Auto Rigging Solution for Remote Mocap Sessions

Daniël van leeuwen

July 2023

1 Abstract

This thesis explores the development and integration of an automatic rigging tool as part of a web-based remote motion capture platform. The main objective is to create a custom Auto Rigger using Python for Maya and seamlessly incorporate it into the existing TPose platform.

The research begins by investigating the process of writing a custom auto-rigger using Python for Maya. Various rigging techniques and algorithms are explored to develop a versatile and efficient auto-rigger that can generate rig setups for different character models. The rigging tool is continuously refined to improve its capabilities, accuracy, and user experience.

Next, a server-client connection is established to enable file uploading, processing, and downloading within the platform. Various system setups are prototyped and researched before the decision is made to make use of the Flask framework. A connection between the server and Maya is established to run the auto-rigging tool which results in a rigged character model.

The Three.js Library is effectively utilized to create and display 3D characters in a web-based environment. The integration of Three.js allows for real-time rendering, animation, and interactivity of character models. Performance optimizations are implemented to ensure smooth rendering and interactivity, even with complex animations and high-polygon models.

The developed auto-rigger tool is then integrated into the TPose platform. User testing and feedback collection help streamline the integration process and improve user experience. Close collaboration with the TPose development team ensures consistency in user interface design and workflow. The auto-rigger is continuously updated based on user feedback and evolving platform requirements.

In conclusion, this thesis presents a comprehensive approach to creating and integrating an automatic rigging tool within a web-based remote motion capture platform. The recommendations include further enhancements to the auto-rigger, optimization of the server-client infrastructure, and refinement of Three.js implementation.

2 Glossary

Glossary

Asynchronous refers to a programming paradigm or technique that allows multiple tasks or operations to be executed concurrently or independently. 15

Auto Rigger is a program that automates the rigging process. A

Byte code refers to a low-level representation of code that is intermediate between the source code and machine code. 8

CORS Protocol stands for Cross-Origin Resource Sharing protocol, it is a mechanism used by web browsers to control access to resources across different domains or origins. It is a security feature implemented to protect users from malicious scripts or unauthorized data access. 14

Data sets refers to a collection of structured or unstructured data that is used for training, validating, or testing machine learning models or conducting data analysis tasks. 5

Deep Learning is a sub-field of machine learning that focuses on training artificial neural networks with multiple layers to perform complex tasks. It is inspired by the structure and function of the human brain, particularly the interconnected network of neurons. 5

Dynamic Templates refers to files that contain HTML (or other markup languages) with placeholders for dynamic content. Templates are used to generate dynamic web pages by combining static markup with data from the application. 9

Event-Driven Communication is a paradigm in software development where the flow of information or actions between components or systems is driven by events. 8

Framework refers to a pre-established set of tools, libraries, and conventions that provide a structured approach to building web applications. 3

HTTP Requests stands for HyperText Transfer Protocol, it is a message sent from a client (such as a web browser) to a server, requesting specific information or an action to be performed. 15

Interpreter is a software program or tool that executes code written in a high-level programming language directly, line by line. It translates and executes the source code statements one at a time, without the need for prior compilation into machine code. 7

Library refers to a collection of precompiled code modules or classes that provide reusable functionalities and resources to simplify the development of Java applications. A

Locators are reference objects used to define specific positions or anchor points on a 3D character model. 9

Maya Autodesk Maya is a powerful and widely-used 3D computer graphics software. A

MEL stands for Maya Embedded Language. It is a scripting language specifically designed for Autodesk Maya, a popular 3D computer graphics software. MEL is used to automate repetitive tasks, create custom tools, and enhance the functionality of Maya. 8

Model-View-Controller (MVC) architectural pattern is a design pattern commonly used in software development to separate the concerns of an application into three interconnected components: the model, the view, and the controller. 9

Motion Capture is a technology used to record human movements and translate it into digital data. 1

NDI Stream stands for Network Device Interface, which refers to a technology developed by NewTek that enables the transfer of high-quality video, audio, and metadata over an IP network. NDI is designed to facilitate real-time video production and streaming workflows by allowing multiple devices and software applications to communicate and share media seamlessly. 23

OpenGL refers to a region of memory used to store data that is utilized during various stages of the rendering pipeline. Buffers serve as containers for different types of data, such as vertex data, texture data, or shader data, and they play a crucial role in efficiently transferring and processing information on the GPU. 14

Point Clouds are a representation of three-dimensional data composed of individual points in space. 6

Python is a versatile, high-level programming language known for its simplicity and readability. A

Raycast is a technique used to determine the intersection of a ray with a scene or geometry. It involves casting a virtual ray from a specific point in a given direction and checking for any intersections or collisions with objects in the scene. 10

Retargeted refers to the process of transferring or adapting motion data from one character to another. 1

Root Values are properties that can be defined once and then referenced and used throughout the entire CSS stylesheet. They allow you to define values that can be easily reused, ensuring consistency and simplifying the process of making changes across multiple styles. 20

Routes refer to the URL patterns that define the endpoints or entry points of your web application. Routes determine how the application responds to different HTTP requests made by clients. 9

RPC stands for Remote Procedure Call, it is a component or software service that enables communication and interaction between different systems or processes over a network. 17

Skin Binding refers to the process of associating a character's geometry or mesh with a skeletal structure or rig. It involves defining the influence or relationship between the joints of the rig and the corresponding vertices or points on the character's mesh. 4

Standalone refers to an application that is self-contained and can run independently on a computer or device without requiring additional dependencies or external resources. 1

Toolkits refer to collections of libraries, modules, and utilities that provide developers with pre-built components and functions to solve specific programming tasks or address particular domains. 8

UV-Mapped means that a two-dimensional coordinate system, called UV coordinates, has been assigned to the vertices of a 3D object's surface. This mapping allows textures or images to be accurately applied to the surface of the object. 24

Vertex Colours also known as per-vertex colours are colour values associated with individual vertices in a 3D mesh or geometry. 24

Voxel is a three-dimensional counterpart of a pixel (picture element). It stands for "volume element" or "volumetric pixel." Instead of representing a single point in a 2D image, a voxel represents a small volume or unit of space in a 3D grid. 13

Web-Based refers to systems, services, or applications that are accessed and operated through a web browser over the internet. 2

WebGL stands for Web Graphics Library and is a JavaScript API (Application Programming Interface) for rendering interactive 2D and 3D graphics within web browsers. 7

Contents

3 Introduction

In the entertainment industry, the demand for Motion Capture data has witnessed a significant surge. While existing motion libraries cater to certain requirements, there is a growing need for personalized and specific motion capture data. However, organizing a traditional motion capture shoot entails substantial costs in terms of logistics, time, and resources. To address this challenge, TPose has developed a cutting-edge web-based platform that offers clients the convenience of remote motion capture shoots, allowing them to direct the process from any location worldwide.

During these remote shoots, clients are provided with a comprehensive set of tools to ensure an immersive experience. The studio employs a live camera feed, enabling clients to witness the real-time performances of the motion capture artists. Additionally, a live preview of MVN, the motion capture software employed for data recording, is made accessible to clients. Furthermore, the integration of a Three.js environment empowers clients to visualize the captured motion, which is Retargeted onto a standard model. This interactive preview enables clients to freely explore and evaluate the motion in a three-dimensional space.

The current standard 3D model employed for motion preview in Three.js is T-O, the iconic mascot of TPose. T-O's anatomical proportions accurately represent those of a typical human, making it suitable for previewing most types of motion. However, clients frequently use characters with non-standard proportions, posing challenges in accurately estimating the fidelity of the recorded data and hindering the mocap artists' ability to adapt their motions based on the digital character's unique proportions.

3.1 The Assignment

The company's assignment involves the development of an auto-rigging tool that is specifically designed to integrate with their TPose platform. This tool is intended to be seamlessly connected to T-Pose within the live motion capture (mocap) pipeline, rather than functioning as a Standalone tool. The primary objective is to create a comprehensive and unified solution that leverages the unique features and advantages offered by T-Pose.

An essential criterion for this project is to ensure that the resulting rig produced by the auto-rigging tool meets or exceeds the quality standards set by existing solutions in the industry. The aim is to deliver a rigging solution that not only matches but potentially surpasses the quality offered by established options.

3.2 The Main Question

How can an automatic rigging tool be created and integrated as part of a web-based remote motion capture platform?

3.3 The Sub Questions

- How can a custom auto-rigger be written using Python for Maya?
- How can a server-client connection be established that allows the uploading, processing, and downloading of files?
- How can Three.js be effectively utilized to create and display a 3D character in a web-based environment?
- How can the created auto rigging tool be implemented in the current TPose platform?

4 Methodology

4.1 Research

In order to get a grasp on what an auto rigger should be, market research was performed into the current auto rigging tools on the market. To perform this research, a list of existing auto-rigging tools was created through desk research. Then each tool was installed and tested by the researcher to uncover its capabilities.

To find out about the academic advancements in the area of auto-rigging, extensive desk research was performed to find articles related to this topic. There are not many published papers about the automation of the rigging process since it is a rather niche subject but the ones that were found contained interesting concepts.

Additional desk research was performed into various technologies used throughout the development process.

4.2 Development

Once a clear idea was formulated of what encompasses an automatic rigging tool. A plan was drawn up for the development of the product. This process was initially split up into three separate development cycles where a proof of concept of each part was created before combining the three elements into a functional prototype. The three cycles were:

Development of an auto-rigging tool in Maya using Python. This process starting point was a functional automatic rigging tool inside Maya, created according to the Maya Python tutorial by Mark Schipper (Schipper, 2019). After extensive debugging, various adaptations, additions, and modifications were made to the scripts in order to facilitate new and improved features.

Development of a Web-Based 3D interface using Three.js. For this development cycle, the starting point was a range of examples provided by the official Three.js web page. By adapting and combining the information gleaned from several examples, a functional prototype was created.

Development of a server-client connection. This process started with desk research into the functionality of servers and the basics required to set up a connection. Based on that research, practical research was performed where a wide range of attempts was made to find a Framework that could support the envisioned product.

4.3 Testing

Prototype V2 was created by combining the three development cycles into a functional product. This Prototype was then tested with both a use case and by benchmarking.

For the use case, the client delivered a character model and a collection of motion capture recordings. The application of the prototype on these assets was performed by the researcher due to the prototype only functioning locally. The goal of this test was to see if the results produced by the auto-rigger were satisfactory for implementation in a final product.

