micro detall al tentacle. Per a que funcioni la evaluació dels nodes ha d'estar establerta en Dependency Graph.

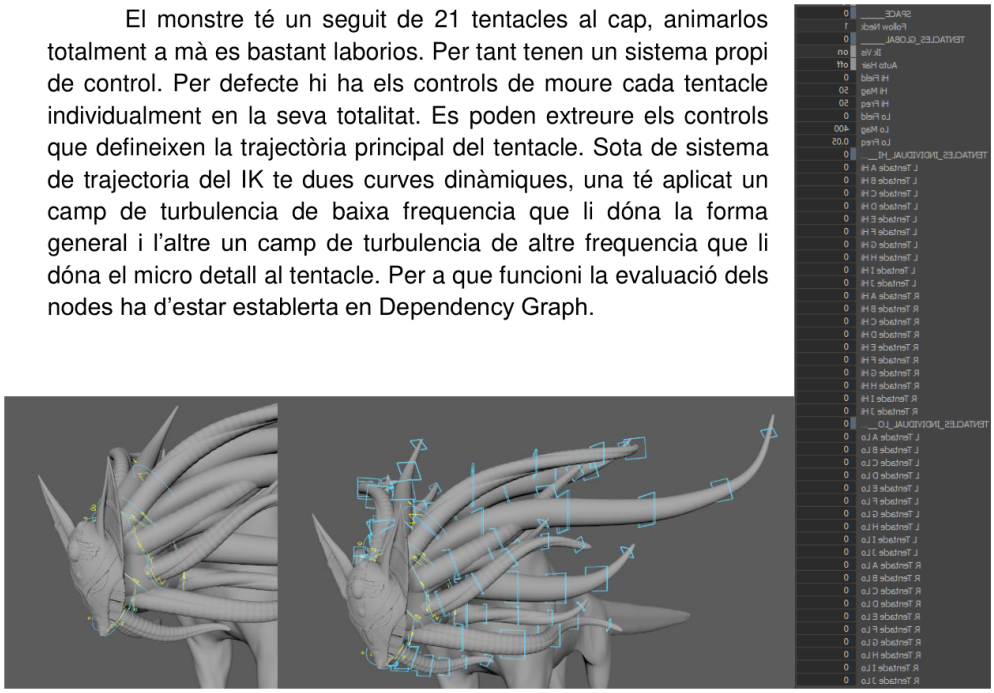

Illustration 50: Directed dynamic system of the tentacles

Nut

It is the "World Of Lost Things" robot. It is a custom-made character since it is not a biped and needs specific features for animation.

In the arms it has a bend-bones system, to give the character waves.

Both in the eyebrows and in the antenna it has an IK system, with two main controls and two tangency controls, that gives it the ability to stretch and make organic shapes out of static and mechanical objects.

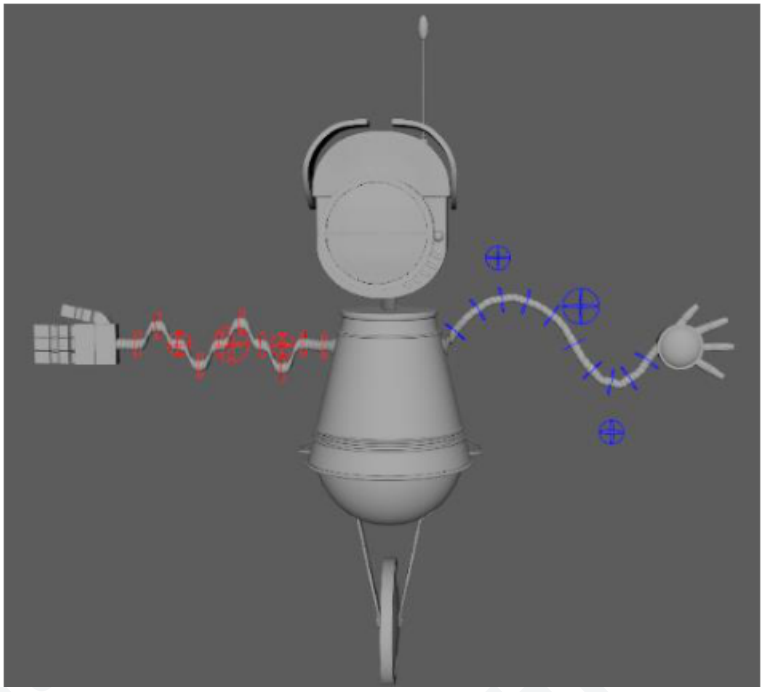

Illustration 51: Bend-bones system on Nut's arms

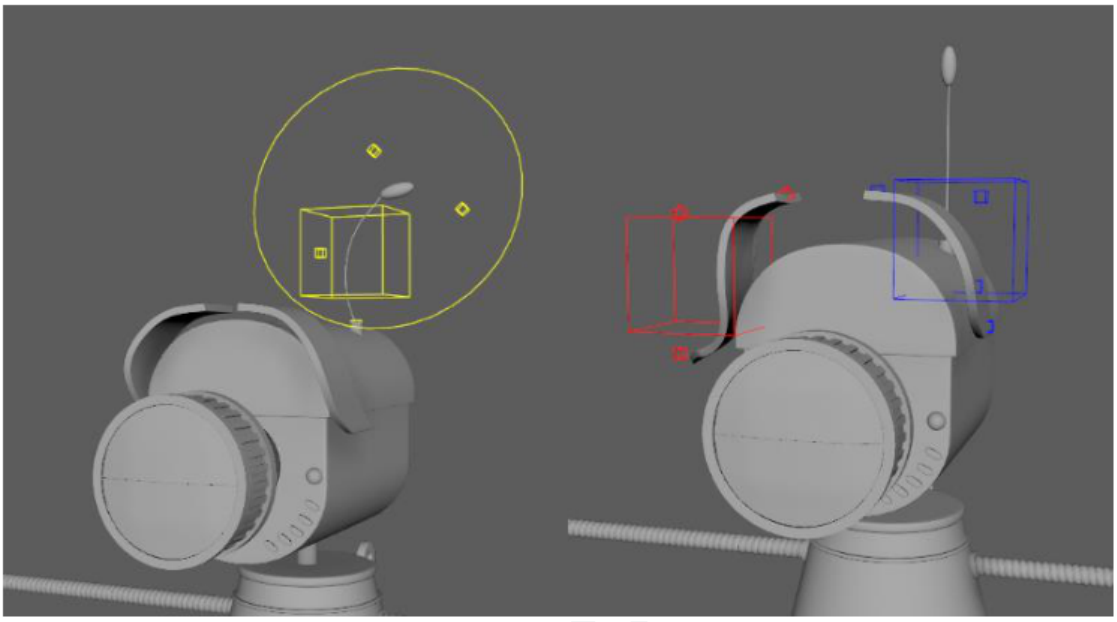

Illustration 52: IK control system for Nut

Human IK

For the “Cardiff University” project, characters that could work with the mocap system were needed. Some had to carry animation on top and the others only the mocap data. The system is also procedural.

