Luka Jokovic Martinez

Registration number 100280300

2023

Building a 3D Animation Rig using Blender

Supervised by Dr. Stephen Laycock

University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

Body movements and their conveyed emotions play a vital role in expression and communication. For hundreds of years, key research in the psychology and physiology fields has conveyed this importance. With the modern age's uprise in 3D film and movies animators have focused on capturing these expressions yet the differential gap between human expression and animated characters expression remains.

Blender is a powerful open-source 3D animation software that has become a popular choice with smaller independent animators for its ease of use and versatility when performing their needs.

This report delves into the creation of a 3D rig that smaller independent animators could use for their several needs. The project itself implements and combines meshes, armatures and deformation tools offered by Blender alongside the linear blend skinning deformation technique to produce the final product, a human-like rig with a fair degree of realistic muscular deformations as well as weight gain deformations.

Keywords: Rigging, Character Animation, Body Animation, Blender, Expressions, Anatomy, Realism.

Acknowledgements

I'd like to thank my family, friends and everyone who has taught me at UEA. Special thanks to David Greenwood and Stephen Laycock for their invaluable help throughout my project.

Contents

List of Figures

List of Tables

1. Introduction

The 3D animation industry has been on the rise, with the advancement in technology lifelike animations within video games and film are becoming more and more demanding and high quality. These industries look towards building 3D animation rigs which act as a foundation for creating realistic body movements and expression in their animation process. Whilst bigger animation companies have the supplies given to them to ease their rigging process, many smaller independent animators face more of a challenge when it comes to creating 3D animation rigs as a result of limited resources. This in turn results in their animations having a lack of realistic movements and expressions. The focus of this project it to develop a flexible, user friendly 3D animation rig that individuals with limited resources and experience can use.

The human body conveys a vast quantity of information through its movements. Thoughts, emotions, feelings and intentions are all able to be read from our body language alone. Accurately translating movements and their expressions to a 3D model requires an understanding of the human body anatomy. With the exception of some circumstances, video game users and film viewers expect a high level of realism, individuals are more likely to feel engaged with characters that accurately display expressions that they can read from body language. In order to apply these expressions our 3D model must directly relate to human body anatomy with bones being able to be rotated in realistic directions and angles and meshes to deform accurately in order to depict muscle movement. This is achievable through the use of a powerful and flexible 3D software, whilst there are many on the market Blender will be used in this project.

Blender is an open source 3D creation program that offers a wide variety of tools for rigging human models including bones, meshes, constraints and shape keys. Blender's interface offers a python scripting API that allows programmers such as myself to immerse with these tools. This brings me onto the main objective of this project, to use Blender's python API to script a realistic 3D animation rig with muscle movements with the primary users intended to be independent animators. On a more specific forefront, I will look into implementing muscle movement algorithms that can be applied to the models, adding constraints onto bones to prevent unrealistic movements and the possible implementation of changing the meshes weight to add variety in how my program is

used. To show the prioritization of my projects features a MoSCoW analysis has been conducted.

My project must have a efficient and realistic rig system, including key components of a rig: a mesh, a skeletal bone system and a form of producing human-like deformations. Making sure it's easy to use and works fluidly at a good speed without any crashes are also must haves.

My project should add constraints to my bones to prevent unrealistic movement. It should also have variety in in how aspects of the project van be done, for example, the ability to import several meshes. Lastly, it should have algorithms implemented that produce a good level of realism that can be displayed through twists, bulges etc.

My project could have a weight scaling factor in order to add variety to how the program can be used. This factor would work alongside muscle changes but should be done in a different manner to further expand my implementation abilities.

Considering my project is to be used by individual animators and the time I have, there won't be techniques applied that are too resource intensive or complicated. There will also be no integration with other 3D modeling software such as Maya or 3DSMax. Lastly, no actual animation processes will be performed in my project and would have to be done manually if the users chose to do so.

Aside from creating the program to aid individual animators I was personally motivated as a result of a career in general video games or films always being an option. I felt as if taking on this project would enhance my skills in programming within these potential fields.

2. Background

For some time now both researchers and animators have been looking into formulating the most efficient approach that creates rigs with lifelike movements. The earliest attempts can be be traced back to the late 1800s with Étienne-Jules Marey an his studies on the mechanics of human movement, Marey (1873). In more recent times research into neural blend shapes, harmonic skeletons, articulation curves and other techniques have been used in order to achieve this desired movement.

2.1. Articulation Curves

Pixar animation studio is a globally recognized film company recognized for their 3D work and so their character articulation approach through the use of profile curves is a useful method in order to grasp a better understanding of how models can be designed Goes et al. (2022). The method uses 2D curves that specify the desired deformation of the throughout several points of a mesh, which is know as a profile curves. The profile curves are created through modifying a selection of vertices to achieve a specific deformation before being used to find a set of joint angles that would produce this deformation by computing the articulation of the character. These angles are optimized before being used for whatever purpose the animator desires.

The paper itself did not provide comparative results against other character articulation methods making it hard to distinguish how effective the method would be with further unclear suggestions that articulation curves can be applied to all types of meshes. The paper did make the mathematical formulation required that computes the process clear, as seen in algorithm 1. The algorithm cannot be expected to understood easily, especially for those who don't work in a programming environment, which raises concerns with how easily it can be implemented and interpreted for use. There's a major trade off to this, being that once it has been applied the rigging process simplifies a lot. Moving onto the positive aspects of the paper and the methodology, articulation curves primarily focus on realistic movements. The paper indicates clear examples of how it's been used for the entire body of the meshes as well as faces and fingers showing the methods versatility and applicability for animators.

Algorithm 1 Algorithm adapted from and used in Goes et al. (2022)

Input : Mesh M, Skeleton S, Profile Curves PC

Output: Deformed Mesh M'

Initialize M' as a copy of M

for *each vertex v in* M' **do**

  Compute the influence of skeleton S on v based on vertex weights Compute the deformation of v using the profile curves C Update the position of v in M'

end

return M'

2.2. Neural Blend Shapes

The most common technique used to create realistic human-like muscles is linear blend skinning. By multiplying the mesh vertices of the bones transformations a muscle-like deformation can be achieved to show a flex Magnenat-Thalmann et al. (1988). Whilst this method can produce realistic results there are other techniques that look to bettering the process' efficiency. An alternative technique is the work of Peizhuo Li et al and their work on articulated blend shapes Li et al. (2021).

In summary, their technique involves using a neural network in order to predict blend shape changes on the mesh. These changes are linked to the difference in bone angles of the armature. This neural network is essentially trained using a function that measures the difference between the predicted blend shapes and the actual results that are given, this difference is the calculated factor that provides the realistic blend shape change. An overview of the method can be seen in 1.

A key factor that paints this technique in a positive light is the fact that it can be applied to animal or fantasy-like meshes as well as human ones. Peizhuo Li et al. describe clearly how the technique works and comparative results to other methods. Their results indicate higher flexibility, accuracy and realistic results compared to other methodologies. On the contrary, the paper doesn't take into consideration that this process is more resource intensive than linear blend skinning. Lower end software and hardware alongside the combination of other processes implemented questions the feasibility of using the methodology in terms of producing fast results. Generally, it's not a technique that is effective as articulation curves nor linear blend skinning performance wise. The

methodology requires training data which would elongate the process or may not be available at all. From the programming perspective its control is limited and would not be easy to use. Applying either linear blend skinning nor neural blend shapes would not account for certain realism aspects such as wrinkles and bulges.