The benchmarking was performed by the in-house staff. The goal was to test both the quality of the results compared to other auto-rigging tools, as well as the amount of time it takes to run through the process from start to finish.

Based on these results, several issues were identified. A list of actionable fixes was created and implemented to create Prototype V3.

To test the third prototype, an user test was conducted with TPose employees. Each was given the task of auto-rigging a model selected at random. After completing the task of auto-rigging a model, a simple questionnaire was filled out by the participant to assess the quality of the resulting rig and gain insights into the usability of the tool. The results are primarily for incorporation into the recommendations.

5 Market Analysis

There is a wide range of automatic rigging solutions currently on the market, most are add-ons for existing 3D software, some are stand-alone applications, and one is fully web-based. Here is a short summary of each tool, each is described in more detail in appendix .1.

- Quick Rig is a tool inside Autodesk Maya. It is a suitable choice when you need a fast and intuitive solution for rigging characters within Maya. It is ideal for projects that require a quick turnaround, such as prototyping, previsualization, or animation tests. (Autodesk, n.d.)
- Auto Rig is a window inside SideFX Houdini. it is a powerful option for rigging characters in Houdini, especially if you're working on complex projects that require advanced rigging features, like creature animations or procedural setups. It is well-suited for VFX, game development, and high-end character animation. ("Autorigs", 2023)

- Auto Character System is a Rigging and animation toolset for bipedal characters for Modo. The Auto Character System in Modo is a great choice if you prefer a streamlined and integrated workflow within Modo. It is suitable for various projects, including game development, character design, and animation, offering customization options and a range of animation helpers. ("Auto Character System 3", 2023)

- The Auto Rig Pro add-on for Blender is a powerful tool that simplifies the process of character rigging and animation. It provides extensive automation options and compatibility with other Blender add-ons, making it a versatile choice for both simple and complex character animation projects, such as short films, commercials, or game development. (Artell, 2023)

- The Rigify add-on in Blender is a powerful tool for creating character rigs and animations. It offers simplicity and ease of use, making it a good choice for beginners or projects with straightforward character animation requirements, like educational animations or personal projects. (Team, 2023)

- AccuRig by Reallusion is a feature-rich standalone tool designed to simplify the process of rigging characters for animation. It is a suitable option if you are using Reallusion software like Character Creator and iClone for character creation and animation. It provides a cohesive pipeline for creating and animating characters within the Reallusion ecosystem, making it ideal for game development, virtual production, and real-time character animation. ("Free Auto Rig for any 3D Character — AccuRIG", 2023)

- Mixamo is an online platform and service by Adobe that offers a range of powerful features for character animation and rigging. It is a convenient choice when you need a vast library of pre-built animations and a quick and easy way to apply them to your characters. It is particularly useful for projects that require a large variety of character animations, such as game development, virtual reality experiences, or animated presentations. ("Mixamo", 2023)

These programs all require different levels of user input but all are generally based on the same principles;

- A mesh is provided by the user

- The auto rigger generates a generic skeleton shape

- The user adjusts the shape to fit their mesh

- a skeleton is generated and constraints are applied such as Skin Binding, IK/FK switches, and animation controllers

This appears to be the expected workflow for every existing auto-rigger.

6 Advanced Technologies

Throughout the years there have been many researches that go into the simplification and improvement of rigging and animation. Several technologies have been discussed and considered for this research but none were applicable to the current situation for various reasons.

6.1 Neural networks

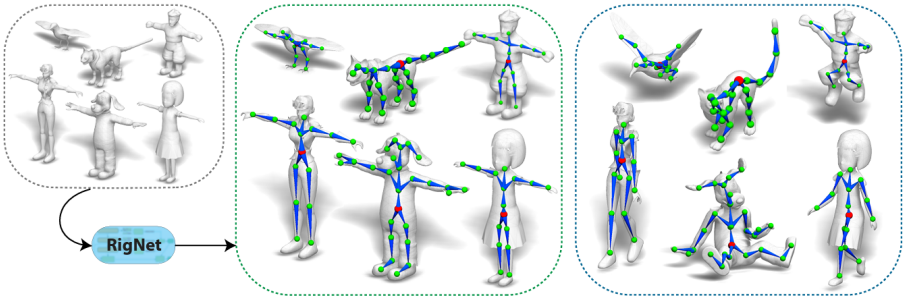


Figure 1: Example of the RigNet workflow Xu et al., 2020

Neural networks, particularly Deep Learning techniques, have been employed in the field of computer graphics and animation to automate the rigging process. These networks can be trained on large Data sets of pre-rigged characters, where the input is the geometry or pose of the character, and the output is the corresponding rig parameters. (Xu et al., 2020)

By leveraging neural networks, the automatic rigging process can be learned and generalized from examples, enabling the network to predict the rigging parameters for a given character model. This approach eliminates the need for manual rigging, which can be time-consuming and require expertise in character animation.

6.2 Point clouds

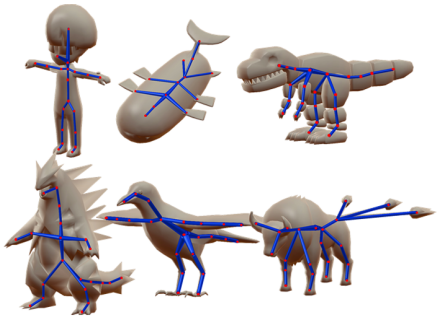


Figure 2: A variation of meshes rigged using point clouds (Xu et al., 2019)

To rig a mesh using Point Clouds, the point cloud data can be processed and analyzed to extract important information, such as the positions and connectivity of the underlying geometry. This information can then be used to generate a mesh that closely represents the object's surface.

Once the mesh is created from the point cloud, the rigging process involves defining a skeleton or an articulated structure that controls the deformations and movements of the mesh. The skeleton typically consists of joints and bones that are connected in a hierarchical manner to mimic the object's underlying structure.

The point cloud data can be utilized in rigging by associating each point with the nearest bone or joint in the skeleton. This association allows the mesh to be influenced by the movements of the corresponding bones, resulting in realistic deformations during animation. (Xu et al., 2019)

6.3 Mesh sequences

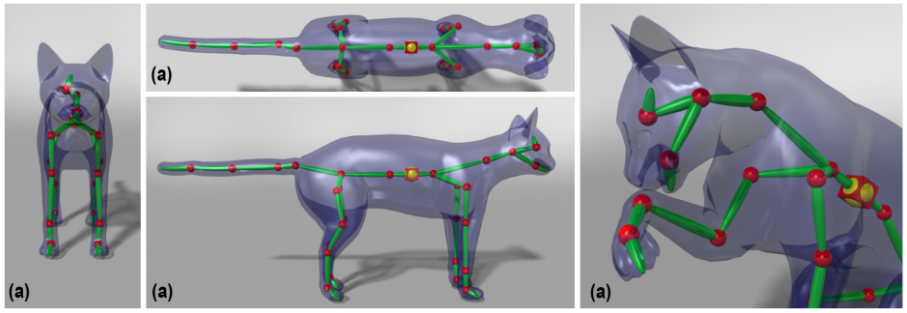


Figure 3: cat model rigged using a mesh sequence (Le and Deng, 2014)

A mesh sequence, also known as a vertex animation sequence or a deformation sequence, is a collection of meshes that represent the changing shape of an object over time. Each mesh in the sequence typically corresponds to a specific frame or keyframe of an animation.

To rig a mesh from a mesh sequence, the rigging process involves defining a skeleton or an articulated structure that controls the deformations and movements of the mesh at each frame. The skeleton typically consists of joints and bones that are connected in a hierarchical manner to mimic the object's underlying structure. (Le and Deng, 2014)

While these technologies are interesting developments in the field of automating the rigging process, they are not applicable for an auto-rigger designated to be part of a remote motion capture pipeline. While these techniques simplify the rigging process to the point where little to no user input is required, the output skeleton is very unpredictable. This would require a lot of manual work for the following step in the motion capture pipeline, retargeting.

7 Supplemental Technologies

during the development and practical research, a number of technologies were researched and applied. A short summary and explanation are given below of the most significant supplemental technologies used.

7.1 Three.js

Three.js is a popular JavaScript library created by Ricardo Cabello (Cabello, 2010) that provides a framework for creating and displaying interactive 3D computer graphics in web browsers. It simplifies the process of working with WebGL, a web-based graphics API, by providing a higher-level abstraction and a range of useful features.

With Three.js, developers can easily create and manipulate 3D scenes, including rendering and animating objects, applying materials and textures, handling lights and shadows, and incorporating camera controls. It abstracts away the complexities of low-level WebGL programming, allowing developers to focus on creating engaging and visually appealing 3D content.

This library is used by both TPose and popular websites such as Mixamo, making it a logical choice for developing the auto-rigger.

7.2 MayaPy

MayaPy refers to the Python Interpreter utilized within Autodesk Maya, a prominent 3D computer graphics application. It allows users to leverage the power and flexibility of the Python programming language to script and automate various tasks in Maya ("The Maya Python Interpreter, MayaPy", 2023).

As the Python interpreter embedded in Maya, MayaPy provides direct access to the Maya API (Application Programming Interface) and extensive libraries

specifically designed for working with Maya's features and functionalities. This includes the ability to manipulate 3D objects, create and modify animations, control rendering settings, and interact with the overall scene and assets.

With MayaPy, users can develop custom tools, write scripts, and create plugins that extend Maya's capabilities. Python's intuitive syntax and extensive library ecosystem make it a popular choice for automating repetitive tasks, building complex workflows, and integrating Maya with other software or pipelines.

By utilizing MayaPy, artists, animators, and developers can enhance their productivity and efficiency in working with Maya, enabling them to accomplish tasks more effectively and streamline their creative processes.

7.3 Socket.io

Socket.IO is a widely used JavaScript library that facilitates real-time, bidirectional communication between web clients and servers. It establishes persistent connections, enabling instant data transfer and Event-Driven Communication. By abstracting the underlying protocols, Socket.IO simplifies the handling of real-time data through a high-level API that supports event-based messaging (Arrachequesne, 2023).

It works seamlessly across platforms and devices, supporting both server-side frameworks and client-side JavaScript. With additional features like rooms and namespaces, Socket.IO allows for targeted communication and offers built-in support for handling disconnections and errors. Its versatility has made it popular for applications requiring real-time synchronization and communication, such as collaborative apps, multiplayer games, and chat systems.