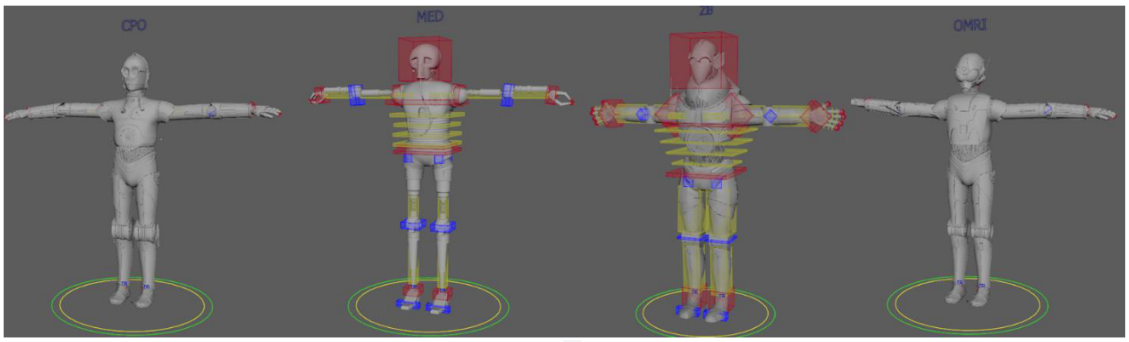

Illustration 53: Human IK characters

Static

Two characters were also needed that would be static but whose position, rotation and scale could be animated.

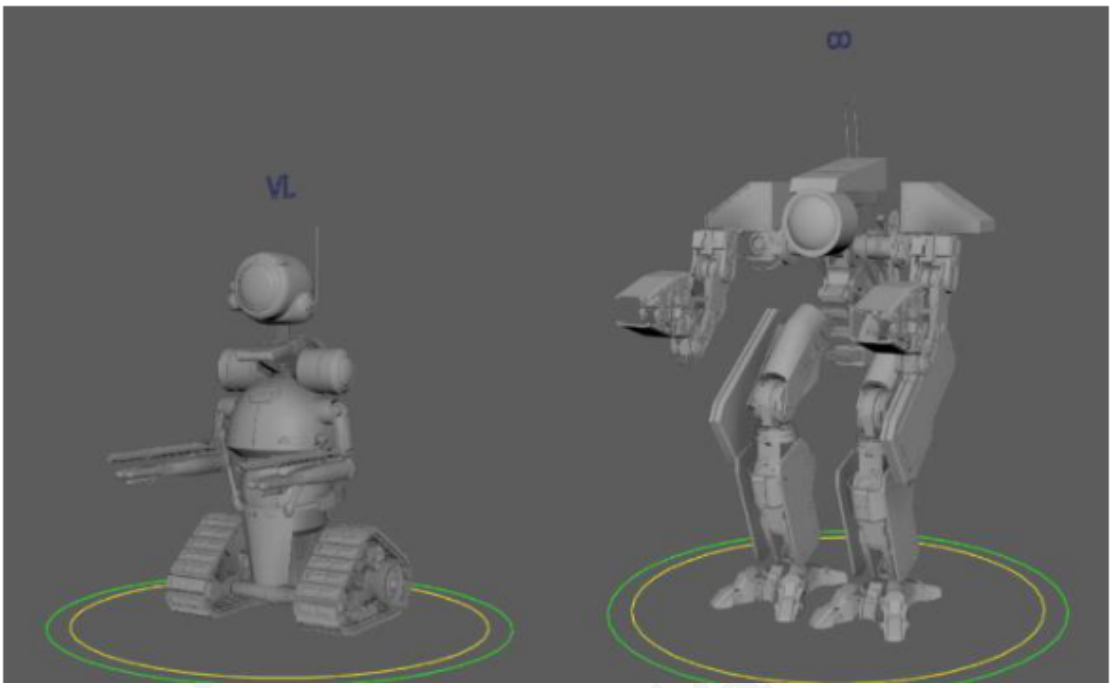

Illustration 54: Characters with general controls

Pipeline

The pipeline is the part of production responsible for managing the exchange of data between the parts of the production. Correct version management is the key to a production running correctly and in an orderly way.

There are tools that control all of this information, for example: “Ftrack” and “Shotgun”.

On the other hand, scene management also has to be controlled; this can be done with various systems and programs. As assemblers there are: “Katana”, “Clarisse” and “Houdini”, programs capable of managing large amounts of geometry. There is also the USD system, developed by “Pixar”, which helps with management across all programs, and “Hydra” to visualize a viewport with the objects.

For the projects, it was decided not to use any of these systems because we did not master them and because there was no support. This complicated the entire management of the project, since all of the asset management was done manually, where human error is more present.

As practical pipeline work, I developed a system so that a set of tools (both my own and pre-existing ones) would be on all computers just by placing them in a folder on the server. This system was named “LS_mayaToolkit”. For it to work, the following script needs to be placed in a userSetup.py file in Maya's scripts path.

```
import os
import pymel.core as pm

LS_PATH = 'T:\\maya'
if LS_PATH in os.sys.path:
os.sys.path.remove(LS_PATH)
os.sys.path.insert(0, LS_PATH)

pm.evalDeferred('import LS_mayaToolset')
```

Table 1: userSetup.py file

There you will find a collection of tools and scripts useful for production.