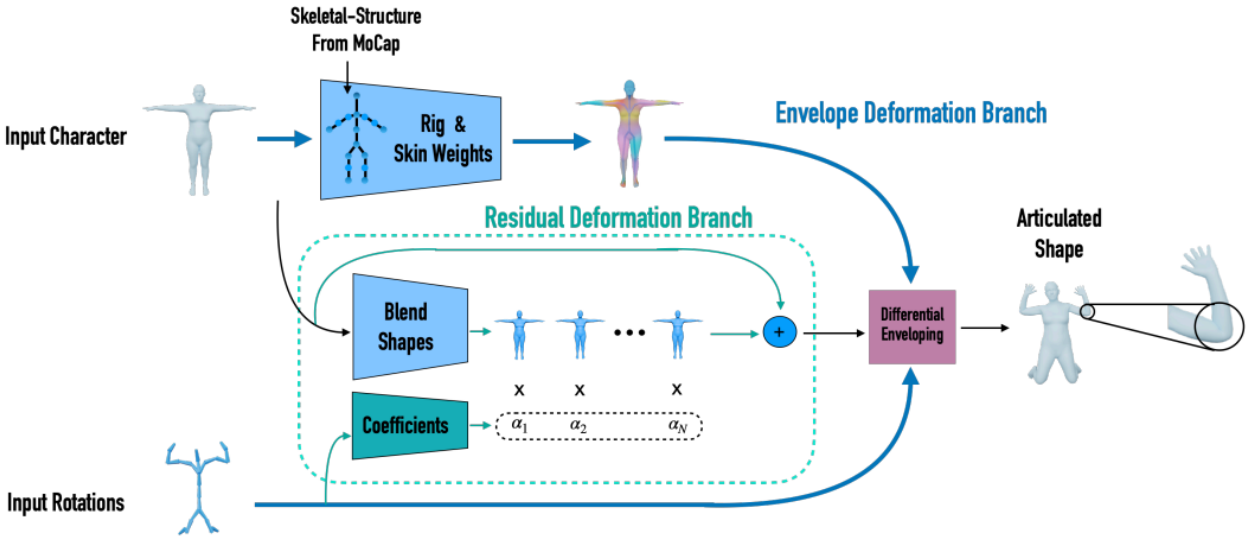

Figure 1: Image from Li et al. (2021), shows an character model using envelop branch to predict corresponding skinning and rigging parameters. Residual deformation branch predicts deformation changes based on expected and actual blend shapes conditioned on the input of join rotations.

2.3. Larger Mesh Deformations

Tackling larger mesh deformations isn't an issue looked at by neural blend shapes and so a more robust technique in the application of the volumetric graph Laplacian is required. The technique builds on surface based methods, such as Poisson mesh but looks to resolving errors that come with it such as a pinching in meshes occurring frequently during deformations Yu et al. (2004). It looks to represent meshes through graphs in which each vertex is a node, the volumetric graph Laplacian aspect then uses the nodes

current position and that of the surrounding nodes in order to predict and compute a new position Zhou et al. (2005). A visual understanding is depicted in 2.

The paper itself doesn't discuss the method implementation in great detail, further independent research is needed to figure out how to implement the process. In the same manner no real analysis on the performance of this method was provided. Based on further findings the method requires large processing to construct the volumetric graph and so this method may not be beneficial to those who don't have high end software or hardware. The method also assumes a specific mesh structure which may not be the case for all meshes and so it would not be applicable for all meshes. On the contrary, the paper does show comparative results and findings which indicate that the method is able to sustain details of the mesh accurately which deforming in large quantities unlike the Poisson method or neural blend shapes. The paper also indicates that this method is versatile when it comes to deformations which a large variety of tasks being able to be performed such as mesh skinning, mesh motion transfer etc. Overall, the technique could be applied to large deformations on meshes with the tetrahedral structure but for smaller scale deformations other techniques such as linear blend skinning, neural blend shapes or applying Laplacian graphs through a different form should be preferred.

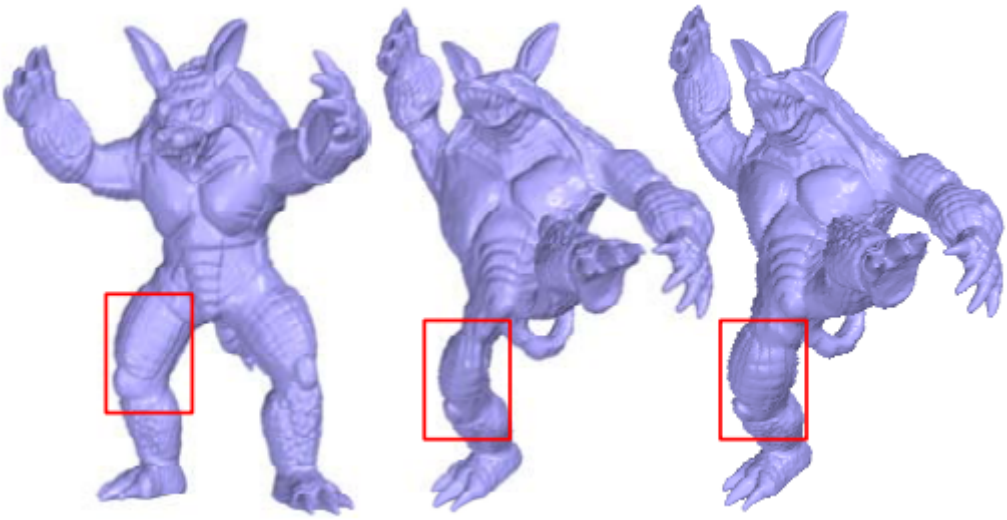

Figure 2: Image from Zhou et al. (2005). On the left there is the original mesh, in the middle is the mesh deformed using Poisson technique and on the right is the mesh deformed using the volumetric graph Laplacian technique.

2.4. Harmonic Skeletons

The skeleton of the rig is an extremely important component and can be constructed through various methods. Grégoire Aujay et al. present an algorithm known as harmonic skeleton which aims to generate skeleton structures for realistic characters Aujay et al. (2007). The process is broken down into several sections.

Graph Computation: The algorithm begins by selecting skeleton points for each bone, one endpoint is selected as the root vertex on the head and its value is set to 0. The distances between these points are then computed using either Laplace's equation or Dijkstra's algorithm, and the result is used as the definition of the harmonic function.

Graph Generation/Filtering A graph, specifically the Reeb graph, captures the structure of a character model and provides key information of this for the latter stages of an animation process including any symmetry detection and skeletal controls. The Reeb graph is computed using McLaughlin's algorithm. Graphs must then be filtered in order to recover the models initial symmetry, the main filtering includes weight re-assignation and weights below or above the required threshold are completely removed.

Harmonic Skeleton Once the harmonic graph is computed additional information including embedding of nodes is required to clean up the final produced harmonic skeleton.

The technique is clearly very complex with the application of three different algorithms being needed as well as specialized software or plugins that may not be available to individuals. This limits the probability of the technique being used in the first place. The technique can also be limited in handling more extreme deformation requirements or specific types of character meshes. On the other hand, the technique looks to produce the most realistic character animations possible and the implementation of this technique would produce very lifelike results. Upon application there is a wide range of flexibility with the use having control over the skeleton and deformation that can be applied with a further deformation technique to produce an extremely muscular movement. The paper doesn't specify any specificity in needing higher end hardware to implement it, however, considering the algorithms used and the process in general it is safe to say lower end hardware will struggle if the technique is implemented.

2.5. Bone Glow Weight Assignment

Bone glowing is a technique for weight assigning that looks to adapt the bone heat technique. The bone heat technique consists of vertices closest to each bone being initialized with surface temperatures in correlation with their distance to the bone to form a heat diffusion system on the mesh's surface. The algorithm seen in 2 is then implemented to form the bone heat technique, however, the technique itself is riddled with disadvantages including; slow performance, everything having to be done manually and limited weight influence. Rich Wareham et al. look to address these disadvantages through bone glow. Their enhanced method takes in vertices surrounding the bone and creates a heat map illumination as a result. The illumination takes into consideration the visibility, direction and transmission when creating the weighting of the bone Wareham and Lseby (2008).

Bone glow allows for a wider variant of deformation techniques that produce more realistic results in comparison to the bone heat method, as shown in **??**. The paper presents more figures as such that indicate lesser incorrect deformations through bone

glow. Those will less sophisticated tools could look towards bone glow as it's more time efficient and performs more efficiently than other methods. On the contrary, there are techniques involved that require extensive knowledge and correct use which means the technique may not be useful for those with lower expertise on the field. This factor alongside the paper not having clear instructions on how bone flow is implemented heavily limits the capability to both implement and use this technique. Bone heat also does not always have performance issues, with only a few being outlined, comparatively, applying bone glow alongside neural blend shapes or the volume graph Laplacian methods may be considered as more time consuming than a simple bone heat with linear blend skinning.

$$\Delta w_i + H(p_i - w_i) = 0 \tag{1}$$

2.: Equation and explanation from Wareham and Lseby (2008), where Δ is the surface Laplacian, H is a diagonal matrix with element H_{jj} being the heat contribution of the nearest bone

3. Methodology

The use of Blender as the primary building software was a decision based on inexperience with programming animation as a whole. As Blender already has a python API embedded python became the non negotiable programming language to use. Blender offers a variety of add-ons that can aid in the creation process, none must be assumed to have been used unless otherwise stated. General plans consist of how the main aspects of the rig were to be created and how methodology data would be applied to aid this, with a final overview of the rigs flow seen in 4.

3.1. Mesh Implementation

A mesh is a 3d object made up of vertices, edges and faces, it can be considered to the human skin for the sake of this project. There are two general methods for implementing a mesh onto a scene.