7.4 Watchdog

Watchdog is a Python library that simplifies file system monitoring. It provides an interface for detecting changes in files and directories, allowing for real-time responses or automated actions. With Watchdog, developers can easily monitor specific directories or files, abstracting the complexities of file system events across platforms. The library aims to automate tasks triggered by file system changes, offering flexibility and an intuitive API for monitoring and responding to modifications, additions, or deletions in the file system (Gorakhargosh, 2023).

7.5 Keystone

Keystone is a command line script that simplifies the distribution of Python Toolkits in Autodesk Maya. By pointing it at a folder containing the toolkit, Keystone compiles Python files into Byte code, packages them into a compressed archive, generates a startup script, and compiles everything into a self-contained MEL file. This single-source distribution requires no additional infrastructure, making it convenient for Maya users (Theodore, 2023).

7.6 Flask

Flask is a lightweight and widely used web framework written in Python. It offers a simple yet powerful approach to web application development, following the Model-View-Controller (MVC) architectural pattern. By defining Routes, handling requests, and rendering Dynamic Templates, developers can easily create web applications and APIs using Python (Ronacher, 2023).

The strength of Flask lies in its minimalistic and modular design. It allows developers to choose and incorporate only the necessary components and extensions based on their specific project requirements. Flask offers essential features such as URL routing, request handling, session management, and template rendering. Integration with popular databases like SQLAlchemy and SQLite enables seamless data storage and retrieval. Additionally, Flask's rich ecosystem of extensions provides additional functionalities such as authentication, form validation, and API development.

What sets Flask apart is its simplicity and ease of learning. Even developers new to web development or Python can quickly grasp its concepts and start building applications. The active Flask community further contributes to its popularity, offering extensive documentation and abundant online resources to support developers in their Flask projects. Overall, Flask is a versatile web framework that empowers developers to create web applications and APIs efficiently using Python, making it a favoured choice for projects of various sizes and complexities.

8 Development

8.1 How can a custom auto-rigger be written using Python for Maya

Based on the market research of what is expected by a client when using an auto-rigger and the company's requirement of having a platform-native auto-rigging solution, the decision was made to write a custom auto-rigger using the MayaPy interpreter. This will allow the auto rigger to be fully integrated into the remote motion capture pipeline and run on a back-end server without the need for UI graphics to be loaded. The Maya Python auto-rigger scripts written by Mark Schipper have been used as a base to expedite the development process.

In appendix .2, the adjustments needed to make the initial setup functional are described and the creation of a Maya-based auto-rigger is completed. From there on, in appendix .3, the creative process begins with changing the way Locators are placed. Appendix .4 describes how inspiration was drawn from the Mixamo web interface which resulted in a similar idea where the user places locators on a 2D image of the model to indicate the location of a limited number of joints.

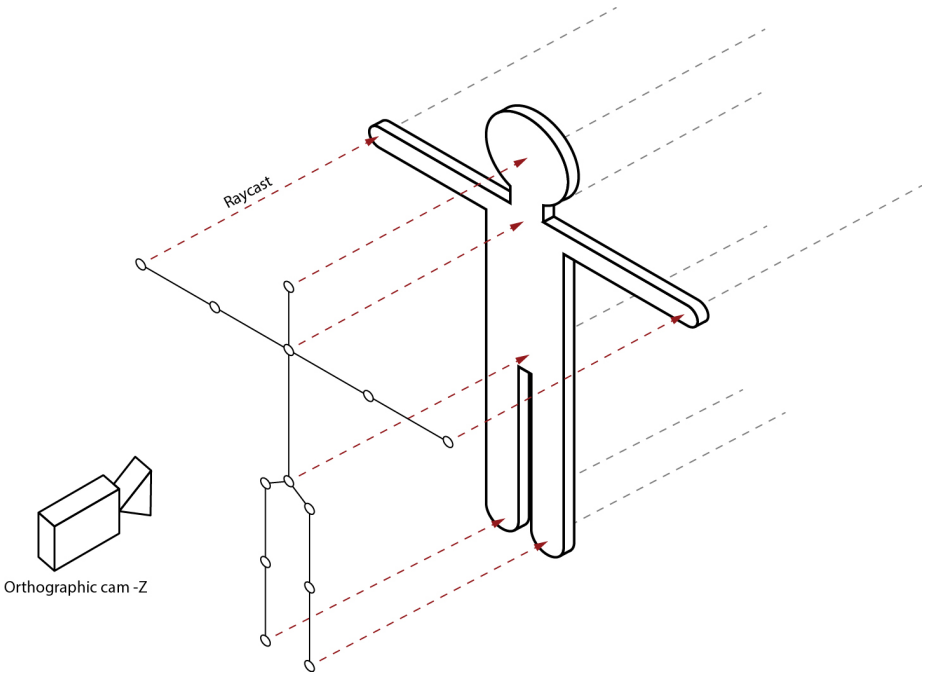


Figure 4: a diagram sketch of the process with which locators are placed from a 2D view

By placing an orthographic camera in the -z direction in front of the model, a flat 2D image is created. The next step is placing locators over the locations of the joints. From each locator, a Raycast function is called and the location of where the model is hit is stored. For a rig, the bones have to be inside the model, not on the surface. To calculate the center of the point where the model is hit, a second raycast is created in the negative direction of which the hit location is also stored. Then the average between the two hits is calculated and a new locator is placed in the center of the points.

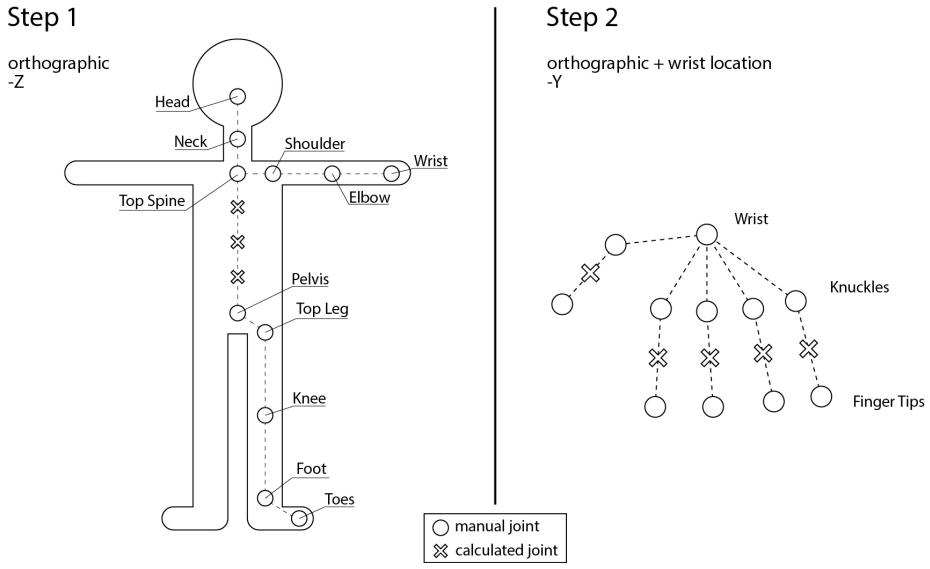


Figure 5: a diagram sketch of the rigging process separated into two steps

Appendix .5 describes how this process quickly revealed a weakness: the hands of 3D characters are for the majority angled with the palms facing down, which makes it impossible to place locators on individual fingers. Since hand motion capture is a major part of the TPose motion capture pipeline, a solution had to be found.

By separating the rigging process into two steps, with the camera changing from -z to -y and repositioning to the wrist locator, it was made possible to accurately add locators to the fingers.

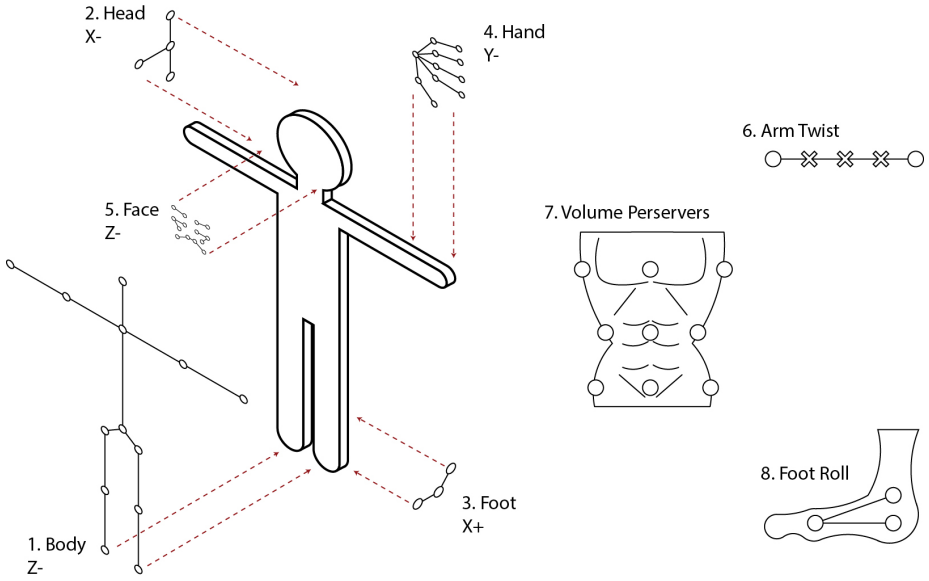


Figure 6: a diagram sketch of a concept for the rigging process with 7 steps

In appendix .6, a brief consideration for a 5-step system was considered to facilitate placing the jaw, feet, and face locators from their respective angles. This was however abandoned due to being overly complex, the jaw not being motion captured, and the position of the feet being automatically calculable.

Additionally, three extra steps were considered for advanced rigging features such as arm twist, foot roll, and volume preservers. This concept was discarded at a later stage in order to keep the auto-rigger easy to use and integrate into the TPose pipeline.

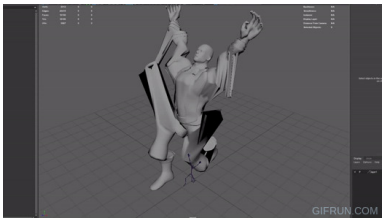
in appendix .14 and appendix .15 the attempted implementation of these steps is described. This process was never finished in its entirety during this research in favour of focusing on the next sub-questions.