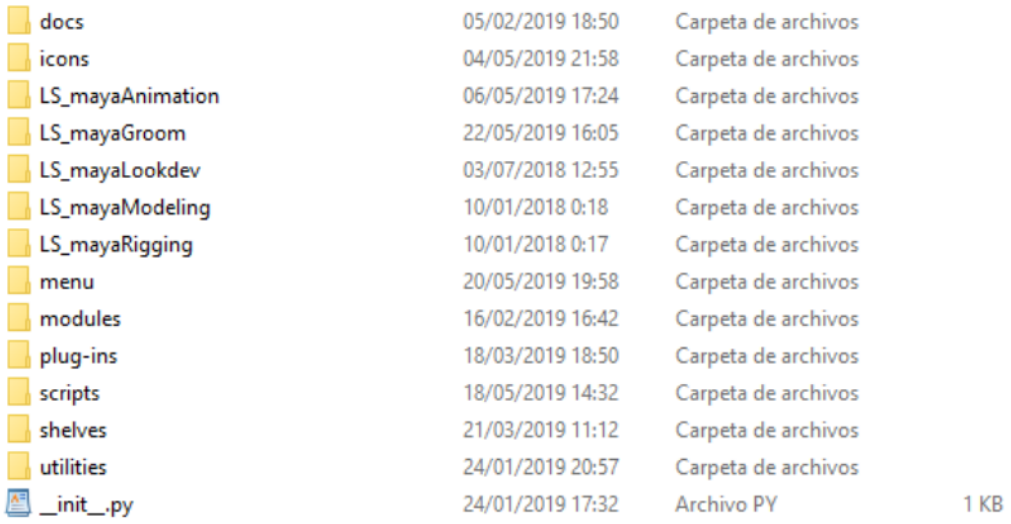

Illustration 55: Folder structure of utilities on the server

Visible tools

LS_Menu: This is the display of a set of tools in a menu in Maya.

LS_mayaAnimation: contains tools to help with animation

- aTools
- AnimFix
- Character importer

LS_mayaGroom: utilities for groom management

- Import
- Export

LS_mayaLookdev:

- Light transformations
- Changing the exposure of the selected lights
- Duplicating objects with history
- Changing color-reading profiles when using OCIO ACES 1.0.3

LS_mayaModeling

- absSymMesh: Tool to operate between geometries, check symmetry and fix it if necessary.
- Braid Creation: Tool to create braids
- goZ: Tool that swaps the Maya scene to zBrush and from zBrush to Maya.

Non-visible tools

These are the non-visible tools that only work by command.

Modules

brSmoothWeights. developed by “brave rabbit”, its function is to smooth the geometry weights with a more advanced algorithm than the one built into Maya

Plug-ins

AnimSchoolPicker. Design and use of a GUI to make control selection sets; it can be designed to taste. It serves to optimize control selection. Created by Anim School.

dcSkin. Developed by David Cuellar, a tool to save and load weights in binary files.

extractDeltas. Developed by “brave rabbit”, it serves to extract the variation between a corrective shape and the same one with the deformers applied. As its name says, it extracts the deltas of the mesh.

jQuadCloth. Developed by Jacopo Ortolani. It retopologizes geometries with many polygons into ones where everything is quadrangulated. It works especially well with clothing.

ngSkinTools. Developed by Viktoras Makauskas. It serves to operate with weight maps. It gives the ability to have additive weight layers. Internally it does all the calculations so that they are normalized.

smoothSkinClusterWeight. Developed by “brave rabbit”, it is the predecessor of the brSmoothWeights module; it does not have as many utilities and works more slowly.

transferSkinCluster. Developed by “brave rabbit”. It serves to copy skinning between geometries.

Scripts

advancedSkeleton5. Auto-rig for creating structures to control the geometry.

cometScripts. Toolkit developed by Michael Comet, containing all kinds of tools. The best known is the Comet Renamer.

cvshapeinverter. Developed by Chad Vernon, it serves to extract the inverse shape of a deformation and keeps it dynamically.

MG-PickerStudio. Picker similar to AnimSchoolPicker, developed by Miguel Gao.

mGear. Rigging framework developed by Miquel Campos, it gives great flexibility for creating rig structures.

ml_tools. A set of tools and utilities for animation and rigging, developed by Morgan Loomis.

studioLibrary. Tool created by the GitHub user “krathjen”, to save poses and animations and thus be able to transfer them between files.

Zen. Toolkit developed by David Belais, containing a set of all kinds of extra tools for Maya.

zooTools. A set of tools developed by Andrew Silke.

cosmos. Tool that manages access to Maya's windows. Developed by Martin Gunnarsson.

sortCircleTool. Tool to make circles on the set of selected edges.

bosSmear. Tool to deform the geometry according to the camera shot.

bSkinSaver. Tool developed by Thomas Bittner, to save the weights of skinCluster nodes.

changeColors. Script to change the colors of the controllers in bulk.

compactRenamer. Script by Erik Lehmann to rename geometries.

delete turtle. Script to delete the turtle node from the scene.

DPK_reorderAttrs. Tool developed by Daniel Pook-Kolb to reorder attributes in the channel box.

Dupes. Script to check whether there are duplicate objects in the scene. Developed by Jorn-Harald Paulsen.

mtAlignTool. Tool to align objects in the viewport.

tweenMachine. Tool to interpolate keyframes, developed by Justin S. Barrett.

ACES

The Academy Color Encoding System (ACES) is becoming established as the standard for color management throughout film and TV production.

It is set up as a linear and lossless color-management process. This way, the color chain remains unaltered from the recording of the audiovisual piece to its display, passing through the creation of the 3D.

Illustration 56: Structure of ACES data management throughout production

To use the ACES configuration inside Maya, you first have to download the OCIO repository from Sony (https://github.com/imageworks/OpenColorIOConfigs/tree/master/aces_1.0.3). Once downloaded, the following configuration has to be activated.

And to change the color-space configuration of all the files that are loaded, they have to be changed to utilities (https://github.com/enriquevelmai/utils/blob/master/toAces.py)