Creating a mesh: Meshes can be created from scratch and defined through vertices, edges and faces to determine its appearance as seen in 3. Further use of modeling tools could aid this process.

Importing a mesh: Existing meshes can be obtained from another source and can be imported to Blender. The only prerequisite is that the exported mesh is under a file format which is compatible with Blender such as .obj, .fbx, .stl, .ply and .dae.

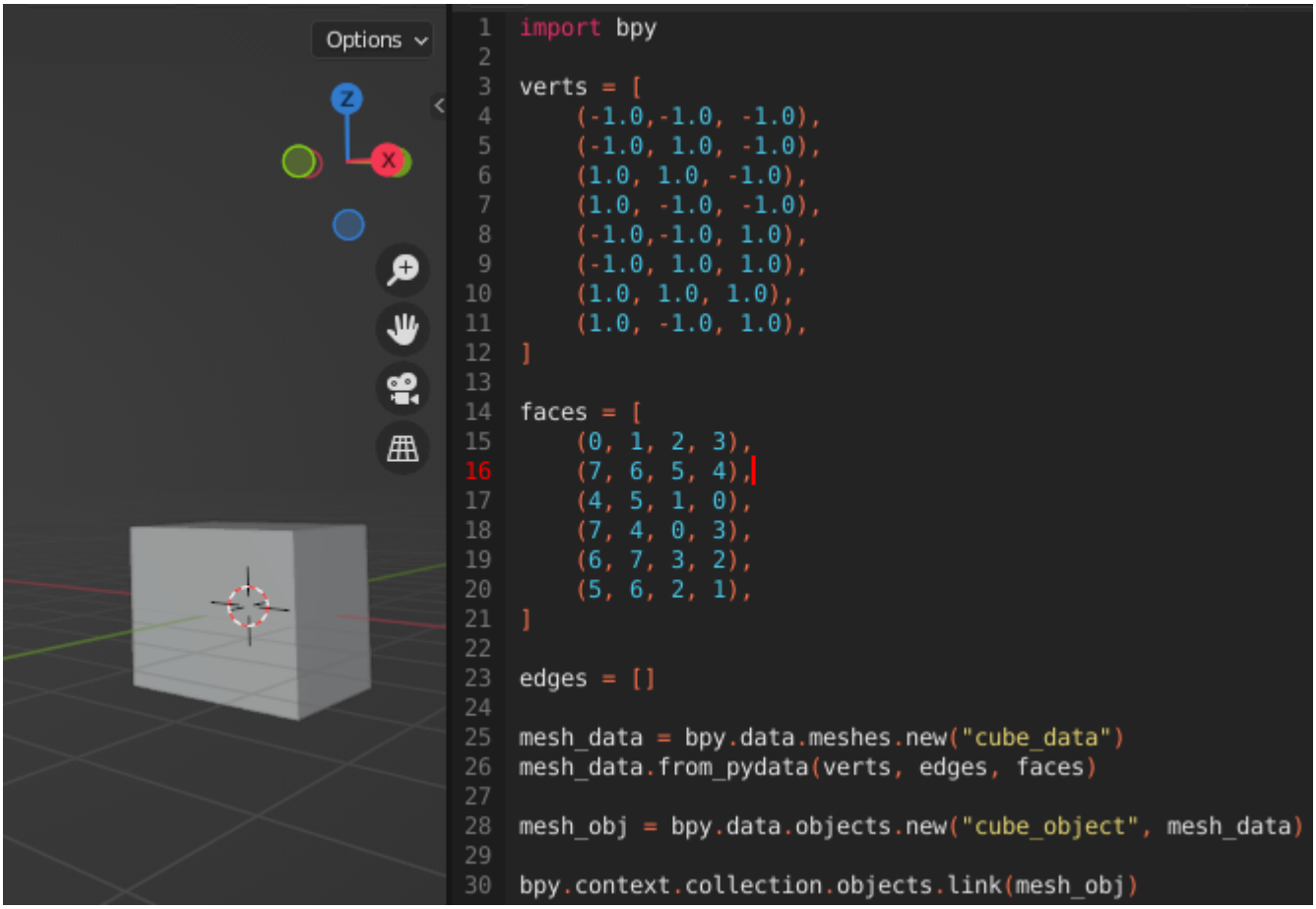

Figure 3: Code snippet and output showing a cube mesh being created from scratch, vertices and faces are manually declared.

3.1.1. Creating or Importing Meshes

A simpler comparative view of these methods can be seen in 1.

Some factors to consider creating over importing would be the flexibility aspect, since the mesh is in the creators hand it could be specifically designed for its intended purpose and can be modified throughout the entirety of the project. This flexibility could produce better integration results with other aspects of the project, a better understanding of vertex alignment allows specific coordination with bones that leads to smoother deformations, etc. On the other hand, creating meshes would require the specification of every vertex. In terms of creating a human-like mesh, importing

pre-existing meshes would be much more time efficient. Users would also be able to have a wider variety of choices when importing meshes and be able to use their own meshes of choice to gain the results they most desired. Imported meshes could be professionally designed which wouldn't be an option to create from scratch given the time and the artistic aspect

Table 1: Comparative table of creating meshes against importing meshes with positives and negatives outlined.

	Creating a Mesh	**Importing a Mesh**
Design Control	Gives you more control of vertex alignment and connection.	No real control over how many vertices there are and where they're positioned.
Efficiency	Very time consuming as each vertex must be defined and the project is for human-like meshes.	Fast process.
File Size	No external files required at all.	File size depends on the mesh, many would need to be given for testing purposes.
Compatibility	No compatibility issues.	Compatibility issues could fall under how the imported meshes were created and the type of file that they are.
Project Suitability	Suitable to the programmers needs and perhaps the users.	Suitable for everyone.
Complexity	Becomes more complex considering the human-like structure.	Not particularly complex.

3.1.2. Mesh Decision

The final decision is to implement meshes as implementing meshes rather than creating them. Users would be able to import online meshes that they seem fit and are given a larger variety pool size to work with, making it more compatible. Consequently, more time would be given for testing out other aspects of the project and perhaps further enhance its functionality. No additional research in coherence with other methodologies

analyzed impacted this decision.

3.2. Bone Implementation

A bone is a modelling tool users would use to animate, it defines the skeletal structure of the character. Bones consist of several key elements including; its head and tail, its relation to other bones and its properties, such as constraints. Alike mesh implementation there are two general methods used to implement a bone and link it to the mesh.

Bone creation methodology: Bone creation either involves a simple declaration or the application of advanced algorithm as that discussed in the 2.4 section. Simple creation work similar to meshes with the heads and tails manually being declared in the scene with any additional properties being declared. The use of harmonic skeletons would require algorithms to be initialized in the code and computation of the Reeb graph, a much more involved process.

Rigify: Blender offers rigify as an add-on that can generate rigs. A simplified representation of the desired rig structure, known as a metarig, is created and can be adjusted before the final rig is generated. Properties such as predefined constrains are already implemented and again have the option to be adjusted for personal need.

3.2.1. Creating or Rigify

A simpler comparative view of these methods can be seen in 2.

Rigify offers predefined rig types with deformations and constraints already set up, at the very most adjustments will have to be made but it's a more efficient process than creating properties from scratch. Rigify also offers higher customization than regular creation with control widgets that can manipulate the rig and its components. It would be a simpler process to implement compared to harmonic skeletons. Harmonic skeletons will however, provide more realistic results. More control is offered for each individual bones detail, without the application of Rigify it's unclear how much control there would be on bones individually and so linking the armatures bones with meshes vertices would become an easier process.

Table 2: Comparative table of creating against importing a mesh with positives and negatives outlined.

	Basic	**Harmonic**	**Rigify**
Design Control	Gives you more control of bone positioning and connection.	Provides control on a complex level.	Offers metarig which you can control to some extent.
Efficiency	Fairly time efficient on a sole rig.	Very time consuming.	Fairly time efficient on multiple armature.
Compatibility	No compatibility issues, can work with imported meshes.	May not always work, dependant on mesh type.	May not produce expected armature.
Flexibility	One dimensional, users can only work on one armature.	One dimensional considering time needed.	Very flexible, can produce multiple armatures for multiple meshes
Complexity	Not complex.	Very complex.	Fairly complex for first time.