The last step in the automatic rigging process is the skin binding. Within Maya, there are 4 weight painting options for the bind skin function:

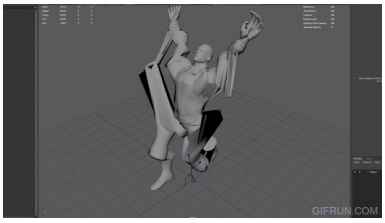
- Closest Distance; this function assigns weights to vertices based on the closest distance to nearby bones or joints. It provides localized influence and accurate deformations, ensuring that vertices are primarily affected by the nearest bone or joint. This method is commonly used to achieve precise control over specific parts of a character's mesh during animation.
- Closest in Hierarchy; this function determines the nearest bone or joint for each vertex of the character's mesh by evaluating the hierarchical relationship between the mesh and the skeleton. By considering the hierarchy, this method ensures accurate influence and deformation of the vertices during character animation.

- Heatmap; this function assigns weights to vertices based on their proximity to bones or joints in the skeleton. The weights are represented visually as a heatmap, with warmer colours indicating higher weights and stronger influence. This method allows for smooth and gradual transitions in vertex influence, resulting in realistic mesh deformations during character animation.
- Geodesic Voxel; this function divides the character's mesh into voxels and assigns weights to each Voxel based on the geodesic distances to nearby bones or joints. By considering the local geometry and proximity, this method provides accurate and realistic deformations during character animation.

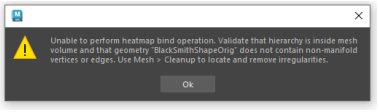
Each option was tested on the same model performing the same motion, the results are displayed in figure 7



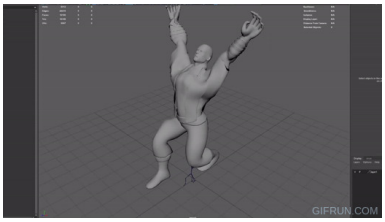
(a) Closest Distance skin bind method performed on the Blacksmith model



(b) Closest in Hierarchy skin bind method performed on the Blacksmith model



(c) Error message when applying the Heatmap skin bind method



(d) Geodesic Voxel skin bind method performed on the Blacksmith model

Figure 7: Test results from the bind methods

As is immediately apparent, only the geodesic voxel method provides a stable solution binding the skin without the possibility of manually adjusting the weighting afterwards.

Both Closest in Hierarchy and Closest Distance struggle with keeping the limbs separate from the torso when the model is not in a T-Pose. Additionally, the lines between different influences are very apparent, causing inorganic deformations.

The Heatmap bind method proves to be impractical due to its clear limitations when it comes to the models it works on. This Blacksmith model had clean topology but some gaps in its shape, causing the heatmap to be unable to fill the shape.

Because Geodesic Voxel uses the geodesic distance, also known as the shortest path, to calculate weights, it won't be able to jump across bigger gaps in the character mesh such as between the forearm and the side of the torso. This preserves the shape of the model much better and creates very organic deformations. Geodesic Voxel also supports models with degenerate and non-watertight geometry. (Dionne and De Lasa, 2013)

During the research, an issue was uncovered with the Geodesic Voxel method. In order to voxelise the mesh, it relies on creating an OpenGL which in turn requires a GPU to load. This is an issue when running the standalone version of Maya to execute the auto-rigger code, since no GUI is created, the GPU can not be accessed and no OpenGL buffer is created.

Several approaches were considered when searching for a solution to this issue, but in the end, the decision was made to delegate the task of running the Geodesic Voxel without the need for an OpenGL buffer to the Autodesk Team. This does mean that the auto-rigger will have to start the full version of Maya which decreases its processing speed and it will require the server it runs on to have a GPU which incurs some additional costs.

More details about resolving the issue with the Geodesic Voxel bind method can be found in appendix .7

9 How can a server-client connection be established that allows the uploading, processing, and downloading of files?

In order to facilitate a web-based service to clients in remote locations, a system for communication has to be set up. During the research, several systems were considered.

For the initial server prototype, a Socket.io JavaScript server was used with a JavaScript client. The process is covered in appendix .8. During development, several issues were encountered such as the CORS Protocol not allowing access to files from places that are not on the web. Eventually, a basic server-client setup was achieved that allows the client to upload a file to the server which is then opened and displayed by a Three.js page containing the proof of concept. A video of this setup can be found in appendix .9.

To establish a connection between the server and the Python-based auto-rigging script, it was decided to create a basic Python server to replace the JavaScript server. Several frameworks were tested to run the Python server.

- Gunicorn, short for "Green Unicorn," is a Python WSGI (Web Server Gateway Interface) HTTP server. It is designed to efficiently handle incoming HTTP requests and serve Python web applications. Gunicorn

acts as a middle layer between the web application and the web server, managing the communication and providing concurrency and scalability (Chesneau, 2023) This server was discarded because it only works on Linux devices and this research was performed on a Windows device.

- Waitress is a pure-Python, production-ready WSGI server designed to serve web applications. It is known for its simplicity, reliability, and ease of use. Waitress implements the WSGI specification, allowing it to handle HTTP requests and communicate with web applications written in Python. ("Waitress", 2022) This server was discarded because it does not support Socket.IO libraries.

- Eventlet is a lightweight concurrency library for Python that allows for the easy development of highly concurrent and scalable applications. It is built on top of Greenlet, a micro-threading library, and provides a simple and intuitive API for writing asynchronous code. ("Eventlet Networking Library", 2023) While functional, it was unclear during the research when a user connected and disconnected from the server.

- Uvicorn is a lightning-fast, lightweight ASGI (Asynchronous Server Gateway Interface) server that is specifically designed to run Python web applications. It leverages the power of Asynchronous programming to provide high-performance and efficient handling of HTTP Requests. ("Uvicorn", 2023) Initially, this server managed to run Socket.IO and displayed when a connection was established to a client as well as when a client disconnected, but when combining it with the earlier file-upload system, it was unable to load the Socket.IO files.

The research stagnated on running a Socket.IO server. Since Socket.IO is a JavaScript library and the server language has been changed to Python, the decision was made to scrap the Socket.IO library and continue development using Flask.

Using this framework allowed for easy use of the POST and GET request methods used in HTTP. GET requests are used to request data from the server, and POST requests are used to submit data to the server. By following an example found on 'GeeksForGeeks.org' (GeeksforGeeks, 2023), the server-client setup was adapted into a flask server serving an HTML template containing a Three.js script alongside a static folder with all the Three.js modules. This made it possible to route the user from the upload page to the Three.js page by responding to the POST and GET requests received by the server. It also allowed the uploaded file to be queried and the file path to be sent to the Three.js content on the next page which ensured the client shows the correct file to the user.

With the server receiving requests, the next step was to start the auto-rigger from the server. In order to facilitate this process, three different approaches were considered:

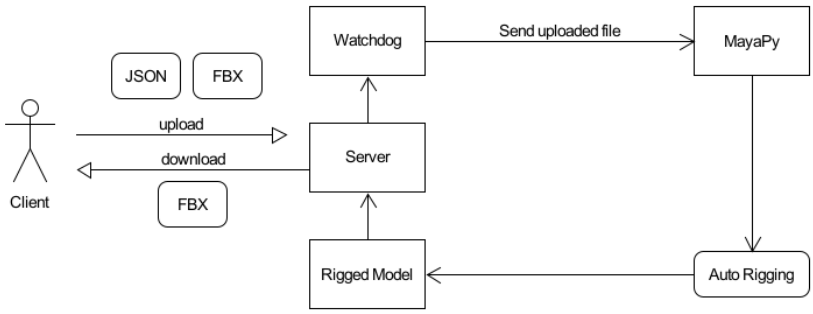


Figure 8: a diagram of the first considered approach

For this idea, the client uploads an fbx file, a JSON file of the locator positions is generated, the Watchdog library is used to detect these uploads, once completed it runs a command in MayaPy which would return a rigged character which is then sent to the server and the server sends it to the client.

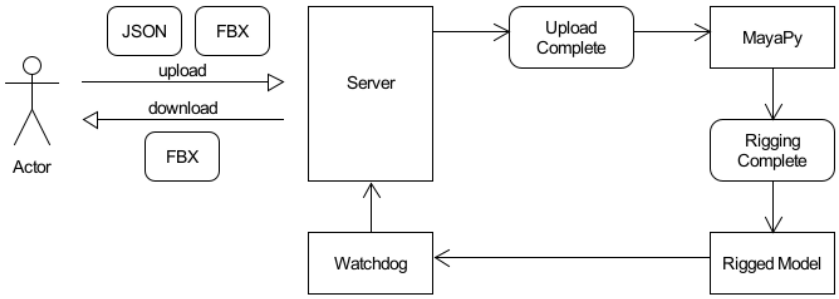


Figure 9: a diagram of the second considered approach

The second considered approach was that while the server is running, a separate script watches over the folder structure of the server, once a folder contains two files (the uploaded fbx and the generated JSON file) it will trigger a new instance of MayaPy that looks for files in the folder it is triggered in. Then Watchdog is used to check for the MayaPy output to then upload it to the server.

This idea was discarded because constantly checking every folder on the server can quickly become a drain on resources

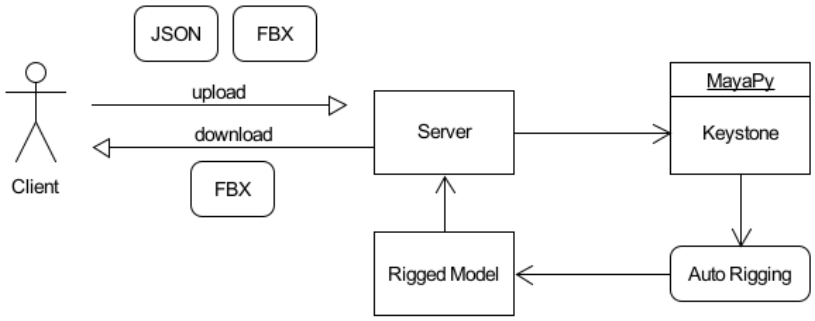


Figure 10: a diagram of the third considered approach

This third considered approach was to have a clone of each client on the server side to facilitate the uploading of files to the server. Since the upload function of the server was working it might have been possible to replicate this on both sides. Keystone was also incorporated to run the auto-rigger from the server since no solution seemed apparent.