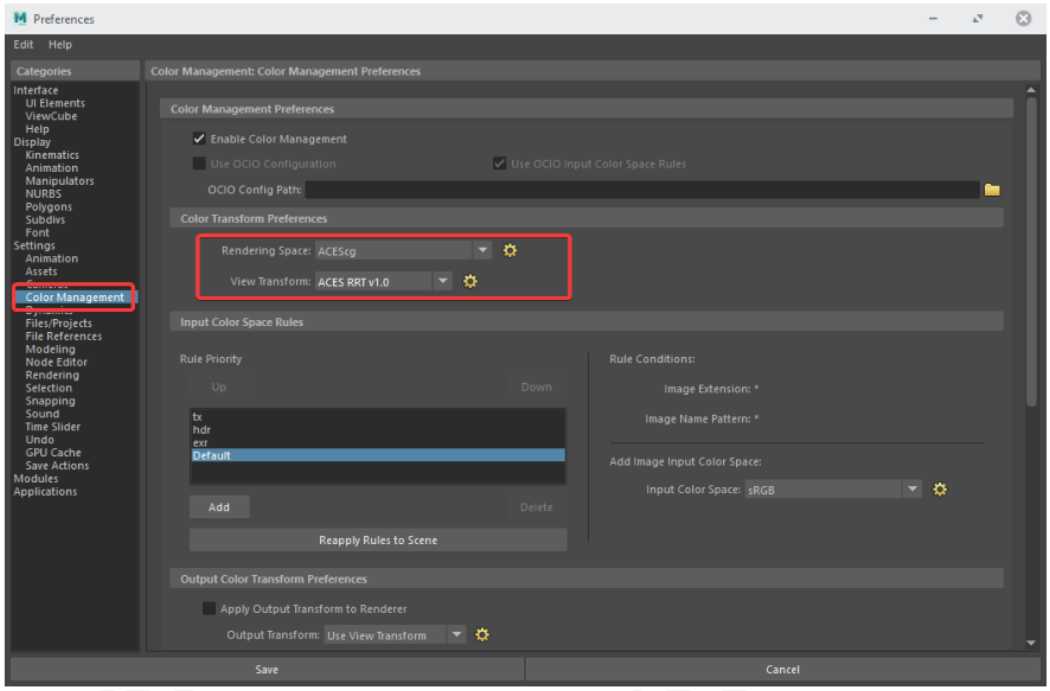

Illustration 57: Color-management configuration inside Maya with the ACES profiles

“Current” script

Since no version manager was used, a “current” script was created which read the latest versions and created a folder with them.

Groom

Managing the groom on the project became a headache, since we all had very little experience managing hair in the pipeline.

It was decided to use Maya's xGen for the hair. After researching in depth how xGen works internally, our own way of managing the hair was derived.

xGen works by having a base geometry to which a set of collections and descriptions are applied.

Collections are the grouping of many descriptions on one or several geometries. In the descriptions is where the hair is groomed from a set of guides, maps and expressions. The maps that are painted are in PTX format.

xGen's information is saved in a dedicated green node in a Maya scene, which is a plug-in that reads information from two places: everything related to the collections, descriptions and expressions is saved in an .xgen file with the saved data; the map information is saved in a folder hierarchy inside the project's xGen folder (or the location where it is defined), where there are folders for each collection, description and modifier.

Structure of the .xgen file

Inside the xgen file the data is structured as follows

Globals

The “palette” is the base, what amounts to the collection.

Definition of the xgen global variables, mapping of where the project's base is.

```
Palette
    name            LB_Yura_default
    parent          groom
    xgDataPath      ${PROJECT}xgen/collections/LB_Yura_default
    xgProjectPath       //nassus/Project/VFX_Project_02/Last_in_Battle/PROJECT/01_Assets/01_Characters/Yura/06_Grooming/Lb_Yura_Grooming/
    xgDogTag
    endAttrs
```

Illustration 58: Global options of the collection

Per description

General description data; there is one for each description

```
Description
    name            Fringe
    flipNormals     false
    strayPercentage     0.0
    lodFlag         false
    averageWidth        1.0
    pixelCullSize       0.0
    pixelFadeSize       20.0
    cullFade        0.1
    minDensity      0.01
    cullWidthRatio      0.01
    maxWidthRatio       20.0
    groom
    descriptionId       1
    xgDataPath      ${PROJECT}xgen/collections/LB_Yura_default/
```

Illustration 59: General options of the description

Display of the guides in the viewport

```
GeometryRenderer
    percent         25.0
    startPercent        0.0
    inCameraOnly        false
    inCameraMargin      0.0
    outputInstances     true
    useWidthRamp        true
    geometryType        0
    shapeType       0
    convertSelected     false
    combineMesh     false
    createStripJoints   false
    stripJointPlacementType 0
    jointNumOnStrip     3
    createGuideJoints   false
    guideJointPlacementType 0
    jointNumOnGuide     3
    meshOrientation     0
    insertWidthSpan     false
    widthSpanNum        1
    curvature       0.500000
    uvInTiles       true
    uvLayoutType        0
    uvTileSeparation    0.000000
    endAttrs
```

Illustration 60: Display options of the primitives in viewport

Display of the guides at render; you can define the type of render you want, which scalp it follows and whether it has motion blur or not. Overrides can be made for each of the variables. You can see that the render engine xgen uses is “RenderMan”, since it is Pixar's render engine.

```
RendermanRenderer
    percent         100.0
    startPercent        0.0
    inCameraOnly        false
    inCameraMargin      0.0
    length_XP       true
    width_XP        true
    T_XP            false
    stray_XP        false
    id_XP           false
    descid_XP       false
    ri_XP           true
    rf_XP           true
    u_XS            true
    v_XS            true
    faceid_XS       true
    geomid_XS       false
    geomName_XS     true
    P_XS            true
    Pref_XS         false
    Pg_XS           false
    Prefg_XS        false
    N_XS            true
    Ng_XS           false
    Nref_XS         false
    Nrefg_XS        false
    dPdu_XS         true
    dPduref_XS      false
    dPdug_XS        false
    dPdurefg_XS     false
    dPdv_XS         true
    dPdvref_XS      false
    dPdvg_XS        false
    dPdvrefg_XS     false
    renderer        None
    renderMethod        2
    draMode         0
    primitiveBound      1.0
    custom__arnold_rendermode   0
    custom__arnold_curveMode    0
    custom__arnold_minPixelWidth    0.0
    custom__arnold_motion_blur  0
    custom__arnold_motion_blur_mode 1
    custom__arnold_motion_blur_steps    2
    custom__arnold_motion_blur_factor   0.5
    custom__arnold_useAuxRenderPatch    0
    custom__arnold_auxRenderPatch   0
    custom__arnold_multithreading   1
    endAttrs
```