3.2.2. Automatic Weights or Bone Glow

With deformation considerations in mind weights are applied to bones. Weights determine how much of a mesh is affected by that particular bone and can be controlled through weight painting. Application off individual weight painting is not a consideration due to the uncertainty and variety of vertices that are imported providing no real method of assigning individual vertices to bones. This leaves either assigning Blenders automatic weight system to the rig or using the bone glow methodology discussed in 2.5 to create this link. Using bone glow would provide more realistic deformations due to the techniques it implements and automatic weights possibly causing vertex groups to overlap one another. It further takes factors such as visibility, direction and transmission into account when creating these weights which would result in fewer incorrect or unwanted deformations. Automatic weights is a more simplistic and time efficient approach with only a few statements of declaring needed in contrast to bone glows algorithm. This, in turn, leads to faster performance if automatic weights are used.

3.2.3. Bone Decision

Driven blend shapes with automatic weights is the final decision for bone implementation. The simplicity aspect and guarantee that the program will run efficiently played a major reasoning. Harmonic skeletons and bone glowing would provide more realistic results but are not guaranteed to align with imported meshes and deformation methodologies. Bones will additionally have constraints applied that limits their rotation beyond specific angles to improve realism, these individual constrains may not have been applicable or editable if a tool such as Rigify was to be implemented.

3.3. Deformations

Deformations are the aspect of the project that give meshes further life. There's a huge catalogue of tools for creating deformations, with driven blend shapes being the most common and appropriate technique.

3.3.1. Blend Shapes/Shape Keys

Blend shapes (also known as shape keys) are tool techniques popular for general mesh deformations. Blend shapes require specific properties to be set up with further methodologies, as discussed in 2, being able to be used alongside them to provide the best results. Generally blend shapes consist of a difference between two vertex points with a driver that controls how the difference can be interchanged. From a programming perspective not many other alternatives were presented, lattice deformation and mesh sculpting are two other deformation techniques used however they tend to be done from an artistic viewpoint.

3.3.2. Driving Techniques

No matter the methodology, blend shapes require a variation of driving to produce any deformation. Drivers (discussed further in the **??**ection) work in many forms, including:

- Bone-based drivers that produce deformations based on the direction in which a bone changes.
- External script drivers.

- GUI based drivers such as a slider or a button.
- Shape-driven drivers, using part of one mesh deformation to drive another.

3.3.3. Linear blend skinning

Based off my the analyzed methodology papers 2, I've decided to used linear blend skinning. Linear blend skinning is straightforward to implement compared to the use of volumetric graph Laplacians or neural blend shapes, all that would be required is to group up vertices and multiply them. Its ease of use extends to how compatible it would be, a function with the technique could be called by shape keys and their drivers, producing the final result. Other methodologies would require graphs or algorithms to be set up before they can be implemented, there isn't a guarantee to how well the neural blend shape algorithm will work with a bone driver etc. From the perspective of the target users, quick response times would be beneficial if not required and so linear blend skinning would be more efficient performance wise compared to other methodologies and efficiency can always be further improved by reducing the number of driving influences. The only downside the implementation of linear blend skinning is that other techniques may produce more realistic results, needless to say effective results can still be produced using this methodology regardless.

3.3.4. Deformation Decision

With all the comparative analysis in in mind the use of blend shapes with linear blend skinning is a given in order to produce deformations. Based on the extent of the project progression different driver techniques can be used, however, bones will be used as the main driving factor. Any driving for the weight changing aspect of the mesh will be done through a slider.

3.4. Program Flow Chart

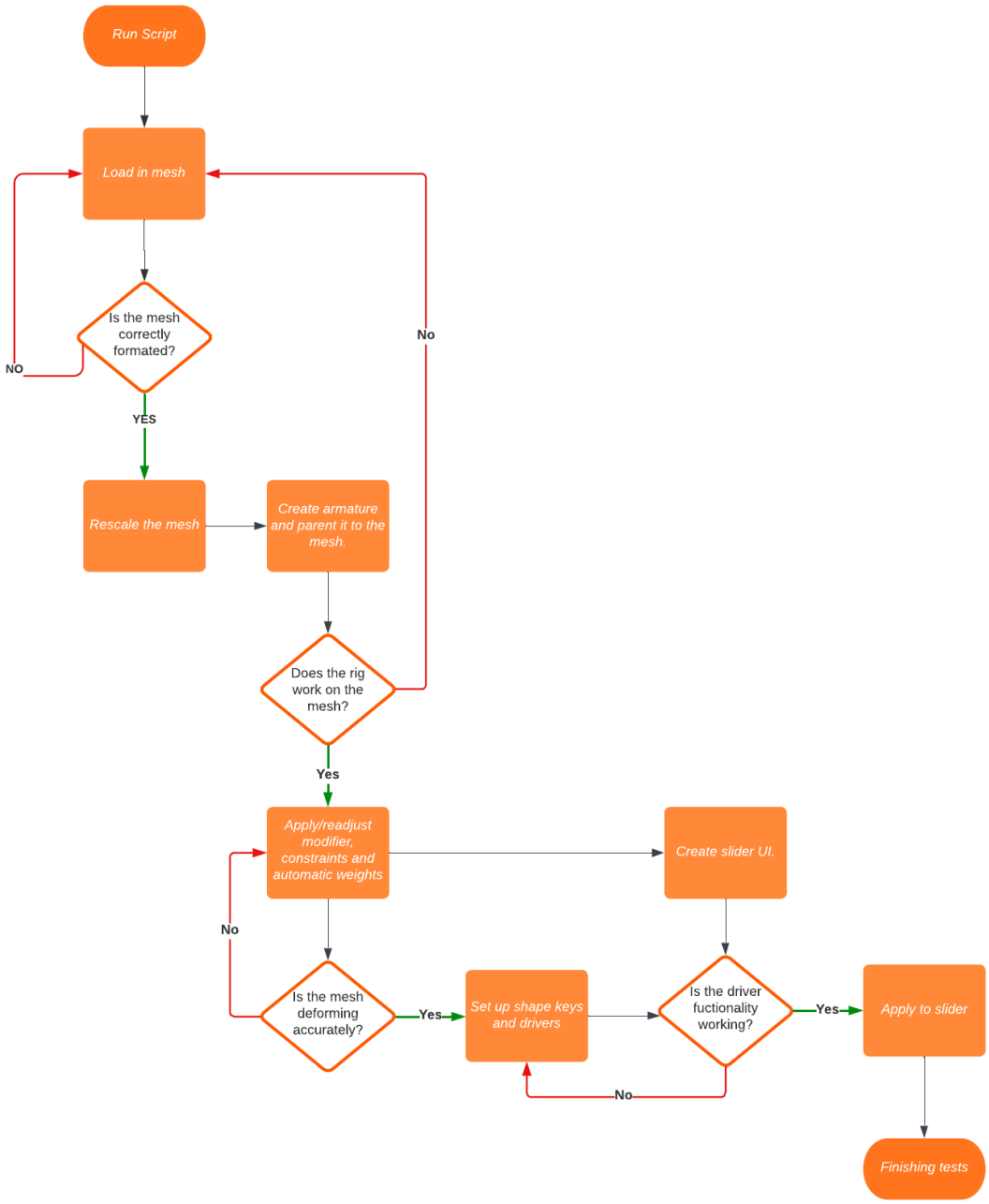

Figure 4: Flowchart view of how project is expected to be constructed and/or programmed. White decision sections indicate main testing.

4. Implementation and Evaluation

Considering this is a computer science project all implementation processes are to be assumed to be completed through the use of Blenders Python API, unless otherwise stated.

4.1. Clearing the Scene

Blenders scenes provide the environments in which users can create and design their desired objects. Every time a script is run the scene stores all components (as seen in 5) and so it's essential to clear out the scenes to ensure older project components, such as older meshes, don't interfere with newer meshes. In addition, clearing out scenes helps Blender free up memory, which is crucial when working on larger, more complex projects. Failing to do so will result in overlapping or intersecting meshes can cause errors or crashes, further discussed in 4.9. Clearing any scene must be done at the beginning of the script with all Blender components being looped through and being removed.