Along with these concepts, an attempt was made to employ Keystone as a way of starting the auto-rigger from the server. This attempt was however unfruitful and has been written about in detail in appendix .16.

While Keystone was being researched, a different way of accessing the Python command line in Maya was discovered Theodore, 2014. Since the Maya command port is unavailable when running a standalone version, an RPC server can be used to run a Maya Standalone that does accept remote commands. This RPC server is started from the command line by calling:

cd C:Program FilesAutodeskMaya2023bin
mayapy.exe Path/to/Documents/maya/2023/scripts/standaloneRPC.py

This revealed that Python scripts can be called through a script by running MayaPy with a path to the desired Python script as an argument.

To incorporate this concept into the Flask Python Server, the subprocess.call() function was used when the JSON file was uploaded to the server through a POST request.

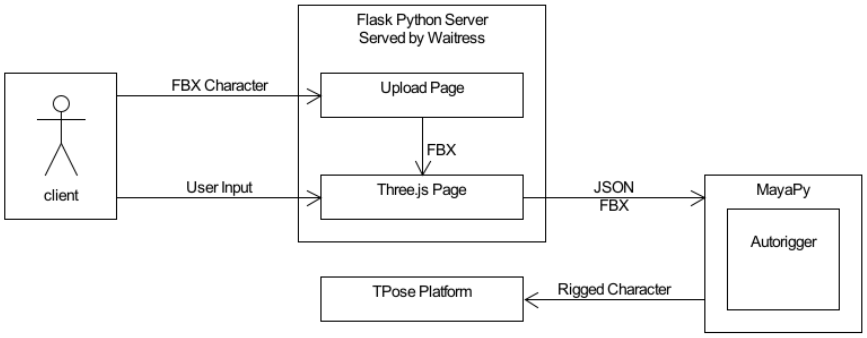


Figure 11: a diagram of the final approach

The final network setup consists of a Flask Python Server that routes the user through two html templates; the upload page where the unrigged character fbx is uploaded, and the Three.js page where the positions of the locators are provided by the user. The server is being served using Waitress. Once the user input is submitted, a JSON with the data and the FBX file are sent to the Maya Python interpreter which in turn runs the auto-rigging scripts. The rigged character is then uploaded to the TPose platform where the client can use it during a shoot to preview recorded motion.

10 How can Three.js be effectively utilized to create and display a 3D character in a web-based environment?

To create a basic 3D environment in Three.js and display it in the web browser, you need three elements:

- A scene; is a container where 3D objects, lights, cameras, and other elements are organized and rendered. It represents the virtual environment where the 3D graphics and animations are displayed.
- A camera; represents the viewpoint through which the 3D scene is observed and rendered. It controls the perspective, framing, and visibility of the scene, allowing developers to define how the 3D world is viewed by the user.
- A renderer; is responsible for converting the virtual 3D scene into a 2D image that can be displayed on the screen. It utilizes WebGL or the 2D canvas API for rendering, handles lighting, shading, and visual effects, and provides customisable settings for the output display.

For this research, a simple scene containing an orthographic camera facing in the -z direction was created. In order to display the uploaded 3D model, a component called FBX Loader was imported and utilized. An FBX loader is a component used to import and parse 3D models saved in the FBX file format. It enables developers to bring in complex scenes, meshes, animations, and materials created in software packages supporting FBX. The loader converts the FBX data into a format compatible with Three.js, allowing developers to incorporate FBX models into their Three.js projects.

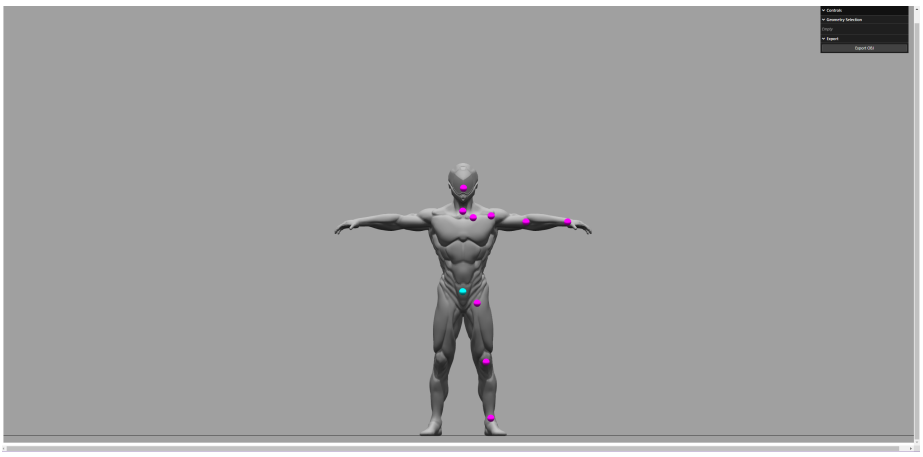


Figure 12: the simple Three.js environment with a 3D character mesh and spheres as Locators

In order to create locators that allow the user to indicate where the joints will be placed, the Drag Controls example found in the Three.js documentation website was used and modified. Drag Controls is a utility that enables interactive dragging and manipulation of 3D objects within a scene. It simplifies the implementation of object-dragging functionality and enhances user interaction by allowing intuitive object manipulation through mouse or touch input.

"Lil-gui" is a commonly utilized user interface (UI) tool within the Three.js ecosystem. It offers an interface for dynamically adjusting the properties of JavaScript objects during run time. As a preferred choice by Three.js, it serves as a standard UI component, providing seamless integration with Three.js projects. With its modern web standards and user-friendly enhancements, "lil-gui" offers an efficient and intuitive way to interact with and modify JavaScript object properties in real time.

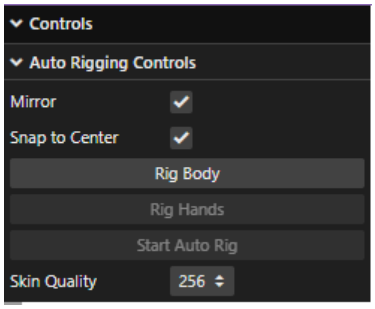


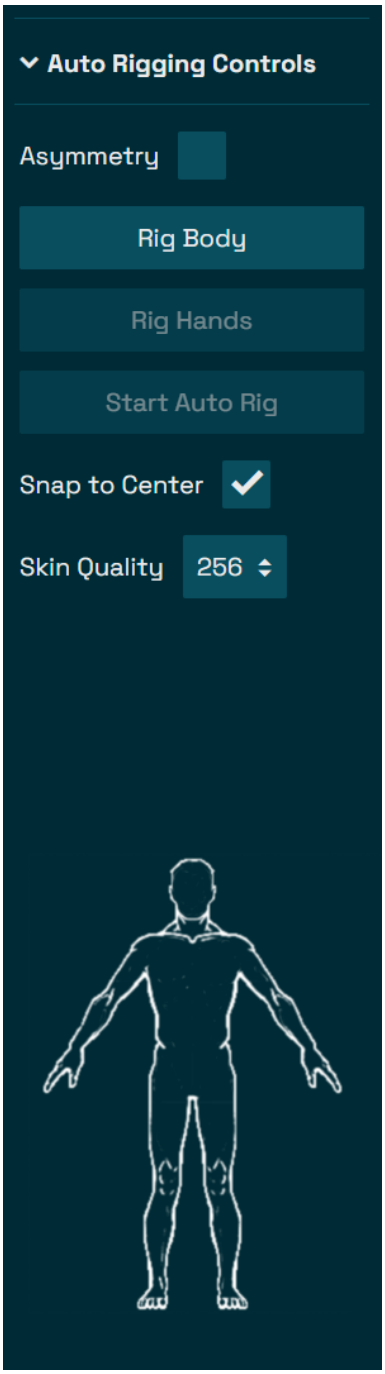
Figure 13: default style sheet

It comes with a built-in style sheet for ease of use, but in order to more seamlessly integrate the auto-rigging system into the TPose platform, the choice was made it write a new style sheet based on style elements found on the TPose website.

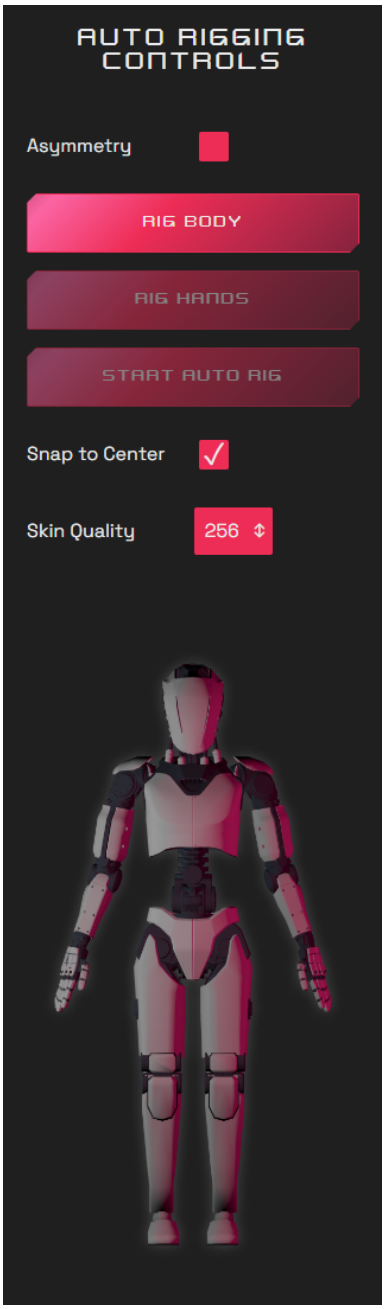
This process took two steps, first research was performed into how the style sheet worked and how it could be adapted dynamically. The first version made use of the variables exposed in the kitchen sink demo which are used as Root Values in the overall style sheet. These allowed for manual-style adaptation while the script was running, but that was not the desired way of handling things.

For the second version, the default style sheet was uncovered in the web page inspector. By stopping the lil-gui window from using the preset style sheet but instead manually applying the style sheet through the ¡style¿ tag in the HTML page, it was possible to remove rather than overwrite values.

With help from the in-house designer, Nick, and the TPose website, a matching UI style was created for the final prototype.