Illustration 61: Display options of the primitives at render

The general data of the description's guides can also be defined.

```
SplinePrimitive
    _patchNames
    length          $a=1.0000;#0.05,5.0\n$a
    width           rand(0.01,0.03,163)
    depth           $a=1.0;#0.05,5.0\n$a
    offU            $a=0.0000;#-2.0,2.0\n$a
    offV            $a=0.0000;#-2.0,2.0\n$a
    offN            $a=0.0000;#-180.0,180.0\n$a
    aboutN          $a=0.0000;#-180.0,180.0\n$a
    regionMap       ${DESC}/RegionFringe
    regionMask      1
    iMethod         1
    useCache        false
    liveMode        true
    _wireNames
    cacheFileName       ${DESC}/guides.abc
    attrCVCount     3
    bendParam[0]        $a=0.5000;#0.0,1.0\n$a
    bendU[0]        $a=0.0000;#-2.0,2.0\n$a
    bendV[0]        $a=0.0000;#-2.0,2.0\n$a
    fxCVCount       35
    uniformCVs      true
    taper           $a=0.0000;#-1.0,1.0\n$a
    taperStart      $a=0.0000;#0.0,1.0\n$a
    displayWidth        true
    faceCamera      true
    tubeShade       false
    tubes
    guideSpacing        1.0
    guideMask       1.0
    cutParam        1.0
    texelsPerUnit       10.0
    CVFrequency     1.0
    widthRamp       rampUI(0.0,0.605263157895,1:0.152597402597,0.75,2:0.808441558442,0.697368421053,1:1.0,0.460526315789,1)
    endAttrs
```

Illustration 62: General options of the guides in viewport

Next is the modifier data; they are loaded as FX packages (established by the API).

```
ClumpingFXModule
    active          true
    mask            $a=map('${DESC}/paintmaps/FringeMask2');#3dpaint,256.0\n$a\n
    name            Clumping1
    cvAttr          false
    mapInitialized      True
    pointDir        ${DESC}/${FXMODULE}/Points/
    mapDir          ${DESC}/${FXMODULE}/Maps/
    clump           1
    clumpScale      rampUI(0.0132450331126,0.539473684211,3:0.331125827815,0.447368421053,1:0.634868421053,0.276315789474,1:1.0,0.0,1)
    clumpVolumize       false
    clumpVariance       0.0
    cut         0.0
    copy            0.0
    copyScale       rampUI(0.0,0.0,3)
    copyVariance        0.0
    curl            0.0
    curlScale       rampUI(0.0,0.5,3)
    offset          0.0
    offsetScale     rampUI(0.0,0.0,3:0.5,1.0,3:1.0,0.0,3)
    flatness        0.0
    flatnessScale       rampUI(0.0,0.0,3)
    frame           0.0
    noise           0.0
    noiseScale      rampUI(0.0,0.0,3)
    noiseFrequency      0.0
    noiseCorrelation    0.0
    exportCurves        false
    exportDir       curves/
    exportFaces
    texelsPerUnit       10.0
    radiusVariance      0.5
    ptDensity       1.0
    ptMask          1.0
    ptLength        1.0
    colorPreview        false
    useControlMaps      1
    controlMask     1
    controlMapDir       ${DESC}/RegionFringe
    endAttrs
```

Illustration 63: Options of the clump modifier

Other examples of data saved by other types of modifiers.

```
NoiseFXModule
    active          false
    mask            1.0
    name            Noise3
    frequency       1.0
    magnitude       $min=0.0100;#0.00,0.20 \n$max=0.2000;#0.20,10.00\n$seed=4;#0,10\n$mag=rand($min,$max,$seed);
    magnitudeScale      rampUI(0.0,0.0,1:0.370860927152,0.421052631579,1:1.0,0.75,1)
    correlation     0.0
    preserveLength      0.0
    mode            0
    bakeDir         ${DESC}/${FXMODULE}/
    endAttrs

CutFXModule
    active          true
    mask            1.0
    name            Cut1
    amount          rand(0.0,0.2)
    rebuildType     1
    endAttrs
```

Illustration 64: Options of the Noise and Cut modifiers

The random generator space handles the density, offset and other generic properties.

```
RandomGenerator
    displacement        $a=0.0000;#-1.0,1.0\n$a
    vectorDisplacement  0
    bump            $a=0.0000;#-1.0,1.0\n$a
    offset          $a=0.0000;#-1.0,1.0\n$a
    cullFlag        false
    cullBackface        false
    cullFrustrum        false
    cullAngleBF     0.0
    cullAngleF      0.0
    cullExpr        $a=0.0000;#0.0,1.0\n$a
    density         250.0
    mask            $a=map('${DESC}/paintmaps/DensityFringe');#3dpaint,256.0\n$a\n
    dcFlag          false
    scFlag          true
    usePoints       false
    pointDir        ${DESC}/Points/
    ptLength        1.0
    endAttrs
```

Illustration 65: General options of the description

Display properties of the guides in the viewport.

```
GLRenderer
    percent         100.0
    startPercent        0.0
    inCameraOnly        true
    inCameraMargin      0.0
    patchNames      false
    faceIds         false
    primIDs         false
    primIDsAt       1.0
    vertices        false
    poly            false
    culled          false
    unitCube        false
    color           map('${DESC}/Clumping4/Maps/')
    guideColor      map('${DESC}/RegionFringe')\n#$a=[1.0,0.4313725,0.0];#color\n#$a
    TEXCOORD3       [ $cWidth, 0, 0 ]  # red channel reserved by XGen
    TEXCOORD4
    TEXCOORD5
    TEXCOORD6
    TEXCOORD7
    splineSegments      2
    primNumLimit        100000000
    endAttrs
```

Illustration 66: Display options of the guides in viewport

The map textures section says where part of the modifier maps are read from. It also indicates which parts of xgen are active for the description in question and which render method is used in the viewport.

```
MapTextures
    SplinePrimitive regionMap   P:/VFX_Project_02/Last_in_Battle/PROJECT/01_Assets/01_Characters/Yura/06_Grooming/Lb_Yur
    Clumping1   mask    P:/VFX_Project_02/Last_in_Battle/PROJECT/01_Assets/01_Characters/Yura/06_Grooming/Lb_Yura_Groomi
    Clumping1   controlMapDir   P:/VFX_Project_02/Last_in_Battle/PROJECT/01_Assets/01_Characters/Yura/06_Grooming/Lb_Yur
    Clumping2   controlMask P:/VFX_Project_02/Last_in_Battle/PROJECT/01_Assets/01_Characters/Yura/06_Grooming/Lb_Yura_Gr
    Clumping2   mask    P:/VFX_Project_02/Last_in_Battle/PROJECT/01_Assets/01_Characters/Yura/06_Grooming/Lb_Yura_Groomi
    Clumping3   controlMask P:/VFX_Project_02/Last_in_Battle/PROJECT/01_Assets/01_Characters/Yura/06_Grooming/Lb_Yura_Gr
    Clumping3   mask    P:/VFX_Project_02/Last_in_Battle/PROJECT/01_Assets/01_Characters/Yura/06_Grooming/Lb_Yura_Groomi
    Clumping4   controlMask P:/VFX_Project_02/Last_in_Battle/PROJECT/01_Assets/01_Characters/Yura/06_Grooming/Lb_Yura_Gr
    Clumping4   mask    P:/VFX_Project_02/Last_in_Battle/PROJECT/01_Assets/01_Characters/Yura/06_Grooming/Lb_Yura_Groomi
    RandomGenerator mask    P:/VFX_Project_02/Last_in_Battle/PROJECT/01_Assets/01_Characters/Yura/06_Grooming/Lb_Yura_Gr
    endAttrs

    Active  SplinePrimitive
    Active  RandomGenerator
    Active  RendermanRenderer
    Preview GLRenderer
```