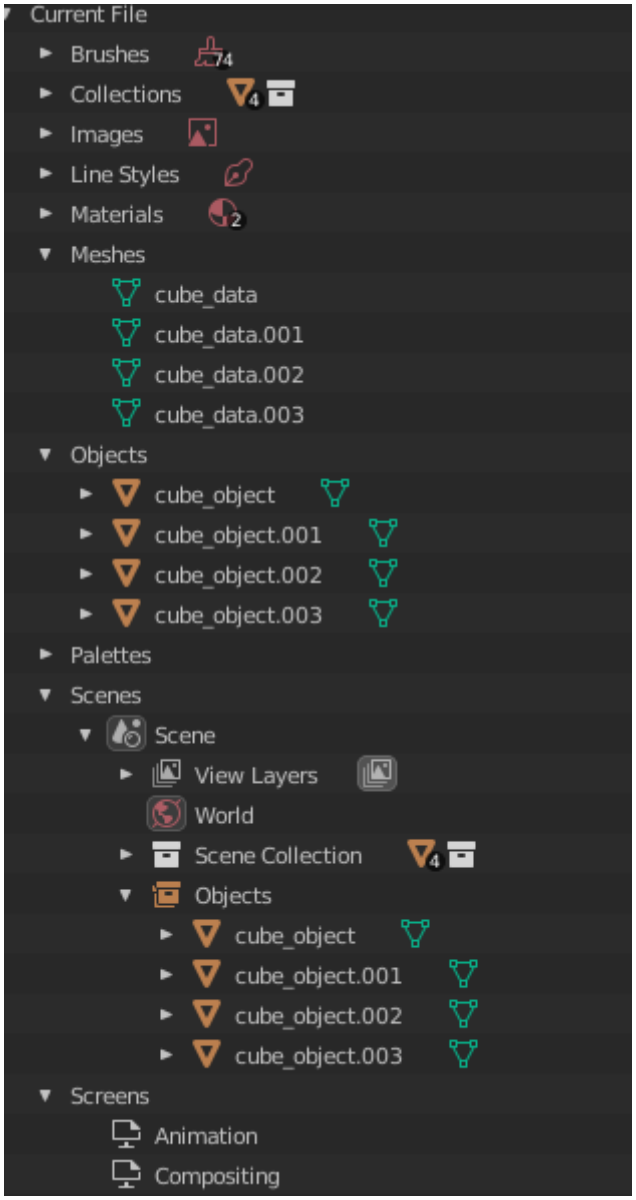

Figure 5: Screenshot that shows how running the same script without clearing the scene piles components up.

4.2. Mesh

The mesh importation decision is based on the discussion in the 3.1 section. Before producing any code models had to be gathered models off websites and could use as the base for the entirety of the project. These models have to come in specific file formats

that Blender can work with, including .obj and .fbx. In terms of the code, files were accessed through the specification of file paths. File paths are entirely user specific and so each individual will have to change this path according to where they saved their files. If this is done correctly the imported mesh can be stored in a variable and called to the scene. It's key to note only humanoid-like meshes should be downloaded and it's advised for basic ones to be used as more complex ones can result in issues as described within the 4.9.1 section.

4.2.1. Mesh Scaling

Once the mesh has been imported the process of scaling comes into play. Scaling is required to ensure all the imported object's dimensions have consistent height and proportions regardless of how they were created. This was done so users don't have to constantly scale imported meshes themselves to an appropriate height and all the issues that surround this, discussed further in 4.9.1. This process requires a basic mathematical algorithm, shown in A. With the final outcome in mind the desired height is created and used for all imported meshes, it's then divided by the current height that the meshes are when originally imported. This results in a scale factor that is used as a means to adjust the object.

4.3. Armature

Armatures are skeletal structures that define bones of a 3D object and are required to allow individuals to create bone constraints that control the movement of a mesh. Before defining the armature, the imported mesh must be set as the active object so that it can be linked to the armature. From there simple naming definitions for the armature and the rig are all that is required, although linking it back to the scene and running the program will not do anything as of now as the armature doesn't contain bones.

4.4. Bones

In order to ensure accurate rigging and movement of a mesh bones are created in the armature and assigned to specific positions. Bones are defined by a list of tuples with key information about each part of the bone (as seen in B), this includes:

- The bone name itself.

- The bone unique identifier in the form of a number.
- The identifier of bones parent (e.g. a neck bones parent would be the spine bone).
- The x, y and z coordinates of each bone head and bone tail.

4.4.1. Bone Parenting

Upon creating the bones it's vital to establish their parent to child relationship so they can move and rotate together properly. This is achieved through creating a key bone that acts as a top-level parent to all the others and as a base for the whole armature. From there, all remaining bones are looped through with their parents being assigned based on their index identifier within the tuple (in my case, index 'p'). This process is shown in C.

4.4.2. Bone Constraints

Bone constraints allow a certain limitation to bones in order to apply the feel of realism. They do so by setting limitations for the x, y and z axis that are defined as minimum and maximum angles. In order for bone constraints to work properly as a group they must work within a local environment. Doing so means bone limitations are relative to the bone's orientation and avoid inaccurate or no rotations, further discussed in the 4.9.3 section.

4.5. Modifiers

In order to allow the meshes to be deformed by changes to the bones we use modifiers. To allow this deformation, before setting the modifier, it's required to set the armature to be the parent of the mesh. From here the modifier is defined and assigned to the newly parented armature which now allows for bones to manipulate the mesh.

4.5.1. Vertex Groups

Assigning vertex groups to the modifier will allow users to perform deformations within the mesh specific to a specific bone. Vertex groups combine vertices around a specific bone together, with the groups being named in accordance to the bone.

4.6. Shape Keys

Shape keys are required to create muscular-like deformations, as discussed in the 3.3.1 section. The base key is created first followed by additional keys assorted by vertex groups. These additional keys had basic number identifiers whilst the weight shape key had a specific name as its properties are adjusted several times throughout the code. After their initialization the bones associated vertex group is looped through in order to be able to modify the vertices positions (as seen in E). The former if statement verifies the existence of the chosen vertex group whilst the latter ensures that the specified group vertices are the only ones being affected. The final decision on the vertices multiplication and the corresponding axis they would multiply by was based on testing results in the 4.9.5 section. The majority of vertices were multiplied by 1 to 2 in the x and/or z axis. Whilst the shape keys have vertex change definitions no changes are visible until there is a driver connected.

4.7. Drivers

Drivers act as the connector between our shape keys and the armature or a slider for a later use. Drivers are created individually based on each shape key and follow a very specific requisite in order to function properly:

- All shape keys are modified in the form of function curves.
- All driver musts be set to 'SCRIPTED', this allows them to be run via Blender's python API.
- All driver expressions must be set to 'var' to reference a variable within the script, which is the bones for this case.
- Variable types must be defined as 'TRANSFORM' which indicates that a change in the bone will act as the trigger.
- The individual target bone is then chose as must align with the shape key you intend to make the bone drive.
- The transform type refers to the rotation angle that is used by the bone. The driver will only trigger on this specific rotation. In my case selecting specific angles was

a result of my testing, with the majority of my bones being driven by x or y rotations.

- The data path for each bone must be set to 'rotation euler[0]' to ensure the driving appropriately occurs as a result of a rotation.
- Variable transform space must be set to 'LOCAL SPACE' to ensure changes are always locally relative to each bone and avoid incorrect driving issues.

When all this code is put together we should have working shape key drivers as depicted by 6.

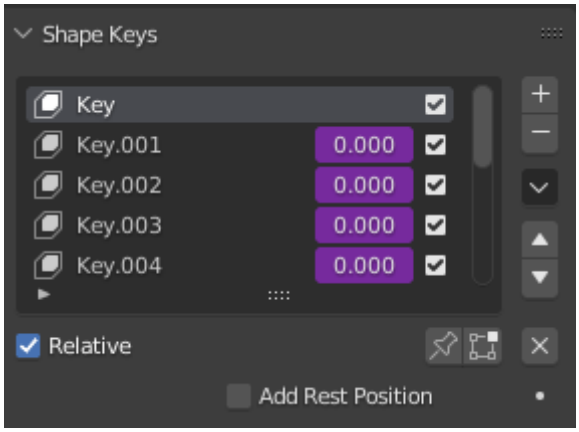

Figure 6: Screenshot that depicts all shape keys, the purple outline indicates drivers are present, upon moving the rig specific values change according to what bone was moved.

4.8. Slider

Whilst bone rotations drive changes to our meshes muscles, slider changes drive changes to the meshes weight. A slider involves creating a property group (in my case named 'MyTool') within the interface that holds onto the slider values, the slider name, including a default value, the sliders minimum value and the sliders maximum value. In addition the slider is set to initialize an update callback function whenever the value of the slider is changed.

4.8.1. Slider Design

Blender allows for sliders to be created and implemented via panels, and so it's required that the panel contains specific attributes including an identifier, a label, a viewport and a category. Within this panel a draw function is created and is intended to set up the interface itself. Two key attributes lie here, being the reference back to the 'MyTool' property group in order to access the slider's value and the creation of a slider label. A correctly drawn panel and driver displays as the following 7.