(a) Closest Distance skin bind method performed on the Blacksmith model



(b) Closest in Hierarchy skin bind method performed on the Blacksmith model

Figure 14: Test results from the bind methods

Data has to be transferred from the Three.js page to the Python auto-rigging tool in Maya. In order to facilitate the transfer of data between two different languages, JSON (JavaScript Object Notation) files are often used. JSON is a lightweight, human-readable data format commonly used for data interchange between web applications and services. It serves as a language-independent format for representing structured data, making it easy for different systems to exchange information. ("JSON", 2023)

The values that had to be transferred between Three.js and Python were:

- the positions of the body locators
- the positions of the left- and right-hand locators
- the direction of the camera used to place the hand locators
- the skin bind quality parameter

All these values were written out into a dictionary with a corresponding key to access each value. By concatenating the created dictionaries together, a single 'JsonContent' value is created which is then turned into a file using the 'JSON.stringify()' function. This file is then invisibly submitted to a form and uploaded alongside the uploaded 3D model.

11 How can the created auto-rigging tool be implemented in the current TPose platform?

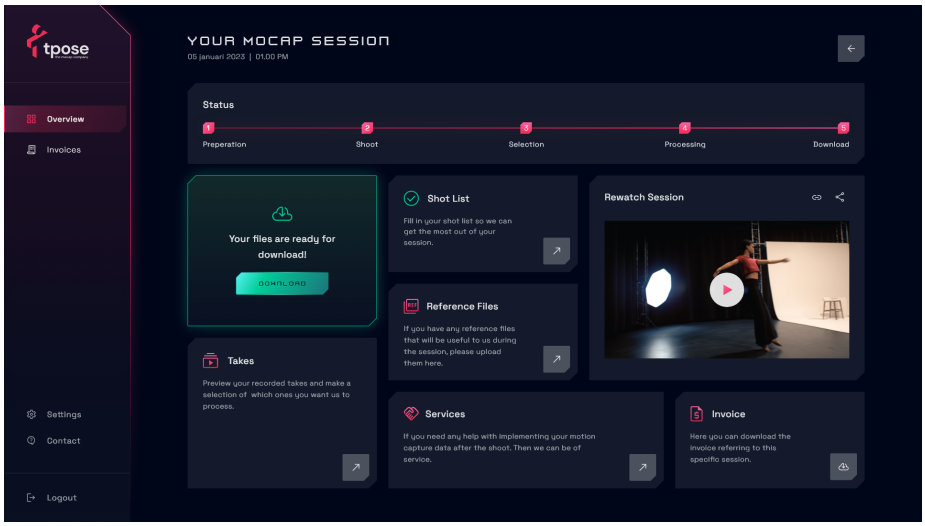


Figure 15: The TPose dashboard design as of June 2023

The development of the TPose platform was still ongoing at the conclusion of this research. As a result, the upcoming chapter will primarily consist of theoretical speculation based on the existing development plans.

The TPose platform functions as a website that enables clients who have scheduled remote motion capture shoots to access their accounts. Once logged in, clients gain access to a dashboard equipped with various features. Among these features is a page where clients can upload their shot list before the shoot begins. TPose then reviews and approves the shot list to ensure that the client's expectations align with what is feasible. Additionally, clients can specify the character they wish to use for each shot, enabling the interface to dynamically switch between characters during the shoot.

The platform provides a dedicated environment for the shoot itself. Clients receive an NDI Stream that showcases the live output from the link motion capture suit in MVN, a retargeted version presented through Three.js on a character, a camera view of the mocap artist in the studio, and conferencing options such as video connections and chat windows. After the shoot, clients can access a list of their recordings, view them after reprocessing, and download them.

As an additional feature, the auto-rigger tool will be integrated into the dashboard, enabling users to add un-rigged characters to the character library. This library allows users to select a character and preview their data with it. However, before the smooth integration of this feature, adjustments needed to be

made to the rigging and exporting processes of the characters to align with the retargeting performed in the Three.js environment. These adjustments involved posing the character to a clean T-Pose after rigging and skinning, as well as setting the joint orientation from the standard xyz to yxz.

12 Testing

12.1 Prototype V2

12.1.1 Case Study

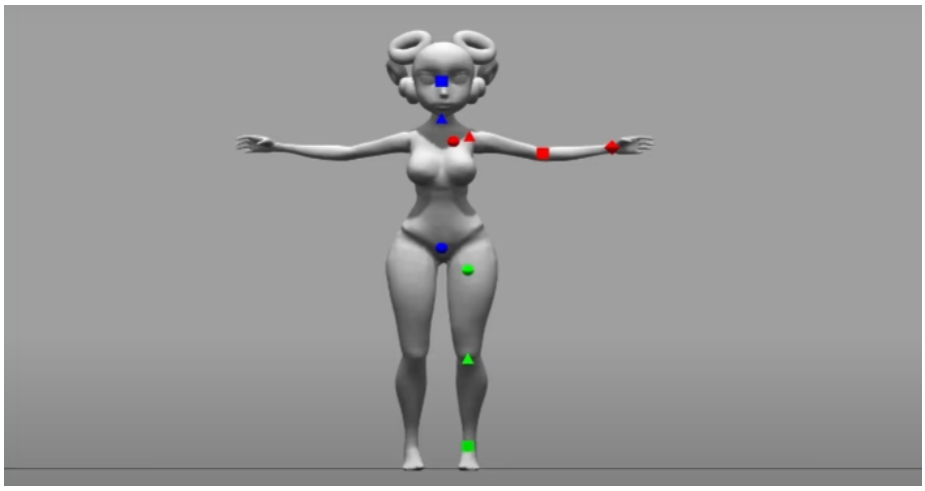


Figure 16: A screenshot of the model used in the use case inside of the three.js environment

Upon completion of the second prototype, the researcher was approached for a potential use case for the auto-rigger. A student from Zuyd University of Applied Sciences created a 3D model using the sculpting tool Nomad and recorded motion capture data using the MVN Link suit. They were struggling with rigging their character and retargeting it to the motion capture data which is exactly what this tool is designed to do. Since the prototype was not running online yet, the tool was applied to the delivered assets by the researcher rather than the user.

The delivered assets had various issues that had to be resolved before they could work in the auto-rigger. These were primarily issues related to conforming to industry standard practices such as the model having proper topology, as well as Nomad-specific issues such as the textures being stored as Vertex Colours rather than UV-Mapped textures.

After these issues were resolved, the auto-rigger worked as desired and an acceptable result was achieved. The result satisfied the users' needs for both

the quality of the rig and the amount of time it took.

For the full account of the Use Case Test, you can consult appendix .10, additionally, videos of the result can be found in appendix .11 and appendix .12

12.1.2 Comparitive Study

The first actively tested prototype was dubbed 'V2' due to the preceding prototypes for uploading files, automatic rigging, and displaying the 3D model in a web browser being considered the first versions of their respective development branches. During the testing of Prototype V2, a wide range of 34 characters was gathered from both in-house sources as well as the Unreal Marketplace. These characters varied greatly in complexity, proportions, and attributes, but were all generally considered to qualify as 'humanoid'. The full extent of the testing can be found in appendix .17, in this paper, only the comparative part of the testing will be covered.

While most issues are apparent, with the goal being to be on par with existing solutions, a comparison is needed. By running a selection of the more challenging models through similar software, a good idea of what the current auto-rigger lacks can be formed. By applying a similar skinning test animation to each model, a good comparison can be made. AccuRig did not have the same skinning test as Mixamo and TPose, the decision was made that it is still possible to judge the results.

- Aurora performed quite well in all three auto-rigging tools. The only issue found in the TPose auto-rigger is in the armpits deforming along with the arms. Something along the lines of volume preservers could resolve this issue.

- Crunch was one of the models with very clear issues, it was interesting to see how other auto-riggers dealt with the intersecting limbs. The results from the TPose auto-rigger and AccuRig are quite similar, the model animates nicely but has long stretched artefacts between the legs and hands. Mixamo was unable to process this model, likely due to its machine learning algorithm being unable to clearly define the limbs.

- Especially the head of the Goblin character was a challenge in each auto-rigger. Since there is no real torso on this character, the 'head' bone tends to create odd deformations in the face. This model pushes the boundaries of what can be considered a humanoid character so these results are expected.

- GRIM is an odd robotic creature controlled by a goblin shape between its legs. Both AccuRig and TPose manage to preserve this goblin shapes volume quite nicely, weighing it to the torso rather than the legs. In Mixamo however, this shape is folded flat. Otherwise, this model moves as desired, in TPose there are some small shoulder issues which again can be solved with some form of volume preservers.

- Sevarog causes issues in each auto-rigging software due to its long strands and lack of legs. In Mixamo the strands turn into jagged artefacts, in TPose and AccuRig they retain their shape better but still rigidly move along with the bones. For the legs, in Mixamo the longest strands are calculated as the legs which works somewhat, in TPose there is an issue where the feet are placed automatically but they don't hit the mesh, causing them to be placed in an odd location which causes some artefacts in the head. in AccuRig the longest strands are also made into the legs, but there is also an option to 'mask' out certain limbs so they don't affect the model.
- The Skeleton King is a challenge for every auto-rigger due to its asymmetry, large prop and lack of topology. AccuRig picks up the large props on the body the best but also struggles with the large sword and the head hunching forward. TPose clearly struggles with this model the most, causing a lot of big artefacts on the cape, sword, and other asymmetric parts. It is likely that some of this can be mitigated by adding asymmetry support to the auto-rigger.