Illustration 67: Read paths of the painted maps

Anchoring

Once all the display and modification data of each of the descriptions has been defined, all the data related to the scalps is defined, where the positions of each of the guides, the patches they attach to, the CVs they have and many more are reflected. Below is an excerpt of this part of the file, since it is the largest section of the file.

```
Patches Fringe  75
Patch   Subd
    name        C_Scalp
    faceIds     478  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 3
    culledPrims 1   267 3  13 71 139
    animCurves  0
Guides  Spline  75
    id          153
    loc         2.4972599744796753e-01 4.3792399764060974e-01 255
    blend           0.0000000000000000e+00
    interp          2.4695467396369639e+00:2.4695367396369639e+00:1.7584112027669696e-01:1.6502745752980459e+00:8.951390
    CVs         10
    1.1121873270923022e+00 2.3192748608178423e-01 -4.6769724034053983e-01
    2.4943217153189980e+00 1.5723913981566215e-01 -6.0031637909566926e-01
    3.8385200089239566e+00 -1.1658772585250568e-02 -5.9032405953457645e-01
    5.0798875946303816e+00 -3.4336530525622094e-01 -8.8084518318328853e-01
    6.2716621619440565e+00 -8.1470959240584917e-01 -1.4184699771725648e+00
    7.3988718233863109e+00 -1.2610266666336747e+00 -2.0203560342579467e+00
    8.5462482870015268e+00 -1.7006292268026326e+00 -2.5085058680913224e+00
    9.6599110699878832e+00 -2.1136057934431891e+00 -3.1745458208112067e+00
    1.0304415305916406e+01 -2.4258583848547848e+00 -4.1200042724164705e+00
```

Illustration 68: Anchoring properties between the guides and the geometry

Map-saving structure

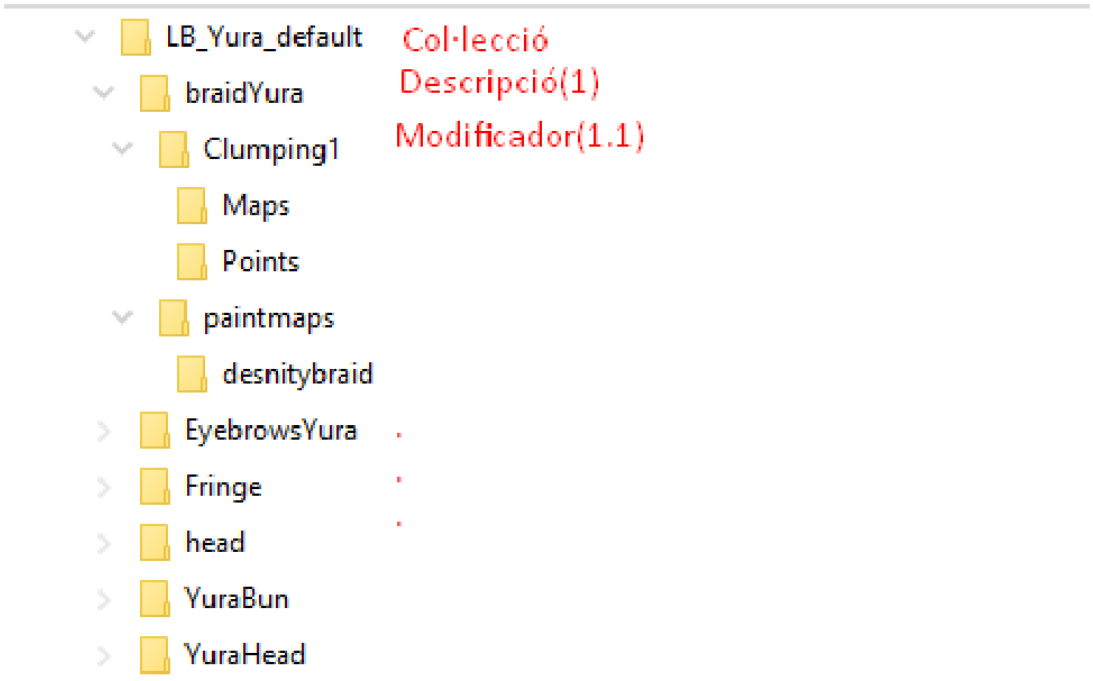

Illustration 69: Folder structure for saving the maps

Packaging concept

As can be seen, there are many variables involved in managing the hair, so the concept of hair packaging is defined. It is about having a container that holds all the data needed to export and reassemble the hair deformation system.

To be able to manage xGen, we define having the following elements:

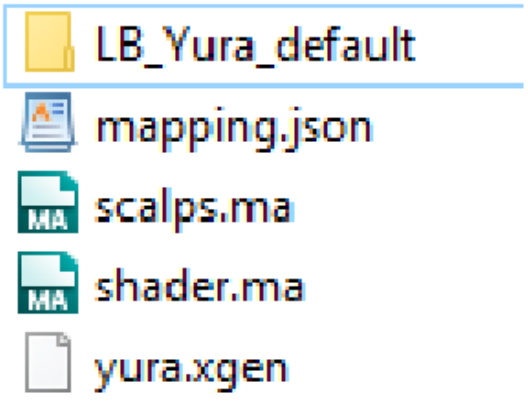

Illustration 70: Custom groom package structure

The folder contains the entire structure of PTX maps of the modifiers and descriptions.

The mapping.json file contains useful information for reassembling the groom.