4.8.2. Slider Driver

The slider driver acts as under an 'update callback' function which is called whenever the slider value changes. The functions searches for the 'Weight' shape key before setting the value found within the key to be the value of the driver. Nothing returning means no 'Weight' shape key was created. This driver differs from our bone drivers as it sets the variable target to the slider in the scene, containing a data path reference to the 'MyTool' property group rather than a bones rotation.

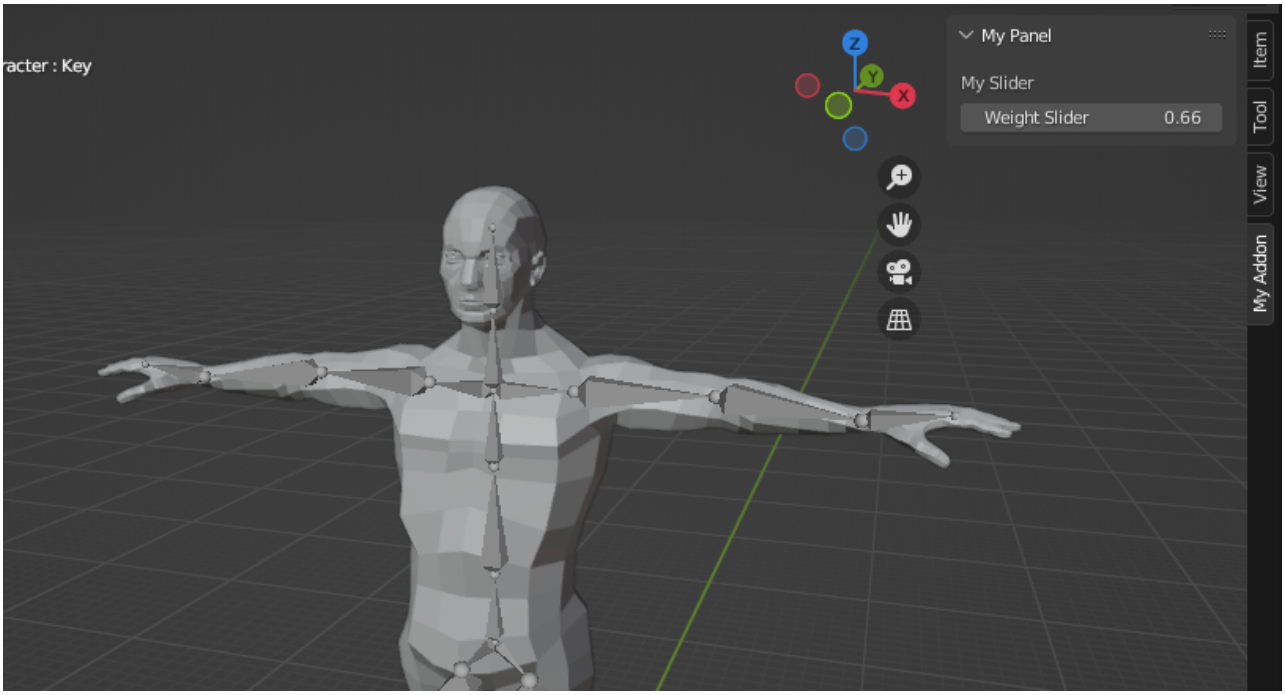

Figure 7: Screenshot of weight changing panel. The mesh is also depicted as slightly fatter as its value on the slider has been changed.

4.9. Testing and Evaluation

Before delving into testing individual key components of the project it's important to look towards whether the program is performing as expected on the forefront. The program is run at a good speed with the single click of a button being all that is required and the scene updating within a maximum of a few seconds. This factor may be dependant on the individuals hardware, software and storage space. General speed testing was simply done over code being applied, if it slowed process down the code would be removed and a new solution would be devised. Overall in terms of efficiency an consistency with my objectives the program runs well with a few exceptions to be discussed.

4.9.1. Mesh Importation

The mesh is the first stepping stone in the rigs creation process and so it's key meshes are imported and scaled correctly so that users can freely use them with minimal to no changes. Different file types including .obj, and .fbx were tested and the majority of meshes were present in the scene, however, the scale, positioning and movements did not account for all the meshes even with the code shown in A. The variation of which factor worked for meshes varied heavily, some meshes did not scale to size whilst others did but didn't deform with the rigs positioning. Unfortunately, there is no simple to solution to this problem as occurs as a result of how the meshes were created rather, if an individual wanted to use a specific model they found they would have to redefine it before exporting and importing again. Despite this problem there are still a variation of meshes that do scale and position accurately proving the process to work efficiently.

4.9.2. Rig Positioning

To maintain an effective rig bone positioning and scaling must follow a human-like structure. The root bone was chosen to correspond to the pelvis as I deemed it to be the most accurate correspondence within the human body. On the forefront testing the bone scaling and positioning meant ensuring the coordinates for each bone were correctly stated. Bones above the pelvis would have their tails facing upwards whilst bones below the pelvis would have their tails facing downwards. Due to variety in difference between imported mesh arms, two forms of rig arm directions were created and would accommodate the majority of the meshes, with the arms either slating downwards or straight to the side in a T-pose position, as shown in 8. The T-pose rig to

was more accurate when applied to meshes that didn't initially import in the form of a T-pose. These meshes would still provide movements across the arm despite the rig not aligning with the mesh, depicted in 9, how accurate these movements were varied heavily on the positioning of the movement. Users can still chose to align the T-pose rig with their mesh if it's slightly off but would have to do so manually. The rig proves flexible, adjusting its position manually to a fair distance from the mesh still affects the mesh but how it does so is dependant on the position, this is again referred to in 9. This flexibility is unfortunately limited, finger and toe bones did not accurately deform all meshes and so had to be scraped. In addition some meshes proved to not be affected by the rig at all, these tended to be more complex meshes that had many components to them.

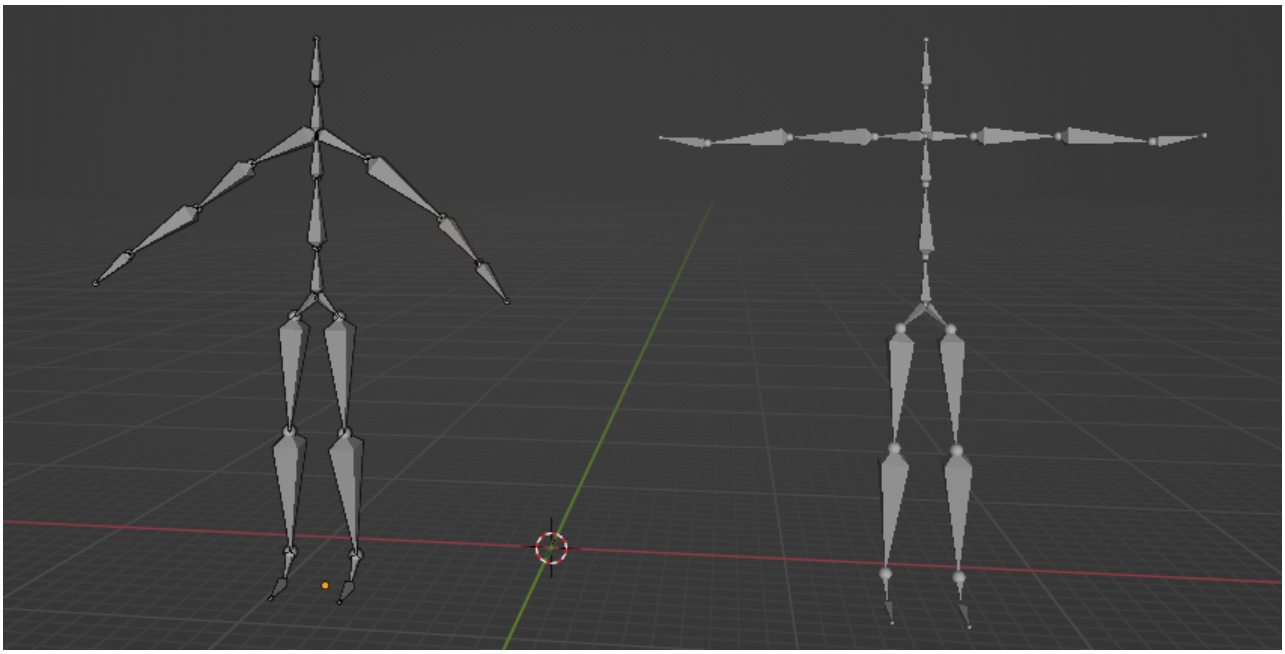

Figure 8: Screenshot of the two forms of rigs created.