12.1.3 Time

Character	Time to place locators	Time to run the rig
Aurora	0:58	0:45
Crunch	0:44	N.a.N.
Goblin	0:31	0:57
GRIM	1:03	1:13
Sevarog	1:05	1:18
Skeleton King	1:03	0:57
Average	0:54	1:02

Mixamo Time

Character	Time to place locators	Time to run the rig
Aurora	1:05	0:48
Crunch	0:53	1:19
Goblin	1:04	0:40
GRIM	1:14	1:09
Sevarog	1:16	0:58
Skeleton King	1:00	0:40
Average	1:05	0:56

TPose Time

Character	Time to place locators	Time to run the rig
Aurora	4:19	0:24
Crunch	4:53	0:37
Goblin	2:55	0:47
GRIM	2:27	0:24
Sevarog	4:00	0:18
Skeleton King	3:49	0:16
Average	3:44	0:28

Accurig Time

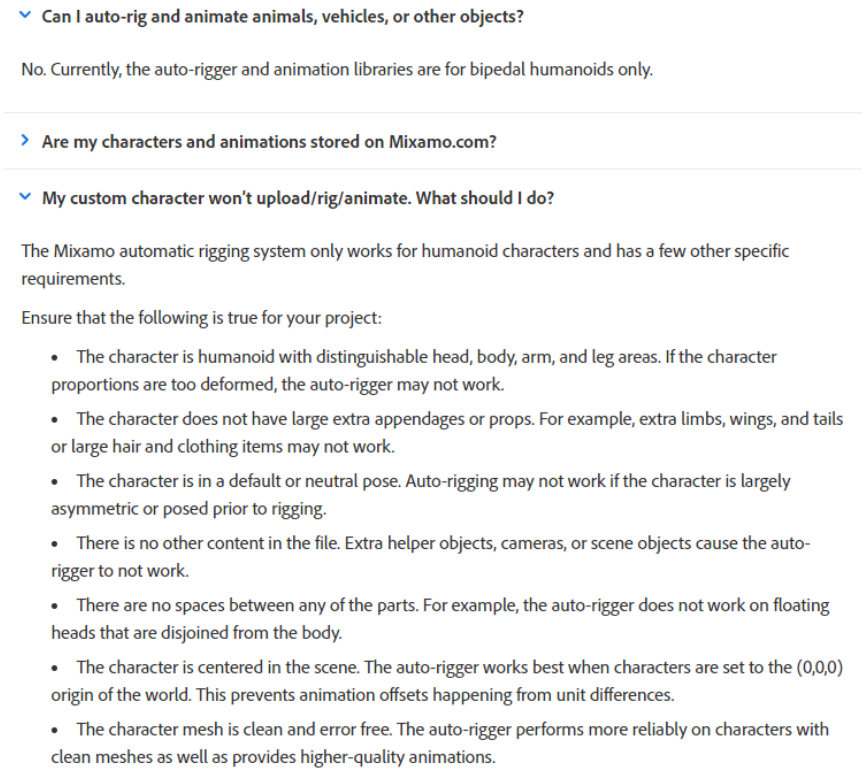
Comparing the time it took to go through the process from upload to finished rig provides insight into the fluidity of using the tool. Both Mixamo and TPose are pretty simplistic in the amount of work needed to rig a character, and the UI is quite intuitive allowing for a quick workflow. AccuRig takes almost 3,5 times longer to place the locators, this is in part due to having to manually place the locators on both hands manually and having to correct the wrongly predicted location of the locators. For TPose, the hand locators also have to be manually placed but the location of both the body and hand locators is not calculated through machine learning, instead, it places them in the approximate human proportions which can save a lot of time compared to Mixamo where they have to be dragged from the left side of the screen to the model and AccuRig which often misplaces the locators.

12.1.4 Conclusions

There are certain requirements for a model to be processed by the current auto-rigger:

- It must be a .fbx file.
- The .fbx file must contain a single mesh object.
- The model must have sufficient topology for deformations.
- The model must be largely symmetrical.
- The model can not contain large props.
- The models' limbs can not intersect each other.
- The skin quality must be higher when the model almost intersects.
- The hands of the model must be visible.
- The model must have clearly defined limbs.
- The model must be humanoid.
- The model must be placed in the center of the scene.

While this list seems long, It is comparable to the list provided by Mixamo:



Can I auto-rig and animate animals, vehicles, or other objects?

No. Currently, the auto-rigger and animation libraries are for bipedal humanoids only.

Are my characters and animations stored on Mixamo.com?

My custom character won't upload/rig/animate. What should I do?

The Mixamo automatic rigging system only works for humanoid characters and has a few other specific requirements.

Ensure that the following is true for your project:

- The character is humanoid with distinguishable head, body, arm, and leg areas. If the character proportions are too deformed, the auto-rigger may not work.
- The character does not have large extra appendages or props. For example, extra limbs, wings, and tails or large hair and clothing items may not work.
- The character is in a default or neutral pose. Auto-rigging may not work if the character is largely asymmetric or posed prior to rigging.
- There is no other content in the file. Extra helper objects, cameras, or scene objects cause the auto-rigger to not work.
- There are no spaces between any of the parts. For example, the auto-rigger does not work on floating heads that are disjoined from the body.
- The character is centered in the scene. The auto-rigger works best when characters are set to the (0,0,0) origin of the world. This prevents animation offsets happening from unit differences.
- The character mesh is clean and error free. The auto-rigger performs more reliably on characters with clean meshes as well as provides higher-quality animations.

Figure 17: A screenshot of the Mixamo FAQ page "Mixamo", 2023

Some notable differences are that for my auto-rigger it must be a .fbx file where Mixamo accepts .fbx, .obj and .zip files, the skin quality is something supplied by the user, the hands having to be visible for the manual placement for the fingers, and the TPose auto rigger does support floating heads.

AccuRig by Reallusion does not have a list like this using it has revealed the following: It supports both .obj and .fbx files. The characters can be in T-pose, A-pose, multi-mesh, or even photo scanned. Characters with large accessories are not supported

When it comes to supporting less than 5 fingers, that is not something that will be implemented in the auto-rigger itself, while it does support less-fingered characters, the mocap data will always contain 5 so in order to record data for a character with 3 fingers, the index and middle finger of the mocap actor will have to be physically tied together, as well as the ring and pinky finger.

Certain requirements uncovered during this test have been selected to be improved upon in the next prototype phase:

- Allow .fbx files that contain a model made of multiple parts.
- Allow .obj files
- Support for asymmetrical models
- Explain skin bind quality to the user
- Add 'advanced' volume preservers for models with arms close to their torso
- Consider IK legs for digitigrade characters
- Find out why the fingers sometimes don't turn the right way
- Make sure the root motion matches the mocap data

12.2 Prototype V3

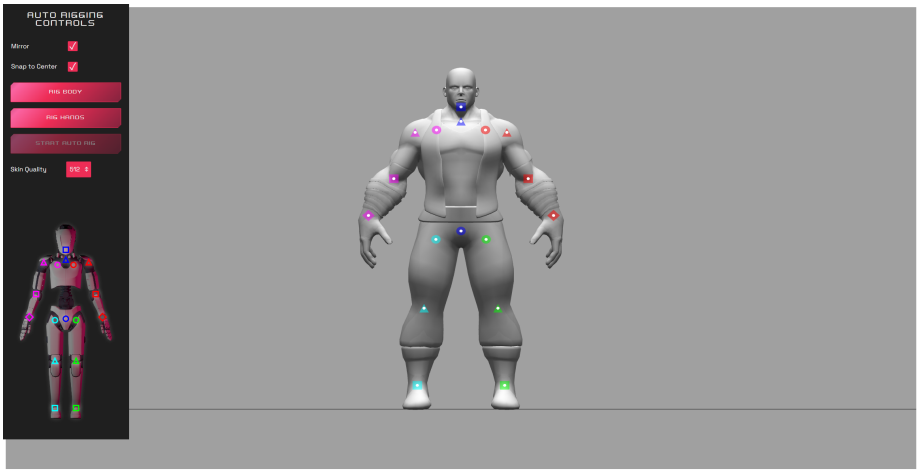


Figure 18: A screenshot of the Prototype V3 being used on the Blacksmith model

12.2.1 User Test

For this round of testing, users were tasked with spinning a wheel of names and then rigging the resulting character using the auto-rigger. During the rigging process, the researcher observed the users but did not actively interfere. Once the rigging was completed, the user was asked to fill out a survey, during this process the user was left unsupervised to promote honesty and anonymity.

After the first test, a significant issue that would impact further testing was uncovered. The auto-rigging process was not a circular process, meaning the user leaves the three.js environment and is redirected to the Maya file. This was quickly mitigated by creating a loading screen animation, Automatically having

Maya run in the background, and adding a three.js preview of the model on the final page of the process.

A full account of the test results can be found in appendix .13

12.2.2 Time

Character	Time to place locators	Time to run the rig
Aurora	0:58	1:08
Crunch	0:45	0:47
Goblin	0:51	0:37
GRIM	0:49	0:45
Sevarog	0:51	0:45
Skeleton King	1:41	0:36
Average	0:59	0:45

By comparing the time it took to go through the process from upload to finished rig in Prototype V3 to Prototype V2 insight can be gained into whether or not the implemented improvements caused any issues with the fluidity of using the tool. While the difference is by no means significant, one outlier can be seen in the time it took to place the locators for the Skeleton King. This was due to the Skeleton King being asymmetrical and thus requiring the placement of locators on both hands thanks to the newly implemented asymmetry support. A comparison to Mixamo and Accurig was omitted from this test due to the results not being any different from the previous test.

12.2.3 Conclusions

The testing revealed a range of remaining issues with the auto-rigger, primarily in the UI/UX direction. This was expected since the UI/UX had not been through testing before and was not the primary focus of this research. The most prevalent of the issues was that the tool was lacking a clear narrative to guide the user through the steps. This could be solved by a range of options, from numbering the buttons to sectioning off portions of the UI and adding explanatory text.

There were also issues with the rig itself, mainly in the feet and the fingers. Since the root motion of the recorded data does not scale with the retargeting to the model, the feet often float when the knees are bent. The fingers also often behave strangely, in part due to the skinning but there are also some rotational issues in the bones that need to be resolved.

Overall the quality of the resulting rig was acceptable for previewing recorded motions and the process of rigging was easy to follow regardless of the UI/UX issues experienced.

Some of the issues uncovered during this test will be resolved during the remainder of the development period, others will be left as recommendations at the end of the report.

13 Conclusions

In order to develop an automatic rigging tool that can be integrated as part of a web-based remote motion capture platform several challenges have to be overcome.

To address the first sub-question, the creation of a custom auto-rigger using Python for Maya involves leveraging the powerful capabilities of the MayaPy interpreter to automate the rigging process. This requires in-depth knowledge of Python scripting and Maya's rigging functionalities to design an efficient and customisable auto-rigger that can generate rig setups for various character models.

The establishment of a server-client connection is vital for enabling the seamless uploading, processing, and downloading of files within the platform. This involves implementing robust file transfer protocols and managing client-server communication. By employing suitable web development technologies and frameworks, such as Flask, along with appropriate file-handling mechanisms, a reliable server-client connection can be achieved.

The effective utilization of Three.js, a popular JavaScript library for 3D rendering, is crucial in creating and displaying a 3D character in a web-based environment. Leveraging the capabilities of Three.js, developers can import and manipulate 3D character models, view animations, and render them in real time within the web interface. This requires a solid understanding of Three.js APIs, 3D modelling principles, and web graphics rendering techniques.