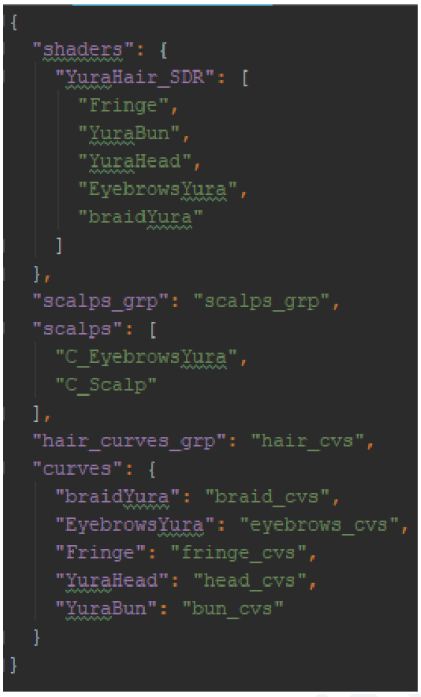

Illustration 71: Custom JSON structure for saving information

The scalps.ma file contains the scalps from which the hair grows; it has to be exported keeping the history of the geometries, since they have maps and file nodes connected so that it can be reassembled without losses.

The shader.ma file contains the hair material.

The .xgen file is the export of the file mentioned above.

Animation vs Simulation

There are two ways to manage the hair moving. The first is to simulate it: once the animation is done, have the hair move following physical dynamics and force fields. The second is to animate a proxy mesh onto which curves are attached, and these are used to move the groom. It is important to export an .abc file with the specific guides for each description and for each shot.

Data management

To manage this data without falling into human error — since there are many steps — we define a script to handle these things.

The following libraries enable the use of the xGen commands:

```
import xgenm as xg
import xgenm.XgExternalAPI as xge
import xgenm.xgGlobal as xgg
```

Table 2: Import of the xGen libraries

These tools are dedicated to importing and exporting the generic groom in a generic way.

The exporter needs to define the xGen node, select the scalps, the shader and the path where the previously defined package will be exported.

The importer needs the path of the package, the alembic with the shot's animation and the xGen path in case it has changed.

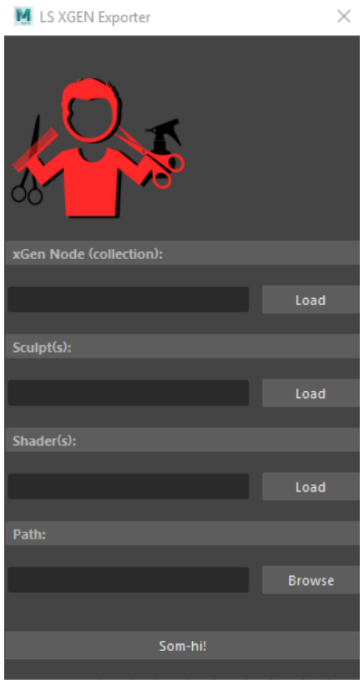

Illustration 72: Tool to export xGen

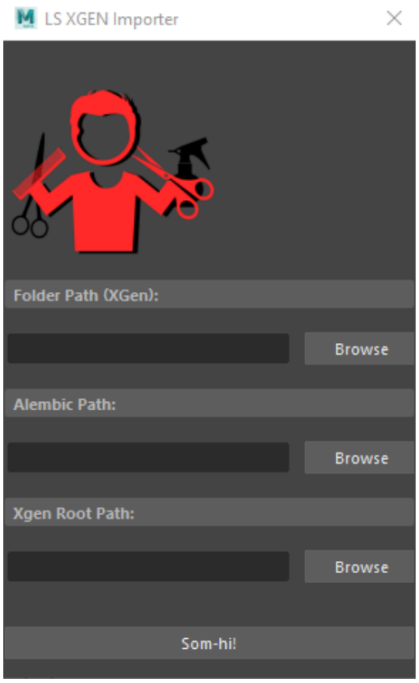

Illustration 73: Tool to import xGen

In addition to the previous tools, there is a tool specific to Last in Battle, based on its own file management. Here you only need to define the asset, the shot and the scalps group.

Illustration 74: Tool to import xGen, adapted for Last in Battle

Fixing

Once the animation is done, there may be geometry interpenetrations or the desired deformation may not be achieved, and you have to move on to the shot sculpting part.

There, the animation cache is taken and, with the following tool, the desired deformation is modeled.

Method of use:

1. Position yourself on the frame where you want to model the correction
2. Open the tool
3. Select the geometries on which you want to model the correction
4. Define the tail frames the correction will have (a blendshape with 3 keys, two of 0 and one of 1)
5. Press the Active button and model them
6. Press the Finish button; the geometries are deleted

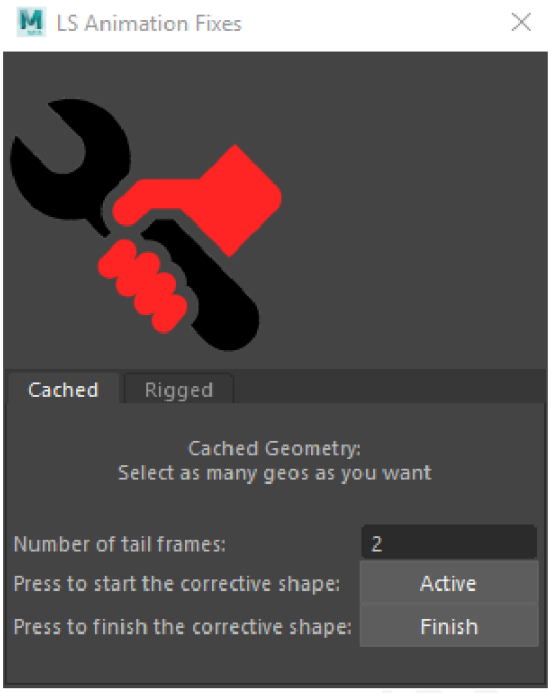

Illustration 75: Tool to do shot sculpting

Conclusions

Carrying out this project has served to gain a global and complete view of the development of a project from start to finish. Going through each of the areas, I have been able to see what the complications and the processes to follow are at each phase. It was very hard to know when to close a process — that is, when to establish that the model, textures, rig, animation, etc. was already enough and the asset had to be let flow through the pipe. Working in a group, we had to organize ourselves, since doing a project entirely alone is not the same. The scenes had to have a specific name, the order in the outliner had to be very carefully kept, and the materials published with the fewest possible errors. Being the first time we faced — as a university and as a group — a full production, the way in which we had to proceed was a challenge. For example, how to manage the hair. Since we did not simulate it but animated it, it was hard to find a way to join the animation of the guides with the groom guides. Once it worked locally, we realized it did not work on the farm, since “Deadline” did not change the mapped network-drive paths configured as a letter to the full path in the xGen files. Another difficulty was how the materials joined with the animated geometry. The options considered were: the rig carries the materials; a piece of code is written that imports and configures the materials once the animation was computed; or a system by Alex Leon consisting of assembling a scene with the materials applied and then having it remain as a reference in the lighting scene.