4.9.3. Rig Constraints

In addition to positioning, rig constraints were tested to ensure that the meshes movement aligned to that of the human body. Each bone and its possible rotations would individually be tested before grouping them together to assume fluid movement across

a whole section of the mesh, such as a whole arm bending accurately. In order to grab the realism aspect I used myself as a reference point, observing my movements before applying them to the rig. Overall, the constraints worked accurately and provided realistic movements as depicted through the examples seen in 10 and 11. The process was however tedious, with the repetition of code needing to be used for each individual bone despite the minor differences in axis constraints. Additionally, due to the fact that it's a singular rig it wouldn't be able to be considered to rig models that are more flexible in nature, such as gymnasts. With consideration back to the lack of finger or toe bones positioning, no constraints were applied to these areas of the mesh.

4.9.4. Driver Testing

Referring back to the 4.7 section the first aspect of driver testing was to ensure that drivers were set up correctly. A working driver is shown through a purple highlighted shape key as seen in 8, whereas driving values and visible changes can only be seen under the assumption that driver properties are correctly configured and the shape key itself performs an action. Driver functionality was tested by changing the target bone rotation type. This was a manual trial and error process in which the bone rotation angle was chose based on a realistic factor. On the whole, once each driver was set up correctly, working as intended. On the contrary the process was tedious, once again a lot of code repetition was necessary due to the fact that each shape key would be assigned an individual driver. Shape keys could theoretically have two drivers, however, they would have to be assigned separately as doing so together meant they didn't work as intended and even doing so separately resulted in inaccurate results and so the idea had to be scraped.

The slider's driver also had to be tested and configured in the same manner. Rather than focusing on a bones rotation we singled out the panel interface, whilst this made the code itself less tedious to write linking the weight shape key and driver to the slider required more effort. Due to the limited rotations and inaccurate mesh deformations that multiple drivers cause it's fair to say that the slider driver works more efficiently and accurately than the bone drivers.

4.9.5. Mesh Deformations

Testing mesh deformations to ensure they resemble human-like muscle flexes was done simultaneously to driver testing. Alike bone constraints referencing was done through myself looking at muscle movements in a mirror and general human anatomy knowledge. In order to made sure realistic proportions were created different multiplication factors for each vertex group were tested . In the same manner for human-like angles the angle of the multiplication in correspondence alongside the drivers rotation was tested. A major downside for the mesh deformations are the vertex groups not being precise enough. Bones within the project are assigned with automatic weights and so in many instances there are more vertices belonging to a particular shape key than what was initially anticipated, these led to overlapping or under lapping deformations when paired. Inaccurate general deformations were also present when meshes did not align to the t-pose aspect of the rig. As a result of this many muscle shape deformation values and positions originally planned were either limited or removed. This factor also made the process of accurately deforming meshes longer as testing for vertex groups had to be done individually followed by several done together to ensure mesh deformations looked natural. Initially the weight slider was to simply increase the entirety of the mesh alongside all axis. This approach was later changed as it didn't seem realistic enough. I split up my weight key across different vertex groups and so the same testing process was performed.

5. Conclusion and Future Work

Through the conducted research and the application of various methodologies a successful 3D animation rig was produced. Research provided valuable insight into a variation of techniques and algorithms that can generally be used in rigging, whether they were actually used was a matter of time and skill considerations. In relation to the devised MoSCoW in the 1 section, the final outcome consists of an armature that automatically connects to imported meshes with muscle like and deformations with the particular use of linear blend skinning research adapted into driven blend shapes aiding heavily in the deformation process. I further included constraints to prevent unrealistic motions and a weight deformation aspect driven through a slider.

Across the development of my project, several aspects proved effective in its completion. The link between the importation of meshes and an armature was the vital driving point of my project and worked fluidly allowing other aspects such as deformations to be produced on top of the rig which in turn created realistic character movements. Adapting two forms of shape key drivers in bone rotations and a slider was another successful implementation, with both deformations functioning as intended, more so deformations can be produced together on top of one another.

On the contrary, there are aspects that could be done differently. Although deformations are present their level of realism didn't correspond to the initial expectation. Alternative techniques to linear blend skinning were not implemented due to expertise within the field, however, there is a strong argument that bone glowing in particular wouldn't have needed drastic knowledge whilst providing better results than automatic weights and linear blend skinning did. Mesh importation makes the armature system seem very one dimensional as a result of its positioning being manually declared. Whilst these armature is easy to manipulate perhaps a better outcome would've been to use Rigify to give users the same flexibility as meshes provide.

In conclusion, a 3D animation rig with basic deformation aspects was successfully developed in this project. However, there is still room for improvement, especially in terms of the realism of the deformations. Bringing this project forward, I would personally focus on pushing the flexibility of the rig rather than the realism aspect.

With the implementation of Rigify the rig could be accommodated to animal meshes alongside human-like ones and cartoon-like deformations, such as an arm twisting multiple times, could be implemented through accurate deformation weighing to offer independent animators a larger working environment with the rig.

References

Aujay, G., Hetroy, F., Lazarus, F., and Depraz, C. (2007). Harmonic skeleton for realistic character animation.

Goes, F. D., Sheffler, W., and Fleischer, K. (2022). Character articulation through profile curves.

Li, P., Aberman, K., Hanocka, R., Lio, L., Sorkie-Hornung, O., and Chen, B. (2021). Learning skeletal articulations with neural blend shapes.

Magnenat-Thalmann, N., Laperrire, R., and Thalmann, D. (1988). Joint dependent local deformations for hand animation and object grasping.

Marey, J. (1873). *La machine animale: locomotion terrestre et aérienne.* Dover Publications.

Wareham, R. and Lseby, J. (2008). Bone glow: An improved method for the assignment of weights for mesh deformation.

Yu, Y., Shou, K., Xu, D., Shi, X., Bao, H., Guo, B., and Shum, H.-Y. (2004). Mesh editing with poisson-based grandient filed manipulation.

Zhou, K., ad John Snyder, J. H., Liu, X., Bao, H., Guo, B., and Shum, H.-Y. (2005). Large mesh deformation sing the volumetric graph laplacian.

A. Code for scaling algorithm

```
desired_height = 25

active_obj = bpy.context.selected_objects[0]
current_height = active_obj.dimensions[1]
scale_factor = desired_height / current_height
active_obj.location = (0, 0, 0)
active_obj.scale = (scale_factor, scale_factor, scale_factor)
```

B. Code for bone properties

```
BONES = (
    ("root", 0, None, 0, -1, 13, 0, -1, 15),
    ("spine", 1, 0, 0, -1, 15, 0, -1, 18),
    ("chest", 2, 1, 0, -1, 18, 0, -1, 20),
    ("neck", 3, 2, 0, -1, 20, 0, -1, 22),
    ("head", 4, 3, 0, -1, 22, 0, -1, 24),
    ("rshoulder", 5, 2, 0, -1, 20, 2, -0.5, 20),
    ("rbicep", 6, 5, 2, -0.5, 20, 5.5, -0, 20),
    ("rforearm", 7, 6, 5.5, -0, 20, 9, -0.5, 19.75),
    ("rhand", 8, 7, 9, -0.5, 19.75, 11, -0.5, 20),
    ("lshoulder", 9, 2, 0, -1, 20, -2, -0.5, 20),
    ("lbicep", 10, 9, -2, -0.5, 20, -5.5, -0, 20),
    ("lforearm", 11, 10, -5.5, -0, 20, -9, -0.5, 19.75),
    ("lhand", 12, 11, -9, -0.5, 19.75, -11, -0.5, 20),
    ("rhip", 13, 0, 0, -1, 13, 1, -1, 12),
    ("rthigh", 14, 13, 1, -1, 12, 1.25, -1, 7),
    ("rcalf", 15, 14, 1.25, -1, 7, 1.5, -0.25, 1.5),
    ("rfoot", 16, 15, 1.5, -0.25, 1.5, 1.5, -1, 0.5),
    ("lhip", 17, 0, 0, -1, 13, -1, -1, 12),
    ("lthigh", 18, 17, -1, -1, 12, -1.25, -1, 7),
    ("lcalf", 19, 18, -1.25, -1, 7, -1.5, -0.25, 1.5),
    ("lfoot", 20, 19, -1.5, -0.25, 1.5, -1.5, -1, 0.5),
    ("rtoes", 21, 16, 1.5, -1, 0.5, 1.5, -2.5, 0.25),
    ("ltoes", 22, 20, -1.5, -1, 0.5, -1.5, -2.5, 0.25),
)
```