Finally, the integration of the created auto-rigging tool into the current TPose platform requires careful consideration of the platform's architecture and workflows. This involves designing a user-friendly interface for users to access and utilize the auto-rigger seamlessly within the platform. Additionally, the rigging tool must be adapted to align with the platform's existing file management, character library, and motion capture workflow.

By addressing these sub-questions and considering the technical requirements and design principles, an automatic rigging tool can be successfully created and integrated into a web-based remote motion capture platform like TPose, enabling users to streamline the rigging process, facilitate previewing motion on a custom character, and enhance the overall motion capture experience.

14 Discussion

The chosen manner of testing the prototypes was befitting of technical research such as this one. It allowed the researcher to uncover a range of flaws with the product that could be addressed in the following development cycle. It could be argued that since the testing was performed by the researcher itself rather than the user, the validity of the testing is in question. However, since the testing was performed in order to compare the results of various models with existing auto-riggers, the variable of various users utilizing the auto-rigger was

intentionally left out.

For the second round of testing, the variable of rigging a randomized 3D model from a pool of selected models interfered somewhat with the statistical test results since some models performed better than others in the auto-rigger. This did however allow for a wider range of feedback to be gathered during the observation part of the testing which accounted for a significant portion of the results.

The created auto-rigger is functional but also still has several issues that are hard to ignore. The fact that the auto-rigger is unable to run in Maya standalone is a big downside for the seamless integration into the TPose platform. Various rotational issues in the rig are also cause for concern if not addressed in the near future. The ease of use of the tool is however comparable to other auto-rigging tools, as well as the speed at which the model is processed.

Overall the created tool fulfils the needs of the client, but optimally a few weeks of additional development and thorough testing is performed to really round off and polish out all the remaining issues.

15 Recommendations

Based on the research and development conducted for the creation and integration of an automatic rigging tool as part of a web-based remote motion capture platform, the following recommendations are proposed:

Further Enhancements to the Automatic Rigging Tool:

- The Geodesic Voxel bind skin method could be recreated in a way that does not require access to an OpenGL buffer by employing more modern Python libraries. This would allow the auto-rigger to run in Maya standalone and therefore not require a server with a GPU to run.
- Additional constraints could be created using the auto-rigger to allow for easy manual animation such as inverse kinematics on the legs, double elbows, and arm twists.
- Conduct usability testing and gather feedback from professional animators and riggers to identify areas for improvement and optimize the user experience of the auto-rigger.

Robust Server-Client Infrastructure:

- Strengthen the server-client connection by implementing secure file transfer protocols, ensuring data integrity, and optimizing file processing and downloading speeds.
- Implement a task queue on the server for more scalability, allowing multiple users to submit a rigging request at the same time without crashing the process.

- When integrating the auto-rigger into the TPose environment, the networking system using Flask will either have to be modified or converted to match the networking system employed by the TPose platform.

Optimization of Three.js Implementation:

- Improve upon the existing retargeting workflow, allowing partial transfer of rotation would allow for more advanced rigging features in the preview such as double elbows and arm twists.
- Conduct performance optimizations to ensure smooth real-time rendering and interactivity of the 3D character models, especially when dealing with complex animations and high-polygon models.
- Provide customization options within the web interface to allow users to adjust graphics settings based on their device capabilities and network conditions.

By implementing these recommendations, the automatic rigging tool can be enhanced to provide more advanced rigging features and a smoother user experience. The server-client infrastructure can be further optimized for secure and efficient file transfer and processing. The Three.js implementation can be refined to deliver high-quality 3D character rendering and interactivity. Finally, the integration with the TPose platform can be strengthened to provide seamless access to the auto-rigging functionality for platform users.

References

Arrachequesne, D. (2023). Socket.IO. https://socket.io/docs/v4/

Artell. (2023). Rig Features — AutoRigPro Doc documentation. http://www.lucky3d.fr/auto-rig-pro/doc/rig_behaviour_doc.html

Auto Character System 3. (2023). https://www.autocharactersystem.com/

Autodesk. (n.d.). Quick Rig Tool. https://help.autodesk.com/view/MAYAUL/2022/ENU/?guid=GUID-DC29C982-D04F-4C20-9DBA-4BBB33E027EF

Autorigs. (2023). https://www.sidefx.com/docs/houdini/character/autorigs.html

Cabello, R. (2010). Ricardo Cabello. https://ricardocabello.com/

Chesneau, B. (2023). Gunicorn - Python WSGI HTTP Server for UNIX. https://gunicorn.org/

Dionne, O., & De Lasa, M. (2013). Geodesic voxel binding for production character meshes. https://doi.org/10.1145/2485895.2485919

Eventlet Networking Library. (2023). https://eventlet.net/

Free Auto Rig for any 3D Character — AccuRIG. (2023). https://actorcore.reallusion.com/auto-rig#rig-characters

GeeksforGeeks. (2023). Flask HTTP methods handle GET POST requests. *GeeksforGeeks*. https://www.geeksforgeeks.org/flask-http-methods-handle-get-post-requests/

Gorakhargosh. (2023). Watchdog. https://github.com/gorakhargosh/watchdog

JSON. (2023). https://www.json.org/json-en.html

Le, B. H., & Deng, Z. (2014). Robust and accurate skeletal rigging from mesh sequences. *ACM Transactions on Graphics*, *33*(4), 1–10. https://doi.org/10.1145/2601097.2601161

Mixamo. (2023). https://www.mixamo.com/

Ronacher, A. (2023). Flask 2.3.2. https://flask.palletsprojects.com/en/2.3.x/

Team, B. D. (2023). Rigify — Blender Manual. https://docs.blender.org/manual/en/2.81/addons/rigging/rigify.html

The Maya Python Interpreter, MayaPy. (2023). https://help.autodesk.com/view/MAYACRE/ENU/?guid=GUID-D64ACA64-2566-42B3-BE0F-BCE843A1702F

Theodore, S. (2014). Earth calling maya.standalone! http://techartsurvival.blogspot.com/2014/04/earth-calling-mayastandalone.html

Theodore, S. (2023). Keystone. https://github.com/theodox/keystone

Uvicorn. (2023). https://www.uvicorn.org/

Waitress. (2022). https://pypi.org/project/waitress/

Xu, Z., Zhou, Y., Kalogerakis, E., Landreth, C., & Singh, K. P. (2020). RigNet. *ACM Transactions on Graphics*, *39*(4). https://doi.org/10.1145/3386569.3392379

Xu, Z., Zhou, Y., Kalogerakis, E., & Singh, K. P. (2019). Predicting Animation Skeletons for 3D Articulated Models via Volumetric Nets. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1908.08506

Appendices

.1 Emphatise and Define

A blog post at:
https://danthelionvfx.wordpress.com/2023/03/12/emphatize-and-define/

.2 Maya Python Auto Rigger 1

A blog post at:
https://danthelionvfx.wordpress.com/2023/02/01/maya-python-auto-rigger/

.3 Maya Python Auto Rigger 2

A blog post at:
https://danthelionvfx.wordpress.com/2023/02/01/maya-python-auto-rigger-2-2/

.4 Auto Rigger Inspiration

A blog post at:
https://danthelionvfx.wordpress.com/2023/02/01/maya-python-auto-rigger-2/

.5 Maya Python Auto Rigger 3

A blog post at:
https://danthelionvfx.wordpress.com/2023/02/01/maya-python-auto-rigger-3/

.6 Maya Python Auto Rigger 4

A blog post at:
https://danthelionvfx.wordpress.com/2023/02/02/maya-python-auto-rigger-4/

.7 Geodesic Voxel in Maya Standalone

A blog post at:
https://danthelionvfx.wordpress.com/2023/04/04/geodesic-voxel-in-maya-standalone/

.8 Three JS Server

A blog post at:
https://danthelionvfx.wordpress.com/2023/03/14/three-js-server/

.9 Socket IO - Server File Upload

A Youtube video at: https://youtu.be/IAj1eGSF1Ic \newline
A demo of the Socket.IO server receiving an uploaded file,
displaying it on the next page, and saving position values to a text file

.10 Use Case Testing

A blog post at:
https://danthelionvfx.wordpress.com/2023/05/03/use-case-testing/

.11 Auto Rigger V2 Use Case Test 2

A Youtube video at: https://youtu.be/6ZukKXc0h-s

.12 Prototype V2: Use Case Test Render

A Youtube video at: https://youtu.be/_IwFw3Y1NZY

.13 User Testing V3

A blog post at:
https://danthelionvfx.wordpress.com/2023/06/14/user-testing-v3/

.14 Maya Auto Rigger V2

A blog post at:
https://danthelionvfx.wordpress.com/2023/02/02/maya-auto-rigger-v2/

.15 Maya Python Auto Rigger 5

A blog post at:
https://danthelionvfx.wordpress.com/2023/02/06/maya-python-auto-rigger-5/

.16 From Server to Running the Auto Rigger

A blog post at:
https://danthelionvfx.wordpress.com/2023/03/21/from-server-to-running-the-auto-rigger/

.17 Quantitative Testing of In-House Models

A blog post at:
https://danthelionvfx.wordpress.com/2023/05/11/quantitative-testing-of-in-house-models/

.18 non-Thesis related work performed for Het Nieuwe Kader studio

Several tasks have been picked up by the researcher while working on the Bachelor Thesis Various blog posts at:

https://danthelionvfx.wordpress.com/2023/03/20/oorlogs-pad-door/
https://danthelionvfx.wordpress.com/2023/03/03/ran-d-mask/
https://danthelionvfx.wordpress.com/2023/03/03/ran-d-portal-creation/

https://danthelionvfx.wordpress.com/2023/02/06/touchdesigner-xsens-link/
https://danthelionvfx.wordpress.com/2023/02/01/camera-tracking-with-trackmen/
https://danthelionvfx.wordpress.com/2023/02/01/camera-tracking-with-htc-vive-puck/
https://danthelionvfx.wordpress.com/2023/01/30/camera-tracking-with-antilatency/
https://danthelionvfx.wordpress.com/2023/01/30/touch-designer/

.19 The Blog

A lot of the work done during this bachelor thesis has been recorded on a blog page, this might give insight into the development of the professional products and the timeline at which it happened.

https://danthelionvfx.wordpress.com/tpose-progress-blog/