Creating a fully procedural, object-oriented system is more tedious than it seems. You have to structure and design it quite a lot beforehand, making assumptions about the obstacles that will appear along the way. I have also kept an eye on the new methods being published and the tools coming out, so as to be able to adapt them quickly to the system created. Personally, creating and maintaining 13 characters with all the rig fixes and geometry changes was very hard for me. Now, seeing it in perspective, I see that choosing to make the rigs procedural was a good decision, since I find myself incapable of having done it the “artisanal” way, in addition to the tool implementation. One of the main problems on the rigs was the flipping of the advanced twist on the IK spline systems; after

days of struggling, I brought the problem to work and they advised me to use the IK spline to compute the position and, from there, create an offset curve to the IK one to mark the position, and use a projection of this onto the offset curve, using this second curve as the up vector of an aim system, where the bone points to the next one. This solution fixed the flippings; the limb, neck, spine and tentacle modules have these systems implemented. Double transformations also caused headaches when scaling the model; creating groups that do not want the transformations in each rig module was the solution to this problem.

Of all the characters, the one that was a greater challenge due to the complexity of the systems was the monster, because of its tentacles. Even though the systems work correctly, the rig runs very slowly when the automatisms are activated. I would like to find a solution later on to fix this point against the rig.

The systems for projecting transform nodes onto a surface were one of the most key points of the whole project, since they gave a lot of play to the facial systems of eyebrows, lips and eyelids.

Painting weights in dual quaternion in some parts has saved modeling many corrective shapes. The use of the new brSmoothSkinWeights tool was also one of the year's biggest discoveries; it has sped up the work a lot.

Being one person who developed the system and has been maintaining it, the production time was limited, and many systems that were planned as extras could not be carried out. Thank goodness for the teammates who were there helping and making things work from the upstream groom and modeling departments. And for the help there was in starting to model the facial shapes. It would also have helped to have someone present day to day who could lend a hand with the tool programming, since many things are beyond me, and having someone with experience would have been very productive; I would also have learned other ways of proceeding, not just mine.

Currently the Rigging Toolkit code is very messy at the code level. I would like to give the system another pass and abstract it even more, to leave only the most generic classes and limit the rig to a spline system and a surface-projection system, with everything inheriting from these. Also, change the use of maya.cmds for PyMel, since it is fully object-oriented. Apart from that, it would be great to give it a graphical interface to make it more usable.

I really enjoyed developing tools for pipeline, since it was something I had never done and it turned out quite well.

References

[1] http://introtorigging.blogspot.com

Bibliography and Webography

ROB O'NEILL. Digital Character Development: Theory and Practice. Second Edition. A K Peters/CRC Press. 27 October 2015. ISBN: 1482250772

W. ELLENBERGER. An Atlas of Animal Anatomy for Artists (Dover Anatomy for Artists). Dover Publications Inc.; New impression. 1 December 1966. ISBN: 0486200825

ELIOT GOLDFINGER. Animal Anatomy for Artists: The Elements of Form. OUP USA. 11 March 2004. ISBN: 0195142144

ALBERTO LOLLI. Struttura uomo. Manuale di anatomia artistica. Colla Editore; 2nd ed. 19 April 2018. ISBN: 8894272214

WILLIAM C. VAUGHAN. The Pushing Points Topology Workbook: Volume 01. CreateSpace Independent Publishing Platform. 5 April 2018. ISBN: 1987728610

TINA O'HAILEY. Rig it Right! Maya Animation Rigging Concepts, 2nd Edition. Routledge; 2nd ed. 27 July 2018. ISBN: 9781138303164

JASON OSIPA. Stop Staring: Facial Modeling and Animation Done Right. John Wiley & Sons Ltd; 3rd ed. 8 October 2010. ISBN: 0470609907

GOPINATH JAGANMOHAN, VENKATESHWARAN LOGANATHAN. PySide GUI Application Development - Second Edition. Packt Publishing; 2nd Revised ed. 28 January 2016. ISBN: 178528245X

ADRIAN HERBEZ. Maya Programming with Python Cookbook. Packt Publishing. 29 July 2016. ISBN: 1785283987

ADAM MECHTLEY, RYAN TROWBRIDGE. Maya Python for Games and Film: A Complete Reference for Maya Python and the Maya Python API. CRC Press; 1st ed. 1 November 2011. ISBN: 0123785782

ROBERT GALANAKIS. Practical Maya Programming with Python. Packt Publishing. 25 July 2014. ISBN: 1849694729

MARK SUMMERFIELD. Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming. Prentice Hall; 1st ed. 4 September 2015. ISBN: 0134393333

KIARAN RITCHIE, JAKE CALLERY, KARIM BIR. The Art of Rigging. Volume I-II-III.

XGen Python API. Available at: https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/Maya/files/GUID-33ECC43B-5CF6-4BE1-8EAE-8C6C0D698020-htm.html

Python API 2.0. Available at: http://help.autodesk.com/view/MAYAUL/2018/ENU/?guid=__files_GUID_B1CD0989_EB49_46BE_934F_F23BFB869142_htm

PyMEL for Maya. Available at: http://help.autodesk.com/view/MAYAUL/2018/ENU/?guid=__PyMel_index_html

Maya Commands. Available at: http://help.autodesk.com/view/MAYAUL/2019/ENU/?guid=__CommandsPython_index_html

Josh Sobel Face Rigging. Available at: https://vimeo.com/ondemand/sobelfacerig/

Cult of Rig. Available at: http://www.cultofrig.com/

Skinning Techniques. Available at: http://www.3dfiggins.com/writeups/paintingWeights/

Blendshape Techniques. Available at: http://petershipkov.com/tutorials/blendShapes/blendShapes.htm

Bind Pose. Available at: https://bindpose.com/

Doug Schieber. Available at: https://www.dougschieber.com/new-index

MICHAEL TODD – xGen. Available at: https://www.mtodd.work/xgen-product-design

Allan McKay Podcast. Available at: https://www.allanmckay.com/1/

Beginners guide to: XGen pipeline for beginners. Available at: https://www.mikecauchiart.com/single-post/2017/08/29/Beginners-guide-to-XGen-pipeline-for-beginners