C. Code for bone parenting

```
for bn, i, p, hx, hy, hz, tx, ty, tz in BONES:
    bone = edit_bones.new(bn)
    bone.head = (hx, hy, hz)
    bone.tail = (tx, ty, tz)

    # Sets the bone parent to "root" and its children are all other bones.
    if p is not None:
        parent = BONES[p][0]
        bone.parent = edit_bones[parent]
```

D. Code for vertex loop

```
# Spine Muscles
spine_group = active_obj.vertex_groups.get("spine")
if spine_group is not None:
    for i, v in enumerate(active_obj.data.vertices):
        if spine_group.index in [g.group for g in v.groups]:
            key1.data[i].co.z *= 1.15
            keyw.data[i].co.z *= 1.25
```

E. Testing screenshots

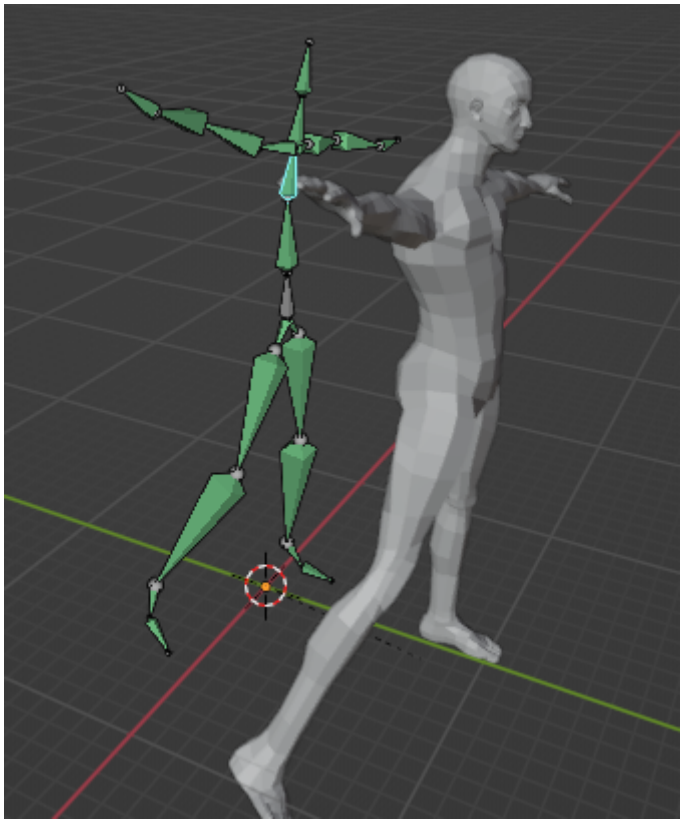

Figure 9: Screenshot depicts the mesh still being deformed even when not physically present with the armature.

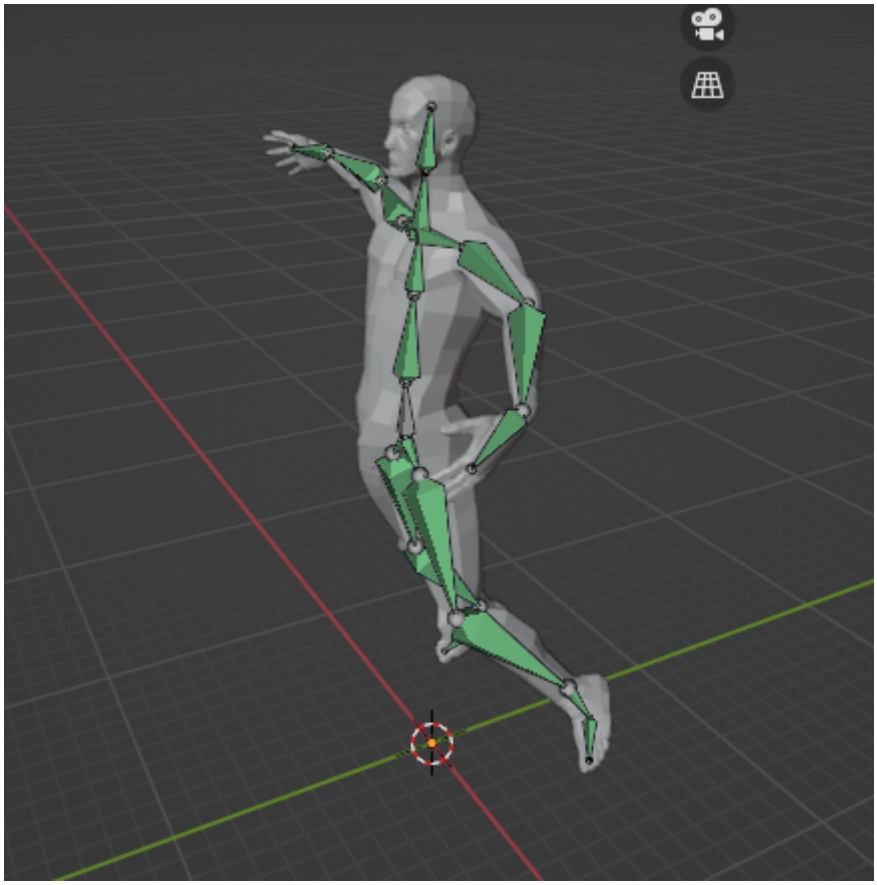

Figure 10: Screenshot depicts a realistic mesh due to constraints added.

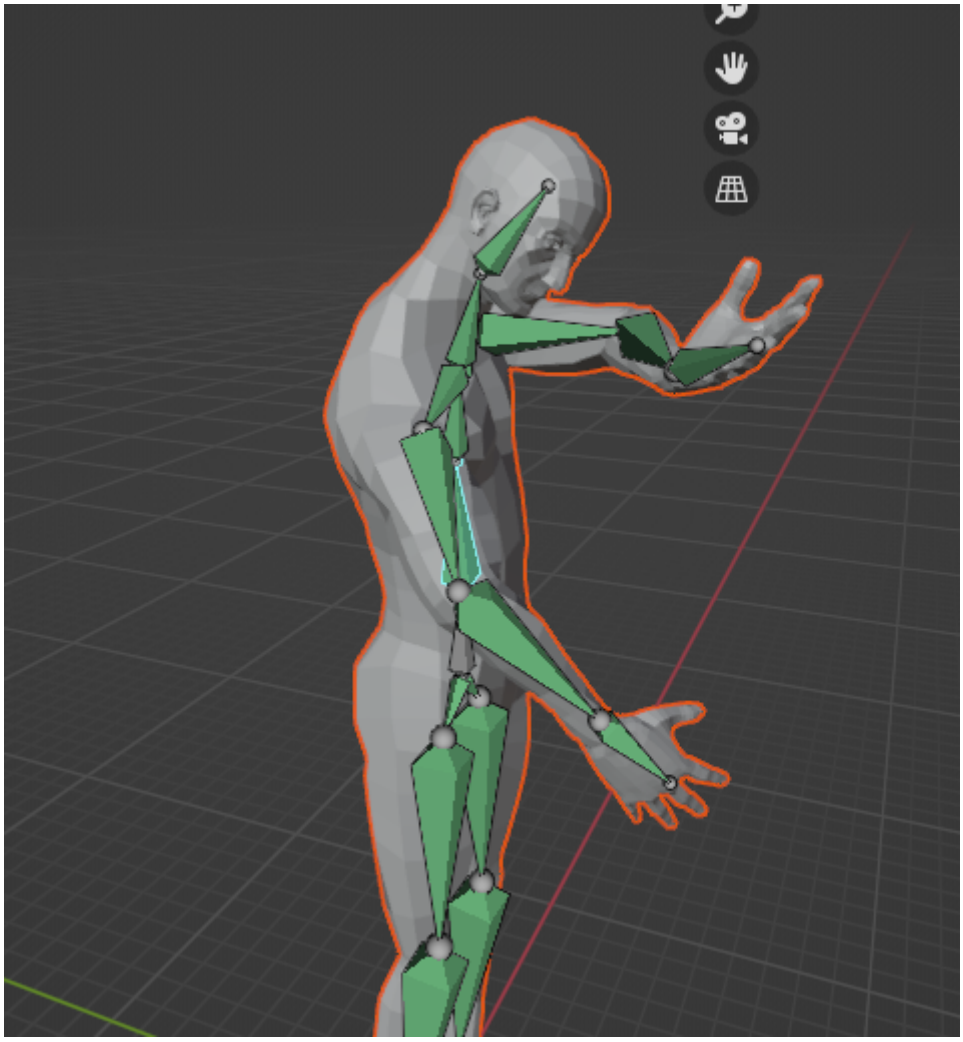

Figure 11: Screenshot depicts a realistic mesh due to constraints added.